Herconfigureerbare optische interconnectienetwerken voor multiprocessorarchitecturen met gedeeld geheugen

Reconfigurable Optical Interconnection Networks for Shared-Memory Multiprocessor Architectures

Wim Heirman

Promotoren: prof. dr. ir. J. Van Campenhout, prof. dr. ir. D. Stroobandt
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. J. Van Campenhout
Faculteit Ingenieurswetenschappen
Academiejaar 2007 - 2008

UNIVERSITEIT
GENT

ISBN 978-90-8578-215-5
NUR 959, 986
Wettelijk depot: D/2008/10.500/34

The larger the island of knowledge,
the longer the shoreline of mystery.
— Mary B. Yates

Acknowledgements

Quite a number of people have contributed in one way or another to the creation of this dissertation. I wish to thank them warmly for their support or participation. Specific thanks go out to:

- My promotors prof. Jan Van Campenhout en prof. Dirk Stroobandt, for giving me the opportunity to spend five wonderful years of my life working in a dynamic and challenging field, here in Ghent and at the various conferences I could attend; and for the valuable input both on this dissertation and on various matters in the years before, at all levels of abstraction ranging from the smallest of language issues to our place in the universe.

- My supervisor dr. Joni Dambre, for the many hours of discussion, the hundreds of pages of corrected manuscripts, and for guiding me in the best possible way: leaving me free to find my own way, but always asking that most important question: *Why?*

- The other members of the examination commission of this dissertation: prof. Daniël De Zutter, prof. Ian O'Connor, prof. Dries Van Thourhout and dr. Christof Debaes; and the many anonymous reviewers of my papers, both those accepted and rejected, who have provided a most valuable asset: the outsider's view.

- Jeroen Beeckman, Alan Benner, Mark Horowitz, David Patterson who helped me out with some of the figures and references in this work.

- The people from Virtutech, who created the simulation tool Simics without which this research would not have been possible, and everyone on their users forum for helping me get started.

- Everyone involved in the IAP-V and -VI networks and the doctoral schools: frequent contact with research groups doing the actual physics, and designing optical components and optical interconnection links, proved very valuable to provide me with background information and kept me in touch with the developments on those fronts. Since my primary training was in computer architecture, their importance can hardly be overstated.

- The people from the Vrije Universiteit Brussel, department TONA-TW, especially prof. Hugo Thienpont, dr. Christof Debaes, dr. Lieven Desmet and ir. Iñigo Artundo, who as members of the IAP network contributed to the above, and who are involved in the design and building of the SOB system implementation, in a work of simulations my (at times only) link to the "real" world.

- Iñigo "Goto" Artundo again, with whom I collaborated most closely during the last five years. Without his frequent input and reflections on our common research I would probably have given up a long time ago. Our joint publications are numerous, our conference trips legendary. . .

- dr. Bui Viet Khoi of the Hanoi University of Technology, Vietnam, for the invitation to an unforgettable stay in his magnificent country, among other things.

- My colleagues from the PARIS research group, both present and past, for the much appreciated scientific input and the pleasant work environment. I cannot mention all of them here, but to omit David, Frederik, Hendrik, Jeroen, Juan, Lieven, Michiel, Michiel, Michiel, Peter, Phillipe, Tom, Sean or Wim would be a disgrace!

- Jorge Cham, Randall Munroe and Scott Adams for the necessary moments of procrastination.

- Catherine, Ben, Peter, Wouter and all my other friends of MaSAC, for sticking with me even when my simulations did not.

- My parents, my brother and family, for their support throughout the years, giving me the opportunity to become who I am today.

- And, of course, my dearest Ine, for so many things. . .

Examination commission

prof. dr. ir. Daniël De Zutter
Chairman
Dean, Faculty of Engineering

dr. ir. Joni Dambre
Secretary
ELIS Department, Faculty of Engineering

dr. ir. Christof Debaes
TONA Department, Faculty of Applied Sciences
Vrije Universiteit Brussel

prof. dr. ir. Ian O'Connor
Lyon Institute of Nanotechnology
Ecole Centrale de Lyon, Ecully, France

prof. dr. ir. Dries Van Thourhout
INTEC Department, Faculty of Engineering

prof. dr. ir. Jan Van Campenhout
Advisor
ELIS Department, Faculty of Engineering

prof. dr. ir. Dirk Stroobandt
Advisor
ELIS Department, Faculty of Engineering

Samenvatting

Parallelle verwerking in computersystemen groeit aan belang. Daar waar de toepassing ervan vroeger beperkt bleef tot supercomputers en grote web- of databank-servers, brengt de multicore processor parallelle verwerking naar desktop PC's, laptops en spelconsoles. Om al deze processors op een nuttige manier aan het werk te zetten aan één enkel probleem, moeten ze kunnen *communiceren*. Hiertoe worden de processors met elkaar, en met de buitenwereld, verbonden door middel van een *interconnectienetwerk*. Van dit netwerk wordt verwacht dat het hoge bandbreedte en lage latentie verzorgt, zodat de processors zich kunnen concentreren op het verzetten van nuttig werk, en niet moeten wachten op het binnensijpelen van gegevens. Echter, huidige interconnectietechnologieën, die gebruik maken van elektronische signalen verstuurd over koperdraden, bereiken snel het einde van hun mogelijkheden.

Optische interconnecties zijn een mogelijke oplossing voor dit probleem, en worden momenteel onderzocht in onderzoeks- en industriële laboratoria, inclusief die van grote spelers zoals IBM en Intel. Verwacht wordt dat optisch communicerende multiprocessors binnen enkele jaren op de markt zullen komen. Multicore processorchips, met optische verbindingen geïntegreerd op de chip zelf, kunnen volgen binnen vijf à tien jaar.

Herconfigureerbare netwerken lossen een ander probleem van interconnectienetwerken op. Het verkeer op deze netwerken is meestal sterk niet-uniform, waardoor sommige verbindingen sterk verzadigd zijn terwijl andere nauwelijks gebruikt worden. De precieze patronen hangen bovendien sterk af van de toepassing, en kunnen zelfs veranderen tijdens de uitvoering van eenzelfde programma. Dit maakt het zeer moeilijk een efficiënte

architectuur te definiëren, die hoge prestaties toelaat maar die niet gedimensioneerd is voor de meest veeleisende verkeerspatronen – dit laatste zou tot een zeer duur, sterk overgedimensioneerd netwerk leiden dat in de meeste gevallen onderbenut wordt.

Gelukkig laten optische interconnecties, die in de nabije toekomst zeer waarschijnlijk toch geïntroduceerd zullen worden, toe om herconfigureerbare netwerken te maken. Die kunnen, tijdens de loop van het programma, op een transparante manier aangepast worden aan het momentele verkeerspatroon. Dit is mogelijk dankzij nieuwe optische componenten, zoals lasers met variabele golflengte, vloeibare-kristalschakelaars, enz.

Deze thesis richt zich op het toepassen van herconfigureerbare, optische verbindingen in de context van multiprocessors met *gedeeld geheugen*. In dit soort van machines is communicatie *impliciet*, en gebeurt dus zonder interventie van de programmeur of de compiler. Machines met gedeeld geheugen zijn daardoor eenvoudig te programmeren, doordat de programmeur geen intieme kennis moet hebben van de implementatiedetails van de machine. Dit soort machines zijn echter wel sterker beïnvloed door een hoge netwerklatentie. Technieken die het mogelijk maken om communicatie te overlappen met nuttige berekeningen, welke regelmatig worden toegepast in *message-passing* architecturen, zijn hier immers niet mogelijk doordat niet op voorhand bekend is wanneer communicatie nodig zal zijn. Machines met gedeeld geheugen hebben daardoor het meeste baat bij herconfigureerbare netwerken, die precies bedoeld zijn om een lagere latentie aan te bieden.

Eén probleem in deze is echter de snelheid waarmee het netwerk geherconfigureerd kan worden. Idealiter zou het netwerk voor elke aanvraag opnieuw ingesteld kunnen worden zodat het steeds een minimale latentie kan aanbieden. De beschikbare optische componenten laten dit echter niet toe. We gingen daarom op zoek naar *lokaliteit* in de communicatiestromen, en vonden dat netwerkverkeer dikwijls gedurende langere perioden relatief constant blijft. Dit feit kan uitgebuit worden door herconfiguratie: de beschikbare bandbreedte wordt dan zó verdeeld dat de meest volumineuze stromen de snelste verbindingen kunnen gebruiken. De totale latentie vermindert hierdoor significant. Dit alles kan gebeuren in een automatische, voor de software onzichtbare manier, wat in lijn blijft met de abstractie van gedeeld geheugen die de programmeur isoleert van de implementatiedetails van de onderliggende machine.

Verkeerspatronen. Vermits herconfiguratie noodzakelijkerwijs verloopt op een tijdsschaal die trager is dan individuele geheugentoegangen, is de trage dynamica van netwerkverkeer heel belangrijk voor dit werk. Het is be-

kend dat geheugentoegangen lokaliteit bezitten, zowel in de ruimte als in de tijd, en dat dit geldt op een fractale manier. Deze lokaliteit wordt uitgebuit door caches, die recent gebruikte gegevens in een klein en snel geheugen houden – omdat de kans groot is dat diezelfde gegevens niet lang daarna opnieuw nodig zullen zijn. Wegens de fractale eigenschap van lokaliteit is dit effect zichtbaar op alle tijdsschalen, gaande van nanoseconden – uitgebuit door eerste-niveau caches – tot micro- en milliseconden, wat zichtbaar is op het interconnectienetwerk. We modelleerden dit gedrag als verkeers*salvo's*, dit zijn perioden van communicatie met hoge intensiteit tussen specifieke processorparen. Deze salvo's werden geobserveerd in realistische simulaties en bleken tot meerdere milliseconden te duren. Ze kwamen voor op een achtergrond van meer uniform verkeer met veel lagere intensiteit.

Een herconfigureerbare netwerkarchitectuur. Dit gedrag bracht ons ertoe de volgende herconfigureerbare netwerkarchitectuur op te stellen: een basisnetwerk met vaste topologie, wordt aangevuld met herconfigureerbare extra verbindingen. Het basisnetwerk heeft een reguliere topologie, zoals een rooster, torus of hyperkubus, en verbindt alle processors. De herconfigureerbare verbindingen worden zó geplaatst dat deze een directe verbinding met hoge bandbreedte vormen tussen de processorparen met de meest intense communicatie. De locaties van deze extra verbindingen worden herzien na elk herconfiguratie-interval, typisch in de orde van enkele milliseconden. Op deze manier kan een groot volume aan verkeer gedragen worden door de extra verbindingen, terwijl het basisnetwerk steeds beschikbaar blijft voor achtergrondverkeer, of tijdens herconfiguratie van de extra verbindingen.

We stellen ook een implementatie van deze architectuur voor, die gebruik maakt van in golflengte verstelbare lasers en een selectieve uitzending. Deze implementatie laat een efficiënte schaling toe naar een groot aantal processors. Ze werd ontworpen in samenwerking met de Vrije Universiteit Brussel, waar nu een prototype gebouwd wordt van de componenten voor selectieve uitzending.

Versnellen van exploraties van de ontwerpruimte. Het ontwerp van een interconnectienetwerk omvat het bepalen van een groot aantal vrije parameters. Herconfiguratie voegt hier nog verschillende parameters aan toe, zoals het herconfiguratie-interval en het aantal ondersteunde extra verbindingen. Bovendien is de prestatie nu veel sterker afhankelijk van de specifieke eigenschappen van het netwerkverkeer, inclusief het tijdsgedrag. Wanneer een interconnectienetwerk ontworpen wordt moet een afweging gemaakt worden, die al deze parameters optimaliseert met als doel een architectuur die een goede prestatie, een laag vermogenverbruik en een beperkte kost

combineert. Dit vereist de evaluatie van een zeer groot aantal alternatieve ontwerpsvoorstellen.

Vermits de prestatie van bestaande, niet-herconfigureerbare netwerken slechts in beperkte mate afhankelijk is van veranderingen in de tijd van het netwerkverkeer, houden bestaande methodes om snel kandidaat-netwerken te evalueren hier geen rekening mee. We hebben daardoor bestaande methoden uitgebreid, en nieuwe methoden ontwikkeld, die wel toelaten een snelle maar voldoende nauwkeurige schatting te krijgen van de prestaties van herconfigureerbare netwerkarchitecturen. Onze methoden zijn bovendien ook bruikbaar op niet-herconfigureerbare netwerken, waar ze een hogere nauwkeurigheid toelaten dan bestaande methoden – precies omdat tijdsafhankelijk gedrag nu wel in rekening gebracht wordt.

Prestatie-evaluatie. Tenslotte evalueerden we de prestatie van onze voorgestelde herconfigureerbare netwerkarchitectuur, over een breed bereik van applicaties – gebruik makende van echte benchmarkprogramma's – en architecturale parameters. We combineerden grootschalige exploraties, gebruik makend van onze snelle methoden zoals hierboven beschreven, met gedetailleerde studies met behulp van zeer nauwkeurige – maar veel tragere – uitvoeringsgebaseerde simulaties.

Ook werden bovengrenzen gezocht voor de prestatie indien bepaalde heuristieken, die tijdens de herconfiguratie gebruikt worden, verbeterd zouden kunnen worden. Zo moet bijvoorbeeld, op het einde van een herconfiguratie-interval, het netwerkverkeer voor het daarop volgende interval voorspeld worden, zodat de extra verbindingen op de juiste plaats gelegd kunnen worden. Dit moet op zeer korte tijd gebeuren, en gebeurt daarom met behulp van een snelle heuristiek. We vonden echter dat, zelfs indien dit verkeer perfect voorspeld zou kunnen worden, slechts een prestatieverbetering met enkele procenten mogelijk is. Dit bewijst de kwaliteit van de gebruikte heuristieken.

Uiteindelijk vonden we dat, binnen de veronderstellingen en architecturale parameterwaarden die in dit werk gebruik werden, de netwerklatentie significant verbeterd kon worden. Afhankelijk van het netwerkverkeer van de specifieke applicatie, en van de eigenschappen van de beschikbare herconfigureerbare componenten, kon de latentie verlaagd worden met 10 à 20% bij kleine, 16 processors tellende netwerken, voor grotere netwerken tot 64 processors liep dit op tot een reductie met 20 tot 40%. En wanneer herconfiguratie reeds aanwezig is in het systeem, bijvoorbeeld voor het verhogen van de betrouwbaarheid, kan deze prestatiewinst bekomen worden aan een zeer lage toegevoegde kost.

Summary

Parallel processing is gaining importance. While its applications used to be limited to supercomputing or large web and database servers, the era of the multicore processor is now bringing parallel computing to desktop PCs, laptops and game consoles. To put all these processors to good use on solving a single problem, requires them to *communicate*. To this end, the processors are connected to each other, and to the outside world, with an *interconnection network*. This network should support communication at high bandwidths and low latencies, to allow the processors to concentrate on doing useful work, rather than having to wait for data to sip in. However, current interconnect technologies, using electronic signaling over copper wires, are quickly reaching the end of their capabilities.

Optical interconnections are a possible solution, one that is at the moment being investigated by academic and industrial labs around the world, including big players such as IBM and Intel. It is expected that optically communicating multiprocessor systems will be on the market in just a few years. Multicore processor chips, with on-chip optical networks, may follow in the next five to ten years.

Reconfigurable networks solve another problem of current interconnection networks. The traffic on these networks is usually very non-uniform, causing some of the network links to be highly saturated while others remain mostly unused. The exact patterns depend on the application, and can even change during the runtime of a single program. This makes it hard to define an efficient architecture, i.e., one that provides high performance but is not scaled for the worst-case traffic patterns – which would result in most of the capacity being unused for a large fraction of time.

Fortunately, optical interconnections, which will most likely be introduced in the near future anyway, provide the possibility of creating runtime reconfigurable networks, in a data-transparent way. This can be realized using novel components such as tunable lasers, liquid crystal switches, etc.

This thesis focuses on the application of reconfigurable, optical interconnection networks in the context of *shared-memory* multiprocessors. In this kind of parallel machine, communication is *implicit*, i.e., it happens without explicit involvement by the programmer or compiler. This makes shared-memory machines easy to program, since the programmer should not have intimate knowledge of the target machine. It does make a shared-memory machine very sensitive to network latency, since several clever tricks that try to overlap communication with useful computation, which are possible when using a message-passing paradigm, cannot be used since one does not always know exactly when communication will occur. Therefore, this type of machine would benefit the most from reconfigurable network architectures, which are designed precisely to provide lower latency.

One miss-match in an otherwise fitting puzzle is the speed of reconfiguration. Ideally, one would like to reconfigure the network such that the minimal latency can be provided for each request. The available optical components do not allow this, however. We therefore looked for *locality* in the communication, and found that network traffic often remains very similar for longer periods of time. This fact can be exploited by reconfiguration: by changing the distribution of available bandwidth in the network, such that the most voluminous traffic streams can use the fastest connections, latency can be brought down significantly. All this can be done in an automatic, application-invisible way, staying true to the idea of the shared-memory abstraction in which the programmer need not be aware of the implementation details of the underlying machine.

Traffic patterns. Since reconfiguration happens, by necessity, at a time scale slower than individual memory accesses, the slow dynamics of network traffic patterns are very important to this work. It is known that memory references exhibit locality in space and time, in a fractal or self-similar way. This locality is exploited by caches, which keep recently accessed data in a small, fast memory – because the probability of the same data being accessed again is high. Due to the self-similar nature of locality, this effect is present at all time scales, from the very fast nanosecond scales exploited by first-level caches, down to micro- and millisecond scales which are visible on the interconnection network. We modeled this behavior as traffic bursts: periods of high-intensity communication between specific processor pairs.

These bursts were observed to be active for up to several milliseconds, on a background of more uniform traffic with a much lower intensity.

A reconfigurable network architecture. This behavior prompted us to design the following reconfigurable network architecture: a base network with fixed topology, augmented with reconfigurable extra links (elinks). The base network has a regular topology such as a mesh, torus or hypercube, and connects all processors. The reconfigurable elinks are placed such that they provide a direct, high-bandwidth connection between those processor pairs that communicate with the highest intensity. The locations of these elinks are changed after every reconfiguration interval, which usually has a length in the order of milliseconds. This way, most of the high-volume burst traffic is carried by the elinks, while the base network is still available for background traffic, or during reconfiguration of the elinks.

We also propose an implementation of this architecture, using tunable lasers and a selective broadcast method that allows for efficient scaling of the architecture to a large number of processors. This implementation was developed in cooperation with the Vrije Universiteit Brussel, who are currently building a prototype of the selective broadcast system.

Speeding up design-space explorations. A large number of free parameters exist in the design of each interconnection network. Reconfiguration adds even more parameters, such as the reconfiguration interval and the number of elinks. Moreover, performance is now much more dependent on the specific characteristics of the network traffic, including its temporal behavior. When an interconnection network is designed, a trade-off has to be made that optimizes all these parameters towards a goal that combines good performance, low power consumption and limited cost. This requires the evaluation of a huge amount of design alternatives.

Because dependence on temporal traffic behavior is present only to a limited extent in existing, non-reconfigurable networks, methods that aim to speed up the initial evaluation of network candidates did not account for this temporal behavior. We therefore had to extend these methods, and proposed new methods that do allow a quick but sufficiently accurate evaluation of reconfigurable network architectures. Our methods are also usable on non-reconfiguring designs, were they can improve on the accuracy of older methods – precisely because temporal behavior is now taken into account.

Performance evaluation. Finally, we explored the performance of our proposed reconfigurable network architecture, under a wide range of work-

loads – using real benchmark applications – and architectural parameters. We combined large-scale explorations, using our faster methods described before, with detailed studies using highly accurate – but much more time consuming – execution-driven simulations.

Also, since elements of the architecture required heuristics to allow their evaluation in a limited time, upper bounds were found on the performance assuming these heuristics could be improved. For instance, the placement of the elinks is based on a prediction of network traffic for the next reconfiguration interval. We showed that even perfect traffic prediction could not increase the overall performance by more that a few percent, showing the quality of the heuristics that were developed.

Overall, we found that reconfigurable networks could, with the assumptions and architectural parameters that were used in our study, improve network performance significantly. Depending on the traffic pattern of the application, and on characteristics of the reconfigurable components available, latency could be lowered by 10 to 20% on small, 16-processor networks, for larger 64-processor networks this increased to a 20 to 40% reduction. And when reconfiguration is already present in the system, for instance for reliability reasons, this performance improvement can be obtained at a very low added cost.

Contents

Acknowledgements i
Examination commission iii
Dutch summary v
English summary ix
Contents xiii
List of Figures xvii
List of Tables xix
List of Acronyms xxi

1 Introduction **1**

1.1 Parallel processing 2
1.1.1 Why do we need parallel processing? 2
1.1.2 Limits to parallelism 4
1.1.3 Communication 6
1.2 Optical communication 16
1.2.1 Data communication using light 16
1.2.2 Short range optical interconnects 20
1.2.3 State of optics in computing 22
1.2.4 Reconfigurable optical interconnects 22
1.3 Reconfigurable processor networks 24
1.4 Contributions 25
1.4.1 Communication requirements 25
1.4.2 Reconfigurable architecture 26
1.4.3 Tools for design-space exploration 27
1.4.4 Performance evaluation 27
1.5 Structure of this thesis 28

2 Related work **29**
2.1 Optical interconnects . . . 29
2.1.1 System level . . . 30
2.1.2 Optical interconnect demonstrators . . . 30
2.1.3 Reconfigurable network architectures . . . 31
2.2 Photonics projects at ELIS . . . 32
2.2.1 IAP PHOTON and photonics@be . . . 32
2.2.2 Optical demonstrators . . . 33
2.2.3 On-chip . . . 33
2.3 Reconfigurable components . . . 34
2.3.1 Tunable VCSELs . . . 34
2.3.2 Liquid crystal switches . . . 36
2.3.3 MEMS mirrors . . . 37
2.3.4 The SOB system . . . 38
2.4 Thread and data migration . . . 39

3 Traffic patterns **41**
3.1 Time scales . . . 42
3.1.1 Packets: nanoseconds . . . 42
3.1.2 Memory accesses: microseconds . . . 44
3.1.3 Communication bursts: milliseconds . . . 46
3.1.4 Context switching: tens of milliseconds . . . 47
3.1.5 Applications: hours . . . 49
3.1.6 Hardware failures: days . . . 52
3.2 Communication bursts . . . 53
3.2.1 Traffic burst length distribution . . . 54
3.2.2 Traffic size fraction . . . 56
3.2.3 Application speedup . . . 56
3.3 Thread and data migration . . . 60
3.4 A case for reconfigurable networks . . . 60

4 A reconfigurable network architecture **63**
4.1 Proposed reconfigurable network architecture . . . 64
4.2 Hardware implementation . . . 70
4.3 Extra link selection . . . 72
4.4 Simulation framework . . . 76
4.5 Benchmarks . . . 79

5 Speeding up design-space explorations **81**
5.1 Predicting network performance . . . 83
5.1.1 Prediction model . . . 83

5.1.2 Prediction accuracy . . . 89
5.1.3 Improving accuracy . . . 92
5.1.4 Reduction in simulation time . . . 92
5.2 Congestion modeling . . . 94
5.2.1 Contention model . . . 94
5.2.2 Results . . . 99
5.2.3 Discussion . . . 100
5.2.4 Improving accuracy . . . 102
5.2.5 Reduction in simulation time . . . 103
5.3 Synthetic network traffic . . . 103
5.3.1 Synthetic traffic generation . . . 105
5.3.2 Generating synthetic traffic patterns . . . 109
5.3.3 Simulating the synthetic traffic flow . . . 109
5.3.4 Results . . . 110
5.3.5 Required trace length . . . 113
5.4 Comparison . . . 115

6 Performance evaluation **119**
6.1 Variability of performance metrics . . . 120
6.2 Small networks: execution-driven simulation . . . 125
6.2.1 Selective broadcast implementation . . . 125
6.2.2 Latency versus throughput . . . 129
6.3 Large networks: synthetic traffic . . . 130
6.3.1 Scaling the reconfigurable architecture . . . 130
6.3.2 Non-reconfigurable networks . . . 134
6.4 Effect of reconfiguration heuristics . . . 137
6.4.1 Optimal elinks placement . . . 137
6.4.2 Perfect traffic prediction . . . 137

7 Conclusions **143**
7.1 Summary . . . 143
7.2 Future research . . . 146
7.2.1 Extensions to the current work . . . 146
7.2.2 Towards a reconfigurable demonstrator . . . 148
7.2.3 On-chip communication . . . 148
7.2.4 Faster reconfiguration . . . 150
7.3 Conclusion . . . 151

Publications **153**

References **157**

List of Figures

1.1 Limits to single-processor performance 3
1.2 Speedup according to Amdahl's and Gustafson's laws 6
1.3 Schematic overview of a multiprocessor machine 8
1.4 Example network topologies 10
1.5 Relative improvement of bandwidth and latency 13
1.6 Power dissipation of electrical vs. optical interconnects 18
1.7 The physical interconnect hierarchy 19
1.8 Artist's impression of an opto-electronic integration approach 21

2.1 Fiber ribbon cable connected to the IO demonstrator 31
2.2 Schematic of a cantilever MEMS tunable VCSEL 35
2.3 SEM micrograph of a beam steering MEMS mirror 37
2.4 The SOB element distributes the signal from each input source to nine destinations 38

3.1 Time scales relevant in multiprocessor network traffic 43
3.2 Packet sequences in a directory-based coherence protocol . . . 45
3.3 Traffic to a node in relation to context switches 48
3.4 Burst length distribution for the `fft` application 54
3.5 Traffic size and latency fractions per burst length 57
3.6 Maximum achievable speedup per burst length 59

4.1 Conceptual reconfigurable network topology 65
4.2 Sequence of events controlling reconfiguration 66
4.3 Circuit switched equivalent of the elink concept 69
4.4 Schematic overview of a reconfigurable network architecture . 71
4.5 Optimized node placement on the SOB element 72

4.6 Pseudo code for the elink selection algorithm 74
4.7 Illustration of the elinks placement through time 77
4.8 Switching architecture of a network node 78

5.1 Speed versus accuracy of network evaluation techniques . . . 82
5.2 Distance distributions of memory operations 87
5.3 Variation of average memory latency per distance 88
5.4 Predicted versus measured latency reduction 90
5.5 Latency improvement after adding elinks 91
5.6 Distribution of congestion over the links 93
5.7 Computation time required by our prediction method 93
5.8 Division into independent queueing systems 96
5.9 Packet waiting time for various numbers of elinks 100
5.10 Packet waiting time per reconfiguration interval 101
5.11 Average waiting time per packet, when dividing time into intervals of different length 102
5.12 Possible sequences of packets or packet groups 105
5.13 Distributions determining a traffic profile 107
5.14 Average packet hop distance and latency 112
5.15 Accuracy of shorter synthetic packet traces 114

6.1 OS scheduling decisions are affected by memory latency . . . 121
6.2 Variability of possible network performance metrics 123
6.3 Variability of performance and statistical significance 124
6.4 Simulation results of a reconfigurable network 126
6.5 Summary of memory access latency 128
6.6 Latency versus throughput for a variety of benchmarks 129
6.7 Performance scaling for increasing numbers of elinks 131
6.8 Performance scaling for increasing the fan-out 132
6.9 Performance trends for varying reconfiguration intervals . . . 133
6.10 Summary of packet latency for different networks 134
6.11 Comparison between reconfigurable and static networks . . . 136
6.12 Comparison with globally optimal elink placement 138
6.13 Comparison with perfect prediction of traffic 140
6.14 Cost of suboptimal traffic prediction 141

List of Tables

3.1 Breakdown of LANL failure data into categories 52
3.2 Applications and estimated speedups 58

4.1 SPLASH-2 benchmark applications and problem sizes 80

5.1 Variability and runtime for trace- and execution-driven simulations 115
5.2 Comparison of network evaluation techniques 116

List of Acronyms

ccNUMA Cache-Coherent NUMA

CMOS Complementary Metal-Oxide-Semiconductor

CPU Central Processing Unit

DBR Distributed Bragg Reflector

DPW Deep Proton Writing

DRAM Dynamic Random-Access Memory

DSM Distributed Shared-Memory

EMI Electro-Magnetic Interference

FFT Fast Fourier Transform

FPGA Field Programmable Gate Array

GaAs Gallium Arsenide

Gbps Gigabit (10^9 bit) per second

GiB Gigabinary (gibi, 2^{30}) byte

HPC High-Performance Computing

IC Integrated Circuit

ILP Instruction Level Parallelism

InP Indium Phosphide

IO Interconnect by Optics

KiB Kilobinary (kibi, 2^{10}) byte

LAN Local Area Network

LC Liquid Crystal

LVDS Low Voltage Differential Signaling

MEMS Micro-Electro-Mechanical Systems

MiB Megabinary (mebi, 2^{20}) byte

MPSoC Multi-Processor SoC

MTBF Mean Time Between Failures

NoC Network-on-Chip

NUMA Non-Uniform Memory Access

OBS Optical Burst Switching

OIIC Optically Interconnected Integrated Circuits

OPS Optical Packet Switching

OptiMMA Optimization of MPSoC Middleware for Event-driven Applications

OLTP On-Line Transaction Processing

OOO Out-of-Order

PCB Printed Circuit Board

PICMOS Photonic Interconnect Layer on CMOS by Waferscale Integration

RCPD Resonant Cavity Photodetector

ROI Reconfigurable Optical Interconnect

SAN Storage Area Network

SIMD Single Instruction, Multiple Data

SMP Symmetric Multi-Processor

SOB Selective Optical Broadcast

SoC System-on-Chip

SOI Silicon-on-Insulator

SRAM Static Random-Access Memory

TiB Terabinary (tebi, 2^{40}) byte

VC Virtual Channel

VCSEL Vertical-Cavity Surface-Emitting Laser

VLSI Very Large Scale Integration

WADIMOS Wavelength Division Multiplexed Photonic Layer on CMOS

WAN Wide Area Network

WDM Wavelength Division Multiplexing

1
Introduction

A journey of a thousand miles
begins with a single step.
— Chinese Proverb

Parallel processing is gaining importance. While its applications used to be limited to supercomputing or large web and database servers, the era of the multicore processor is now bringing parallel computing to desktop PCs, laptops and game consoles. A major bottleneck in building an efficient multiprocessor architecture is the interconnection network. Since the degree of parallelism, or the number of processors, are only expected to grow, the gap between processor speeds and interconnect performance will widen further. Optical communication is expected to provide the solution here. This work looks out even further, and studies reconfigurable optical interconnects. They allow networks to better fit the communication needs of the processors, and the application running on them, at each point in time. In this way, the performance gap can be bridged, leading the way towards efficient multiprocessing solutions.

1.1 Parallel processing

Software is slowing faster than hardware is accelerating.
— Martin Reiser

1.1.1 Why do we need parallel processing?

One constant in the history of computing is that today's machines are always too slow. Users of computation power continuously want increasingly more capable machines, to allow them to make more accurate weather forecasts, support more simultaneous visitors to their web sites, play ever more realistic 3-D games, etc. To be able to provide this power, the beating heart of the computer – the processor – has undergone a dramatic evolution over the past several decades, increasing its calculation speed by several orders of magnitude.

At each point in time, there were users who wanted – and could afford – machines with multiple processors. Having them all work at the same time, *in parallel*, on the same problem, allowed them to do more calculations today, rather than having to wait for tomorrow's processor. This idea has been in use since the 1960s and '70s in early supercomputers and mainframes. With the advent of the microprocessor in the 1980s the idea became very attractive, since high numbers of cheap, mass produced processor chips could be combined into relatively inexpensive but very powerful supercomputers.

Today parallel computing is again very much alive – and this time in a much wider range of the market, not just for top-end machines – because of the problems that are currently plaguing microprocessor design. Moore's law [Moore, 1965], which states that *the most cost-effective number of transistors to put on a chip doubles every two years*, still holds, so transistor budgets keep increasing exponentially. However, it is proving ever more difficult to put all these transistors to good use on a single instruction stream. Most of the Instruction Level Parallelism (ILP) inherent to today's applications is already being exploited, and cache sizes have long ago entered the region of diminishing returns. Also, the bandwidth and latency of on-chip wires are not scaling fast enough. This limits designers to having smaller, more independent structures with local communication rather than a single entity spanning the entire chip (for high-speed designs, it is today not possible to

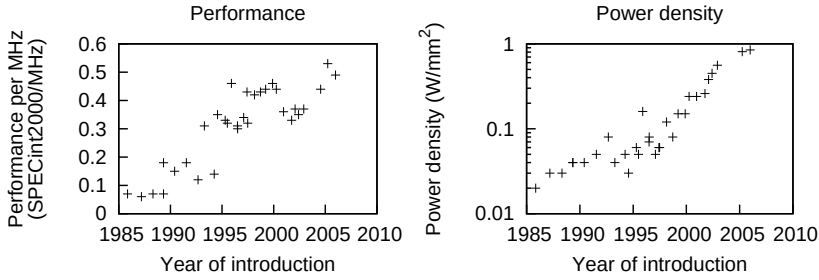

Figure 1.1: Limits to single-processor performance: SPECint performance per MHz (left), and power density (right) for Intel's 80386 to the Pentium 4 [Horowitz, 2007]

communicate across the chip in a single clock cycle). Finally, physical limitations such as the speed of light and extreme power dissipation prevent the clock speed from rising much further. Figure 1.1 illustrates this point, plotting the saturating per-clock-cycle performance and the exponentially rising power density throughout the years for Intel's desktop microprocessors from the 80386 to the Pentium 4. Designing single processors that provide the exponential increase in performance that we have come to expect is therefore no longer feasible. Instead, the available number of transistors is now used to provide multiple processor *cores* on a single chip. All large microprocessor manufacturers have fully embraced the *multicore* paradigm [Geer, 2005; Held et al., 2006] and are selling multicore chips for the desktop and laptop market (Intel Core2 Duo, AMD Athlon X2). This trend is expected to continue towards more cores, as is evident from Intel's 80-core demonstration chip [Vangal et al., 2008]. The downside is that this evolution now places the burden upon the software writer: while previously each new generation of microprocessor provided a free increase in single-threaded performance, programs now have to be parallelized in order to benefit from the increased capacity of new chips [Sutter, 2005].

Evidently, multiprocessing has, throughout the years, made its way from the top end of the market – High-Performance Computing (HPC) or supercomputing and high-end Symmetric Multi-Processor (SMP) servers since the 1980s – down to the desktop today. Even PDAs and embedded applications already have multiple (specialized) processors [Goddard, 2003].

1.1.2 Limits to parallelism

If you were plowing a field, which would you rather use: two strong oxen or 1,024 chickens?
— Seymour R. Cray

As the quote above by Seymour Cray, a pioneer in and highly successful architect of parallel computer systems, suggests, there must be a drawback to using multiple smaller processors over a (hypothetical) single, more powerful one. Apart from the technical hurdles of designing and building efficient multiprocessor machines, some problems remain. First of all, the parallelism in the program must be expressed explicitly, which is an added task for the programmer.[1] Current programmers are trained to convert algorithms into a sequential program flow, and must be re-educated to *think in parallel*. Next, dependencies between blocks of communication limit the amount of work that can be done concurrently. Finally, communication and synchronization between computing elements can become a bottleneck.

A parallel program is usually specified as a number of independent pieces or *threads* that can execute at the same time, each on its own processor. Creating such a specification is a complex task, since care must be taken that dependencies, pieces of data computed by one thread and used by another thread, are respected. This requires synchronization: before a thread can use a data structure, it must wait until the thread computing these values signals that it has reached a given point in its execution, and that the result is available. Not implementing synchronization correctly introduces a whole new range of bugs that did not exist in sequential code.

Due to these dependencies, some parts of the program cannot run concurrently. One can compute the *speedup* S, this is the reduction in runtime of a program when moving from a sequential (i.e., on one processor) to a parallel execution (using N processors). *Linear speedup*, which means that the program speed increases linearly with the number of threads or processors, denoted by $S = N$, cannot be achieved in most cases. Indeed, most realistic programs have parts that are inherently sequential. Consider for instance a parallel program in which the sequential part accounts for a fraction s of the total runtime. When the parallel part (with a runtime fraction p in a

[1] Several technologies exist for *automatic parallelization* [Tseng, 1989; Rogers and Pingali, 1994]. And while research in this field is still very much active, none of these techniques have seen much wide-spread use, due to problems in performance, usability, market adoption, etc.

sequential execution) is divided over N processors running in parallel, the total runtime of the application drops from $s + p = 1$ in the sequential execution to $s + p/N$ when using all N processors. The speedup achieved by parallelization is therefore:

$$S = \frac{1}{s + (1 - s)/N} \tag{1.1}$$

From this we can see that linear speedup, $S = N$, requires $s = 0$ or that there is no sequential part. Realistic programs with $s > 0$ will, even if we have an unlimited number of processors available ($N \to \infty$), have a maximum speedup that is limited by their sequential fraction:

$$S \to \frac{1}{s}$$

Therefore, for most practical programs, performance cannot rise indefinitely when adding more processors because of the program's inherently sequential part. This observation is known as Amdahl's law [Amdahl, 1967].

This does not mean, however, that parallelization has little practical use, or that future machines with a larger number of processors are unnecessary. For most applications, with some clever re-ordering of computations, the sequential part can be made very small. Moreover, Amdahl's law assumes that the problem size remains constant. When evaluating future machines though, we should also consider future problem sizes, which are expected to scale too. This has led Gustafson [1988] to propose an alternative law that assumes a scaling of the problem size, in such a way that the program runtime remains constant. A real-life example is the 5-day weather forecast, computed every afternoon based on measurements collected during the morning. Its execution should finish in at most 8 hours, in time for the evening TV weather talk. If meteorologists are given a more powerful computer that can do the task in less time, they will simply scale up the problem (increase the number of grid points, which improves accuracy) to bring the runtime back up to their maximum accepted duration of 8 hours. The same holds for 3-D games. They have a constant per-frame runtime (to achieve a frame rate of 50 frames per second, the time available is 20 ms), but, with every new generation of games and machines, more graphical detail, more intelligent enemies, etc., are squeezed into those 20 ms.

When the problem is scaled in this way, the speedup obtained from parallelization becomes (see [Gustafson, 1988] for its derivation):

$$S = N - s \cdot (N - 1)$$

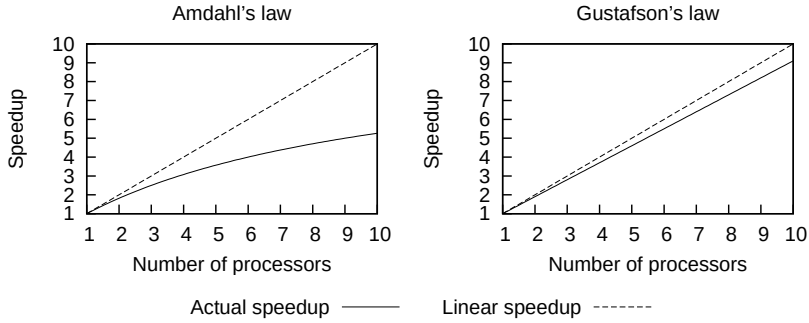

Figure 1.2: Speedup according to Amdahl's and Gustafson's laws compared with linear speedup, for a program with a serial fraction of 10%. When the program size is constant (Amdahl), the speedup saturates. If the program is scaled to maintain a constant runtime (Gustafson), the speedup – and thus problem size – grow linearly.

Here, the speedup does not saturate for $N \to \infty$. Large parallel machines *are* therefore useful, assuming there are problems large enough to keep them busy. This has, in practice, never proved to be much of a problem.

Figure 1.2 compares both observations, for a program with a serial fraction of 10%, and compares the actual speedup with the ideal, linear speedup. When the program size is constant (Amdahl), the speedup saturates to $1/s = 10$. When scaling the program to maintain a constant runtime (Gustafson), the speedup grows linearly, with a slope of $1-s = 0.9$. Note that for this to happen, the *problem size* also has to grow linearly. So, in contrast to the graph for Amdahl's law, a different program (size) is considered at each point. Also, the relation between the actual size of the input data and the (sequential) runtime is not considered here. This relation is usually not linear: if n is some scaling factor of the data set (number of grid points, matrix size, etc.), then the computational complexity (the number of instructions that need to be executed, which is roughly proportional to the runtime of a sequential execution) will often scale with for instance $n \log n$, $\sqrt{n}$, n, n^2, $n!$, etc.

1.1.3 Communication

When lots of processors work together on the same problem, they must *communicate*: input data and program code must be distributed from one processor to the others, partial results must be exchanged so that other

processors can continue working on them, and *synchronization* must take place so that all processors know when to continue processing and when to wait for other processors to complete work on certain pieces of data.

Just like Amdahl's law, communication makes parallelization only feasible up to a certain number of parallel threads, depending on the exact algorithm, the problem size, and the performance of the machine on which the program will be run. Beyond that, the threads would have too many dependencies and require much communication among them – this degrades performance because processors will often have to wait for data rather than doing useful computations. Note that here, technical properties of the multiprocessor machine, aside from just the number of processors, become relevant. A different parallelization strategy is therefore necessary for different (classes of) machines. The most important property here is the cost of communication, which is in its turn determined by the properties of the *communication network.* This network connects the processors with each other and allows them to exchange information at very high speeds. It is on this component that this thesis will focus.

The shared-memory abstraction

Communication between two processors can be initiated in two ways. It can be specified *explicitly* by the programmer, through a programming library such as MPI [Snir et al., 1995]. In this case a processor executes a certain sequence of instructions that sends a specific piece of data to another processor. Communication can also be *implicit:* here neither the exact points in the program where communication happens, nor the data words to be transmitted, are specified directly. Instead, the concept of *shared memory* is used. A range of memory addresses is shared by multiple processors. When processor A writes to a shared-memory address, a subsequent read by processor B from that same address will return the data previously written by A – this way communication has occurred between processors A and B. This can be implemented in a number of ways. For instance, the data can reside close to processor A until it is read by B, at which time it is sent over the network. If B should never need this data, no communication bandwidth will be wasted. Alternatively, the data can be sent directly to B at the time of the write by A, so that B has immediate access to it, rather than having to first request the data over the (relatively) slow communication network. The main advantage of machines providing this abstraction is that the programmer does not need intimate knowledge of these implementation details. Moreover, if a program has been written once, the same program can run on different machines with varying implementations, as long as they

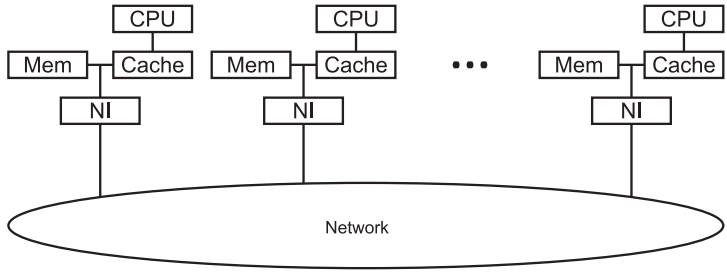

Figure 1.3: Schematic overview of a multiprocessor machine. Network traffic is generated by the network interfaces (NI) in response to non-local memory accesses by a processor.

all provide the abstraction of shared memory. The fact that communication is implicit further helps to ease programming. This makes shared-memory machines very popular in diverse areas, from scientific supercomputers over commercial servers and mainframes down to desktop computing.

This work will focus on Distributed Shared-Memory (DSM) machines with hardware-based coherence. In such machines, main memory is physically distributed over *nodes*. These are building blocks of the system that each contain a processor with its cache subsystem, a part of main memory, and a network interface linking the nodes to the communication network and thus to each other.[2] A schematic overview of this architecture is shown in Figure 1.3. All memory is accessible by all processors, but accesses to *remote* memory locations (i.e., those in some part of main memory that is not on the same node as the processor making the request) are intercepted by the network interface. This architecture is called a Non-Uniform Memory Access (NUMA) architecture, because the access characteristics of main memory are not the same for each address – remote addresses are much slower to read and write than addresses located on the same network node. Shared-memory locations can still be cached by the local cache of each processor. However, since multiple copies of the same memory location may now exist on different nodes, they must be kept coherent (i.e., contain the same information) – hence the designation Cache-Coherent NUMA (ccNUMA). The task of the network interfaces is thus to support reading and writing of shared-memory locations, and to enforce coherence among all caches and main memory.

[2]Often, machines are created in which several processors share a network interface, this is usually the case when multicore processor chips are used. In this work however, for the sake of clarity we will always assume that each node contains one processor, this does not significantly affect network traffic.

Communication networks

At a conceptual level, we want every processor to be able to communicate with every other processor. This is usually not possible to realize physically. The number of pins that can be placed on a processor chip is limited, which places an upper bound on the available bandwidth. Splitting up this bandwidth into separate connections to potentially hundreds of other processors is not feasible: communication is often sparse so most of the pins would be unused for a large fraction of time. This does not constitute an economical use of resources. Rather, several pins are combined into a small number of *links* (one link is a connection between two processors, possibly allowing transmission of multiple bits simultaneously), usually no more than four.[3] This means that each processor can be directly connected to just four others.

Real-world communication networks need to connect tens, if not hundreds of processors or network nodes. Not all nodes can thus be connected directly. All nodes are connected, in a usually regular *topology*, to just a small set of other nodes (its *neighbors*). Each node can have at most as many neighbors as the number of links it supports (this number is called the *fan-out* or *radix*). Messages sent between nodes without a direct connection go through a set of intermediate nodes which forward the message. The number of links a message needs to traverse is called the *inter-node distance*. Directly connected nodes are at a distance of one. Shorter inter-node distances are preferable, because messages that need few intermediate hops can reach their destination more quickly, and use the resources of a smaller part of the network. Figure 1.4 shows some common topologies. An important trade-off when comparing topologies is that between maximum fan-out (usually limited by the technology available) and the maximum inter-node distance (the network's *diameter*).

Aside from network topology, *placement* of processes and data is of equal importance. Threads, each of which runs on one processor, communicate with each other, and make accesses to data words in memory. The average distance messages have to travel over the network can be minimized by doing a good placement: threads that communicate often are placed close together (ideally they are on neighboring nodes), data should be placed close to (ideally on the same node as) the processor running the thread that needs access to it the most.

An alternative to this scheme is the *bus*, here all communication partners are connected to the same (set of) wire(s). If one of them transmits a message,

[3] For instance, AMD's Socket 940 for the Opteron processor uses 228 of its 940 pins for three HyperTransport links that connect to other processors, in addition to 226 pins for the DDR memory interface and 424 pins for supplying power to the chip [AMD, 2004].

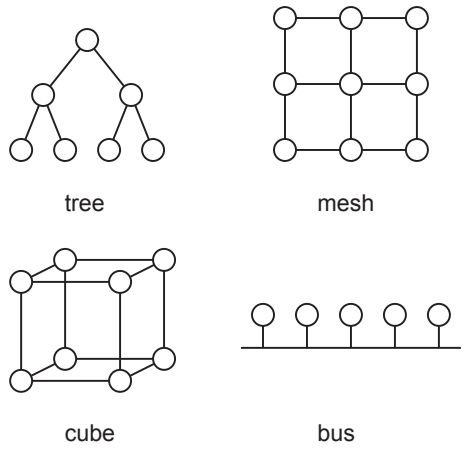

Figure 1.4: Some example network topologies

all other processors are able to receive it. There is no need for forwarding by intermediate nodes, so the topology and placement problems are avoided. However, at each point in time just one processor can use the bus. In contrast to other topologies such as trees or meshes, adding network nodes does not increase the available bandwidth on the bus. This makes scaling this solution to a large number of processors difficult. Also, the electrical bus has severe physical limitations: the bus should be viewed as a transmission line, the added capacitive load of all network nodes connected to the bus reduces the signal propagation speed so that the bus should be either very short or have a very low bandwidth. For this reason, over the past decade, buses have been phased out as means of inter-chip communication in favor of point-to-point connections. For example, the 1997 Sun Enterprise E6000 was the last of Sun's large bus-based servers. The 64-processor E10000 model used an intermediate approach, where addresses were broadcasted on four address buses, and data was transferred over point-to-point links [Charlesworth et al., 1997]. Their successor, the 2001 Sun Fire line, has an interconnection structure with point-to-point links only [Charlesworth, 2001]. SMPs with up to eight processors, for small servers and desktops, can now use AMD's HyperTransport [Keltcher et al., 2003], while Intel has only recently announced to abandon its bus-based approach, in use since the 1995 Pentium Pro, in favor of its new QuickPath point-to-point interconnect [Kanter, 2007].

Bandwidth versus latency

> *There is an old network saying: Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed – you can't bribe God.*
> **— David Clark, MIT**

When looking at the performance of a communication network, there are two important metrics to consider:

Bandwidth or throughput is the amount of data (measured in bytes) that can be sent from one communication partner to another in a given time.

Latency is the time between the sending of a message by the source and the destination receiving it.

The latency a message experiences when sent over the network is the time between the moment when the sender starts transmission and the moment the receiver has completely received the message. This time can be split up into two parts. The first part is the time that the sender needs to transmit the message. For the second part we follow the last bit of the message through the network until it arrives at the receiver, at which point it has received the complete message. The first component, the *transmission time*, is only dependent on the bandwidth: a message of size S sent through a network of bandwidth B needs a time S/B. The second component is influenced by a number of factors. First of all, the speed of light and the physical distance between sender and receiver determine the *time of flight*, this is the time it takes for a signal (a change in the electrical potential of a copper wire, a pulse of light, etc.) to travel from sender to receiver. Next, the network itself often needs some time to determine how messages should be routed through it, during which time messages also experience delay (*routing time*). Finally, network links are shared among several processors. This means there often is a conflict between multiple processors trying to use the same network resource, in which case some of the messages need to be delayed (*waiting time*). The total latency is the sum of *transmission time*, *time of flight*, *routing time* and *waiting time*.

Bandwidth can be improved by using higher bit rates (shortening the time per bit) or by increasing parallelism (sending more bits at the same time). Improving latency is often less straightforward. Parallelism is again helpful here: if a message is transmitted completely in parallel, the last

bit arrives at the destination at the same time as the first bit. In contrast, when sending the message serially, the last bit arrives much later. Therefore, when multiple parallel channels are available, it is, at least from a latency standpoint, advantageous to transmit a single message over all channels simultaneously, rather than serializing messages over one channel and using the different channels for different messages. Skew (differential latency among the different channels) complicates this situation, since it may require *resynchronization* circuits to align all bits at the receiver.

Time of flight is usually not under the control of a network designer. The speed of light is fixed, only the physical distance between communicating partners can be reduced. This is in practice usually not an option: the physical size of the processors, memory chips and supporting hardware prevents an infinitely close packing of the components. Also, thermal considerations need to be made: each processor radiates in the order of 100 W of heat, cooling them is a challenge in itself and again requires sufficient area.

Routing time is influenced by the router architecture. A simple routing protocol helps in this respect. Also, since each intermediate node incurs extra routing overhead, total routing time can be reduced by minimizing the inter-node distance by means of a properly chosen network topology.

Finally, waiting time is influenced by most of the other factors that were already mentioned. Messages have to wait when the next link the message needs to traverse is unavailable. The (average) duration of this time therefore increases when the *load* of the links – this is the fraction of time the link is sending messages, in contrast to being idle – increases. When the volume of traffic is constant, link load can be decreased by making links faster, spreading out traffic such that *hot-spots* – links with loads much higher than the average – are avoided, or reducing the average number of intermediate nodes a message needs as this reduces the number of links occupied per message, and therefore the total link load.

When we look at the historical improvements made in both bandwidth and latency, one can see that bandwidth has always increased more rapidly than latency. This is evident from Figure 1.5, which plots the relative improvements of bandwidth and latency through several technological milestones between 1982 and 2003, for microprocessors, memory chips, networks and hard disk drives. The trend is clear: while latency improved roughly tenfold during the last 20 years for all four of these key components, their bandwidths improved by 100× to 1000×. Patterson [2004] gives a number of physical and psychological motivations for this: increasing bandwidth is usually easier to achieve (adding parallel processing units can linearly increase throughput, but it does not change latency) while latency is often caused by distance (a 300-m Ethernet connection can never reach a latency

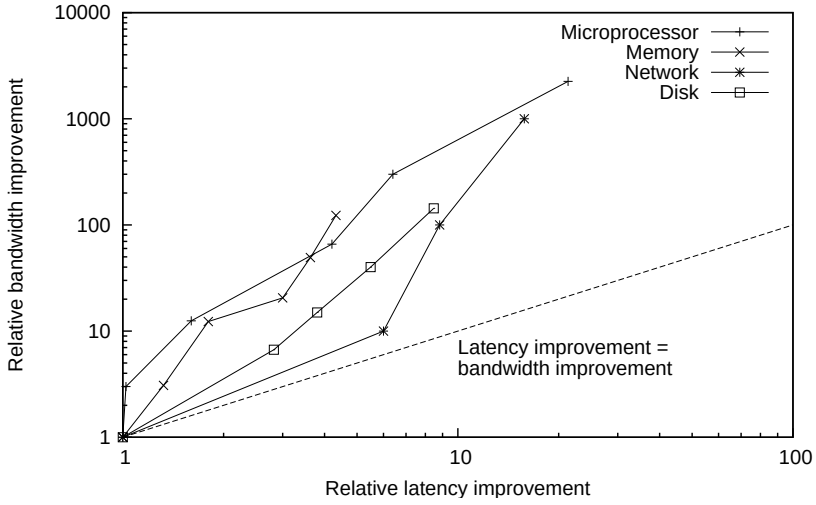

Figure 1.5: Log-log plot of bandwidth and latency milestones between 1982 and 2003. Note that latency improved only about 10× while bandwidth improved about 100× to 1000× [Patterson, 2004].

lower than 1 μs, due to the time of flight), and bandwidth often *sells* more than latency (most people are more accustomed to dealing with bandwidth, the benefits of a 10-Gbps network are likely easier to explain than a 10-μs latency). He concludes that, therefore, this disparity will continue in the future. Systems architects should thus try to cope with high latency, rather than hoping for technological advances that reduce it.

Yet, in a shared-memory communication network, latency has much more influence on performance than bandwidth. When a processor wants to read from a remote data word, a read request is sent over the network to the node where this word is located, and the reply with the data contents is sent back. During this time, the processor cannot continue. And while higher bandwidth can reduce the transmission time, the other latency components are not, or sometimes even negatively, affected by bandwidth. So in the end, it is the message latency that determines the remote memory access time, and therefore, the performance of a shared-memory multiprocessor machine.

Hiding latency

Although it is usually most effective to solve problems at the root, in this case, to *reduce* latency, there are several techniques that are often able to effectively *hide* latency. A well-known technique is a *cache* which tries to hide the

latency of a DRAM access by keeping often-accessed data in a small, much faster SRAM memory. Caches can be used effectively in shared-memory machines also, provided coherence is maintained properly (if the system allows multiple caches to have a copy of the same address, these copies need to be kept synchronized). This way, in a typical configuration less than 1% of all memory accesses to remote memory cannot be satisfied by a processor's local cache and require network communication. However, due to the extreme difference in latency between a cache hit (a few nanoseconds for a level-1 cache) and a remote memory access (up to several microseconds), this 1% is enough to significantly restrict the performance of large parallel machines.

A technique to cope with high latency memory accesses is Out-of-Order (OOO) execution. A processor supporting OOO execution of instructions allows the execution of non-dependent instructions to be re-ordered. For instance, while waiting for a long-latency memory load operation, instructions that do not need the data that is being fetched, can be executed. This way, the long load is overlapped with useful computation, the processor stays busy and no time is wasted so the latency is effectively hidden. However, the number of instructions that do not depend on the load is usually limited to a few dozen. Also, the hardware needed to re-order instructions grows quickly when allowing it to look further down the instruction stream for finding non-dependent operations. In current microprocessors this *instruction window* is never more than a few hundred instructions large. This translates to an ability to overlap 10-100 ns of latency, insufficient to completely hide a single remote memory access.

In situations where communication is explicit (message-passing architectures), another method is possible: when the latency of a message is known at compile-time, the scheduling of communication and computation can be made such that messages are sent sufficiently in advance so that they are received by the target processor *before* they are needed. This mimics OOO execution in that communication and computation are overlapped, but can be done at much longer time-scales because the programmer (or compiler) has a much larger view of the program than the processor has (up to the complete program versus a few hundred instructions).

A shared-memory equivalent of scheduling communication is to insert prefetch instructions. These instructions start the load of a certain data word, so that it is already in the cache by the time the word is needed. It is however not always possible to predict in advance which data will be used later, or how much in advance the prefetch instructions need to be placed since both can be data-dependent, and vary when the same code is executed on machines with different implementations. Moreover, it requires

significant intervention by the programmer, who would again need an intimate knowledge of the target architecture. This is against the philosophy of the shared-memory paradigm, which states that communication should be implicit and invisible to the programmer, and that code should be portable across machines with different implementation details. The practical use of prefetch instructions to hide long-latency remote memory operations is therefore limited.

Communication network requirements

We have established that a good communication network provides low latency, so that remote memory accesses can be completed quickly and most time can be spent on computation, rather than communication. Part of this latency is limited by the technology available, another part can be influenced by choosing a good network topology and making other architectural decisions. The main goal should be to reduce congestion, as this is the main source of network delays that the network designer can affect. Congestion can be avoided if network traffic were to be spread out evenly through time (no sudden bursts of traffic) and space (no hot-spots in the communication pattern). Unfortunately, reality does not exactly cooperate.

As we will further explore in Chapter 3, the network traffic that can be measured on the communication network of a realistic shared-memory machine, running a real application, is far from uniform. Most nodes communicate mainly with just a few other nodes, causing hot-spots on some network links while others are barely used. Also, communication tends to occur in *bursts*. These are short periods of intense communication between two nodes, followed by long periods during which relatively little communication occurs. This makes the network traffic highly non-uniform in both space and time. Network links are thus loaded irregularly, periods of high load and high congestion are followed by periods of low load. Dimensioning all links for their peak loads would not be economical, since this peak load is attained only for short periods. Also, when viewing the network at one point in time, the different links of a network exhibit different loads. A network topology could be designed that would optimally accommodate this spatial traffic distribution, in combination with a process and data distribution that minimizes distances for most of the traffic. Unfortunately, the optimal topology and placement usually change quite radically through time.

Towards reconfigurable networks

An ideal solution for this problem would be a reconfigurable interconnection network: one that is able to change its topology through time, so that at each point in time the topology is optimal for the current traffic pattern. Solutions to this problem are being developed in the electrical domain. But in the near future, rising bandwidth density requirements will call for a replacement technology such as optical interconnects. It is the aim of this work to explore the possibilities of reconfigurable optical interconnects, and to describe and analyze a possible implementation of such a network using novel reconfigurable optical components, applied to the context of shared-memory multiprocessor machines.

1.2 Optical communication

I speak Spanish to God, Italian to women,
French to men, and German to my horse.
— Charles V

Electrical interconnects are running into several limitations. Differential latency (skew) limits the parallelism that can be used on connections of over a few centimeter, the serialization that is therefore required increases latency. Power requirements and cross-sectional area are steadily increasing. Even for short connections, such as those between a processor and its memory over a Printed Circuit Board (PCB), frequency-dependent losses are becoming a problem requiring the use of complicated techniques such as pre-emphasis. Cross-talk is a big issue, even when using Low Voltage Differential Signaling (LVDS). Since processing power, and therefore bandwidth requirements, are expected to keep increasing exponentially, a solution will have to be found quickly.

1.2.1 Data communication using light

An alternative to using electrical signals to carry information, is to use light. A *laser* provides a coherent light source, which is modulated – either by turning the laser on and off, or by using a separate modulator component – by the data stream that is to be transmitted. This light is sent through

a *waveguide*, such as an optical fiber, which guides the light to the receiver where a *photodetector* converts it back to an electrical signal.

Power usage, volume and cost of an optical connection depend on the required bandwidth and the distance the connection needs to span. Since those properties often scale slower for higher bandwidth or longer distances than for electrical connections, in many cases optics is the best choice, especially for high bandwidth, long distance connections. Other advantages of optical communication include the insensitivity to interference, both from adjacent connections – light cannot leave one optical fiber and enter an adjacent one[4] – and from external Electro-Magnetic Interference (EMI) caused by for instance the proximity of strong magnetic fields. Also, the information stream of several Gbps is modulated onto a carrier of a much higher frequency (usually infrared light with a wavelength of 850 or 1550 nm, which corresponds to a frequency of 350 or 192 THz, respectively). To the waveguide, the data itself only seems like a very small variation in frequency. It is therefore much easier to increase the bandwidth of the connection, by increasing the modulation frequency or by using Wavelength Division Multiplexing (WDM), without significant impacts on the required quality of the optical fiber. In contrast, electrical communication uses much more of the theoretically available bandwidth of, for instance, a coaxial cable or PCB track.

Fundamental bandwidth density limitations

The maximal bandwidth achievable over an electrical interconnection is limited by conductor and dielectric material imperfections. Miller and Ozaktas [1997] derived a theoretical limit on the information bandwidth attainable over a non-repeatered connection of a given length and available cross-sectional area, even when the physical parallelism realized within this area is optimized. The limit is of the form $B \sim B_0 A/l^2$, where B is the bandwidth, A the cross-sectional area, l the interconnection length and B_0 a factor in the range 10^6–10^9 Gbps (dependent on the feasibility of certain materials, topologies and features at different scales). This (presently still large) theoretical limit is reduced by dissipation and extra spacing required to limit crosstalk between adjacent interconnections. In practice, the bandwidth of electrical interconnections is therefore limited by the trade-off among interconnection length, available cross-sectional area and power dissipation.

For optical communication, this trade-off is different. Attenuation of light can be made to be as low as a few dB/km, and is almost indepen-

[4] This assumes the fibers are more than one wavelength (a few micrometers) apart. On a PCB, this is normally the case; on-chip, this requires more careful design.

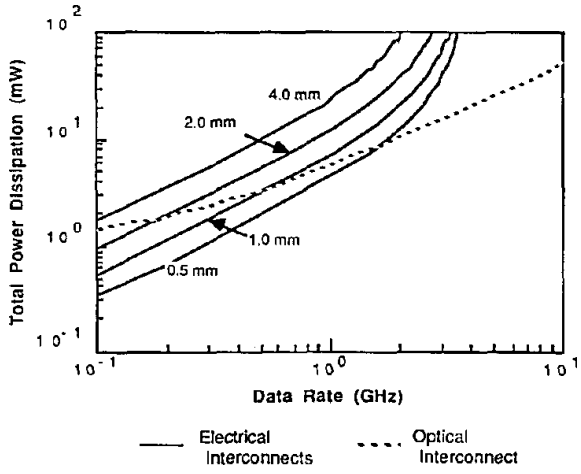

Figure 1.6: Power dissipation of electrical versus optical interconnects as a function of data rate [Feldman et al., 1988]

dent of the information bandwidth. The attainable bandwidth and required power of optical links therefore change much slower as a function of the interconnect distance. This means that, for a given bandwidth and available cross-sectional area, above a given distance optical communication will be at the advantage. Feldman et al. [1988] make such a comparison, shown in Figure 1.6. The optical power output by the laser does not vary (much) for changing distances or data rates, the electrical power of the electrical-to-optical conversion circuits changes only with data rate. In contrast, the power required by an electrical link grows (approximately) linear for higher data rates and quadratic with longer distances. This means that, for a given data rate, optical links will require less power than electrical links from a certain distance onward. Conversely, since the required bandwidth increases throughout the years to keep up with higher processing speeds and densities, the distance at which optics beats electronics is ever shorter.

This theoretical derivation is followed in practice: optics is being used for ever shorter ranged communication, as is shown in Figure 1.7. Intercontinental communication cables have been using optical technology since the 1980s. Currently optical Local Area Networks (LANs) are finding mainstream usage. Optical communication at the PCB level has been demonstrated in lab settings [Brunfaut et al., 2001; Mohammed et al., 2004; Bockstaele et al., 2004; Schares et al., 2006] and is currently making its way into commercial applications. Even for on-chip communication, optics may soon become reality [Haurylau et al., 2006; O'Connor et al., 2007; Van Campenhout, 2007].

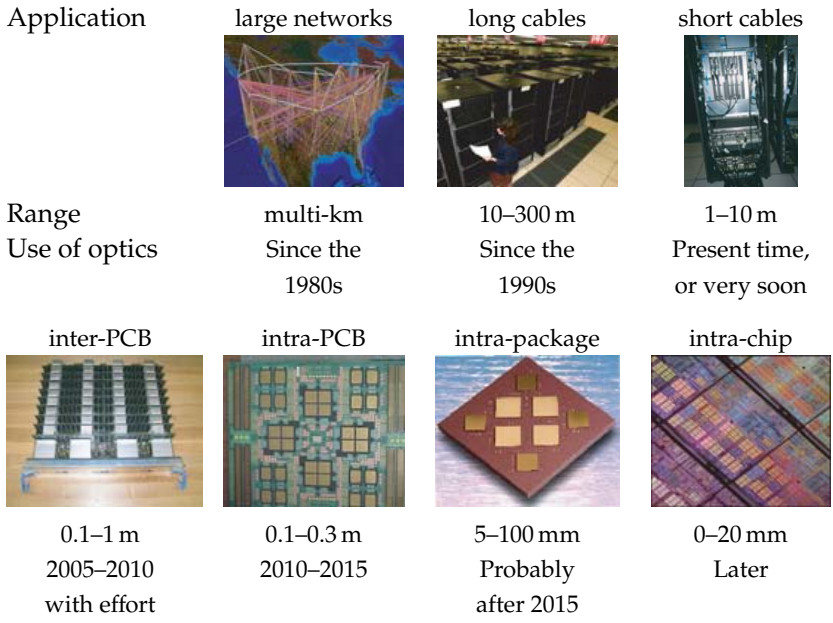

Figure 1.7: The physical interconnect hierarchy: digital interconnections can span widely different distances. Optical interconnections are being used over ever shorter distances [Benner et al., 2005].

1.2.2 Short range optical interconnects

When comparing the interconnect technologies used on different distance scales, one important shift in attitude becomes apparent. Information can never travel faster than the speed of light (c = 300.000 km/s, or about 3 ns per meter). At long distances (multiple kilometers) this introduces a delay of several microseconds or even milliseconds. This time dwarfs most other delays associated with the communication, such as the time required for serializing/deserializing, resynchronization, protocol overhead, etc. Therefore, a long optical link is optimized only for high bandwidth, not for low latency.

At short distances, this situation reverses. Here the communication link is designed on a much lower architectural level: the link is no longer a connection between different systems, but is an integral part of the system itself, connecting its components. To distinguish this from a communication link, which is designed as being external to the system, the term *interconnect* is often used in this context. In practice, this means that much more emphasis is placed on latency, rather than bandwidth (at this level of the system hierarchy, where components are closely connected, it is far more difficult to tolerate latency). And since the speed of light is no longer the main bottleneck (the time-of-flight for links of under one meter is less than 5 ns), reducing latency of the link components will constitute an important design effort [O'Connor, 2004].

To this end, short distance interconnects are often parallel, not just to obtain higher bandwidths but also because this avoids the delay of serialization and deserialization. From a physical point of view, the loss in the transmission medium has much less influence here: multi-km optical fibers typically have a loss of under 1 dB per kilometer, while short-range interconnects can tolerate several dB per meter. This allows for the use of much cheaper glass or even plastic optical fiber. It also allows more components (connectors, beam-splitters, ...) to be placed in the light path. Finally, component cost has a much higher influence. Multi-km links have only a few transmitters and receivers, these can be bulky (several cubic centimeters) and expensive (€1000+ per component). Inter-chip optical interconnect, on the contrary, requires hundreds of single-bit links between two chips placed only centimeters apart, so each component needs to be very small (often integrated into 2-D arrays, with each component occupying less than 1 mm^2) and cheap (at most a few euro per link).

A possible integration approach of such a chip-to-chip, optical interconnection is shown in Figure 1.8. 2-D arrays of Vertical-Cavity Surface-Emitting Laser (VCSEL) transmitters and photodetectors, each on a piece of III-V ma-

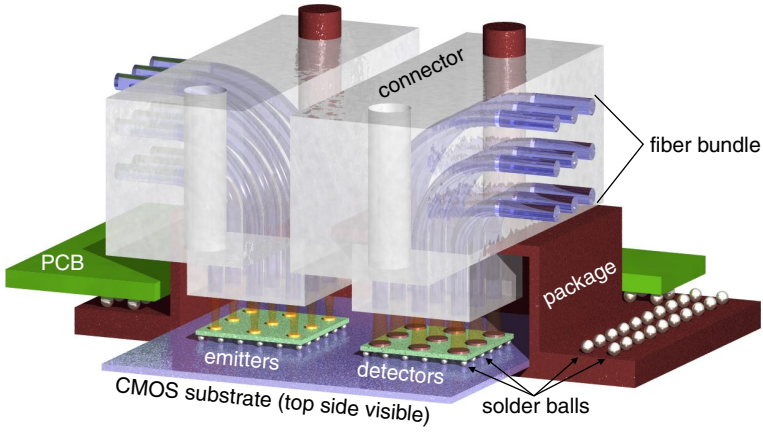

Figure 1.8: Artist's impression of a possible opto-electronic integration approach [De Wilde, 2007]

terial such as Indium Phosphide (InP) or Gallium Arsenide (GaAs), are flip-chip bonded onto a Complementary Metal-Oxide-Semiconductor (CMOS) chip which contains the computing elements, as well as the amplification and synchronization circuits completing the optical link [De Wilde, 2007]. This provides for a very tight integration between processors and their optical interconnections. From a manufacturability standpoint it would be beneficial to more tightly integrate the VCSELs into the CMOS chip. It is not feasible to make the VCSELs directly from silicon, since the indirect bandgap makes it inefficient to generate light with it. Therefore the hybrid approach using III-V compound semiconductors is required. At first those were wafer-bonded onto the CMOS die, later, heterogeneous integration allowed InP to be deposited directly onto a Silicon-on-Insulator (SOI) substrate so that VCSELs, photodetectors and SOI optical waveguides can be created entirely using CMOS compatible processing [Roelkens et al., 2007; Van Campenhout et al., 2008]. Still, all-silicon photonics, the holy grail of opto-electronic integration, is not impossible. The light-generation problem of silicon can be avoided by generating light using an external source, and modulating this light using silicon modulator components. This approach is being followed by Intel's labs [Mohammed et al., 2004], among many others.

Using this collection of technologies, a much closer pitch can be used (100 μm or less) than is attainable using electric pins, yielding potentially 10,000 links per chip allowing highly parallel connections to be made. Putting this many electrical data pins on a single chip is challenging at best, routing them on a PCB is next to impossible.

1.2.3 State of optics in computing

Optical interconnections are being used already in very large multiprocessor machines. Ethernet, the most popular technology for LANs, has optical varieties from the 1 Gbps standard onwards. From there, other technologies evolved that focus more on low latency, making them more suitable for tightly interconnecting processors into one large server or supercomputer (as opposed to networking multiple computers, which assumes a much looser coupling, both through higher hardware latency and on a software level). Notable examples include FDDI [1987] and Myrinet [Boden et al., 1995]. Optical Storage Area Networks (SANs) such as Inifiniband [2000], normally used to connect servers and their storage media, are also being used in for instance NASA's Columbia supercomputer [Biswas et al., 2005]. In most of these cases, optics is used for connecting racks of equipment into one very large supercomputer, or at the inter-board (backplane) level. The optical links are viewed as drop-in replacements for electrical cables, providing higher bandwidth and lower power requirements.

For shorter connections, between chips on the same PCB, only very few commercial examples can be found to this date. Yet the advantages are clear: higher parallelism lowers packet latency, larger bandwidth reduces congestion which again has a positive influence on latency. More connections per chip allow the construction of networks with higher fan-out, lowering the average inter-node distance. Also, since distance has much less influence on the design of a link, it is possible to use similar technologies both for short distances (processors on the same PCB) and for longer connections (between processors on opposite sides of the machine, up to several meters). It is therefore our expectation that optical interconnections are an ideal match for the problems currently arising in multiprocessor networks, and that more commercial applications of short-range (under one meter) optical interconnects will follow very soon [Collet et al., 2000; Benner et al., 2005].

1.2.4 Reconfigurable optical interconnects

Several components used for optical interconnect have properties that allow for runtime reconfiguration of the network. For instance, tunable VCSELs have the ability to change the wavelength at which they emit light. This is 'invisible' to the data stream with which the light is modulated. When this light of variable wavelength is sent through a prism or grating, different output destinations can be selected, effectively reconfiguring the communication network. An equivalent architecture can be implemented by broadcasting the light to several destinations, and equipping the receiving

end with photodetectors sensitive to just a small wavelength band. Other components such as Micro-Electro-Mechanical Systems (MEMS) switchable mirrors or liquid crystal components can be placed in the light path to deviate the light beam and again select different destination nodes in a data-transparent way.

Similar architectures could be constructed using only electronic components, such as FETs influencing a data path or even a full-blown multiplexer. However, these electrical solutions have a significant impact on the bandwidth and latency of the link being switched. The FET or multiplexer effectively works as a low-pass filter on the (base band) communication stream, placing an upper limit on the data rate.[5] In contrast, optical reconfiguration technologies have a very different way of affecting the signal. Since the data stream (several Gbps) is modulated on a carrier of much higher frequency (for instance 350 THz), the material response here is much more uniform. Dispersion does cause the upper and lower frequency components of the carrier frequency plus data signal to behave differently (difference in delay, loss). Shifting the carrier frequency using a tunable laser enlarges this effect. In practice, however, dispersion affects the maximum bandwidth much less than would be the case in an electronic reconfigurable system.

Another reason to choose optical reconfiguration over electrical becomes obvious when the communication links are optical anyway, to support the high data rates of the future. Conversion from the optical to the electrical domain and back is costly (conversion adds latency, power, component costs). If reconfiguration can also be done in the optical domain, this conversion step is avoided. In telecommunication, much research is done in Optical Packet Switching (OPS), the aim being to convert a signal to the optical domain once at the sender, do all routing through the network using optics, and only at the final destination convert the signal back to the electronic domain. An important research domain here is Optical Burst Switching (OBS) [Neilson, 2006; Shacham et al., 2005]. With this technique a large number of data packets with matching destinations are aggregated. This is needed to limit the overhead of for instance optical *labels*, which are used for routing the packet through the network. Of course, aggregating packets (i.e., intentionally delaying them) is not very useful when low latency is such an important design objective. This makes that techniques and results from the OBS research field will need to be adapted to be useful in an interconnect setting. Also, because it is applied to long-distance communication, OPS of-

[5] For instance, the clock speed attainable using a Field Programmable Gate Array (FPGA) is usually an order of magnitude slower than that of a hard-wired circuit – precisely because of the FPGA's reconfigurability, which introduces extra logic in the signal paths.

ten uses expensive components that are too bulky and power hungry. They are therefore not suited for on-board or on-chip integration.

Cheap, small and low-power components, acceptable for use in interconnects, have different characteristics. The tuning speed of a tunable laser for instance ranges from a few nanoseconds for telecom lasers [Akulova et al., 2002] to several hundred microseconds for VCSELs suitable for interconnect [Chang-Hasnain, 2000]. This means that, in short-range applications, packet switching using tunable lasers is probably not feasible, because in this situation the laser would need to be re-tuned for every packet (the total time a packet spends in the network is usually no more than a few hundred nanoseconds). Other components mostly have the same limitations. Liquid crystals suitable for use in reconfigurable optical interconnects, for instance, have a switching speed of several milliseconds [d'Alessandro and Asquini, 2003]. Finally, several telecom-related switching techniques require strongly coherent light sources. Cheap VCSELs and multimode, or even plastic, optical fibers used for interconnect cannot provide this. Therefore, alternative schemes to optical packet switching will have to be used, such as circuit switching, which keep the network configuration constant for longer periods of time.

1.3 Reconfigurable processor networks

In Section 1.1.3, we established that shared-memory multiprocessor machines have difficulties tolerating high latencies in their interconnection network. Communication times are often significant compared to computation times. Moreover, the very nature of the shared-memory programming paradigm, which insulates a programmer from the implementation details of the machine, makes it difficult to schedule communication effectively in order to hide latency. Further, the traffic pattern that is exhibited on the communication network is far from uniform. This makes a static, uniform network a bad match for the requirements imposed by the network traffic, but suggests that reconfigurable networks should be used instead. Finally, since processing power and densities will continue to rise, electrical interconnects are expected to reach their limits very soon, a problem that will likely be solved by using optical interconnection networks.

Therefore, we feel that using reconfigurable optical interconnection networks inside shared-memory multiprocessor machines is a viable solution. It is this idea that is explored, at an architectural level, in this thesis. Our goal is to present a reconfigurable network architecture, that can be implemented

using existing and future components that are suitable for use in a low cost, reliable interconnection network. We will characterize the performance of this proposed architecture under several workloads. To do this characterization effectively, we also propose a number of techniques that can speed up the evaluation of these networks. Since our techniques can present a more realistic workload to the interconnection network under test, compared to existing methods of network evaluation, they should also prove useful in the design of other, non-reconfigurable or non-optical on- and off-chip networks.

If a reconfigurable architecture is to be suitable for use in a short-range interconnection setting, then all components, including those used to implement the reconfiguration, must be small, cheap, low power, etc. The main challenge here will be that this implies a limited reconfiguration speed. This means that reconfiguration will have to take place at time scales significantly longer than the life of a single network packet (hundreds of nanoseconds) or even a remote memory access operation (up to a few microseconds). Still, as will be evident from our characterization of network traffic in Chapter 3, there is enough locality to be found at longer time scales, making slow reconfiguration a viable method for improving network performance.

1.4 Contributions

> *The scientist is not a person who gives the right answers, he's one who asks the right questions.*
> **— Claude Lévi-Strauss**

This section gives a short overview of the research work performed for this thesis. Through the IAP program, a close collaboration was maintained with the TONA department of the Vrije Universiteit Brussel (VUB), headed by prof. dr. ir. Hugo Thienpont. For joint efforts, their contributions are acknowledged. References are provided for the main publications where this work first appeared. A more detailed discussion is given from Chapter 3 onwards.

1.4.1 Communication requirements

Existing works by for instance Duato et al. [2003] or Dally and Towles [2004] acknowledge that the destinations of network traffic on multiprocessor

interconnection networks are not uniformly distributed. Characterization of a network architecture is therefore often done using a mix of uniform and hot-spot traffic. How this mix varies through time, or how the hot-spot destination nodes change, is usually not very influential on static networks and is therefore not considered. For reconfigurable networks, however, this is a very important property: the network must be able to reconfigure itself fast enough to cope with the dynamics of the network traffic.

Therefore, first of all we have characterized the dynamic behavior of network traffic, during the execution of a range of benchmark application programs running on the multiprocessor machine. We found that this communication tends to happen in bursts of elevated communication between node pairs, on a background of more uniform traffic of lower intensity. Each burst can last up to several milliseconds, multiple bursts can be active on the network at the same time. These bursts occur both inside applications [Heirman et al., 2005], and after context switches if multiple applications are executed on a single system [Artundo et al., 2006b].

Acknowledgements The work regarding traffic behavior during context switches was performed as a Masters thesis by ir. Daniel Manjarrés at the VUB, under guidance of ir. Iñigo Artundo and dr. ir. Christof Debaes. Analysis of communication within programs was a personal effort.

1.4.2 Reconfigurable architecture

Next, a reconfigurable network architecture was created. The choice was made to have two basic parts in the architecture: a fixed, non-reconfigurable base network of regular topology, and a reconfigurable part that would make extra links or *elinks* between those node pairs involved in a communication burst. This way, a lot of freedom was given to the reconfiguration algorithm. Since the base network is always available, no care needs to be taken to keep all nodes connected or to always provide a reasonable communication bandwidth among all nodes – the algorithm can completely concentrate on the speedup of communication bursts. How many elinks a given architecture can provide, and the limitations on which elinks can be active concurrently, are left as parameters that will be determined by an actual implementation. One such implementation has been proposed in [Artundo et al., 2006a] using a Selective Optical Broadcast (SOB) device.

Acknowledgements The general architecture was conceived jointly with my advisor dr. ir. Joni Dambre, for the selective broadcast implementation I had input from ir. Iñigo Artundo. The SOB device was developed at the

VUB, initially by dr. ir. Lieven Desmet and is now being completed by ir. Iñigo Artundo.

1.4.3 Tools for design-space exploration

Since the temporal behavior of the network traffic is decisive for performance, any experiment that aims to characterize network performance should keep this temporal behavior intact. In practice, this usually means that the benchmark application that generates the network traffic must be run entirely. Existing techniques such as sampling change the burstiness of the communication pattern and are therefore not suitable, leaving network designers with full simulations as their only choice once a certain level of accuracy is required. Therefore, there was room for a set of new techniques to speed up the evaluation of reconfigurable networks. Three such techniques are explored as part of this dissertation. Two of them are performance prediction tools. The first one is a memory access latency prediction tool, published in [Heirman et al., 2008a], with an earlier version appearing in [Heirman et al., 2007b]. A second prediction model accounting for congestion is described in [Heirman et al., 2006]. The last technique is a method of statistically generating network traffic with the required temporal behavior, which allows one to use a shorter trace than a complete execution of the benchmark program [Heirman et al., 2007c]. Together, they form a range of techniques to evaluate reconfigurable network performance, each of them with a different trade-off between simulation speed and accuracy. They can help a network designer through the different stages of a design-space exploration, and were also helpful during this PhD thesis to explore the performance of reconfigurable networks.

Acknowledgements ir. Wouter Rogiest and ir. Koen De Turck from the Stochastic Modeling and Analysis of Communication Systems (SMACS) group, Ghent University, refreshed my knowledge on the queueing theory used in the congestion model. The formulation and evaluation of the exploration tools are personal efforts.

1.4.4 Performance evaluation

Armed with a simulation environment for doing detailed, accurate simulations, and a set of techniques for fast design-space explorations, we were able to evaluate different designs of reconfigurable networks being subjected to the traffic of a range of benchmark applications. A number of design trade-offs were investigated, including the limitations imposed by both the SOB

design [Artundo et al., 2006a] and by the heuristic reconfiguration algorithms that were used in the implementation [Heirman et al., 2008a]. How this idea scales to larger networks was explored in [Heirman et al., 2007a].

Acknowledgements The reconfigurable network implementation using the SOB device was modeled, simulated and characterized in cooperation with ir. Iñigo Artundo from VUB. The simulation infrastructure was a personal effort, and built upon Virtutech's Simics simulator [Magnusson et al., 2002].

1.5 Structure of this thesis

We begin, in Chapter 2, with an overview of related work, including a brief survey of the components that can be used in implementing reconfigurable interconnection networks. Since the behavior of network traffic at different time scales is very influential, we devoted Chapter 3 to its analysis. In Chapter 4, details are given about the reconfigurable network architecture we envisage, the implementation of an optical reconfigurable network using the SOB device from the VUB, and the simulation framework that was the basis of most of the experimental results throughout this thesis. Chapter 5 describes our techniques to speed up design-space exploration. Chapter 6 uses these techniques, in addition to classical full-system simulations, to evaluate the performance gain possible when using reconfigurable networks. Finally, in Chapter 7 conclusions are made and some links to current and future work related to reconfigurable interconnection networks are provided.

2
Related work

Mathematitians stand on each other's shoulders,
while computer scientists stand on each other's toes.
— R. Hamming

In this chapter, an overview is given of existing work related to this thesis. First we visit the background of optical interconnect research, and provide references to system level overview papers, demonstrators, and reconfigurable architectures. Next, the optics-related research done at the PARIS group, where this PhD work was performed, is summarized. Finally, Section 2.3 provides some details on reconfigurable components that can be used in the implementation of a reconfigurable optical network.

2.1 Optical interconnects

The work in this thesis combines several aspects: the view, at a system level, of reconfigurable optical interconnections applied to large, shared-memory parallel computer systems, focusing mainly on the interactions between traffic patterns and the interconnection network. This combination has not been explored in detail before. Prior art does exist in each of these sub domains.

2.1.1 System level

At the system level, Collet et al. [2000]; Trezza et al. [2003]; Huang et al. [2003] and Benner et al. [2005] each explore the potential of optical interconnections in several types of computing systems. All agree that optics can be used effectively in the class of tightly coupled, medium to large scale SMP systems, in the next five years. On-chip, a similar trend is visible: long, chip-spanning connections are already slower and more power-hungry than designers would like, optical replacements are therefore predicted in the near future by O'Connor [2004]; Kirman et al. [2007] and many others.

Optical interconnects are also being heavily investigated in core network routers. There, bandwidth requirements are very high, and especially at the backplane level optics can solve the interconnect density problem that designers currently face. Since long-distance communication is already optical and most connections to and from this type of router are fibers, continuing the use of optics further inside the machine is only a logical path. This idea is explored by for instance Neilson [2006].

2.1.2 Optical interconnect demonstrators

Short-range optical communication, between chips on adjacent PCBs, or even on the same board, requires the interconnect to be closely integrated with the Very Large Scale Integration (VLSI) chip. This is so on an architectural level (the interconnect is an internal part of the system, rather than a connection to the outside world), but also on a physical level. Indeed, going off-chip through electrical pins to a separate optical transceiver module adds latency, design effort and again limits the available bandwidth to that of the pins. A solution providing much tighter integration, which allows for higher bandwidth and smaller physical dimensions, is to integrate the optical components on the VLSI chip.

This approach has been explored in the European projects "Optically Interconnected Integrated Circuits" (OIIC) and its successor "Interconnect by Optics" (IO), using a setup much like the one depicted in Figure 1.8. The PARIS research group, where this PhD research was done, took part in both these projects. The OIIC project (1996-2000) proved the feasibility of providing on-chip optical access by connecting two FPGA chips in 0.35 μm CMOS technology through a 2-D-array of 8×8 optical channels, providing an aggregate bandwidth of 19 Gbps spanning a distance of 12 cm, with only 10 ns latency for the complete link, including transmitter and receiver circuits [Brunfaut et al., 2001].

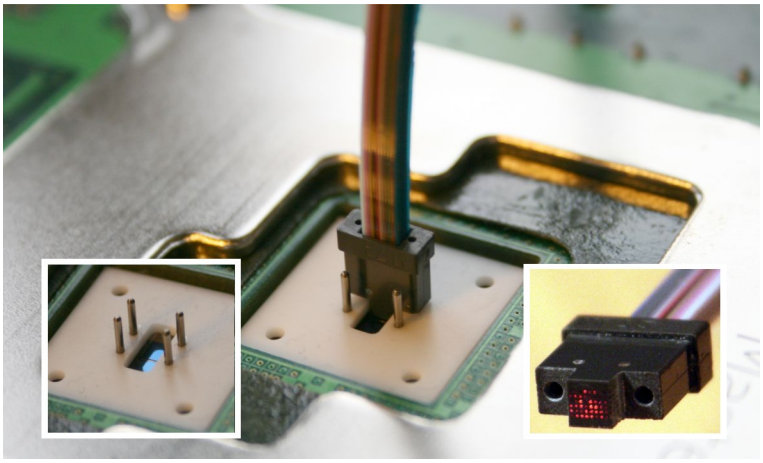

Figure 2.1: Fiber ribbon cable connected to the IO demonstrator chip [De Wilde, 2007]

In the IO project (2001-2005) emphasis was laid on the manufacturability of such a setup, solving practical problems like connectors and fiber alignment (see Figure 2.1) so that commercial application of this technology would be possible [Rits et al., 2006].

Several other projects of similar nature are or have been ongoing, both in academic settings and in industrial labs. For instance, Schares et al. [2006] describe the Terabus project performed at IBM Research, in which a set of Terabit/second-class optical backplane interconnect technologies was developed. Mohammed et al. [2004] report on the current state of Intel's research in optical interconnections.

2.1.3 Reconfigurable network architectures

On the reconfigurability side, early work done by Snyder [1982] introduces the reconfigurable computer. He notes that an adaptive network can make a parallel computer achieve good performance for a range of applications, each possessing different communication patterns. Garcia and Duato [1993] explore routing and reconfiguration algorithms for reconfigurable transputer networks. Pinkston and Goodman [1994] describe the GLORI system, a possible implementation of an architecture similar to Snyder's, using optical reconfiguration technology.

Recent attempts to implement optical reconfiguration of interconnection networks include the chip-to-chip (N-to-N) reconfigurable optical intercon-

nect reported by Aljada et al. [2006]. Here, the reconfiguration is done by two arrays of Liquid Crystal (LC) cells driven by a VLSI circuit placed on the side of a prism-like structure. They generate digital holographic diffraction gratings to steer and multicast optical beams, this way the paths of 2.5 Gbps optical links are switched.

Another, 1.25 Gbps free space 1-to-N reconfigurable optical interconnect is described by Henderson et al. [2006], employing VCSEL-PIN links, and binary phase gratings on a ferro-electric LC Spatial Light Modulator (SLM). Beams are steered free-space over a surface of 6.4 mm^2 with a resolution of 50 μm.

The "Self-organized micro-opto-electronic system" (SELMOS) is another board-level reconfigurable optical interconnect over film–waveguide 3-D structures [Yoshimura et al., 2003]. It is composed of an embedded 3-D 1024×1024 micro-optical switching system based on waveguide-prism-deflector switches, and a self-organized optical network, that couples light paths between two waveguides automatically.

Other architectures, some of which have been developed into demonstrators, include the simultaneous optical multiprocessor exchange bus (SOME-bus) from Drexel University [Katsinis, 2001], the optical centralized shared bus from the University of Texas at Austin [Han and Chen, 2004], and Columbia University's data vortex optical packet switching interconnection network [Hawkins et al., 2007].

2.2 Photonics projects at ELIS

This PhD work was performed at the PARIS research group of the ELIS department at Ghent University, Belgium. This group has been involved in a number of projects on optical interconnects. A short overview of those projects is now given, each of which in some way influenced this work.

2.2.1 IAP PHOTON and photonics@be

The "Inter-university Attraction Poles" program, funded by the Belgian Federal Science Policy Office, aims at giving a temporary impetus to the formation of inter-university networks of excellence in basic research. Phase V ran from 2002 to 2006, in the 18th *PHOTON* network seven Belgian, Dutch and French universities were combined under the title "Photons and Photonics: From basic physics to system concepts." Work package 4 of this project, "Reconfigurable Optical Interconnects," combined research into both the

components needed to perform optical reconfiguration, and this work on the architectural exploration of reconfigurable networks for multiprocessor systems. In 2007, this research was continued through the next phase of the IAP program as IAP VI-10 "Photonics@be: Micro-, nano- and quantum-photonics."

Frequent contact with research groups doing the actual physics, and designing optical components and optical interconnection links, proved very valuable to provide me with background information and kept me in touch with developments on those fronts.

2.2.2 Optical demonstrators

Two optical chip-to-chip interconnect demonstration projects in which the PARIS research group took part, OIIC and IO, have been mentioned in Section 2.1.2. Both projects were co-operations between industry and academia throughout Europe. Much of the VLSI design, and the final integration and characterization of the prototypes for both projects was done at the PARIS group. Also, the link uniformity among the IO demonstrator's 64 channels was studied extensively. This property will dictate, for instance, to what extent clock synchronization circuitry can be shared between parallel channels. This would result in large area and power savings, and turns out to be within reach in carefully designed systems [De Wilde et al., 2008].

These projects, tightly integrating VLSI chips with optical interconnections, are closely related to how an actual implementation of this work might look. The values of several technological parameters that were required in this work have therefore been chosen based on the properties of these demonstration projects.

2.2.3 On-chip

Following the expected trend for ever shorter ranged optical interconnects, the OIIC and IO projects were followed by the on-chip optical interconnect project "Photonic Interconnect Layer on CMOS by Waferscale Integration" (PICMOS) (2004-2007). Here a photonic communication layer was integrated onto a CMOS chip, for communication over distances of at most a few centimeters. Since the end of OIIC and IO, the technology allowing heterogeneous integration of InP on SOI substrate had advanced enough so the VCSELs and photodetectors could now be placed anywhere on the chip, without the need for separate, flip-chip bonded III-V wafers. On-chip, optical point-to-point communication thus became a reality.

The PARIS research group was responsible for setting up a design methodology for the optical interconnects [O'Connor et al., 2007]. This way, optical connections can be designed, analyzed and simulated, in a manner not too different from electrical interconnections. This drastically lowers the design efforts, paving the way for broad adoption by chip designers.

PICMOS is being succeeded by the "Wavelength Division Multiplexed Photonic Layer on CMOS" (WADIMOS) project (2008-2010) which aims to create an optical Network-on-Chip (NoC), integrating an 8×8 optical router and all necessary optical and electronic components onto a highly integrated System-on-Chip (SoC), such as the chip powering a set-top box. When applied to a Multi-Processor SoC (MPSoC) and equipped with tunable VCSELs, this setup would enable an on-chip reconfigurable optical multiprocessor network, effectively realizing a single-chip version of the architecture proposed in this work.

2.3 Reconfigurable components

This section gives a non-exhaustive overview of the optical components that can be used to implement a reconfigurable network. Since this work is focused more towards computer architecture, rather than micro-optics, our description will mainly focus on the properties that affect the network's architecture, of which reconfiguration time is the most important. References are provided in which a detailed explanation of the components' operation and design trade-offs can be found. Also, the fabrication costs, reliability, etc., can vary wildly, and are in constant flux due to ongoing developments. Therefore, we will not try to come up with *the* optimal reconfigurable network architecture using one specific component (although, as an example we do analyze one possible implementation in more detail, see Section 4.2), but rather give a general overview of some components and architectures. A more thorough evaluation is left as future work, and will have to be done with more specific consideration of the cost, reliability, performance and other requirements imposed by the envisioned application.

2.3.1 Tunable VCSELs

Vertical-Cavity Surface-Emitting Lasers (VCSELs) consist of two Distributed Bragg Reflector (DBR) mirrors parallel to the wafer surface with an active region for the laser light generation in between. The DBR mirrors consist of layers with alternating high and low refractive indices. Each layer has

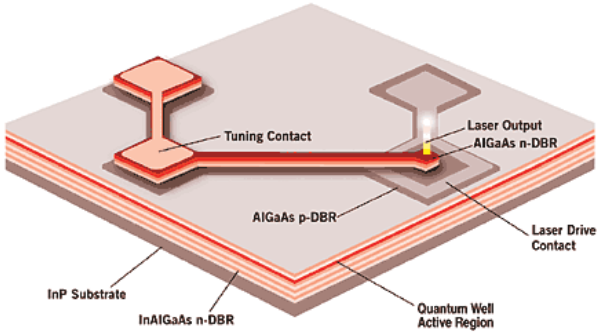

Figure 2.2: Schematic of a cantilever MEMS tunable VCSEL [Nabiev and Yuen, 2003]

a thickness of a quarter of the laser wavelength, resulting in a very high reflectivity. The main advantage of VCSELs is that they emit light vertically (i.e., perpendicular to the plane of the substrate). Therefore, they can be tested in an early stage of fabrication (edge-emitting lasers have to be cut out first) which reduces their cost, and allows a tight integration into 2-D arrays with a pitch of less than 100 μm. This makes them ideal for low-cost, high bandwidth-density applications such as short-range optical interconnect.

By changing the distance between the DBR-mirrors, the wavelength of the VCSEL can be tuned. One way to do this is to suspend the top mirror freely above the laser cavity, supported via a cantilever structure (see Figure 2.2). By applying a voltage difference, the cantilever can be moved electrostatically, changing the cavity length and thus the VCSEL's output wavelength. The tuning speed is determined mostly by the dimensions of the cantilever structure (usually between 25–100 μm), and is typically in the range 100 μs–10 ms. Depending on the distance over which the mirror can be moved, a tuning range of 10–50 nm can be achieved [Chang-Hasnain, 2000; Nabiev and Yuen, 2003; Filios et al., 2003; Koyama, 2006].

Faster tuning

Instead of the cantilever DBR-mirror, other approaches are also possible. Huang et al. [2008] describe a tunable VCSEL in which the top mirror is replaced by a high-contrast sub-wavelength grating (HCG), which is much smaller (a factor of ten in all three dimensions). The size, weight and thus the tuning times are therefore a thousandfold smaller than that of a classic DBR-mirror based VCSEL.

Another method is to divide the DBR-mirror into multiple sections, each of which can be activated independently which changes the laser's wavelength in discrete steps [Akulova et al., 2002]. Since no moving components are involved, tuning can be as fast as a few nanoseconds. However, very complicated processing technology makes their manufacturing difficult [Nabiev and Yuen, 2003], the application of these types of fast tuning lasers is therefore limited to telecommunication applications.

Finally, one can place multiple VCSELs with different wavelengths in such an arrangement that they emit their light into a single fiber [Hu et al., 1998]. If all these lasers are activated at the same time, this enables an integrated WDM approach which can be used if even higher bandwidths are required. By using one laser at a time, a tunable laser is created in which the wavelenght switching happens electronically, i.e., almost instantaneously. However, this approach only supports a small number of discrete wavelengths, rather than the continuous range of wavelengths obtainable from a regular tunable VCSEL.

2.3.2 Liquid crystal switches

Liquid Crystals (LCs) are most known for their application in displays. They consist of a thin layer of molecules, sandwiched between two glass plates. When a voltage is applied over the plates, the molecules align themselves with the electric field. When the field is removed, the molecules slowly (in a matter of milliseconds) return to their initial state. Depending on the orientation of the molecules, a polarized light beam undergoes a change in polarization. In combination with fixed polarizers, an element is thus created that can be made transparent or opaque in response to an electric field, making up the basic element or pixel of a display. Using this on-off capability of a LC, an optical switch is made which may form the basic building block of a reconfigurable network [d'Alessandro and Asquini, 2003].

Another property of liquid crystals becomes apparent when they are used *in-plane*, i.e., light is transmitted parallel to the glass plates: the refractive index changes in response to the applied voltage [Beeckman et al., 2006]. This effect can be used to steer an optical data beam towards one of a number of output channels, providing an alternative reconfigurable network building block.

The main problem, when used in our context, is the slow switching speed of liquid crystals. Especially the relaxation phase can be slow, since, after removing the electric field, one essentially has to wait until the molecules have reverted to their initial orientation, which can take up to 10 ms. A possible solution can be found in *ferro-electric* liquid crystals. In contrast to the

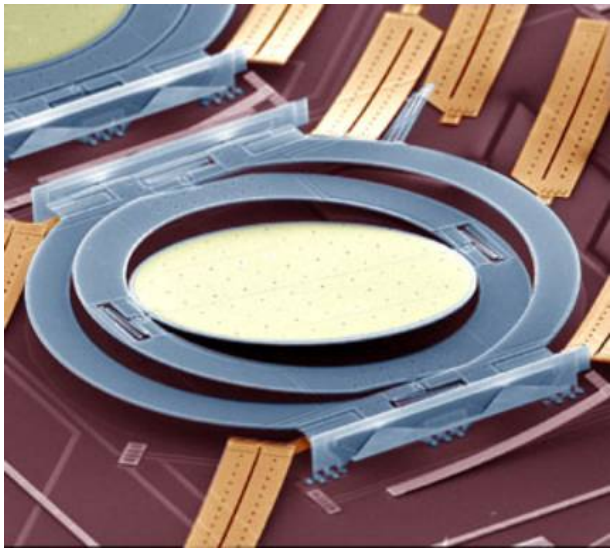

Figure 2.3: SEM micrograph of a 500-μm diameter, 45° rotating beam steering MEMS mirror [Lucent Technologies]

more traditional, *nematic* LCs, the ferro-electric variety is bistable: they can be alternated between two states with different polarization, both of which require a different and opposite electrical field. Because both orientations are now *forced* by the field, faster switching times (around 10–100 μs), are possible [Clark and Handshy, 1990; Gros and Dupont, 2001]. Optical switches using this technology have been demonstrated [Crossland et al., 2000; Aljada et al., 2006; Henderson et al., 2006].

2.3.3 MEMS mirrors

With Micro-Electro-Mechanical Systems (MEMS) technology, small (500-μm diameter and less) movable mirrors can be created that rotate in response to an electric field, with a 400 μs switching time [Lee et al., 1999]. This way, either an on-off or a 1×2 switch can be made. Pardo et al. [2003] use this basic building block to build a completely non-blocking, single-stage optical cross-connect with up to 1100 in- and output ports. They claim complete data transparency (bit rate and data format independence), compactness, moderate switching speed and high reliability. As with most of these components, which are targeted towards the telecommunications market, their cost may be an issue though for applications in an interconnect

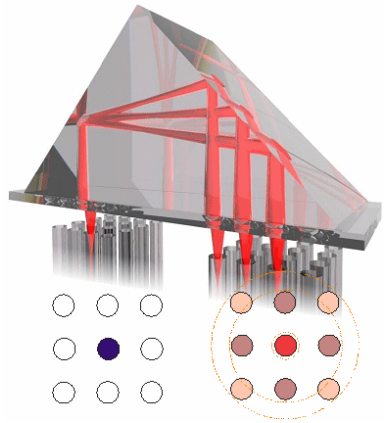

Figure 2.4: The Selective Optical Broadcast element distributes the signal from each input source to nine destinations [Artundo et al., 2006a].

setting. Devices with a 2-D array of these mirrors are used as digital cinema projectors (e.g., Texas Instruments' Digital Mirror Device [Dudley et al., 2003]). Evolutions of this technology, enabling wide-spread use in home cinema projectors, could at the same time make this technology available for short-range interconnects.

2.3.4 The SOB system

A technique aiming for low-cost applications and specifically targeted to optical interconnect, Deep Proton Writing (DPW), is in use at the Vrije Universiteit Brussel (VUB). This technique uses a focused ion beam, with energies of a few mega-electron volt (MeV), to *write* into a piece of PMMA plastic [Van Kan et al., 1999]. This way, features smaller than 100 nm can be created, which is suitable for the definition of micro-optical components. Since the ion beam has to be scanned over the component, the process is rather time-consuming (several hours of access time on a proton accelerator is required). It is therefore in itself not a cheap technique, although it does allow one to rapidly create prototypes, which can later be the master component for mass fabrication using cheap replication techniques [Debaes et al., 2006].

One set of components fabricated using DPW is the Selective Optical Broadcast (SOB) system [Artundo et al., 2006a, 2008b]. It is composed of a right angle prism, an input array of diffractive microlenses with a 3×3 splitting grating and an output array of refractive microlenses. This way

it implements a *selective broadcast*, in which light from the input sources is broadcast each to its own (overlapping) subset of destination nodes (see Figure 2.4). Since the number of broadcast destinations is fixed rather than growing with the total number of outputs, this device allows for scalable designs. This is to be put into contrast with for instance a star-coupler which does all-to-all broadcasting. In this case the required transmitted power, and the number of required wavelengths if this is to be combined with a WDM approach, grow linearly with the number of destinations.

2.4 Thread and data migration

An existing technique to cope with changing network requirements is the relocation, at runtime, of threads across processors and blocks of data across memories [Chandra et al., 1994]. This way, the same goal as with reconfiguration can be reached: communication over long distances is minimized.

Moving threads is usually done at context switch granularity (tens of milliseconds). If a thread is observed to communicate heavily with a network node that is far away, the operating system can consider to move the thread to this same node, or to a node closer to the node the thread is communicating with. Data migration is done by changing the virtual-to-physical address mapping – this way it can be done transparently to the application – which occurs with page size granularity (8 KiB in our study). Simultaneously the data is copied to the main memory of another node.

Some studies of data migration on existing machines can be found in [Chandra et al., 1994; Verghese et al., 1996; Jiang and Singh, 1999; Noordergraaf and van der Pas, 1999]. Specifically, Verghese et al. [1996] describe the implementation of page migration and replication on the Stanford FLASH [Kuskin et al., 1994] machine. The algorithm's prevalent time is the *reset interval*, which is set to 100 ms. During this time, a memory page needs to be accessed a minimum number of times in order to be considered for migration. Clearly, this method will not be of benefit when communication behavior changes at a time scale faster than those 100 ms.

The obvious advantage of thread and data relocation is that only software is needed instead of expensive additional hardware. The drawback of a software based approach is that one is still limited to the topology of the network. Also, the moving of tasks and data occurs at a time scale of seconds to minutes. This is quite slow compared to the typical reconfiguration speeds of the optical components described above. Thread and data migration is therefore less suitable to respond to rapid changes in application behavior.

3
Traffic patterns

We are drowning in information
but starved for knowledge.
— John Naisbitt

In a typical multiprocessor, millions of packets transit the network every second. In order to gain insight into how this network traffic interacts with the network resources, we have to look for patterns in the traffic. This chapter is devoted to analyzing network traffic. How traffic patterns change at different time scales is studied in Section 3.1. Section 3.2 will in more detail examine traffic bursts, which is the main mechanism causing locality that will be exploited by reconfiguration in this work. Next, Section 3.3 compares thread and data migration, which is the most important existing technology used to accommodate changing traffic patterns, to reconfigurable networks, and shows its complementarity to our own approach. In Section 3.4 we will make the case for our reconfigurable networks, and provide the background for the architectural decisions we made. The architecture itself of our reconfigurable networks will be detailed in Chapter 4.

3.1 Time scales

Time discovers truth.
— Seneca

Network traffic can be viewed at at different time scales. At each of these scales, a different mechanism is at play providing structure to the network traffic, and, if understood by a network designer, providing insight into how traffic and network interact. This in turn can lead to opportunities for improving network performance. Figure 3.1 summarizes several mechanisms and the time scales at which they occur, and provides a comparison with the switching times of several optical components that can be used in an implementation of a reconfigurable network.

In Sections 3.1.1 and 3.1.2 we briefly survey how network packets are routed, and how several packets take part in the execution of one remote memory access operation. We limit ourselves here to how this was implemented in our simulator, detailed information and other design options can be found in text books such as [Culler and Singh, 1999] for multiprocessors in general, or [Dally and Towles, 2004; Duato et al., 2003] which deal with interconnection networks specifically. Section 3.1.3 introduces communication bursts: these are periods of elevated communication between node pairs, caused by regularity in the application running on the multiprocessor machine. Next, Section 3.1.4 looks at the operating system level and finds that bursts can also be the result of context switches [Artundo et al., 2006b]. Finally, Section 3.1.5 looks at the applications themselves and how regularity in the algorithms they implement again gives rise to communication bursts. For each time scale, we consider what reconfiguration opportunities exist.

3.1.1 Packets: nanoseconds

In Section 1.1.3 we saw that not all nodes can be connected directly. Therefore, when two nodes without a direct connection want to communicate, one or more intermediate nodes have to forward the information. To this end, all information is sent in *packets*. Each packet has a *header* which identifies the source and destination nodes. A *routing protocol* defines what path packets should follow between given source and destination nodes. Routing can be *deterministic* or *adaptive*: with deterministic routing the path between a given node pair is fixed and determined at design time. Adaptive routing

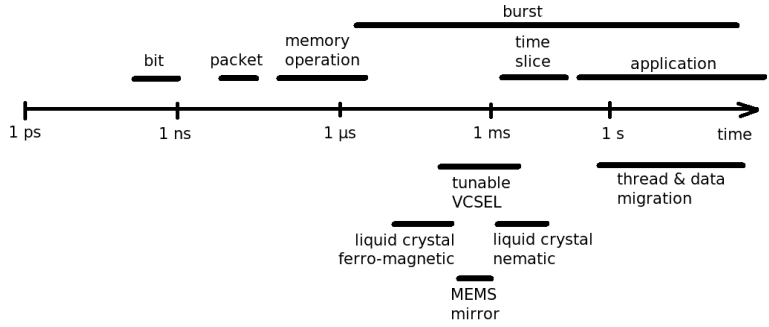

Figure 3.1: Time scales relevant in multiprocessor network traffic. Top: mechanisms of locality in network traffic. Bottom: reconfiguration time of some optical components that can be used to implement a reconfigurable network. Thread and data migration is an existing software technique and is included for comparison.

allows the network to change the path, for instance to avoid congested links or to route around failures within the network. *Minimal* routing means that all packets follow the shortest possible path (there may be multiple shortest paths, adaptive routing will choose one of them). With non-minimal routing packets can follow longer routes, which on a congested network can still be the fastest path.

In contrast to telecommunication networks, for interconnection networks low latency is preferred so routing decisions should be made very quickly. Also, the routing protocol is implemented in hardware. Therefore, the solution that is used is usually rather simple. In this work, *deterministic, minimal* routing is used, so packets between any given node pair always follow the same path, which is also (one of) the shortest.

Reconfiguration opportunities Since the link usually has (much) less parallel connections than the size of the packet, each packet takes multiple clock cycles to transmit. This introduces a form of locality: once the header of a packet traverses a certain link, we know the remainder of the packet will follow. A reconfiguration opportunity here would be to read the packet destination, and reconfigure the network such that this packet arrives at its destination faster. Optical Packet Switching (OPS) can be considered a form of this: the destination is read optically, components downstream then route the packet to the right destination without the need for conversion of the packet to the electronic domain.

Yet, the time one packet spends inside the network is in the order of a few tens to hundreds of clock cycles, which, at current several GHz clock rates,

corresponds to only a few hundred nanoseconds. This limits OPS, or any other reconfiguration method at the packet level, to the telecommunication domain where expensive components with fast reconfiguration times are available.

3.1.2 Memory accesses: microseconds

Cache coherence throughout the machine is maintained by a directory based coherence protocol. This protocol is executed by a hardware entity on each node that is connected to the local caches and memory on one side, and interfaces to the communication network on the other side. By sending and receiving network messages, care is taken that multiple copies of the same data word in the caches on different nodes always have the same value.

Main memory, which is globally accessible and logically uniform, is physically distributed over all nodes of the machine. Each block of memory, on a cache line granularity (64 bytes for the UltraSPARC processor, which is used throughout this work), has a *home node*, this is the network node where this data block resides in main memory. Other nodes are allowed to keep copies of this block in their caches for efficient access, but have to tell this to the home node, which keeps a *directory* of where all its blocks are cached. When a node wants to write to a data block, it can request *exclusive access*: all copies of this block in other nodes are invalidated (i.e., removed from their caches), the directory contains the information that the contents in main memory for this data block is no longer up to date but that the actual value resides in the cache of the *owner* node. When some other node again wants to access this data, the owner node has to relinquish its exclusive access and send the new contents of the data block back to the home node, which updates the data in main memory and also forwards the data to the node requesting access. Each data block can thus be in three different states: *uncached* (no node has the block in its cache), *shared* (one or more nodes have the block cached with read-only access permission), or *modified/exclusive* (exactly one node has the block cached with read/write permission, the contents of this block in main memory are outdated).

When we translate these activities from the coherence protocol to network packets, we get the following packet sequences (also shown in Figure 3.2):

- A request `REQ` is sent from the node initiating the remote memory access, to the home node, requesting the data from, or exclusive access to, a data block. The home node's response is a `REPLY` packet which is sent back to the initiating node, providing the data and/or granting exclusive access, Figure 3.2(a).

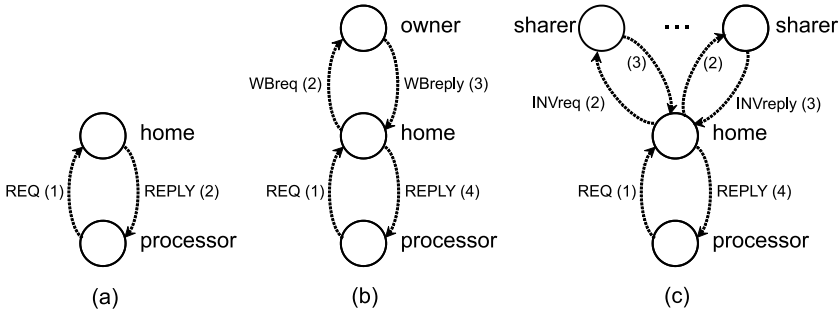

Figure 3.2: Possible sequences of packets in a directory-based coherence protocol. (a) no third-party nodes involved, (b) data is in state modified/exclusive at another node and must first be written back, (c) data is in shared state at other nodes and must be invalidated.

- When another node (further referred to as third-party node) has the data block in modified state, the home node sends it a *write back request* `WBreq`, demanding the owner node to give up exclusive access and write the modified data back to the home node in a `WBreply` message. This happens between the `REQ` and `REPLY` phases, Figure 3.2(b).
- When one or more third-party nodes have the data block in shared (read-only) state and the initiating node wants to write to it, which requires exclusive access, the home node sends an *invalidation request* `INVreq` to each of the sharing nodes asking them to remove the data from their cache. Each node acknowledges this with an `INVreply`, Figure 3.2(c).

Note that some nodes can perform a dual role, for instance when a processor performs a memory access to a data word that has its home on the same node. Or the home node can at the same time be the owner or a sharing node, if the processor located on the home node has the data word that is being requested in its cache. In these situations, some packets will not be visible on the interconnection network, although the same information is sent internally in the node (albeit in a more appropriate form for on-board or on-chip communication, such as over a processor local bus or a set of dedicated wires rather than in a packetized way).

In a real-world implementation, more situations than those depicted in Figure 3.2 can arise. When the home node is busy communicating with a third-party node (i.e., it has sent `WBreq` or `INVreq` packets and is waiting for the `WBreply`/`INVreply` messages), and a request from another node arrives for the same data word, processing of this new request must wait until the

first one is finished. This second request is not buffered at the home node, since this can cause very unfavorable worst-case memory usage if several requests are in this situation – remember that the protocol is executed by special hardware, so resources are limited. Instead, a `NAK` packet (negative acknowledgement) is sent back to the second requesting node, causing it to retry the operation at a later time. Another situation happens when a cache has a block in modified/exclusive state, but decides to evict it in response to a capacity or conflict miss. The cache will now, on its own initiative, send the data back to its home node (this situation is called a *cache-initiated write-back*). All these situations are modeled in our simulator, and influence the traffic on the communication network.

Reconfiguration opportunities The coherence protocol has a request-response nature. This means that once a request goes through the network, some time later a reply will come back. Often, this reply contains a data block and is thus much larger than the request. A complete memory transaction usually takes less than 1 µs, or a few thousands of clock cycles. Reply packets, which typically carry about 70% of the data size of all network traffic, can therefore be predicted at most 1 µs in advance. Similar as for packet-scale reconfiguration, telecom-grade reconfigurable components could be used to exploit this reconfiguration opportunity. In an interconnect setting, we will have to look at even longer time scales.

3.1.3 Communication bursts: milliseconds

McNutt [2000] notes that data references exhibit *fractal* or *self-similar* behavior, this means that similar behavior, in this case locality in space and time, can be observed at different hierarchical levels. Caches exploit this locality: they use a small, fast memory to store the value of recently requested data words (locality in time), and *prefetch* data that immediately succeeds them (locality in space). Subsequent accesses to these data words, which are more likely than random accesses due to the locality effect, can be satisfied from the cache instead of requiring a much slower access to main memory. More on the fractal behavior of communication in parallel applications can be found in [Greenfield and Moore, 2008].

Because of the self-similar behavior, the same locality is present in the stream of memory requests that miss in the level 2 cache. In practice, this is caused by the working set of the application being larger than the largest cache, or because the memory requests concern addresses written to by other processors. Therefore, network traffic streams should exhibit the same type of locality.

We have analyzed this behavior by measuring *communication bursts*. These are periods of elevated communication between node pairs. Their duration has been measured to be up to several milliseconds. Multiple bursts are usually active on the network at the same time, on a background of lower intensity traffic between most other node pairs. Measurements of burst lengths, the amount of traffic contained in them, and estimates on how much reconfiguration based on these bursts can help performance, are shown in detail in Section 3.2.

Reconfiguration opportunities Once it is know that a burst of communication is occurring between a node pair, the network can be reconfigured to provide a high-bandwidth, low-latency connection between these two nodes. If the burst is significantly longer than the network reconfiguration time, most of the traffic in the burst will benefit from the fast connection, and the performance of the system improves. Since bursts can be millions of clock cycles, or several milliseconds long, components such as tunable VCSELs and (ferro-magnetic) liquid crystals can provide the required reconfiguration speed.

The difficulty is in predicting when a burst will take place, between which node pair, and how long it will last for. A possible heuristic is to constantly measure communication and look for bursts. Once a burst is detected that has been present for a certain length of time, it can be assumed this burst will continue and the network is reconfigured. A simplification of this approach is used in this work: traffic is monitored during time intervals of fixed length, at the end of each interval the network is reconfigured to optimally match the traffic pattern that was just measured. When traffic changes slowly in consecutive intervals, which means that bursts should span several intervals, this method can give good results, as will be evident from our experiments later in this work.

3.1.4 Context switching: tens of milliseconds

Most parallel computers run several applications at the same time, with the operating system time-multiplexing threads among processors. For supercomputers, which mostly run just a single application at a time, this mechanism is usually not very visible. On servers and mainframes, however, it is all the more so: threads waiting for I/O-activity are often switched out, or a high number of threads serving different requests time-share a much smaller number of processors. Every time a processor switches to a different thread, this new thread needs to load its data set into the cache which causes a large burst of cache misses. Sometimes all of the thread's data is in the local mem-

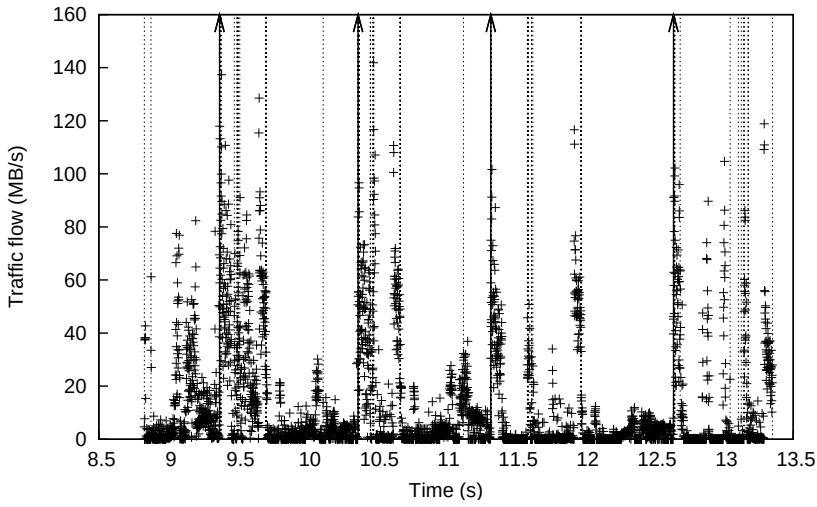

Figure 3.3: Traffic observed in and out of a single node through time, while running four 16-thread `cholesky` applications on a single 16-node machine. Solid arrows are shown when a context switch occurs on this node, dashed lines denote context switches on other nodes [Artundo et al., 2006b].

ory of the processor's node, but often remote memory accesses are required. In this case, the thread switch causes a communication burst. One such example is the case where a thread just woke up because its I/O-request was completed, the thread will now read or write new data on another node's memory or I/O-interface.

A study of these context switch induced bursts was done in [Artundo et al., 2006b]. One experiment time-shared multiple SPLASH-2 benchmarks on the same machine, another used the Apache web server loaded with the SURGE request generator [Barford and Crovella, 1998] to study a highly I/O-intensive workload. A clear correlation between context switches and bursts was found. This is illustrated in Figure 3.3, which shows the traffic generated by a single node through time and the points where context switches occurred. Here, four instances of the `cholesky` benchmark, with 16 threads each, were run on a single 16-node machine. Solid lines denote a context switch on this node, at this point a burst of outgoing memory requests is generated to fill the local cache with the new thread's working set. Dashed lines show context switches on other nodes. For some of these, the other node generates a burst of accesses to memory on this node, again resulting in a communication burst to the node shown. Other bursts are due to structure

in the application as described in Section 3.1.3. Burst duration was again several milliseconds.

Reconfiguration opportunities Traffic bursts caused by context switches typically involve intense communication, and can be several milliseconds long. The opportunities for reconfiguration are therefore similar to those for traffic bursts inherent to the application, as described before. One added advantage of context switches is that they are more predictable: the operating system's scheduler often knows in advance when a context switch will occur (at the end of the current thread's time quantum), at that moment a communication burst will most likely start at the node where the context switch occurs.[1] Also, if the new thread is known, the destination of the traffic burst can be predicted. The burst is mostly caused by the thread's working set being moved into the level 2 cache. Mostly this working set is the same as, or only slightly different from, the previous time the thread was running. The destination of the bursts will therefore be the same as the last time the same thread was scheduled. This information can be used by the reconfiguration controller, to reconfigure the network pro-actively, rather than reacting to measured network traffic. In this work, we will not further expand upon this reconfiguration method, more on this subject can be found in [Artundo et al., 2006b].

3.1.5 Applications: hours

The application running on the multiprocessor machine, executes a certain algorithm, which is split up among several processors. Each of the processors usually works on a subset of the data. One example is the simulation of oceanic currents, where each processor is responsible for part of the simulated ocean. Neighboring parts of the ocean influence each other because water flows from one part to the other. In the same way, information (current velocities and direction, water temperature) flows between the processors responsible for these parts. Clearly, if the processors themselves are neighbors on the communication network (i.e., connected directly), this makes for very efficient communication because a large fraction of network traffic does not need intermediate nodes. There is a similar communication pattern in several other physical simulations, where data is distributed by dividing space in 1-D, 2-D or 3-D grids and communication mainly happens between neighboring grid points. Other physical mechanisms, such as gravity, work

[1] Often, the operating system tries to avoid context switches at the same time on all nodes as this would initiate communication bursts on all nodes simultaneously, this can easily saturate the whole network.

over long distances. Cosmic simulations therefore require communication among all processors (although the traffic intensity is not uniform).

An important property here is how many communication partners each processor has. In some cases, the number of communication partners is higher than the network fan-out, or the topology, created by connecting all communication partners, cannot be mapped to the network topology using single-hop connections only. Then, some packets will then have to be forwarded by intermediate nodes, making communication less efficient. For instance, when communication is structured as a tree, which is the case for several sorting algorithms, it is not obvious how threads and data should be placed on a ring network. In a client-server architecture, where one thread is the server which answers questions from all other threads, the fan-out of the server thread is extremely high. The node that runs this thread will never have an equally high physical fan-out. In those cases, a large fraction of network traffic will require forwarding.

Moreover, for some applications each node's communication partners change through time. This happens for instance in algorithms where the work on each data set is not equal, and redistribution of work or data takes place to balance the workload of all processors. Another situation is in *scatter-gather* algorithms, in which data is distributed to or collected from a large number of nodes by a single thread – which will thus communicate in turn with different nodes. And sometimes the data set of one processor just does not fit in its local memory, and has to be distributed over several nodes. In this case, for part of the data, external memory accesses are required.

In [Heirman et al., 2008b], a different approach is used to characterize applications. A hierarchical partitioning of network nodes is made, based on the network bandwidth they consume, such that the bandwidth that leaves each cluster is minimal. This is similar to how electronic circuits can be partitioned, where the number of pins (or terminals) leaving the partition is minimized. According to Rent's rule [Landman and Russo, 1971], most realistic circuits are characterized by the existence of a power law relation between the number of pins external to, and the number of components inside the circuit partition. The exponent in this power law, the *Rent exponent*, denotes the amount of locality: a low exponent means that relatively few connections are needed between clusters at higher hierarchical levels, signifying that communication is more localized inside smaller clusters. A bandwidth version of this rule also exists in parallel applications, here the bandwidth external to a cluster of threads or processors grows as a function of the cluster's size, again with a power law relation [Greenfield et al., 2007]. By comparing partitionings based on traffic from different time intervals, the dynamics of network traffic can also be investigated.

Reconfiguration opportunities Regularity in the application is again visible on the network as communication bursts. Highly regular applications like the ocean simulation will have bursts, between the nodes simulating neighboring parts of the ocean, that span the entire length of the program. For other applications, communication is less regular, but even there, bursts of significant lengths (several milliseconds) can be detected. They are also visible in the measurements of Section 3.2, and can be exploited by the same techniques that exploit bursts caused by other mechanisms explored here.

Another method of exploiting regular communication patterns in the application is to have the program specify these patterns, and reconfigure the network accordingly. Since this can be done at a high abstraction level (source code or algorithmic level), by someone with a view of the complete program and algorithm (programmer or compiler), it can be expected that this method allows for very accurate prediction of the communication pattern, and would therefore result in the largest gains. It does, however, require a large effort to analyze the application in this way. Moreover, due to dependencies on the input data, it is not always possible to predict, at compile time, a fraction of total communication that is large enough to be of benefit.

This last approach was therefore not followed in this work, which instead focuses on reconfiguration that is completely transparent to the programmer and to the application. Other work explores this application-driven reconfiguration in more detail. For instance, the Interconnection Cached Network (ICN) combines many small, fast crossbars with a large, slow switching crossbar [Gupta and Schenfeld, 1994]. By choosing the right configuration of the large crossbar, a large class of communication patterns can be supported efficiently: meshed, tori, trees, etc., can be embedded in the architecture. The large crossbar thus acts (under control of the application) as a *cache* – hence the name ICN – for the most commonly used connections. This approach is also used, to some extent, in the Earth Simulator [Habata et al., 2004]. Its architecture centers around a 640×640 crossbar, on which communication between 640 processing nodes occurs through application-defined circuits. Inside each processing node, eight vector processors are connected through a smaller but higher data-rate crossbar. Barker et al. [2005] build on the ICN concept, and describe a dual network approach. Long-lived burst transfers use a fast Optical Circuit Switching (OCS) network, which is reconfigured using MEMS mirrors (with a switching time of a few milliseconds) under control of the application. The other, irregular traffic – which is usually of a much lower volume – uses a secondary network, the Electronic Packet Switching (EPS) network, which is implemented as a classic electrical network with lower bandwidth but higher fan-outs, to obtain low routing latencies on uniform traffic patterns.

Failure category		Fraction
Hardware		60 %
of which	CPU	*25* %
	Interconnect / network	*13* %
	Memory	*12* %
Software		23 %
Environment	(power failures, etc.)	1.5%
Human error		0.6%
Undetermined		11 %

Table 3.1: Breakdown of LANL failure data into categories

3.1.6 Hardware failures: days

A recent study performed on failure data of the supercomputers and clusters in use by the Los Alamos National Laboratory (LANL) estimates the Mean Time Between Failures (MTBF) for a next-generation peta-scale HPC system [Schroeder and Gibson, 2007]. This is a machine with an aggregate computational power of one petaflop, or 10^{15} floating point operations per second.[2] Extrapolating from current HPC system performance, scale, and failure data, this study estimates the actual time spent for useful computation between full system recovery and the next failure. On a petaflop machine with over 10,000 processors, this time is suggested to fall to just a mere 1.25 hours.[3] The state-of-the-art fault tolerance strategy for such a system, checkpoint/restart, saves checkpoints of the machine state at regular intervals so that the program can be restarted from an intermediate state after a machine failure. Periodically writing a dump of all system memory, which can be up to 1 TiB, takes time though, as does loading the checkpoint after each failure. Because of this overhead, the study estimates that over 60% of the cycles (and investment) on next-generation peta-scale HPC systems may be lost due to dealing with reliability issues.

Reconfiguration opportunities Reconfigurable networks are often touted as the panacea for routing around failing hardware. Everything depends

[2] The first such machine is LANL's own Roadrunner, built and installed by IBM just recently in May 2008. It consists of 6,480 dual-core AMD Opteron processors and 12,960 Cell processor chips, for a total of 129,600 computing cores, interconnected using optical Infiniband links.

[3] By comparison, the first electronic computer was the ENIAC, installed in 1946. It broke down every two days on average, when one of its 18,000 vacuum tubes failed.

of course on whether reconfiguration can solve, or at least hide, the failure. When for instance a DRAM chip, part of main memory, fails, reconfiguring the network will not bring its data contents back. Schroeder and Gibson [2007] note that the failure rate of large HPC installations scales approximately with the number of sockets (processor chips) in the system, while being largely independent of the number of processor cores per chip. Further analysis of the LANL [2005] data set (see the breakdown by category in Table 3.1) reveals that at least 25% of the 23,741 recorded machine interruptions in the set were processor related. This does suggest that the electrical connections between the processor and its surroundings are at least in some part to blame. In this case, an optical access, directly to the processor chip and including reconfiguration abilities, can be used to route around failing parts of the interconnect. Network failures (which account for 13% in the LANL data) can also effectively be hidden by a reconfigurable network.

The idea of using reconfigurable networks to cope with failing hardware will not be further explored in the context of this work. Other work that does investigate this idea includes [Shively et al., 1989], which describes AT&T's DSP-3 parallel processor. This machine includes an interconnection network, connecting 128 processing nodes, with redundant paths to enable topology reconfiguration in support of fault tolerance. US Patent 6,871,294 [Phelps et al., 2005] covers the dynamically reconfigurable interconnect of the Sun Enterprise 10000, a commercial server with up to 64 processors. It too provides redundancy and can continue running (in a degraded mode) when part of the machine's centerplane fails. Intel holds a similar patent (US Patent 7,328,368), which describes the dynamic width reduction of interconnection links in response to transmission errors [Mannava et al., 2008]. This technology will be part of Intel's upcoming QuickPath system interconnect [Kanter, 2007].

Using optical reconfigurable technology, similar solutions would be possible, with the added advantage that optics allows for data-transparent reconfiguration without introducing additional latency.

3.2 Communication bursts

Early on in this PhD work, measurements were made of the communication bursts occurring on multiprocessor networks. This was done using a simulation setup and benchmark programs similar to those that will be described in Chapter 4. These measurements allowed us to characterize the opportunities for reconfiguration and quantify the gain that could be expected from an

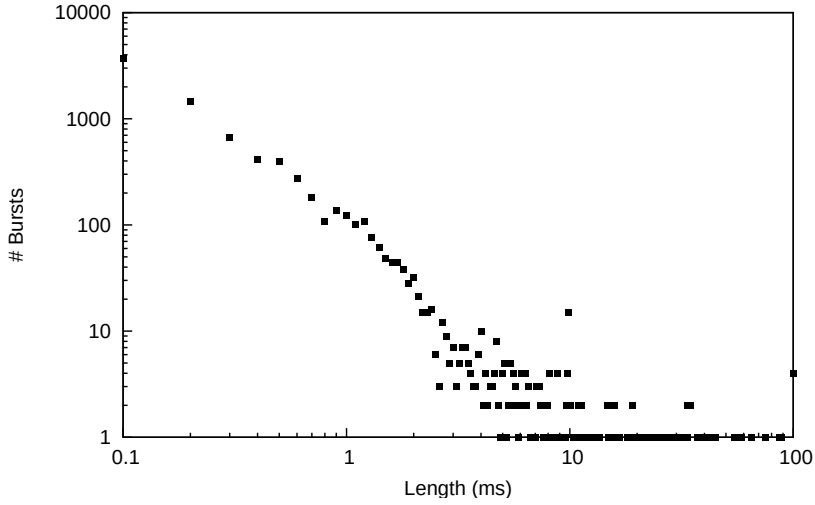

Figure 3.4: Burst length distribution for the `fft` application, for a 16-node network

actual implementation. This way, the reconfigurable network architecture could be designed later, supported by the results obtained previously.

We simulated the execution of a set of benchmark programs from the SPLASH-2 suite [Woo et al., 1995]. A trace was collected of all network traffic, which was subsequently analyzed. In this trace we looked at the duration of traffic bursts between node pairs. When we know heavy communication is taking place for a period of time that is significantly longer than the switching time of the reconfigurable component that would be used in a network implementation, it becomes viable to reconfigure the network to accommodate this burst. By looking at the number of bursts, and the amount of traffic in them for given burst lengths, an estimate can be made of how a network with a given reconfiguration time will perform, or conversely, what reconfiguration time and therefore which components are needed to implement a network with the required performance.

3.2.1 Traffic burst length distribution

We start by dividing time in intervals of length Δt. For the interval starting at time t, the traffic that flows between nodes i and j is accumulated into $T_{i,j}(t)$. Only traffic that has the processor at node i as source and the processor at node j as destination or vice versa is considered, not traffic these

nodes are forwarding en route to somewhere else. This makes the analysis independent of the network topology. Furthermore we consider all links to be bidirectional and symmetric, so we sum traffic in both directions and always consider i to be smaller than j.

At the end of each interval, all node pairs are sorted according to the traffic that has flown between them. The top n of those pairs are marked for this time interval, where n is the fraction of node pairs that is expected to benefit directly from reconfiguration. We'll assume that this number of node pairs n will scale with the number of processors, so for a 16-processor network, with p denoting the number of processors, we choose $n = p = 16$. Next, we count the number of consecutive intervals a certain node pair was in this top n. Multiplied by Δt this gives us the length of a burst of communication that took place between that node pair. This burst will be represented by the 4-tuple (i, j, t, l), indicating the nodes that were involved (i and j), the starting time t and the length l. The distribution of the burst lengths (over all node pairs at any starting time) for one of the benchmark applications, the `fft` kernel, is shown in Figure 3.4. The interval length Δt is 100 μs, for 16 processors and $n = 16$. This distribution closely resembles a power law, augmented with some very long bursts that span a length of up to the duration of the entire program. This should not come as a surprise, since fractal or self-similar behavior of memory access patterns – resulting in power-law relations – has been shown before [McNutt, 2000]. The long bursts are due to references to global data, configuration parameters, operating system structures and synchronization variables that are needed throughout the program.

Such fractal behavior is actually very common, not only in memory access patterns, but in all kinds of system properties related to communication. Indeed, even memory accesses can be viewed as communication between points in time [Greenfield and Moore, 2008]. Connections in digital electronic circuits, which are the result of a spatial mapping of a given algorithm – whereas a sequential program would represent a temporal mapping – also exhibit fractal behavior: the number of connections needed depends – via a power law relationship – on the size of the circuit, this is commonly known as Rent's Rule. This relationship was first described empirically by Landman and Russo [1971] and later theoretically derived by Christie and Stroobandt [2000]. Wire length distributions on VLSI chips, the dual of communication distances on interconnection networks – both resulting from a spatial mapping of fractally communicating partitions of the circuit or software algorithm – again follow a power law behavior, which is extensively described by Stroobandt [2001].

3.2.2 Traffic size fraction

Now that we know there is temporal locality in the traffic, we would like to quantitatively describe how much traffic is involved in these bursts. This is important, since only traffic that is contained inside a burst of sufficient length will be able to benefit from a reconfiguration of the network. Only if this traffic represents a sizable percentage of all traffic, gain can be expected from adding reconfiguration abilities.

We determine how much traffic is contained in bursts of a given minimal length. First, we sum the total amount of traffic in each of the bursts (i, j, t, l):

$$T(i, j, t_0, l) = \sum_{t=t_0}^{t_0+l} T_{i,j}(t)$$

This gives the number of bytes that participate in each of the bursts that were detected. Next we calculate the total traffic size per burst length:

$$T(l) = \sum_{i,j,t} T(i, j, t, l)$$

With T_t the total amount of traffic that was sent across the network during the benchmark execution, the distribution of traffic by burst length will be given by $T(l)/T_t$. Since we are interested in bursts that have a length of *at least* l cycles, we compute the complementary cumulative distribution:

$$T_C(l_0) = \sum_{l=l_0}^{\infty} T(l)$$

We now know that a fraction $T_C(l)/T_t$ of all traffic is part of a burst of length l or longer. This distribution is shown in Figure 3.5 for the `fft` application. The graph shows us that around 60% of all traffic is contained in bursts of 1 ms or longer, bursts of at least 10 ms long still account for 30% of all traffic.

3.2.3 Application speedup

Next, we want to characterize the reduction in runtime, or the application speedup, we can expect from a reconfigurable network. Without a full simulation this is difficult to predict, since overlapping latencies, the appearance or disappearance of congestion and the influence of latency on the control flow of the program can make the relationship between network latency and program duration somewhat irregular. However, running this full simula-

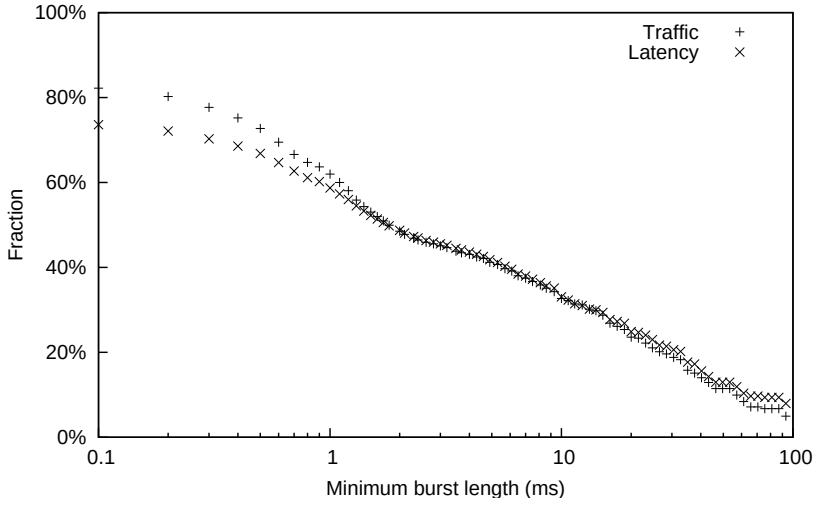

Figure 3.5: Traffic size and latency fractions per minimum burst length for `fft`, for a 16-node network

tion would require us to make several design decisions, such as the topology of the reconfigurable network and the algorithms that determine when and how reconfiguration takes place. To make an estimate that is independent of all these implementation details, we have tried to find a reasonable approximation of the expected speedup based only on results from a single simulation with a non-reconfigurable network. The metric we use for this is the fraction of memory access latency that is the result of traffic contained inside bursts.

For each memory access that requires network traffic, we store the starting time of the access and the requesting and responding nodes. Next we see whether, at that time, a communication burst was present between these two nodes. If this is the case, the memory access is said to be related to this burst.

Now we repeat the computation from the previous section, but instead of summing the number of bytes in the packets that make up the burst, we sum the latency of all memory accesses related to the burst. This gives us a total time $L(i, j, t, l)$ for each burst. Next we again sum all bursts of the same length yielding $L(l)$. With L_t the total memory access latency for all memory accesses made during the execution of the program, $L(l)/L_t$ will give the distribution of access latency related to bursts of a given length. Again we compute the complementary cumulative distribution $L_C(l)/L_t$, which is also plotted in Figure 3.5.

Code	Speedup
barnes	15%
cholesky	19%
fft	36%
fmm	30%
lu	72%
ocean	33%
radiosity	86%
radix	16%
raytrace	12%
volrend	8%
water.nsq	2%
water.sp	6%
Average	26%

Table 3.2: Applications and estimated speedups, all on 16 nodes

Now that we found the fraction of memory access latency that can be influenced by a reconfigurable network, we will estimate the application speedup. First we assume that access latency is evenly distributed over all processors. The total access latency corresponding to bursts longer than l, this is $L_C(l)$, is also considered to be distributed evenly. Therefore, with p the number of processors and t the runtime of the program, reconfiguration will be able to influence a fraction of the total program runtime equal to:

$$f = \frac{L_C(l)}{p \cdot t}$$

We conservatively estimate that memory accesses benefiting from a reconfigured network have their latency reduced by a factor of four.[4] Now we can use Amdahl's law (Equation 1.1) to compute the total speedup S if a fraction f of the program is accelerated by a factor of four:

$$S = \frac{1}{(1 - f) + f/4}$$

This calculation has been done for all SPLASH-2 programs, using a minimum burst length of 1 ms. The resulting speedups are given in Table 3.2. The esti-

[4]We estimate this factor as follows: the average inter-node distance in a 4×4 torus network is 2.13, adding an *elink* (see Chapter 4) reduces the distance between the corresponding node pair to just one. The added bandwidth lowers congestion, providing an extra latency reduction.

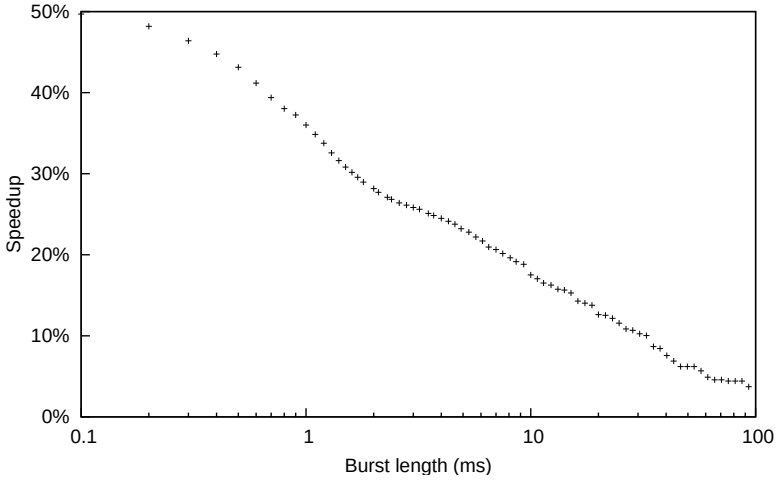

Figure 3.6: Maximum achievable speedup for `fft` depending on minimal burst length, for a 16-node network

mated achievable speedup ranges from 2% (`water.nsq`) to 86% (`radiosity`), depending on the application, with a geometric average of 26%. The main cause for the variation in these numbers is the fraction of total execution time that is spent waiting for memory accesses. Programs like `water` and `volrend` have a good computation to communication ratio and therefore cannot be accelerated much by improving the network. There is however sufficient network traffic locality in these programs, as is evident from their burst length distributions (not shown), so under harder network conditions they also can be accelerated significantly using a reconfigurable network.

For slower technologies, Figure 3.6 shows the achievable speedup for the `fft` kernel with minimal burst lengths other than 1 ms. Shorter reconfiguration times could allow the performance gain to be up to 50%. Note however that the network reconfiguration time needs to be significantly shorter than the burst length. For instance, if detecting the burst takes 100 µs and reconfiguration takes the next 100 µs, we would be able to affect only 80% of the traffic in a 1 ms burst, reducing the actual speedup. Mistakes in burst prediction will cause some of the network resources to provide little gain, again resulting in an actual speedup that is less than the projected upper bound calculated above.

On the other hand, the redistribution of traffic over a reconfigured topology often will not only speed up the traffic inside communication bursts. When a large traffic stream, such as a burst, is moved elsewhere, the traffic

remaining on this part of the network now encounters less congestion and will therefore also experience lower latency. This effect has not been taken into account in the previous discussion, and might cause the actual speedup to be higher than the reported estimates.

3.3 Thread and data migration

An existing software technique that can minimize network traffic, thread and data migration, is described in Section 2.4. It requires no or very limited hardware cooperation, and is therefore relatively cheap to implement. Its simplicity of course results in some drawbacks that are not present with reconfigurable networks. The moving of tasks and data occurs at a time scale of seconds to minutes, which is several orders of magnitude slower than the reconfigurable networks we study. Also, the algorithm performed by the application may exhibit a communication pattern that cannot be mapped efficiently, i.e., using single-hop connections only, on for instance a ring or mesh network. A reconfigurable network, on the other hand, supports a much richer topology than the basic ring or mesh.

Moreover, the act of moving a thread or a data page in itself causes a burst of communication. Each time a thread is scheduled on a new processor, a burst of cache misses loads the thread's working set into the local cache, from the cache of the node the thread ran on previously, or from main memory. This results in a communication burst. In the case of data migration, the copy operation, which implements the data movement, again results in a burst of remote memory accesses by the processor executing the move. In both cases, a network reconfiguration, adapting the network to optimally support this communication burst, would be very useful! Therefore, in our view, both techniques are complementary: software thread and data migration can cheaply handle slow changes in the communication pattern, while the reconfigurable network copes with faster changes, supports the migration of threads and data efficiently, and provides a richer network topology when a good placement of threads and data on the basic topology is not possible.

3.4 A case for reconfigurable networks

We can summarize the results of this chapter as follows. Since we focus on a short-range interconnection network, rather than a long-range telecommu-

nications network, the components available for reconfiguration have, by necessity, a relatively long reconfiguration time. This means optical packet switching, or reconfiguration at the time scale of single packets or even remote memory requests, is not feasible. Yet, at longer time scales, there is also locality, as is evident from our measurements in Section 3.2. It is this locality we will exploit in our reconfigurable network architecture. Therefore, the main design requirement for this architecture is:

> *To efficiently support communication bursts, in the knowledge that multiple bursts are active at the same time and that bursts have a lifespan of up to several milliseconds.*

Since the reconfiguration time will be in the order of hundreds of microseconds up to a few milliseconds, the algorithm that controls reconfiguration – by measuring traffic and adapting the topology – must have a similar runtime. This means there is not much time available to find the optimal topology. Instead, fast heuristics will have to be used that can quickly find a near-optimal solution. Predicting network traffic bursts – necessary because the network needs to be in the right topology once (the bulk of) the burst arrives – should also be done inside this time frame. Input from the application on this is not desired: just like the shared-memory paradigm insulates the application and its programmer from the implementation details of memory (which node data is stored on, cache coherence), we want network reconfiguration to be transparent to the application. This is even impossible to do otherwise: since communication is implicit, the programmer doesn't know about network traffic and can therefore hardly be expected to give hints about reconfiguration! How our reconfigurable network architecture was designed, based on these requirements, can be found in the next chapter.

4

A reconfigurable network architecture

On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.
— Charles Babbage

This chapter outlines the reconfigurable network architecture we propose for use in a distributed shared-memory multiprocessor machine. The network architecture itself, consisting of a base network with fixed topology, augmented with *extra links* supporting communication bursts, is presented in Section 4.1. In Section 4.2 we propose a possible implementation of this architecture, based on Wavelength Division Multiplexing (WDM) and Selective Optical Broadcast (SOB) technologies, using a broadcast component designed at the Vrije Universiteit Brussel. The algorithm controlling reconfiguration, which determines the new topology based on the expected traffic pattern, is given in Section 4.3. Also, we provide details on the simulation methodology that was used throughout this thesis to evaluate the performance of our proposed reconfigurable network implementations. Section 4.4 gives an overview of our simulation platform, while Section 4.5 describes the SPLASH-2 benchmark suite that was used as the workload on our simulated multiprocessor machine.

4.1 Proposed reconfigurable network architecture

If you think good architecture is expensive,
try bad architecture.
— Brian Foote and Joseph Yodertein

Previous studies concerning reconfigurable networks, for instance [Pinkston and Goodman, 1994] or [Sánchez et al., 1998], have mainly dealt with fixed topologies (usually a mesh or a hypercube) that allowed swapping of node pairs, incrementally evolving the network to a state in which processors that often communicate are in neighboring positions. However, this incremental determination of optimum processor placement turned out to converge slowly, or not at all when the characteristics of the network traffic change rapidly. Determining a global placement in a single step is also not feasible, because computing this placement would take too long (it is an NP hard problem, and will not scale well when increasing the network size). Also, a reconfiguration method that is this invasive to the network topology, requires all network traffic to be halted during a node rearrangement. This negatively impacts performance, and has a large influence on worst-case response times which can make this type of network unsuitable for real-time applications. Moreover, implementing such a network usually needs to be done with large free-space optical structures which are not compatible with the integrated, highly reliable nature of large parallel computers. Finally, the communication pattern exhibited by the program running on the parallel computer may have a structure that cannot be mapped efficiently (i.e., using only single-hop connections) on the chosen base network topology, suggesting that reconfiguration should be used to offer a richer topology than that of the base network.

Therefore, we assume a different network architecture. We start with a packet-switched *base network* with a fixed, regular (such as mesh or torus) topology. In addition, we provide a second network that can realize a limited number of connections between arbitrary node pairs – these will be referred to as *extra links* or *elinks* – which will be placed between those node pairs involved in a communication burst. A schematic overview is given in Figure 4.1. An advantage of this setup, compared to other topologies that allow for more general reconfiguration, is that the base network is

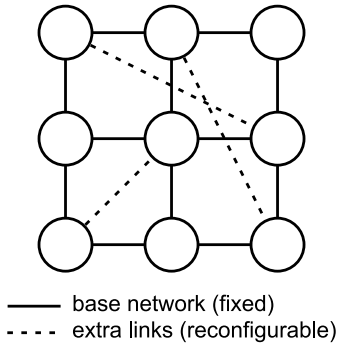

Figure 4.1: Conceptual reconfigurable network topology. The network consists of a base network, augmented with a limited number of direct, reconfigurable links.

always available. This is most important when the elinks are undergoing reconfiguration and may not be usable. Periods in which no communication is possible, are therefore avoided, and worst-case communication latency is now much more predictable which is very important in real-time settings. Routing and reconfiguration decisions are also simplified: it is not possible to completely disconnect a node from the others – a connection with reasonable bandwidth will always be available through the base network – instead, the algorithm can completely concentrate on the speedup of communication bursts.

Performance increase now results from a number of effects. First, bandwidth is added to the network, increasing its overall capacity. This could of course be obtained without complicated reconfiguration so we should subtract this component of the performance increase, this will be looked into in the chapter on performance evaluation. Secondly, by providing single-hop connections for a large fraction of traffic, this traffic does not suffer contention at a large number of intermediate hops where it will potentially need to be buffered. Also, the conversion from the optical to the electronic domain and back at each intermediate node is avoided – saving time and power, since most of the routing can now be done in the optical domain, rather than using packet switching at each hop. Lastly, by moving large volumes of traffic away from the base network, congestion is lowered, resulting in a reduction of waiting times for all network packets.

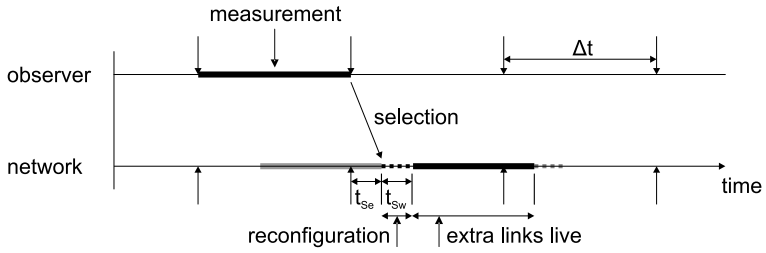

Figure 4.2: Sequence of events controlling reconfiguration. The observer measures network traffic, and after each interval of length Δt makes a decision where to place the elinks. This calculation takes an amount of time called the *selection time* (t_{Se}). During the *switching time* (t_{Sw}), reconfiguration will take place making the elinks temporarily unusable.

Sequence of events

Reconfiguration takes place at specific intervals, the length of each interval being a (fixed) parameter of the network architecture. Traffic is observed by a reconfiguration entity during the course of an interval, and total traffic between each node pair is computed. At the end of the interval, the new positions of the elinks are determined, within the constraints of the network architecture, such that node pairs that exchanged the most data in the previous interval will be 'closer together': the distance, defined as the number of hops a packet sent between the pair must traverse in the new network topology, is minimized. This way, a large percentage of the traffic has a short path and a correspondingly low uncongested latency. Also, congestion is lowered because heavy traffic is no longer spread out over a large number of links. After selecting the new network configuration, the network is reconfigured and a new interval begins. This sequence of events is depicted in Figure 4.2.

In most of our simulations, we assumed for simplicity that both the computation to select the new elinks (the *selection time)* and the physical reconfiguration (the *switching time*, done by switching mirrors, tuning lasers, etc.) are performed instantly. This is not the case in reality: depending on the technology, the switching time can take from tens of microseconds up to several milliseconds. The selection time can also be significant, depending on the complexity of the elink selection algorithm. Therefore, when interpreting simulation results in this work in which a sweep over several reconfiguration intervals is made, one should keep in mind that this reconfiguration interval is not an entirely free parameter: it must be chosen large enough so that the

selection and switching times are negligible. In Section 6.2.1, however, we did include realistic selection and switching times in the simulation, and set both to 10% of the reconfiguration interval.

Parametric network model

To avoid pinning our discussion down on the peculiarities of a specific network architecture, we construct a hypothetical parametrized architecture that can be adapted to represent several possible network implementations.

Our parametric network model provides the infrastructure to potentially place an elink between any two given nodes. Two constraints are made on the set of elinks that are active at the same time:

- a maximum of n elinks can be active concurrently,
- the fan-out of each node (not including connections to the base network) is limited to f.

The time between reconfigurations, called the reconfiguration interval Δt, is the third parameter. The results in this work will be based on different sets of values for these three parameters.

This simple network model provides us with a unified, parametrized architecture that allows us to simulate several types of implementations, and analyze the sensitivity of network performance to these basic parameters. A realistic implementation, like the one described in the next section, can impose additional limitations. For instance, the SOB device only allows a subset of all node pairs to be directly connected with an elink. This will be modeled by placing additional constraints on the set of concurrently active elinks.

Physical implications of the parameters

Each of the parameters n, f and Δt will have a physical origin once an implementation is being made. Consider for instance an implementation in which each node has a number of tunable lasers, the signal of all of them is broadcast to all receiving nodes. Nodes also have a number of wavelength-selective receivers, each sensitive to a unique wavelength. By tuning the wavelength on each laser, a destination node can thus be selected.

The fan-out f determines how many elinks can originate at one node. For every elink, the node needs one laser, which has its wavelength tuned depending on the destination of the elink. f will in this case equal the number of tunable lasers per node. The same happens for the fan-in, the number of

elinks terminating at one node, where each incoming elink needs its own wavelength-selective photodetector. Throughout this work, we assume fan-in and fan-out are both limited to the same number.

Assuming all f lasers and photodetectors on a node can be used at the same time, the number of elinks n will equal $f \cdot p$ (with p the number of processors or nodes). Since light from all lasers is combined in the broadcast stage, they should all use a different frequency. The number of possible frequencies[1] can therefore limit n.

Finally, because the elinks are unusable while the lasers are being retuned, for optimal performance this period needs to be a small fraction of the total time. The reconfiguration interval Δt should thus be chosen significantly larger than the tuning time.

One might wonder why, if connections exist between all node pairs in the system, only a few of them – the activated elinks – are used at the same time. Why not simply use all available connections at all times, and have a fully connected network instead? In this work, we assume this is not possible. A practical reason for this is the fact that nodes do not support the bandwidth to directly connect to such a high number of other nodes. Indeed, dividing the scarce bandwidth into for instance six high-bandwidth channels to other nodes (four neighbors on a mesh network, and two dynamic neighbors using elinks), would be a much more economical use of resources than connecting to all nodes with much slower channels, most of which would be unused for a large fraction of time anyway.

Another reason is arbitration: if multiple nodes were allowed to transmit on the same frequency, this can result in collisions in the optical domain when multiple nodes target the same receiver. Some kind of arbitration among the source nodes would then be needed, which increases complexity and network latency. We want to avoid this, and view the elinks as point-to-point connections. This greatly simplifies router design. All arbitration is instead done at reconfiguration time by assigning only a single source node per frequency.

Comparison with circuit switching

Circuit switching is an alternative to packet switching. Once a circuit has been set up between two nodes, a dedicated link exists. Along its path, no arbitration is needed, buffering delays are therefore avoided. Often bandwidth and latency guarantees are also provided. The circuit's path will usually go through the same intermediate nodes the packet would have gone through

[1] This is a technological parameter of the tunable lasers, usually a trade-off needs to be made between the number of frequencies and the tuning time.

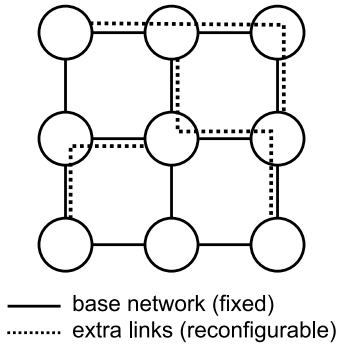

Figure 4.3: A possible circuit switched equivalent of the topology using elinks from Figure 4.1

if packet switching was used, the only difference is that when using packet switching the packet has to contend for its next link at every intermediate node.

We can view the creation of circuits as being equivalent to placing elinks: both provide dedicated, contention-free links between specific node pairs. Figure 4.3 shows this: three circuits, implemented by having pre-set paths through intermediate nodes, logically provide the same topology as the elinks in Figure 4.1. It is entirely possible to implement the elink concept in exactly this way: a mesh network where a percentage of bandwidth is reserved by circuits for long-living, high volume traffic (the elinks) and the remaining part of the mesh is used in a best-effort packet switching way for other traffic (the base network), is a valid way of implementing our architecture. The circuit switching analogy is somewhat limited, however, in the sense that it seems to *require* that the intermediate nodes play some role in forwarding packets. This is not necessary, and is not the case in for instance the implementation proposed in the next section. We will therefore use the elink concept throughout this work, regardless of whether its implementation looks like a circuit switched network or not.

Reconfiguration as a spatial cache

Our architecture consists of a base network that – with limited capacity – connects all nodes, and on top of that provides faster, direct connections between node pairs that communicate frequently. In this way it exploits locality in communication, in much the same way as a small, fast cache memory exploits locality in memory references by reducing the need to ac-

cess the slower main memory. In this sense, the Interconnection Cached Network (ICN), described on page 51 could be more aptly named the Interconnection *Scratchpad* Memory, since there the direct links are in control of the application – just as a scratchpad memory is managed by software. In contrast, our architecture provides an automatic way of measuring, adapting for and thus exploiting communication locality in an software-transparent way, and in this way acts much more similarly to a cache.

4.2 Hardware implementation

Computer science isn't about computers
any more than astronomy is about telescopes.
— Edsger W. Dijkstra

Using Wavelength Division Multiplexing (WDM) technology and a Selective Optical Broadcast (SOB) component designed and fabricated at the Vrije Universiteit Brussel (VUB), we propose a hardware implementation of a reconfigurable optical network. This architecture was developed in cooperation with ir. Iñigo Artundo from VUB. Here we will describe the basic architecture, some design decisions and their implications on the reconfigurable topology. More details can be found in [Artundo et al., 2006a] and [Artundo et al., 2008b].

The implementation we envisage consists of low-cost tunable laser sources, a broadcast-and-select scheme for providing the extra optical links, and wavelength-selective receivers on every node (Figure 4.4). For the transmission side, VCSELs are preferred for their low power consumption, easy array integration and coupling into optical fibers. Their tuning range (a few tens of channels) and speed (between 100 μs and 10 ms) is adequate for following the traffic patterns targeted in this work.

The broadcasting can be done through the use of a star coupler-like element that reaches all nodes. By tuning the laser source, the right destination is addressed. When scaling up to tens of nodes or more this is no longer feasible: the number of available wavelengths is finite, also, such a wide broadcast would waste too much of the transmitted power. In this case, a SOB component like the diffractive optical prism described in Section 2.3.4 can be used, which broadcasts light from each node to its own subset of receiving nodes. On the receiving side, Resonant Cavity Photodetectors (RCPDs)

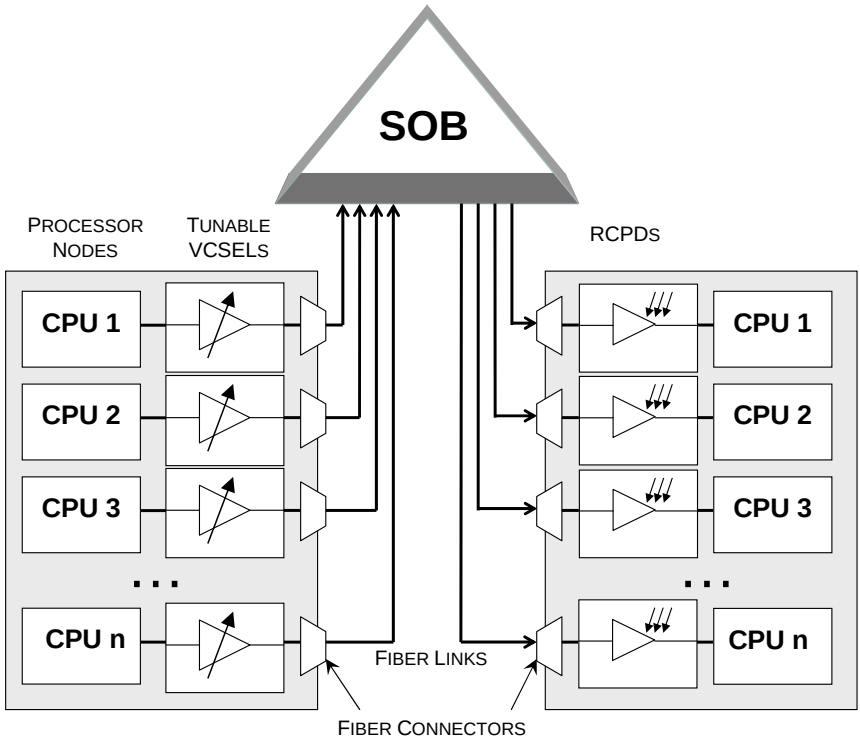

Figure 4.4: Schematic overview of a reconfigurable interconnection architecture using tunable lasers and selective optical broadcasting

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow[\text{BROADCAST}]{\text{SELECTIVE}} \begin{bmatrix} 7 & 8 & 9 & 5 \\ 11 & 12 & 13 & 10 \\ 15 & 16 & 14 & 3 \\ 1 & 2 & 4 & 6 \end{bmatrix}$$

Figure 4.5: Optimized node placement on the SOB element for 1-to-9 broadcast in a 16-node network

make each node susceptible to just one wavelength (the prism itself does not distinguish signals by wavelength – each incoming beam is projected onto nine spots, the difference in spot position throughout the wavelength range used is negligible). Note that the routing to and from the SOB component will be such that nodes have different neighbors on the broadcast element than those on the base network. This way the elinks will span a distance on the base network that is larger than one. The optimum placement on the SOB component was determined by using simulated annealing to maximize the possible distance reduction, and is shown in Figure 4.5 for a 16-node network. For instance, node 6 can set up an elink with nodes in the subset $[7, 8, 9, 11, 12, 13, 15, 16, 14]$. Due to its corner position, node 13 can reach only the four nodes in the set $[15, 16, 1, 2]$.

This topology can be modeled with our parametrized architecture by using $n = 16$, $f = 1$, and an additional set of constraints on which destinations (only 9 out of 16) can be reached from each source node. Note that this architecture is also not symmetric (for instance, node 6 can connect to node 7, but not vice versa), so for this implementation we use unidirectional elinks. Most of the simulations presented further in this work will include results for this implementation. Specifically, Section 6.2.1 will compare the performance of a network allowing full broadcast with the selective broadcast scheme presented here.

4.3 Extra link selection

For every reconfiguration interval, a decision has to be made on which elinks to activate, within the constraints imposed by the architecture, and based on the expected traffic during that interval. In our current implementation, it is expected that traffic in the next interval will be the same as the traffic pattern measured during the previous interval. As explained in Section 4.1, we want to minimize the number of hops for most of the traffic. We do this by

minimizing a cost function that expresses the total number of network hops traversed by all bytes being transferred. This cost function can be written as

$$C = \sum_{i<j} d(i,j) \cdot T(i,j)$$

with $d(i,j)$ the distance (in number of hops) between nodes i and j, which is a function of the elinks that are selected to be active, and $T(i,j)$ the number of bytes exchanged between the node pair in the time interval of interest. If bidirectional elinks are used, $T(i,j)$ is the sum of traffic in both directions.

The time available to perform the elink selection is of the same order of magnitude as the switching time, because both need to be significantly shorter than the reconfiguration interval. Since the switching time will typically be at most a few milliseconds, we need a fast heuristic that can quickly find a set of active elinks that satisfy the constraints imposed by the architecture and has an associated cost close to the global optimum.

We have constructed a greedy algorithm that works as follows (Figure 4.6 shows the algorithm in pseudo code):

1. Construct a list of all node pairs (i,j), sorted by $d(i,j) \cdot T(i,j)$ in descending order, with $d(i,j)$ the distance between nodes i and j when using only the base network connections.

2. Initialize the set of active elinks E_a to be empty, and the set of possibly active elinks E_p to contain all elinks that can be supported by the architecture (but not necessarily at the same time). For our parametric architecture, E_p contains all $p(p-1)$ (unidirectional elinks) or $p(p-1)/2$ (bidirectional elinks) node pairs (with p the number of processors or nodes). In the implementation with the SOB component from Section 4.2, E_p contains a subset hereof (just $9 \cdot p$ entries).

3. For the node pair at the top of the list, determine which *new* elink (one that is not already in E_a but is still in E_p) is the *most interesting*, i.e., when enabled, would give the greatest reduction in distance between these two nodes. This elink is removed from E_p and added to E_a, also, the current node pair is removed from the top of the list. If bidirectional elinks are used, the reverse link is also enabled (i.e., moved from E_p to E_a).
   If an elink resulting in a direct connection between the node pair is still available in E_p this one will of course be selected, since it reduces the distance to one. If none of the elinks in E_p can provide a distance lower than the one over the base network or over an elink already in E_a, no new elink is activated.

```
nodepairs = [ all (src, dst) pairs with src < dst ]
nodepairs.sort_descending(
  sortby = distance_using_basenetwork(src, dst)
           * traffic(src, dst)
)

active = []
possible = [ all elinks supported by the architecture ]

for each (src, dst) in nodepairs:
  elink = most_interesting_elink(src, dst)

  if distance_using_elinks(src, dst, active + elink)
      >= distance_using_elinks(src, dst, active):
    # no additional gain by turning on elink
    continue

  # turn on elink!
  possible.remove(elink)
  active.add(elink)

  # make sure we obey all implementation constraints,
  # such as maximum fan-out
  for elink in possible:
    if conflicts(active, elink):
      possible.remove(elink)

  if possible.empty(): exit for

activate_elinks(active)
```

Figure 4.6: Pseudo code for the elink selection algorithm

To quickly do this selection, a table that gives the distance between each node pair as a function of the activated elinks is pre-computed. If we assume at most one elink is used in each path, this table has a maximum of {number of node-pairs} × {number of elinks} entries, and usually much less since only a small number of elinks can decrease the base distance for a specific node pair.

4. Once a link has been added to E_a, check the constraints imposed by the network architecture. If activation of one of the links in E_p would cause a node to exceed its fan-out limit f, this elink is removed from E_p and is therefore no longer considered for activation in the following iterations of the algorithm. When the maximum number of elinks n is reached, all elinks should be removed from E_p.

5. As long as there are node pairs on the list, and the set of possible elinks E_p is not empty, continue with step 3. Else, end the algorithm. E_a is now the set of elinks that will be enabled during the next time interval.

Note that, after deciding to enable one of the elinks in step 3, one could recompute the distances between node pairs, taking this new elink into account. The list of remaining node pairs would then be updated based on the new distance distribution, before starting a new iteration. This is considered too time-consuming, however, and has not been implemented in our algorithm.

Figure 4.7 illustrates the elink placement through consecutive intervals of length Δt. These pictures were created from a simulation of the `radix` benchmark on a 4×4 mesh, with the reconfiguration abilities characterized by $n = 4$, $f = 2$ and $\Delta t = 100$ µs. In the top graphs, traffic intensities are shown on each link. Additionally, the four highest communicating node pairs are connected with a black line. These are the node pairs that will most likely get an elink in the next interval. For instance, nodes 1 and 14 communicate heavily in the first interval shown, from the next interval onwards an elink provides a direct connection between them. Nodes 4 and 15 already have an elink since they have been communicating heavily for a longer time. They continue to do this until the second interval shown, during the third interval the elink is still there but communication has slowed down. This elink is therefore removed in the fourth interval.

The bottom set of graphs show the average waiting time per link. One can see that, for instance, in the first interval there is a lot of congestion between nodes 9 and 13, and between nodes 13 and 14. This is most likely caused by the heavy communication between nodes 1 and 14. In the next intervals,

an elink supports this communication burst so part of the congestion on the base network is alleviated.

In order for the selection to be done fast enough, two important simplifications are made here. The first one is that the elinks are selected using a greedy heuristic, rather than exploring all possible elink selections and choosing the one yielding the global optimum of the cost function considered. The second simplification is that we do not predict the traffic for the upcoming time interval, but rather assume this traffic distribution will be equal to the one measured in the preceding interval. We analyzed the impact of both these optimizations in Sections 6.4.1 and 6.4.2 respectively, and found that they do allow for a network performance that is close to one using optimal placement and perfect traffic prediction.

4.4 Simulation framework

We have based our simulation platform on the commercially available Simics simulator [Magnusson et al., 2002]. It was configured to simulate a multiprocessor machine resembling the Sun Fire 6800 server [Sun Microsystems, 2003b], with up to 64 UltraSPARC III processors clocked at 1 GHz and running the Solaris 9 operating system. Stall times for caches and main memory are set to the values found in Sun's data sheet for the UltraSPARC III Cu [Sun Microsystems, 2003a]: 2 cycles access time for L1 caches, 19 cycles for L2 and 100 cycles for SDRAM.

Cache coherence is enforced by a directory-based protocol, similar to the one pioneered in the Standford DASH multiprocessor machine described by Lenoski et al. [1992]. The coherence controllers and the interconnection network are custom extensions to Simics. They model a full bit-vector directory-based MSI-protocol and a packet-switched 4×4 torus network with contention and cut-through routing. To model a reconfigurable network, a number of extra point-to-point links can be added to the torus topology at any point during the simulation. A model of the network router, which connects each processor node to its four neighbors on the base network and to a number of elinks, is shown in Figure 4.8. Contention in the central crossbar is not modeled, therefore only output buffering is needed to hold packets while the links are occupied.

The network links in the base network are 16 bits wide and are clocked at 100 MHz. In the reported experiments, the characteristics of an elink were assumed to be equal to those in the base network, yielding a per-hop latency that is the same for an elink as for a single base network link. Both coherence

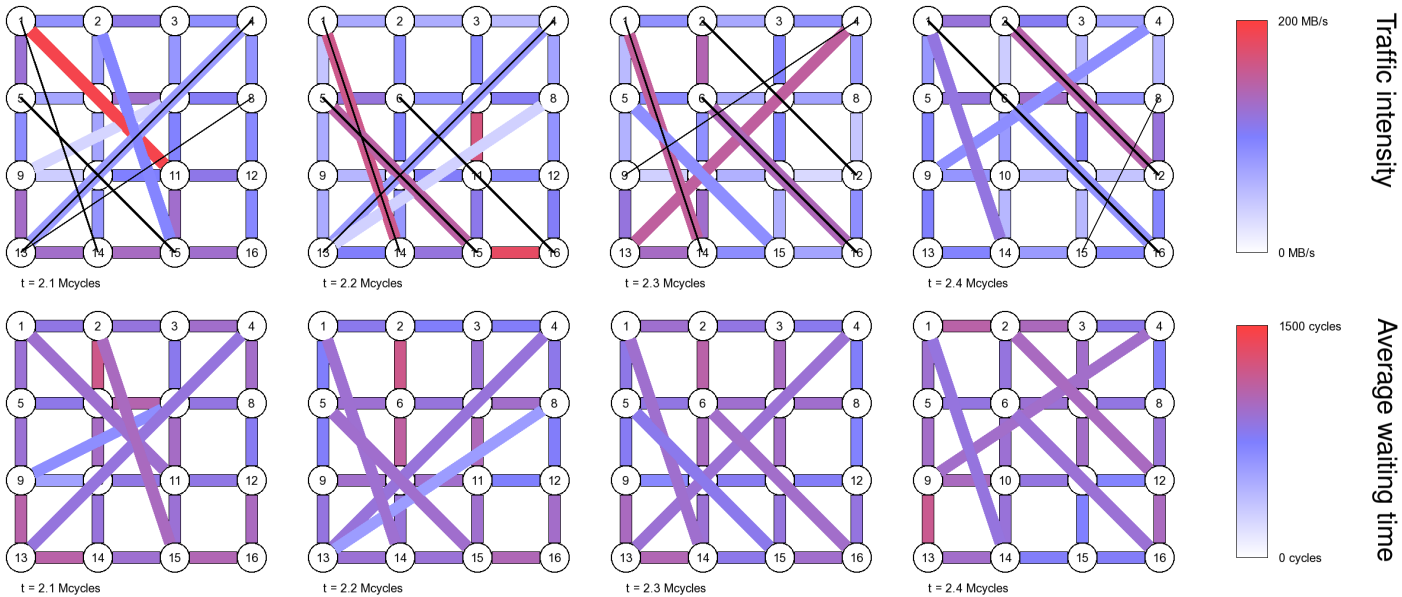

Figure 4.7: Illustration of the elinks placement through time. Top: traffic intensity on each link, and the four highest communicating node pairs (black lines). Bottom: average waiting time per link. Both are for a simulation of the `radix` benchmark on a 4×4 mesh and reconfiguration characterized by $n = 4$, $f = 2$ and $\Delta t = 100$ µs.

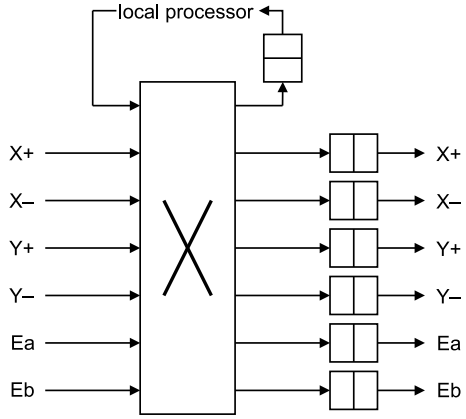

Figure 4.8: Switching architecture of a network node. Packets from the local processor, four base network neighbors ('+' and '–' directions in both *X* and *Y* dimensions) and, in this case, two incoming elinks (*Ea* and *Eb*) enter the network router at the left. The data is routed through the central crossbar to the correct output port, and buffered until the outgoing link is available.

traffic (read requests, invalidation messages etc.) and data (the actual cache lines) are sent over the network. The resulting remote memory access times are representative for a Sun Fire server (around 1 µs on average).

To avoid deadlocks, dimension routing is used on the base network. Each packet can go through one elink on its path, after that it switches to another Virtual Channel (VC)[2] to avoid deadlocks of packets across elinks. For routing packets through the elinks we use a static routing table in each node. For each destination, it tells the node to route packets either through an elink starting at that node, to the start of an elink on another node, or straight to its destination, the latter two using normal dimension routing. At every network reconfiguration, all routing tables are updated.

Since the simulated caches are not infinitely large, the network traffic will be the result of both coherence misses and cold, capacity and conflict misses. To make sure that private data transfers do not become excessive, a first-touch memory allocation was used that places data pages of 8 KiB on the node of the processor that first references them. Each thread is pinned down to one processor (using the Solaris `processor_bind()` system call), this way the thread stays on the same node as its private data for the duration of the program.

[2]Actually another *set* of VCs is used since we already employ separate request and reply VCs to avoid *fetch deadlock* [Leiserson et al., 1996] at the protocol level.

4.5 Benchmarks

The SPLASH-2 benchmark suite, described by Woo et al. [1995], was chosen as our machine's workload. It consists of a number of scientific and technical applications using a multi-threaded, shared-memory programming model. Thread creation and synchronization are done using the UPC PARMACS macro's [Artiaga et al., 1998], employing the *solaris.threads* threading model. Because some of the default benchmark sizes are too big to simulate their execution in a reasonable time, smaller problem sizes were used (see Table 4.1). For some of the benchmarks, we added a `*scale` variant in which the size of the data set is scaled linearly with the number of processors.[3]

Since our scaling down of the problem size influences the working set, and thus the cache hit rate, the level 2 cache was resized from an actual 8 MiB on a real UltraSPARC III to 512 KiB. Also, the associativity was increased to 4-way (compared to 2-way for the US-III) after we experienced excessive conflict misses in Solaris' internal structures with the 2-way caches. Overall, this resulted in realistic, 93–97% hit rates for the L2 caches. 50–60% of L2 misses were cataloged as coherence misses (resulting in communication among different processors), the remaining 40-50% were cold, conflict or capacity misses.

[3] The `fft` benchmark requires its input set to change in multiples of four, which is why we did not run `fftscale` on a 32-node network.

Code	Problem size
barnes	8192 particles
cholesky	tk15.O
cholesky29	tk29.O
fft	256K points
fft4M	4M points
fftscale	256K / - / 1M points
fmm	8192 particles
lu	512×512 matrix
luscale	512^2 / 1024^2 / 2048^2 matrix
ocean.cont	258×258 ocean
ocean.contscale	258^2 / 514^2 / 1026^2 ocean
radiosity	test
radix	1M integers, 1024 radix
radixscale	1M / 2M / 4M integers, 1024 radix
raytrace	teapot
volrend	head
water.nsq	512 molecules
water.sp	512 molecules

Table 4.1: SPLASH-2 benchmark applications and their problem sizes that were used throughout this study. For the *scale variants, the input sizes given are for 16-, 32- and 64-node networks respectively.

5

Speeding up design-space explorations

See everything;
overlook a great deal;
correct a little.
— Pope John XXIII

When designing the interconnection network for a multiprocessor machine, a multitude of design decisions needs to be made. This is the case for conventional, static networks, but even more so for reconfigurable networks, which have a number of additional parameters such as the number of supported elinks or the reconfiguration interval. Some of these will be fixed by the choice of technology. In other cases, the technology that is to be used is still under consideration, and comparisons need to be made among the performance of different choices. Even within a given technology there are often trade-offs to be made. With tunable VCSELs for instance, the number of wavelength channels supported depends on the required tuning speed.

From all these possible choices, one network architecture needs to be found that is *optimal,* in regard to a combination of design cost, fabrication cost, power requirements, performance, etc. Without having specific knowledge of the technologies that might be used in future implementations, we will limit ourselves in this work to the performance aspect as a function of basic technological parameters. To this end, we use the parametric network model from Section 4.1 with a range of paramter values. Additionally, we show simulation results for the selective broadcast implementation from Section 4.2.

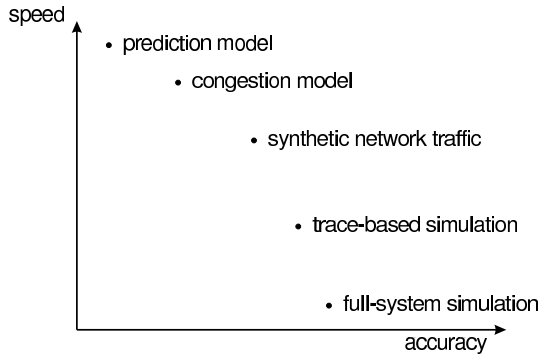

Figure 5.1: Speed versus accuracy of different reconfigurable network evaluation techniques

Our simulation platform allows us to make measurements on the interconnection network, while traffic is being fed to it by processors running actual applications. This makes the traffic highly realistic, compared to other approaches where the network traffic is synthetically generated using simple models (uniform, hot-spot, etc.). Unfortunately, simulating the execution of a complete application in this way takes at least several hours. This is acceptable in the final stages of a design, where the performance of one proposed implementation needs to be validated against a preset performance target, or when the stability of the network under a wide range of inputs must be verified. In this case, highly detailed simulations are required that model as much aspects of the proposed network implementation as possible.

This slow, detailed simulation, however, does not allow a network designer to explore the design space, i.e., to quickly compare design points and gain insight into the relationships and trade-offs that exist among different parameters. For this, a different method is needed that can quickly provide a prediction of the performance of a given design. Note that at this stage, mainly *relative* predictions are needed because the designer wants to *compare* designs. *Absolute* predictions are usually not necessary, or can be obtained by doing one detailed simulation with which the predicted results can be calibrated. Most prediction techniques therefore focus on relative accuracy, which is easier to obtain. The fact that they often have systematic errors and thus only yield a low absolute accuracy is usually not a problem in practice.

In this chapter, we present three tools that allow for quick comparisons among network design points. Section 5.1 describes a technique that takes the traffic trace recorded in one detailed simulation, and uses this traffic to predict the remote memory access latency of machines with other inter-

connection networks. In Section 5.2, another prediction model is described that accounts for network congestion. The third technique, presented in Section 5.3, is a method of statistically generating network traffic with the required temporal behavior, but allows one to use a shorter trace than a complete execution of the benchmark program. Together, they form a range of techniques to evaluate reconfigurable network performance, each with a different trade-off between simulation speed and accuracy (see Figure 5.1 for a schematic representation). They can help a network designer throughout the different stages of a design-space exploration, and will also be used in Chapter 6 in our performance exploration of reconfigurable networks.

5.1 Predicting network performance

In this section, we present a fast prediction method for the performance of reconfigurable networks. It is based on trace-based simulation, but goes one step further: we start with a trace of the traffic from one full-system simulation, but we do not actually simulate the flow of packets in our subsequent evaluation of the different networks. Since the reconfiguration of the network manifests itself only as a modification of the network topology, it is possible to determine the distance a packet will have to travel in the new network as compared to the old network. This allows us to rapidly predict new packet latencies, and estimate, to a certain level of accuracy, the resulting new average remote memory access latency. This will be our performance metric for the network.

5.1.1 Prediction model

Our performance prediction method is based on only one full-system simulation run per benchmark. The prediction is parametrized on the constraints imposed by the network (n, f and Δt, or the properties of the SOB system from Section 4.2), and can therefore predict the performance of a range of candidate networks, while still relying on only a small number of long running simulations. For each benchmark, this prediction is derived using the following steps:

- A single full simulation is done of each benchmark, using a non-reconfigurable network (referred to as the baseline simulation), yielding a list of memory accesses and a list of network packets.

- Using the list of network packets, the traffic exchanged between each node pair is calculated for each interval of duration Δt.
- The placement of the elinks, given the traffic pattern just computed, is determined for each interval using the elink selection algorithm.
- The latency of each memory access is reviewed, for accesses that would benefit from an elink this latency is reduced.
- Using the new latency distribution over the different processors, an average latency reduction is derived.

Now, each of the above stages is explained in more detail.

Full simulation

We start by doing one full-system simulation (per benchmark), using the simulation platform described in Section 4.4. Only the base network is active, so this simulation also serves as the baseline against which we calculate the speedup to determine the performance of a reconfigurable network. Our simulator creates a list of memory references that cannot be satisfied by the local node, and a second list of all packets that were sent through the network. Each memory reference is annotated with the time the request started, the requesting node, the home node and the measured access latency. For network packets, we store the sending time, the source and destination nodes and the packet size. Note that there are only two sizes of messages generated by the coherence protocol: 16-byte packets with only control information (read request, invalidate, etc.) and 80-byte packets that contain control information plus a complete cache line of 64 bytes.

Determining the elink placements

The packet trace is divided into intervals of duration Δt. For each interval, sums are made of the number of bytes that were exchanged between each of the $p(p-1)/2$ node pairs (with p the number of processors or nodes). If bidirectional elinks are used, traffic in both directions is added together. When we have the traffic profile for the interval, we use the normal elink selection algorithm, described in Section 4.3, to determine elink placements for the next interval.

Correlating memory accesses

The metric that makes network performance visible to the processors, is the remote memory access latency. Therefore, we have chosen the relative mem-

ory access latency reduction (compared to the baseline memory latency) as the metric with which to compare different networks. We will now estimate the new memory access latency as a function of the selected elinks.

We enumerate the memory accesses of the execution and represent each access by its sequence number $i \in \{1, 2, \ldots, m\} = I$. Every memory access that requires network traffic is initiated by the processor on one node and serviced by the directory on another node, the home node of the memory word. We therefore connect this memory access to the node pair made up by these two nodes. The distance between these nodes is measured, both before and after adding the elinks, and the memory access i is tagged with these two distances: its *baseline distance* $d_b(i)$ and its *elink distance* $d_e(i)$.

Memory access latencies (taking at most a few microseconds) are significantly shorter than the considered reconfiguration intervals (100 µs and upwards), so there should be no problem of accesses spanning several intervals. There are memory accesses that require intervention by a third node, in particular if the memory access is a write and some third node needs to invalidate or write back the word. However, these transactions involving three or more nodes are not very common (in our simulations, their fraction in total memory access latency was always less than 10%). Besides, about half the time of these accesses is still spent in communicating between the two primary nodes. Therefore, in this model, we ignore speedups of the traffic between nodes other than the two primary nodes.

Calculating new latencies

We shall use the notation $\langle x(y) \rangle_{\text{range}(y)}$ to denote the average of the function $x(y)$ over the specified range of y. Let $L_b(i)$ be the baseline latency of memory access i. First, we calculate the average memory access latency, over the course of the baseline simulation, for all memory accesses with the same distance:

$$L(d) = \langle L_b(i) \rangle_{\{i\,:\,d_b(i)=d\}}$$

The average baseline access latency can be computed as a weighted average of these per distance latencies, with the number of accesses per baseline distance $N_b(d)$ as weights (N is the total number of remote memory accesses in the simulation):

$$L_b = \langle L_b(i) \rangle_I = \frac{1}{N} \sum_d N_b(d) \cdot L(d)$$

$L(d)$ gives the estimated memory access latency as a function of the distance between its primary nodes. As indicated by the notation, we assume latency

is *only* a function of this distance, and does not change when adding reconfigurable elinks to the network. Therefore, the predicted access latency of memory access i after adding elinks, $\hat{L_e}(i)$, would be the $L(d)$ associated with the elink distance d_e:

$$\hat{L_e}(i) = L\left[d_e(i)\right]$$

Note that we will use $\hat{L}$ to denote the value of L as estimated by our model, a measurement of L using simulation will be written as L. Our estimate for the average memory access latency after adding the elinks, $\hat{L_e}$, can now be computed. We again count the number of accesses per elink distance $N_e(d)$ to compute the new average as a weighted average of $L(d)$ using $N_e(d)$ as weights:

$$\hat{L_e} = \left\langle \hat{L_e}(i) \right\rangle_I = \frac{1}{N} \sum_i L\left(d_e(i)\right) = \frac{1}{N} \sum_d N_e(d) \cdot L(d)$$

When written this way, it can be seen that our prediction will be accurate if (1) the new distance distribution $N_e(d)$ can be accurately predicted and (2) the per distance latency $L(d)$ does not change after adding elinks.

Figure 5.2 shows that we can accurately estimate the number of accesses per elink distance, using the traffic pattern from a simulation with a non-reconfigurable network. For each distance, the graph shows the number of memory accesses in the baseline simulation N_b, the predicted number of accesses N_e using the method described above, and the actual number of accesses, measured in a simulation where the reconfigurable network is added to the machine. We can clearly see that adding a reconfigurable network greatly reduces the average distance, also this new distance distribution can be estimated accurately based on traffic patterns obtained from the baseline simulation run.

The next question is whether memory latency is indeed only a function of distance, and does not vary with other topological parameters. Figure 5.3 shows this memory latency $L(d)$ for the baseline (bars) and a few different networks (cross-hairs), as measured in simulations with the reconfigurable network in place. For $d = 4$ the difference is obvious. However, $d = 4$ accesses are almost eliminated after adding elinks (as can be seen in Figure 5.2, where the gray and black bars for $d = 4$ are barely visible), making this measured average (and the apparent rise in latency for most of the networks) unreliable. Moreover, since the number of $d = 4$ accesses, and thus the weight of $L(4)$ in the computation of $\hat{L_e}$, is so small, the value of $L(4)$ does not influence the result much. Lower distances show far less variation, the difference being due to the reduction of congestion after adding more links to the network. This congestion is not modeled in this prediction method, its influence will be further explored in Section 5.2.

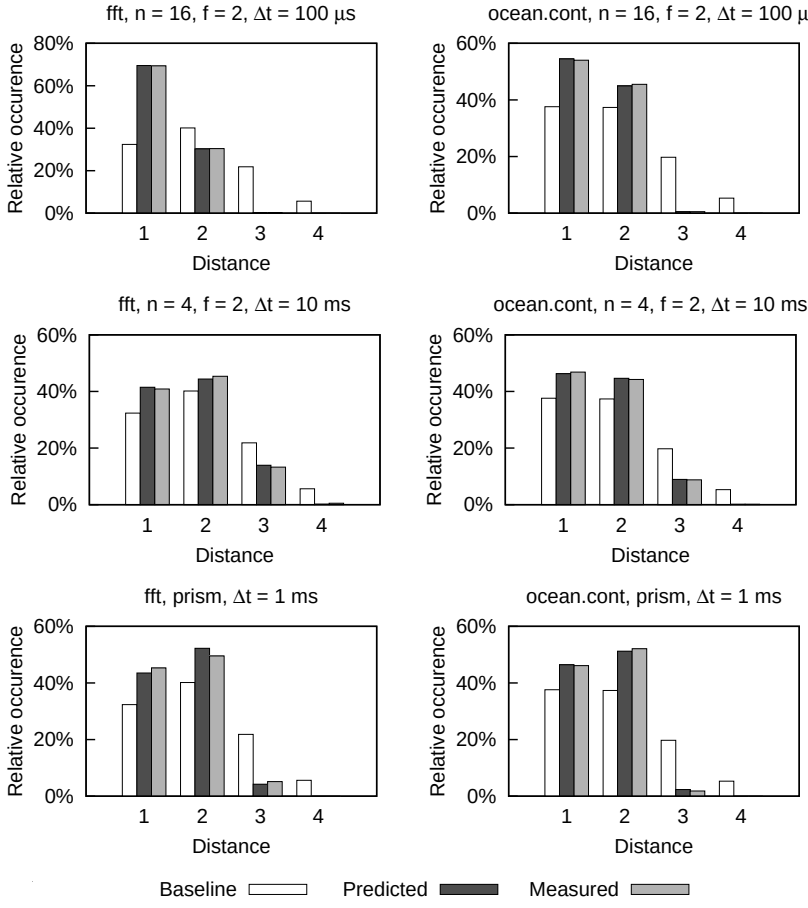

Figure 5.2: Baseline, estimated and actual distance distribution of memory operations for the `fft` and `ocean.cont` benchmarks on a selection of 16-node networks. When adding reconfiguration abilities to the network (moving from *baseline* to *measured*), the fraction of long-distance memory accesses drops significantly. Moreover, the *measured* and *predicted* distributions are in good agreement.

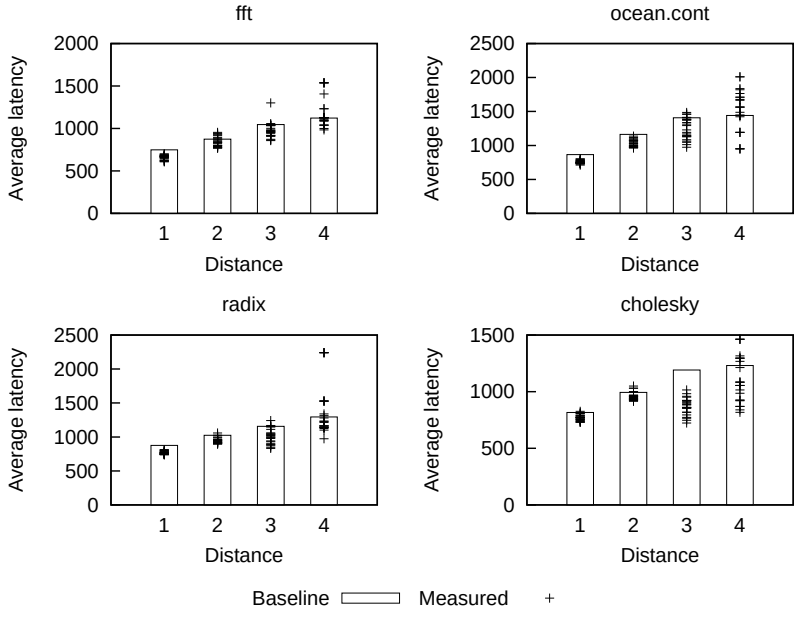

Figure 5.3: Variation of average memory latency per distance for different network parameters on a 4×4 torus network

5.1.2 Prediction accuracy

Figure 5.4 shows the result of our prediction model, the memory access latency improvement over the baseline after adding elinks, compared to the values measured in an execution-driven simulation, for a number of different reconfigurable 16-node networks. A linear regression of the form

$$\hat{P} = \alpha + \beta \cdot M$$

is calculated (with P and M the predicted and measured latency reduction, respectively). The correlation coefficient r is high, so a strong, linear correlation exists between measurement and prediction. Our method can therefore be used to very quickly compare different proposals for network parameters. This makes it a very useful tool for design-space explorations, where the optimal solution needs to be found from a large collection of candidate networks.

In Figure 5.5 the *predicted* and *measured* latency improvements are shown again. This is done for a number of elinks ($n = 2, 4, 8$ and for the implementation using the prism from Section 4.2) and different reconfiguration intervals (Δt = 100 μs, 1 ms, 10 ms). For comparison, the *fixed* case is added: here, the same number of elinks is added in random positions (favoring longer links), but they are not reconfigured at runtime (the average result from five different placements is reported).

From this graph, it is obvious there is a systematic underestimation present. This is mostly due to the fact that we have not included congestion in our method. However, the *relative* prediction accuracy among different network parameters, which is the most important value when comparing different suggested network implementations, is much better. By doing one more execution-driven simulation per benchmark, with which our model can be *calibrated*, better absolute accuracies can be obtained. The third bars in Figure 5.5 show this *corrected* data. They are obtained by doing one extra simulation (per benchmark) with a reconfigurable network, defined by n = 16, f = 2, Δt = 100 μs, in place. The predicted and measured latency improvements are then known for this network. All predictions are subsequently scaled with a constant factor, dependent only on the benchmark application, such that the corrected prediction of the n = 16, f = 2, Δt = 100 μs network matches its measured value. When a network with a large latency improvement is chosen to determine the calibration factor, the influence of the offset of our predictions (the α parameter in Figure 5.4) is minimized.

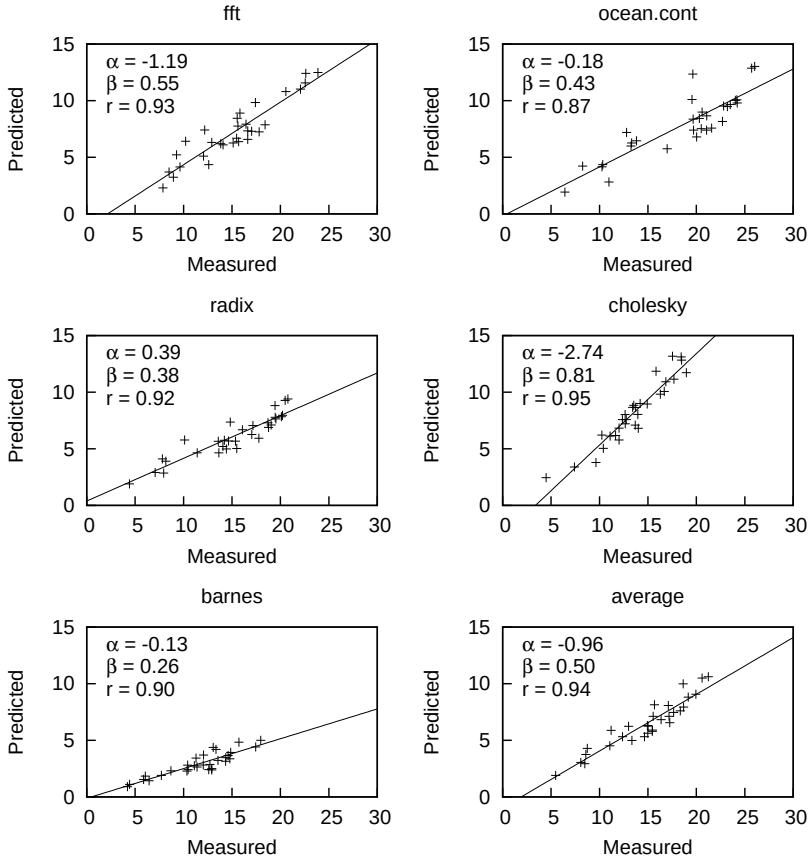

Figure 5.4: Predicted versus measured latency reduction for a variety of 16-node network implementations. α, β and r represent a linear regression of the form $\hat{P} = \alpha + \beta \cdot M$ and its correlation coefficient.

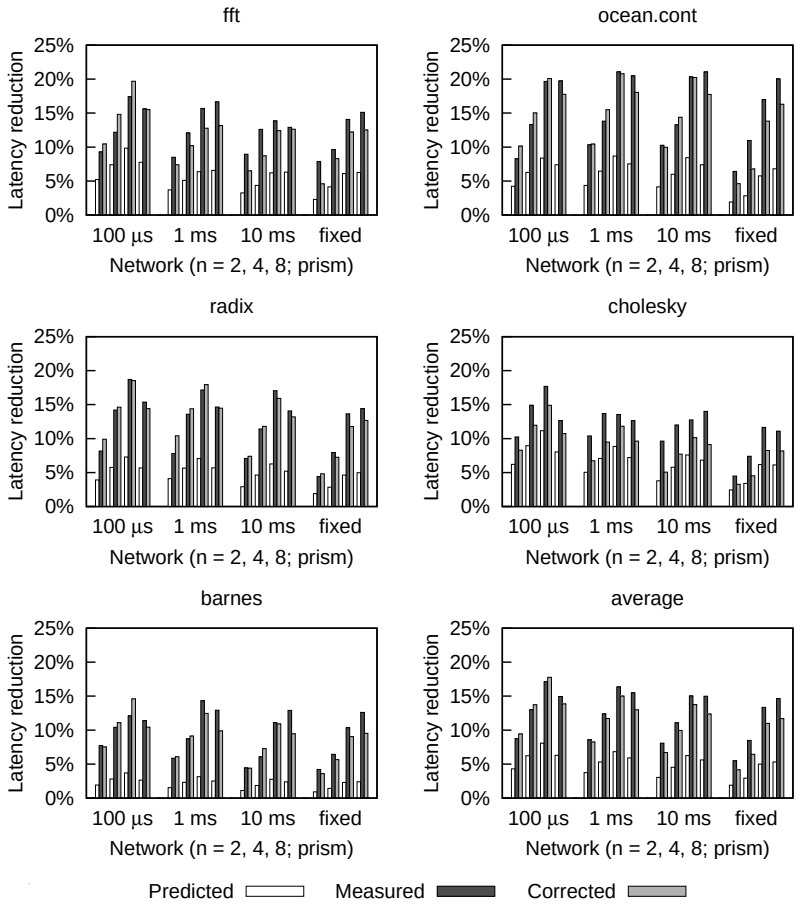

Figure 5.5: Latency improvement after adding elinks: *estimated* using our predictor, *measured* in simulation, and the *corrected* prediction using a constant factor per benchmark

5.1.3 Improving accuracy

Don't look where you fall,
but where you slipped.
— African Proverb

A number of assumptions were made in the algorithm described in Section 5.1.1. First of all, the traffic pattern from the baseline simulation is used. When adding elinks, traffic streams could potentially appear or disappear. Since the communication pattern is the result of an algorithm that is implemented by the benchmark code, which is in most cases unaffected by the platform on which it is running, this pattern should not change too much. Selecting the elinks based on the traffic pattern is done using the same method as that used at runtime, so here no additional error can be introduced. Figure 5.2 showed that we can very accurately predict the distance distribution of memory operations after adding a reconfigurable network. Memory latency is, however, as Figure 5.3 clearly shows, not just a function of hop distance, as was assumed in Section 5.1.1.

The component which causes the largest error in our prediction is the reduction of congestion. Clearly, adding links to the network increases bisection bandwidth and thus increases the total capacity of the network. We even place the elinks such that large traffic flows are moved away from the base network, speeding up not only the traffic that was moved but also the traffic that remains on the base network. This can be clearly seen in Figure 5.6 which shows the distribution of packet waiting times: for the baseline this distribution drops off slower, which means that there are more packets that experience high congestion. This shows there is room for a more complicated, slower method in which congestion is modeled resulting in greater accuracy. Such a method is presented in Section 5.2.

5.1.4 Reduction in simulation time

Figure 5.7 shows the computation times required for both the full-system simulations and our prediction model, the former taking several hours while the latter can be completed in just a few minutes. We did not include the cost of the initial simulations in the computation time for our method, since this should only be done once and can subsequently be reused for any number of network parameter sets. Our method therefore allows a reduction in computation time by about two orders of magnitude. Note that we already

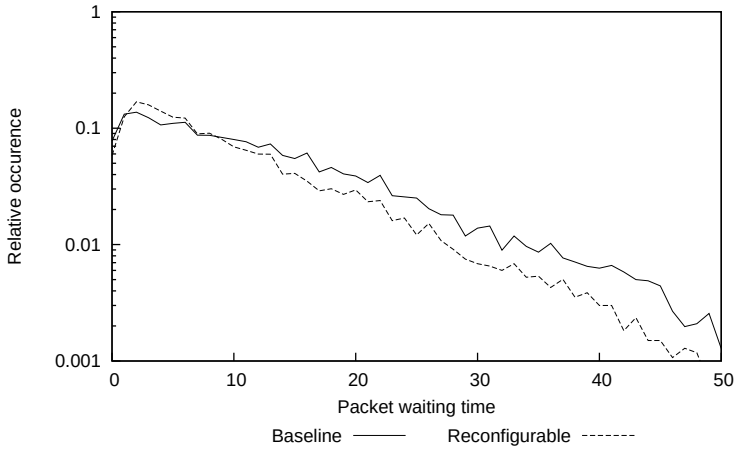

Figure 5.6: Distribution of the waiting time per packet, for the baseline simulation and a reconfigurable network simulation (with $n = 4$, $f = 2$ and $\Delta t = 100$ μs) of the `fft` benchmark run on a 16-node network

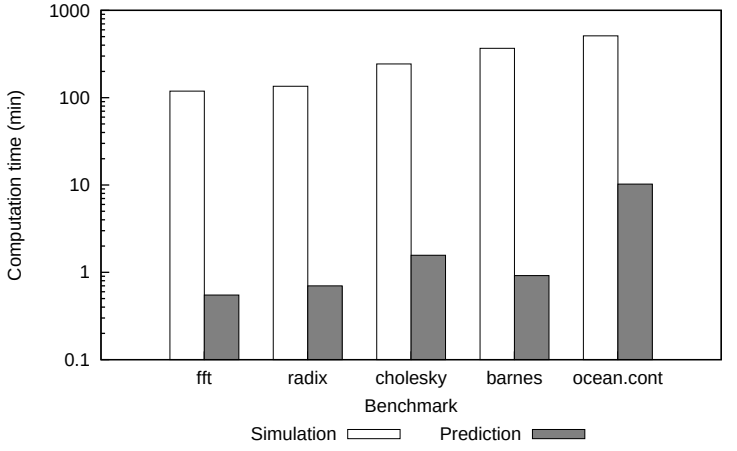

Figure 5.7: Computation time (in minutes) required for a full simulation and our prediction for four of the benchmarks considered. Note that the Y-axis is in a logarithmic scale.

employed scaled-down benchmarks and an in-order processor model. If one were to do these simulations with a highly detailed simulator, the computation time can easily be an order of magnitude higher. In contrast, the prediction model was implemented by a Python script, optimized for maintainability and extensibility. A speed-optimized implementation written in, for instance, C would make the difference in computation time even larger.

5.2 Congestion modeling

The simplicity of the previous method resulted in very short runtimes allowing quick design-space exploration. The fact that congestion was not modeled is a source of error, however, which in some cases should not be ignored. Clearly, congestion plays an important role in the performance of a network where traffic can be directed away from hot-spots, as is attempted by adding elinks. In this section, we model congestion on the communication network, yielding a different performance prediction method which has higher accuracy in congested networks, at the cost of being more complex and having a slightly longer runtime.

For each reconfiguration interval, the average waiting time for packets on each link is computed using the Pollaczek-Khinchin (PK) mean formula by regarding each network link and its preceding buffer as an M/G/1 queueing system. At each network node, a set of buffer+link systems is connected. The PK mean formula requires the packet inter-arrival time and service time distributions, which are also derived. Adding waiting times over all links on the path of a packet gives us the total waiting time. We finally compute the average expected packet latency which is used as the performance indicator for the network.

Note that the queueing theorems that are used, including the PK means formula, are only valid when the system is operating in steady state. This is not the case for a realistic network. Applying the PK means formula in this settings is therefore not theoretically accurate. It does, however, provide a reasonably good heuristic, as will become clear in the results section.

5.2.1 Contention model

We will now present our contention model for reconfigurable networks, based on only one full-system simulation run per benchmark. This model is again parametrized on the values of n, f and Δt, and can therefore predict the contention on a range of candidate networks, while still relying on only

a small number of slow, full-system simulations. For each benchmark, this prediction is derived using the following steps:

- A single execution-driven simulation is done of the benchmark, using a non-reconfigurable network (also referred to as the baseline simulation), yielding a list of network packets.
- Using the list of network packets, the traffic exchanged between each node pair is calculated for each interval of duration Δt.
- The placement of the elinks, given the traffic patterns just computed, is determined for each interval.
- By viewing each buffer/link-combination in the network as an M/G/1 queueing system, we derive the mean waiting time for a packet traversing this link.
- Using the waiting times on each link, and the distribution of packets over the links, a global average waiting time can be computed.

In the rest of this section, each of the above stages is explained in more detail.

Full simulation

The first step is the same as for our prediction model from Section 5.1.1, i.e., we do one full-system simulation per benchmark and store the packet trace. Memory accesses were not recorded in our experiments, since we only estimated packet latency. Connecting packets to memory accesses, and subsequently estimating the improvement in memory access latency, would be a trivial extension to this model.

Determining the elink placements

Just as in the prediction model, we divide the trace into intervals of length Δt and use the elink selection algorithm to determine elink placements.

Average waiting time per link

For each interval, we again have the topology of the network (defined by the sum of base network links and selected elinks) during that interval, and the number of packets sent between each node pair. Since the path each packet follows is deterministic (determined by the static routing table for the elinks, and by dimension routing for the base network, see Section 4.4), we also know the load on each network link.

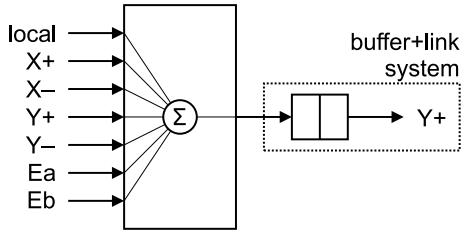

Figure 5.8: We divide the network into independent queueing systems, each consisting of one output buffer and its succeeding external link which provides the *service*. Traffic can enter the system via the crossbar switch through several incoming links.

We will view each network link and its preceding buffer as an M/G/1 queueing system (Figure 5.8). This is a system in which packet arrivals are specified by a memory-less (Poisson) process yielding an independent and exponentially distributed inter-arrival time, an arbitrary service time distribution and one server (the *service* provided in this system is transmission of a packet over a slow – compared to the internal speed of the router – external network link). The buffer size is considered infinite. The mean waiting time in this system is given by the Pollaczek-Khinchin (PK) mean formula [Robertazzi, 2000]:

$$E[W] = \frac{\lambda E[S^2]}{2(1-\rho)} \tag{5.1}$$

with W the waiting time, S the service time, λ the intensity of the arrival process, ρ the server load and $E[X]$ denoting the stochastic expected value of the quantity X. This will be done for all links in the system, both the links in base network and the elinks. Note that in this discussion, links are regarded as unidirectional. Indeed, even though we are using a request-reply protocol, the replies are usually bigger than the requests (since most of them contain an additional 64 bytes of data) so the load on links in one direction may be much higher than the load in the other direction.

First we determine the arrival rate λ. Assuming packet arrival is regulated by a Poisson process (it is not, but we will compensate for that later), λ is the probability that a packet arrives in a given clock cycle. There are $\Delta t \cdot C$ cycles in each reconfiguration interval (with C the frequency of the system clock), and we have already computed the number of packets that will traverse this link during the current interval (call this N). Therefore,

$$\lambda = \frac{N}{\Delta t \cdot C}$$

Next, we describe the service time. In our system this means the transmission time of the packet. This time is used to compute how long a packet following the present one will have to wait before it can use the link. The fact that transmission of a packet is pipelined through several links has no influence on the service time. Each link has a specified transfer rate (say B, measured in bytes/second). A packet can either be small (16 bytes) or large (80 bytes). We count how many packets of each length traverse each link during each reconfiguration interval (N_{16} small packets and N_{80} large packets). The probability of a given packet being small is therefore $p_{16} = N_{16}/N$, the probability of it being large is $p_{80} = 1 - p_{16} = N_{80}/N$. Small packets require a service time of $S_{16} = 16/B$, for large packets this is $S_{80} = 80/B$. The service time is therefore a Bernoulli distribution, its second moment $E[S^2]$, which we need to compute the mean waiting time, is:

$$E[S^2] = p_{16} \cdot {S_{16}}^2 + p_{80} \cdot {S_{80}}^2$$

The server load ρ translates to the link utilization factor. We transmit $16 \cdot N_{16} + 80 \cdot N_{80}$ bytes over the link, which has a total capacity to send $B \cdot \Delta t$ bytes during one interval, thus:

$$\rho = \frac{16 \cdot N_{16} + 80 \cdot N_{80}}{B \cdot \Delta t}$$

We can plug these values into the PK mean formula (Equation 5.1), which will yield the mean waiting time, or the average time a packet spends in the transmit buffer preceding the link under investigation.

There is one large deficiency in our reasoning thus far: the packet inter-arrival process is considered memory-less (i.e., not dependent on the number of arriving packets in previous cycles). In reality, this is clearly not the case. Indeed, traffic entering a buffer+link system comes from a limited number of other, incoming links, and each of these links also has a limited throughput. Therefore, if all incoming links deposit a packet in the buffer during this cycle, *no other packets* can enter the system for at least a time S_{16}, the time required for a small packet to traverse an incoming link. Only *after* this time a new packet can arrive, which clearly violates the assumption of a memory-less arrival process.

The derivation of the PK mean formula depends heavily on the arrival process being memory-less. It is therefore not easily done, and probably not even possible analytically, to repeat the PK derivation with a generalized inter-arrival process. This is mostly because the derivation assumes the number of packets in the queue is ergodic, which is referred to as the *PASTA* property of a Poisson process: *Poisson arrivals see time averages*. In short, it

states that the number of packets in the queue *at the instant a packet arrives* (a Poisson arrival), has the same distribution as the number of packets in the queue *at any given time* (the time average). This implies a statistical independence between the arrival times of packets already in the queue and the arrival time of the newly arriving packet.

Furthermore, two packets arriving through the same incoming link into this buffer+link system will never have to wait for each other *in this system* (one may have waited for the other in the buffer preceding the incoming link, but since we are now focusing on just one buffer+link system this is not to be considered here). The second packet can only arrive a time S_{16} or S_{80} after the first one (the first packet has to clear the incoming link before the second packet can arrive). In an uncontended system, the first packet can clear the next link during this time so the second packet can continue without waiting. In a system with contention, the first packet may be delayed in this systems buffer due to packets from other incoming links. This will delay the first packet, possibly also delaying the second packet, but all delay of the second packet can be attributed to the packets from the other incoming links. Therefore we conclude that packets, entering the system through the same incoming link, have no influence on each others' waiting time in this system.

Combining these two observations, the wish for statistical independence between an arriving packet and the packets already in the queue, and the fact that packets entering the system through the same link do not influence each other, we propose the following modification to our model: instead of aggregating all packets flowing over this link into one big N_{16} and N_{80}, we separate them by incoming link into N_{16}^i and N_{80}^i, for each i being the identifier for one of the incoming links. The waiting time $E[W]^i$, which is the time spent in the buffer of this system, valid for packets entering the system using incoming link i, is now computed with the same PK mean formula as before but with

$$N_{16} = N_{16}^{\neg i} = \sum_{j \neq i} N_{16}^j$$

and likewise for N_{80}: the system is now viewed as being loaded with packets from all links *except* link i, packets entering from link i will now experience a delay that is caused only by packets from other links.

This way, we have achieved both goals stated before: packets coming in through the same link do not influence each other (N_{16}^i and N_{80}^i are not used to compute $E[W]^i$); all packets that *are* used to compute $E[W]^i$ came in through another link. The arrival times of packets in the queue are now statistically independent of the arrival time of packets entering through link i, this makes

the PASTA property applicable. The statistical dependence between some packets in $N_{16}^{\neg i}$ and $N_{80}^{\neg i}$ (because they entered through the same incoming link j) has not been removed, we will later see how this influences the results.

Calculating the total packet latency

Since we know the path each packet will take, we can add the expected waiting times in each of the buffers the packet will pass through to compute the total expected waiting time for the packet. This can be done for all packets and averaged across all reconfiguration intervals, yielding a prediction for the global average packet waiting time. If total packet latency is the desired metric, this can be computed by adding the uncontended packet latency, which is the time required for a packet to go from source to destination if no other packets are using the network, to the waiting time.

Computational complexity of the model

The computational complexity of a complete network simulation is of the order of the total number of packets sent throughout the runtime of the benchmark (for each packet, a number of computation steps are done per simulated clock cycle). The complexity of our model is lower: the final two steps work on quantities derived by aggregating all packets sent between one node pair (in one direction) and in one reconfiguration interval. Therefore, the computational complexity of our prediction model is lower than that of a full network simulation by a factor of the order of $N_{16}^{i} + N_{80}^{i}$, which, in practice, amounts to a factor of ten at least, depending on the length of the reconfiguration interval.

5.2.2 Results

Figure 5.9 summarizes the results for our model. For two benchmarks, `radix` and `barnes`, we ran the model on the traffic patterns obtained from the baseline simulation and predicted the average waiting time for a number of different values of n. We also simulated the network with the elinks and measured the waiting time for comparison. We can see that the relative accuracy, for different values of n, is good, so we can use our prediction model to evaluate the effect of adding more or less links to the network.

Since we have already done the baseline simulation, the measured value of the waiting time on a non-reconfigurable network is essentially free. If we compare a prediction of the waiting time for a network without elinks with this measurement, we can calibrate the model (we used a linear calibration

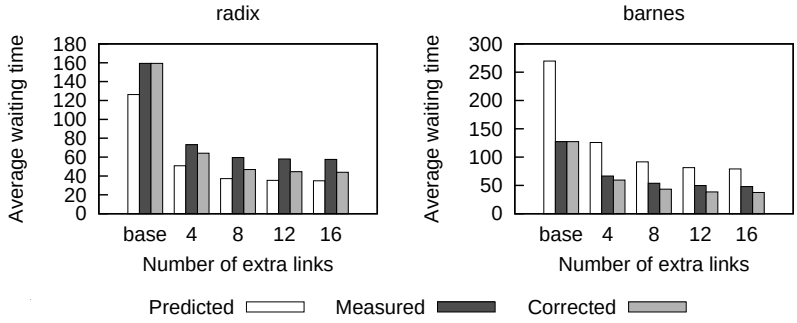

Figure 5.9: Average waiting time per packet for `radix` and `barnes`: estimated value, measured value and corrected estimate, for networks with a different number of elinks (all with $f = 2$, $\Delta t = 1$ ms and 16 processors)

factor for simplicity). The third bars in Figure 5.9 represent this corrected prediction, with a calibration factor of 1.26 for `radix` and 0.47 for `barnes`.

In Figure 5.10, we tried to predict congestion when changing the reconfiguration interval. The measurements were done for the `barnes` benchmark, with the number of elinks held constant at eight. Here, the predictions do not follow the trend of the measurements. We will try to explain this fact in the following section.

5.2.3 Discussion

The PK mean formula used in Section 5.2.1 states that waiting time is a linear function of $(1 - \rho)^{-1}$. This means that for $\rho \to 1$, the expected waiting time rises to infinity. According to queueing theory, a value of $\rho > 1$ is not even possible, since this would mean that a link has a utilization of more than 100%. However, when calculating the different expected waiting times in Section 5.2.1 we often see values of ρ that are as high as 1.2. This is because our buffer+link systems are not operating in steady state, but rather continuously stay in a transient state. Consider the situation in which, during one interval, more packets enter a buffer+link system than the link can process in that interval. Our model will compute a value of $\rho > 1$ and fail. In reality, the excess packets will just stay in the buffer and are transmitted during a later interval.

Even when link utilization approaches but does not exceed 100%, the waiting time rises without bounds. However, in our measurements the

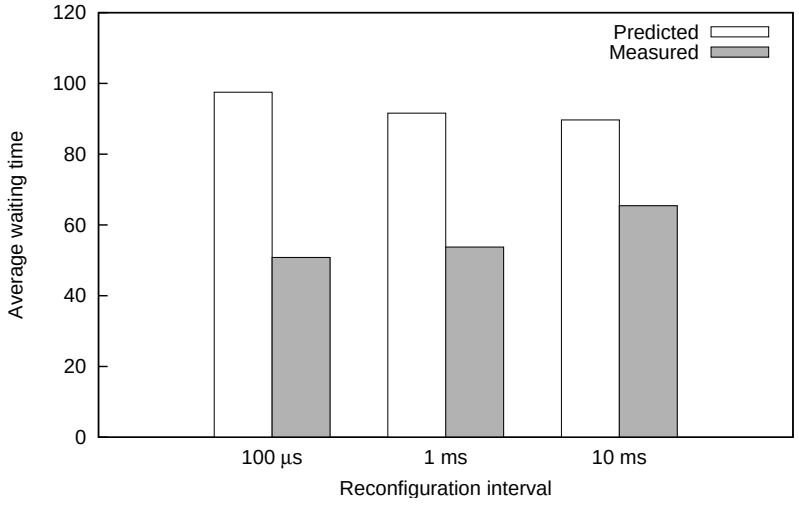

Figure 5.10: Average waiting time per packet for `barnes`: estimated and measured value, for networks with a different reconfiguration interval (all with $n = 8$, $f = 2$ and 16 processors)

waiting time never exceeds more than a few hundred clock cycles. Therefore, it is obvious that the PK mean formula as it is used now is not able to accurately model a real buffer+link system for highly utilized links. In our implementation, we have tried to counter this by limiting ρ to 90%. This way, predicted waiting times do not rise unlimited but are kept at reasonable values.

As the length of the interval decreases, fewer packets are contained in each of the N_{16}^i and N_{80}^i's (less than 10 for $\Delta t = 100$ µs). We now experience the classical problem one faces when trying to extract statistics from a very limited sampling set. The packet inter-arrival time and service time distributions can now no longer be estimated accurately. Also, network behavior seems even more dynamic because we do not average out network traffic over longer periods of time. This lowers our prediction accuracy. To isolate this effect of the reconfiguration interval, we ran our prediction model three times on a network without elinks, but still divided time into a number of fixed length intervals (see Figure 5.11). Since the network for all three predictions is the same, we should predict the same congestion. However, we see that for short intervals congestion is highly overestimated, while the accuracy improves when increasing the interval length. The same effect plays in Figure 5.10, where congestion is overestimated more for shorter

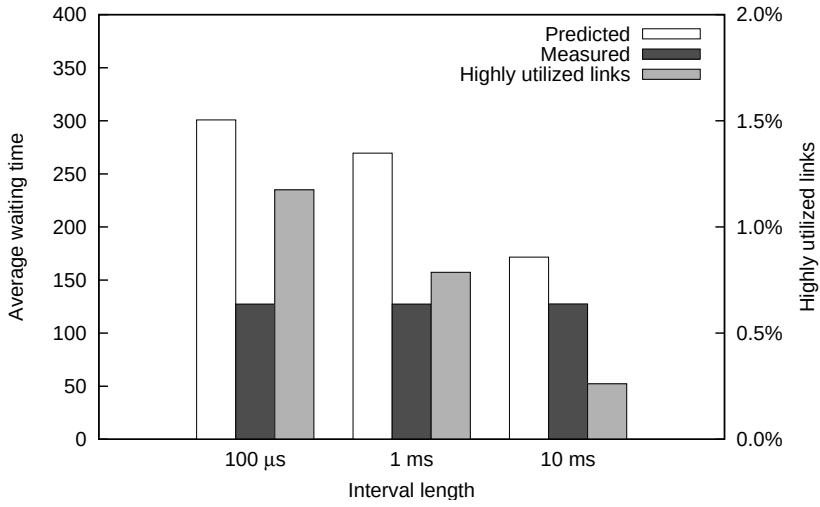

Figure 5.11: Average waiting time per packet for `barnes`: estimated and measured value, for a non-reconfigurable 16-node network. The predictions are made by dividing time in intervals of different length.

reconfiguration intervals. Figure 5.11 also shows the relative occurrence of the model having to limit link utilization to 90%. Clearly, this fraction and the prediction inaccuracy have a positive correlation.

This effect can also explain the difference in calibration factors needed for Figure 5.9: `barnes` has more communication than `radix`, and thus more highly utilized links. This results in the observed overestimation of waiting times for `barnes` which does not occur for the `radix` benchmark.

5.2.4 Improving accuracy

The handling of highly utilized links is probably the largest source of errors in our current model. Related to this is the fact that, in an M/G/1 system, the buffer size is considered infinite. In a real network this is not the case: once a buffer fills up, packets upstream are delayed. One could say that the upstream buffers now represent an extension of the full buffer making its apparent size much larger. However, traffic that uses the upstream buffers but not the one that first filled up is also affected. This effect is not included in the current model.

We also did not model a delay when a packet enters the network. Packets are created by one of the network interfaces, than they are stored in the buffer

of the first link they will go through. Remember that in our model, these packets are described as entering the system through their own incoming link. In this buffer+link system, they experience a waiting time that is dependent on traffic from other incoming links, as is described by our model. However, no waiting time is attributed to these packets interfering with each other. If the network interface is only able to generate and send packets at a slow rate – which is generally the case since request-reply protocols are used, and the number of outstanding requests is limited – this may be accurate. But if several packets are dropped into the buffer in short succession – which happens in reality when a coherence controller sends out several invalidation messages – they will have to wait there. This is the case even when no other traffic enters the buffer+link system through other incoming links, therefore our model would incorrectly state that $E[W]^i = 0$.

5.2.5 Reduction in simulation time

A typical reconfiguration interval is 1 ms or one million clock cycles. N^i_{16}+N^i_{80} now contains a few tens of packets. Since they can be processed in combination, rather than separately, this makes the computational complexity of our model at least one order of magnitude less than that of a full network simulation. Our implementation confirms this: full simulations take, for 16-node networks, about one hour, while the model usually runs in less than five minutes.

5.3 Synthetic network traffic

For measurements requiring more accuracy than our prediction methods of the previous sections can provide, simulation has to be used. Simulating the processors, caches and coherence controllers can usually be avoided once the packet trace of a benchmark's execution has been recorded. This trace can subsequently be played back on different networks.[1] Since there is a lot of similarity inside a typical packet trace, it is inefficient to play the complete trace for each simulation.

Sampling could be used, where only part(s) of the trace are played and the results for the complete program are extrapolated. However, selecting

[1]There can be interaction between network timing and application behavior, as is shown in Section 6.1, causing the network traffic on execution-driven simulations to deviate between different networks. Still, the most important properties of the traffic, such as burstiness and locality, remain the same.

the correct parts is not trivial. Especially for reconfigurable networks, care must be taken that the dynamic behavior of the traffic, which is exploited by reconfiguration, remains intact. Bertels and Stroobandt [2006] use reservoir sampling to sample accesses to a shared memory to measure communication. They want to obtain a confidence level of 95% that communication streams representing a fraction of total bandwidth of at least 0.1%, are represented in the sample population with a relative error smaller than 5%. To this end, over 1.5 million samples need to be made. Moreover, this only guarantees that all communication streams have been sampled once. Information about the instantaneous bandwidth and burstiness of the communication, which significantly affect network behavior, require an even larger sampling set.

Another solution is to generate synthetic network traffic, with the same properties as the traffic generated by the execution of a real application. In practice, simple traffic patterns are used such as uniform, hot-spot, perfect shuffle, etc., modeling traffic destinations, combined with exponential inter-arrival times [Ridruejo et al., 2005]. Yet, these patterns lack the required time-varying behavior, making them inappropriate to evaluate reconfigurable networks. Additionally, they generate single, independent packets. While this may suffice for the evaluation of message-passing architectures, traffic inside a shared-memory machine is characterized by the fact that messages are highly structured in – sometimes multi-level – request-response structures. This makes for more correlation between individual messages, which can invalidate assumptions based on the independence of packets that are commonly made when working with existing generators.

Therefore, we developed a technique to generate synthetic network traffic that includes the required spatial and temporal behavior. Our synthetic traffic has the advantage that the relevant traffic properties of a real traffic flow are preserved, but that the flow can be much shorter, equally reducing simulation time. In addition, the processors and caches, of which detailed models are needed in an execution-driven simulation, no longer need to be considered when synthetic traffic is used. This greatly reduces the complexity of the simulator and again decreases the simulation time significantly. In contrast to our prediction models, which can only predict some aspects of network performance like average packet or memory access latency, a synthetic traffic flow can be fed to a detailed network simulation model, allowing all network parameters and effects (including routing protocols, required buffer sizes, possibility of deadlock, etc.) to be taken into account. The correlation between packets is maintained by generating what we will call *packet groups*. These are sets of packets that are generated as a unit, they will stay connected throughout the simulation so that proper sequencing of related packets is maintained.

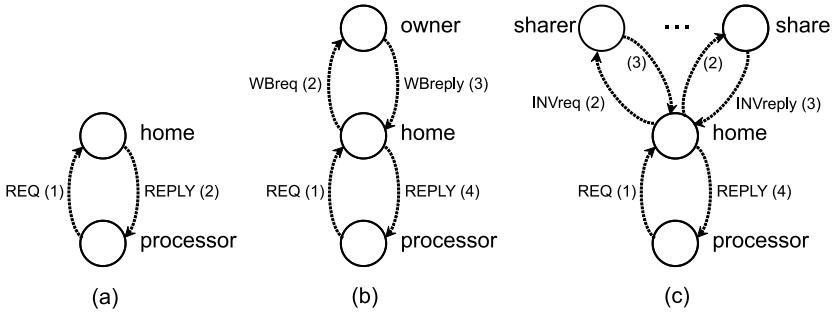

Figure 5.12: Possible sequences of packets, synthesized as *packet groups*. (a) no third-party nodes involved, (b) data is in state modified/exclusive at another node and must first be written back, and (c) data is in shared state at other nodes and must be invalidated.

5.3.1 Synthetic traffic generation

The synthetic network traffic will be generated based on a profile of the traffic we want it to represent. To this end, we define properties of the network traffic and measure their values during an execution-driven simulation. Here, the benchmark application is in control of the processors, and the traffic on the network is the result of remote memory accesses performed by this application. This results in a statistical profile of the traffic flow, which is specific for each benchmark application. This profile will subsequently be used to synthesize a new, similar traffic flow.

Packet groups

Network traffic is the result of memory accesses performed by the application. Each memory access results in a variable number of packets sent over the network, structured into request-response structures. When analyzing traffic, and later when synthesizing a traffic flow, we would like to keep this structure of memory access operations intact. Therefore we will analyze, and generate, *groups* of packets, rather than individual packets. This keeps the behavior of the synthetic traffic much closer to that of the real traffic.

The coherence protocol, described in Section 3.1.2, is in charge of supplying remote data words to the processors while keeping cached copies of the same words coherent. A remote memory access starts when the requesting node sends a request message to the home node. This home node will return the requested data word, possibly after communicating with other *third-party nodes* to enforce cache coherence. Figure 5.12 repeats the situ-

ations that can occur. (a) denotes a simple transaction in which only one request-response pair (`REQ` and `REPLY` packets) is needed. Situations (b) and (c) require involvement of one or more third-party nodes, here a number of extra request-response pairs are exchanged. They are all started in parallel upon receipt of the initial `REQ` packet, the final `REPLY` is sent once the last `WBreply` or `INVreply` arrives at the home node. Each of these three situations is generated by one *packet group*. The other aspects of the coherence protocol such as `NAK` packets and cache-initiated write-backs, only account for a very small fraction of total network traffic and are therefore not modeled here.

Number of involved nodes

As can be seen in Figure 5.12, the number of nodes involved in a remote memory operation, which will later be synthesized as one packet group, is either 2 (situation a), 3 (b) or $n > 2$ (c). By properly annotating memory accesses in the log files of an execution-driven simulation, this node count is determined for each memory access, and a distribution is made. Figure 5.13(a) shows such a distribution, for the `fft` benchmark run on a 16-processor machine. It can be seen that simple memory accesses, involving no third-party nodes, are the most common. Accesses in which the owner or just one sharer needs to be contacted account for 13%, situations where data on more than one node must be invalidated are evenly distributed and together account for about 1%.

Distribution of home nodes

In a shared-memory machine, each node contains a fraction of the total system memory. All this memory is accessible transparently to the processors in a single physical address space. Some bits of the physical address, in our implementation the uppermost ones, determine which node the contents of this address is located at.

Using memory management units and virtual addressing, available on all current microprocessors, one can play with the virtual to physical address mapping. This way the operating system can, at a page granularity (8 KiB), decide which node data should be placed on. In our simulator this is done using a first-touch algorithm: each page is placed on the node that first writes to it. This way, data private to a thread is always on the same node, requiring no network traffic.

The home node is thus determined by the address that is referenced. Since address streams, even when measured between the L2 cache and main memory, exhibit spatial and temporal locality, we would expect the ‘stream’

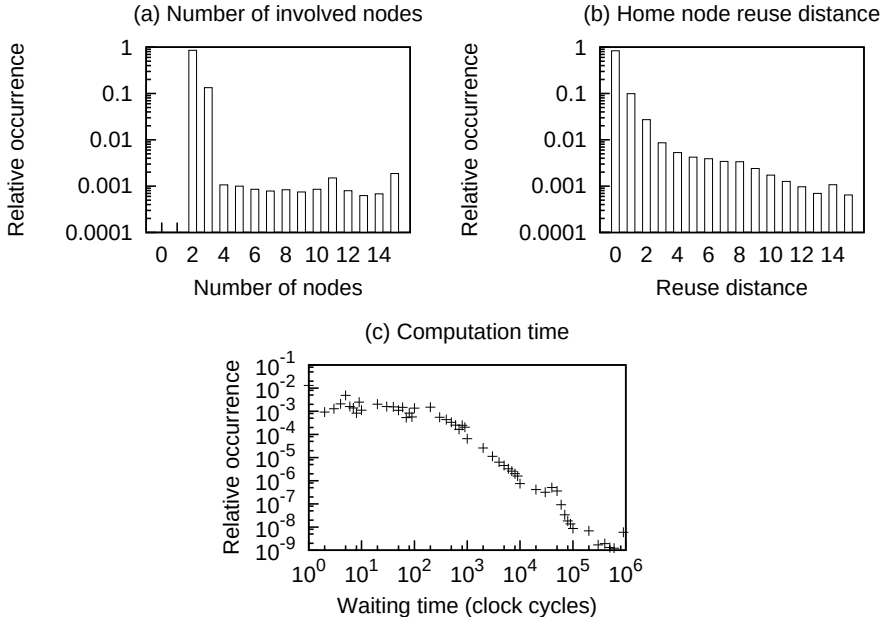

Figure 5.13: Distributions determining a traffic profile: (a) the number of involved nodes per memory operation, (b) the reuse distance of home nodes per requesting node, (c) computation time (time between requests from the same node), all for the `fft` benchmark executed on 16 processors

of home nodes also to exhibit a high degree of temporal locality. Spatial locality in the address stream, when within the same 8 KiB page, translates to the same home node being accessed again. Spatial locality in the home nodes would require spatial locality in the address stream beyond page boundaries, which is usually low and is therefore not modeled here.

We measured the temporal locality using the concept of *reuse distance* [Beyls and D'Hollander, 2001], this is the number of distinct nodes that are accessed between two subsequent accesses of the same node, by a given requesting node. If the contacted nodes are put on a stack (when contacting a node already on the stack it is moved to the top), the reuse distance is given by the distance of this node to the top of the stack (at the time before the new memory access is performed, so before moving the node to the top). Figure 5.13(b) shows an example distribution of this reuse distance. We can see that distance zero occurs very frequently, this means the same node is contacted twice or more in succession. Beyond that, the reuse distance drops

off sharply, meaning that there is indeed a high degree of temporal locality in the stream of home nodes that are being contacted. This is expected, a longer period during which the same home node remains at the top of the stack results in a communication burst between requesting and home nodes, as was observed before.

Distribution of owner and sharing nodes

Most traffic is exchanged between the requesting node and the home node (in `REQ` and `REPLY` packets). While memory accesses requiring invalidations can potentially lead to much more packets, they are also relatively infrequent. Therefore, in order to limit the complexity of our packet generator, we have decided not to model the destination of these write-back and invalidate packets. Only their number is modeled, the destinations will be generated using a uniform distribution. This way, the global network load will still be relatively accurate (the correct number of packets will be generated), only the (source, destination) distribution will be distorted slightly.

Time between requests

During the execution of a real application, a large fraction of time will (hopefully) be spent by the processors doing calculations. At certain instants, these calculations need data in external memory and a remote memory access is performed. An important parameter in this respect is the communication-to-computation ratio, which tells us whether the execution of a certain application is dominated by useful computation, versus waiting for remote memory accesses. If, in our synthetic traffic, we would only generate requests and not model the computation time, much more requests would be generated per unit of time, overloading the network and causing much more congestion than there would be in reality. Therefore, we measure the time between subsequent requests from the same node (actually, between the completion of one request and the start of the next one). In the context of communication networks, this time is also referred to as the *think time*, during which the processor or user thinks about what request will be made next. The distribution of this time, as measured during a simulation of the `fft` benchmark, is shown in Figure 5.13(c). When generating requests, each node will insert delays between subsequent requests to model this computation time. This way a realistic network load can be generated.

5.3.2 Generating synthetic traffic patterns

The measurements from the previous section provide us with a statistical profile about the memory accesses performed, containing the following information:

- the distribution of the number of involved nodes,
- the distribution of the reuse distance of home nodes from a certain requesting node, and
- the distribution of delays between launching new requests.

Our synthetic traffic generator takes these distributions and generates *scripts*, one for each node, which are executed by an entity that generates network traffic in subsequent simulations. These scripts will contain the type of packet group to be generated ((a), (b) or (c) in Figure 5.12), the identities of the home node and possible third-party nodes, and the delay that should be taken into account before launching the next request in the script.

The number of involved nodes and the delay are generated randomly, using a random number generator that matches the distributions given in the profile. For home node identifiers, a reuse distance is generated according to the distribution provided. This reuse distance is used to look up the home node on a stack that, for each requesting node, contains the last accessed home nodes. After generating each access, this home node is moved to the top of the stack. Identifiers for third-party nodes are generated uniformly.

To validate certain assumptions, we also included the possibility of writing a script that closely follows the memory accesses performed in the packet trace recorded from an execution-driven simulation. To this end, a packet group is generated for each memory operation, with the same home node, the same number of involved nodes and followed by the same delay. The locations of the third-party nodes are not maintained but are randomized using a uniform distribution. Also, retries and cache-initiated write-backs are removed from the trace. This enables testing the effects of the simplifications made.

5.3.3 Simulating the synthetic traffic flow

For simulations with synthetic packet traces, the same simulation platform is used as for doing execution-driven simulations. This guarantees that the network model used in both cases is identical, and reduces implementation work. The processors, caches and directory controllers are now disconnected, and a special packet generator is connected to each of the network

nodes instead. This packet generator creates request packets and injects them into the network, according to the scripts generated previously.

Each packet, injected into the network by the packet generators, contains a reference to the description of the packet group it belongs to. When the packet arrives at its destination, the packet generator at that node can therefore know what actions are required to continue generation of the complete packet group. These actions can be to send a reply packet after a certain amount of time (modeling the time required to look up a data word in main memory), or to send further packets (`WBreq` or `INVreq` requests), await arrival of their corresponding replies (`WBreply` or `INVreply`) and only then send the `REPLY` packet back. `WBreq` and `INVreq` packets contain the same reference so the third-party nodes know which home node to send their `WBreply` or `INVreply` to, again after some delay modeling the invalidation in or write-back from the third-party node's cache.

The packet generators thus perform two actions simultaneously and independently: generate new requests according to the script provided, and take part in the completion of requests of other nodes by receiving network packets and sending back replies. Each packet generator also measures the time that transpires between sending the `REQ` packet for a request and the arrival of the corresponding `REPLY`. This way remote memory access latencies can be measured.

The CPU time required for the generation and simulation of a synthetic trace for the 16-processor `fft` benchmark was less than 10 minutes, compared to 92 minutes for an execution-driven simulation, both for a simulated time of 90 million clock cycles. This is because, in an execution-driven simulation, most of the computational work is in the instruction set simulation of the processors and the simulation of memory accesses that hit in the level 1 and level 2 caches. Only memory accesses missing in the level 2 cache cause network traffic, so during a simulation based on traffic traces only those will need to be taken into account. In Section 5.3.5, we will show that the simulation time can be further reduced by using shorter synthetic traces, at only a slight expense of accuracy. Note that for each benchmark a single execution-driven simulation will still be necessary to compute the statistical traffic profile, but its cost can again be amortized over a large number of trace-based simulations.

5.3.4 Results

Figure 5.14 shows our simulation results for a number of networks when running the `fft` benchmark on a 16-processor machine. Two network properties are measured: the average distance a packet needs to travel (weighted

by its size, top graph), and the average packet latency (bottom graph). From left to right, five different networks are shown. In the first one only the base network is active. The next three are instances of our parametric reconfigurable network model with $f = 2$ and varying n and Δt. The final one, *prism*, refers to the implementation using the SOB element described in Section 4.2.

When reading the bars corresponding to (a) through (d), four measurements were done of both packet distance and latency for each network. The first one, *(a) execution-driven*, is a normal execution-driven simulation. This is the most accurate simulation we can do, but also the most time-consuming one. It will be used to determine the accuracy of the other steps. Results on the relative runtimes of different methods are provided in the following section.

In the next situation, *(b) trace from same network*, the memory operations from the first simulation are translated to packet groups, as described at the end of Section 5.3.2. The difference between (a) and (b) is due to our approximations when generating the packet traces. Possible reasons are ignoring `NAK`s and retries, ignoring packets related to cache eviction of modified blocks, and the randomization of third-party nodes. Also, the generation of requests by different nodes is no longer synchronized: each node executes its script at a pace dependent on the time the individual requests take. If initially there is a high correlation among the behavior of the different nodes, but during the course of the simulation some nodes are sped up or slowed down more than others, this correlation will slowly disappear.

Situation *(c) trace from base network* resembles (b), but here a trace is used that was extracted from a baseline simulation (i.e., a simulation without elinks). In this set of measurements, the change (or lack thereof) in network traffic behavior can be seen when the same application is executed on different networks. The difference is minor, which means we only need to do one execution-driven simulation (the baseline), and can use its traffic profile to evaluate a large number of other networks. Finally, in situation *(d) synthetic traffic* our synthetic traffic is used.

When comparing results between the different networks, we can see that, although the absolute value of the packet distance or latency cannot be predicted very accurately, their relative change – while changing network parameters – remains intact. Our synthetic traces can therefore be used to compare several properties of different network implementations.

From the systematic deviations in the absolute predictions, we can derive two conclusions about our traffic synthesis algorithm. The packet distance is too high, but only in situation (d). This means that the distribution of (source, destination)-pairs, which is mainly the result of the reuse distance distribution, is not accurate. The fact that owner and sharing nodes are

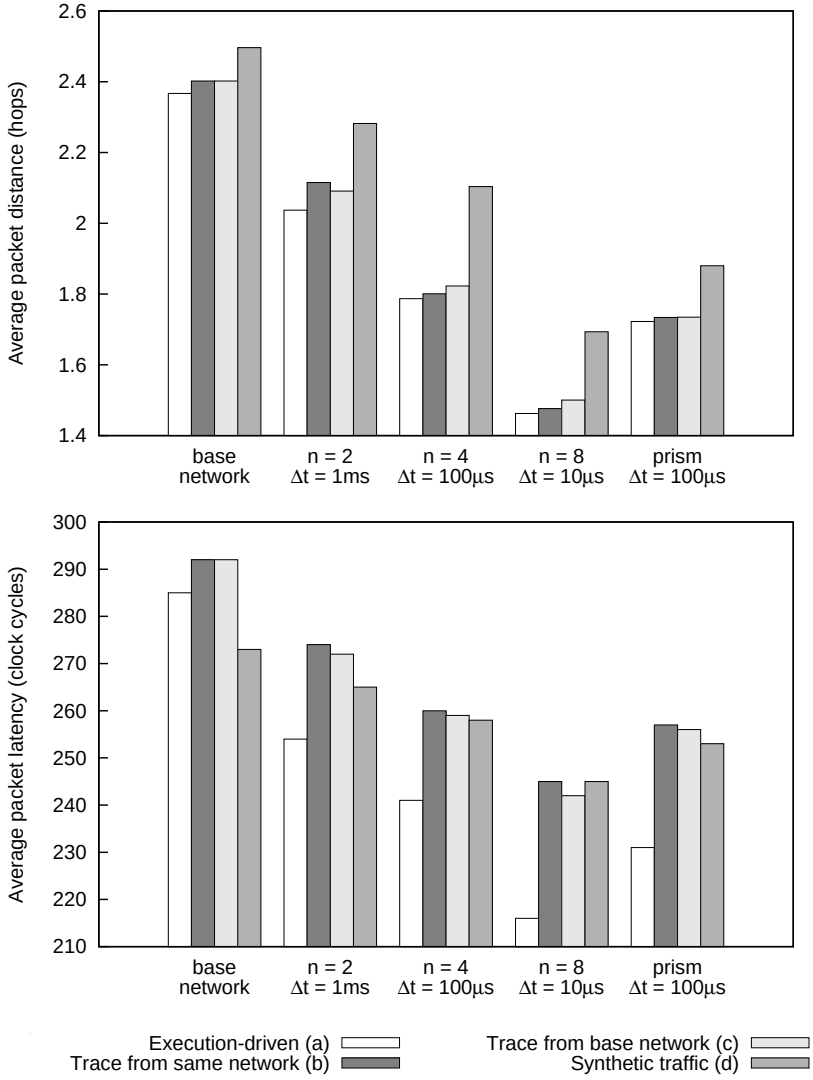

Figure 5.14: Average packet hop distance (top) and packet latency (bottom) for a number of reconfigurable 16-node networks and the `fft` benchmark, from execution driven simulation (a), up to synthetic traffic (d)

modeled as uniformly distributed is not the cause of this error, since this uniform distribution was also applied in situation (b), where the average packet distance does not yet show this error. Our stack-based algorithm should therefore be the first component to be upgraded if better modelation of the temporal traffic locality is required.

The second observation we can make is that the average packet latency is also estimated too high, but here the main error occurs between situations (a) and (b). As described above, a number of packet types, such as `NAK`s and retries, were omitted from the trace in situation (b), reducing the total network load somewhat. The average latency went *up*, however, suggesting that there is *more* congestion even though there are *less* packets. This means that the removal of these uncommon packet types did not cause the error. A different effect turned out to be the cause here: when generating packet groups, we always assumed that all `REPLY` packets contain a cache line. This is not the case in reality: up to 20% of the time, the `REQ` packet was an *upgrade*, meaning that the requesting node already had the cache line in its cache in a shared state, and requested an upgrade of this state, from shared to exclusive access, because it now wanted to write to the line. The node already had the data though, in this case the `REPLY` packet only needs to contain an acknowledgement. In situation (b), therefore, the average size of the `REPLY` packets is too large, causing a higher network load, resulting in too much congestion and an artificially high average packet latency. A solution would be to measure the fraction of upgrades, and model the size distribution of the `REPLY` packets accordingly.

5.3.5 Required trace length

True wisdom is to know what to leave out.
— David Patterson

The main reason to use synthetic traffic traces was the promise that they could be shorter than a complete execution trace, but still retain all relevant information. The question now is, how much shorter can they be, while still providing sufficient accuracy? This question is addressed in Figure 5.15.

For this figure, a large number of short traces is generated using the profile of the `fft` benchmark, and executed on a reconfigurable 16-node network with parameters $n = 4$, $f = 2$ and $\Delta t = 100$ µs. In each trace the average packet latency is computed. These measurements are then aggregated for all traces of the same length. Figure 5.15 shows average (center line), standard

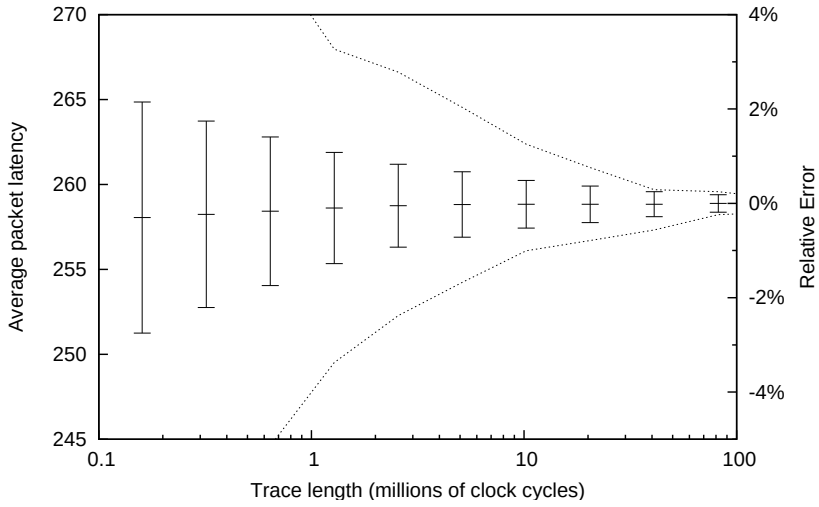

Figure 5.15: Accuracy of shorter synthetic packet traces. The average packet latency is measured in multiple traces, and grouped by trace length. Average, standard deviation, minimum and maximum (dashed lines) of the per trace averages are plotted. `fft` benchmark on a reconfigurable 16-node network.

deviation (error bars), minimum and maximum (dashed lines) statistics of the per trace averages, for each of the trace lengths considered.

One can now read from the graph the expected accuracy that shorter synthetic traces would be able to attain: the measurement of a single short trace executing in for instance 1.2M clock cycles could be anywhere between 249 and 268 (minimum and maximum values), and will be 258 on average with a standard deviation of 3.3 cycles. For longer runs this deviation diminishes, at the expense of an increase in execution time.

Table 5.1 compares these results with the required simulation time. From left to right, the table shows the length of the trace (in simulated clock cycles), the standard deviation of different measurements with the same trace length (in cycles), the difference between minimum and maximum measurements (cycles), and the CPU time required for one simulation of a trace of this length (seconds). The last column shows the total CPU time required, including the initial execution-driven simulation to measure the traffic profile, assuming this profile is re-used 100 times.

By comparison, the complete execution of the `fft` benchmark takes 89M cycles, and results in an average packet latency of 241 cycles. Since the errors in our trace-driven simulation are systematic – as evident from Figure 5.14(a)

Trace length	σ	|diff|	wall time	+profiling
160 k	6.80	50	2.5	64
640 k	4.37	29	9.8	71
3 M	2.44	14	39.2	100
10 M	1.40	6	157.7	218
41 M	0.73	2	627.2	688
execution-driven (89M)	2.48	7	7514.1	

Table 5.1: Comparison of variability and runtime between trace- and execution-driven simulations of the `fft` benchmark

and (d) – this difference in average packet latency (241 versus 258±3.3 for synthetic 1.2M-cycle trace) does not mean one should expect a high variability from the synthetic trace-based results. Comparison should instead be made with a trace-driven simulation using the complete trace (situation (c) in Figure 5.14), which also yielded a measurement of 258 cycles. This falls within the expected accuracy range for synthetic trace-based results.

Moreover, execution-driven simulation induces its own variability. The last line of Table 5.1 shows this: its stability (σ = 2.48 cycles) is only as good as that of a synthetic trace that is a factor of 30 shorter (3M clock cycles, with σ = 2.44 cycles). The required simulation time, however, is more than 100 times longer (over 2 hours versus less than 1 minute). More experiments on the variability of execution-driven simulations can be found in Section 6.1.

5.4 Comparison

A comparison of the different evaluation techniques presented is made in Table 5.2. This table shows the prediction technique based solely on home node distances from Section 5.1, the method accounting for congestion from Section 5.2, the synthetic traffic introduced in Section 5.3 and finally the most realistic method, execution-driven simulation.

The *measure* column shows which properties can be measured using the technique. Both prediction methods only allow packet or memory access latency to be measured. With synthetic traffic all properties related to the network – throughput and latency, possibility of deadlocks, buffer requirements – can be explored. Execution-driven simulation allows all system

Method	Measure	Congestion	Stable	Runtime
Distance	latency	no	yes	50s
Congestion	latency	yes	no	3m
Synthetic	all network	yes	no	10m
Exec-driven	all	yes	yes	2h

Table 5.2: Comparison of the different reconfigurable network evaluation techniques

properties to be measured. The next column shows whether congestion is considered, this is the case in all methods except the first one.

Column *stable* denotes whether the method yields a stable result when something is predicted that is already known, for instance the latency of a baseline network. The first method uses the distribution of home node distances, and the average latency per distance value, of a baseline simulation. When using this method to predict the average latency of the baseline, the method computes a weighted average of the per-distance latencies. This will, by definition, result in the correct average latency of the baseline. The *congestion* method, in contrast, is not able to accurately predict this baseline latency (as was evident from Figure 5.11), due to the insufficiency of the queueing model for highly loaded links. The non-stability of a prediction technique does not have to pose a serious problem, however, because it only pertains to *absolute* accuracy. Indeed, when comparing different network implementations, *relative* accuracy is much more important. The third technique, synthetic network traffic, also exhibits non-stability: the traffic profile, derived from an exact traffic trace, makes some simplifications – for instance, retries are ignored, third-party nodes are randomized – making the generated traffic stream differ from the original one.

Finally, the last column shows a typical runtime of all simulation methods, taken from a simulation of the `fft` benchmark on a 16-node SOB network. Note that for the first three techniques, one execution-driven simulation is needed to obtain the traffic profile, its execution time is not included here. This time can be amortized though over the evaluation of multiple network candidates. Clearly, simulation accuracy has to be traded for time. The correct solution is therefore not a single network evaluation technique, but rather a range of techniques with different trade-offs between speed and accuracy. For early design-space explorations, the first techniques can study the sensitivity of performance to the various parameters, and quickly select (ranges of) interesting design options. Later on, synthetic traffic can be used

to fine-tune the parameters. Execution-driven simulation can, in the last stage, characterize and verify the final design solution. All these techniques will be used in the next chapter.

6
Performance evaluation

The best performance improvement is the transition from the nonworking state to the working state.
— J. Osterhout

In this chapter, we will explore the performance improvement that can be obtained by adding a reconfigurable network to a Distributed Shared-Memory (DSM) machine. The main metric that is used here is the reduction in remote memory access latency, compared to a base network only implementation. First, we discuss the variability of this metric and compare it to other possible choices. Next, we characterize memory access time reductions for a number of different networks, both using our parametrized network model and the proposed implementation using the SOB device. Also, the sensitivity to the various parameters will be explored. For small networks (16 processors) we can do this using full-system simulation, using our simulation platform detailed in Section 4.4. The evaluation of larger networks (32 and 64 processors) is done with the synthetic network traffic introduced in Section 5.3. Finally, the impact of two simplifications made in our reconfiguration algorithm is investigated: the heuristic elink placement algorithm is compared to the optimal placement, and the impact of basing the elink placement on traffic from the previous interval, rather than the current one, is investigated.

6.1 Variability of performance metrics

Statistics are like a bikini. What they reveal is suggestive, but what they conceal is vital.
— Aaron Levenstein

Statistics: The only science that enables different experts using the same figures to draw different conclusions.
— Evan Esar

When designing a reconfigurable network, a useful trade-off has to be found among several parameters, such as power dissipation, system performance, design and fabrication costs, etc. Most of these are very dependent on the technology that is being used, so in the context of this work we cannot predict the extra cost or power dissipation when, for instance, increasing the fan-out of the network. What we can do is look at system performance. Indeed, our simulation platform allows us to answer questions such as "how much performance is gained by implementing a larger fan-out". To judge whether the added cost and power are justified by the increase in performance, will be up to the network designer.

Several metrics can be used to study system performance. High-level metrics look at application performance. If the machine is used as a web server, one can distinguish between throughput (number of requests handled per second) or latency (average, worst case or 97% percentile of the completion time of requests). Other, low-level metrics look at a more detailed level inside the machine, such as remote memory access time or packet latency.

High level metrics are usually more useful to an end user because they relate directly to something the user of the machine can see, such as the runtime of their application or the machine's performance as a web or database server. However, those high-level metrics are influenced by a great deal more factors than the low level metrics. For instance, several of the SPLASH-2 benchmarks used in this work use *load balancing*. This means that, if one processor is executing its workload more slowly, some of its work will be distributed to other processors, resulting in a potentially dramatic shift in

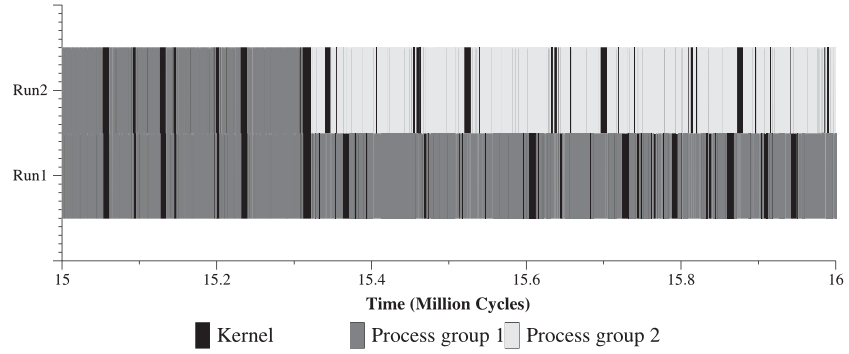

Figure 6.1: OS scheduling decisions are affected by memory latency [Alameldeen et al., 2002]

high-level behavior of the application. This can happen even if the parameters of the multiprocessor machine, or the influence of other applications running on the same machine, are changed even slightly. Synchronization can also affect the program's execution. Assume two processors want to acquire a lock at nearly the same time. If processor A does this one cycle ahead of processor B, it can continue, while B has to wait. A delay in the execution speed of processor A by just a few clock cycles, caused for instance by extra congestion in the network, can reverse this situation, so A now has to wait. Usually, other processors are dependent on the work done by A, so this change will propagate throughout the execution. All other situations where a choice is made that will later affect the program's execution, such as which thread to schedule next, which virtual memory page to replace on a page fault, etc., have the same effect.

Alameldeen et al. [2002] investigate this problem of non-deterministic workload behavior further. They simulate the execution of an On-Line Transaction Processing (OLTP) application and introduce artificial cache misses every 100 accesses. This experiment is repeated twice, with just one change: the misses are introduced at accesses 0, 100, etc., for the first simulation, and at accesses 50, 150, etc., for the second one. Although the average cache hit rate is exactly the same in both cases, the effects of load balancing, scheduling and synchronization as described before, cause the execution time to deviate by up to 9% between both simulations. Figure 6.1 shows a trace of the scheduling decisions made in both simulation runs. One can see that the alteration between kernel mode (black) and user mode (gray) is different, and that at a time of about 15.3M cycles the execution diverges completely when a different process is scheduled during the second run.

Changing the network topology clearly influences packet latency and thus remote memory access times. It therefore changes the speed with which processors execute their work, and affects the outcome of synchronization races, scheduler decisions, etc., which can change the amount of work done. Therefore, when comparing simulation results of two different networks, it is not correct to assume that the same work is being done in both simulations.

To characterize these effects, and to investigate which performance metrics are least influenced, and therefore most usable to compare different network implementations, the following experiment was done. For each benchmark, two simulations are performed: one with a base network only and one with a reconfigurable network (a 16-node network characterized by $n = 16$, $f = 2$, $\Delta t = 1$ ms was used). The improvement after adding reconfiguration is calculated for several performance metrics. This experiment is repeated five times. Each time the scheduler is in a different state at the start of the benchmark, this way variability is introduced in the measurement.[1]

The average and standard deviation of these five measurements of performance improvement are shown in Figure 6.2. Program runtime clearly has some problems when used as a performance metric. For some benchmarks (those that have little communication), the runtime barely changes or even degrades when adding reconfiguration. The number of instructions executed is not really a network performance metric, but it does allow one to gain insight into the reason of the variability (the instruction count of first processor in the system is shown). Usually, if the amount of work does not change, one would expect the number of executed instructions to remain the same – i.e., the improvement should be zero. Especially for the `radiosity` benchmark, this is not the case: this is one of the benchmarks where load balancing is used, which clearly shows its influence in this experiment.

The low level metrics have much less variability. Their disadvantage is that the relationship between a low level metric and application-level performance isn't always obvious, but they do allow one to make more reliable statements about the performance of the network, without being influenced by the peculiarities of other system components. They also enable a designer to gain insight into what is happening in the system, and can suggest areas where improvement is necessary. Note that, although much of the influence of the benchmark is now removed from the metric, the difference in benchmark behavior still has a clear impact on network performance, through mechanisms such as communication locality or burstiness.

[1] Our simulator is completely deterministic: executing a simulation with the same parameters twice, will gave the same results. Therefore we delay the start of the simulation by executing a `sleep` command on the UNIX command line of the simulated machine, before starting the benchmark, to change the state of the scheduler.

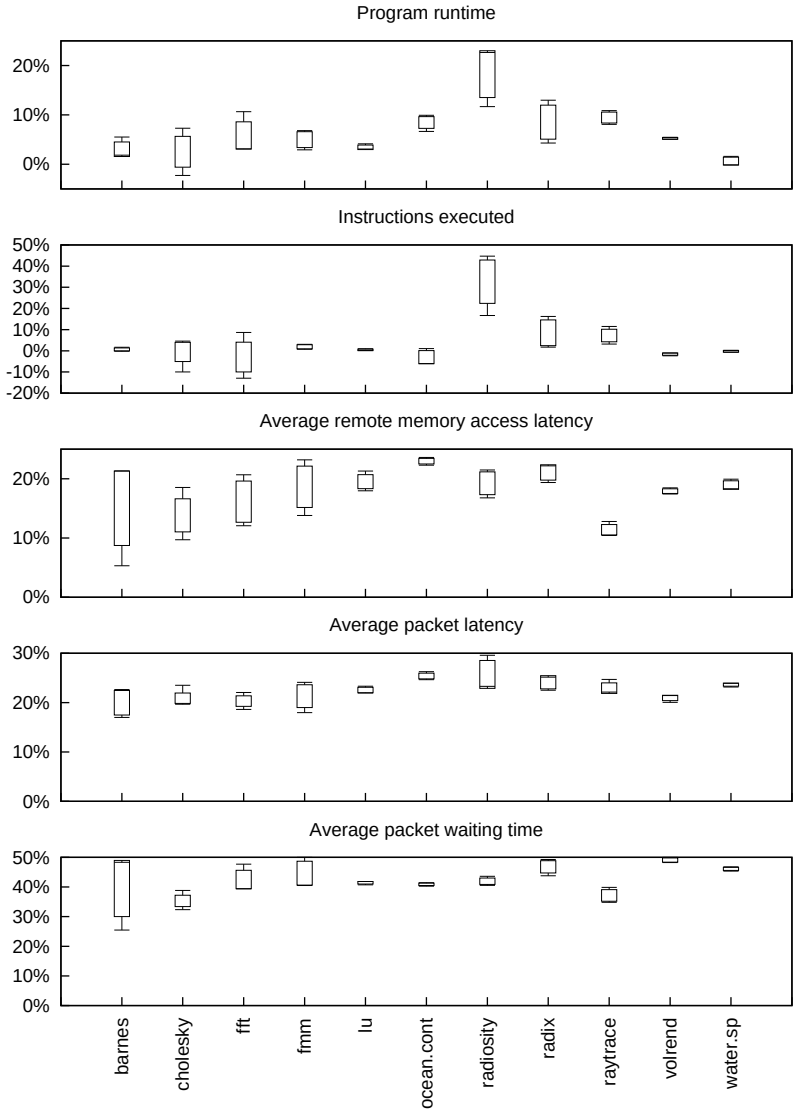

Figure 6.2: Variability of possible network performance metric improvements ($\mu \pm \sigma$, minimum and maximum values) when the improvement of each metric, after adding a reconfigurable network, is measured five times

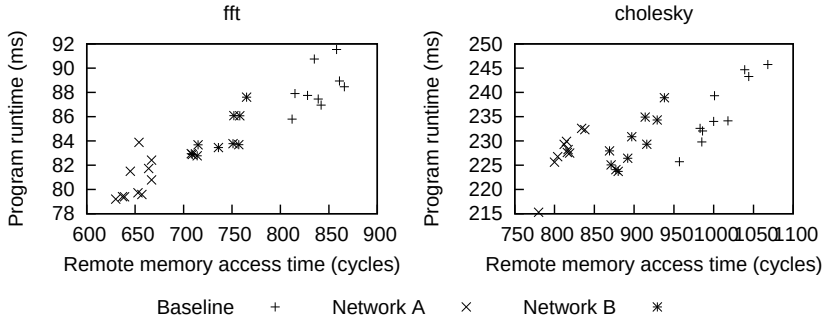

Figure 6.3: A network is only statistically significantly better if its improvement of a performance metric is higher than the variability of that metric. All for a 16-node network, baseline: no reconfiguration; network A: $n = 16$, $f = 4$, $\Delta t = 100$ µs; network B: $n = 16$, $f = 1$, $\Delta t = 1$ ms.

Another way to look at variability is shown in Figure 6.3. For three types of networks, the simulation of the `fft` and `cholesky` benchmarks is repeated ten times. The metrics "program runtime" and "average remote memory access time" are plotted against each other for each simulation. Before we can conclude that network A is better than network B, a *statistically significant* improvement of the relevant performance metric must be visible. The variability of the program runtime – about 5 ms for `fft` – is just as large as its average improvement between the different networks. By measuring the program runtime improvement just once per network, we can therefore not conclude that network B is better than network A, because the improvement – if any – visible between one pair of simulations can just as well be attributed to noise. When using the average remote memory access time as performance metric, the improvements are larger than the average absolute value of the noise, so in this case we *can* conclude that B is better than A.

Comparing these results with Table 5.1, which showed the variability of synthetic trace-driven simulations, it is clear that execution-driven simulation, due to the interactions between network behavior and the software, has a much higher variability than the trace-based approach. One can argue that, in a real machine, these interactions are also present and therefore must be modeled by the simulation tool. In practice though, their effect is very unpredictable and effectively obscures much of what we might learn about network behavior. Therefore, it is our opinion that research aiming to understand interconnect behavior should make abstraction of the interactions with the software (i.e., use a trace-based or similar technique) to gain clear

insight into what plays at the interconnect level. The fact that the performance figures obtained in this way will, in practice, be only average values due to software interactions, must of course always be kept in mind. A more thorough evaluation of this problem, which is also relevant to other microarchitectural research on multiprocessor and multicore architectures, will be presented in an upcoming publication [Heirman et al., 2008c].

6.2 Small networks: execution-driven simulation

6.2.1 Selective broadcast implementation

We identify how the properties of a network implementation using the selective broadcast (SOB) component affects high-level network parameters, and translate these properties into limitations that are imposed on our network model. Three limitations are identified. Component count and cost limit the fan-out of our network. The tuning range of the transmitters, combined with the need for efficient use of transmitted power, forces us to partition the network using the SOB device, this restricts the topology of the reconfigurable part of the network. Finally, the tuning speed of the VCSELs affects the reconfiguration speed. For each of these limitations a number of simulations are done with different parameters. This way, we can determine the effect on the performance of the complete machine for each of the limitations separately, and determine which of them are the most likely candidates for future improvement.

The SOB-based implementation can be described using the parametric network model from Section 4.1. We fix the network size at 16 processors or nodes ($p = 16$). With one tunable laser per node, the number of unidirectional elinks is fixed at $n = 16$. We gradually add the additional constraints set by the implementation: a limited fan-out $f = 1$, and the restriction on the node pairs as prescribed by the placement matrix of the SOB (only 9 out of 16 possible destinations). Finally, we sweep over a range of reconfiguration intervals. The selection time t_{Se} and the switching time t_{Sw} are both set to 10% of the reconfiguration interval Δt. Note that, in practice, t_{Sw} is determined by the tuning speed of VCSELs used, so the reconfiguration interval Δt should be chosen to be for instance $10\, t_{Sw}$.

Figure 6.4 summarizes the simulation results. In Figure 6.4(a), we show the average remote memory access latency for a selection of SPLASH-2

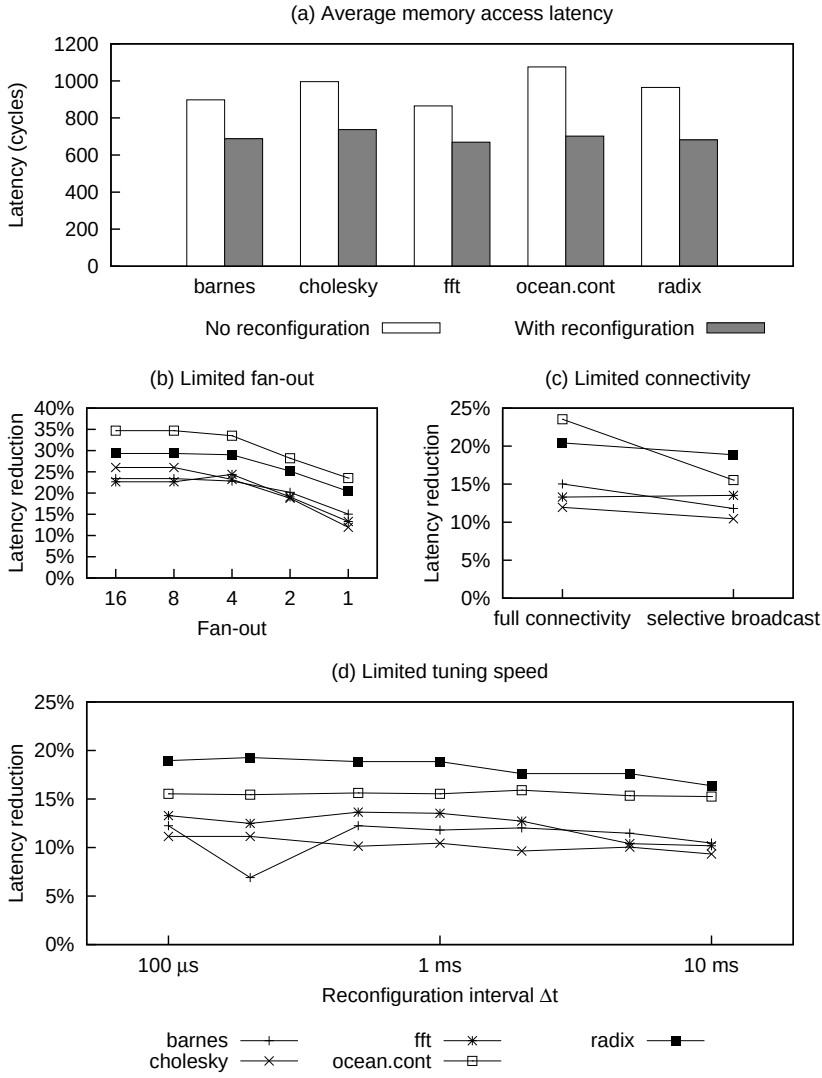

Figure 6.4: Results for a 16-node network: from an idealized $n = 16$, $\Delta t = 1$ ms network (a) to the SOB-based implementation with different reconfiguration intervals (d)

benchmarks, first using only the 4×4 torus (base) network, next when a reconfigurable network is added. This is done using the idealized network model as described in Section 4.1, with 16 unidirectional elinks, a reconfiguration interval of 1 ms and no constraints on connectivity or fan-out. The access latency is significantly reduced, from 960 clock cycles to only 696 cycles (averaged over the five benchmarks used), this is a reduction of 27%. The next graphs show what remains of this improvement, once we transform our idealized network model into something that can be physically implemented by introducing the constraints imposed by the architecture.

First, we account for the fact that only one reconfigurable transmitter and one wavelength-selective receiver are available per node. This means that only one elink can terminate at each node. Figure 6.4(b) shows the reduction of memory access latency over the baseline (using only the base network), when we gradually reduce the maximum fan-out from $f = 16$ (there are only 16 elinks, so this effectively means no fan-out limitation) to just one elink per node. The reconfiguration interval is fixed at 1 ms. Usually, network traffic is localized, which means most nodes only talk to a limited set of other nodes. This is visible in the graph: we can reduce the fan-out to four almost without sacrificing performance. After that, the network is unable to provide speedup for all of the important memory accesses, and latency increases again (latency reduction goes down). Still, half to two-thirds of the reduction is maintained after limiting fan-out to one. By placing two or more tunable lasers and receivers per node, one could implement a network with higher performance. This would of course drive up the cost. According to Figure 6.4(b), using more than four would not provide additional gain.

Next, we introduce the properties of the SOB device into our simulations. This limits the choice of possible node pairs that can be connected directly using an elink from $p \cdot (p - 1)$ to just $9 \cdot p$. If two nodes communicating heavily are not in the list of $9 \cdot p$ possible node pairs, there cannot be a direct link between them so we expect performance will suffer from this restricted connectivity. As Figure 6.4(c) shows, this is only the case to a limited extent. Indeed, even though a node pair cannot have a direct connection, it may still be possible to offer a shorter path using one elink and one base network link, compared to up to four base network links when no elinks are available. Also, connecting two nodes with slightly less traffic between them with an elink can free up part of the base network for use by other heavy traffic streams. This makes that a reconfigurable network implementation using the SOB device can still provide a significant performance improvement, while being scalable to a large number of nodes.

Finally, we look at how the reconfiguration interval affects performance. Since traffic is expected to change over time, we would like to reconfigure

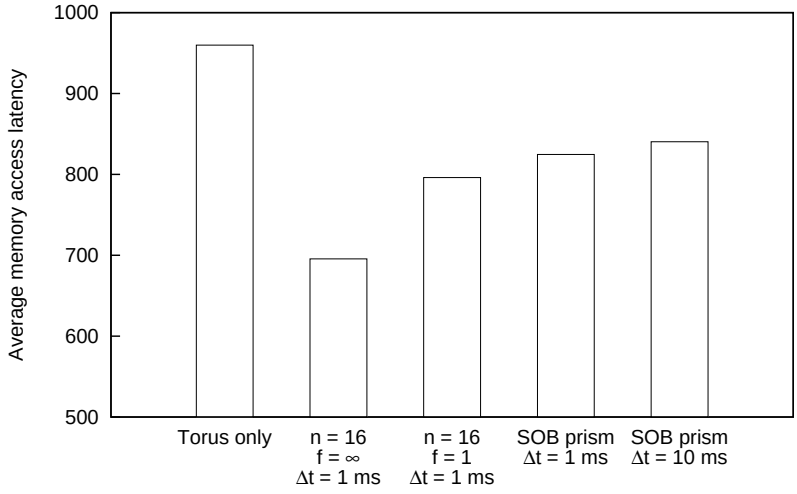

Figure 6.5: Summary of memory access latency, averaged over all benchmarks

the network as often as possible. Remember though that the reconfiguration interval is set to ten times the switching time of the VCSELs, setting a shorter interval without using faster VCSELs would mean that the elinks are unavailable for more than 10% of the time, limiting their usefulness. Figure 6.4(d) shows the latency reduction for a number of reconfiguration intervals, using the reconfigurable network implementation with the SOB device. If VCSELs with a tuning speed of 1 ms are available, the reconfiguration interval would be 10 ms which still provides a good speedup for most of the benchmark applications. Note that the applications differ most in how fast their traffic patterns change, so here the results are more differentiated among the different benchmarks compared to the previous graphs.

Figure 6.5 summarizes these results, plotting the average memory access latency, averaged over all benchmarks. Overall, we can conclude that limiting fan-out to one and using selective instead of full broadcast are necessary to allow the network to be implemented at a reasonable cost, but they reduce the access latency improvement to about half of what we could achieve if these limitations were not present. With a reconfiguration interval of up to 10 ms, requiring a VCSEL tuning time of about 1 ms, no additional performance is sacrificed.

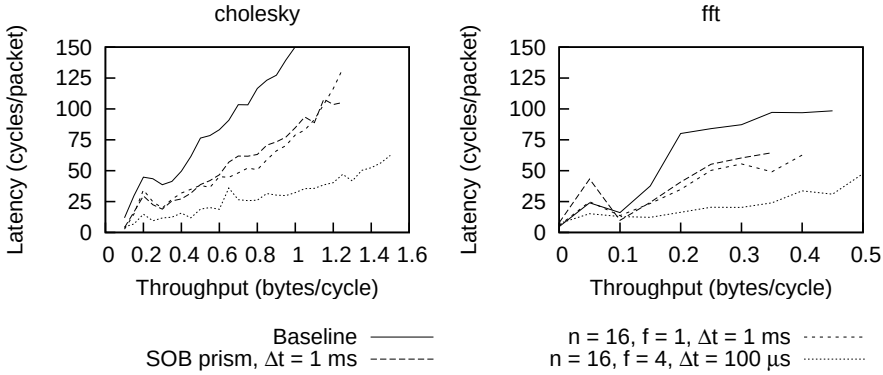

Figure 6.6: Latency versus throughput, for a variety of benchmarks and 16-node networks

6.2.2 Latency versus throughput

Figure 6.6 shows a classic network evaluation metric: the latency versus throughput curve. Here, the average delay packets experience on their way from source to destination (only waiting times are considered here, not the transmission time) is shown as a function of the network load (the number of bytes that are transmitted per clock cycle throughout the network, computed in intervals of 100 µs). We have shown this relation for two benchmarks being executed on a torus-only network and three different reconfigurable networks, all with 16 nodes.

Generally, more traffic on the network (higher throughput) causes more congestion and thus higher latency. Networks with more available bandwidth (all three reconfigurable networks shown have 25% more bandwidth than the baseline) should have less congestion, this is visible on both graphs. A network that is able to better provide bandwidth where it is needed, through higher fan-out or faster reconfiguration, can again reduce congestion and provide lower latency – this explains the difference between the $\Delta t = 1$ ms networks and the $\Delta t = 100$ µs one. The difference between the SOB prism network and the more general $n = 16$, $f = 1$ network is minor, showing that the selective broadcast does not restrict performance too much.

Also note that on the baseline network, the same high values of throughput as the reconfigurable networks are not reached: since the processors are waiting for response packets to come back, which are being delayed, the processors themselves execute slower. This reduces the rate at which they can generate new requests, and thus limits throughput.

Finally, we can see clear differences between the `cholesky` and `fft` benchmarks. `fft` requires less bandwidth, because communication is spread out more in time, while `cholesky` has a bursty behavior resulting in periods with high throughput. Note that one throughput value may yield a different latency for both benchmarks, because the (source, destination) distributions may be different. `fft` usually has a higher latency for the same throughput, showing that during the execution of `fft`, the traffic is more localized to *hot-spots*: small parts of the network with a disproportionally high load, compared to the average load, which results in longer waiting times.

Especially during high-throughput phases, the reconfigurable networks can provide a significant reduction in congestion, as is evident from the graphs. In measurements of the total packet latency, such as those used in the rest of this work, this reduction is less spectacular because the significant reduction during periods with high-intensity traffic is averaged out with a much more modest latency reduction during the rest of the benchmark's execution. Also, measuring total latency, rather than just the waiting time, adds latency components (transmission time, time of flight and routing time) that are not influenced by congestion – again reducing the relative improvement visible. On the other hand, the elinks can cause a reduction of the inter-node distance. This reduces the routing time and thus also total packet latency in a different way than through the waiting time shown in Figure 6.6.

6.3 Large networks: synthetic traffic

6.3.1 Scaling the reconfigurable architecture

Because full-system simulations are very slow (up to several hours each), it is not feasible to do large-scale design-space explorations with them. Our techniques from Chapter 5 do allow this. In this section, we use the synthetic network traffic generator, presented in Section 5.3, to explore reconfigurable network performance for networks of up to 64 nodes. Throughout this section, the improvement of the average network latency, relative to a base network only implementation, is used as the performance metric.

In Figure 6.7 the effect of adding more elinks is shown. The fan-out is always $f = 4$, the reconfiguration interval Δt is fixed at 100 μs. As expected, adding more elinks increases performance. An unlimited amount of elinks is of course not technologically feasible. Further exploration will therefore be done with as many elinks as there are CPUs ($n = p$). This is also the case for our implementation with the SOB device. Also visible is a performance

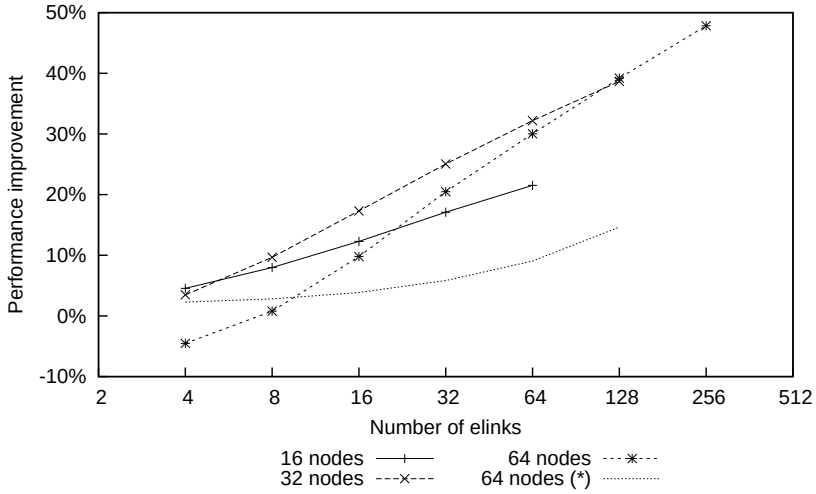

Figure 6.7: Performance scaling for increasing the number of elinks, with $f = 4$ and $\Delta t = 100$ µs, averaged over all benchmarks. The 64 nodes (*) case limits the use of each elink to a single node pair.

degradation compared to the baseline (negative improvement) in the 64-node case with only four elinks. Here, too many nodes want to use the few elinks causing, highly increased congestion. This has a large negative impact on performance. The 64 nodes (*) case avoids this extra congestion by allowing each elink to be used only by a single node pair. There, no more performance degradation is observed. From 16 elinks upwards the congestion problem is no longer present, the extra routing limitation now degrades network performance compared to the original situation.

Next, we study the influence of the fan-out. If one node communicates frequently with a large set of other nodes, it would be interesting to connect multiple elinks to this first node. This is not technologically feasible, indeed, in our prism implementation the fan-out is just one. Figure 6.8 shows the effect of different fan-out limitations, all for $n = p$ and $\Delta t = 100$ µs. As can be expected, improving the fan-out from one to two increases performance. Higher fan-outs however do not result in more performance gains. This last observation is not in agreement with Figure 6.4(b), however, which showed a more significant improvement for networks with higher fan-out. Since this graph was measured using execution-driven simulation it is probably more accurate than Figure 6.8, in which synthetic network traffic was used. This shows that, although simulations based on synthetic traffic are able to

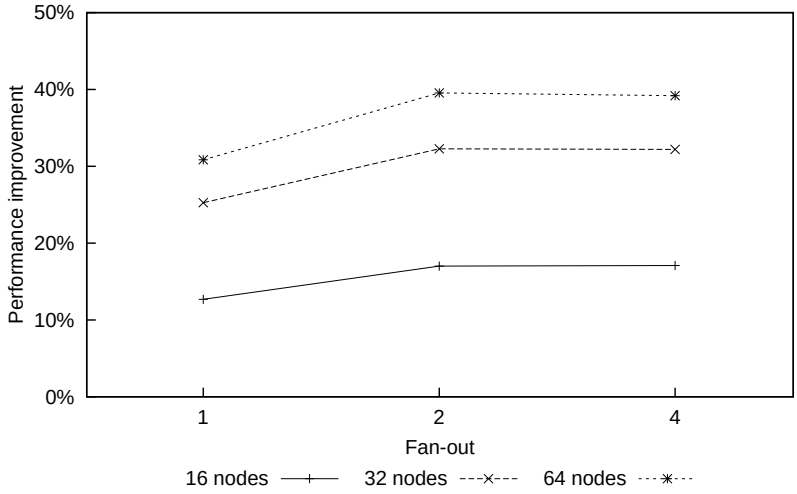

Figure 6.8: Performance scaling for increasing the fan-out, with $n = p$ and $\Delta t = 100$ µs, averaged over all benchmarks

provide quick first-order approximations, for detailed measurements one still has to revert to more detailed (and slower) methods of simulation.

Figure 6.9 shows the effect of faster reconfiguration. Here we vary Δt between 1 µs and 10 ms. The left side uses an idealized network model with $n = p$ and $f = 2$. On the right side, results for the SOB implementation are shown, which can be modeled as $n = p$, $f = 1$ and an extra limitation on which nodes are reachable from each processor. The different benchmarks are plotted separately in these graphs, showing that the behavior of the program being executed on the multiprocessor machine highly influences the performance improvement obtained.

The limited connectivity of the SOB network is represented by a smaller improvement in message latency. This is especially visible for very fast reconfiguration, where the networks with higher connectivity are able to better follow rapid changes in the application's traffic pattern. Remember though that the switching time, required from a component implementing these reconfigurable networks, needs to be about ten times faster than the reconfiguration times shown. The left side of the graphs, showing $\Delta t = 1$ µs – requiring 100 ns switching times – are therefore shown more as theoretical results, rather than improvements that can be achieved in practice, at least with the current generation of optical components.

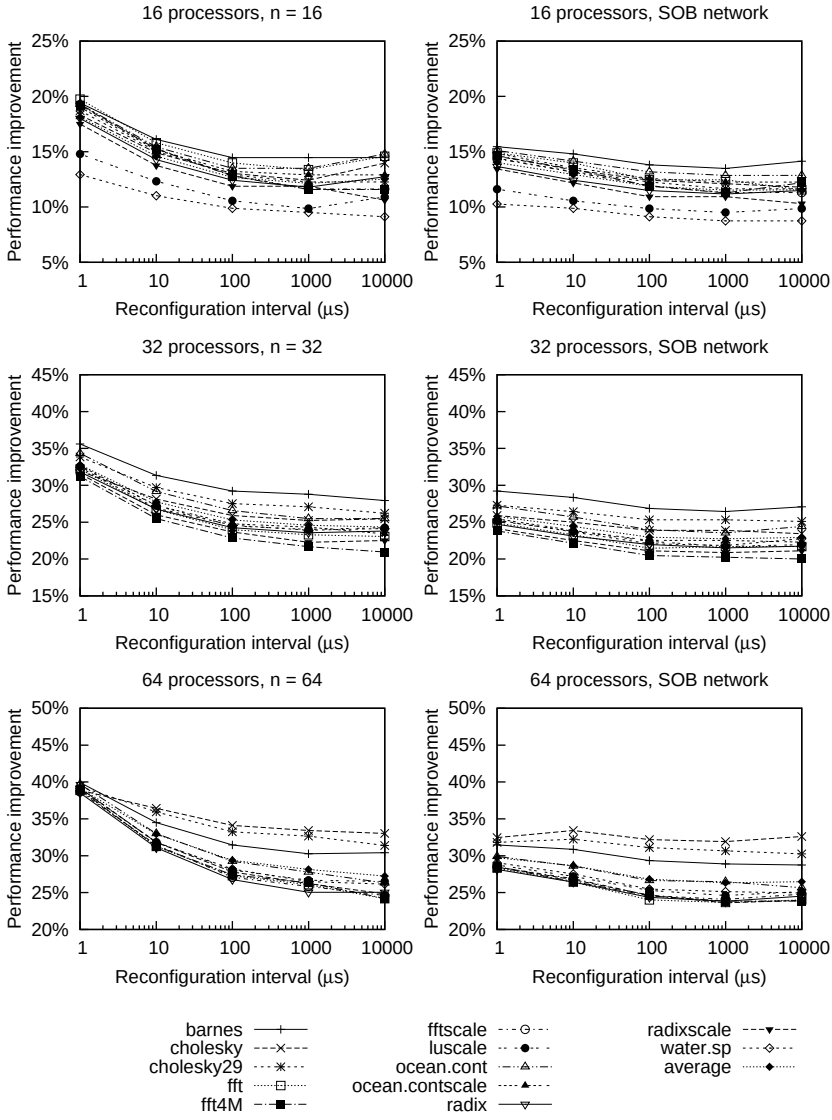

Figure 6.9: Performance trends for varying reconfiguration intervals, for an idealized network with $n = p$, $f = 2$ (left) and the SOB implementation (right)

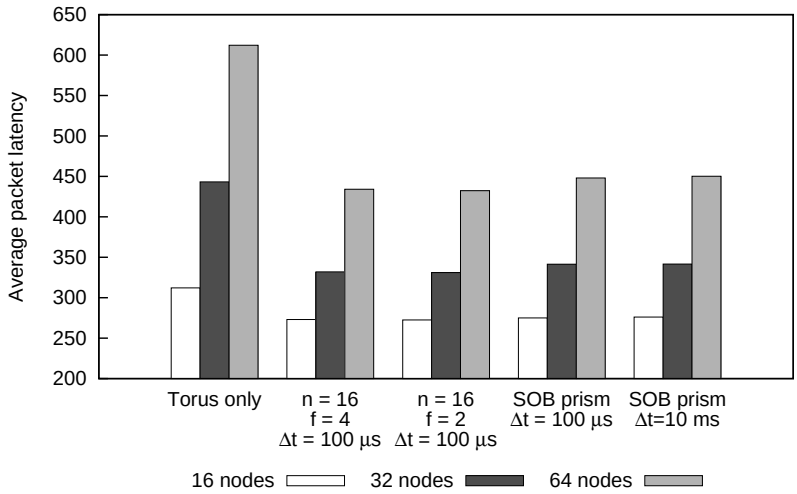

Figure 6.10: Summary of the average packet latency for different network types, averaged over all benchmarks

In Figure 6.10 a summary is shown of the average packet latency for different network types. First, a non-reconfigurable torus-only network is shown. The second set of bars plots the latency for an idealized, $n = p$, $f = 4$, $\Delta t = 100$ µs network. Next, the fan-out is restricted to $f = 2$, then the selective broadcast component is introduced. Finally, the reconfiguration interval is lengthened to 10 ms. The average latency can be seen to drop when introducing a reconfigurable network. Some of this gain has to be relinquished when implementation constraints (fan-out limitation, selective broadcast, slower tuning) are introduced, but even then a very visible performance improvement can be obtained.

6.3.2 Non-reconfigurable networks

Adding extra links to the basic torus network improves network performance for three main reasons:

- more bandwidth is available which reduces congestion,
- the average distance is reduced, and
- reconfiguration adapts the network to current demands.

All performance improvements observed up to now have been the result of a combination of all these mechanisms. However, only the last one is due to the network being reconfigurable. We can achieve a higher available bandwidth and reduce the distance without costly reconfigurable components. Therefore, it would be interesting to know how much of the observed speedups is actually due to reconfiguration, and how much of it can be achieved by simpler network adaptations.

Figure 6.11 compares the improvement in average packet latency for three types of networks, all with the same total bandwidth. First a reconfigurable network is shown, with $n = p$, $f = 2$ and varying reconfiguration intervals. Since a torus has $4 \cdot p$ (unidirectional links), the n elinks represent an additional total bandwidth of 25%. The *global* measurements in the figure show a torus network in which each link has a bandwidth that is 25% higher than the link bandwidth in the previous situation, the total network bandwidth is therefore the same as in the $n = p$ reconfigurable case. Finally, in the *random* situation n non-reconfigurable links are added to the torus topology at random positions, favoring longer links: a link's probability is proportional to the distance it spans on the base network. The minimum, average and maximum improvements are shown for five different random placements. Note that the maximum improvement in this case is obtained with different placements for each benchmark. Therefore, in an actual implementation with just one hardwired link placement, this maximum will only be achieved on some benchmarks – making the *average* measurement a more realistic representation of the improvement that should be expected in practice.

For small networks with only 16 nodes, keeping the torus topology but increasing the bandwidth of all links is the best solution – assuming this is allowed by the technology that is used. This solution avoids the costs of reconfiguration altogether, and has a slightly higher performance than a reconfigurable network with the same total bandwidth. This is because the distances in a 4×4 torus network are always small; elinks, reconfigurable or not, cannot reduce this distance much. For larger networks with 32 and 64 nodes, the maximum distance through the torus network is a lot higher. Here, elinks *can* provide a much shorter path and a corresponding reduction in packet latency.

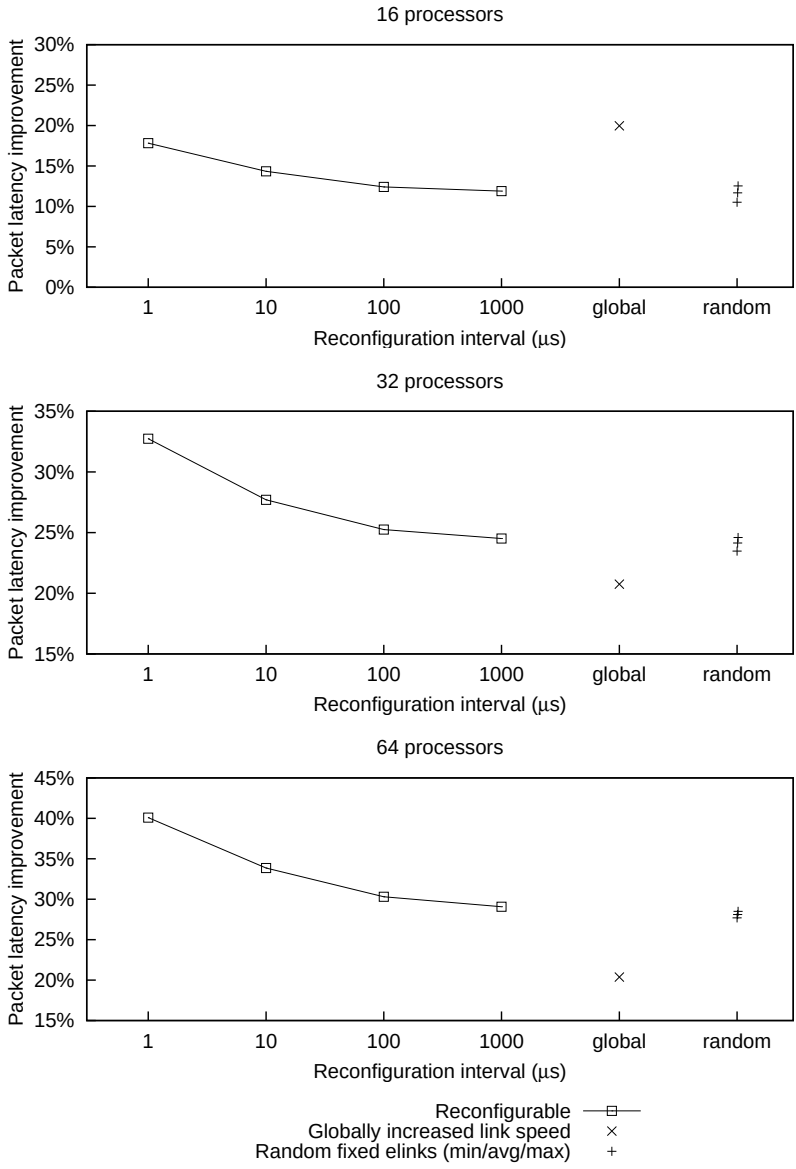

Figure 6.11: Comparison between reconfigurable and non-reconfigurable networks, with equal total bandwidth

6.4 Effect of reconfiguration heuristics

6.4.1 Optimal elinks placement

Section 4.3 introduced the algorithm that selects which elinks to activate, based on expected network traffic. Since this typically needs to be done in less than about 1 ms, a greedy heuristic is used to minimize the cost function:

$$C = \sum_{i<j} d(i, j) \cdot T(i, j)$$

Using a branch and bound method, it proved possible to, for each interval, determine the elink placement that results in the global minimum of C. This takes several minutes to execute for each traffic pattern, which underlines the need for a fast heuristic. In a simulated environment this does not matter of course, so we ran some simulations to see the difference in network performance between this optimal selector and our heuristic. Figure 6.12 shows the results for three reconfiguration intervals and three network implementations: two in which two or four elinks can be placed freely (with an imposed fan-out limit of two) and the *prism* scenario which uses the implementation with the prism from Section 4.2. In a few cases, the optimal elink placement results in a *slower* network than the pseudo-optimal placement from our heuristic. This is because the elink placement is only optimal for the traffic pattern from the *previous* interval, the traffic in the *current* interval may have shifted such that the heuristic selector now gives a better result. In most cases, however, the result is as expected, with the network using the optimal selector being just a few percent faster. This means our heuristic does a good job and only slightly affects network performance.

6.4.2 Perfect traffic prediction

> *Prediction is difficult,*
> *especially of the future.*
> **— Niels Bohr**

In Section 4.1, it was noted that, to place the elinks, an estimate is needed of what traffic is to be expected during the reconfiguration interval that is about to start (the *next* interval). Our implementation assumes this traffic

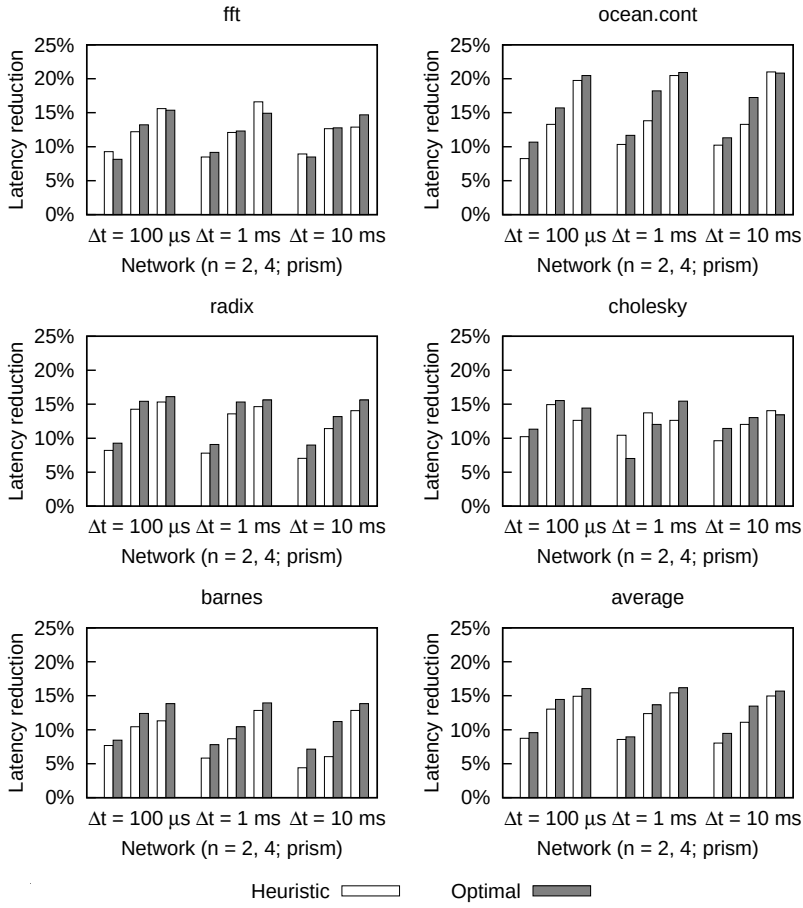

Figure 6.12: Comparison of the heuristic elink selection algorithm and the globally optimal elink placement

will be equal to that measured during the *previous* interval. We can now ask the question if this assumption is valid, and, if this turns out not to be the case, what increased performance can be expected of a system where traffic can be predicted more accurately.

To answer this question using normal simulation we would have to develop such a traffic predictor first, which is not trivial. On the other hand, with our performance prediction method from Section 5.1, an upper bound can be determined for the performance of a system with an ideal traffic predictor: since, while running the performance prediction, the execution-driven simulation yielding all network traffic has completed, all traffic is known, including that of the next interval. So in the prediction model, at the point where the elink placement for a certain interval is determined, we can use the traffic for the next interval instead of that for the previous interval. This mimics the behavior of a system with perfect traffic prediction.

The results of this change are shown in Figure 6.13, which shows the predicted improvements in memory access latency. As expected, the *next* case, where elinks are placed at locations ideal for the next traffic, performs consistently better than the *previous* case which uses the realistic placement based on past traffic. The difference is not dramatic though, which shows that our assumption that traffic does not change significantly between reconfiguration intervals holds.

Figure 6.14 plots the cost – in relative increase of the average memory access latency – of a suboptimal traffic prediction for each benchmark, on a 16-node network with $n = 8$, $f = 2$ and varying Δt. For longer reconfiguration intervals, the cost is usually less, because in that case the placement is based on a less specialized traffic profile. Here, a lot of the high-speed dynamics have already been averaged out, so the traffic patterns in the previous and next intervals are much more alike. `fft` and `cholesky` are notable exceptions, they have slow but important dynamics that cannot be exploited as good by a simple prediction method. Still, the cost is always less than 5% (of the baseline memory access time). Since the *next* case represents the upper limit for all traffic prediction methods, this means that even a very sophisticated and expensive traffic prediction algorithm can only result in a very limited performance increase.

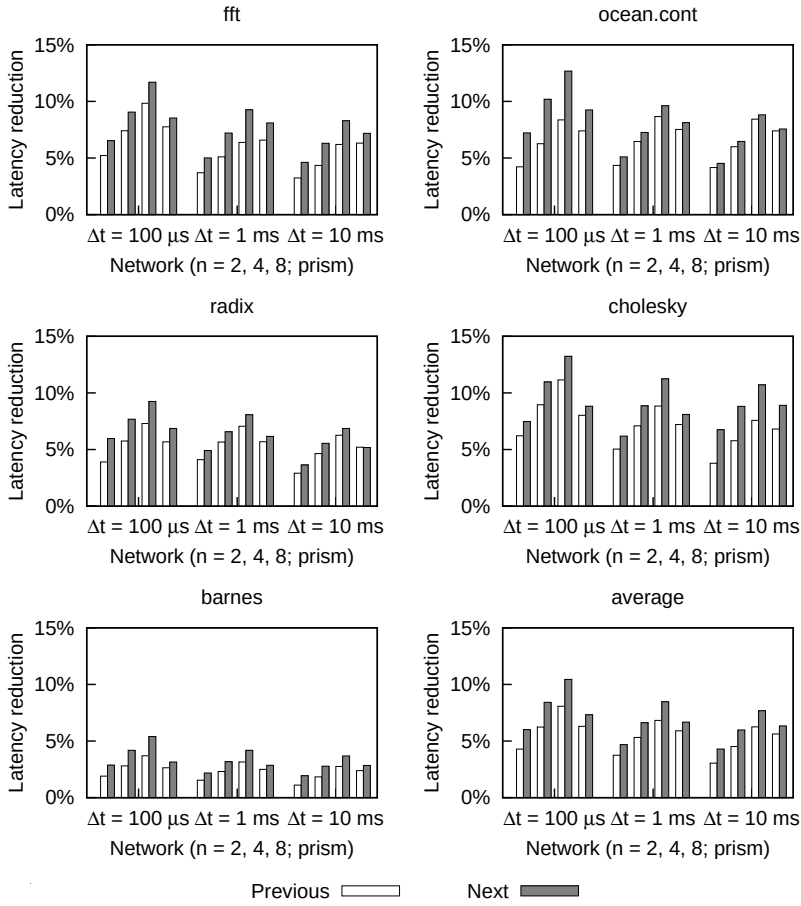

Figure 6.13: Comparison of placing elinks based on traffic in the *previous* or the *next* interval

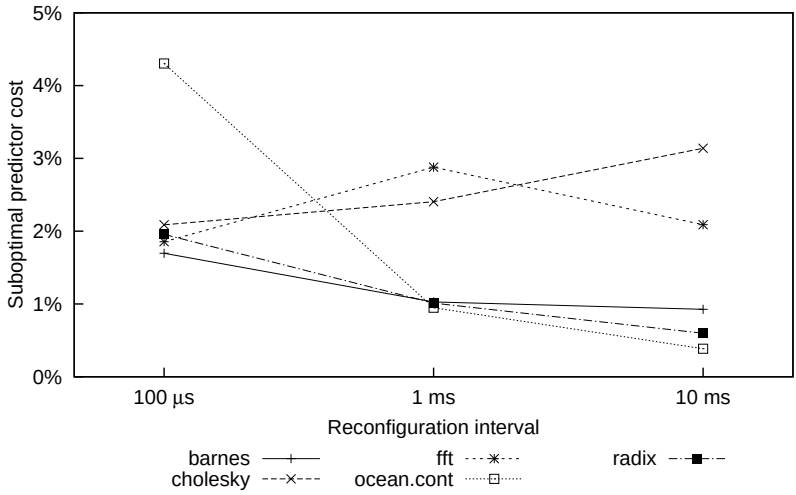

Figure 6.14: Cost of suboptimal traffic prediction

7 Conclusions

L'imagination est la seule arme
dans la guerre contre la réalité.
— Jules de Gaultier

In this final chapter, we will look back at what was accomplished in this work, and look forward at what may lay ahead. Section 7.1 provides a summary of the results obtained in this work. In Section 7.2 continuations of this work are explored. We start with a few direct extensions, giving pointers as to how the accuracy of the performance prediction techniques proposed here can be increased. Next, the projects in which this work will be contiued, or its results will be used, are described. Finally, in Section 7.3 our conclusions are drawn.

7.1 Summary

Network traffic is irregular

In a distributed shared-memory multiprocessor machine, the interconnection network, connecting processors and memories, is part of the memory hierarchy [Lenoski et al., 1992]. Because the network is at such a low architectural level, where components are very closely connected, it is very hard to tolerate network latency. Also, because communication in shared-memory machines is implicit, the programmer has a limited ability to schedule communication, which makes it difficult to effectively hide latency. This makes

the performance of shared-memory machines very dependent upon their network's latency.

Current interconnection networks, which are regular and have a topology that remains fixed through time, work best when network traffic is uniformly distributed in both space and time. Unfortunately, realistic network traffic, generated in response to memory accesses made by the application running on the multiprocessor machine, is neither. Fixed networks therefore often experience congestion, caused by hot-spots in the communication pattern. This can dramatically increase network latency.

We characterized this network traffic, and found that it can be described as consisting of *bursts*, which are periods of high-intensity communication between specific node pairs, on a background of more uniform traffic of much lower intensity. Multiple bursts are usually active concurrently, each can last up to several milliseconds [Heirman et al., 2005; Artundo et al., 2006b].

An ideal solution for this problem would be a reconfigurable interconnection network: one that is able to change its topology through time, so that at each point in time the topology is optimal for the current traffic pattern. The possibilities of this solution were explored in this work.

Optical reconfigurable networks can exploit traffic bursts

Future multiprocessor interconnection networks are expected to be optical, which will become necessary to accommodate the bandwidth density requirements that rise with each new generation of machines [Collet et al., 2000; Benner et al., 2005]. At the same time, optics presents us with a range of components that allow for data-transparent reconfiguration [Chang-Hasnain, 2000; d'Alessandro and Asquini, 2003; Pardo et al., 2003]. Since the switching times of these components – at least the cheap, small, low-power ones that are suitable for use in a short-range interconnection setting – are slow compared to single network packets, it is not feasible to do packet switching with them. Rather, reconfiguration should take place on a slower time scale, such as that of communication bursts.

Using these components, we proposed a reconfigurable network architecture, that can efficiently support communication bursts. Our architecture consists of a fixed base network for regular traffic, augmented with reconfigurable extra links to support high-intensity bursts [Heirman et al., 2008a]. Just like the shared-memory paradigm insulates the application and its programmer from the implementation details of memory (data placement, cache coherence), we wanted network reconfiguration to be transparent to the application. Therefore, reconfiguration is solely based on observed network

traffic, in which bursts are detected automatically. No input from the programmer or compiler is needed.

Aside from a parametric architecture, with a configurable number of extra links, fan-out and reconfiguration, we also propose a possible implementation of this architecture, using tunable VCSELs, a Selective Optical Broadcast (SOB) system and wavelength-selective photodetectors. The SOB system was developed at the Vrije Universiteit Brussel (VUB), and implements a 1-to-9 optical broadcast [Artundo et al., 2008b].

On the evaluation of reconfigurable networks

Evaluating the performance of interconnection networks is done most accurately using execution-driven simulation. This way, network traffic is extremely realistic, and interactions between network performance and software behavior can be studied. This method is also the slowest, requiring several hours per benchmark and per network that is to be analyzed. Design-space explorations are therefore necessarily conducted using less accurate, but much faster techniques.

Since the operation of a reconfigurable network depends on the non-uniformity of network traffic, existing techniques – which use simple traffic models such as uniform or hot-spot distributions – did not suffice, however. New techniques were therefore developed, that do allow an accurate prediction of a reconfigurable network's performance to be made, using more realistic network loads [Heirman et al., 2006, 2007c, 2008a].

Using these performance prediction techniques, augmented with a limited number of execution-driven simulations, we were able to do a thorough analysis of reconfigurable network performance: depending on the network parameters and available reconfiguration speeds, improvements in memory access latency of up to 40% can be obtained [Heirman et al., 2007a; Artundo et al., 2008a].

We also compared our reconfigurable network solution with the most prevalent existing technique used to cope with changing network traffic behavior, which is thread and data migration. Since this is a software technique, it lives at a time scale of seconds, rather than milliseconds for a reconfigurable network. Both techniques are therefore complementary: thread and data migration can cheaply handle slow changes in the application's dynamics, while network reconfiguration can cope with faster changes, and moreover support the migration of threads and data efficiently – by creating fast communication paths for efficiently executing the migration of data blocks throughout the network. Also, a reconfigurable network can provide

a richer network topology when a good placement of threads and data on the basic topology is not possible.

7.2 Future research

> *"Would you tell me, please, which way I ought to go from here?" asked Alice. "That depends a good deal on where you want to get to," said the Cat.*
> **— Alice in Wonderland**

7.2.1 Extensions to the current work

While the network performance prediction methods, presented in Chapter 5, posseses sufficient relative accuracy for their use in a design-space exploration, higher accuracy – especially in an absolute sense – is always useful. Here, we will give some pointers as to how this may be achieved. Also, we suggest some extensions to the models, that allow them to be used in a broader context.

Congestion model

Because the network is in a constant state of flux, the steady state queueing model used in Section 5.2 does not accurately describe what is happening inside the network. Two approaches can be followed. One can keep using the steady state model as a heuristic, but add special handling for links that have a high utilization factor. This could increase accuracy at only a slight increase in complexity. The other choice is to remodel the network as a dynamic system, probably with significantly higher accuracy, but at the expense of reduced simplicity and simulation speed. Since, both in research and during network design, there is room for multiple methods, each with a different trade-off between speed and accuracy, both approaches can be useful.

Another, more straightforward adaptation, which can also be applied to our other models, is to allow different bandwidths per link. This would allow the investigation of a wider range of architectures, for instance that from Figure 4.3, where bandwidth on a physical link is divided into separate logical links for packet routing (the base network) and circuits (the elinks).

Asymmetric link bandwidths

In Section 4.1, we described the parallel of an interconnection cached network, with the base network and reconfigurable extra links being the communication equivalents of main and cache memories. When drawing this parallel even further, one might introduce a significant asymmetry between bandwidth and latency properties of the base network links and those of the elinks. The base network would now become a very slow *last resort* communication method, while the bulk of the traffic could be carried by just a few, but much faster, reconfigurable elinks. This of course requires a rather high *hit rate* of the elinks, i.e., depending on the amount of asymmetry, a very high fraction of traffic has to be able to use the elinks in order for the network to provide an average latency characteristic close to that of the elinks. Nonetheless, there seems to be potential in this idea, as it has been successfully realized by Barker et al. [2005] – although this was done in a context where reconfiguration was under control of the application. This type of implementation does allow a much more global view of communication, since the whole program, including its future behavior, is known the programmer or compiler. An automatic approach such as ours, on the other hand, can only learn from past communication patterns.

Other architectural trade-offs

The introduction of the elinks concept enables a whole new range of trade-offs. Since time and space constraints prevented us from including much more simulation results in this work – instead favoring the description of topics that represented more novelty in a methodological sense – several interesting questions remain.

One future line of research may consist of the adaptation of the cache hierarchy, its coherence protocol, or the network routing protocol to make better use of both the added bandwidth provided by optics, and the existence of extra links. Consider for instance the situation in which a processor is accessing a certain data word in its cache, and that there happens to be an elink connecting the processor to the home node of this cache line. The coherence protocol could now temporarily switch this line to a write-update mode. This would use the ample bandwidth to the home node that is available at that time, and avoid a later write-back which would need to happen using the slower base network, and while another processor is actively waiting for it.

Also, several optimizations are possible in the way reconfiguration is now done. The reconfiguration interval can be made dynamic, reacting

to changes (or the lack thereof) in network traffic. This can result both in a quicker response time and in less unneeded reconfigurations when the traffic pattern remains constant. The 'down-time' of elinks, while they are undergoing reconfiguration, can be avoided using multiple sources per node, one of which is being re-tuned while the other one is still sending data.

Synthetic traffic

The simulations based on synthetic traffic traces suffer some systematic inaccuracies. The main problem here is that the network traffic behavior changes throughout the execution of the program, so that just one traffic profile does not suffice. Using known *program phase behavior* techniques [Sherwood et al., 2002] this can be investigated, by dividing the program into several distinct *phases.* Each phase has a different behavior, and should be represented by a separate traffic profile. The performance of the network for each benchmark application will thus be determined by its performance under a multitude of traffic profiles, one per program phase, averaged according to the relative occurrences of each phase in the program.

Another idea is to parametrize the traffic profile (the distributions shown in Figure 5.13), making it possible to generate traces for different benchmarks or execution on larger machines by intra- or extrapolating the profile's parameters, rather than measuring the distributions again in execution-driven simulations. This way, applications with similar traffic patterns can be recognized by clustering them on their profile parameters, and future applications can be represented by extrapolation of parameters for current benchmarks.

7.2.2 Towards a reconfigurable demonstrator

Currently, the design and fabrication of the SOB system is underway at the VUB. The optical transmission and broadcast properties of this system will be characterized, providing measurements of the efficiency and power consumption of our proposed reconfigurable network implementation. A complete demonstration of real-time network reconfiguration, driven by network traffic from an actual multiprocessor, is currently not the aim, but might be attempted in the future.

7.2.3 On-chip communication

The research results and methodologies from this work will be used, both in the continuation of the exploration of the specific concept of reconfigurable optical interconnects at the VUB, but also in two new projects, both in an

on-chip setting: WADIMOS will focus on on-chip optical networks, while OptiMMA aims at optimizing resource usage – which includes network resources – in an embedded multiprocessor-on-chip environment.

WADIMOS: on-chip optical interconnects

The "Wavelength Division Multiplexed Photonic Layer on CMOS" (WADIMOS) project was mentioned before in Section 2.2. Its aim is to build an optical Network-on-Chip (NoC). Indeed, even on-chip the limitations of electrical interconnections will become apparent in the near future [Haurylau et al., 2006]. One of the planned demonstrator chips in WADIMOS will integrate an 8×8 lambda-router – which performs routing from eight sources to eight destinations based on the source's wavelength – with microdisc laser sources, photodetectors and driver electronics, onto a System-on-Chip (SoC). One such SoC can implement, for instance, a complete set-top box, integrating networking, video compression and decompression, etc., in a single chip.

For this project, characterization of the complete system will be performed, including interactions with software through the application's specific traffic patterns, using the simulation setup and prediction tools described in this work. Moreover, when equipped with tunable lasers, this architecture would effectively implement a single-chip version of the reconfigurable network architecture proposed in this work. This concept can also be explored, allowing a further development of the reconfigurable architecture described here – this time aimed at an actual implementation, with regards for its specific characteristics.

OptiMMA: optimization of on-chip communication

Another project pertaining to on-chip multiprocessor communication, is called "Optimization of MPSoC Middleware for Event-driven Applications" (OptiMMA). In this project, a middleware component will be developed that can optimize resource usage, based on inherent properties of event-driven applications, in a Multi-Processor SoC (MPSoC) environment. The 'resources' considered in this context are processors, memories, and, of course, the communication network. While a reconfigurable network is not necessarily envisaged in this project, the traffic analysis methods from Chapter 3, and the performance prediction tools described in Chapter 5, can be directly applied to this project.

7.2.4 Faster reconfiguration

As was mentioned in Section 2.3, components that allow very fast reconfiguration, in the order of micro- or even nanoseconds, are becoming available. Moreover, even if the right components do not exist today, system-level research can, in advance, show whether continued research into faster components would be beneficial.

From our analysis in Chapter 6 it is clear that faster reconfiguration generally improves performance. Furthermore, Figure 6.9 did not show an obvious point of performance saturation, for reconfiguration intervals down to one microsecond, suggesting that more performance can be found in even faster reconfiguration. However, this analysis did not take the so-called selection and switching times into account, instead assuming that the computation of the new topology and routing tables, the distribution of this information to all concerned entities in the system, and the physical reconfiguration, can be done instantaneous. Yet, in reality, even if the physical components can support sub-microsecond switching times, the topology calculation will at some point become a bottleneck.

Clearly, if cheap components with tuning times of just a few nanoseconds would be available, the kind of *slow* reconfiguration as assumed in this work – which invests a significant amount of time to compute a topology resulting in the globally optimal use of available resources – makes no sense. Rather, as described in Section 3.1.1, a technique such as Optical Packet Switching (OPS) would be much more efficient in this domain. When tuning times range from several nanoseconds up to some tens of microseconds, however, all is not so clear. Packet switching is not possible there, while the global reconfiguration proposed in this work will still incur too much overhead. The solution will probably be some kind of hybrid approach between packet switching and full reconfiguration, such as a form of circuit switching or local reconfiguration, striking a balance between an optimal use of the available resources and the time required to make the necessary calculations.

7.3 Conclusion

To know the road ahead,
ask those coming back.
— Chinese Proverb

In its current proposed implementation, a reconfigurable optical network adds a large design and manufacturing cost to the system, while resulting in – at least for small networks – only modest improvements in performance. Also, an unsolved problem is that of the compatibility of parallel interconnects with reconfiguration: will it be possible to increase single link bandwidths enough to satisfy communication requirements using serial links, or will parallel connections still be needed, possibly necessitating the replication of all reconfigurable components across the parallel channels?

However, reconfigurable networks are also very promising for goals other than performance improvement, such as fault tolerance. If reconfiguration is added for this reason, then the complexity investment has already been made. In this case the use of an existing reconfiguration infrastructure can, using our techniques, result in a 40% speed improvement at very little additional cost.

Moreover, the applicability of this work is not limited to the specific implementation from Section 4.2. The analysis of network traffic, made in Chapter 3, is valid for all multiprocessor systems, no matter what interconnection network is used in them. New reconfigurable components and network architectures can be developed, based on the lessons learned in this work. Since the burstiness of network traffic is an essential property, inherent to the communication resulting from most parallel algorithms, the useful scope of reconfiguration is not limited to shared-memory environments. A similar method of reconfiguration can be implemented in architectures different from our distributed shared-memory multiprocessor, such as message-passing machines, or even inside Single Instruction, Multiple Data (SIMD) processors or graphics accelerators. Finally, the tools and methodologies developed here, specifically the methods from Chapter 5 to speed up design-space explorations, can be, and are being, applied in network design of both static and reconfigurable, on- and off-chip interconnection networks.

Publications

Journal papers

- **Heirman, W.**, Dambre, J., Artundo, I., Debaes, C., Thienpont, H., Stroobandt, D., and Van Campenhout, J. (2008). Predicting the performance of reconfigurable optical interconnects in distributed shared-memory systems. *Photonic Network Communications*. 15(1):25–40.
- **Heirman, W.**, Dambre, J., Artundo, I., Debaes, C., Thienpont, H., Stroobandt, D., and Van Campenhout, J. (2007). Predicting reconfigurable interconnect performance in distributed shared-memory systems. *Integration, the VLSI Journal*, 40(4):382–393.
- Artundo, I., Desmet, L., **Heirman, W.**, Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2006). Selective optical broadcast component for reconfigurable multiprocessor interconnects. *IEEE Journal of Selected Topics in Quantum Electronics: Special Issue on Optical Communication*, 12(4):828–837.

Conference papers

- **Heirman, W.**, Dambre, J., Stroobandt, D., Van Campenhout, J. (2008). Runtime variability in scientific parallel applications. In *Proceedings of the Fourth Workshop on Modeling, Benchmarking and Simulation at ISCA-35*, pages 37–46, Beijing, China.
- Artundo, I., **Heirman, W.**, Debaes, C., Dambre, J., Van Campenhout, J., Thienpont, H. (2008). Design of a reconfigurable optical interconnect for large-scale multiprocessor networks. In *Proceedings of SPIE Photonics Europe*, volume 6996, page 69961H, Strasbourg, France.

- **Heirman, W.**, Dambre, J., Stroobandt, D., Van Campenhout, J. (2008). Rent's rule and parallel programs: characterizing network traffic behavior. In *Proceedings of the 10th International Workshop on System Level Interconnect Prediction (SLIP 2008)*, pages 87–94, Newcastle, United Kingdom.

- Artundo, I., **Heirman, W.**, Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2007). Performance of large-scale reconfigurable optical interconnection networks in DSM systems. In *Proceedings of the IEEE/LEOS Symposium Benelux Chapter*, pages 123–126, Brussels, Belgium.

- **Heirman, W.**, Artundo, I., Dambre, J., Debaes, C., Pham Doan, T., Bui Viet, K., Thienpont, H., and Van Campenhout, J. (2007). Performance evaluation of large reconfigurable interconnects for multiprocessor systems. In *Proceedings of the International Symposium on Electrical – Electronics Engineering (ISEE 2007)*, pages 145–150, Ho Chi Minh City, Vietnam.

- **Heirman, W.**, Dambre, J., and Van Campenhout, J. (2007). Synthetic traffic generation as a tool for dynamic interconnect evaluation. In *Proceedings of the 9th International Workshop on System Level Interconnect Prediction (SLIP 2007)*, pages 65–72, Austin, Texas.

- Artundo, I., Manjarres, D., **Heirman, W.**, Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2006). Reconfigurable interconnects in DSM systems: a focus on context switch behavior. In *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, volume 4331, pages 311–321, Sorrento, Italy.

- Bui Viet, K., Pham Doan, T., Nguyen Nam, Q., Artundo, I., Manjarres, D., **Heirman, W.**, Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2006). Reconfigurable interconnection networks in distributed shared memory systems: a study on communication patterns. In *Proceedings of the First International Conference on Communications and Electronics (HUT-ICCE 2006)*, pages 343–347, Hanoi, Vietnam.

- Artundo, I., Desmet, L., **Heirman, W.**, Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2006). Selective optical broadcasting in reconfigurable multiprocessor interconnects. In *Proceedings of SPIE Photonics Europe*, volume 6185, page 61850J, Strasbourg, France.

- **Heirman, W.**, Dambre, J., and Van Campenhout, J. (2006). Congestion modeling for reconfigurable inter-processor networks. In *Proceedings*

of the 8th International Workshop on System Level Interconnect Prediction (SLIP 2006), pages 59–66, Munich, Germany.

- **Heirman, W.**, Artundo, I., Desmet, L., Dambre, J., Debaes, C., Thienpont, H., and Van Campenhout, J. (2006). Speeding up multiprocessor machines with reconfigurable optical interconnects. In *Proceedings of SPIE, Optoelectronic Integrated Circuits VIII, Photonics West*, volume 6124, page 61240K, San Jose, California.

- Artundo, I., Desmet, L., **Heirman, W.**, Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2005). Selective broadcasting for reconfigurable optical interconnects in DSM systems. In *Proceedings of the IEEE/LEOS Symposium Benelux Chapter*, pages 225–228, Mons, Belgium.

- **Heirman, W.**, Dambre, J., and Van Campenhout, J. (2005). Predicting the performance of reconfigurable interconnects in distributed shared-memory systems. In *Proceedings of the 16th ProRISC Workshop*, pages 508–517, Veldhoven, the Netherlands.

- **Heirman, W.**, Artundo, I., Carvajal, D., Desmet, L., Dambre, J., Debaes, C., Thienpont, H., and Van Campenhout, J. (2005). Wavelength tuneable reconfigurable optical interconnection network for shared-memory machines. In *Proceedings of the 31st European Conference on Optical Communication (ECOC 2005)*, volume 3, pages 527–528, Glasgow, United Kingdom.

- **Heirman, W.**, Dambre, J., Van Campenhout, J., Debaes, C., and Thienpont, H. (2005). Traffic temporal analysis for reconfigurable interconnects in shared-memory systems. In *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2005)*, page 150, Denver, Colorado.

- **Heirman, W.**, Dambre, J., Stroobandt, D., Van Campenhout, J., Debaes, C., and Thienpont, H. (2005). Prediction model for evaluation of reconfigurable interconnects in distributed shared-memory systems. In *Proceedings of the 7th International Workshop on System Level Interconnect Prediction (SLIP 2005)*, pages 51–58, San Francisco, California.

- Debaes, C., Artundo, I., **Heirman, W.**, Dambre, J., Bui Viet, K., and Thienpont, H. (2004). Architectural study of the opportunities for reconfigurable optical interconnects in distributed shared memory systems. In *Proceedings of the IEEE/LEOS Symposium Benelux Chapter*, pages 275–278, Ghent, Belgium.

- **Heirman, W.**, Dambre, J., Debaes, C., Van Campenhout, J., and Thienpont, H. (2004). Traffic pattern analysis for reconfigurable interconnects in shared-memory systems. In *Proceedings of the 15th ProRISC Workshop*, pages 39–44, Veldhoven, the Netherlands.

Poster presentations and abstracts

- **Heirman, W.**, Dambre, J., and Van Campenhout, J. (2006). Predicting the performance of reconfigurable interconnects in shared-memory systems. In *Seventh FirW PhD Symposium*, page 84, Ghent, Belgium.

- **Heirman, W.**, Dambre, J., Artundo, I., Debaes, C., Thienpont, H., Stroobandt, D., and Van Campenhout, J. (2006). Predicting the performance of reconfigurable interconnects in distributed shared-memory systems. In *Architectures and Compilers for Embedded Systems (ACES 2006): Symposium Proceedings*, pages 31–34, Edegem, Belgium.

- Vandeputte, F., Eeckhout, L., De Bosschere, K., and **Heirman, W**. (2006). Identifying program phase behavior in parallel programs on distributed shared-memory systems. In *Advanced Computer Architecture and Compilation for Embedded Systems (ACACES 2006)*, Ghent, Belgium.

- **Heirman, W.**, Dambre, J., O'Connor, I., and Van Campenhout, J. (2006). Reconfigurable optical networks for on-chip multiprocessors. In *Future Interconnects and Networks on Chip Workshop at DATE 2006*, Munich, Germany.

- **Heirman, W.**, Dambre, J., and Van Campenhout, J. (2005). Reconfigurable optical interconnects for distributed shared-memory systems. In *Advanced Computer Architecture and Compilation for Embedded Systems (ACACES 2005)*, pages 19–22, Ghent, Belgium.

- **Heirman, W.**, Dambre, J., and Van Campenhout, J. (2005). Reconfigurable optical interconnects for distributed shared-memory multiprocessors. In *PhD Forum at Design, Automation and Test in Europe (DATE 2005)*, Munich, Germany.

- **Heirman, W.**, Dambre, J., and Van Campenhout, J. (2004). Traffic locality analysis for reconfigurable interconnects in shared-memory systems. In *Fifth FTW PhD Symposium*, Ghent, Belgium.

References

Akulova, Y., Fish, G., Koh, P.-C., Schow, C., Kozodoy, P., Dahl, A., Nakagawa, S., Larson, M., Mack, M., Strand, T., Coldren, C., Hegblom, E., Penniman, S., Wipiejewski, T., and Coldren, L. (2002). Widely tunable electroabsorption-modulated sampled-grating dbr laser transmitter. *IEEE Journal of Selected Topics in Quantum Electronics*, 8(6):1349–1357.

Alameldeen, A., Mauer, C., Xu, M., Harper, P., Martin, M., Sorin, D., Hill, M., and Wood, D. (2002). Evaluating non-deterministic multi-threaded commercial workloads. In *Proceedings of the Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads*, pages 30–38.

Aljada, M., Alameh, K. E., Lee, Y.-T., , and Chung, I.-S. (2006). High-speed (2.5 Gbps) reconfigurable inter-chip optical interconnects using opto-VLSI processors. *Optics Express*, 14(15):6823–6836.

AMD (2004). Functional data sheet, 940 pin package, `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/31412.pdf`.

Amdahl, G. (1967). Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, Atlantic City, New Jersey.

Artiaga, E., Martorell, X., Becerra, Y., and Navarro, N. (1998). Experiences on implementing PARMACS macros to run the SPLASH-2 suite on multiprocessors. In *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, pages 64–69, Madrid, Spain.

Artundo, I., Desmet, L., Heirman, W., Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2006a). Selective optical broadcast com-

ponent for reconfigurable multiprocessor interconnects. *IEEE Journal of Selected Topics in Quantum Electronics: Special Issue on Optical Communication*, 12(4):828–837.

Artundo, I., Heirman, W., Bui Viet, K., Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2008a). Performance evaluation of reconfigurable interconnects for large distributed shared-memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*. Submitted for review.

Artundo, I., Heirman, W., Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2008b). Design of a reconfigurable optical interconnect for large-scale multiprocessor networks. In *Proc. of SPIE Photonics Europe*, volume 6996, page 69961H, Strasbourg, France. To appear.

Artundo, I., Manjarres, D., Heirman, W., Debaes, C., Dambre, J., Van Campenhout, J., and Thienpont, H. (2006b). Reconfigurable interconnects in DSM systems: A focus on context switch behavior. In *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, volume 4331, pages 311–321, Sorrento, Italy. Springer Berlin / Heidelberg.

Barford, P. and Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, pages 151–160, Madison, Wisconsin.

Barker, K. J., Benner, A., Hoare, R., Hoisie, A., Jones, A. K., Kerbyson, D. K., Li, D., Melhem, R., Rajamony, R., Schenfeld, E., Shao, S., Stunkel, C., and Walker, P. (2005). On the feasibility of optical circuit switching for high performance computing systems. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 16, Washington, DC. IEEE Computer Society.

Beeckman, J., Neyts, K., and Haelterman, M. (2006). Patterned electrode steering of nematicons. *Journal of Optics A: Pure and Applied Optics*, 8:214–220.

Benner, A. F., Ignatowski, M., Kash, J. A., Kuchta, D. M., and Ritter, M. B. (2005). Exploitation of optical interconnects in future server architectures. *IBM Journal of Research and Development*, 49(4/5):755–776.

Bertels, P. and Stroobandt, D. (2006). Profiling based estimation of communication for system partitioning. In *Proceedings of the 17th Annual ProRISC Workshop*, pages 233–239.

Beyls, K. and D'Hollander, E. (2001). Reuse distance as a metric for cache behavior. In Gonzalez, T., editor, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 617–622, Anaheim, California.

Biswas, R., Djomehri, M. J., Hood, R., Jin, H., Kiris, C., and Saini, S. (2005). An application-based performance characterization of the columbia supercluster. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing (SC'05)*, page 26, Washington, DC. IEEE Computer Society.

Bockstaele, R., De Wilde, M., Meeus, W., Rits, O., Lambrecht, H., Van Campenhout, J., De Baets, J., Van Daele, P., van den Berg, E., Clemenc, M., Eitel, S., Annen, R., Van Koetsem, J., Widawski, G., Goudeau, J., Bareel, B., Le Moine, P., Fries, R., Straub, P., and Baets, R. (2004). A parallel optical interconnect link with on-chip optical access. In Thienpont, H., Choquette, K. D., and Taghizadeh, M. R., editors, *Micro-Optics, VCSELs, and Photonic Interconnects*, volume 5453 of *Proceedings of SPIE*, pages 124–133.

Boden, N., Cohen, D., Felderman, R., Kulawik, A., Seitz, C., Seizovic, J., and Su, W.-K. (1995). Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 1(15):29–38.

Brunfaut, M., Meeus, W., Van Campenhout, J., Annen, R., Zenklusen, P., Melchior, H., Bockstaele, R., Vanwassenhove, L., Hall, J., Wittman, B., Nayer, A., Heremans, P., Van Koetsem, J., King, R., Thienpont, H., and Baets, R. (2001). Demonstrating optoelectronic interconnect in a FPGA based prototype system using flip chip mounted 2D arrays of optical components and 2D POF-ribbon arrays as optical pathways. In *Proceedings of SPIE*, volume 4455, pages 160–171, Bellingham, Washington.

Chandra, R., Devine, S., Verghese, B., Gupta, A., and Rosenblum, M. (1994). Scheduling and page migration for multiprocessor compute servers. In *ASPLOS-VI: Proceedings of the sixth international conference on Architectural support for programming languages and operating systems*, pages 12–24, San Jose, California. ACM.

Chang-Hasnain, C. (2000). Tunable VCSEL. *IEEE Journal of Selected Topics in Quantum Electronics*, 6(6):978–987.

Charlesworth, A. (2001). The Sun Fireplane system interconnect. In *ACM/IEEE Conference on Supercomputing*, page 7, Denver, Colorado.

Charlesworth, A., Phelps, A., Williams, R., and Gilbert, G. (1997). Gigaplane-XB: extending the ultra enterprise family. In *Proceedings of Hot Interconnects V*, pages 97–112, Stanford, California.

Christie, P. and Stroobandt, D. (2000). The interpretation and application of Rent's rule. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(6):639–648.

Clark, N. and Handshy, M. (1990). Surface-stabilized ferroelectric liquid-crystal electro-optic waveguide switch. *Applied Physics Letters*, 57:1852–1854.

Collet, J., Litaize, D., Campenhout, J. V., Desmulliez, M., Jesshope, C., Thienpont, H., Goodman, J., and Louri, A. (2000). Architectural approach to the role of optics in monoprocessor and multiprocessor machines. *Applied Optics*, 39(5):671–682.

Crossland, W., Manolis, I., Redmond, M., Tan, K., Wilkinson, T., Holmes, M., Parker, T., Chu, H., Croucher, J., Handerek, V., Warr, S., Robertson, B., Bonas, I., Franklin, R., Stace, C., H.White, R.Woolley, and Henshall, G. (2000). Holographic optical switching: The 'roses' demonstrator. *IEEE/OSA Journal of Lightwave Technology*, 18:1845–1854.

Culler, D. E. and Singh, J. P. (1999). *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, Inc., San Francisco, California.

d'Alessandro, A. and Asquini, R. (2003). Liquid crystal devices for photonic switching applications: State of the art and future developments. *Molecular Crystals and Liquid Crystals*, 398:207–221.

Dally, W. J. and Towles, B. P. (2004). *Principles and Practices of Interconnection Networks*. Morgan Kaufmann.

De Wilde, M. (2007). *Modeling and Integration of Highly Parallel Optical Interconnect in Electronic Systems*. PhD thesis, Universiteit Gent.

De Wilde, M., Rits, O., Baets, R., and Van Campenhout, J. (2008). Synchronous parallel optical I/O on CMOS: A case study of the uniformity issue. *IEEE/OSA Journal of Lightwave Technology*, 26(2):257–275.

Debaes, C., Erps, J. V., Vervaeke, M., Volckaerts, B., Ottevaere, H., Gomez, V., Vynck, P., Desmet, L., Krajewski, R., Ishii, Y., Hermanne, A., and Thienpont, H. (2006). Deep proton writing: a rapid prototyping polymer micro-fabrication tool for micro-optical modules. *New Journal of Physics*, 8(11):270.

Duato, J., Yalamanchili, S., and Ni, L. (2003). *Interconnection Networks: an Engineering Approach*. Morgan Kaufmann.

Dudley, D., Duncan, W. M., and Slaughter, J. (2003). Emerging digital micromirror device (DMD) applications. In *Proceedings of SPIE: MOEMS Display and Imaging Systems*, volume 4985, pages 14–25, San Jose, California. SPIE.

FDDI (1987). Fiber-distributed data interface (FDDI) – Token ring media access control (MAC). American National Standard for Information Systems ANSI X3.139-1987.

Feldman, M. R., Esener, S. C., Guest, C. C., and Lee, S. H. (1988). Comparisons between optical and electrical interconnects based on power and speed considerations. *Applied Optics*, 27(9):1742.

Filios, A., Gutiérrez-Castrejón, R., Tomkos, I., Hallock, B., Vodhanel, R., Coombe, A., Yuen, W., Moreland, R., Garrett, B., Duvall, C., and Chang-Hasnain, C. (2003). Transmission performance of a 1.5 μm 2.5 Gb/s directly modulated tunable VCSEL. *IEEE Photonics Technology Letters*, 15(4):599–601.

Garcia, J. and Duato, J. (1993). Dynamic reconfiguration of multicomputer networks: limitations and tradeoffs. In *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing*, pages 317–323, Gran Canaria, Spain.

Geer, D. (2005). Chip makers turn to multicore processors. *IEEE Computer*, 38(5):11–13.

Goddard, I. (2003). Division of labor in embedded systems. *ACM Queue*, 1(2):32.

Greenfield, D., Banerjee, A., Lee, J.-G., and Moore, S. (2007). Implications of Rent's rule for NoC design and its fault-tolerance. In *Proceedings of the First International Symposium on Networks-on-Chips (NOCS'07)*, pages 283–294, Princeton, New Jersey.

Greenfield, D. and Moore, S. (2008). Fractal communication in software data dependency graphs. In *Proceedings of the 20th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'08)*, pages 116–118, Munich, Germany.

Gros, E. and Dupont, L. (2001). Ferroelectric liquid crystal optical waveguide switches using the double-refraction effect. *IEEE Photonics Technology Letters*, 13(2):115–117.

Gupta, V. and Schenfeld, E. (1994). Performance analysis of a synchronous, circuit-switched interconnection cached network. In *ICS '94: Proceedings of the 8th international conference on Supercomputing*, pages 246–255, Manchester, England. ACM.

Gustafson, J. (1988). Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533.

Habata, S., Umezawa, K., Yokokawa, M., and Kitawaki, S. (2004). Hardware system of the earth simulator. *Parallel Computing*, 30(12):1287–1313.

Han, X. and Chen, R. T. (2004). Improvement of multiprocessing performance by using optical centralized shared bus. In *Proceedings of the SPIE*, volume 5358, pages 80–89.

Haurylau, M., Chen, G., Chen, H., Zhang, J., Nelson, N., Albonesi, D., Friedman, E., and Fauchet, P. (2006). On-chip optical interconnect roadmap: Challenges and critical directions. *IEEE Journal of Selected Topics in Quantum Electronics: Special Issue on Silicon Photonics*, 12(6):1699–1705.

Hawkins, C., Small, B. A., Wills, D. S., and Bergman, K. (2007). The data vortex, an all optical path multicomputer interconnection network. *IEEE Transactions on Parallel and Distributed Systems*, 18(3):409–420.

Heirman, W., Artundo, I., Dambre, J., Debaes, C., Pham Doan, T., Bui Viet, K., Thienpont, H., and Van Campenhout, J. (2007a). Performance evaluation of large reconfigurable interconnects for multiprocessor systems. In *Proceedings of the International Symposium on Electrical - Electronics Engineering (ISEE 2007)*, pages 145–150, Ho Chi Minh City, Vietnam.

Heirman, W., Dambre, J., Artundo, I., Debaes, C., Thienpont, H., Stroobandt, D., and Van Campenhout, J. (2007b). Predicting reconfigurable interconnect performance in distributed shared-memory systems. *Integration, the VLSI Journal*, 40(4):382–393.

Heirman, W., Dambre, J., Artundo, I., Debaes, C., Thienpont, H., Stroobandt, D., and Van Campenhout, J. (2008a). Predicting the performance of reconfigurable optical interconnects in distributed shared-memory systems. *Photonic Network Communications*, 15(1):25–40.

Heirman, W., Dambre, J., Stroobandt, D., and Van Campenhout, J. (2008b). Rent's rule and parallel programs: Characterizing network traffic behavior. In *Proceedings of the 2008 International Workshop on System Level Interconnect Prediction (SLIP'08)*, pages 87–94, Newcastle, United Kingdom. ACM.

Heirman, W., Dambre, J., Stroobandt, D., and Van Campenhout, J. (2008c). Runtime variability in scientific parallel applications. In *Proceedings of the Fourth Workshop on Modeling, Benchmarking and Simulation at ISCA-35*, pages 37–46, Beijing, China.

Heirman, W., Dambre, J., and Van Campenhout, J. (2006). Congestion modeling for reconfigurable inter-processor networks. In *Proceedings of the 2006 International Workshop on System Level Interconnect Prediction (SLIP'06)*, pages 59–66, Munich, Germany. ACM Press.

Heirman, W., Dambre, J., and Van Campenhout, J. (2007c). Synthetic traffic generation as a tool for dynamic interconnect evaluation. In *Proceedings of the 2007 International Workshop on System Level Interconnect Prediction (SLIP'07)*, pages 65–72, Austin, Texas. ACM Press.

Heirman, W., Dambre, J., Van Campenhout, J., Debaes, C., and Thienpont, H. (2005). Traffic temporal analysis for reconfigurable interconnects in shared-memory systems. In *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium*, page 150, Denver, Colorado. IEEE Computer Society.

Held, J., Bautista, J., and Koehl, S. (2006). From a few cores to many: A tera-scale computing research overview. Research at Intel White Paper.

Henderson, C. J., Leyva, D. G., and Wilkinson, T. D. (2006). Free space adaptive optical interconnect at 1.25 Gb/s, with beam steering using a ferroelectric liquid-crystal SLM. *IEEE/OSA Journal of Lightwave Technology*, 24(5):1989–1997.

Horowitz, M. (2007). Microprocessors through the ages, `http://www-vlsi.stanford.edu/group/chips_micropro.html`.

Hu, S.-Y., Ko, J., Hegblom, E., and Coldren, L. (Aug 1998). Multimode wdm optical data links with monolithically integrated multiple-channel vcsel and photodetector arrays. *IEEE Journal of Quantum Electronics*, 34(8):1403–1414.

Huang, D., Sze, T., Landin, A., Lytel, R., and Davidson, H. (2003). Optical interconnects: out of the box forever? *IEEE Journal of Selected Topics in Quantum Electronics*, 9(2):614–623.

Huang, M. C. Y., Zhou, Y., and Chang-Hasnain, C. J. (2008). A nanoelectromechanical tunable laser. *Nature Photonics*, 2(3):180–184.

Inifiniband (2000). `http://www.infinibandta.com/`.

Jiang, D. and Singh, J. P. (1999). Scaling application performance on a cache-coherent multiprocessors. In *Proceedings of the 26th International Symposium on Computer Architecture*, pages 305–316, Atlanta, Georgia.

Kanter, D. (2007). The Common System Interface: Intel's future interconnect. Real World Technologies.

Katsinis, C. (2001). Performance analysis of the simultaneous optical multiprocessor exchange bus. *Parallel Computing*, 27(8):1079–1115.

Keltcher, C., McGrath, K., and Ahmed, A.and Conway, P. (2003). The AMD Opteron processor for multiprocessor servers. *IEEE Micro*, 23(2):66–76.

Kirman, N., Kirman, M., Dokania, R., Martinez, J., Apsel, A., Watkins, M., and Albonesi, D. (2007). On-chip optical technology in future bus-based multicore designs. *IEEE Micro*, 27(1):56–66.

Koyama, F. (2006). Recent advances of VCSEL photonics. *IEEE/OSA Journal of Lightwave Technology*, 24(12):4502–4513.

Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, J., Nakahira, D., Baxter, J., Horowitz, M., Gupta, A., Rosenblum, M., and Hennessy, J. (1994). The Stanford FLASH multiprocessor. *ACM SIGARCH Computer Architecture News*, 22(2):302–313.

Landman, B. S. and Russo, R. L. (1971). On a pin versus block relationship for partitions of logic graphs. *IEEE Transactions on Computers*, C-20(12):1469–1479.

Lee, S.-S., Huang, L.-S., Kim, C.-J., and Wu, M. (1999). Free-space fiber-optic switches based on mems vertical torsion mirrors. *IEEE/OSA Journal of Lightwave Technology*, 17(1):7–13.

Leiserson, C. E., Abuhamdeh, Z. S., Douglas, D. C., Feynman, C. R., Ganmukhi, M. N., Hill, J. V., Hillis, W. D., Kuszmaul, B. C., Pierre, M. A. S., Wells, D. S., Wong-Chan, M. C., Yang, S.-W., and Zak, R. (1996). The network architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158.

Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J. L., Horowitz, M., and Lam, M. S. (1992). The Stanford DASH multiprocessor. *IEEE Computer*, 25(3):63–79.

Los Alamos National Laboratory (2005). Operational data to support and enable computer science research,
`http://institutes.lanl.gov/data/fdata/`.

Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., and Werner, B. (2002). Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58.

Mannava, P. K., Lee, V. W., Kumar, A., Jayasimha, D. N., and Schoinas, I. T. (2008). Dynamic interconnect width reduction to improve interconnect availability. US Patent 7,328,368. Intel.

McNutt, B. (2000). *The Fractal Structure of Data Reference: Applications to the Memory Hierarchy*. Kluwer Academic Publishers.

Miller, D. A. B. and Ozaktas, H. M. (1997). Limit to the bit-rate capacity of electrical interconnects from the aspect ratio of the system architecture. *Journal of Parallel and Distributed Computing*, 41(1):42–52.

Mohammed, E. et al. (2004). Optical interconnect system integration for ultra-short-reach applications. *Intel Technology Journal*, 8(2):115–127.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8):144–116.

Nabiev, R. and Yuen, W. (2003). Tunable lasers for multichannel fiber-optic sensors. *Sensors*. Available online.

Neilson, D. T. (2006). Photonics for switching and routing. *IEEE Journal of Selected Topics in Quantum Electronics*, 12(4):669–678.

Noordergraaf, L. and van der Pas, R. (1999). Performance experiences on Sun's WildFire prototype. In *Proceedings of Supercomputing '99*, Portland, Oregon.

O'Connor, I. (2004). Optical solutions for system-level interconnect. In *Proceedings of the 2004 International Workshop on System Level Interconnect Prediction (SLIP'04)*, pages 79–88, Paris, France.

O'Connor, I., Tissafi-Drissi, F., Gaffiot, F., Dambre, J., De Wilde, M., Van Campenhout, J., Van Thourhout, D., Van Campenhout, J., and Stroobandt, D. (2007). Systematic simulation-based predictive synthesis of integrated optical interconnect. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(8):927–940.

Pardo, F., Aksyuk, V. A., Arney, S., Bair, H., Basavanhally, N. R., Bishop, D. J., Bogart, G. R., Bolle, C. A., Bower, J. E., and Carr, D. (2003). Optical MEMS devices for telecom systems. In *Proceedings of SPIE*, volume SPIE-5116, pages 435–444. SPIE.

Patterson, D. A. (2004). Latency lags bandwith. *Communications of the ACM*, 47(10):71–75.

Phelps, A. E., Drogichen, D. P., and Kay, D. B. (2005). Dynamically reconfigurable interconnection. US Patent 6,871,294. Sun Microsystems.

Pinkston, T. M. and Goodman, J. W. (1994). Design of an optical reconfigurable shared-bus-hypercube interconnect. *Applied Optics*, 33(8):1434–1443.

Ridruejo, F., Gonzalez, A., and Miguel-Alonso, J. (2005). TrGen: A traffic generation system for interconnection network simulators. In *1st. Int. Workshop on Performance Evaluation of Networks for Parallel, Cluster and Grid Computing Systems (PEN-PCGCS'05)*, pages 547–553, Olso, Norway.

Rits, O., De Wilde, M., Roelkens, G., Bockstaele, R., Annen, R., Bossard, M., Marion, F., and Baets, R. (2006). 2d parallel optical interconnects between cmos ics. In Eldada, L. and Lee, E.-H., editors, *Proceedings of SPIE, Optoelectronic Integrated Circuits VIII, Photonics West*, volume 6124, pages 168–179, San Jose, California. SPIE.

Robertazzi, T. (2000). *Computer Networks & Systems: Queueing Theory and Performance Evaluation*. Springer.

Roelkens, G., Van Campenhout, J., Brouckaert, J., Van Thourhout, D., Baets, R., Rojo Romeo, P., Regreny, P., Kazmierczak, A., Seassal, C., Letartre, X., Hollinger, G., Fedeli, J., Di Cioccio, L., and Lagahe-Blanchard, C. (2007). III-V/Si photonics by die-to-wafer bonding. *Materials Today*, 10(7-8):36–43.

Rogers, A. and Pingali, K. (1994). Compiling for distributed memory architectures. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):281–298.

Sánchez, J. L., Duato, J., and García, J. M. (1998). Using channel pipelining in reconfigurable interconnection networks. In *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing*.

Schares, L., Kash, J., Doany, F., Schow, C., Schuster, C., Kuchta, D., Pepeljugoski, P., Trewhella, J., Baks, C., John, R., Shan, L., Kwark, Y., Budd, R., Chiniwalla, P., Libsch, F., Rosner, J., Tsang, C., Patel, C., Schaub, J., Dangel, R., Horst, F., Offrein, B., Kucharski, D., Guckenberger, D., Hegde, S., Nyikal, H., Lin, C.-K., Tandon, A., Trott, G., Nystrom, M., Bour, D., Tan, M., and Dolfi, D. (2006). Terabus: Terabit/second-class card-level optical interconnect technologies. *IEEE Journal of Selected Topics in Quantum Electronics*, 12(5):1032–1044.

Schroeder, B. and Gibson, G. A. (2007). Understanding failures in petascale computers. *Journal of Physics: Conference Series*, 78:012022 (11pp).

Shacham, A., Small, B. A., Liboiron-Ladouceur, O., and Bergman, K. (2005). A fully implemented 12×12 data vortex optical packet switching interconnection network. *IEEE/OSA Journal of Lightwave Technology*, 23(10):3066–3075.

Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. (2002). Automatically characterizing large scale program behavior. In *ASPLOS-X: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, San Jose, California.

Shively, R. R., Morgan, E. B., Copley, T. W., and Gorin, A. L. (1989). A high performance reconfigurable parallel processing architecture. In *Supercomputing '89: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, pages 505–509. ACM.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J. (1995). *MPI: The Complete Reference*. MIT Press.

Snyder, L. (1982). Introduction to the configurable, highly parallel computer. *Computer*, 15(1):47–56.

Stroobandt, D. (2001). *A Priori Wire Length Estimates for Digital Design*. Kluwer Academic Publishers, Boston / Dordrecht / London.

Sun Microsystems (2003a). An overview of UltraSPARC III Cu, `http://www.sun.com/processors/whitepapers/usiiicuoverview.pdf`.

Sun Microsystems (2003b). Sun Fire 6800 midframe server datasheet, `http://www.sun.com/servers/midrange/sunfire6800/datasheet.pdf`.

Sutter, H. (2005). The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3):16–20.

Trezza, J., Hamster, H., Iamartino, J., Bagheri, H., and DeCusatis, C. (2003). Parallel optical interconnects for enterprise class server clusters: needs and technology solutions. *IEEE Communications Magazine*, 41(2):S36–S42.

Tseng, P.-S. (1989). *A parallelizing compiler for distributed memory parallel computers*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pensilvania.

Van Campenhout, J. (2007). *Thin-Film Microlasers for the Intergration of Electronic and Photonic Intergrated Circuits*. PhD thesis, Universiteit Gent.

Van Campenhout, J., Romeo, P. R., Van Thourhout, D., Seassal, C., Regreny, P., Di Cioccio, L., Fedeli, J.-M., and Baets, R. (2008). Design and optimization of electrically injected InP-based microdisk lasers integrated on and coupled to a SOI waveguide circuit. *IEEE/OSA Journal of Lightwave Technology*, 26(1):52–63.

Van Kan, J., Sanchez, J., Xu, B., Osipowicz, T., and Watt, F. (1999). Micromachining using focused high energy ion beams: Deep ion beam lithography. *Nuclear Instruments and Methods in Physics Research B*, 148:1085–1089.

Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Singh, A., Jacob, T., Jain, S., Erraguntla, V., Roberts, C., Hoskote, Y., Borkar, N., and Borkar, S. (2008). An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41.

Verghese, B., Devine, S., Gupta, A., and Rosenblum, M. (1996). Operating system support for improving data locality on CC-NUMA compute servers. In *ASPLOS-VII: Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 279–289, Cambridge, Massachusetts. ACM.

Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. (1995). The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22th International Symposium on Computer Architecture*, pages 24–36, Santa Margherita Ligure, Italy.

Yoshimura, T., Ojima, M., Arai, Y., and Asama, K. (2003). Three-dimensional self-organized microoptoelectronic systems for board-level reconfigurable optical interconnects-performance modeling and simulation. *IEEE Journal of Selected Topics in Quantum Electronics*, 9(2):492–511.