THE HATHI PROJECT

A Research Project on
Parallel Programming Technology

1986-1988
(Final Report)

M. Aspnäs, R.J.R. Back, K. Sere

Ser. A, No 95, 1989

Dept. of Computer Science
Lemminkäinengatan 14
SF-20520 Åbo, Finland

November 1989

ISBN 951-649-654-7
ISSN 0358-0563

Acknowledgment

We would like to thank the following organizations for their generous support of the research reported here:

The Technology Development Centre of Finland (TEKES)

The Academy of Finland

The Ministry of Education (Doctoral program in information technology)

Åbo Akademi

The Technical Research Centre of Finland (Computer Technology Laboratory) Oulu

The Foundation for Åbo Akademi

Contents

Chapter 1

Introduction

Parallel programming technology has emerged recently and rather suddenly as a possible and also comercially practical alternative to achieving high computing efficiencies. The massively parallel computer architectures are particularly interesting, as they promise both very cost-effective computing and seemingly unlimited scalability. The problems associated with this technology are, however, many. One of the most central problems has been what kind of computer architecture to use for massive parallellism. There are a number of different competing approaches to building large parallel computers, and none of these has emerged as a clearly superior architecture. Another important problem has been how to program these massively parallel computers. The very nature of parallelism seems to require a new way of constructing parallel programs. Finally, there is the problem of whether the massive parallelism available in such a computer can actually be utilized in practical applications, and if so, what kind of applications are most suitable for parallel implementations.

The purpose of the *Hathi-project* was to study these problems, both from a practical and a theoretical point of view. The Hathi-project was actually a combination of two larger research projects, one on multiprocessing technology supported by the Technology Development Centre of Finland (TEKES) for the period of 1.8.1986 - 31.12.1988, and one on formal methods for constructing distributed systems, supported by the Academy of Finland for the period of 1.1.1986 - 31.12.1988. The two projects were in practice merged into a single project, as they were seen to complement each other well. Both projects were mainly located at the Department of Computer Science at Åbo Akademi, with Ralph-Johan Back as project leader. The research on multiprocessor technology was a joint project with the Technical Research Centre of Finland (Computer Technology Laboratory) in Oulu (VTT/TKO).

The area of research in the Hathi-project can be loosely seen as falling into three different parts:

(A) The design and construction of a massively parallel multiprocessor system based on the Inmos transputer.

(B) The study and development of formal methods for constructing parallel and distributed systems.

(C) Evaluating the efficiency of parallel processing technology in practical applications.

The research on multiprocessor systems (A) started in August 1986, with the purpose of building a massively parallel multiprocessor system using the recently announced Inmos transputer chips. The Technical Research Centre of Finland in Oulu was responsible for the hardware

design and construction while Åbo Akademi was responsible for building the system software. The main result of this project is a large multiprocessor system, Hathi–2, which is installed at Åbo Akademi and is now used by a number of research projects on parallel computing.

The research on formal methods (B) had started somewhat earlier, in January 1986, but was based on a longer research tradition in this area. The main result of this project is a new method for constructing parallel programs by stepwise refinement which was studied both theoretically and in practical case studies. The main co-operation in this project was with professor Reino Kurki-Suonio at the Tampere University of Technology.

The last part of the Hathi-project, the experimental study of parallel processing technology (C) was carried out in co-operation with a number of other research institutes. The main partners were the Technical Research Centre of Finland (Laboratory for Information Processing) in Helsinki (VTT/TIK), the Faculty of Chemical Engineering at Åbo Akademi and the Department of Physics at the University of Jyväskylä. A number of application projects were also carried out internally at the Department of Computer Science at Åbo Akademi. Funding was partly from TEKES and the Academy of Finland, and partly by internal funding of the departments and research institutes involved.

Funding The projects were supported by a number of different funding agents. The most important sources were the Technology Development Centre of Finland (TEKES), the Academy of Finland (SA), the Ministry of Education (OPM), Åbo Akademi (ÅA) and the Technical Research Centre of Finland in Oulu (VTT/TKO). The total funding for the project, including a calculated overhead for personel as a contribution from Åbo Akademi and the Technical Research Centre of Finland, was approximately FIM 5 million .

TEKES funding was FIM 1.4 million, most of it for equipment and for planning and building the Hathi-2 multiprocessor system. The Academy of Finland contributed by funding the basic research project on formal methods, to a total amount of FIM 730 000. The Ministry of Education supported the research project through its program for intensifying the Ph.D. education in Information Technology: Ralph-Johan Back was employed as co-ordinating professor for the subprogram on Ph.D. education in Computer Science (January 1986 – July 1988), and a number of other researchers in the Hathi–project were also partly supported by the subprograms in Computer Science and in Computer Technology. The internal funding of Åbo Akademi and the Foundation of Åbo Akademi was also important, both for employing temporary personnel in the research projects and for aquiring equipment.

Personnel The following researchers have participated in the Hathi–project. The source of financial support for each person is given in Appendix 1, together with the extend of their involvement.

Åbo Akademi:

Laboratory manager Mats Aspnäs, M.Sc.
Professor Ralph–Johan Back, Ph.D.
Lecturer Patrik Eklund, Ph.D.
Assistant Eeva Hartikainen, Ph.Lic.
Research assistant Viking Högnäs, M.Sc.
Assistant Antti Raunio, M.Sc
Assistant professor Kaisa Sere, Ph.Lic.
Research assistant Hong Shen, M.Sc.
Assistant Ulla Solin, M.Sc.
Assistant Eva Söderholm, M.Sc.

Assistant professor Joakim von Wright, Ph.Lic. (Swedish School of Economics and Business Education in Vaasa)

VTT/TKO:

Section head Kari Leppälä, M.Sc.(Eng)
Research scientist Kari Pehkonen, Ph.Lic.
Research assistant Kyösti Rautiola, M.Sc.(Eng)
Senior research scientist Tapani Äijänen, Ph. Lic.

A number of students at the Department of Computer Science at Åbo Akademi have also been involved in the project, most of them for the purpose of writing their M.Sc. theses. Many of these students have also been employed in the project for a longer or shorter period:

John Aspnäs
Jonny Boman
Edward Eriksson
Kristian Frantz
Jens Granlund
Jukkapekka Hekanaho
Stefan Levander
Tor–Erik Malén
Yngve Nyman
Dan–Johan Still
Lena Ståhl
Marina Walldén
Patrick Waxlax

Related activities Seminars and summer schools were organized in order to support the research done in the Hathi–project. In August 1987 a two–week Nordic Summer School on Design of Distributed Systems was organized in Nässlingen, Sweden, by Ralph–Johan Back, Dag Belsnes (Norway), Björn Pehrsson (Sweden) and Anders Ravn (Denmark). It was funded by the Nordic Research Courses organization. The number of participants was around 60.

Ralph–Johan Back organized in co–operation with R. Kurki-Suonio from Tampere University of Technology a one–week International Summer School on Formal Methods in Programming in Orivesi, Finland, July 1988. The summer school was held in connection with the ICALP conference in Tampere 1988. The number of participants was approximately 60.

A number of research seminars and courses on parallel computing and formal methods have also been held at Åbo Akademi and also at other places.

Continuation of the Hathi-project The Hathi–project has been followed up by a whole research program on parallel computation, now also including neural networks. This program, called *FINSOFT III: Parallel computation and neural network*, is part of the large FINSOFT research program funded by TEKES. The duration is April 1988–March 1991. Ralph–Johan Back was asked by TEKES to plan this subprogram, and is the director and scientific leader of it. Ph.D. Atte Kortekangas (VTT/TIK) is the co-ordinator for this program. FINSOFT III consists of 12 different research projects (two projects each in Åbo Akademi, VTT/TKO, Tampere University of Technology and in University of Helsinki and one project in VTT/TIK, Technical University of Helsinki, University of Jyväskylä and in Lappeenranta University of Technology). The total budget for this subprogram is approximately FIM 18 million, for the whole 3–year period.

The *Millipede-project* is a continuation of the multiprocessor research in the Hathi-project, with a more focused aim. It is one of the FINSOFT III projects. It was started in April 1988 and ends in March 1991. The total TEKES funding for the 3-year period is approximately FIM 3 million (the actual funding is decided on an annual basis). Ralph-Johan Back is project leader and Mats Aspnäs is administrative leader for this project. The main emphasis is on building a user-friendly interface to the multiprocessor system, which hides the details and machine dependencies of the underlying hardware.

Formal methods are studied in the *Centipede-project*, also funded by TEKES and part of the FINSOFT III research program. It was started April 1988 and will end in March 1991. Ralph-Johan Back is project leader and Kaisa Sere the administrative leader of this project. The project continues the work on formal methods for constructing parallel programs and aims at constructing a graphical environment for formal derivations of programs. Total funding is approximately FIM 2.2 million (the actual funding is again decided on an annual basis).

Overview The report describes the main research areas and results of the Hahti-project. The research on multiprocessor technology is described in Chapter 2, the research on formal methods is presented in Chapter 3 and the research on parallel program applications is described in Chapter 4.

A list of publications prepared during the Hathi-project is presented in Chapter 5, with short abstracts of their contents. As the Hathi-project partially overlapped with its successors, the Millipede and the Centipede projects, it was not possible to make a sharp distinction between what publications really should be attributed to what project. The topic of study in these projects is, however, the same, so we decided to list all publications from these projects up til September 1989, with the understanding that a part of the research reported in 1989 really should be attributed to either Millipede or Centipede.

Chapter 2

The Hathi-2 Multiprocessor System

The design and construction of the Hathi-2 multiprocessor system was a central part of the Hathi project. The system was specified jointly by the Technical Research Center of Finland (VTT/TKO) in Oulu and Åbo Akademi, while the hardware design and construction was done by the former. The system software for the Hathi-2 system has been mainly constructed at Åbo Akademi.

Hathi-2 is a reconfigurable general purpose multiprocessor system consisting of 100 32-bit IMS T800 transputers, 25 16-bit IMS T212 transputers and 25 IMS C004 crossbar switches. The system can be characterized as a loosely coupled MIMD multiprocessor, with a reconfigurable distributed interconnection network and a modular design. The total parallel computing power is 150 MFLOP/1000 (Risc)MIPS, which puts the system in the super computer class. The Hathi–2 system architecture is described below. A more detailed description of the Hathi-2 architecture can be found in [Aspnäs et al. 89b] and [Pehkonen 89]. The distributed switching network is described in [Äijänen 88a].

Hathi-2 board Hathi-2 consists of 25 identical boards, each containing four T800 transputers, one T212 transputer and one 32 link crossbar switch. The T800 transputers are connected pairwise to each other via one of the four communication links. The three remaining links are connected to the crossbar switch (see Figure 2.1). Three links from each switch are used as I/O links, i.e., to connect users host computers and peripheral units to the system. The remaining 16 links from the crossbar switch are used to connect the transputers on the board to transputers on other boards, as explained below.

The C004 crossbar switch is controlled by the T212 transputer via a control link. Another link on the T212 is connected to the crossbar switch and can be connected via the switch to any other transputer link. The two remaining links on the T212 are used to connect the T212 transputers into a ring, thus forming the distributed control system.

The switching network The crossbar switches on the Hathi-2 boards are connected to each other in a static torus connection. Each pair of neighbouring boards are connected by four links (see Figure 2.2). The crossbar switches form a distributed switching network for the communication links of the T800 transputers. It enables the system to be reconfigured by software, without making any changes in the actual hardware connections.

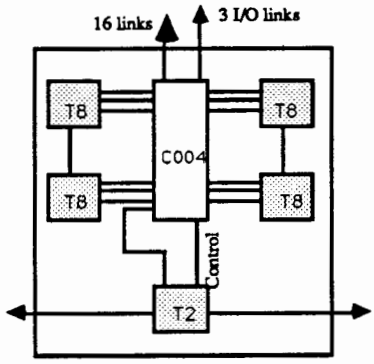

Figure 2.1: Hathi-2 board architecture

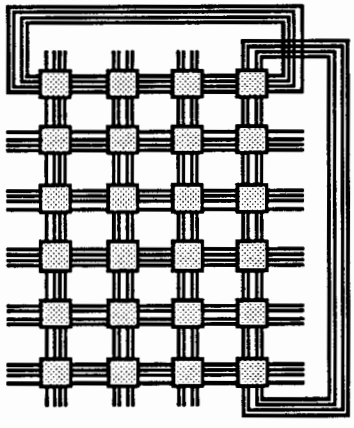

Figure 2.2: Hathi-2 board connections

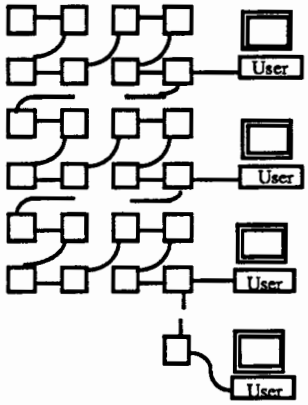

Figure 2.3: Partitioning the system

Use of Hathi-2 Hathi-2 is used as a back-end computing resource. The user edits, compiles and links his programs on a host computer (a Sun workstation). The program can then be loaded on to the multiprocessor system and executed.

The Hathi-2 system can be shared between a number of simultaneous users by partitioning it into several smaller independent multiprocessor systems (see Figure 2.3). All users are allocated a separate partition which is independent of all other partitions. A user has full control over his own partition, but can not interfere with other users.

The control system The T212 transputers are connected to each other in a ring, thus forming a separate control system which controls the switching network (see Figure 2.4). The control system is totally independent from the rest of the system. The only connection between the user and the control system is via a link connecting one T212 transputer to the users host computer. The user can request system services by sending commands to the control system via this link.

The control system has two main tasks: to control the distributed switching network and to monitor the activities in the system. The Hathi-2 architecture contains hardware dedicated to monitoring the resource utilization in the system. The monitoring hardware consist of a CPU load meter which measures the CPU utilization by observing the bus activity and four FIFO buffers connecting the T800 transputers on a board to the controlling T212 transputer. The FIFO buffer can be used for sending reports about resource utilization from the T800 to the T212 without affecting the communication links.

The control system also contains an interrupt subsystem implemented using the transputers EVENT interrupt. A processor in the control system can send an interrupt signal to all processors in the same partition. This interrupt is used in the monitoring system to generate a synchronizing signal which divides the time into short time intervals. The CPU and link utilization are measured for each interval and reported to the user.

Scalability The hardware design for Hathi-2 is easily scalable. One can build a much bigger Hathi-2 system by simply using more of the Hathi-2 boards. A 1000 processor system could thus be built with 250 of these boards. The system is also relatively cheap. Because of the distributed architecture, all the boards are identical, and the cost of the board is dominated by the components, i.e. the processors (T800 and T212), the switch and the memory. This means

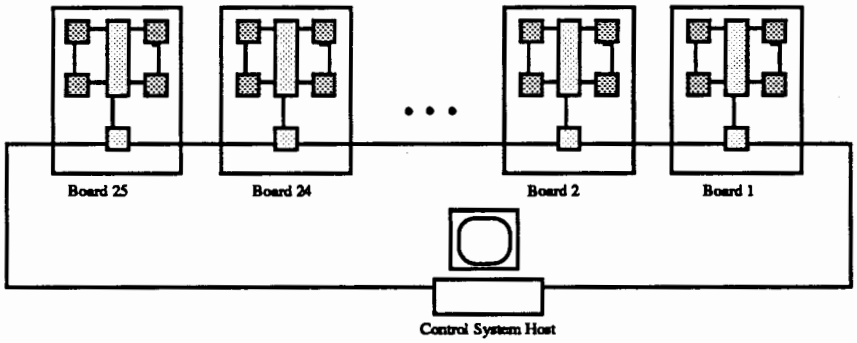

Figure 2.4: The control system

that the hardware cost of a Hathi-2 multiprocessor system is more or less linear in the number of processors, the amount of memory and the number of peripheral units that one uses.

System software The system software developed in the Hathi-2 project is concentrated on two main areas: configuration software to control the distributed switching network and monitoring and animation software to observe the behaviour of a parallel program executing on the system.

The system software for Hathi-2 executes on the control system, which is dedicated to this purpose. The system software utilities are all based on a general message-passing system implemented on the control system. The message-passing system consists of a small communication kernel process executing on all control processors. The communication kernel handles transparent message passing between the processes in the control system. To this message passing kernel, a number of *service processes* can be attached.

The main subprojects in this research area are described in more detail below.

2.1 Hathi-2 hardware architecture

The Hathi-2 hardware system described above was constructed at the Technical Research Center in Oulu, by a design team including Kari Leppälä, Kari Pehkonen, Kyösti Rautiala, Lauri Ståhle and Tapani Äijänen. The hardware design started in autumn 1986, and the Hathi-2 system was delivered to Åbo Akademi in March 1988.

The original hardware specification had aimed at 128 T414 processors with 1 MB of central memory each. When the T800 floating point transputers where announced, it was decided that the the T414 transputers were to be exchanged for the more efficient (but also more expensive) T800 transputers. The number of T800 processors then had to be limited to 100 and only 256 KB of memory per processor could be afforded. A smaller Hathi-2 system, with 16 processors, was also built for the Technical Research Center in Oulu.

The main emphasis in the beginning of the project was on designing the interconnection network. The decision to build this as a distributed switching network made it possible to design the Hathi-2 system as a collection of identical boards. This in turn implies that the system has good scalability, so that one can build a bigger Hathi-2 system by simply using more of these boards. The use of Inmos C004 crossbar switches simplified this construction task considerably.

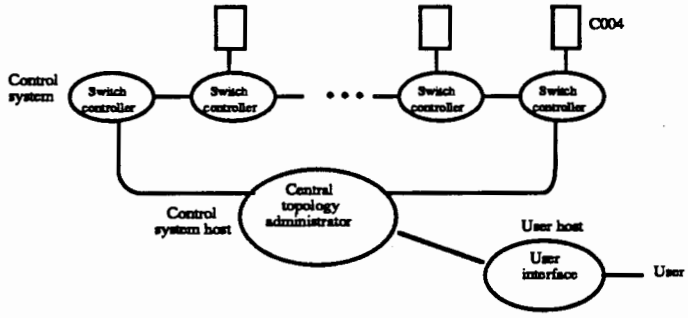

Figure 2.5: The configuration system software

Another focus in the design was on building hardware support for the monitoring system and for the interrupt system. The main idea was to build a separate hardware facility by which the processors could report on their activities without using the transputer channels. This makes it possible to monitor the multiprocessor system without interfering with the ongoing parallel computation.

The details of the hardware design are documented in more detail in [Leppälä and Rautiola 87, Äijänen 87, Pehkonen 88] while [Pehkonen 89] contains a general description of the Hahti-2 hardware architecture. The hardware project has generated one Ph.Lic. thesis in engineering (Kari Pehkonen) at the University of Oulu. A patent application on the switching network used in Hathi-2 has also been made.

2.2 Configuration of transputer systems

The initial version of the configuration software for Hathi-2, called the Transputer Network Topology administrator (TNT administrator) was constructed by Tor-Erik Malén (M.Sc. thesis). A prototype of the configuration system was developed before the Hathi-2 system was delivered to Åbo Akademi using a Hathi-2 architecture simulator.

The switching network in Hathi-2 consists of 25 of crossbar link switches, each controlled by an independent control processor. The configuration software is thus a distributed program that executes on all the processors in the control system.

The configuration software consists of two components: a *switch controller process*, which executes on each control processor and a *central topology administrator*, which executes on the control system host transputer (see Figure 2.5). The user interface to the system consists of a program executing on the users host transputer and which presents a menu of possible processor network topologies to the user. The user can also design his own topologies if he so wishes.

The central topology administrator keeps information about all established link connections. It also keeps information about how the system is partitioned among the users, so that it can check that a user does not try to change the topology of another users partition. When a user wants to change the interconnection of the processors in his partition, a description of the desired configuration is sent via the message passing system to the central topology administrator. The central topology administrator calculates which crossbar link connections are needed to establish this topology, using a simple shortest path algorithm. The algorithm takes into consideration the limitations imposed by the transputers architecture and the architecture of the Hathi-2 system.

The result of this calculation is a table which contains all the link connections that have

to be established for each crossbar switch in order to connect the processors into the desired configuration. The table is distributed to the respective switch controller processes via the message passing system. The switch controller process is constantly waiting for link connection commands, which it sends directly to the crossbar switch.

Even though there exists configurations that can not be established due to the architecure of the Hathi-2 system, the configuration system has proved to work well in practice. The most severe limitation is that there is only four intra-board links connecting any pair of neighbouring boards in the static torus connection between the boards. The commonly used regular interconnections, like trees, grids, toruses etc. can all be established using the TNT administrator.

Theoretical work on the configuration of transputer systems has also been done by Patrik Eklund and Tor–Erik Malén [Eklund 88a, Eklund and Malén 88]. They have been working on heuristic approaches to establishing the necessary connections in a distributed switching network, and more generally on how to map a process graph onto a configurable processor graph. This line of investigation is being continued in the Millipede project by Hong Shen [Shen 89b, 89c, 89d].

2.3 Monitoring and debugging multiprocessor systems

The initial version of the monitoring system was implemented by Stefan Levander (M.Sc. thesis) in the Hathi-project. The system monitors the utilization of resources in the multiprocessor system during program execution. The monitoring system is used for finding bottlenecks in parallel programs and to provide information about the load balance of programs. Monitoring is done by observing the CPU and link activity in the transputer network. The monitoring software is based on the monitoring hardware built into the Hathi-2 architecture, which makes it possible to monitor the system without introducing any substantial overhead on the main computation. The hardware consists of the CPU load meter, the interrupt subsystem and the FIFO buffers connecting each T800 processors with the control processor on the board.

The time during which monitoring is done is divided into short time intervals by a global clock pulse generated by the control system using the interrupt subsystem in Hathi-2. For each time interval, the monitoring system measures the utilization of the hardware resources in the system and sends the measured values to the control system host.

The CPU load meter registers are read by the control transputers for each time interval, and give a direct indication of how much computation a processor has performed during the interval. The FIFO buffers are used in the monitoring system as a link-independent communication path between the T800 transputers and the control system. The transputer links cannot themselves be used for sending monitoring data, as this would change the behavior of the monitored program.

The monitoring system consists of four types of processes: a *monitor data multiplexer (mux)* which executes in parallel with the user program on the processors in the users partition, a *monitor controller process* which executes on each processor in the control system, a *monitor data collector process* which executes on the control system host, and a *monitor data presentation process* which executes on the users host transputer (see Figure 2.6).

The time during which monitoring is done is divided into short time intervals (typically 100 to 500 milliseconds), and for each interval the monitoring system records the percentual utilization of the CPU, the number of bytes transmitted over a link and the time a process has spent waiting for a communication to take place. This information is gathered by the monitor data multiplexer, which receives reports of resource usage from the users program. The multiplexer process sends the accumulated information for each time interval to the monitor control process via the FIFO buffer.

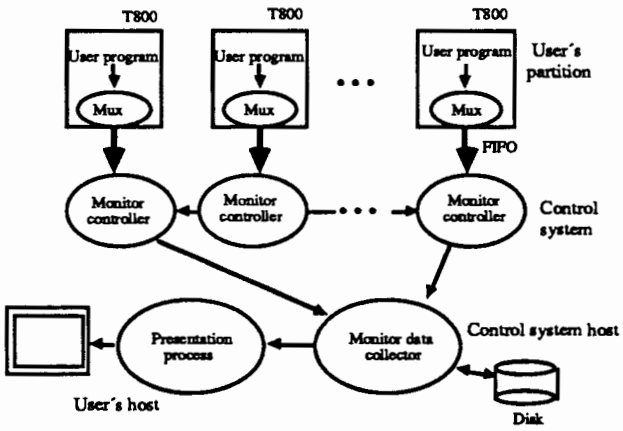

Figure 2.6: The monitoring system

The monitor control process forwards the monitoring data to the collector process via the message-passing system. The collector process writes the data to a file, from which it can be read by the presentation process. The monitoring data is presented to the user as the average utilization of the transputers links and CPUs.

2.4 Parallel Algorithm Animation

Algorithm animation means visualization of program execution. The aim is to show the dynamic working of an algorithm during execution, in graphical form. Animation is a powerful tool for program debugging and for identifying excecution bottlenecks. This is especially true for parallel programs, as animation can give an overall perspective of a parallel execution that can be very difficult to grasp in any other way.

The research carried out in the Hathi project has produced a non-interferent method for animating parallel algorithms [Solin 86, 88, 89]. The method is based on uniformly delaying the execution of all processes in the program, so that execution runs at a speed that is convenient for the user. The delaying technique makes it possible to simulate the execution of a many-processor parallel program on a single processor.

The main result of this project is an animation environment for debugging, testing and analysis of Occam programs. A prototype system has already been built in the Hathi-project, and is now running on the Hathi-2 Sun-workstation host. The prototype implementation has been done by Ulla Solin, Lena Ståhl, Yngve Nyman (M.Sc. thesis) and Marie-Louise Lindström (M.Sc. thesis).

The animation is done by inserting commands into processes in the Occam program, which send messages about their present state of execution to the animation process. The animation process receives these messages and translates them into graphical commands that update the screen. The user has to specify on which points in the execution the state of the program should be reported. The user also has to describe how each state should be represented graphically in the animated picture. This is done by a graphical tool by which the user can draw the pictures of which the animation consists.

2.5 Continuation in Millipede

The Hathi-2 system is being developed further within the Millipede-project. The amount of memory is increased to 1.25 Mb per processor, giving the system a total of 125 Mb of central memory. A number of peripheral units, such as disk storage units and graphics controllers will also be connected to the system.

The work on configuration and monitoring of multiprocessor systems is being continued also in the Millipede project, as well as the work on animation of parallel programs (Hong Shen (Ph.D. work), Thomas Långbacka (M.Sc. work), Tony Riissanen (M.Sc. work), Mikael Norrbo (M.Sc. work)). The research on transputer hardware architecture is being continued at the State Research Center withing another project (PIRE), with a special emphasis on using transputers for real-time applications.

A central task in the Millipede project is to construct an integrated programming environment for Hathi-2. The purpose is to hide the hardware structure from the programmer so that he only has to work with a simple conceptual parallel programming model. This model is basically the process graph model of Occam, where the graph nodes are the logical (possibly nested) processes and the edges are the logical communication channels.

This enviroment will integrate the existing programming tools under a common graphical user interface, allowing the programmer to specify the process structure of a distributed program in graphical form. The programmer can combine processes into tasks, which are then automatically allocated to the processors in the system by a mapping utility. The mapping utility also automatically configures the multiprocessor system to the desired topology. The monitoring utility and the animation utility will also be integrated into the programming environment. The results from monitoring and animation are presented to the user in a graphical way, where the presentation is based on the process structure of the distributed program. The programming environment is being designed by Mats Aspnäs and Ralph-Johan Back [Aspnäs and Back 89], and is implemented by Jens Granlund (M.Sc. work) and Henrik Gullberg (M.Sc. work).

The animation research is being continued in the Millipede project in order to provide graphical monitoring and debugging of programs running on many processors in the Hathi-2 system. Animation on the Hathi-2 system is implemented using the same hardware features as the monitoring system, i.e. the FIFO buffers. In the animation system, the data sent from the transputers to the control system contain information that controls the graphical animation of the executed program. This data is interpreted as graphical commands, which are executed by an animation process and yield a graphical illustration of the program execution.

Chapter 3

Formal methods for parallel program construction

The second main objective of the Hathi–project was to develop formal methods for deriving correct and efficient parallel programs. The approach taken has been to combine the *action systems* approach developed by Ralph-Johan Back and Reino Kurki-Suonio [Back and Kurki-Suonio 83, 88a] with the *calculus for program refinements* developed by Back in his Ph.D. thesis [Back 78]. This gives a method for constructing parallel programs by stepwise refinement. One starts by constructing a high level and possibly sequential specification of the required program. This is then transformed by a sequence of successive correctness preserving transformations into an executable and efficient multiprocessor program that is guaranteed to satisfy the original specification by construction.

Refinement calculus The stepwise refinement method for sequential programs was formalized in [Back78, Back80]. The basic notion in the refinement calculus is a relation of refinement between program statements: statement S is said to be *(correctly) refined* by statement S', denoted $S \leq S'$, if

$$P[S]Q \Rightarrow P[S']Q, \text{ for every } P \text{ and } Q \ .$$

Here $P[S]Q$ stands for total correctness of S with respect to precondition P and postcondition Q. An equivalent characterization in terms of weakest preconditions is that

$$\mathrm{wp}(S, R) \Rightarrow \mathrm{wp}(S', R), \text{ for every } R \ .$$

The refinement relation is reflexive and transitive. Hence, if we can prove that $S_0 \leq S_1 \leq \ldots \leq S_{n-1} \leq S_n$, then $S_0 \leq S_n$. This models the successive refinement steps in program development. S_0 is the initial high level specification statement and S_n is the final executable program that we have derived through the intermediate program versions $S_1, \ldots, S_{n-1}$. Each refinement step preserves the correctness of the previous step, so the final program version must preserve the correctness of the original specification statement. Specifications are treated as generalized statements in the refinement calculus and may include higher level and not necessary computable notions.

The refinement relation is monotonic with respect to the usual statement constructors. This means that if T is a statement where S occurs as a substatement, i.e. $T = T(S)$, then

$$S \leq S' \Rightarrow T(S) \leq T(S').$$

```
var x.1; ...; x.n : integer;
begin
      x.1, ..., x.n := X.1, ..., X.n;
      do
      [] x.1 > x.2 → x.1, x.2 := x.2, x.1
      :
      [] x.(n − 1) > x.n → x.(n − 1), x.n := x.n, x.(n − 1)
      od
end
```

Figure 3.1: Sorting as an action system

In other words, we may replace a substatement by its refinement in any program context. This justifies the top-down method of program constructions in the refinement calculus.

The refinement calculus was originally used to formalize different aspects of the informal stepwise refinement method, such as the use of specifications as program statements, procedural abstraction, the piecewise refinement of data representations in programs, the use of context information to justify refinement of program components and the use of program transformation rules. The foundations of this calculus is further developed in [Back 87a]. The calculus is extended in [Back87b] to procedure mechanisms and in [Back 88b] the original method for data refinement is further studied and extended. Other recent work on the refinement calculus is [Morgan et al 88, Morris 87].

Action systems The *action system* formalism for describing and analyzing parallel and distributed computations was introduced in [Back and Kurki-Suonio 83, 84a]. A recent survey is given in [Back and Kurki-Suonio 88a]. The behaviour of parallel and distributed programs is described in terms of actions which processes in the system carry out in co-operating with each other. Several actions can be executed in parallel, as long as the actions do not have any variables in common. The actions are atomic: if an action is chosen for execution, it is executed to completion without any interference from the other actions in the system.

More precisely, an *action system* $\mathcal{A}$ is a collection of *actions* $\{A_1, \ldots, A_m\}$ on some set of *state variables* $x = \{x_1, \ldots, x_n\}$. Each action A_i is of the form $g_i \rightarrow S_i$ where the *guard* g_i is a boolean condition and the *body* S_i a sequential, possibly nondeterministic statement on the state variables. An *initialization statement* S_0 assigns initial values to the variables x.

The behaviour of a *sequential action system* $\mathcal{A}$ is that of the guarded iteration statement [Dijkstra76]

$$S_0; \textbf{ do } A_1 \; [] \; \ldots \; [] \; A_m \; \textbf{ od}$$

on the state variables x. This is the simplest possible semantics for an action system.

Figure 3.1 gives an example of an action system. This program will sort the n integers $X.1, \ldots, X.n$ in ascending order. We can look upon this program as an action system with an initialization statement and $n - 1$ sorting actions. The program obviously terminates in a state where the array x is a permutation of the original array and where $x.i \leq x.(i + 1)$ for $i = 1, \ldots, n - 1$.

Parallel action system Action systems can, however, also be executed in parallel. The main idea is that two or more actions can be executed in parallel as long as they do not share

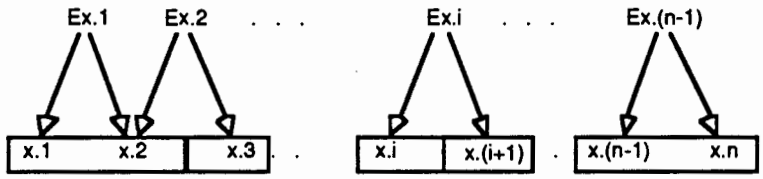

Figure 3.2: Distributed sorting

any common variables. A parallel execution of an action system will then give the same result as a sequential execution (disregarding possible fairness constraints).

In a *concurrent action system* each *action* is assigned to some specific process. A variable that is refered to by at least two different actions in two different processes is *shared*, while a variable that is refered to only by actions in one process is *private* to that process.

The actions in the system are executed in parallel, with the restriction that all actions are *atomic*: two actions that share a common variable may not execute at the same time. Two actions belonging to the same process may not execute in parallel either, even when they have no variables in common.

The example sorting program can be interpreted as a concurrent action system by assuming the existence of $n-1$ processes and assigning each of the $n-1$ actions to a process of its own. The variables $x.1, \ldots, x.n$ are shared among the processes (each variable is shared by at most two processes).

In a *distributed action system* each *variable* is assigned to a unique process. An action is *shared*, if it refers to variables in two or more processes. If all the variables refered to by an action belong to the same process, then the action is *private* to that process. A shared action is assumed to be executed jointly by all the processes sharing this action. The processes are therefore synchronized for execution of the shared action. Shared actions also provide communication between the process: a variable of one process may be updated in a way that depends on the values of variables in other processes involved in the shared action.

The actions in a distributed action system are executed in parallel, again with the restriction enforced by atomicity. This means that no two actions involving a common process may execute simultaneously. In other words, a process may only participate in one action at a time.

We can interprete the sorting program as a distributed action system by assigning one element of the array x to each process. This is shown in Figure 3.2, except that we choose to put the two boarder elements on the same processor. Each action in the system, except for the two outermost ones, is then shared between two processes. The outermost actions, $Ex.1$ and $Ex.(n-1)$ are private to the boarder processes.

Stepwise refinement of parallel programs The use of action systems permits the design of the logical behaviour of a system to be separated from the issue of how this system is to be implemented. The refinement calculus provides a framework within which one can derive sequential action systems. These can then be implemented in a parallel fashion, either as concurrent or as distributed action systems. Atomicity guarantees that a parallel execution of an action system gives the same results as a sequential and nondeterministic execution.

The decision whether the action system is to be executed in a sequential, concurrent or distributed fashion can be done at a later stage, after the logical behaviour of the action system has been designed (a similar separation between logical behaviour and implementation is also made in [Chandy and Misra 88]). The construction of parallel programs can thus be done within a single unifying framework.

3.1 Action systems

The action system approach has been developed in a joint project between Ralph-Johan Back and Reino Kurki-Suonio (Tampere University of Technology) which started in 1983. The original aim was to develope formal methods for constructing distributed computer systems. These systems present programming problems that are often very difficult, and where the pay-off of formal methods therefore is potentially large. The advantage of the action system approach is that it becomes relatively easy to prove properties of the whole system execution, and standard techniques based on partial and total correctness and temporal logic can be applied directly to the verification of distributed systems [Back and Kurki-Suonio 83, 84a, 84b, 85, 88b]. The work on action systems continued in the Hathi-project, focusing on two main issues: fairness in parallel execution and efficient parallel implementations of action systems.

The fairness properties that are useful for reasoning about liveness properties of action systems in a sequential execution are not the same fairness assumptions that one can reasonably assume that a parallel and distributed implementation will guarantee. The relationsship between these fairness properties were studied in detail in [Back and Kurki-Suonio 88a, 88b].

The simplification of program design that the action system formalism permits is partially offset by a more complicated implementation problem. Hovewer, it turns out that action systems can be quite efficiently implemented on local area networks employing a shared broadcasting channel. The problem of implementing action systems was studied in co-operation with Eeva Hartikainen and Patrik Eklund [Back and Eklund 84, Back et al 85, Eklund 84c] in a previous research project. This work was then continued in the Hathi-project [Aspnäs 87, Hartikainen 88, Aspnäs et al 89a]. Eeva Hartikainen got her Ph.Lic degree for her work on modelling broadcast implementations of action systems.

For point-to-point networks it is more difficult to find efficient distributed implementations of action systems. However, as shown in joint work by Ralph-Johan Back and Kaisa Sere, efficient implementations are possible provided that certain reasonable restrictions are accepted [Back and Sere 89]. These restrictions are of the same nature as those that forbid the use of output guards in CSP and Occam, and can thus be considered acceptable. This work is now being continued in the Centipede project. There is already one implementation of (unrestricted) action systems based on a protocol by Bagrodia (University of Texas), built by Lena Ståhl [Ståhl and Back 89]. However, this implementation is not very efficient, and work is in progress on more efficient implementations of (restricted) action systems.

3.2 Calculus of program refinement

The refinement calculus described above has become quite intesively studied during the last three years, especially in Åbo Akademi, the United Kingdom (Oxford, Glasgow) and in the Netherlands (Groningen). This work has also resulted in quite a number of further developments of this calculus, and in a renewed interest in the calculus itself and in its potentials.

The mathematical basis for the refinement calculus has been developed further in the Hathi-project by Ralph-Johan Back. The main task has been to extend the formal framework to

handle difficult program features such as unbounded nondeterminism, full recursion and procedures with parameters [Back 87a, 87b, 88a]. The original method for data refinement has also been extended to permit non–functional data refinements [Back 88b, 89a].

3.3 Stepwise refinement of parallel and distributed systems

One of the main original incentives to develope the action system approach was that it made it possible to construct distributed systems by stepwise refinement, by a method now known as *superposition*. This approach was studied in 1983 by Ralph–Johan Back and Reino Kurki–Suonio (see [Back and Kurki-Suonio 89] for a survey).

The interest for this method was renewed in the Hathi–project, when it was realized that it could also be applied to proving total correctness of action systems within the refinement calculus. Joint work by Ralph–Johan Back and Kaisa Sere on developing this approach has proved to be very fruitful [Back 89b, Back and Sere 88a, 89a, 89b, Sere 87b, Sere 88a, 88b, 88c] . A systematic method for stepwise refinement of parallel programs within the refinement calculus has been developed, and it has been applied to a number of quite difficult parallel and distributed algorithms, including matrix multiplication and solving linear systems of equations on a ring of processors, and the implementation of processor farms on different kinds of processor topologies. Kaisa Sere got her Ph.Lic. degree from this project [Sere 88c], and is now working on her Ph.D. on the same topic.

A drawback of the method for stepwise refinement of parallel programs has been that it only preserves input-output correctness. This is sufficient for *multiprocessor algorithms* where the main purpose is to parallelize sequential algorithms in order to gain speed. However, *reactive systems*, for which the behaviour of the system during execution is also important, tend to arise quite easily when breaking a single parallel program into modules. This kind of distributed programs could not be handled in the original refinement calculus. Recently Ralph-Johan Back has, however, shown how to use the technique previously developed for data refinement for the stepwise refinement of reactive action system [Back 89c]. The actual method generalizes and formalizes an earlier method proposed by Leslie Lamport. Hence, the weakest precondition technique of Dijkstra, on which the refinement calculus is based, becomes applicable also to the stepwise refinement of reactive systems.

3.4 Continuation in Centipede

The work on formal methods for program construction is beeing continued in the Centipede–project. The work on the mathematical basis of the refinement calculus has continued between Ralph-Johan Back and Joakim von Wright. The main result here has been the construction of a very general lattice-theoretic basis for the refinement calculus [Back and vWright 89a, 89b, 89c, 89d]. The mathematical basis for the calculus is simplified considerably while at the same time the applicability of the calculus is very much extended. As a side product, this framework also provided a considerable generalization of Dijkstra's original weakest precondition technique. Joakim von Wright has got his Ph.Lic. degree in this project [vWright 89b] and is continuing his work for the Ph.D. degree.

One of the main problems in constructing programs by stepwise refinement is to master the sheer number of details and administer the large number of derivation steps and successive program versions that arise in a derivation. In addition, one needs to show that the successive program refinements do in fact preserve correctness, i.e. that no errors are introduced during

the refinement. When doing refinements by hand, it is very easy to make simple clerical errors which lead the derivation astray.

A central tasks in the Centipede–project is therefore to construct a workstation–based programming environment to support the stepwise refinement method and mechanize as much as possible of the derivation process. The environment will support the formal derivation of sequential and parallel algorithms, as well as derivation of reactive programs. Verification aspect of the methodology will be supported by proof assistants like the Cambridge LCF and Higher Order Logic (HOL) systems [Back and vWright 89c].

The formal derivation environment will be complemented by a simulation and animation environment where intermediate versions in the program derivation can be executed and their efficiency (speed, degree of parallellism, efficiency bottlenecks etc.) studied. A compiler which takes an action system and compiles it to occam for execution in the Hathi–2 multiprocessor will also be implemented, as described above. The environment will thus provide an integrated toolset for the whole derivation process, from initial high-level specification via formal derivations and actual testing of intermediate versions to a multiprocessor application running on a transputer system.

A prototype of the program refinement environment is being constructed by Ralph-Johan Back, Jan Komorowski and Patrick Waxlax (M. Sc work), while Jukka-Pekka Hekanaho (M. Sc work) is studying an alternative implementation of the environment, based of the program transformation system Refine on Sun-4. Kaisa Sere is working on a collection of program refinement rules thatr would be included in the system (integrating these rules with another collection of rules designed at Oxford University/PRG [Sere 89]. Dan-Johan Still (M. Sc work) is working on implementing the simulated parallel execution environment for action systems. Work is also in progress in Centipede on building an efficient implementation of action systems for the Hathi-2, by Peter Dahl (M.Sc work).

Chapter 4

Multiprocessor applications

One of the main goals in the Hathi-project was to experimentally try out the efficiency of parallel computation in practical applications. To this end, a number of application projects were initiated, some internal to the department and some in co-operation with other research institutes. Most of these applications turned out to be quite successful and deliver the efficiencies that were expected. The applications were initially carried out on a smaller 16 transputer system (Hathi-1), and then ported to the larger Hathi-2 when it became ready, in order to measure speedups and efficiencies when more masssive parallellism was available. Some of the applications are now being developed further as independent projects, for instance the fluid dynamic modelling and the real-time transformation of satellite pictures.

Programming multiprocessor applications An efficient implementation of a computational problem on a MIMD-type multiprocessor system does not necessarily follow the same ideas as an implementation on a sequential processor. Generally, it is not possible to construct a parallel solution using the same methods as in a sequential solution. To write a parallel implementation of a problem on a multiprocessor system, the programmer needs insight in the problem to be able to decompose it into a number of parallel processes.

All communication in the parallel solution to a problem can be considered as overhead introduced by the decomposition. On most existing multiprocessor systems, communication is slow compared to computation. To get an efficient parallel implementation, the processor must therefore perform a sufficient amount of calculations for each communication. Generally, this means that one should try to avoid communication in parallel algorithms, even at the cost of additional computation.

Let the execution time for a program executed on N processors be denoted by T_N. The speed-up factor S_N for a parallel program executed on N processors is then defined as

$$S_N = \frac{T_1}{T_N}$$

and the efficiency E_N of a parallel program executed on N processors is

$$E_N = \frac{S_N}{N}$$

The sequential program that we compare the parallel program to should be implemented using the best known sequential algorithm for the problem. This is not always possible as it might require an extensive amount of programming. At the very least, the sequential program

should not be constructed by executing the parallel algorithm on one processor using timeslicing between the parallel processes. In that case, the execution time for the sequential algorithm will contain overhead caused by the scheduling of the parallel program on one processor, and the measured speed-up will not be accurate.

In the ideal case, the speed-up S_N is equal to N, the number of processors used. More important is that the speed-up grows linearly with an increasing number of processors. However, when we increase the number of processors, the amount of work per processor will decrease, which will have a negative effect on the speed-up. When increasing the number of processors we must therefore also increase the problem size, so that the amount of computation per processor remains constant. This means that a problem should be big enough in order to gain efficiency from a parallel implementation. This fact can be clearly seen in the applications described below. For a fixed problem size, there exists a limit where additional processors do not result in better performance. However, if the size of the problem can be scaled up, by increasing the amount of data or by increasing the quality of the solution, linear speed-ups can be acheived for a large class of computational problems.

4.1 Chemical reactor, heat transfer and fluid flow modelling

The chemical reactor, heat transfer and fluid flow modelling applications were developed in co-operation with the Process Design and the Heat Engineering Laboratories in the Department of Chemical Engineering at Åbo Akademi. The simulation of a chemical reactor was carried out by Tom Björkholm (M.Sc. thesis, Eng.), who used the so called processor farm approach. See also [Kilpinen et al 89]. A set of heat transfer and fluid flow problems was solved by Pekka Kuusela (M.Sc. thesis), Tor-Erik Malén and Göran Öhman [Öhman et al 88].

Problem description The reactor process modelled was a packed-bed two phase (gas-solid) chemical reactor for iron ore reduction. One of the objectives was to design transient (batch) experiments in order to obtain accurate parameter estimates in the reduction kinetics model. Another objective was to use the model for on-line simulation of blast furnace.

Simulation of the packed bed reactor involves solution of 30 ordinary differential equations (ODEs). The CPU time consumed on a micro VAX II computer is on an average between one and ten minutes depending on the problem formulation.

The heat transfer and fluid flow study started from the numerical solution of the two-dimensional Laplace equation, which describes the steady heat conduction in a solid plate, and advanced through the solution of the three-dimensional Laplace equation to the case of steady laminar fluid flow in a two-dimensional box at Reynolds numbers up to 20. Hereby the stream function-vorticity method was first applied and then the SIMPLER method.

The essential principles which were used in the parallel solution of these problems are illustrated by the physically simplest case, the two-dimensional heat transfer problem. Consider a solid square plate of size x_0 by x_0 which is thermally insulated on both sides. The temperature along the edges of the plate is given and is assumed to be fixed. The problem is to calculate the steady temperature distribution in the interior part of the plate. The problem is illustrated in Figure 4.1.

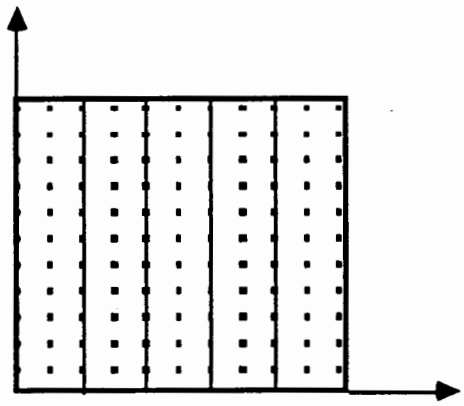

Figure 4.1: The discretized temperature field partitioned into slices

Mathematical model Mathematically, the steady temperature distribution $\theta(x, y)$ in a homogeneous solid plate is described by the Laplace differential equation

$$\frac{\partial^2\theta}{\partial x^2} + \frac{\partial^2\theta}{\partial y^2} = 0$$

for $0 \leq x \leq x_0, 0 \leq y \leq x_0$. The solution must also satisfy the initial boundary conditions describing the temperature along the edges of the plate.

This Laplace differential equations can be solved numerically by discretizising the field and calculating the temperature only in a finite set of points on the plate. Choosing an equal spacing h in both x and y directions, the temperature in an arbitrary point (x, y) is expressed as an algebraic equation

$$\theta(x, y) = [\theta(x + h, y) + \theta(x - h, y) + \theta(x, y + h) + \theta(x, y - h)]/4$$

The equation is transformed into a dimensionless form by introducing the dimensionless coordinates $X = x/h$ and $Y = y/h$. It can then be rewritten as

$$T(X, Y) = [T(X + 1, Y) + T(X - 1, Y) + T(X, Y + 1) + T(X, Y - 1)]/4$$

i.e., at each internal point in the plate the temperature must be equal to the arithmetic mean of the temperatures at its four neighbouring points.

The equation is solved numerically using the Gauss-Seidel method with overrelaxation, where the temperature of a point (X, Y) in iteration step $i + 1$ is calculated by

$$T_{i+1}(X, Y) = \omega(\sum T^{nb})/4 + (1 - \omega)T_i(X, Y),$$

where $\sum T^{nb}$ is the sum of the most recent temparature values in the four neighbouring points and ω is the overrelaxation factor. The temperature for each point in the plate is calculated iteratively until a sufficient number of iterations has been performed. Initially, each point is given a guessed initial value, often chosen to be 0. The accuracy of the solution depends on the number of iterations and the spacing h.

Parallel solution The problem was solved using *geometrical parallelism*, where the data domain of the problem is distributed among the processors and all processors execute identical code. The matrix describing the temperatures in the discrete points of the plate is divided into N equally large slices in the X-dimension, where N is the number of processors.

For all points that are not on the border between two slices, the processor holding this part of the matrix can compute the new temperature during an iteration step without communicating with any other processor. In order to calculate the temperature of a borderpoint, a processor has to excange information about the temperatures on the border line. Each processor has a copy of the neighbouring processors border values. For each iteration, the processors exchange the border values and then compute the new values for its own grid points.

Because a processor only needs to communicate with two neighbours, the processors were arranged in a ring. To be able to overlap communication with computation, high priority buffer processes were introduced, which take care of the communication independently of the calculating process. The root processor does not participate in the calculation, it only initiates the computation by sending out the initial values and accepts the results after the computation has finished.

Performance The algorithm was executed on 4, 8, 12 and 16 processors with a grid size of 20*20, 40*20, 60*20 and 80*20 points respectively. The number of iterations varied from 20 to 1000.

The results of the test runs showed an almost linear speed-up. The efficiency varied between 99 % (for four processors) and 80 % (for 16 processors). The tests also showed that by increasing the grid size and the number of iterations, the speed-up is approaching N, the number of processors.

This shows that these types of problems can be solved very efficiently on MIMD-type multiprocessor systems, given that the problem is large enough to be partitioned among a number of processor.

4.2 Real-time transformation of satellite pictures

The real-time satellite data transformation application was developed by Atte Kortekangas, Aarne Rantala, Antti Raunio and Dan-Johan Still [Rantala et al 89]. The problem originates from a project carried out jointly by the Tecnical Research Centre of Finland (VTT/TIK), Vaisala OY and the Finnish Meteorological Institute.

Problem description A polar orbiting NOAA-series satellite, used for weather forecasting, takes pictures of Scandinavia and sends the picture data down to the earth, where it is received. The received pictures are distorted due to the curvature of the earth, the eccentricity of the satellite orbit, the varying viewing angle of the camera etc. From the received raw data, the users want to produce pictures in some known cartographic projection, e.g. polarstereographic projection. The transformation from raw satellite data to a cartographic projection is a very computationally intense task.

Data is received from the satellite at a rate of about 133 Kbytes/s during an overflight, which as a maximum lasts about 12 minutes. The total amount of data received during an overflight can be up to about 115 Mbytes. A real-time system, i.e., a system that performs the transformation at the same time as the data is received, should be able to produce transformed images at a rate of about 88 Kbytes/s. In this application, the problem size is fixed and can not easily be scaled up. However, more important than to acheive a good speed-up in a parallel solution is to fulfill the stated real-time requirements.

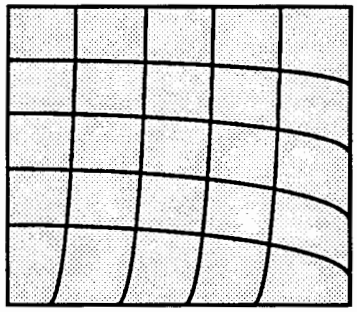

Figure 4.2: Raw data satellite data covered by the grid

Sequential solution Let the original picture received from the satellite be denoted by a matrix Q of pixel values and the transformed picture to be computed by a matrix P. The transformation from Q to P can be described by a set of pixel tuples $(x_q, y_q), (x_p, y_p)$, stating that the pixel positioned in (x_q, y_q) in the original data corresponds to the pixel in (x_p, y_p) in the transformed picture. The numerical value of the pixel does not change in the transformation, only the position of the pixel in the picture.

Because of neighbourhood preserving properties in the transformation, it is sufficient to calculate this exact correspondance between pixels in the raw data and pixels in the transformed image only for a small number of pixels, which form a sparse grid covering the data. The rest of the pixels are approximated by interpolation. The method is illustrated in Figure 4.2.

As soon as the orbit parameters for the overflight are received in form of a telex, the transformation for the pixels in the grid can be calculated. The grid size in our example is 32*32 pixels.

Parallel solution A prototype version of the satellite data transformation system, with a much smaller amount of data and artificially generated distortions, was implemented in the project as a *processor farm*. A processor farm consists of a master processor who divides the task between a number of slave processors. The slaves all execute the same code: they receive a subproblem from the master, solve this subproblem and send the results back to the master, upon which they receive a new subproblem. The master acts only as an administrator who distributes the problem to the slaves.

The master processor holds the original distorted picture Q which it has received from the satellite. It divides the matrix into a number of submatrixes by placing a sparse grid onto the picture. For each predefined grid point (x_g, y_g) in Q, it computes the corresponding position in the transformed picture P. The transformation function is determined by the satellite orbit parameters. This computation can be started as soon as the orbit parameters for the overflight are available and does not have to take place in real-time, during the overflight.

In this way, the problem is partitioned into a number of independent subtasks, which can be solved in parallel. A subtask consists of the four corner points of the grid, their coordinates in the transformed picture and the pixels in the distorted picture that fall inside these grid points. The processors are connected to each other in a one-dimensional array. The master sends output packets, consisting of the data associated with one subproblem, and accepts result packets. The slaves execute three simultaneous processes, as illustrated in Figure 4.3:

1. a communication process that handles communication to the processor and forwarding of messages to other processors

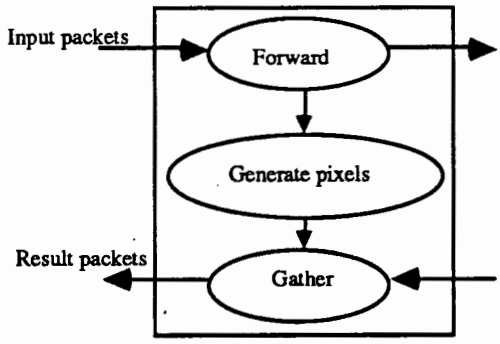

Figure 4.3: Process structure of the slaves

Processors	Without gather	With gather
1	7.3	8.7
2	3.9	5.0
4	2.3	3.3
8	1.5	2.4
16	1.3	2.2

Table 4.1: Processing times for 512*512 pixels (in seconds)

2. a pixel mapping process, that based on the earlier calculated transformed values for the corner points in a grid block interpolates the values of the rest of the pixels in the block

3. a gathering process, that collects pixel lines from the mapping process into complete rectangular output blocks, which are sent back to the master.

The gather process is needed when large transformed pictures are to be written to a disk. If the picture is large, the whole picture can not be stored in main memory, but it has to be written to a disk in reasonable large blocks. If the picture is small it can be stored in the main memory of the master processor and written to the disk when the whole picture has been assembled.

Results The system was tested with artificially generated distorted pictures. The image size used in the tests is 512*512 pixels and the number of processors varies between 1 and 16. The results of the tests are presented in Table 4.1. Time is measured in seconds and represent the time taken to transform one picture of 512*512 pixels.

Tests have been carried out both with and without the gathering phase. As can be seen, the gathering causes a significant overhead on the computation. However, for full-scale satellite pictures, this phase is necessary as the picture has to be written to a disk in smaller blocks.

In the best case (using 16 T800 transputers), transformed images was produced at a rate of about 116 Kb/s, which clearly satisfies the real-time requirements of about 88 Kb/s.

4.3 Three-dimensional cluster indentification in nuclear accelerator data

This application was done in co-operation with the Department of Physics at the University of Jyväskylä and the department of Physics at Åbo Akademi, by Ralph-Johan Back, Jens Granlund, Jorma Hattula, Tom Lönnroth and Patrick Waxlax.

Problem description The object of the study was to help in data analysis for nuclear physics experiment carried out on a nuclear accellerator at the University of Jyväskylä. A spectrometer detecting gamma radiation produces 10^8 to 10^9 observations at a rate of 2000 to 5000 observations per second. Each observation consists of a coordinate in a three-dimensional space of size 4096*4096*4096. A large amout of the observations is randomly distributed noise, while the rest of the observations, representing actual physical events, form clusters in the observation space. A cluster is defined as a concentration of observations which contain at least n observations within a radius of r units in the three-dimensional space, for some given values of n and r. Of the incoming observations, up to 90% can consist of noise and the remaining 10% correspond to data from real observed events. The problem is to identify the clusters, their position in the space and their intensities, i.e, the number of observations in the cluster.

Solutions The problem in this application is that the amount of data is very large. If all observations could be stored in a matrix of size 4K*4K*4K, the solution would be straightforward. However, this is not possible as it requires 64 Gbytes of memory to store the data in.

An alternative solution is to store the observations in a bit-map of 4K*4K*4K bits, which would reduce the memory requirement to 8 Gbytes. The bit-map would be very sparse, as no more than 1.5 percent of the entries would contain any observation. It must also be possible to register more than one observation in one position, which would require additional memory and also would complicate the algorithm. This solution was therefore also rejected because of the large memory requirement.

In the solution adopted, the observations are stored in a two-dimensional array consisting of linked lists of observations. Each element in the array consists of the (x, y) coordinates of the observations and a pointer to a list of z-values. The elements in the z-list consist of the z-coordinate, a counter indicating the number of observations in this point and a pointer to the next observation with the same (x, y) coordinates. The data structure is illustrated in Figure 4.4.

With this representation, one pointer for each (x, y) coordinate pair is needed, i.e., 4*4K*4K = 64 Mbytes. For each observation, one has to store the z value, the counter and a pointer to the next observation, giving a total of 7 bytes per observation. For 10^8 observations, the maximum memory requirement will be about 764 Mbytes. However, measurements showed that the average number of observations for the points that occur in the observations is between 4 and 5, so the actual memory requirement can be reduce by a factor of 4, giving a total memory requirement of about 190 Mbytes.

Implementation Because of the limited amount of memory in Hathi-2, the implementation was scaled down to a size of 400*400*4000 points. The problem was decomposed using geometrical parallelism, where each processor is responsible for a part of the space. The data domain is divided in equally large blocks in the (x, y) plane. Each processor keeps record of the observations falling inside its own domain, and whenever a new observation arrives, the processor inserts the observation into the appropriate z-list and checks, by searching the z-lists of the neighbouring points, if a cluster was formed. When a cluster is found, the observations

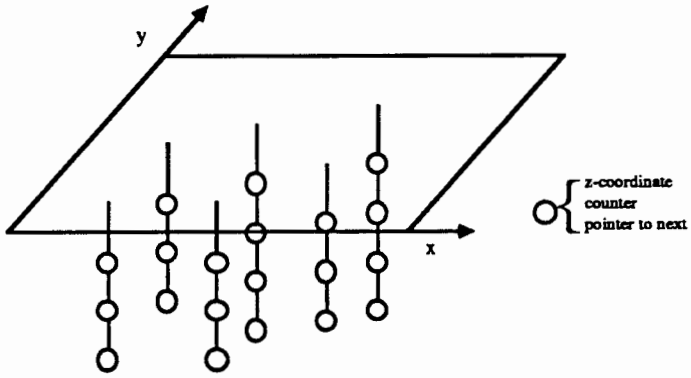

Figure 4.4: Data structure for storing observations

belonging to this cluster are removed from the z-lists and only the position and intensity of the cluster is stored in a separate list. New observations are also checked against this cluster-list to see if the observation falls into some already detected cluster.

The processors are connected to each other in a ring. The host processor sends the observations in form of (x, y, z) coordinates to the slaves. When a slave receives an observation, it checks wether the observation belongs to its own domain, in which case it processes the observation, or if it should send the observation to the next processor in the ring. In the experiments, up to 16 processors was used.

Results The system was first tested with artificially generated input data. A separate processor was used to produce input data with the same distribution as data generated in the experiment. However, the random number generator used to produce the input data was unable to calculate more than about 3000 observations per second, so these test could not be carried out for input rates highet than this. The program was able to process these 3000 observations per second.

As the next step, the system was tested with real data from the experiment. In this case, the bottleneck proved to be the interface to the data files on the user host processor (in this case a Sun 3/160 workstaton). Observations could be read from the disk at a maximum rate of about 1570 observations per second. The reason for this poor input/output performance was the primitive file store interface in the TDS programming environment.

Some experiments were carried out to measure the performance of the system without any input or output. These tests were carried out by processing the same input data a large number of times, so that data had to be input only once. The results from this test indicated that the system could handle over 4000 observations per second.

4.4 A multiprocessor system for full-text retrieval

The parallel full-text retrieval application was written by Marina Walldén (M.Sc. thesis) and Kaisa Sere [Walldén and Sere 89].

Problem description The problem studied in this application was fast retrieval of information in large text databases. Consider a database containing a large amount of documents from different articles (e.g. from newspapers), law text, bibliographies etc. The user wishes to search

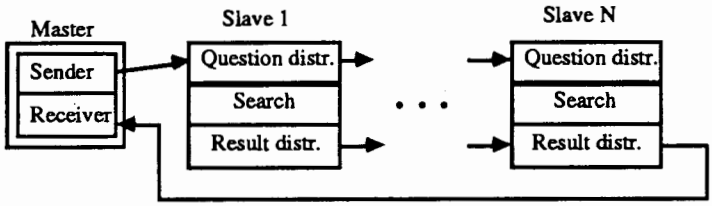

Figure 4.5: A ring processor farm

the database by making queries in the form of search words which describe the topic the user is interested in. The system should report to the user the documents that contain the given search words.

Full-text databases can be very large and consequently a search for a given pattern can be very time-consuming. To solve this problem, a parallel implementation of a full-text retrieval system was constructed.

Database representation The database is distributed among a number of processors, so that each processor holds only a part of the database. Each document is stored as a whole in one processor. Documents are stored using a *surrogate coding* technique, which makes it possible to quickly determine whether a word occurs in a document or not.

A *surrogate table* is an array of k bits. For each search word that we wish to insert into the table, we calculate i hash codes using some suitable hash functions, each with a value between 0 and $k-1$. To store a word in the surrogate table, the bit-positions in the table given by the hash-codes are set to 1. The number of hash codes per word, i, is often between 10 and 30 and k, the length of the surrogate tables is often 512 or 1024 bits. When we search for a word, the i hash-codes are calculated from each search word and the resulting bit-pattern is compared with the stored surrogate tables. If the corresponding bit positions in the surrogate table contain 1-bits, the word has been found in the database.

Parallel implementation The full-text retrieval system was implemented as a processor farm, with one master processor broadcasting queries to a number of identical slaves which search their own partition of the databas for the specified search words.

The master processor consists of two independent processes: a *sender* process which broadcasts search words to the slaves and a *receiver* process that accepts results from the slaves. A slave processor consists of three processes: a question distributer process which takes care of incoming search words, a search process which searches the processors part of the database for the given search words and a result distributor that sends the results from the search back to the master. The question distributor process contains buffers for search words so that the search process does not have to wait for communication, but can continue with the next search as soon as the previos has been completed.

The processor farm has been implemented on two different processor interconnection structures. The processor farm mapped to a unidirectional ring of processors is illustrated in Figure 4.5. The ring contains $N+1$ processors of which one acts as a master and the other N act as slaves. Tests were carried out with 3, 7, 15, 31, and 63 processors. The size of the database was 0.5, 1, 3 and 10 Mbytes. The speed-up factors from the tests are summarized in Table 4.2. A dash (-) in the table means that the corresponding test case could not be implemented due to shortage of memory on the slave transputers.

Processors	0.5 Mbyte	1 Mbyte	3 Mbyte	10 Mbyte
3	3.0	-	-	-
7	6.3	6.8	-	-
15	11.1	12.7	14.0	14.2
31	9.7	11.4	13.0	13.7
63	5.3	5.4	6.0	6.1

Table 4.2: Observed speed-up for ring structure

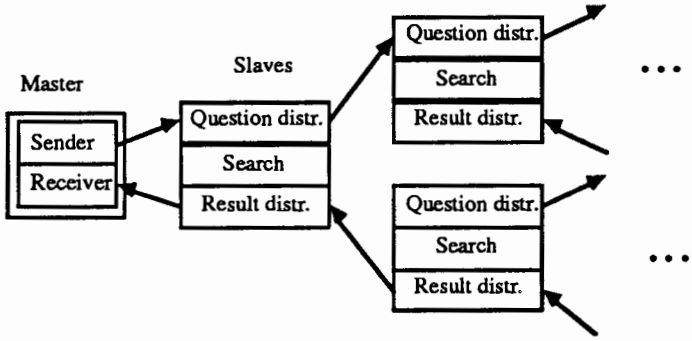

Figure 4.6: A tree processor farm

The processor farm mapped to a binary tree is illustrated in Figure 4.6. The same tests were carried out on the tree structure as for the ring structure. The measured speed-up factors from the tests are presented in Table 4.3.

Conclusions From Table 4.2 and Table 4.3 we can see that configurations with up to 15 processors give a linear speed-up, which is very close to the number of processors used. For tests with 31 or more processors, the speed-up decreases. The reason for this is that the portion of the database that is allocated to one processor becomes too small and thus the ratio between calculation and communication becomes too small. However, the limited amount of memory available prevented tests with larger databases to be carried out. We can also see that slightly better results can be observed for the binary tree. The reason for this is that the average length of the communication path is much shorter in a tree than in a ring.

Processors	0.5 Mbyte	1 Mbyte	3 Mbyte	10 Mbyte
3	3.0	-	-	-
7	6.6	6.8	-	-
15	11.9	13.5	14.9	14.9
31	9.6	11.3	13.0	13.7
63	5.4	6.2	6.9	7.3

Table 4.3: Observed speed-up for tree structure

4.5 Parallel implementation of the Rete-algorithm used in OPS5

This work was done as a M.Sc. thesis by Edward Eriksson. A similar parallel implementation of a production system architecture, not based on the Rete-algorithm but following the same lines as presented here, was done as a M.Sc. thesis by Johnny Boman.

Problem description OPS5 is a language for programming rule-based production systems. A production system consists of a number of if-then rules, which form the production memory, and a number of facts, which form the working memory. A rule consists of an if-part, which can be composed of one or more conditions elements, and one or more actions. The rules whose if-parts are satisfied by the facts in the working memory are said to be ready to fire and form the conflict set.

A production system is executed by determining the conflict set by matching the if-part of each production against the facts in the working memory. One of the productions in the conflict set is choosen and executed (fired), which results in changes in the working memory. The execution of the production system terminates when the conflict set becomes empty.

The *Rete-algorithm* is used in OPS5 for matching the productions against the contents of the working memory. In the Rete-algorithm the conditions of the productions are coded into a network consisting of the following types of nodes:

- Constant-test nodes, which test constants appearing in conditions against facts in the working memory
- Two-input nodes, which test whether two conditions in the left-hand side of a production are satisfied or not
- One-input nodes, which test variables inside a condition
- Terminal nodes, which report the result to the conflict set

Tokens are sent between the nodes in the network. A token consists of a type flag and one or more elements from the working memory. A + flag indicates that the element should be inserted into the working memory and a − flag that the element should be removed from the working memory.

The productions are coded as a network through which the facts, represented by tokens, are sent. When a terminal node, representing a production, receives a token with a + flag, this production is added to the conflict set.

One of the productions in the conflict set is then choosen, according to some algorithm, and executed. The execution of a production updates the facts in the working memory, and the cycle repeats until no more productions can be executed.

Parallel implementation The implementation was based on geometrical parallelism, where the production memory is distributed among the processors and each processor executes a local copy of the Rete-algorithm.

The production memory is partitioned into N groups, where N is the number of matching processors. Each matching processor executes a separate Rete-algorithm, matching the productions in its local production memory against the facts in the working memory. When a matching node has computed its local conflict set, it chooses one of the productions in the conflict set and reports this to a head-node. The head-node receives one production from each matching processor and selects one of these productions to be fired.

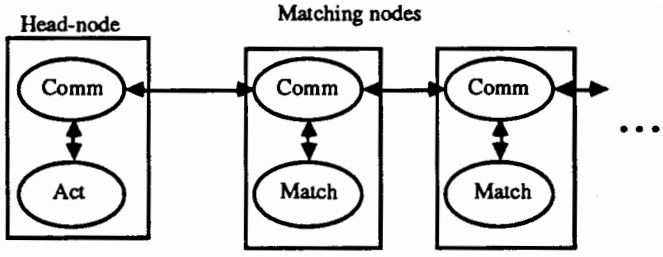

Figure 4.7: The process structure used in the implementation

Processors	Pipeline	Tree	Ring
2	33.8	-	33.7
3	38.4	38.2	38.3
5	48.4	54.5	48.0
10	60.7	60.4	59.5

Table 4.4: Number of firings per second in Sieve

The head node is responsible for maintaining the working memory and broadcasting the facts in the working memory to the matching processors. Thus, the head-node does not need any information about how many many matching processors there are, how they are connected to each other or how the productions are distributed among the matching processors.

The head-node executes two parallel processes: a communication handling process and a process that selects and exececutes productions. The matching processors also consist of two processes: a communication handler and a matching process executing a Rete-algorithm on the local working memory. The process structure is illustrated in Figure 4.7.

Results The program was tested with up to 15 processors connected into a pipeline, ring or binary tree. The test programs used were small and simple programs. The first test program is the Sieve of Erathostenes in which the prime numbers between 1 and 100 are calculated. The Sieve program consists of 10 production rules. The results of the tests, measured as the number of firings per second, is presented in Table 4.4.

The second program, called Count, is a synthethic test program consisting of 15 productions. The program increments a counter from 0 up to 500. The Count program is designed so that all productions will be matched for each iteration. The results from the Count program is presented in Table 4.5.

Processors	Pipeline	Tree	Ring
2	39.1	-	38.5
3	52.8	52.4	52.6
5	76.0	75.2	75.5
15	131.2	128.5	128.4

Table 4.5: Number of firings per second in Count

In the Sieve program, the measured speed-up was quite low, in the best case only about 3.8 (using 10 processors). In the Count program, the speed-up is considerably higher, in the best case about 9.9 (using 15 processors). We can see from the test cases that the physical interconnection structure has a very small influence on the performance of the program.

For both test programs, the efficiency was quite good for a small number of processors, but for a larger number of processors the efficiency decreases rapidly. The reason for this is that the test programs were small, and thus the amount of work allocated to one processor becomes small when we increase the number of processors. The production systems had to be translated by hand into corresponding Rete-networks, which made it impossible to carry out tests for large programs.

4.6 Other applications

A number of other applications of parallel programming have also been carried out in the Hathi project.

Parallel global optimization In this application, the applicability of parallel processing to global optimization, i.e., finding the global minimum value of a function f in a given region A, was studied by Aimo Törn [Törn 88c], [Törn and Žillinskas 89], [Walld'en 87]. Several commonly used global optimization methods can be implemented very efficiently on multiprocessor systems as the need for communication between the processors is very small.

Monte Carlo methods can be parallelized by executing the same sequential Monte Carlo algorithm on each processor, but with different random numbers. Each processor searches for the minimum function value independently of the other processors, by generating random points in the region A and evaluating the function in these points. The processors do not need to communicate with each other at all during the execution. Compared to a sequential Monte Carlo algorithm, the speed-up is equal to the number of processors.

Geometric parallelism can also be used, by dividing the optimization region A into N equally sized subregions. Each of the N processors are allocated an own subregion of A and searches only in this local region. If presumed minimum values are investigated using local search techniques, some interaction between neighbouring processors might be necessary in order to investigate values that are on the border between two processors.

Two parallel global optimization programs have been implemented, one using geometric parallelism with the processors arranged as a tree and the other as a processor farm, with the processors connected into a pipeline. Both programs showed a linear speed-up and very high efficiencies, as expected.

Simulation of analog circuits The problem studied in this application was simulation of analog circuits using the TOPSIM program. This work was done by Tarmo Leinonen at the University of Turku in cooperation with Nokia Research Center.

The TOPSIM program is used for simulating the behaviour of analog devices which are built of a number of analog circuits. Simulation models of the devices are built by combining software models of the analoog cicuits in a way reflecting the design of the device. The simulation model can then be executed and the behaviour of the analog device can be investigated before it is actually constructed.

The implemented program is a parallel execution of a simple TOPSIM simulation model. Starting from this sequential program, an equivalent parallel program was constructed and executed on 7 processors. The original sequential program was written in Fortran, so it was

decided to implement the parallel version using the same language (3L Parallel Fortran) in order to be able to reuse as much as possible of the code in the sequential program.

The main result from this application was to experimentally verify that large amounts of code from sequential programs can be directly incorporated in parallel programs without any modifications. However, the decomposition of a sequential program into a parallel program requires a thorough understanding of the problem and the program implementing the solution of the problem. The dependencies between the components in the program must be identified in order to decide what can be executed in parallel and how the paralle processes are to be arranged so that the amount of communication is minimized.

Chapter 5

Publications in the Hathi-project

5.1 Articles and technical reports

1986

Technical Reports

[Solin 86] Solin, U., *Parallel Algorithm Animation.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 50, 1986.

Abstract. A method for animating parallel program systems is introduced and a software environment for testing, analyzing and evaluating parallel program systems based on animation is outlined. Three animated parallel programs written in Occam are presented.

1987

Journals

[Leppälä 87] Leppälä, K., Utilization of Parallelism in transputer-based Real-Time Control Systems. *Microprocessing and Microprogramming* 21 1987, pp. 629-636. North-Holland.

Abstract. Transputer processors provide several attractive properties for real-time control system implementation: high processing power, modularity and low cost multiprocessor extension for fault tolerance, increased processing capability and physical processor distribution. However, there are some basic differences which must be examined. The present occam implementations support multitasking on two priority levels and time sharing on the low level. The communication mechanism between asynchronous occam processes must be matched with real-time event synchronized task processing. Multitasking system performance can be calculated, if real-time event occurence rates, service deadlines and service task processing times are given. Characteristic parameters - burst factor and embedded parallelism - are defined to evaluate multitasking extension to parallel multiprocessing. The model prototype of a board sorting line control system is analyzed under different real-time loading conditions. It is compared with other types of real-time control applications.

Proceedings

[Back 87a] Back, R.J.R., A Calculus for Refinements for Program Derivations. *Proc. of the Workshop on Programming Logic*, Editors: Dybjer et al. Report 37, pp. 271-299. Chalmers University of Technology and University of Göteborg. Göteborg 1987.

Abstract. For abstract see [Back 88a]

[Sere 87b] Sere K., Stepwise removal of virtual channels in distributed algorithms. *Proc. Distributed Algorithms, 2nd International workshop*, Amsterdam, Netherlands, July, 1987. Lecture Notes in Computer Science 312, pp. 408-428, (Revised version of [Sere87a]).

Abstract. A stepwise refinement method for the design of correct distributed algorithms is studied. The method frees the program designer from all the details of the target architecture of the system in early stages of the design process. The method is applied to a new aspect in the construction of distributed systems, the removal of virtual channels. We exemplify the design method by deriving a distributed algorithm. We show that the performed refinements preserve the correctness of the algorithm.

Technical Reports

[Back 87b] Back, R.J.R., *Procedural Abstraction in the Refinement Calculus.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 55, 1987. Submitted for publication.

Abstract. The refinement calculus provides a way in which to study correctness preserving program refinements, and gives criteria by which the correctness of replacements in program statements can be judged. We use the refinement calculus to describe the procedure mechanism, and to show how to handle procedure calls and procedure implementations in proofs of correctness. We will give proof rules for recursive procedure definitions. We also consider different parameter mechanisms in the proof of program correctness, and show how the correctness of large programs that are modularized by the procedure mechanism can be treated in the refinement calculus.

[Sere87a] Sere, K., *Stepwise removal of virtual channels in distributed algorithms: A case study.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 53, 1987.

Abstract. We study a stepwise refinement method for deriving correct distributed algorithms, and apply the method to a new aspect, the removal of virtual channels. The atomicity of the system is discussed in this context. We exemplify the method by deriving a distributed algorithm from a very high level description of a problem all the way down to a distributed program, which is in principle ready for execution. The correctness of each step is verified. Our notation to describe the algorithm in the different refinement steps has been influenced by production systems.

[Walldén 87] Walldén, M., *A Case Study: Performance of a Distributed Algorithm.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. B, No. 5, 1987.

1988

Journals

[Back 88a] Back, R.J.R., A Calculus of Refinements for Program Derivations. *Acta Informatica*, Vol 25, No 6, 1988, pp. 593-624, (Revised version of [Back87a]).

Abstract. A calculus of program refinements is described, to be used as a tool for the step-by-step derivation of correct programs. A derivation step is considered correct if the new program preserves the total correctness of the old program. This requirement is expressed as a relation of (correct) refinement between nondeterministic program statements. The properties of this relation are studied in detail.The usual sequential statement constructors are shown to be monotone with respect to this relation and it is shown how refinement between statements can be reduced to a proof of total correctness of the refining statement.

A special emphasis is put on the correctness of replacement steps, where some component of a program is replaced by another component. A method by which assertions can be added to statements to justify replacements in specific contexts is developed. The paper extends the weakest precondition technique of Dijkstra to proving correctness of larger program derivation steps, thus providing a unified framework for the axiomatic, the stepwise refinement and the transformational approach to program construction and verification.

[**Back and Kurki-Suonio 88a**] Back, R.J.R., Kurki-Suonio, R., Distributed Co-operation with Action Systems. *ACM Transactions on Programming Languages and Systems*, Vol 10, No 4, 1988, pp. 513-554.

Abstract. Action systems provide a method to program distributed systems that emphasizes the overall behavior of the system. System behaviour is described in terms of the possible interactions (*actions*) that the processes can engage in, rather than in terms of the sequential code that the processes execute. The actions provide a symmetric communication mechanism that permits an arbitrary number of processes to be synchronized by a common handshake. This is a generalization of the usual approach, employed in languages like CSP and Ada, in which communication is asymmetric and restricted to involve only two processes. Two different execution models are given for action systems, a sequential one and a concurrent one. The sequential model is easier to use for reasoning, and is essentially equivalent to the guarded iteration statement by Dijkstra. It is well suited for reasoning about system properties in temporal logic, but requires a stronger fairness notion than it is reasonable to assume a distributed implementation to support. The concurrent execution model reflects the true concurrency that is present in a distributed execution, and corresponds to the way in which the system is actually implemented. An efficient distributed implementation of action systems on a local area network is described. The fairness assumptions of the concurrent model can be guaranteed in this implementation. The relationship between the two execution models is studied in detail in the paper. For systems that will be called fairly serializable, the two models are shown to be equivalent. Proof methods are given for verifying this property of action systems. It is shown that for fairly serializable systems, properties that hold for any concurrent execution of the system can be established by temporal proofs that are conducted entirely within the simpler sequential execution model.

[**Eklund 88a**] Eklund, P., Mapping Processes onto Processors. *Wissenschaftlische Zeitschrift*, TH, Leipzig, 12(1988), pp. 105-109. Also published in 4. Tagung, Modellierung, Optimierung und Steuerung von Systemen. Wissenschaftlische Berichte der Technischen Hochschule Leipzig. June 1987, pp. 36-37.

[**Äijänen 88a**] Äijänen, T., Distributed Interconnection of a Reconfigurable Multicmputer System. *Microprocessing and Microprogramming* 23 1988, pp. 243-246. North-Holland.

Abstract. This paper describes a distributed interconnection system for a MIMD-type multicomputer system. The system is based on a regular network of switching elements. the computing elements are connected to the switches. The control of the switchers can be local or global. A parallel configurable computer based on transputers is being designed using this principle. The architecture of the computer is described. the main features of the machine are: reconfigurability, modularity, scalability and multi-usability.

Proceedings

[**Back and Kurki-Suonio 88b**] Back, R.J.R., Kurki-Suonio, R., Serializability in Distributed Systems with Handshaking. *15th International Colloquium on Automata, Languages and Programming*, Tampere, July 1988. Lecture Notes in Computer Science 317, pp. 52-66. Springer-Verlag. Heidelberg, 1988.

Abstract. Computations in distributed systems can be described in terms of *actions* in which one or more processes synchronize by common handshakes. A general formulation for such action systems is given, together with two interleaved execution models: a *serial* model that allows simple temporal reasoning, and a *concurrent* model that reflects a distributed execution environment more faithfully. The equivalence of the two models is shown, up to fairness properties. The relationships between the natural fairness and justice notions in the two models are analyzed. This leads to sufficient conditions under which reasoning in terms of the serial model is valid even when the execution environment guarantees the weaker properties of the concurrent model only. Proving that these conditions hold for a particular system can be carried out totally whithin the simpler serial model. Finally, the results are discussed from the point of view of partial order computations.

[**Eklund and Malén 88**] Eklund, P., Malén,T-E., Block Placement in Switching Networks. *Proc. CONPAR88*, Manchester, September 12-16, 1988, pp. 289-295. Cambridge University Press. 1988.

[**Solin 88**] Solin, U., Animation Techniques for Parallel Algorithms.

Proc. International Conference on Parallel Processing and Applications, September, 1987, L'Aquila, Italy, pp. 437-445. Editors: E. Chiricorozzi, A. D'Amico. Elsevier Science Publishers B.V. (North-Holland), 1988.

Abstract. An animating method suitable for testing, analyzing and evaluating of parallel programs, is presented. The method incorporates means for simulating parallel program execution on an arbitrary number of processors using one processor. The method is based on a technique of uniform delaying of parallel process execution allowing the animation to be produced during execution of the program.

[**Törn88c**] Törn, A.A., Parallel Monte Carlo with application to global optimization. In: K. Juslin and E. Silvennoinen (eds.), *Numerical Simulation of Processes, SIMS-88 Symposium*, The 30th Annual Meeting of the Scandinavian Simulation Society (SIMS), Espoo, Finland 21-22 April, 1988, VTT Symposium 84, (Technical Research Centre of Finland, Espoo) 156-165.

Abstract. Different types of parallel machines and existing systems are described and principles for parallelization of algorithms are discussed. Algorithms proposed for global optimization are reviewed and based on this some conclusions regarding parallelization of global optimization algorithms are presented.

Technical Reports

[**Back and Sere 88a**] Back,R.J.R., Sere, K., *An Exercise in Deriving Parallel Algorithms: Gaussian Elimination.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 65, 1988.

Abstract. The combination of the refinement calculus and the action system formalism has been proposed as a method for constructing parallel and distributed algorithms by stepwise refinement. We apply the approach to the derivation of a parallel version of the Gaussian elimination method

for solving simultaneous linear equation systems. We start from a completely sequential and traditional algorithm based on the familiar Gaussian elimination and in a sequence of refinement steps derive a highly parallel action system for the algorithm. Each step is verified in the framework of the refinement calculus.

[Back 88b] Back, R.J.R., *Data Refinement in the Refinement Calculus.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 68, 1988. Submitted for publication.

Abstract. For abstract see [Back 89a].

[Öhman et al 88] Öhman, G., Malén, T-E., Kuusela, P., *Numerical Fluid Flow and Heat Transfer Calculations on Multiprocessor Systems.* Report 88-3. Dept. of Chemical Engineering, Heat Engineering Laboratory, Åbo Akademi. 1988.

Abstract. The first part of the report presents the basic principles of parallel processing, and factors influecing the efficiency of practical applications are discussed. In a multiprocessor computer, different parts of the program code are executed in parallel, i.e. simultanmeous with respect to time, on different processors, and thus it becomes possible to decrease the overall computation time by a factor, which in the ideal case is equal to the number of processors. The application study starts from the numerical solution of the two-dimensional Laplace equation, which describes the steady heat conduction in a solid plate, and advances through the solution of the three-dimensional Laplace equation to the case of steady laminar fluid flow in a two-dimensional box at Reynolds numbers up to 20. Hereby the stream function-vorticity method is first applied and then the SIMPLER method. The conventional (sequential) numerical algorithms for these fluid flow ans heat transfer problems are found not to be ideally suited for concersion to parallel computionation, but speed-up ratios considerably above 50 achieved in the runs. The numerical procedures were coded in the OCCAM-2 language and the test runs were performed at Åbo Akademi on the experimental HATHI-computers containing 16 T414 and 100 INMOS T800 transputers respectively.

[Sere 88a] Sere, K., *Transforming Communication Topology in Distributed Algorithms.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 67, 1988. Submitted for publication.

Abstract. A stepwise refinement method for the construction of provably correct parallel and distributed algorithms is studied. One special type of refinement, the removal of virtual channels, is the main topic of this report. In our system a distributed algorithm can be constructed for a completely connected process net without respecting the process boundaries. The correctness of the algorithm is verified. The virtual connections are in some later steps replaced by existing connections. We develop a general method for these transformations based on refinement calculus and superimposition techniques. Relying on these techniques we show that the method generates a correct implementation of a virtual channel.

[Sere 88b] Sere, K., *Deriving Action Systems for Processor Farms.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 69, 1988. Submitted for publication.

Abstract. Parallel algorithms for processor farms are formally derived starting from a sequential algorithm. The derivation procedure is carried out in a series of correct refinement steps. The main concern is the formal derivation of a communication network for arbitrary processor farms. We show how an algorithm designed for a fully connected processor network is transformed into an algorithm which possesses an equivalent logical behaviour on a partially connected processor network. It is shown how the resulting algorithm is mapped on several different processor networks.

[Walldén and Sere 88] Walldén, M., Sere, K., *Design and Implementation of Full-Text Retrieval on Transputer Networks.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 63, 1988.

Abstract. We describe an implementation of a document retrieval system developed on Hathi–2. Hathi–2 is a multi–processor system which is built of IMS T800 transputers. An efficient distributed search strategy in large text databases is derived following the processor farm paradigm for programming parallel and distributed systems. The processor farm approach is considered for several different transputer network configurations. The main purpose of this paper is to study the applicability of this approach on transputer networks and to report on the observed performance figures for the document retrieval system.

1989

Journals

[Aspnäs et al 89b] Aspnäs, M., Back, R.J.R., Malén, T-E., The Hathi-2 Multiprocessor System. To appear in *Microprocessors and Microsystems.* Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 80, 1989.

Abstract. Hathi-2 is a reconfigurable general purpose loosely coupled MIMD multiprocessor system consisting of 100 32-bit IMS T800 transputers and 25 16-bit IMS T212 transputers. Hathi-2 has a distributed switching network and a distributed control system wich contains an interrupt system and hardware support for distributed monitoring. This report describes the architecture of the Hathi-2 system and the design goals of the system. The system software that has been developed for the system is also presented.

[Back and Kurki-Suonio 89] Back, R.J.R., Kurki-Suonio, R., Decentralization of Process Nets with Centralized Control. *Distributed Computing*, Vol 3, No 2, May 1989, pp. 73-87. Also published in Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 58.

Abstract. The behavior of a net of interconnected, communicating processes is described in terms of the joint actions in which the processes can participate. A distinction is made between centralized and decentralized action systems. In the former, a central agent with complete information about the state of the system controls the execution of the actions; in the latter no such agent is needed. Properties of joint action systems are expressed in temporal logic. Centralized action systems allow for simple description of system behavior. Decentralized (two-process) action systems again can be mechanically compiled into a collection of CSP processes. A method for transforming centralized action systems into decentralized ones is described. The correctness of this method is proved, and its use is illustrated by deriving a process net that distributedly sorts successive lists of integers.

[Back and Sere 89b] Back,R.J.R., Sere, K., Stepwise Refinement of Parallel Algorithms. To appear in *Science of Computer Programming.* Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 64, 1988.

Abstract. The refinement calculus and the action system formalism are combined to provide a uniform method for constructing parallel and distributed algorithms by stepwise refinement. It is shown that the sequential refinement calculus can be used as such for most of the derivation steps. Parallelism is introduced during the derivation by refinement of atomicity. The approach is applied to the derivation of a parallel version of the Gaussian elimination method for solving simultaneous linear equation systems.

[Back and vWright 89b] Back, R.J.R., v. Wright, J., Duality in Specification Languages: A Lattice Theoretic Approach. To appear in *Acta Informatica*. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 77, 1989.

Abstract. A very general lattice-based language of commands, based on the primitive operations of substitution and test for equality, is constructed. This base language permits unbounded nondeterminism, demonic and angelic nondeterminism. A dual language permitting miracles is constructed. Combining these two languages yields an extended base language which is complete, in the sense that all monotonic predicate transformers can be constructed in it. The extended base language provides a unifying framework for various specification languages; we show how two Dijkstra-style specification languages can be embedded in it.

[Walldén and Sere 89] Walldén, M., Sere, K., Free-text Retrieval on Transputer Networks. *Microprocessors and Microsystems*, Vol 13 No 3, April 1989, pp. 179-187. (Revised version of [Walldén and Sere 88]).

Abstract. An implementation of a document retrieval system developed on Hathi–2 is described. Hathi–2 is a multi–processor system built of IMS T800 transputers. An efficient distributed search strategy in large free–text databases is derived following the processor farm paradigm for programming parallel and distributed systems. The processor farm approach is considered for several different transputer network configurations. The main purpose of this paper is to study the applicability of this approach on transputer networks and to report on the observed performance figures for the document retrieval system.

Proceedings

[Aspnäs and Back 89] Aspnäs, M., Back, R.J.R., A Programming Environment for a Transputer-Based Multiprocessor System. To appear in *Proc. First Finnish-Hungarian Workshop on Programming Languages and Software Tools*, Szeged, Hungary, 7-11.8.1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 82, 1989.

Abstract. This paper presents a transputer-based multiprocessor system, Hathi-2, and the programming environment being developed for this system. Hathi-2 is mainly programmed in the language Occam, and thus the programming environment is based on the Occam model of parallelism and communication. The programming environment gives the user an abstraction of the physical structure of the multiprocessor system. The user sees the multiprocessor system as a pool of resources (processors and communication links), which are allocated to the the users program and connected to the topology described by the program structure. The environment is implemented on a Sun 3 graphical workstation.

[Back 89a] Back, R.J.R., Changing Data Representation in the Refinement Calculus. *Proceedings on Hawaii International Conference on System Sciences (HICSS-22)*, 3-6.1.1989, Kailua-Kona, Hawaii, (Revised version of [Back 88b]).

Abstract. The refinement calculus gives a formal basis for the stepwise refinement method of program construction. The caculus is an extension of the weakest precondition calculus of Dijkstra. In this paper we show how to systematically change the data representation in programs (*data refinement*)within this calculus. The original method for data refinement in the refinement calculus [Ba80] was only defined for functional data abstractions. We show here that this method can be extended to non-functional data abstraction, by extending the weakest precondition calculus with conjugate statements. These model the total correctness of statements with don't know (or angelic) nondeterminism rather than the usual don't care (or demonic) nondeterminism assumed by Dijkstra. The data refinement method described here is very flexible, permitting

different data abstractions to be combined in the same statement, as well as refinements of the data abstractions themeselves. An example of using this flexibility in program derivation is given in the paper.

[Back89b] Back, R.J.R., Refining Atomicity in Parallel Algorithms. To appear in *Proc. Conference on Parallel Architechtures and Languages Europe*, Eindhoven, the Netherlands, June12-16, 1989. Lecture Notes in Computer Science. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 57.

Abstract. The paper extends the refinement calculus of sequential programs to handle stepwise refinement of parallel and distributed programs. The refinement steps preserve the total correctness of the programs. The action system approach is used to model the behaviour of parallel programs. The central topic is refinement of atomicity in such programs. It is shown how the refinement calculus can be used as a foundation for the derivation of parallel and distributed program by systematically decreasing the grain of interleaving in these and thereby increasing the amount of potential parallelism in the programs.

[Back and Sere 89a] Back, R.J.R., Sere, K., Stepwise Refinement of Action Systems. *Proc. of the Conference on Mathematics of Program Construction*, Groningen, the Netherlands, June 26-30,1989. Lecture Notes in Computer Science No 375, pp. 115-138. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 78.

Abstract. A stepwise refinement method for the formal development of provably correct parallel algorithms is presented. Following our systematic method the entire derivation procedure is carried out in the context of purely sequential programs. The resulting parallel algorithm can be effectively executed in different architectures. The methodology is illustrated by showing the leading derivation steps in a construction of a parallel algorithm for square matrix multiplication.

[Back and vWright 89a] Back, R.J.R., v. Wright, J., A Lattice-theoretic basis for a specification language. *Proc. of the Conference on Mathematics of Program Construction*, Groningen, the Netherlands, June 26-30,1989. Lecture Notes in Computer Science No 375, pp. 139-156. (Short version of [Back and von Wright 89b].)

[Kilpinen et al 89] Kilpinen, A., Björkholm, T., Westerlund, T., Parallel Calculation of a Packed Bed Reactor. *Proc. Modelling, Identification and Control*, 8th IAESTED International Conference, February 7-12, 1989, Grindewald, Switzerland. Also published in Report 89-102-A, Åbo Akademi, Dept of Chemical Engineering, Process Design Laboratory, 1989.

Abstract. Simulation of chemical reactors with complex models for the transport phenomena is usually time consuming. The computations therefor often limit the use of such models, for example in parameters estimation and in design of experiments. We modified the computations for the simulation of a packed bed reactor so that they can be performed in parallel to a part. We found that the simulation of the reactor is well suited for parallel processing on a transputer based machine. The amount of data to be transferred between the processors is small and the calculations are rather time consuming. In present application a so-called processor farm approach has been used. It is shown that the computation time in our application was reduced from days to hours.

[Rantala et al 89] Rantala, A., Raunio, A., Still, D-J., A Multiprocessor System for Fast Geometric Image Transformation. *Proc. 6th Scandinavian Conference on Image Analysis*, Oulu, June 19-22,1989, Vol. II, pp. 1182-1189. The Pattern Recognition Society of Finland 1989. Also published in Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 66, 1988.

Abstract. A prototype for a transputer based image processing system is described. The system performs a user-definable geometric (space domain) transformation that ca be used to correct distortions in image data received from a satellite and to produce images using a predefined projection and orientation. In order to shorten the processing time the problem has been partitioned for a processor farm. The results from the prototype suggests that a transputer based solution is efficient for this kind of real time applications.

[Shen 89a] Shen, H., A Fast Parallel Algorithm for Integer Sorting. To appear in *Advances in Parallel Comouting*, proc. of Parallel Computing 89, Leiden, The Netherlands, August 29 - September 1, 1989. North-Holland. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 70, 1988.

Abstract. This paper presents a deterministic and optimal parallel algorithm to solve the integer sorting problem. The first version of the algorithm is to sort n distinct integers and the second version is to sort n random integers. The algorithm runs in $O(\log n)$ time on $n/\log n$ processors in the EREW PRAM model.

[Shen 89b] Shen, H., Mapping Parallel Programs onto Transputer Networks. In *Proc. of Australian Transputer and Occam User Group Conference*, Melbourne, Australia, July 5-6, 1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 79, 1989.

Abstract. A fundmental problem of great significance in parallel processing is to map parallel programs onto parallel computers. In this paper we begin with studying the mathematical model for the mapping problem, then discuss different strategies to solve this problem, and finally present a fast heuristic algorithm to solve this problem on transputer networks. Our mapping algorithm functionally consists of partitioning (grouping), placement and routing that work cooperatively in a divide-and-conquer way. It finally produces a satisfactory result for the mapping problem. The proposed algorithm can also be easily realized in a parallel way.

[Shen 89c] Shen, H., Fast Path-disjoint Routing in Transputer Networks. To appear in *Proc. of First Finnish-Hungarian Worshop on Programming Languages and Software Tools*, Szeged, Hungary, August 7-11, 1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 83, 1989.

Abstract. This paper addresses the problem of path-disjoint routing in transputer networks. In this paper, we first study the criteria for path-disjoint routing, then give heuristic approaches to the criteria, and finally present a fast heuristic algorithm to solve this problem on transputer networks. For routing k disjoint paths, our algorithm works on a $m \times n$ mesh, multigrid or torus structure in $O(km^2n^2)$ time. This algorithm has also been implemented in Occam on the Hathi-2 transputer network. The implementation result shows a layout with minimum path length and least path bends for all of the produced paths.

[Shen 89d] Shen, H., Self-adjusting mapping:a heuristic mapping algorithm for mapping parallel programs onto transputer networks. Ed. Wexler, J., *Developing Transputer Applications* (OUG 11). IOS Amsterdam 1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 87, 1989.

Abstract. The problem of mapping parallel programs onto multiprocessor system is a fundmental problem of great significance in parallel processing, but it is NP-hard in general. In this paper we propose a fast heuristic algorithm to solve this problem on transputer networks. Our mapping algorithm consists of three modules: grouping, placement and routing, where grouping groups processes in the program into tasks which can be placed onto processors in the transputer network in a way of one-to-one, placement places the grouped tasks onto the processors and

routing produces physical communication paths for logical communication requirements. The three modules work co-operatively in a way of progressive self-adjusting, and finally produce a satisfactory solution for the mapping problem.

Technical Reports

[**Aspnäs et al 89a**] Aspnäs, M., Back, R.J.R., Kurki-Suonio, R., *Efficient Implementation of Multi-process Handshaking on Broadcasting Networks.* Åbo Akademi. Reports on Computer Science & Mathematics, Ser. A. No 75, 1989.

Abstract. Multi-process handshaking is a generalization of ordinary handshake communication between two processes. It allows an arbitrary number of processes to be synchronized in a common handshake, for the purpose of carrying out a joint action, consisting of a sequential statement that is executed in the combined state space of the processes participating in the handshake. The statement updates the local variables of the processes in a manner that depends on the values of the local variables of the other processes. The paper is concerned with implementing this communication mechanism on networks with a broadcasting facility. The implementation is described and its correctness is proved. The efficiency is calculated analytically and verified experimentally by simulation studies.

[**Aspnäs and Malén 89**] Aspnäs, M., Malén, T-E., *Hathi-2 Users Guide, version 1.0.* Åbo Akademi. Reports on Computer Sceince & Mathematics, Ser. B, No 6, 1989.

Abstract. Hathi-2 is a reconfigurable general purpose loosely coupled MIMD multiprocessor system consisting of 100 32-bit IMS T800 transputers and 25 16-bit IMS T212 transputers. Hathi-2 has a distributed switching network built of IMS C004 crossbar switches and a distributed control system. The system can be used by several independent users simultaneously. Each user is assigned a separate subsystem of Hathi-2. This report describes the Hathi-2 system and how it can be used.

[**Back and vWright 89c**] Back, R.J.R., v. Wright, J., *Refinement Concepts Formalized in Higher Order Logic.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 85, 1989. Submitted for publication.

Abstract. A theory of commands with weakest precondition semantics is formalized using the HOL proof assistant system. The concept of refinement between commands is formalized, a number of refinement rules are proved and it is shown how the formalization can be used for proving refinements of actual program texts correct.

[**Back and vWright 89d**] Back, R.J.R., v. Wright, J., *Combining Angels, Demons and Miracles in Program Specifications.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 86, 1989.

Abstract. The command lattice C, containing all monotonic predicate transformers, is divided into sublanguages using criteria of demonic and angelic nondeterminism, termination and absence of miracles. We investigate dualities between the sublanguages and how they can be generated from simple primitive commands. The notions of total correctness and refinement are generalized to the command lattice C.

[**Pehkonen 89a**] Pehkonen, K., *The Implementation of Linnainmaa and Harwood's Pose Determination Algorithm on Shared and Distributed Memory Parallel Computers.* University of Maryland, Center of Automation Research. report No CAR-TR-423, CS-TR-2191, DACA76-88-C-0008. 1989.

Abstract. This report describes the implementation of Linnainmaa and Harwood's pose determination algorithm on the Butterfly Parallel Processor (BPP) and Hathi-2 parallel computers. The architecture of the BPP is based on a shared memory model, whereas the hathi-2 is based on a distributed memory model. The algorithm is computationally very intensive, which makes it suitable for parallel processing. The program was parallelized using the processor farm technique, thus enabling automatic load balancing. the experiments show that the algorithm is very easy to parallelize. furthermore comparison of the two architectures shows that the Hathi-2 is much more powerful than the BPP. Due to different implementation technologies, it is not, however, possible to say whether one of the architectures is in general better than the other.

[**Ståhl and Back 89**] Ståhl, L., Back, R.J.R., An Implementation of Multiprocess Handshaking on Transputer networks. Åbo Akademi. Reports on Computer Science & Mathematics, Ser. A, No 76, 1989.

Abstract. Multiprocess handshaking is a generalization of ordinary handshake communication between two processes. It allows an arbitrary , though pre-defined number of asynchronous processes to be synchronized in a common handshake, for the purpose of carrying out a communication event. This is a sequential statement that is executed in the combined state space of the processes participating in the handshake. The statement updates the local variables of the processes, in a manner that depends on the values of the local variables of the other processes. The paper is concerned with implementing this communication mechanism on transputer networks. Bagrodias Event-manager algorithm is used for synchronizating the processes.

[**vWright 89a**] v. Wright, J., *Stepwise Derivation of a Parallel Matrix Multiplication Algorithm.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 84, 1989.

Abstract. A standard multiplication algorithm for square matrices is transformed into a distributed algorithm. Every transformation step is verified within the refinement calculus. Some high-level rules of refinement that are used in the derivation are proved.

Working papers

[**Aspnäs and Back 89**] Aspnäs, M., Back, R.J.R., Multiprocessor Applications in the Hathi–project. To appear in Scientific Computing in Finland. Manuscript 1989.

[**Back 89c**] Back, R.J.R., Refinement of reactive action systems. To appear in *Proc. REX Workshop for Refinement of Distibuted Systems*, Nijmegen July 1989.

[**Back and Sere 89b**] Back, R.J.R., Sere, K., *Implementing action systems in Occam.* Manuscript 1989.

[**Sere 89**] Sere, K., *Laws of Action Systems Programming.* Manuscript 1989.

5.2 Publications for academic degrees

Ph.Lic. theses

[**Hartikainen 88**] Hartikainen, E., Specification and design of distributed programs that use broadcasting. University of Helsinki, Department of Computer Science. Ph.Lic. thesis 1988.

[**Sere 88c**] Sere, K., Stepwise Refinement of Parallel Algorithms. Åbo Akademi, Dept. of Computer Science. Ph.Lic. thesis 1988.

[**Pehkonen 89b**] Pehkonen, K., A Dynamically Reconfigurable Parallel Computer Hathi-2. University of Oulu, Dept. of Electrical Engineering. Ph.Lic. thesis 1989.

[**Solin 89**] Animation of Parallel Algorithms. Åbo Akademi, Dept. of Computer Science. Ph.Lic. thesis 1989. Under evaluation.

[**vWright 89b**] A Lattice-theoretic Basis for Stepwise Refinement of Programs. Åbo Akademi, Dept. of Computer Science. Ph.Lic. thesis 1989.

M.Sc. theses

[**Aspnäs 87**] Aspnäs, M., An Implementation of Joint Actions in Modula-2. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1987. In Swedish.

[**Raunio 87**] Raunio, A., Modularity of Action Systems. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1987. In Swedish.

[**Boman 88**] Boman, J., Computer Architechtures for Production System Execution. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1988. In Swedish.

[**Eriksson 88**] Eriksson, E., A Parallel Rete Algorithm. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1988. In Swedish.

[**Kuusela 88**] Kuusela, P., A Numerical Fluid-flow Calculations with the SIMPLER Method on an Multiprocessor System. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1988. In Swedish.

[**Malén 88**] Malén, T-E., CoSSo, a Tool for Configuring a Multiprocessor System. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1988. In Swedish.

[**Ståhl 89**] Ståhl, L., An Implementation of Multiprocess Handshaking on Transputer Networks. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1989. In Swedish.

[**Waldén 88**] Waldén, M., Full Text Searching on Multiprocessor Systems. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1988. In Swedish.

[**Levander 89**] Levander, S., MoSSu, A Tool for Monitoring a Multiprocessor System. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1989. In Swedish.

[**Lindström 89**] Lindström, M-L., APA - A Tool for Animating Parallel Algorithms – An Implementation of Animation Processes. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1989. In Swedish.

[**Nyman 89**] Nyman, Y., APA - A Tool for Animating Parallel Algorithms. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1989. In Swedish.

Special Assignment

[**Boman and Eriksson 86a**] Boman, J., Eriksson, E., Distributed Sortering. Åbo Akademi, Dept. of Computer Science. 1986. In Swedish.

[**Boman and Eriksson 86b**] Boman, J., Eriksson, E., Instrumentation of Occam-programs. Åbo Akademi, Dept. of Computer Science. 1986. In Swedish.

[**Waldén 87b**] Waldén, M., Distributed Global Optimization. Åbo Akademi, Dept. of Computer Science. 1987. In Swedish.

[**Granlund and Waxlax 88**] Granlund, J., Waxlax, P., Investigation of Cluster Centra with a Multiprocessor System. Åbo Akademi, Dept. of Computer Science. 1988. In Swedish.

[**Leinonen 88**] Leinonen, T., On the Parallelization of TOPSIM Systems. University of Turku, Dept. of Computer Science.1988. In Finnish.

[**Nyman 88**] Nyman, Y., A Prototype System for Interactive Design of Pictures on SUN Workstations. Åbo Akademi, Dept. of Computer Science. 1988. In Swedish.

[**Santara 88**] Santara, J., Modelling Parallel Systems on a Main Frame Computer. University of Turku, Dept. of Computer Science. 1988. In Finnish.

[**Ståhl 88**] Ståhl, L., A Prototype for Pluto. Åbo Akademi, Dept. of Computer Science. 1988. In Swedish.

[**Hekanaho 89**] Hekanaho, J., TransEnvironment. Åbo Akademi, Dept. of Computer Science. 1989. In Swedish.

[**Långbacka 89**] Långbacka, T., Monitoring of Hathi-2. Åbo Akademi, Dept. of Computer Science. 1989. In Swedish.

5.3 Manuals, patents and journals (in finnish)

Manuals

[**Leppälä and Rautiola 87**] Leppälä, K., Rautiola, K., Rinnakkaistietokone HATHI-2. VTT/TKO, Oulu. 1987.

[**Pehkonen 87**] Pehkonen, K.P., Hajautettu Tietokone HATHI-2. VTT/TKO, Oulu. 1987.

[**Äijänen 87c**] Äijänen, T., Rinnakkaiset tietokoneet. VTT/TKO, Oulu. 1987

[**Pehkonen 88a**] Pehkonen, K.P., HATHI-2 teknillinen dokumentti. VTT/TKO, Oulu. 1988

Patents

[**Leppälä 88**] Leppälä, K., Kytkentäjärjestelmä. Finnish patent application 880754, 17.2.1988.

[**Äijänen 88b**] Äijänen, T., Hajautettu Kytkentäjärjestelmä. Finnish patent application 880753, 17.2.1988..

[**Leppälä 89**] Leppälä, K., Switching System. International patent application PCT/F189/00027. February 15, 1989.

[**Äijänen 89**] Äijänen, T., Distributed Switching System. International patent application PCT/F189/00026. February 15, 1989.

Finnish Journals

[**Back 87c**] Back, R.J.R., Rinnakkaisprosessoinnin ongelmat. Tietotekniikka 1/87, ss. 33-35, 56.

[**Äijänen 87a**] Äijänen, T., Rinnakkaisuudella tehoa laskentaan. Korkeakoulujen atk-uutiset, nro. 2/1987, ss. 43-45. Valtion Tietokonekeskus, korkeakoulujen palveluosasto.

[**Äijänen 87b**] Äijänen, T., Hyperkuutioistako kilpailija Craylle. Tietotekniikka 1/87, ss. 51-54.

[**Aspnäs and Raunio 88**] Aspnäs, M., Raunio, A., Moniprosessorijärjestelmän sovelluksia HATHI. Tietotekniikka 5/88, ss. 24-25.

[**Insinööriuutiset 88**] Insinööriuutiset, 23.9.1988, s. 24. 1000 mipsiä puolen miljoonan markan koneella. (Interview with prof. Ralph-Johan Back).

[**Leppälä and Rautiola 88**] Leppälä, K., Rautiola, K., Kotimainen Superkone. Prosessori, 9/1988, ss. 25-28.

[**Pehkonen 88b**] Pehkonen, K., Rinnakkaisprosessointia – luonnollisesti. Tietotekniikka 5/88, ss.16-18.

[**Forum 89**] Forum 6/1989, ss. 53, 54. Nästa datalyft: Parallel Processering.

[**Helsingin Sanomat 89**] Helsingin Sanomat, 4.3.1989, s. 2. Suomikin on siirtynyt supertietokoneaikaan.

[**Kauppalehti 89**] Kauppalehti , No 16, 23.1.1989, s. 28. Tuhannen mipsin HATHI.

References

[**Aspnäs 87**] Aspnäs, M., An Implementation of Joint Actions in Modula-2. Åbo Akademi, Dept. of Computer Science. M.Sc. thesis 1987. In Swedish.

[**Aspnäs and Back 89**] Aspnäs, M., Back, R.J.R., A Programming Environment for a Transputer-Based Multiprocessor System. To appear in *Proc. First Finnish-Hungarian Workshop on Programming Languages and Software Tools*, Szeged, Hungary, 7-11.8.1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 82, 1989.

[**Aspnäs et al 89b**] Aspnäs, M., Back, R.J.R., Malén, T-E., The Hathi-2 Multiprocessor System. To appear in *Microprocessors and Microsystems.* Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 80, 1989.

[**Aspnäs et al 89a**] Aspnäs, M., Back, R.J.R., Kurki-Suonio, R., *Efficient Implementation of Multi-process Handshaking on Broadcasting Networks.* Åbo Akademi. Reports on Computer Science & Mathematics, Ser. A. No 75, 1989.

[**Back 78**] Back, R.J.R., *On the Correctness of Refinement in Program Development.* Ph.D. thesis. Report A–1978–4, Department of Computer Science, University of Helsinki, 1978.

[**Back 80**] Back, R.J.R., *Correctness Preserving Program Refinements: Proof Theory and Applications.* Mathematical Center Tracts 131, Mathematical Centre, Amsterdam 1980.

[**Back 87a**] Back, R.J.R., A Calculus for Refinements for Program Derivations. *Proc. of the Workshop on Programming Logic*, Editors: Dybjer et al. Report 37, pp. 271-299. Chalmers University of Technology and University of Göteborg. Göteborg 1987.

[**Back 87b**] Back, R.J.R., *Procedural Abstraction in the Refinement Calculus.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 55, 1987. Submitted for publication.

[**Back 88a**] Back, R.J.R., A Calculus of Refinements for Program Derivations. *Acta Informatica*, Vol 25, No 6, 1988, pp. 593-624, (Revised version of [Back87a]).

[**Back 88b**] Back, R.J.R., *Data Refinement in the Refinement Calculus.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 68, 1988. Submitted for publication.

[**Back 89a**] Back, R.J.R., Changing Data Representation in the Refinement Calculus. *Proceedings on Hawaii International Conference on System Sciences (HICSS-22)*, 3-6.1.1989, Kailua-Kona, Hawaii, (Revised version of [Back 88b]).

[**Back 89b**] Back, R.J.R., Refining Atomicity in Parallel Algorithms. To appear in *Proc. Conference on Parallel Architechtures and Languages Europe*, Eindhoven, the Netherlands, June12-16, 1989. Lecture Notes in Computer Science. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 57.

[**Back 89c**] Back, R.J.R., Refinement of reactive action systems. To appear in *Proc. REX Workshop for Refinement of Distibuted Systems*, Nijmegen July 1989.

[**Back and Kurki-Suonio 83**] Back, R.J.R., Kurki-Suonio, R., Decentralization of process nets with centralized control. *2nd ACM SIGACT-SIGTOPS Symp. on Principles of Distributed Computing*, Montreal, Canada, August 1983, pp. 131-142.

[**Back and Kurki-Suonio 84a**] Back, R.J.R., Kurki-Suonio, R., A case study in constructing distributed algorithms: distributed exchage sort. *Proc. Winter School on Theoretical Computer science*, Lammi, Finland,August 1984. Finnish Society of Information Processing Science, pp. 1-33.

[**Back and Kurki-Suonio 84b**] Back, R.J.R., Kurki-Suonio, R., *Co-operation in distributed systems using symmetric multi-process handshaking.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 34.

[**Back and Kurki-Suonio 85**] Back, R.J.R., Kurki-Suonio, R., *Serializability in distributed systems.* Carnegie-Mellon University, Dept. of Computer Science. Research Reports CMU–CS–109, 1985.

[**Back and Kurki-Suonio 88a**] Back, R.J.R., Kurki-Suonio, R., Distributed Co-operation with Action Systems. *ACM Transactions on Programming Languages and Systems*, Vol 10, No 4, 1988, pp. 513-554.

[**Back and Kurki-Suonio 89**] Back, R.J.R., Kurki-Suonio, R., Decentralization of Process Nets with Centralized Control. *Distributed Computing*, Vol 3, No 2, May 1989, pp. 73-87. Also published in Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 58.

[**Back and Sere 88a**] Back,R.J.R., Sere, K., *An Exercise in Deriving Parallel Algorithms: Gaussian Elimination.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 65, 1988.

[**Back and Sere 89a**] Back, R.J.R., Sere, K., Stepwise Refinement of Action Systems. *Proc. of the Conference on Mathematics of Program Construction*, Groningen, the Netherlands, June 26-30,1989. Lecture Notes in Computer Science No 375, pp. 115-138. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 78.

[**Back and vWright 89a**] Back, R.J.R., v. Wright, J., A Lattice-theoretic basis for a specification language. *Proc. of the Conference on Mathematics of Program Construction*, Groningen, the Netherlands, June 26-30,1989. Lecture Notes in Computer Science No 375, pp. 139-156. (Short version of [Back and von Wright 89b].)

[**Back and vWright 89b**] Back, R.J.R., v. Wright, J., Duality in Specification Languages: A Lattice Theoretic Approach. To appear in *Acta Informatica*. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 77, 1989.

[**Back and vWright 89c**] Back, R.J.R., v. Wright, J., *Refinement Concepts Formalized in Higher Order Logic.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 85, 1989. Submitted for publication.

[**Back and vWright 89d**] Back, R.J.R., v. Wright, J., *Combining Angels, Demons and Miracles in Program Specifications.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 86, 1989.

[**Chandy and Misra 88**] Chandy, K.M., Misra, J., *A Foundation of Parallel Program Design.* Addison-Wesley, 1988, forthcoming.

[**Dijkstra 76**] Dijkstra, E.W., *A Discipline of Programming.* Prentice-Hall International, 1976.

[**Eklund 88a**] Eklund, P., Mapping Processes onto Processors. *Wissenschaftlische Zeitschrift*, TH, Leipzig, 12(1988), pp. 105-109. Also published in 4. Tagung, Modellierung, Optimierung und Steuerung von Systemen. Wissenschaftlische Berichte der Technischen Hochschule Leipzig. June 1987, pp. 36-37.

[**Eklund and Malén 88**] Eklund, P., Malén,T-E., Block Placement in Switching Networks. *Proc. CONPAR88*, Manchester, September 12-16, 1988, pp. 289-295. Cambridge University Press. 1988.

[**Hartikainen 88**] Hartikainen, E., *Specification and design of distributed programs that use broadcasting.* University of Helsinki, Department of Computer Science. Ph.Lic. thesis, 1988.

[**Kilpinen et al 89**] Kilpinen, A., Björkholm, T., Westerlund, T., Parallel Calculation of a Packed Bed Reactor. *Proc. of the IAESTED International Symposium, Modelling, Identification and Control, MIC'89*, Grindewald, Switzerland, February 7-12, 1989. Editor M.H. Hamza, Acta Press. Also published in Report 89-102-A, Åbo Akademi, Dept of Chemical Engineering, Process Design Laboratory, 1989.

[**Leppälä 87**] Leppälä, K., Utilization of Parallelism in transputer-based Real-Time Control Systems. *Microprocessing and Microprogramming* 21 1987, pp. 629-636. North-Holland.

[**Morgan et al 88**] Morgan, C., Robinson, K.A., Gardiner, P.H.B., *On the refinement calculus.* Draft, April 1988.

[**Morris 87**] Morris, J.M., A Theoretical Basis for Stepwise Refinement and the Programming Calculus. *Science or Computer Programming 9*, 1987, pp. 287-306. North–Holland.

[**Pehkonen 88a**] Pehkonen, K.P., HATHI-2 teknillinen dokumentti. VTT/TKO, Oulu. 1988

[**Pehkonen 89a**] Pehkonen, K., *The Implementation of Linnainmaa and harwood's Pose Determination Algorithm on Shared and Distributed Memory Parallel Computers.* University of Maryland, Center of Automation Research. report No CAR-TR-423, CS-TR-2191, DACA76-88-C-0008. 1989.

[**Pehkonen 89b**] Pehkonen, K., A Dynamically Reconfigurable Parallel Computer Hathi-2. University of Oulu, Dept. of Electrical Engineering. Ph.Lic. thesis 1989.

[**Rantala et al 89**] Rantala, A., Raunio, A., Still, D-J., A Multiprocessor System for Fast Geometric Image Transformation. *Proc. 6th Scandinavian Conference on Image Analysis*, Oulu, June 19-22,1989, Vol. II, pp. 1182-1189. The Pattern Recognition Society of Finland 1989. Also published in Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 66, 1988.

[**Sere 87b**] Sere K., Stepwise removal of virtual channels in distributed algorithms. *Proc. Distributed Algorithms, 2nd International workshop*, Amsterdam, Netherlands, July, 1987. Lecture Notes in Computer Science 312, pp. 408-428, (Revised version of [Sere87a]).

[**Sere 88a**] Sere, K., *Transforming Communication Topology in Distributed Algorithms.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 67, 1988. Submitted for publication.

[**Sere 88b**] Sere, K., *Deriving Action Systems for Processor Farms.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No 69, 1988. Submitted for publication.

[**Sere 88c**] Sere, K., Stepwise Refinement of Parallel Algorithms. Åbo Akademi, Dept. of Computer Science. Ph.Lic. thesis 1988.

[**Sere 88c**] Sere, K., *Laws of Action Systems Programming.* Manuscript 1989.

[**Shen 89b**] Shen, H., Mapping Parallel Programs onto Transputer Networks. In *Proc. of Australian Transputer and Occam User Group Conference*, Melbourne, Australia, July 5-6, 1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 79, 1989.

[**Shen 89c**] Shen, H., Fast Path-disjoint Routing in Transputer Networks. To appear in *Proc. of First Finnish-Hungarian Worshop on Programming Languages and Software Tools*, Szeged, Hungary, August 7-11, 1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 83, 1989.

[**Shen 89d**] Shen, H., Self-adjusting mapping:a heuristic mapping algorithm for mapping parallel programs onto transputer networks. Ed. Wexler, J., *Developing Transputer Applications* (OUG 11). IOS Amsterdam 1989. Also published in reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 87, 1989.

[**Solin 86**] Solin, U., *Parallel Algorithm Animation.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. A, No. 50, 1986.

[**Solin 88**] Solin, U., Animation Techniques for Parallel Algorithms. *Proc. International Conference on Parallel Processing and Applications*, September, 1987, L'Aquila, Italy, pp. 437-445. Editors: E. Chiricorozzi, A. D'Amico. Elsevier Science Publishers B.V. (North-Holland), 1988.

[**Solin 89**] Animation of Parallel Algorithms. Åbo Akademi, Dept. of Computer Science. Ph.Lic. thesis 1989. Under evaluation.

[**Ståhl and Back 89**] Ståhl, L., Back, R.J.R., An Implementation of Multiprocess Handshaking on Transputer networks. Åbo Akademi. Reports on Computer Science & Mathematics, Ser. A, No 76, 1989.

[**Törn 88c**] Törn, A.A., Parallel Monte Carlo with application to global optimization. In: K. Juslin and E. Silvennoinen (eds.), *Numerical Simulation of Processes, SIMS-88 Symposium*, The 30th Annual Meeting of the Scandinavian Simulation Society (SIMS), Espoo, Finland 21-22 April, 1988, VTT Symposium 84, (Technical Research Centre of Finland, Espoo) 156-165.

[**vWright 89b**] A Lattice-theoretic Basis for Stepwise Refinement of Programs. Åbo Akademi, Dept. of Computer Science. Ph.Lic. thesis 1989.

[**Leppälä and Rautiola 87**] Leppälä, K., Rautiola, K., Rinnakkaistietokone HATHI-2. VTT/TKO, Oulu. 1987.

[**Walldén 87**] Walldén, M., *A Case Study: Performance of a Distributed Algorithm.* Reports on Computer Science & Mathematics, Åbo Akademi, Ser. B, No. 5, 1987.

[**Walldén and Sere 89**] Walldén, M., Sere, K., Free-text Retrieval on Transputer Networks. *Microprocessors and Microsystems*, Vol 13 No 3, April 1989, pp. 179-187. (Revised version of [Walldén and Sere 88]).

[**Äijänen 87c**] Äijänen, T., Rinnakkaiset tietokoneet. VTT/TKO, Oulu. 1987

[**Äijänen 88a**] Äijänen, T., Distributed Interconnection of a Reconfigurable Multicmputer System. *Microprocessing and Microprogramming* 23 1988, pp. 243-246. North-Holland.

[**Öhman et al 88**] Öhman, G., Malén, T-E., Kuusela, P., *Numerical Fluid Flow and Heat Transfer Calculations on Multiprocessor Systems.* Report 88-3. Dept. of Chemical Engineering, Heat Engineering Laboratory, Åbo Akademi. 1988.

Financial Support for the Personnel

Name	Period			Months	Source	Note
Aspnäs John	01.08.1986	–	31.07.1988	6	ÅA	25 % of time
	01.08.1988	–	31.12.1988	5	TEKES	
Aspnäs Mats	01.08.1986	–	31.12.1986	1,25	ÅA	25 % of time
	01.01.1987	–	31.03.1988	15	TEKES	
Back Ralph-Johan	01.08.1986	–	31.07.1988	6	OPM	25 % of time
Boman Jonny	15.09.1986	–	31.01.1987	4,5	SA	
Eklund Patrik	01.08.1986	–	31.05.1987	2	ÅA	25 % of time
Eriksson Edward	15.09.1986	–	31.01.1987	4,5	SA	
Frantz Kristian	01.06.1987	–	31.12.1987	7	SA	
Granlund Jens	01.05.1987	–	31.07.1987	3	ÅA	trainee
	01.08.1987	–	31.03.1988	2	ÅA	25 % of time
Hekanaho Jukka	01.06.1988	–	31.07.1988	2	SA	
Högnäs Viking	01.01.1987	–	28.02.1987	2		trainee
	15.10.1988	–	31.12.1988	2,5	SA	
Kuusela Pekka	01.01.1988	–	31.05.1988	5	ÅA	scholarship
Leppälä Kari	01.08.1986	–	31.12.1988		VTT/TKO	per hour
Levander Stefan	01.06.1988	–	30.06.1988	1	SA	
Malen Tor-Erik	01.05.1987	–	31.12.1987			
	01.01.1988	–	31.08.1988			
	01.09.1988	–	31.12.1988	20	SA	
Nyman Yngve	01.04.1988	–	31.07.1988	4	TEKES	
	01.08.1988	–	31.12.1988	5	SA	
Pehkonen Kari	01.08.1986	–	31.12.1988		VTT/TKO	per hour
Raunio Antti	01.08.1986	–	31.10.1986	0,75	ÅA	25 % of time
	01.11.1987	–	31.12.1987			
	01.01.1988	–	31.03.1988	5	SA	
Rautiola Kyösti	01.08.1986	–	31.12.1988		VTT/TKO	per hour
Sere Kaisa	01.08.1986	–	31.12.1986			
	01.01.1987	–	31.07.1987	2,1	ÅA	30 % of time
	01.08.1987	–	31.12.1987	1,25	ÅA	25 % of time
	01.01.1988	–	31.05.1988	5	OPM	
	01.06.1988	–	31.08.1988	0,75	ÅA	25 % of time
	01.09.1988	–	31.12.1988	4	ÅA	
Shen Hong	01.04.1988	–	31.12.1988	9	SA	

Solin Ulla	01.08.1986	–	31.07.1988	29	OPM	
Still Dan-Johan	01.02.1988	–	31.05.1988			50 % of time
	01.07.1988	–	31.07.1988	2,5		50 % of time
	01.08.1988	–	31.12.1988	5	SA	
Ståhl Lena	01.11.1987	–	31.12.1987			
	01.11.1988	–	31.12.1988	4	SA	
Söderholm Eva	01.01.1987	–	08.06.1987	1,25	ÅA	25 % of time
	08.06.1987	–	31.07.1988	2	SA	
Walden Marina	01.09.1986	–	31.08.1987			50 % of time
	01.01.1988	–	31.08.1988	14	SA	
Waxlax Patrick	01.05.1987	–	31.07.1987	3	ÅA	trainee
	01.08.1987	–	31.07.1988	3	ÅA	25 % of time
	01.11.1988	–	31.12.1988	2	SA	
Äijänen Tapani	01.08.1986	–	31.12.1987		VTT/TKO	per hour