A low cost parallel computing system for photomask pattern data preprocessing

BJÖRN LUNDBERG

Master of Science Thesis
Stockholm, Sweden 2004

IMIT/LECS-2004-9

A low cost parallel computing system for photomask pattern data preprocessing

Master of Science Thesis

BJÖRN LUNDBERG

Stockholm, March 2004

Supervisors:

Anders Thurén, Micronic Laser Systems AB

Vladimir Vlassov, Department of Microelectronics and Information Technology, Royal Institute of Technology

Examiner:

Vladimir Vlassov, Department of Microelectronics and Information Technology, Royal Institute of Technology

MICRONIC LASER SYSTEMS

Micronic Laser Systems AB
P.O Box 3141
Nytorpsvägen 9
SE-183 03 Täby, SWEDEN

Dept of Microelectronics and Information Technology
Royal Institute of Technology
SE-100 44 Stockholm, SWEDEN

Abstract

This master thesis project is a part of the preparations for the production of Micronic Laser Systems AB's next generation laser pattern generator, Sigma 8100, and it is also included in the feasibility study for a *maskless* system that writes directly on silicon.

Clustering Linux/x86 computer nodes is today considered a low cost yet high performing solution for high performance computing. The goal of the project is to give suggestions on how to build a scalable low cost Linux/x86 cluster for photomask pattern data preprocessing. A survey of available common off the shelf products such as interconnects and disks are included as a part of this thesis. A workload simulator written in C that models the workload of the preprocessing software complements the thesis and allows for more accurate benchmarking. This workload simulator is designed to be easily portable and could be used to evaluate different types of hardware.

One of the recommended solutions presented in this thesis is based on standard rack mounted servers with Intel Xeon processors and Gigabit Ethernet as interconnect. Another recommendation is the MPI standard for communication between the computer nodes.

The only hardware tested and verified in this thesis is a disk solution presented by the company VMETRO. It proved to be reasonably low cost and it delivers more than the 360 Mbyte/s output and 90 Mbyte/s input that are required, however it has a few drawbacks related to a very low-level programming interface. It also contradicts the general goal of the project to use common off the shelf products.

The lack of hardware during the course of the project meant that this thesis has been shaped to be a collection of knowledge, techniques and a benchmarking tool for further evaluations.

Acknowledgements

This master thesis project has been an inspiring experience right from the start. All the way from working out the boundaries and keeping the project within reasonable limits and to the finalizing of the report, but also the challenge of leading the small but intense evaluation projects in cooperation with VMETRO and VSYSTEMS.

My supervisor Vladimir Vlassov at IMIT/KTH deserves my gratitude for his support when formulating the report. He has also been a great help concerning all the bureaucracy surrounding the master thesis project.

I would like to thank all the people at Micronic Laser Systems AB for their support during this project. My industrial supervisor Anders Thurén and Fredric Ihrén at Micronic Laser Systems AB deserve a special acknowledgement for their constant support and for giving me highly valued feedback on my ideas. I would also like to thank Anders Thurén and Rigmor Remahl for telling the history behind Micronic Laser Systems AB.

Björn Lundberg

Täby, Sweden

March 2004

Table of contents

Abstract ..... **i**

Acknowledgements ..... **ii**

Table of contents ..... **iii**

Table of Figures ..... **v**

Table of Tables ..... **vi**

1 Introduction **1**

1.1 Problem description 1
1.2 Thesis overview 2

2 Background **3**

2.1 Micronic Laser Systems AB 3
2.1.1 The history behind Micronic Laser Systems AB *3*
2.2 The laser pattern generator 4
2.3 Pattern data processing 5
2.3.1 The pattern data processing hardware *7*

3 System requirements **8**

3.1 Performance and functionality 8
3.2 Software 10
3.3 Summary 10

4 Survey of common off the shelf products **11**

4.1 OS and software 11
4.2 Motherboard and processor 12
4.2.1 Suggested products *12*
4.3 Interconnect 13
4.3.1 Gigabit Ethernet *13*
4.3.2 Myrinet *14*
4.3.3 SCI *14*
4.3.4 Quadrics *15*
4.3.5 InfiniBand *15*
4.3.6 Summary *16*
4.4 Disk 16
4.4.1 RAID in general *17*
4.4.2 Fibre Channel in general *18*
4.4.3 Dell\EMC CLARiiON CX-series *18*
4.4.4 Just a Bunch Of Disks (JBOD) *19*
4.4.5 Distributed local disk *20*
4.5 Conclusions 21

5 A preview of the JBOD **23**

5.1 Test setup 1 23
5.1.1 Hardware *23*
5.1.2 Software *23*
5.2 Tests 24
5.3 Analysis 25
5.4 Test setup 2 25
5.4.1 Hardware *26*
5.4.2 Software *26*
5.5 Tests 26
5.6 Analysis 28
5.7 Conclusions 28

6 The preprocessing program 29

6.1 Analyzing the preprocessing program 29
6.1.1 Memory usage 29
6.1.2 Time complexity 30
6.1.3 Planned improvements 30
6.2 Redesigning the preprocessing program 31
6.2.1 The input process 31
6.2.2 The output processes 32
6.3 Analyzing pattern data 32
6.3.1 Conclusions 34
6.4 Summary 34

7 The workload simulator 35

7.1 Modeling the real preprocessing program 35
7.1.1 Parameters 37
7.2 System and programming notes 39
7.3 Summary 39

8 Testing the workload simulator 40

8.1 Test setup 40
8.2 Tests 41
8.3 Analysis 41
8.4 Conclusions 42

9 Conclusions 43

10 Future work 45

11 References 46

12 Abbreviations 48

Table of Figures

Figure 1.1 A block diagram of the preprocessing system in principle 1
Figure 2.1 Acousto-optic device (from [1])..... 4
Figure 2.2 The writing principle using acousto-optic devises (from [1]) 4
Figure 2.3 The writing principal using a SLM chip (from [3]) 5
Figure 2.4 Data volume increase (from [3]) 6
Figure 2.5 A simplified view of pattern data Fracturing 6
Figure 3.1 A block diagram of the preprocessing system in principle 8
Figure 3.2 A block diagram of the preprocessing stage 9
Figure 3.3 A block diagram of the pattern data loading stage 9
Figure 3.4 A block diagram of the pattern data extraction during write-time 9
Figure 3.5 The preprocessing cluster/NUMA in detail 10
Figure 4.1 A specialized JBOD solution by VMETRO 19
Figure 4.2 A computer cluster using two separate subnets (left) and a computer cluster with a single net but with a load balancing Gigabit Ethernet adapter (right) 21
Figure 5.1 Test setup 1 of the JBOD preview 23
Figure 5.2 The read bandwidth (left) and the write bandwidth (right) as a function of read block size and read operations. 25
Figure 5.3 Test setup 2 of the JBOD preview 26
Figure 5.4 The read bandwidth (left) and the write bandwidth (right) as a function of read block size and read operations. 27
Figure 6.1 The current structure of the preprocessing program (CFRAC) 29
Figure 6.2 A possible structure of the preprocessing program for a computing cluster 31
Figure 6.3 Splitting a repetition block using round robin 32
Figure 7.1 The distribution using a combination of round robin and bag of tasks. 36
Figure 7.2 Requesting data in the writing process. 37
Figure 7.3 A parameter file for the workload simulator. 38

Table of Tables

Table 4.1 A selection of motherboards and processors 13
Table 4.2 A comparison of the available interconnects 16
Table 5.1 Test results from the preview of the specialized JBOD. 24
Table 5.2 Test results from the second test of the JBOD 27
Table 5.3 Test results when writing directly from RAM 28
Table 6.1 Profile of file: 1_input 33
Table 6.2 Profile of file: 2_input 33
Table 6.3 Profile of file: 1_output 33
Table 6.4 Profile of file: 2_output 34
Table 8.1 The three parameter files used in the workload simulator test 40
Table 8.2 The results from the first subtest, testing mainly message passing 41
Table 8.3 The results from the second subtest, testing mainly disk capacity 41
Table 8.4 The results from the final subtest, testing the complete workload simulator 41

1 Introduction

This master thesis project is my final step towards a Master of Science degree at the Royal Institute of Technology (KTH) in Stockholm Sweden. It was done at Micronic Laser Systems AB under supervision of the Department of Microelectronics and Information Technology at KTH.

With an appealing price-performance ratio it is not surprising that Intel based Linux clusters are continuously gaining ground. The increasing performance of standard PC components and Gigabit Ethernet networking makes the idea of common-off-the-shelf products into an attractive alternative. The increasing popularity of the operating system Linux gives it a number of positive characteristics such as a high level of compatibility with different hardware and it is becoming a stable well-tested platform. Perhaps one of the most underestimated benefits is the increasing number of skilled people with first hand experience of parallel computing with Linux. It is easier to find a programmer with good knowledge in MPI and Linux than a programmer specialized in low-level DMA communications for a specific embedded system.

New on the market are the 64-bit CPU architectures (IA-64) and interconnects like Gigabit Ethernet [30], Quadrics [25] and InfiniBand [21]. These new types of interconnect makes the clusters scalable beyond thousands of nodes. Thereby letting the clusters compete with high-end embedded industrial systems with RapidIO [31] or RACE++ [32] in terms of performance. However there will almost always be a tradeoff in terms of space and in some cases power consumption.

1.1 Problem description

This master thesis project is a part of the preparations for Micronic Laser Systems AB's next generation laser pattern generator, Sigma 8100, and it will also be included in the feasibility study for a *maskless* system that writes directly on silicon. The current solutions for pattern preprocessing with systems from SUN Microsystems is considered both expensive and not easily scalable. Porting this complete software structure to Linux/x86 is one of the proposed solutions and will be evaluated in this thesis project. The system must be easily scalable in order to adapt to future demands on performance. The preprocessing stage sets very high demands on disk performance; this is so vital that it is included in this thesis project. In order to limit the size of this master thesis project the throughput has been predefined and a suggested layout in principle has been suggested in form of a block diagram, as seen in Figure 1.1.

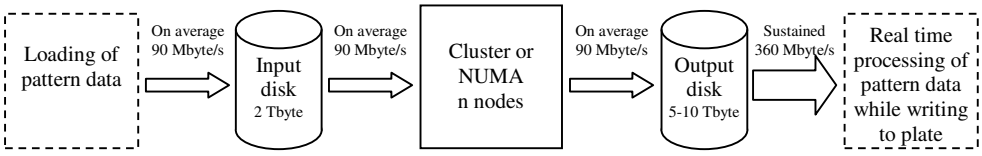

Figure 1.1 A block diagram of the preprocessing system in principle

In the long term perspective thoughts are of replacing the real time processing system with some form of clustered standard Linux/x86 solution. The thesis could be used as a base of knowledge for this type of project as well.

The main goal of this thesis is to evaluate if a Linux cluster is a good choice in terms of performance and scalability. To give suggestions on hardware configurations for the next generation preprocessing system, using standard off the shelf components and considering reliability as an important factor. Give general recommendations on how to port Micronics current software, not in detail but rather in terms of possible C-libraries for threading, communication, etc.

A secondary goal of the project is to suggest and evaluate different proposed solutions for the output disk. This is included in the project since it is the most probable bottleneck of the system. It is also considered to be the most critical part since it has to deliver in near real-time conditions.

To achieve this goal I start by studying the given requirements of the preprocessing system. The second stage is to survey the relevant available common off the shelf products. The most natural is then to test and analyze the most interesting of these products. Due to the complex nature of the preprocessing system I design a workload simulator that models the workload of the real program. This simulator works as a kind of specialized benchmarking tool. This allows for a more accurate way of comparing the performance of different systems. In order to design this workload simulator I analyze the current preprocessing program and relevant pattern data, this also requires some estimations of the future since the workload simulator has to evaluate according to future demands.

1.2 Thesis overview

Section 1 gives an introduction to the master thesis project and to the thesis itself. Due to the complexity of the system Section 2 starts with an introduction to Micronic Laser Systems AB and the laser pattern writers. Section 2.3 is absolutely fundamental for the understanding of this thesis, it describes the principals behind the preprocessing and processing of pattern data. The goal of the project and the requirements of the system are further described in Section 3.

The path to achieving the goals for this project really starts with a survey of existing x86 CPU boards, interconnects and disk solutions in Section 4. This Section is not meant to be a complete market survey but rather selective in terms of how well the products fit into the estimated system. After the survey and making some selections the next logical step is to test these systems. This however is always a matter of availability and time. A disk solution proposed by VMETRO [36] is briefly tested during two sessions in Section 5. The current preprocessing program is analyzed in Section 6. This section also tries to predict future changes in the preprocessing program. The knowledge obtained in the previous section is used to design a workload simulator in Section 7. This workload simulator can later be used as a benchmarking tool. A small test of the workload simulator is done in Section 8.

A summary of my results and conclusions are presented in Section 9. Since most hardware was not available during the course of the project the remaining work had to be postponed for the future. Other considerations for future work are also discussed in Section 10.

2 Background

The main goal of this section is to give some form of motivation to the system requirements and to why the system is designed the way it is. Section 2.3 is particularly important for this thesis.

2.1 Micronic Laser Systems AB

Micronic Laser Systems AB produces laser pattern generators used in the production of photomasks. These photomasks are in turn used in the production of displays and semiconductors. This technology is called microlithography. The typical buyer of a laser pattern generator is a manufacturer, also called a maskshop, who deliver photomasks to producers of electronic products. Some of the biggest producers of electronic products might buy their own laser pattern generator.

Micronic Laser Systems AB has regional offices in Japan, USA, Korea and Taiwan. At the end of 2002 the company had a total number of 338 employees of which 269 of them reside in Sweden [5].

There are three main markets for laser pattern generators: The smallest one is Multi Purpose, where laser pattern generators are mainly used for electronic packaging. Micronic Laser Systems AB does have a significant part of the high-end part of this market. The display market is the collective name for a market that is producing shadow masks for CRTs and TVs. Other parts of the display market uses photomasks for PDP, LCD, TFT and color filters for TFT. Micronic Laser Systems AB has had a near 100% share of this market for a number of years now. The largest market for pattern generators is the semiconductor market. This market used to be dominated by the electron-beam technology. But since this technology is very expensive, slow and the gap in writing quality is shrinking, these machines are often replaced by laser pattern generators today. However electron-beams will still be around for some time for the really delicate patterns. It is likely that just a few layers in a modern processor have been made by an electron-beam machine while all other layers has been made by a laser pattern generator.

2.1.1 The history behind Micronic Laser Systems AB

It all started some time in the 1970s [6], when Dr Gerhard Westerberg and his group began researching into microlithography at the Royal Institute of Technology (KTH) in Stockholm. The primary goal was the semiconductor industry and in 1977 the first machine was sold to a company in France called SGS Thomson. Photomasks from this particular machine was used to produce the first Motorola 68000 processor. The same company bought a second similar machine just two years later.

Micronic did not become commercialized until 1984 when Dr Gerhard Westerberg and seven employees founded the company called Micronic Laser Systems. Until then the company was just called Micronic and produced hand held terminals for logistics, in some sense this company or rather some parts of it still remains in the company now called Minec Systems [7]. Micronic Laser Systems was not able to sell any laser pattern generators until Svenska Grindmatriser AB (SGA) in Linköping bought one in 1989. Dr Gerhard Westerberg died in 1989. Micronic Lasers Systems AB was then restarted and founded by the employees and Småföretagsfonden. A friend of Dr Gerhard Westerberg, Lic Nils Björk, was assigned as the new CEO (1989-1996).

Terapixel Ltd in Finland bought their first 180 mm semiconductor laser pattern generator from Micronic in 1990. They used it for a wide variety of purposes and soon requested a larger writing area; hence the first 600 mm writer was born. This started a trend for large area laser pattern generators. From that point on Micronic Laser Systems has had a near 100% share of the display market, but to the expense of leaving the semiconductor market. Micronic Laser Systems made the first step back into the semiconductor market in 1998 with the LRS 200 system.

2.2 The laser pattern generator

The technology is based on two acousto-optic devices: an acousto-optic modulator (AOM) for controlling the intensity of the laser beam and an acousto-optic deflector (AOD) for generating a sweep [1]. This type of acousto-optic device is basically a crystal with an applied acoustic drive signal. The acoustic drive signal changes the density of the crystal making it behave like a grating diffracting the passing laser beam, see Figure 2.1. The angle of the diffracting light is dependent of the frequency of the light as well as the density of the grating. Hence the diffracting angle is possible to change by changing the frequency of the acoustic drive signal to the AOD. The intensity of the diffracting light is dependent of how much the acoustic drive signal is changing the density of the crystal. Hence by changing the amplitude of the drive signal to the AOM it is possible to change the intensity of the laser beam.

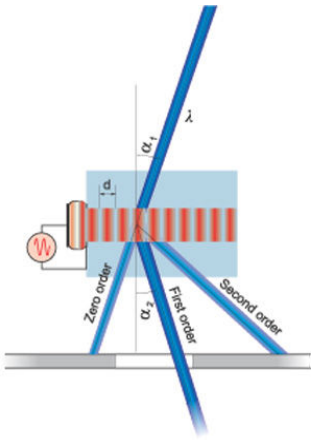

Figure 2.1 Acousto-optic device (from [1])

This technique is combined with a movable stage with a laser interferometer positioning system. The stage is moved with a constant speed in the X-direction while the AOD makes the beam sweep over a limited width in the Y-direction creating a scan-strip. Then the stage is moved a certain distance in Y and the next scan-strip can be started. The pattern data is converted into amplitude variations to the AOM. This is shown in Figure 2.2 picturing the principal layout of the 5-beam Omega semiconductor laser pattern generator.

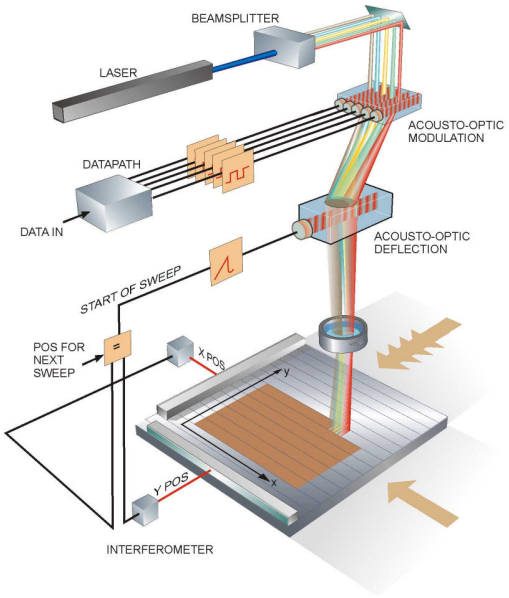

Figure 2.2 The writing principle using acousto-optic devises (from [1])

The latest technology does however not include the acousto-optic devices. The new technology uses a Spatial Light Modulator (SLM) [2] and is developed in cooperation with the Fraunhofer Institute IMS [4]. The SLM is a chip consisting of one million mirrors; each mirror is 16µm x 16µm in size and has the ability to individually tilt one quarter of a wavelength or 62 nm. A laser beam is flashed at the mirrors reflecting as a stamp on the plate creating the pattern. The idea with a moving stage is the same as earlier. 64 different scales of gray can be achieved by letting a mirror make a very small movement causing phase modulations. These phase modulations will diffract the beam as it passes through a Fourier lens and can thereby be partly or completely filtered away in the Fourier plane by an aperture leaving only the desired image on the plate, see Figure 2.3. One of the biggest benefits with this technology is the possibility to use a shorter wavelength and thereby being able to draw smaller features. Today the 16µm x 16µm mirrors are projected on the plate as 100nm x 100nm pixels using a 248 nm wavelength. Since each pixel has 64 gray scales, the result is a 1.56 nm address grid.

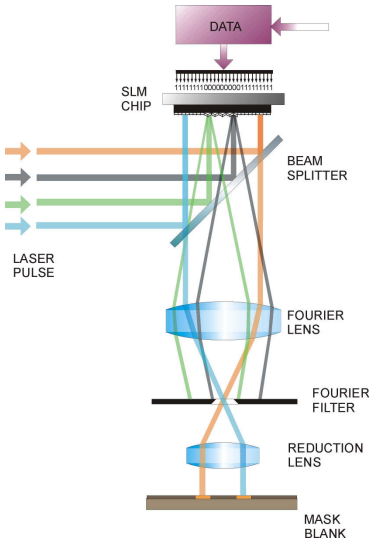

Figure 2.3 The writing principal using a SLM chip (from [3])

2.3 Pattern data processing

The pattern data going in to a laser pattern generator is generated by a CAD system and is described in a hierarchical vector format with repetitions and layers. Layers within a file are combined to produce the final pattern using Boolean operations such as *and*, *or*, *xor* etc. The final result is a rasterized (bitmap) format that in turn can be translated into control signals for an AOM or SLM. The first basic step is to fracture the pattern data into the individual scan-strips. Still in a vector format, but the independent scan-strips can now be rasterized in parallel using a number of computer nodes, each processing one scan-strip at a time. This method is used in the large area laser pattern generators for the display market, but it does not process data fast enough for the latest laser pattern generators aimed at the semiconductor market.

Since the semiconductor technologies seems to be evolving as predicted in Moore's law, the number of features on a photomask is increasing exponentially as a function of time. However each feature has to be described by more and more extra assisting features in order to compensate for optical and etching phenomena, see Figure 2.4. Hence the data volume is increasing faster than Moore's law.

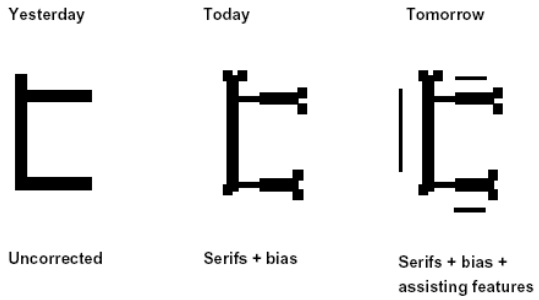

Figure 2.4 Data volume increase (from [3])

In order to cope with the amount of data requested by the SLM chip the area covered by the SLM is divided into several rendering windows and rasterized individually. Each rendering window has its own data channel with a dedicated FPGA rasterizer. In order to feed these rendering window modules, each scan-strip must be divided into individual sub-strips. Since the data volume increases every time the pattern is divided into individual parts, each sub-strip is not fractured in the x-direction to fit the SLM chip. Instead each rendering module has an extra amount of memory enabling it to hold more data. In this way a sub-strip only has to be fractured into a few numbers of dependent fracturing windows in the x-direction. Since the rendering module can keep data that will be used later each fracturing window does not need to be independent, thus reducing the duplication of data.

The trend of adding assisting features and increased fracturing of pattern data, to enable parallel processing, sets higher demands on preprocessing. The data volumes increase ever more in the preprocessing stages making it necessary to move some of the preprocessing work to a pipelined data channel to process the data in real time. It is no longer possible to fracture the pattern data into scan-strips or sub-strips at an off-line stage. Instead the off-line stage is limited to fracturing the data into larger dependent sub-areas, called buckets, as well as doing a workload distribution by simply splitting the pattern data into a number of non-geometrical independent groups called File Memory Buffers (FMB). Since these FMBs are independent they can be fractured into scan-strips, sub-strips and fracturing windows concurrently in real time during writing. Of course a certain fracturing window has to be created by merging the respective fracturing window from each FMB before sending it to the rendering module.

A simplified version of the pattern data fracturing can be seen in Figure 2.5. The original pattern data is shown in a). In step b) the data has been partitioned into two independent FMBs and has been sorted into two buckets in y (symbolized by the dotted line). This step is done in a preprocessing stage. The following stages c) and d) are done in real-time during writing. The data is now fractured into 5 independent sub-strips. Note that the figure has been simplified and there is no fracturing into fracturing windows. The fracturing into scan-strips cannot be seen since a scan-strip is merely a collection of sub-strips. The final step d) is the merging of each sub-strip. These sub-strips, or rather the fracturing windows, are then passed on to be rasterized (not shown in the figure).

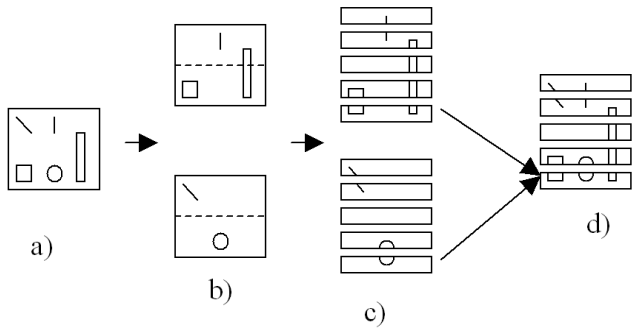

Figure 2.5 A simplified view of pattern data Fracturing

In a real case the input file in step a) would reach 200-1000 Gbyte in size. The number of FMBs in step b) would be 10-30 and the number of buckets approximately 1000 per FMB. At this stage the pattern data is stored to disk and should not have increased in size and would still be 200-1000 Gbyte. The total number of sub-strips in step c) would be 10000-30000 per FMB. In step d) these sub-strips would be merged into a total of 10000-30000. At this stage the data has doubled to 400-2000 Gbyte. See Section 6.3 for more information and an analysis of pattern data.

2.3.1 The pattern data processing hardware

The hardware currently used for pattern data processing is based on three different systems. First is the preprocessing computer; a Sun Fire V880 with four UltraSPARC III processors. It both reads and writes to the same Sun StorEdge T3 in a RAID-0 configuration. The output from this computation is read from the T3 by a separate process on the V880 pushing the data through an optical fiber. The receiving end is a 9u VME system with 26 PowerPC processors from Mercury Computer Systems Inc. The output from these nodes is merged in bulk memory nodes, compare to step d) in Figure 2.5. The data is then passed on from the bulk memory to be rasterized in the FPGAs.

3 System requirements

The goal for this project is to try to give suggestions on how to configure a low cost pattern preprocessing system based on some form of Linux cluster or NUMA architecture. The best way to achieve low costs is usually to use common off the shelf products. This has a second very important benefit; it is easier to find people with previous experience of the components. Commonly used hardware also means a rich selection of software such as optimization and debugging tools which otherwise might be unavailable.

Due to the rather long, 10-year, lifetime of a laser pattern generator it is very important that all spare parts are available. It is not acceptable that a component is made unavailable without an early warning or a compatible replacement.

Other parameters of the systems are however not that critical. There are for instance no extreme demands on the physical size. Hence 19-inch standard rack components are preferred but not a must. Power consumption, noise, vibrations and such are described in [8] and [9]. These standardized documents do not normally affect the selection of normal standard computer equipment, but rather more the way they are mounted in racks. For instance ergonomics, safety markings, remaining a low center of gravity and adding a safety switch if the power consumption exceeds a specified level. However this level of detail is not further considered in this thesis.

3.1 Performance and functionality

In terms of functionality the system must first and foremost be able to process all pattern data that is presented. It is always possible to design a test pattern that could bring any preprocessing system to its knees. Real customer pattern data is a different thing. A worst-case badly designed pattern should still be processed in the correct way. It is however not necessary to do it quickly. Only the expected normal cases needs to be highly optimized. The system must be scalable in order to cope with ever increasing demands.

The single most important requirement presented to this project is the performance of the preprocessing system. The performance is defined as the throughput capacity achieved by the preprocessing program. This is presented in Figure 3.1 as the average and in one case the sustained bandwidths to and from the disks in the system. An average bandwidth is defined as the average throughput capacity achieved during the complete writing time of the laser pattern generator or approximately 5 hours. The sustained bandwidth is defined as a link with a guaranteed capacity. A loss of bandwidth in this link could be tolerated for up to a couple of seconds but not longer; the total average capacity of the link still has to be sustained.

Figure 3.1 only shows a basic functional scheme of the proposed solution. Hence the input disk and the output disk are not necessarily two separate disks for instance. The specified bandwidths of 90 Mbyte/s in and out of the computing cluster is the average over total writing time of the laser pattern generator. The same 90 Mbyte/s bandwidth required for downloading the pattern data to the input disk has also got to be accounted for. The most critical part is the 360 Mbyte/s output from the output disk. It is absolutely critical that this bandwidth can be sustained during the entire write time. Note that all these links must be able to deliver these bandwidth requirements at the same time and not just one at the time.

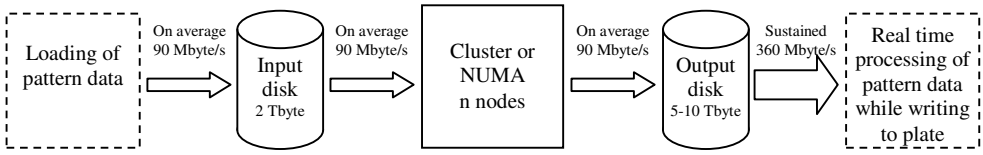

Figure 3.1 A block diagram of the preprocessing system in principle

Figure 3.1 does actually describe more than the preprocessing stage. The preprocessing is just one part of the pattern data processing that the preprocessing system has to handle. The system could be seen as a queue. The preprocessing stage by it self can be seen in

Figure 3.2. In this stage the pattern data is read from the input disk, processed and written to the output disk. We could give the pattern data at this stage queue number *Q*.

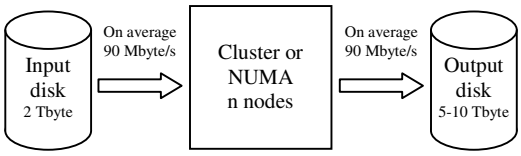

Figure 3.2 A block diagram of the preprocessing stage

The previous step before preprocessing is the loading of the pattern data. This is done from a server not provided by Micronic Laser Systems AB. The data is downloaded to the input disk as seen in Figure 3.3. The corresponding queue number for the pattern data in this stage would be *Q*+1.

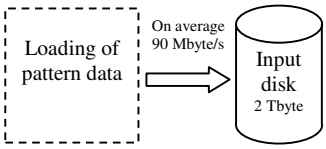

Figure 3.3 A block diagram of the pattern data loading stage

The final stage as seen in Figure 3.4 is done while the laser pattern generator is actually writing the pattern to a plate. This means that this processing stage must have the highest priority. In this stage the pattern data is read four consecutive times from the output disk, hence four times higher bandwidth requirements. This stage could be described as done in real time. However it is not real time as in the correct meaning of the word. There is no need for any real time operating systems. It is actually an on average operation with a slim margin for error. The processes extracting the data from the output disk are buffered and the stage after that is also buffered. All subsystems have a waiting state. The high data rate means that even with large buffers the output disk cannot be unresponsive for more than a number of seconds. With a queue number of *Q*-1 this adds up to a total queue of three simultaneously operating independent processing stages on the same system.

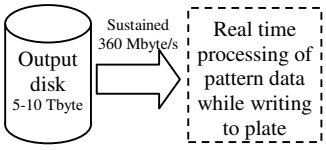

Figure 3.4 A block diagram of the pattern data extraction during write-time

In order to evaluate the requirements on interconnects and CPU boards the cluster/ NUMA box in Figure 3.1 has to be examined more carefully. Figure 3.5 shows the idea of how the preprocessing would work. Some details are not yet specified, for instance if there must be a node collecting pattern data from the processing nodes and handling the output disk. One thing that is clear is that a single node reads the pattern data from the input disk and distributes the data to the processing nodes, compare to Figure 2.5.

It could seem possible to use several processes reading on different locations in the same file. However different hierarchical structures in the file might expand differently and can cause severe unbalance in the distribution. It is not unusual that a single hierarchical structure can cover almost a complete pattern data file. This would mean that all processes would have to read the complete file in order to achieve a proper distribution.

Since there is a clear risk for the input node to become a bottleneck the work done in this node has to be reduced to a minimum. The most likely type of interconnect network used between the nodes would be switched. This means that the interconnect between the input node and the switch has to be able to handle 90 Mbyte/s plus overhead. Assumed that the disk access is not handled by the same interconnect. If a special node is needed to access the output disk then the switch is going to have to handle twice the amount of data. The required amount of interconnection between processing nodes are small or even non-existing.

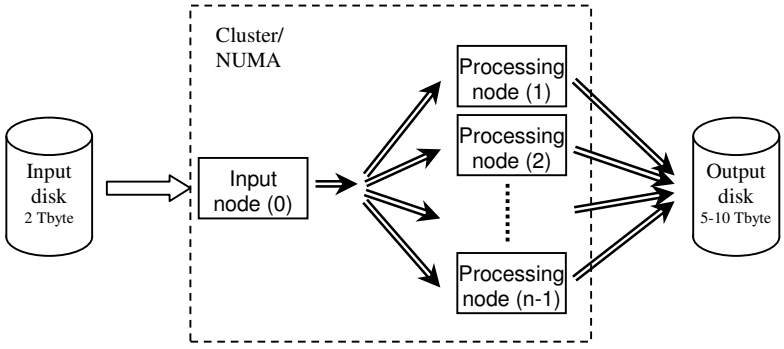

Figure 3.5 The preprocessing cluster/NUMA in detail

Translating the required bandwidth to how much hardware is actually needed demands some considerations. Especially since the program has never been tested on an IA-32 or IA-64 CPU. The fact that the program is very dependent of the pattern data also makes matters more complex. It is however known that the program in its current state is limited by disk-I/O. The proposed solution to this problem is to design a reasonable and scalable cluster or NUMA system and then use an easily portable workload model of the program to verify the design. This method is acceptable since it is expected to result in just a handful of CPUs. Switched interconnects should scale well with the size of the system. Due to the formulation of the requirements, it is possible to verify at an early stage that the disks can deliver the bandwidth during the given circumstances.

The requirements on storage space are quite loosely specified compared to the bandwidth. This requirement is however not to be considered lightly. The input disk must be able to store 2 Tbyte. The output disk is more loosely specified and should preferably be able to store 5-10 Tbyte.

3.2 Software

Due to the general requirements of availability and compatibility it is advisable to use standardized software. Well-known and tested software is preferred compared to specialized low-level solutions, but only if it can be done without any greater loss in performance. It is a great advantage if a software or API is available on more than one platform. APIs for interconnects should preferably be standardized and independent of the underlying hardware. In short, software should be as high level as possible without sacrificing too much performance.

3.3 Summary

- The system should preferably consist of common off the shelf products.
- The type of processor should be x86, set as a preset requirement to this project. Either IA-32 or IA-64.
- The preferred format is standard 19 inch rack mounted equipment.
- The system must be scalable in order to adapt to future demands.
- The input disk has to be able to receive on average 90 Mbyte/s from loading the next pattern and simultaneously deliver 90 Mbyte/s on average to the cluster/NUMA. The storage capacity needed is at least 2 Tbyte.
- The cluster/NUMA must have an interconnect that can sustain the required bandwidth to forward the data read from the input disk plus overhead, see Figure 3.5
- The output disk has to be able to receive 90 Mbyte/s on average from the cluster/NUMA and simultaneously deliver 360 Mbyte/s sustained to the real time processing unit. The storage capacity needed is approximately 5-10 Tbyte.

4 Survey of common off the shelf products

The survey done in this section is not intended to be a complete summary of all available products. It is merely a brief analysis of products that were thought to be relevant or were recommended by other people. This means that not all products in this section proved to be suitable for a preprocessing application, neither does this thesis claim to include all products available that might be of interest for this project. The analysis of each product is in no way complete and is predominantly considered exclusively from a preprocessing point of view.

4.1 OS and software

The operating system Linux is a preset requirement to this project. However no particular flavor is specified. The distribution actually does not have a tremendous affect for the task of pattern data preprocessing. Things that come into question are the availability of updates and support. Some distributions have license fees based on the number of servers. These types of recommendations are not part of this project. However it seems to be a good choice to choose one of the larger distributions such as Red Hat [10], S.u.S.E. [11] or Mandrake [12]. Most of these distributors normally offer an enterprise edition that guarantees continuous updates and support during a number of years. This type of license is often debited per year and not a one time cost.

The only thing that would make a real difference in terms of performance is the kernel. Some of the new features in the Linux 2.6 kernel [13] might give considerable improvements. Some improvements are more important than others such as: Support for the new 64 bit CPUs from AMD. Being able to address disks larger than 2 Tbyte. Address 64 Gbyte of RAM through the support of Intel's Physical Address Extension (PAE) in 32 bit CPUs. Greatly improved the support for NUMA systems. Plus many more other improvements aimed directly for use in large computer systems with many concurrent threads and processes. The drawback of the new kernel is that it is new and it will take some time before all bugs are sorted out.

Message Passing Interface (MPI) is my choice for the communication between the clustered computer nodes. MPI is not a product in itself but rather a library specification. The MPI specification was developed as a joint effort between a number of companies, laboratories and universities. It has a reasonably high level API with possibilities for asynchronous transfers. It is simple to use and is becoming increasingly popular for high performance computing (HPC). As an effect of this, there are appearing more and more profiling tools such as for instance Vampir [14] and other useful development tools. A hardware manufacturer of a specific type of interconnect typically supplies its own set of MPI functions that are highly tuned for that specific type of interconnect. I have in my tests used a free and general distribution of MPI called MPICH. MPICH is a reference implementation of MPI developed by Argonne National Laboratory and Mississippi State University [27]. Since it is possible to execute several processes communicating through MPICH on a single machine it makes it possible to develop programs on small cheap machines. The same code can then later be compiled and executed on a large machine with several clustered computer nodes.

The choice of compiler is not obvious and is not primarily covered by this project. I have used the freely available GNU Compiler Collection (GCC) [15] C-compiler in my tests. A few other examples are Borland C++ Builder X [16] and Intel C++ builder for Linux [17]. My own subjective belief is that the difference between compilers today is not as much the performance, but rather the available support and included development tools. Some of the most appreciated professional compilers/development environments traditionally come from Borland and GNU. In other words much the same questions as when deciding for a Linux distribution.

4.2 Motherboard and processor

The requirements presented in Section 3.3 only give one specific hard demand in this area and that is that it should be some form of x86 based hardware. A general goal of this project is however to use common off the shelf products. The preferred format is standard 19 inch racks. Higher density is not an issue neither is power consumption. Hot swap ability is not required.

The requirements presented to in Section 3.3 gave two alternatives: a clustered system or a NUMA system. I have however discarded the NUMA suggestion at a very early stage. The reason is that a NUMA system with the required performance is more expensive and is more dynamic than what is needed for this application. The preprocessing program does not need to be fine grained and the data is processed in a pipelined fashion. The system can be fitted to suit this particular use case without compromises since no other applications will run on it. A cluster scales more easily than most NUMA systems.

Some specific requirements are not specified in Section 3.3 but are nonetheless important. The number of PCI slots should be sufficient in order to allow for interconnect and disk I/O adapters. The minimum should be at least two high-speed slots. It will most likely be sufficient with two 66 MHz/64 bit slots. This assumption is based on the experience with the Fibre Channel adapter in Section 5. However the most common choice for high performance PCI today is PCI-X, which will give more than enough throughput capacity. It might be a benefit to have more than one PCI-X bus to ensure scalability, but it is not considered a necessity for the system requirements mentioned in Section 3.3. Many blade servers today have integrated Gigabit Ethernet adapters. In the case that Gigabit Ethernet would be used as interconnect it would loosen the requirements to only one PCI/PCI-X slot.

The number of CPUs per motherboard is not critical. Two CPUs could be useful for multithreading each writing process in the preprocessing program, see Section 6.2. More CPUs than two would not be useful other than in an attempt to build a NUMA system. It could however be advisable to use a dual CPU node for the reading process even if the writing processes are running on single CPU nodes, it depends on how much work has to be done in the reading process in terms of load balancing and other pattern data specific compensations.

Perhaps one of the most important requirements is the amount of memory per node, see Section 6.1.1 for further explanation. Each CPU card has to be heavily equipped with memory. However since each node only executes one critical process all memory has to be accessible from that process. This means in the 32 bit case that only 4 Gbyte can be effectively used. This problem does not exist in the case of a 64 bit CPU running a 64 bit program. The minimum requirement should be at least 4 Gbyte per node. Memory bandwidth should be as high as possible due to the I/O intensive nature of preprocessing.

There are other benefits of 64 bit processors than just memory addressing. In general 64 bit calculations are becoming more popular in pattern data file formats at Micronic Laser Systems AB. These calculations can be executed quicker leading to a higher throughput capacity. The drawback is the higher price, however the price difference will most likely decrease with time as the number of 64 bit applications increase.

4.2.1 Suggested products

All these requirements still leave some reasonably cost efficient alternatives. Almost all major computer manufacturers have some suitable 1u or 2u sized servers. Table 4.1 shows a selection of alternatives:

Table 4.1 A selection of motherboards and processors

Model:	Dell [18] PowerEdge 1750	Dell [18] PowerEdge 3250	NEXCOM [19] HDB 44722R3	HP [20] Integrity rx1600	HP [20] Integrity rx2600
Format:	1u	2u	5 vertical blades in 4u	1u	2u
CPUs:	1-2x Xeon 2.4 – 3.2 GHz	1-2x Itanium 2 1.0 – 1.5 GHz	2x Xeon 2.4 –2.8 GHz	1-2x Itanium 2 low voltage 1.0 GHz	1-2x Itanium 2 1.0 – 1.5 GHz
Memory:	Up to 8 Gbyte 266MHz ECC DDR SDRAM	Up to 16 Gbyte 266MHz ECC DDR SDRAM	Up to 4 Gbyte 266MHz ECC DDR SDRAM	Up to 16 Gbyte 266MHz ECC DDR SDRAM	Up to 24 Gbyte 266MHz ECC DDR SDRAM
PCI-X slots:	2x 133 MHz/64 bit	1x 133 MHz/64 bit 2x 100 MHz/64 bit	3x 133 MHz/64 bit	2x 133 MHz/64 bit	4x 133 MHz/64 bit

The Dell PowerEdge 1750 is comparably a low cost server but equipped with two 133 MHz/64 bit PCI-X slots on two separate busses. It is not the fastest server mentioned in this table but it might be cheaper even if clustered with more nodes.

The DELL PowerEdge 3250 is the more expensive alternative from Dell. Dell recommends it specifically for use in HPC applications.

NEXCOM HDB 44722R3 is a double sized version of the normal blade servers suited for NEXCOM´s 4u sized HS 420 chassis. They have been expanded to give room for up to three PCI-X 133 MHz/64 bit slots on a single bus. In total 5 of these blades can be fitted within a 4u sized HS 420.

Due to the limited airflow in a 1u sized server, the HP Integrity rx1600 is equipped with two low voltage 64 bit Itanium 2 CPUs. The limited space also affects the I/O connectivity, only one of the PCI-X slots is full-length the other one is only half-length.

HP Integrity rx2600 is slightly more equipped than the rx1600, with faster CPUs and higher I/O bandwidth to supply all four PCI-X slots.

4.3 Interconnect

The most successful way of building a scalable interconnect fabric is to design it as a switched network. A bus can never scale as well since all nodes have to share the same physical wires. There are problems with switched networks as well. Hot spots can occur especially in switches high up in a tree topology. Using parallel switches in higher levels is a costly but effective way of solving this. The most powerful solution, in theory and when money is no object, is something called a fat-tree topology and is described in section 4.3.4.

The interconnect fabric necessary for this project does not need to be a fat tree topology. Since all pattern data is delivered from one single computer node it is likely that the node or the capacity of the connection between the node and the switch is the bottleneck, rather than the switch itself. The system can therefore never be scaled beyond the capacity of a connection in the interconnect. The amount of feedback data transferred back from the writing computer nodes can be neglected in comparison. This means that there is no real risk of overloading the switch.

4.3.1 Gigabit Ethernet

Gigabit Ethernet (1000BASE-T) [30] is the standard bound to replace Fast Ethernet (100BASE-TX) for computer networks. 100BASE-TX sends three-level binary encoded symbols across a link at 125 Mbaud; thereby achieving 100 Mbit/s. 125 Mbaud is needed since 100BASE-TX uses 4B/5B coding (4 bits of data is sent as a 5 bit code). 100BASE-TX uses one pair for sending and another one for receiving. 1000BASE-T uses the same category 5 cables and the same symbol rate of 125 Mbaud. But 1000BASE-T uses all four pairs for sending and receiving simultaneously and a more sophisticated five-level binary coding. This sums up to 4pairs x 125Mbaud x 2bit/symbol = 1Gbit/s. This is the theory but

in reality 1000BASE-T must compensate for problems caused by echo and crosstalk. Gigabit Ethernet is also available over optical fiber.

Gigabit Ethernet is a very attractive solution for low-end solutions since it is comparably cheap and easy to implement. Most motherboards already have at least one Gigabit Ethernet connection, therefore leaving free PCI slots and keeping the form factor small. The fact that it is possible to connect Gigabit Ethernet to a Fast Ethernet network makes it easy to manage. This allows Gigabit Ethernet to be used as a cluster-interconnect in those cases when the bandwidth requirements are not to extreme and the long latency caused by the IP protocol could be tolerated.

The combination of the IP protocol and Gigabit Ethernet gives a possibility to use several host adapters to access different subnets. Accessing the different subnets is completely transparent from the user level since the routing is done at a lower level.

10 Gigabit Ethernet [30] is the latest standard in line and provides 10 times the bandwidth of normal Gigabit Ethernet. It is currently only available using optical fibers. The higher bandwidth should make this the optimum choice in applications where high throughput is needed and a longer latency is acceptable. Typically running coarse-grained programs with just a few or no synchronizations.

4.3.2 Myrinet

Myrinet is a packet switched fabric and low latency protocol designed by Myricom Inc [24]. The network interface cards use an on board processor to relieve the main processor from work of protocol handling. Historically Myrinet has appeared in bandwidths from 512Mbit/s to 1.28Gbit/s and the latest version supports 2Gbit/s in each direction at full duplex. Dual optical multimode fibers are used for the communication links. Myrinet is an ANSI/VITA standard (26-1998). The link and routing specifications are open and can be downloaded from Myricom Inc's web page.

Myrinet network interfaces are available both in PCI and PCI-X format; the later one is equipped with a slightly faster onboard processor. Boxed switches are available in sizes ranging from 8 to 128 ports. Myrinet is however scalable up to tens of thousands of nodes by combining these switches in tree structures. Each switch is self-installing in the sense that there is no need for routing tables and they are capable of handling multiple paths between hosts. The switches are also available with monitor capabilities through an Ethernet connection.

Myrinet software supports Linux, Windows, Solaris, AIX, MAC OS X, True64, FreeBSD and VxWorks. A number of programming APIs are available for Myrinet but they are all based on an API called GM. GM is built to "bypass" the operating system making it insensitive to what operating system is used. GM only supports a low-level message-passing communication. MPI is also available as a more standardized message-passing interface; supposedly without any major performance drawbacks compared to pure GM. Socket communication is available directly over GM without the TCP/IP stack. However both TCP/IP and UDP/IP is possible to use on top of GM but it is not recommended since it uses a large amount of host processor time. Other types of middleware like VI and PVM are also available over GM; they are however not of interest for this thesis.

4.3.3 SCI

Scalable Coherent Interface (SCI) [28] was based on the 1988 IEEE project Futurebus+. SCI was finished in 1991 and became an open public ANSI/IEEE standard in 1992. It was first thought to replace the traditional processor-memory-I/O bus as well as being a standard for local area network communication. This never became reality. SCI is a switched network with 36 signaling pins per link. The bandwidth has increased over the years and is now hundreds of Mbyte/s. Serial optical fibers are available as an alternative to copper.

It is mentioned in [28] that a lot of people consider SCI to be dead; the author of the web page of course rejects this. It is very difficult to find any information on SCI that is not older than 3-4 years. This fact makes at least me a bit unwilling to explore it further. One way to explore SCI further is to take contact with a provider of SCI solutions such as Dolphin Interconnect Solutions Inc. [29].

4.3.4 Quadrics

QsNet [25] could be considered to be the luxury line of interconnects. Not only due to its performance but most of all due to the high level of service provided by the hardware and thereby offloading the main CPU. QsNet is a 400 Mbaud 10 bit wide packet switched network. A peak bandwidth of 340 Mbyte/s after protocol in each direction is achieved using parallel copper interconnect. QsNet is designed for SMP systems with the standard PCI 2.1 I/O bus.

$QsNet^{II}$ is the next generation using the same 10 bit wide copper connection, now featuring the PCI-X I/O bus and 1.333 Gbaud delivering a peak bandwidth of 900 Mbyte/s after protocol. $QsNet^{II}$ also offers the possibility of optical connections and thereby extending the maximum distance to over 100 m.

QsNet has the ability to perform I/O to and from paged virtual memory. This allows for communication without the need to lock down or copy pages. The data transfer is handled by a DMA engine for the output and a hardware handler for input. A dedicated I/O processor helps to offload the main CPU from protocol handling. The first version of QsNet uses a 32 bit virtual address; this limits the amount of directly accessible memory to 4 GB per process. $QsNet^{II}$ uses 64 bits for virtual addressing and therefore it does not have this limitation.

Each switched QsNet network is built from two basic blocks: the programmable network interface Elan and the communication switch Elite.

Since the network interface Elan is very closely bounded to the hardware the supported hosts are quite limited. IA-32 and the latest IA-64 processor architectures from Intel are supported as well as True64™ for Alpha processors. The Shmem programming library enables get and put operations to be mapped directly to remote read and write hardware primitives. Quadrics MPI is a complement to the NUMA environment provided by Shmem. Quadrics MPI is an optimized version of MPI 1.2 and is based on MPICH from Argonne National Laboratory. One-sided communications, as defined in MPI-2 [26], is also supported but not the complete MPI-2 standard as a whole. In the case that optimum performance is desired despite the loss of portability, it is possible to use Quadrics native communication library – libelan.

A QsNet network is built up from a number of 8 port switches. The heart of each switch is the Elite chip. These switches can be combined into a fat-tree topology that scales the number of nodes in powers of 4, reaching at most 4096 nodes for $QsNet^{II}$ and 1024 for QsNet. In each stage there are 4 different routes up the tree and 4 nodes/switches down. This gives a network with a bandwidth that scales linearly with the number of nodes. Each packet is routed in the least loaded path. This gives good performance as well as reliable redundancy since disabled links will be circumvented. In the case of a broadcast the packet is routed up to the point that the complete broadcast range is reachable. Then the packet is automatically copied and sent down the branches. The acknowledgements from the recipients will be recombined as they go back the same way, so that a broadcast will only succeed when all destination have been reached. This type of hardware broadcast allows for an easy implementation of barrier synchronizations that are properly scalable.

In [26] it is shown in benchmarks that the latency can be as low as 2 µs and the bandwidth as high as 335 Mbyte/s. These are however very brief benchmarks and only concerns QsNet and not $QsNet^{II}$.

4.3.5 InfiniBand

InfiniBand is a serial I/O, channel based, packet switched fabric developed by the InfiniBand Trade Association [21]. The InfiniBand Trade Association was formed in August 1999 by a group of 7 companies. Currently it consists of more than 190 companies including some really well known ones such as: Hewlett-Packard, IBM, Dell Computer Company, Hitachi, Intel Corporation, Motorola Computer Group, Sun Microsystems and many more. The first specification was finished in October 2000. IDC has estimated that 50% of all servers will be using InfiniBand by 2005.

InfiniBand is offered in three different bandwidths 2.5 Gbit/s and 10 Gbit/s today and a road map up to 30Gbit/s. It is also possible to use it not only on boards but both copper

(<17m) and optical fiber (<10km) as well [21]. InfiniBand uses its own protocol, covering the physical, link, network and transport layers. The protocol also features 128 bit addresses, with 16 bits for each subnet. This protocol also enables a very low latency as well as features like remote direct memory access (RDMA). Other projects like SVP, which aims to map SCSI over InfiniBand, are also forming [23].

In the initial state, InfiniBand is being deployed on PCI-X adapter cards. However the members of InfiniBand Trade Association expect to start producing native InfiniBand implementations as the technology evolves. All the members will then produce a wide variety of compatible products [21].

Mellanox Technologies [22] is a company specialized on delivering InfiniBand hardware solutions. Their current InfiniHost device is supported by several MPI software sources: MPI/Pro from MPI Software Technology, OSU MPI from Ohio State University, NCSA MPI from NCSA and Scali MPI Connect from Scali. Scali also deliver software for cluster management.

4.3.6 Summary

A short summarizing table of comparison is presented in Table 4.2. The table shows the bandwidth with the protocol overhead included, with an exception for Quadrics QsNet. The latency is also shown in those cases the latency is presented by the manufacturer. The medium of the interconnect is shown in the table, whether it is copper wires or optical fibers. The row labeled network specifies what type of network structure it supports, whether the network is switched or not and if the network supports multiple paths. The table also shows if specialized MPI libraries are available. In some cases the manufacturers themselves provide a specially designed MPI library that is highly tuned specifically for their type of interconnect.

There are in some cases several generations of the same basic interconnect. The different generations are presented on individual lines in those cases, except for the network and specialized MPI rows.

Table 4.2 A comparison of the available interconnects

	Gigabit Ethernet	**Myrinet**	**SCI**	**Quadrics QsNet**	**InfiniBand**
Bandwidth:	1 Gbit/s 10 Gbit/s	512 Mbit/s 1.28 Gbit/s 2 Gbit/s	1.6 Gbit/s	2.7 Gbit/s 7.2 Gbit/s (After protocol)	2.5 Gbit/s 10 Gbit/s 30 Gbit/s
Latency:	~50 µs ?	10 µs 10 µs ?	5 µs	2 µs	? 4.5 µs ?
Medium:	Copper/Fiber Fiber	Fiber Fiber Fiber	Copper/fiber	Copper Copper/Fiber	Copper/Fiber Copper/Fiber Copper/Fiber
Network:	Switched Multiple paths	Switched Multiple paths	Switched ?	Switched Multiple paths	Switched Multiple paths
Specialized MPI:	No	Yes	?	Yes	Yes

The different latency values presented in this table are measured, under what must be presumed to be slightly different conditions, by the individual manufacturers. The latency values should therefore only be considered to be approximate. The latency is commonly measured when transferring a small message of just a few bytes.

It is possible to argue about whether Gigabit Ethernet supports multiple paths or not. A Gigabit Ethernet switch in a LAN can support aggregated links, according to the IEEE 802.3ad standard, while a large Internet router may support multiple paths in general.

4.4 Disk

The disks in a preprocessing system are traditionally the given bottleneck. It is also by far the most expensive part of the whole system. The disk products discussed in this section

will be primarily the output disk, described in Section 3.1. Even though some of the information gather in this section might be of use for the input disk it is not directly addressed.

The basic idea is to use some form of host adapter in each of the computer nodes. Fibre Channel is a not too unlikely type of adapter that could be used. This means that all writing nodes are connected to the output disk individually.

It is not a necessity that the output disk is a single disk system. The disks do not have to work as a switch, which means that each disk system can be treated as a closed channel. A channel with a writing node in one end and an extraction node in the other. Splitting the output disk into several smaller disk systems might also save money since the price for disk arrays often increase almost exponentially relative the performance.

4.4.1 RAID in general

Redundant Array of Independent Disks (RAID) is the most common way of achieving high redundancy and bandwidth with disks. Just a Bunch Of Disks (JBOD) is the base of any RAID. The point of interest is not the disks but rather the storage processor. The storage processor is often the bottleneck in larger high performance RAIDs, depending on how the different RAID levels are implemented in the storage processor. Below follows a short reminder of the different RAID levels [33]:

RAID level 0: This is not really a true RAID since there is no redundancy. The data is striped over the available disks, resulting in a very high throughput. Since no redundant information is stored the available disk volume is used with maximum efficiency. The storage processor only has to push data without doing any calculations and can therefore be quite simple and cheap.

RAID level 1: The opposite of RAID level 0, all data is duplicated over all available disks. The throughput is not increased compare to a single disk but with good redundancy. Since the data is just duplicated as it is, there is no need for a fast storage processor and there is no reconstruction necessary if a disk would fail. The disk volume is used very inefficiently since the storage capacity never will be greater than one single disk.

RAID level 2: This method could be used when the disk lacks a built in error correction. Each word that is written to a disk generates a Hamming error correction code (ECC) that is saved on separate disks. Each time a word is read the respective ECC is read, compared and if necessary the word is corrected. This is hardly ever used since any modern SCSI disk has built in error correction.

RAID level 3: Similar to RAID level 4, but working on a byte level rather than blocks. Each byte has its own parity saved on a separate disk. This demands for specialized hardware in order to get a high throughput. But with a specialized hardware it is faster than RAID level 4 for small random writes since the parity does not need to be calculated on complete blocks. Otherwise the pros and cons are about the same as for RAID level 4.

RAID level 4: The data is striped on a block level over several disks and the parity is saved on one specific disk. This allows for data to be rebuilt in case one disk fails, it is however both difficult and inefficient. Writing requires a fast storage processor in order to get a high throughput, especially with small random writes. Reading is much faster and is comparable with RAID level 0. The disk volume is used quite efficiently due to the low number of parity disks.

RAID level 5: Striped on a block level, similar to RAID level 4 but without a specific disk for parity. The parity is distributed over all disks instead of on just one. The disk volume efficiency is the same as for RAID level 4. The time for rebuilding lost data might be faster since parities could be read from different disks in parallel.

RAID level 6: Very similar to RAID level 5 but with a two-dimensional parity distributed over all disks. This allows for continuous operation even in case of multiple disk failures. These parity calculations sets very high demands on the storage processor and results in a lower write performance and lower disk volume efficiency compare to RAID level 5.

RAID level 7: Unlike the other RAID levels this one is not an industry standard. It is a one vendor proprietary solution. It could be described a combination of RAID level 3 and 4 but enhanced in order to counter the downsides of these RAID levels. A great deal of cache is added and a real time processor handling the parity asynchronously. This is a fast and efficient solution but also expensive and only supported by one vendor.

Dual levels: RAID levels can be combined in more than one level. For example two RAID level 1 arrays can be combined as a RAID level 0. In this way combining the strengths of both types, this combination is often called RAID level 0+1 or 10. Other used combinations are 50 and 30, the later one is often mistakenly called 53.

4.4.2 Fibre Channel in general

Fibre Channel [34] is designed to allow constructions of Storage Area Networks (SANs). The underlying fabric can either be single or multimode optical fiber, but also supports twisted pair and coaxial cable. The structure of the fabric can either be directly point-to-point, switched or an arbitrated loop. A Fibre Channel loop can at most include 127 nodes.

The Fibre Channel Protocol (FCP) uses serialized SCSI commands inside Fibre Channel frames. IP is also used to allow for SNMP network management. Fibre Channel uses the same physical layer as Gigabit Ethernet and an 8B/10B encoding. In general, FCP is optimized for large block transfers as opposed to IP, which is optimized for small blocks.

The most common transmission rate is 2.125; which gives a total throughput of 400 Mbyte/s for a duplex connection. There are also specified versions of up to 2400 Mbyte/s mentioned on [34].

4.4.3 Dell|EMC CLARiiON CX-series

The Dell|EMC CLARiiON CX-series [18] [35] is a disk array capable of operating as direct attach storage (DAS) or in a storage area network (SAN) or to Dell PowerVault network attach storage (NAS) (only the CX600). All disk arrays in the CX-series support hot swap, hot sparing and RAID 0, 1, 10, 3 and 5. A standby power supply is available in order to protect the data in the cache in case of a power failure. The CX200 and CX400 are both upgradeable without disruption. For instance a CX200 can be upgraded to a CX400 or a CX600 without loosing any data or turning of the system or stopping host access to data.

The Dell|EMC CLARiiON CX200 can hold a maximum of 30 Fibre Channel disks in two separate 3u sized units. Two separate Fibre Channel interfaces can be accessed through two switches giving a maximum of four directly connected servers. It is however possible to attach up to 15 hosts to a single disk array in a SAN. The two storage processors can handle at most 15000 I/O operations/s and a bandwidth of 200 Mbyte/s. Each storage processor is equipped with an 800 MHz Intel Pentium III and 512 Mbyte of cache, i.e. a total of 1 Gbyte of cache in a storage system. A smaller version of the CX200 is also available. It has only one storage processor and is able to handle at most 15 disks.

The Dell|EMC CLARiiON CX400 is a larger version with up to 60 disks in a total of four 3u sized units. Four separate Fibre Channel interfaces are available on the front. Connectivity between the 3u sized disk units is done through 4 separate Fibre Channels along the back. The two storage processors can handle up to 60000 I/O operations/s and a bandwidth of 680 Mbyte/s. Each storage processor is equipped with an 800 MHz Intel Pentium III and 1 Gbyte of cache, a total of 2 Gbyte in a storage system.

The Dell|EMC CLARiiON CX600 is the largest version with up to 240 disks in a total of sixteen 3u sized units. Four separate Fibre Channel interfaces per storage processor available, i.e. a total of eight separate Fibre Channel interfaces on the front. Connectivity between the 3u sized disk units is done through 4 separate Fibre Channels along the back. The two storage processors can handle up to 150000 I/O operations/s and a bandwidth of 1300 Mbyte/s. Each storage processor is equipped with dual 2 GHz Intel Pentium IV Xeon processors and a maximum of 4 Gbyte of cache, a total of 8 Gbyte in a storage system.

Right at the end of this thesis project a new improved series of disk arrays replaced the old one. The CX200, CX400 and CX600 were replaced by the CX300, CX500 and CX700 respectively.

4.4.4 Just a Bunch Of Disks (JBOD)

VMETRO [36] is a provider of board-level solutions for high-performance embedded real-time systems. Their solutions are based on industry standards like VMEbus, RACE++/ RACEway, PCI-X/PCI and Fibre Channel. They offer, among other things, a series of real-time data recorders based on a Fibre Channel JBOD and an interface adapter. These recorders are normally used to record raw data from radars, sonars etc. This type of recorder could be adapted to fit the needs for pattern preprocessing.

Each JBOD consists of 7+7 Fibre Channel disks on a split backplane. The data is striped over all disks in a RAID level 0 configuration for maximum performance. It is possible to connect up to 4 separate 2Gbit/s Fibre Channels to a JBOD. The disks could be accessed through a common Fibre Channel adapter or via VMETRO's own designed Custom Programmable – MIDAS Data Recorder (CP-MDR). The CP-MDR is based on the VME form factor and can be fitted with a variety of connections like RACE++ and Serial-FPDP.

The disk solution offered by VMETRO [36] is based on the idea of three separate JBODs. At the input side are all JBODs accessed through the same 2Gbit/s Fibre Channel loop via a PCI Fibre Channel adapter on the SUN/PC hardware. The output from each JBOD would consist of dual 2Gbit/s Fibre Channels per JBOD, one per half of each backplane. These would be leading to three parallel CP-MDR cards as an interface with one 2.5Gbit/s Serial-FPDP per card as the output.

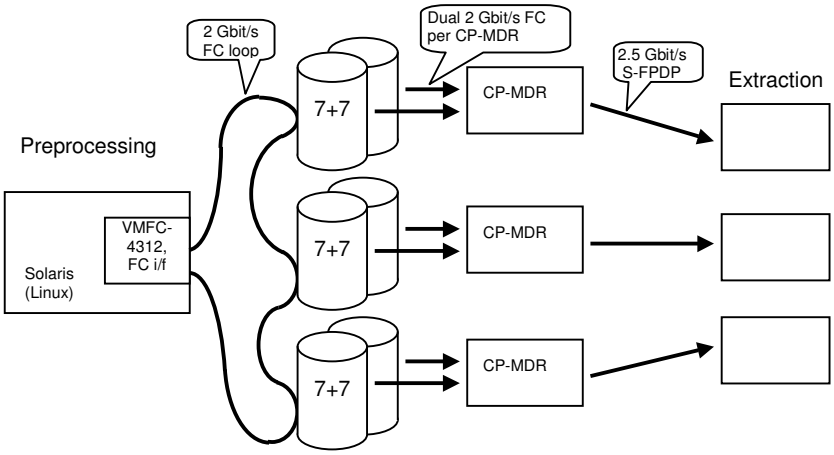

Figure 4.1 A specialized JBOD solution by VMETRO

The reasons for the CP-MDR cards are not only due to performance but also that the extraction nodes on the Mercury system lack the option of Fibre Channel connectivity. VMETRO offers a specially designed API designed for the Mercury platform. This will allow for the extraction nodes to request data directly from the CP-MDRs.

From a Solaris/Linux point of view the disks appear as unformatted individual Fibre Channel disks. The disks are accessed through a software API that stripes the data over the raw disks. A separate lightweight file system per JBOD, only accessible through the API, is used. Files cannot be fragmented and must therefore be pre-allocated before starting to write. A file can however either be truncated or extended depending on the needs. When extending a file there must not be another file directly following the current file or it will be overwritten. Each file is described in the file allocation table by a start position, an end position and a number of attributes. The benefit of this lightweight file system is the possibility of high and predictable performance.

There are a few other values that could be used in order to tune the performance of the system. The size of the stripe blocks could be reduced in order to distribute the workload among the disks. On the other hand smaller stripe blocks could be expected to give a reduced performance per disk. Enlarging the read and write blocks from the application

would also lead to better possibilities for load distribution and performance. A quite unique feature available in this API is the possibility to do disk operations asynchronously. Each disk operation is set up similarly to a DMA transfer. Hence by using a method of multiple buffers it would be possible to set up a number of concurrent asynchronous disk operations. Queuing a larger amount of the asynchronous read and write requests would lead to a better load balance, since they could be executed concurrently over the disks. This method for load distribution combined with larger read and write blocks would reduce the need for smaller stripe blocks, hence increasing the performance of each individual disk.

VMETRO [36] has made a preliminary benchmark based on the demands for a pattern preprocessing system. The test was conducted on a single JBOD with 7+7 disk, one Fibre Channel input and dual Fibre Channels converted to a single Serial-FPDP output, i.e. one third of the complete system described above. The benchmark was done with simultaneous reads and writes of 1Mbyte blocks. The result was 90Mbyte/s sustained writing and 225Mbyte/s sustained reading. However VMETRO warns that the writings could be limited on a PCI card and they mention figures around 85Mbyte/s on SUN hardware. These figures for an individual channel are highly relevant since the system scales linearly with size. If the performance is too low another JBOD could be added without any major overhead. These benchmark figures show that the performance needed for pattern preprocessing could be achieved with just two JBODs, however VMETRO recommends one extra JBOD as a safety margin.

In order to use the performance offered by this system the preprocessing software has to be adapted, both in terms of the API but also to balance the load between the JBODs. The load balance could be satisfied by adjusting the number of FMBs (described in Section 2.3) to be evenly divided by the number of JBODs.

4.4.5 Distributed local disk

An interesting idea to solve the scalability issue of the storage problem is to distribute the disks among the computer nodes [37]. This can either be done with all disks connected in a dedicated storage area network, called a shared-disk architecture or it can be done with a general purpose network, called partitioned-disk architecture.

The most interesting version in this case is the partitioned-disk architecture, since there is already a need for a high capacity interconnect. The usual way of doing this is select a number of nodes to be disk servers. All data written to the disks are then load balanced among these disk servers. This includes metadata as well. It is possible to distribute the data using some RAID level in order to gain redundancy.

It is also feasible to make all nodes active disk servers. This will increase the storage capacity but also increase the workload for the processors. Software controlling this type of distributed disk system is quickly getting very complex. How to reach and view such a file system from outside the clustered computer nodes is not obvious.

There are however a few practical problems. A normal 1-2u sized server only has room for 2-3 SCSI disks. An estimated preprocessing system only needs a handful of computer nodes. This means that the number of computer nodes needs to be increased just in order to get enough storage space. Assuming that the input disk and the output disk specified in Section 3 are implemented as a distributed disk solution. This would mean that the distributed disks would have to deliver 90 Mbyte/s + 360 Mbyte/s = 450 Mbyte/s while receiving 90 Mbyte/s + 90 Mbyte/s = 180 Mbyte/s, in total a combined capacity of 630 Mbyte/s. Hence it is likely that the system has to be scaled up considerably in order to deliver enough throughput capacity to and from disk.

I have decided not to investigate more into this matter due to extreme difficulties in predicting a possible throughput capacity and to the unnecessarily complex nature of the solution. Distributed disks are not specifically aimed for applications where data is read only once in one end and written in the other. Applications suited for distributed disks are more like for instance large databases.

4.5 Conclusions

A general advice when building the computer cluster would be not to build it symmetrically but to use a computer node that is as fast as possible for the reading process and computer nodes with a lot of memory for the writing processes.

It should give enough performance to build a cluster using a handful of servers equipped with dual Xeon processors and standard Gigabit Ethernet as the interconnect. In order to guarantee the performance it might be necessary to use more than one Gigabit Ethernet adapters from the input node and divide the processing nodes into subnets, illustrated in Figure 4.2 (left). This would mean that the available bandwidth between the subnets is quite limited, but this is acceptable since there is no obvious reason for the processing nodes to interact with each other. The scalability of this solution largely depends on how many Gigabit Ethernet adapters can be added to the input node and how much performance the input node can deliver. For a reasonably small number of processing nodes it might be enough with one subnet, one switch and an IEEE 802.3ad compatible adapter card with two or four load balancing ports, illustrated in Figure 4.2 (right). A general assumption for all these cases is that one Gigabit Ethernet adapter per processing node gives more than enough bandwidth to provide the processing nodes with pattern data.

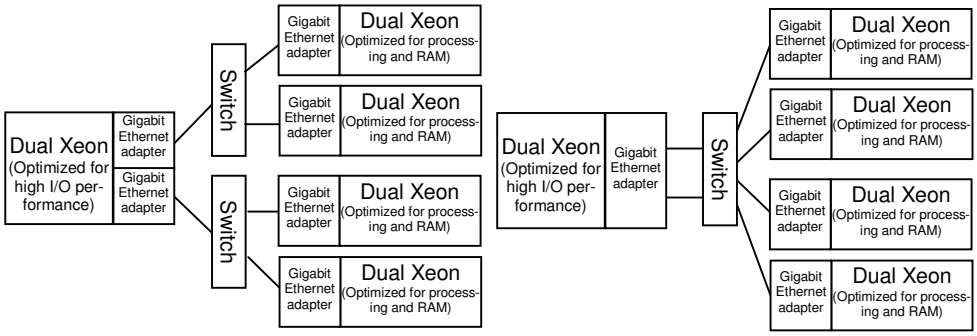

Figure 4.2 A computer cluster using two separate subnets (left) and a computer cluster with a single net but with a load balancing Gigabit Ethernet adapter (right)

If it becomes necessary to build an even more scalable system my choice would be to use a combination of Itanium 2 processors and InfiniBand. For instance a HP rx2600 for the reading process and a few HP rx1600 for the writing processes.

When 10 gigabit Ethernet becomes widely available it might be the optimum choice as interconnect for preprocessing. Common for all these configurations is that the same properly written code with MPI can be compiled and executed on any of these systems.

The hardest challenge is however not the choice of computer nodes or interconnect but the disks. DELL|EMC, see Section 4.4.2, proposed a disk solution for the preprocessing system based on a CX200 with 15 disks as the input disk and a CX600 with 105-120 disks as the output disk. Since the solution was based on slightly misunderstood grounds, I will not go deeper into it. Even though the misunderstanding has been cleared out they have not delivered a new proposition before the end of this project. The new solution for the output disk would at least not be smaller and the already proposed solution costs about three times as much as the VMETRO solution, presented in Section 4.4.4.

Since the cost of disk arrays, like most other computer hardware, increase almost exponentially with performance; it might be a good idea to distribute the workload over several smaller disk arrays. One example of doing this would be to connect one CX200 to each computer node running a writing process. DELL|EMC mentioned, in their presentation of the previously mention solution, that a CX200 with 15 disks can handle at least 90 Mbyte/s in and out simultaneously. This means that four moderately equipped CX200s would be enough in terms of performance and storage capacity and it would be a completely scalable solution. The same idea could also be realized with one or two CX400 disk arrays. The immediate drawbacks are the increased maintenance for the larger number of independently configured subparts and the increased demands of proper load balance between the writing processes. The last problem is however already unavoidable in the case of the VMETRO solution.

The most cost effective solution is the one from VMETRO. Perhaps one of the most important benefits of this solution is that the pattern data is read and forwarded by CP-MDR cards instead of by one or several computer nodes. This means that it is more predictable and easier to guarantee the throughput capacity of the system. The system is originally designed for a very similar task and does not include unnecessary and costly features. The solution is however not without drawbacks, the lack of a fragmenting file system is most noticeable and increases the development time and costs for the preprocessing program. The solution lacks the possibility for any other RAID level other than RAID 0.

5 A preview of the JBOD

VMETRO [36] agreed to give a demonstration of their custom designed JBOD solution. A closer description of the solution and the included parts are presented in Section 4.4.4. Note that this is only to be considered to be a preview of the specialized JBOD. Due to some limitations in the software used the disks could not be loaded with enough writing operations, thereby limiting the relevance of these tests. New software are however under development at VMETRO.

This preview aims to verify the possibility of using this JBOD solution as the output disk, as described in Section 3.1. Thus meeting the requirements of 360 Mbyte/s reading and 90 Mbyte/s writing concurrently. Each test setup represents one third of the complete system, hence the required goal is 120 Mbyte/s reading and 30 Mbyte/s writing concurrently.

Kjartan Mikkelsen, Bengt-Olof Larsson and I did the tests during December 2003 and January 2004 at Micronic Laser Systems AB in Täby, Sweden. Kjartan Mikkelsen is a developer from VMETRO and Bengt-Olof Larsson is from VSYSTEMS, which is an affiliate of VMETRO.

5.1 Test setup 1

This test setup, as seen in Figure 5.1, was provided entirely by VMETRO.

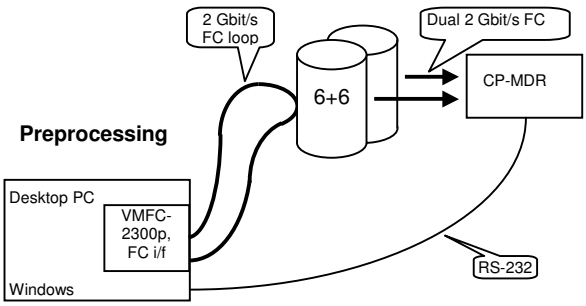

Figure 5.1 Test setup 1 of the JBOD preview

5.1.1 Hardware

Due to the linear nature of the solution during scaling it was considered enough to test the performance of one JBOD and one CP-MDR.

The JBOD (EL-SB2G) was fitted with 12 Seagate Cheetah 10000 rpm disks. Ten disks were 73 Gbyte in size (ST373405FC, firmware rev 38) while two where 36 Gbyte (ST336605FC, firmware rev 0002). The disks were set up as a single disk group, without splitting the backplane.

A VMFC 2300p (QLogic) Fibre Channel host adapter was installed in a desktop PC on a 33 MHz 32 bit PCI slot. The PC was equipped with a 500 MHz Pentium III processor and 100 MHz sdram. The desktop PC was installed with Windows 2000 professional.

A CP-MDR equipped with a Fibre Channel PMC was controlled by the desktop PC through a RS-232 connection. A VME rack was used in order to power the CP-MDR.

5.1.2 Software

Since there were no extraction computer to acquire any data, all data requests by the CP-MDR was generated on the CP-MDR itself. These requests were generated by a program loop on the CP-MDR. The program was controlled through the RS-232 connection. This program is part of VMETRO's standard set of test tools. In order to load the disks properly read positions were varying between the start and the end of a 20 Gbyte file.

The available variables for the tests were the read block size, the number of concurrent asynchronous read operations and the number of read iterations. The stripe block size was fixed to 512 kbyte.

A C++ program with the CP-MDR API running on the desktop PC was used to write to the JBOD. The reason for using Windows instead of Linux is that the CP-MDR API has not yet been adapted for Linux. Since VMETRO had expressed that the PCI bus on the desktop PC was expected to be the limiting factor, no great efforts were made to achieve high performance but rather just to load the disks as much as possible. Each write operation was done synchronously with a 1 Mbyte block varying the position between the end and the start of the 20 Gbyte target file. Just the number of write iterations was altered during the tests in order to match the time of the read sequence.

5.2 Tests

In Table 5.1 and Figure 5.2 follows the measured results of the tests, as the CP-MDR and the program running on the desktop PC presented them. The two main parameters for the test were the read block size and the number of concurrent read operations. The third parameter, the number of read iterations, was increased as much as possible within reasonable execution time. The resulting values from the tests were the read bandwidth and the standard deviation of the read bandwidth. The write bandwidth was also measured, however the values are not that relevant due to the adverse conditions under which the write operations were conducted.

Table 5.1 Test results from the preview of the specialized JBOD.

Block size /(Kbyte)	Concurrent reads	Iterations	Read /(Mbyte/s)	Std /(Mbyte/s)	Write /(Mbyte/s)
1024	8	500	242	3,9	24,9
1024	16	500	316	10,2	26,7
1024	32	500	377	8,1	28,1
1024	64	500	377	13,2	27,9
512	8	1200	186	1,4	34,0
512	16	1200	248	6,3	30,5
512	32	1200	330	6,0	29,5
512	64	1200	373	14,4	24,8
256	8	1800	134	21,0	38,0
256	16	1800	194	36,2	31,3
256	32	1800	229	3,6	26,2
256	64	1800	315	8,4	29,3
128	8	2500	90	12,7	41,3
128	16	2500	108	5,1	36,4
128	32	2500	138	4,3	38,3
128	64	2500	185	3,6	38,4
64	8	3000	73	19,0	43,2
64	16	3000	89	28,3	43,0
64	32	3000	128	41,0	50,0
64	64	3000	117	12,4	40,7
32	8	5000	56	17,0	47,2
32	16	5000	64	20,9	44,0
32	32	5000	80	27,6	43,6
32	64	5000	85	28,7	40,7

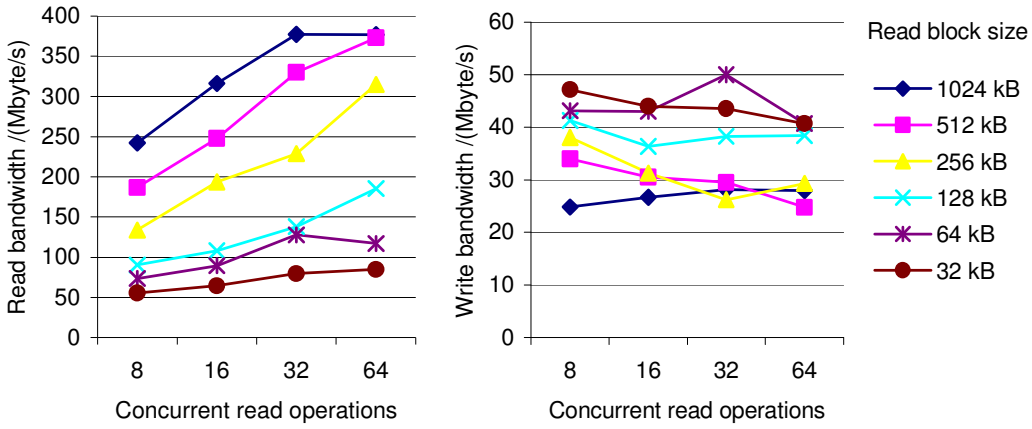

Figure 5.2 The read bandwidth (left) and the write bandwidth (right) as a function of read block size and read operations.

5.3 Analysis

The high standard deviation values and variations of the same could be explained by the method in which the test programs were executed. Both the test loop on the CP-MDR and the desktop PC were started manually. Hence they were not always started at the exact same time. The execution times, for the two programs, were matched manually by adjusting the number of read and write iterations. If this is not done perfectly one of the programs has a short period of time with unrestricted disk access.

VMETRO mentioned during the tests that they in the best conditions experience a combined, read plus write, bandwidth of 420 Mbyte/s. This is roughly what these tests show as well. A read bandwidth of over 350 Mbyte/s, as seen in these tests, should not be confused with how much real bandwidth the CP-MDR can deliver on the output. The 2.5 Gbit/s S-FPDP first of all limits the output. But there is also an internal limit; the internal PCI bus on the CP-MDR does not handle more than 220 Mbyte/s. This leaves a theoretical 200 Mbyte/s of the combined bandwidth in the JBOD for writing.

There was, at the time for this test, no API that could generate several concurrent write operations. This in combination with the supposedly slow PCI bus limits the bandwidth. Sequential write operations are affected more easily by read operations than the average of several concurrent write operations.

5.4 Test setup 2

VMETRO and Micronic Laser Systems AB combined equipment to provide the second test setup. The most noticeable differences between this setup and the previous one is that this time Micronic Laser Systems AB provided a Sun Fire V880 with a considerably faster PCI bus. The disks were also replaced, since VMETRO was unable to bring the same disk as used in the previous session. The number of disks was also slightly reduced from 12 in the previous test to 10; this should be compared to a fully equipped JBOD that has 14.

The reason for this second test was to lay more effort into writing to the disks. To verify that it really would be possible to push 90 Mbyte/s of data to the JBODs in the final system.

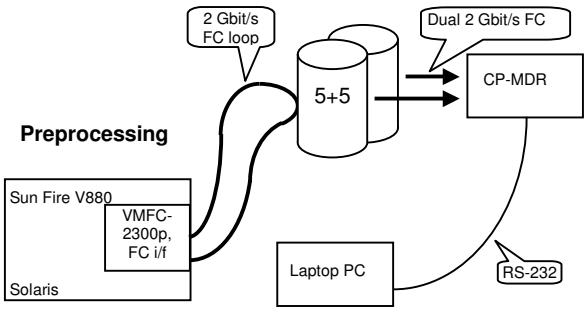

Figure 5.3 Test setup 2 of the JBOD preview

5.4.1 Hardware

Like in the previous test the performance was tested with just one JBOD and one CP-MDR.

The JBOD (EL-SB2G) was fitted with 10 Seagate Cheetah 10000 rpm disks, 36 Gbyte in size (ST336607FC, firmware rev 0004). The disks were set up as a single disk group, without splitting the backplane.

A VMFC 2300p (QLogic) Fibre Channel host adapter was installed in a Sun Fire V880 on a 66 MHz 64 bit PCI slot. The Sun Fire V880 was equipped with two UltraSPARC III processors at 1050 MHz and 4 Gbyte (150 MHz and 8-way interleaved) of DIMMs. Sun Solaris 9 was installed.

A CP-MDR equipped with a Fibre Channel PMC was controlled by a laptop PC through a RS-232 connection. A VME rack was used in order to power the CP-MDR.

5.4.2 Software

Since there was still no extraction computer to acquire any data, all data requests by the CP-MDR was generated on the CP-MDR itself. This program is part of VMETRO's standard set of test tools. In order to load the disks properly read positions were varied between the start and the end of a 20 Gbyte source file. The possible parameters for the tests were the read block size, the number of concurrent asynchronous read operations and the number of read iterations. The stripe block size was fixed to 512 kbyte.

A C++ program, using an old version of the CP-MDR API running on the Sun Fire V880, was used to write to the JBOD. The old API did not support asynchronous disk operations. This off course limits the practical writing performance. A second problem was that VMETRO did not have a suitable test program for Solaris that could generate data from RAM and write directly to the JBOD. The only program available copied a file from the local disk on the Sun Fire V880 and wrote it to the JBOD. When this was clarified at the day of the test a programmer at VMETRO in Norway started to make a "quick and dirty" fix. Unfortunately the fixed program that could write directly from RAM arrived too late to enable any extensive test series but we were able to do at least a few. This program still used the old API and was therefore not able to do asynchronous disk operations.

5.5 Tests

The first tests presented in Table 5.2 and Figure 5.4 was done with the first program that read from the local disks on the Sun Fire V880. A temporary file of 1 Gbyte was created on the Sun Fire V880 and transferred repeatedly to the JBOD. There were no reasonable way of starting and stopping both the reading and writing programs at exactly the same time since they were executed from different machines. The solution was to do each measurement twice. First with unlimited writing time and limited reading time and then vice versa. This means that the number of iterations listed in Table 5.2 only represents the limited read operations. The write measurements are the results of transferring the 1 Gbyte file once.

Table 5.2 Test results from the second test of the JBOD

Block size /(kbyte)	Concurrent reads	Iterations	Read /(Mbyte/s)	Std /(Mbyte/s)	Write /(Mbyte/s)
1024	8	1000	288	3,4	25,5
1024	16	1000	360	4,9	12,0
1024	32	1000	409	1,9	4,5
1024	64	1000	399	3,4	4,2
512	8	2500	203	2,2	34,3
512	16	2500	290	2,8	28,9
512	32	2500	360	2,2	16,3
512	64	2500	403	1,7	8,5
256	8	3000	140	3,1	37,9
256	16	3000	218	3,4	37,9
256	32	3000	308	0,8	26,8
256	64	3000	361	3,1	18,4
128	8	4000	120	2,3	61,1
128	16	4000	156	6,1	52,4
128	32	4000	242	6,7	37,9
128	64	4000	316	6,6	28,9
64	8	4000	91	6,4	68,7
64	16	4000	64	2,4	57,9
64	32	4000	91	1,7	55,0
64	64	4000	133	3,8	52,4
32	8	5000	62	1,2	47,2
32	16	5000	64	2,3	44,0
32	32	5000	53	0,9	43,6
32	64	5000	68	3,4	40,7

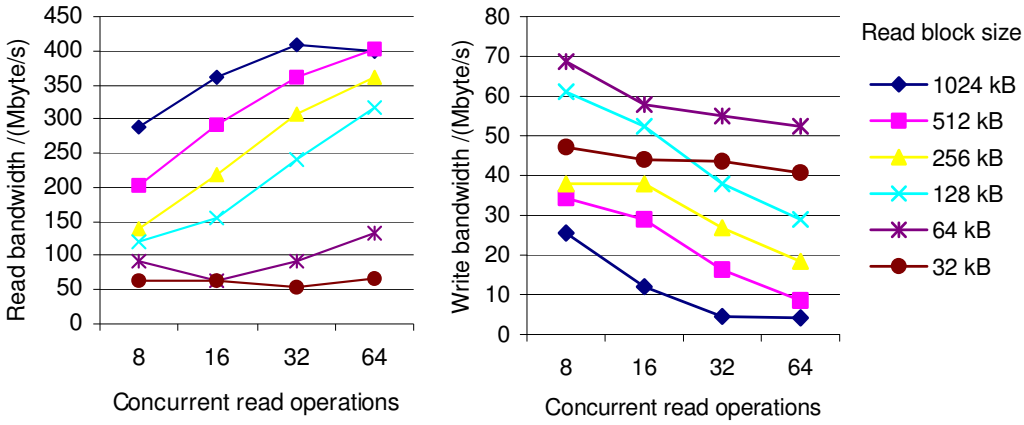

Figure 5.4 The read bandwidth (left) and the write bandwidth (right) as a function of read block size and read operations.

Table 5.3 shows the few measurements we had time for with the fixed program. This program can be considered to be equal to the program used on the desktop PC in Section 5.1. One of the tests done was to write to the JBOD without reading at the same time. This was done in order to get an idea of the maximum capacity of the Fibre Channel host adapter and the Sun Fire V880. It is however likely that the disks could still be the limiting factor. The reason is that with 1 Mbyte write blocks and 512 kbyte stripe size only two disks were accessed at each disk operation.

Table 5.3 Test results when writing directly from RAM

Block size /(kbyte)	Concurrent reads	Iterations	Read /(Mbyte/s)	Std /(Mbyte/s)	Write /(Mbyte/s)
512	8	2000	209	7,5	42,9
256	8	3000	194	4,1	56,5
0	0	0	0	0	107,4

5.6 Analysis

This method of doing each measurement twice drastically reduced the high levels of standard deviation seen in the first test; compare Table 5.2 with Table 5.1.

Studying Table 5.3 makes it clear how much the synchronous write operations suffer from the asynchronous read operations. The right-hand graph in Figure 5.4 shows the same thing with a write bandwidth of 4 Mbyte/s during the highest read loads.

It should be emphasized again that a read bandwidth of 350-400 Mbyte/s is not possible to get out from the CP-MDR. The local PCI bus on the CP-MDR is the narrowest bottleneck with a bandwidth of 220 Mbyte/s. This means that it should be plenty of disk bandwidth left for writing.

A write bandwidth of over 100 Mbyte/s when not loading the disks with read operations, proves that neither the Fibre Channel host adapter nor the Sun Fire V880 is the limiting factor. This means that the system could be scaled beyond one JBOD in order to achieve higher performance. This also indicates that it is likely that the write performance would increase considerably if asynchronous write operations could be used.

5.7 Conclusions

The specified demands of 120 Mbyte/s reading and 30 Mbyte/s writing bandwidth per JBOD are clearly within reach. This refers to the in Section 3.1 defined requirements of 360 Mbyte/s read and 90 Mbyte/s write bandwidth divided by three.

It is possible to improve performance by adjusting the different parameters, as predicted in Section 4.4.4. It is however apparently not a matter of fine-tuning but rather a necessity in order for the whole concept to work.

The limited capacity of the CP-MDR and the S-FPDP interface guarantees that the write operations cannot be starved. Neither the single 2 Gbit/s Fibre Channel loop on the input of the JBODs can by it self starve the read operations. These limitations should in theory limit the need for any supervised load balancing between read and write operations.

There are a few things in theory that could be done in order to improve the read and write performance. The stripe size could be reduced to 256 kbyte or even 128 kbyte. This should help if it is difficult to achieve a high number of concurrent disk operations, since it will distribute each operation over a larger number of disks. This could also be useful if it is difficult to achieve large read and write blocks. A smaller stripe size could however also limit the maximum capacity of a JBOD. Another improvement that could be done is to adjust the alignment of the data. This is actually done in the tests above. This means that a 1 Mbyte block would be written to just two disks, given a stripe size of 512 kbyte, instead of to three disks, which would be the most likely case.

The performance could, on a more global scale, be improved by simply scaling the system with more JBODs. An inevitable point would be reached when additional Fibre Channel host adapters would have to be added.

6 The preprocessing program

Micronic Laser Systems AB is currently developing a preprocessing program called CFRAC. This program is being designed for a shared memory Ultrasparc/Solaris system. It is not practical at this point to rewrite the program only to be able to analyze and verify a clustered x86/Linux system. So instead I have decided to design a benchmarking program that will simulate the approximate workload of CFRAC. In order to do so the current program has to be analyzed in terms of memory usage and time complexity. The current program can however not be used to model all features, first of all because all features are not yet implemented and secondly because of the completely different target platform.

The behavior of a preprocessing program is most of all dependent of the characteristics of the pattern data file being processed. This makes it just as important to analyze pattern data as analyzing the program itself. Since pattern data can have very different characteristics it is necessary to narrow the number of cases down. The most relevant case for this project is an estimated worst-case scenario.

6.1 Analyzing the preprocessing program

The preprocessing program (CFRAC) is used to convert laser pattern data from one vector format into the next, see Section 2.3 for more information. The process includes geometric transformations, merging several input files, fracturing figures into several figures of limited size and translating the data format.

The current structure of CFRAC, illustrated in Figure 6.1, is based on Pthreads and shared memory. One thread reads the input file and pushes the pattern data into a FIFO. In the mean time a second thread reads the pattern data from the FIFO, processes it and writes it to the output file. Even though the current system is limited by disk-I/O this method actually reduces execution time. The reason is that the whole program does not have to wait while reading from disk. The distribution in the structure is prepared for adding additional output threads while only having one input thread.

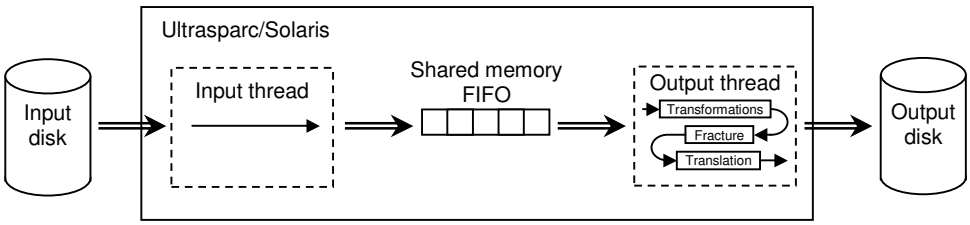

Figure 6.1 The current structure of the preprocessing program (CFRAC)

6.1.1 Memory usage

The amount of memory used by CFRAC is adjustable from command line. The amount of memory used does not have a tremendous impact on the performance of CFRAC, it does however play a major roll in the performance in the application reading the output file from CFRAC. The explanation for the high demands for memory is that most of the memory is used as buffers for the output file. The output file is divided into a number of independent FMBs and each FMB contains a number of buckets, see Section 2.3 for more information. A normal setup would be about 24 FMBs and 1500 buckets. Each bucket in each FMB has to have its own buffer memory. Hence the amount of memory for each bucket is often quite scarce. This affects the fragmentation within the output file. This only poses a problem while reading the file since disk caches and other later software caches are not properly utilized. This demands for large memory requirements, preferably in the area of several Gbyte. A preferred buffer memory size per bucket would be one of the larger read blocks mentioned in Section 5.7.

6.1.2 Time complexity

The time complexity of CFRAC is generally very low. Most of the processing is O(N) with a few exceptions. One of those exceptions is the conversion from general polygons to trapezoids and rectangles. Each polygon can have at most 254 sides. The corner coordinates for a polygon are sorted only ones using a standard O(N*log(N)) algorithm. The following fracturing is done in an O(N) fashion.

There is however a few circumstances that make CFRAC read the complete input file a multiple number of times instead of just once. The first and most common reason is the presence of more than one pattern layer in one input file. This will cause CFRAC to read the whole input file once for each layer. The reason is that all layers in the output file have to be in the correct order. This restriction is however not fulfilled in the input file. It should be noted that the combination of a large input file and many layers is rare. A large input file often consists of only one layer. The second reason is that the dependences between buckets are too space consuming. That is, if the following program, for a given bucket, has to keep too much data from previous buckets. This limitation is always checked during the whole execution time of CFRAC. If CFRAC recognizes that the dependences consume too much space it will restart itself and adjust the preset distribution parameters. This should however be very unlikely during real production and the preset distribution parameters are subject for adjustments during the development of the program.

6.1.3 Planned improvements

There are a number of planned improvements pending. The number of output threads is planed to be adjustable from command line. This means that there will be a single thread reading and distributing pattern data and an adjustable number of threads processing and writing the pattern data. Since there will be more than one thread writing to the output disk, it will no longer be efficient to use a single output file. At least one file per output thread is needed.

A software optimization called healing is also planned. The meaning of healing is basically to merge a subset of small connecting figures whenever it is possible. The most desired result from healing is however not to reduce the number of figures but change their shape. Long and thin figures, called slivers, are not desired and a way of reducing them without just dividing them is to do healing. One general way of doing this is to merge all connecting figures into a polygon and later split the polygon in a different direction using the already existing code. This would increase the time complexity of the program. Merge the figures by simply comparing them with each other would be at least O(N^2) and splitting would be the usual O(N*log(N)) sorting and O(N) fracturing. This means that healing cannot be done with a large number of figures. A reasonable number of figures would probably be 10-50. The amount of memory used for this feature would therefore be insignificant.

The ability to handle several pattern layers simultaneously would drastically reduce the execution time by only having to read the input file once. Redefining the output so that a file does not contain more than one layer would allow the following program to open the files in the correct order. This would however also mean that the number of memory buffer has to be multiplied by the number of layers.

A better memory management could increase the size of the memory buffers and decrease the fragmentation in the output file. Hence increasing the performance of the following program. One way of improving the memory management could be to use a pool of large memory buffers and in that way exploits the geometric locality of the pattern data. This in combination with asynchronous disk operations would increase disk-I/O performance.

Another way of decreasing the fragmentation of the output file is it to have several output files. One file per bucket and FMB would result in no fragmentation at all, this assumes a non-fragmenting file system such as the one described in Section 4.4.4. The drawback would be the logistic problems of handling tens of thousands of simultaneously growing files in a non-fragmenting file system. Using a fragmenting file system and one file per bucket and FMB might still be a good idea since software caches are utilized more efficiently compared to a single fragmented file.

6.2 Redesigning the preprocessing program

Since the preprocessing program (CFRAC) is currently only available for shared memory systems a new design has to be created, at least on paper. This newly designed preprocessing program is going to be modeled and used in the workload simulator. Some of the ideas presented in this section are my own and some are from other developers at Micronic Laser Systems AB.

The basic idea is to keep most of the current design. The input process will distribute the pattern data to the output processes. Swing buffers will allow for asynchronous message passing. This is illustrated in Figure 6.2. The data should be distributed using a combination of *round robin* and *bag of tasks*. The reason for not using a pure round robin is that the pattern data can expand differently in the different output processes. Adding a bag of tasks functionality will give an even distribution of the output, assuming that the execution time of each output process largely depends on how much data it writes to the output disk.

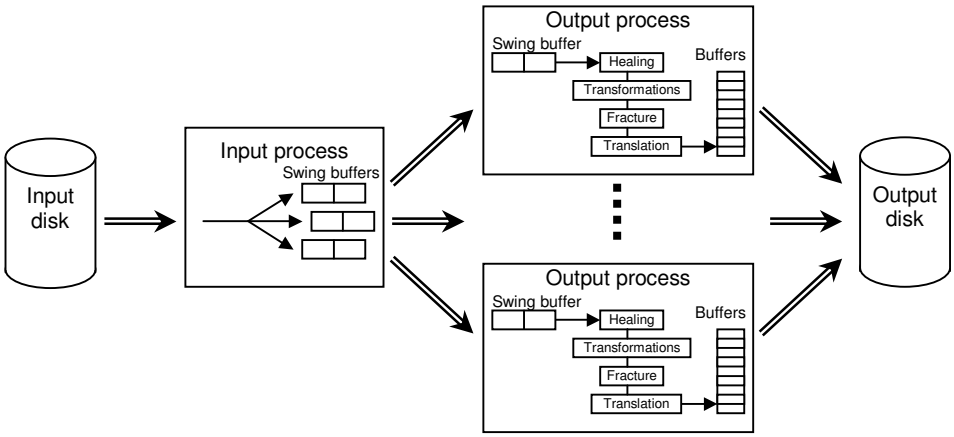

Figure 6.2 A possible structure of the preprocessing program for a computing cluster

The distribution to different FMBs is currently done in the output thread using round robin; this would have to be moved to the input process. Each output process would process a suitable number of FMBs; for example 6 FMBs per output processes in the case of 4 output processes and 24 FMBs in total. This would increase the amount of memory available per FMB compared to a design where all output processes handles a portion of all FMBs. A fundamental assumption for the design discussions is that the number of output processes is less or equal to the number of FMBs.

6.2.1 The input process

The new distribution of FMBs could be done using round robin. However it would have to be done on blocks of pattern data instead of on single pattern data records. This is necessary in order for the healing to work. Such a block would typically be the same number of records as number of records used in the healing process, i.e. 10-50 records. To increase the efficiency of message passing, several of these blocks would have to be added together in each swing buffer to a total of 1000 records or more. Some form of workload distribution could be achieved by filling only available swing buffers using round robin. The output processes would request pattern data seeing the input process as a bag of tasks, even though it is one bag per output process. If an output process is heavily loaded, the corresponding swing buffer will be skipped by the round robin once it has been filled.

Using the method bag of tasks brings some demands; all distribution blocks have to be independent or the input process has to remember the state of each output process. An independent distribution block means that it has to have complete hierarchical structures describing layers and all levels of repetitions. This means unnecessary overhead and an increase in size. A slightly more complex way is to keep track of which hierarchical level each process is in. This has no real disadvantages besides slightly more complex programming.

It might be necessary to use an extra thread in the input process: A dedicated thread for reading from the disk, if asynchronous disk operations are not available. In that case a shared memory FIFO would typically be used between the threads.

It might also be necessary to use a slightly more intelligent way of distributing pattern data. An extreme case is when a pattern data file is only 10 records long and all pattern data is in a geometrically large repetition block. The result would be that just one output process gets all the pattern data. The repetition would later be divided and the amount of data is dramatically increased. A possible way of solving this is to use a special case for very large repetitions. If the number of iterations in a repetition reaches a predetermined limit the records within the repetition block is distributed over the output processes using true round robin, see Figure 6.3. This method is excessive if there is a large enough amount of repetition blocks. It consumes both memory and time. Another drawback is, as mentioned earlier, that the healing process is crippled by round robin on individual records.

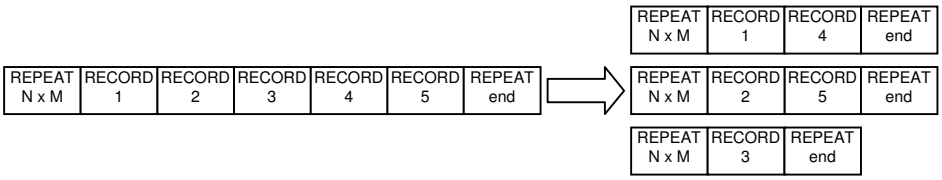

Figure 6.3 Splitting a repetition block using round robin

6.2.2 The output processes

Receiving pattern data from the input process is straightforward, using a swing buffer and asynchronous receive operations. If the CPU boards are equipped with two CPUs with shared memory it might be a good idea to split the process into two threads. The first thread receives data and does healing, the second thread does the rest of the processing and writes the pattern data to the disk. If the CPU boards are equipped with only one CPU per board it would be more efficient to scale the system by simply increasing the number of CPU boards and output processes.

The most important part is to get enough memory for each bucket. One possible way is to exploit the geometrical locality in the input pattern data file. Instead of allocating one memory buffer for each bucket in each FMB a pool of buffers could be used instead. This pool could consist of fewer buffers than the total number of buckets. If a buffer becomes full it will be disconnected from the bucket and an asynchronous write operation writes the data to disk. If all buffers are being used and a new one is needed a buffer could be flushed based on chronological order with the least recently used first. It might be necessary to flush buffers at an earlier stage in order to prevent a state when the process is waiting for free buffers. If asynchronous disk operations are unavailable, such operations could be constructed using threads.

An alternative to using fewer memory buffers than buckets is to use more buffers. For example twice as many buffers as buckets. Half of the buffers would be in a buffer pool. The rest of the buffers would be fixed to its corresponding bucket. When a buffer gets full a new buffer is taken from the pool and linked to the fixed one. When the new buffer gets full a second buffer would be linked on to the first one and so on. If no free buffers are available in the pool a fixed buffer with corresponding linked buffers would be flushed and the linked buffers returned to the pool. This method might give better results in the case of poor geometrical locality. However the available memory is used more statically and might in the normal cases lead to smaller blocks in the final output file.

As mentioned in Section 6.1.3 it will no longer be possible to use a single output file. A suitable solution could be to have one file per FMB and layer. This would in a normal case result in no more than 100 files in total.

6.3 Analyzing pattern data

Analyzing typical pattern data is just as important as analyzing CFRAC itself in order to model the workload. The interesting parameters are the distribution between different record types, hierarchical structures and number of bytes per record.

A major semiconductor manufacturer has provided a number of typical pattern data files. Each of these pattern data files describes a circuit layer within an integrated circuit. Table 6.1 and Table 6.2 shows two circuit layers form the same integrated circuit. These files are, except for the size, quite typical as input to CFRAC. A worst-case scenario is predicted to be 1200 times as dense according to studies done by Anders Thurén at Micronic Laser Systems AB for the 65nm node.

The parameters shown are the number of figure records. This is the number of records in the file and is not the same as the number of features on the finished plate. The number of bytes per feature is taken from the file format specification [38]. The total summary of all bytes for all records is calculated without any file headers or such.

Table 6.1 Profile of file: 1_input

Record type	Records	Bytes/record /(byte)	Bytes /(byte)
Rectangles	44 392 193	14	621 490 702
Trapezoids	832	20	16 640
XY-Repeated Rectangles	2 693 863	26	70 040 438
X-Repeated Rectangles	15 724 795	20	314 495 900
Total	62 811 683		1 006 043 680

Table 6.2 Profile of file: 2_input

Record type	Records	Bytes/record /(byte)	Bytes /(byte)
Rectangles	40 450 364	14	566 305 096
Trapezoids	754	20	15 080
XY-Repeated Rectangles	2 800 516	26	72 813 416
X-Repeated Rectangles	13 920 397	20	278 407 940
Total	57 172 031		917 541 532

Worth noticing is that there are only single repeated objects and no large repeated structures. The lack of large repeated structures makes this type of pattern files larger in size but also more predictable. A highly hierarchical pattern file, though harder to predict, would be much smaller and is therefore considered an easier case to handle. These highly hierarchical cases are for that reason not further analyzed in this thesis.

I have rewritten an existing test program in order to extract comparable information from the output files. Table 6.3 and Table 6.4 present the output files from the corresponding files in Table 6.1 and Table 6.2. The parameters in these tables have been altered in order to be more easily comparable with the input files. For instance the output format does not include a specific record for repeated rectangles; instead a repeat record is appended to a rectangle record. The number of bytes per record is dynamic in the format and the average number is therefore presented instead.

Table 6.3 Profile of file: 1_output

Record type	Records	Bytes/record /(byte)	Bytes /(byte)
Rectangles	19 445 145	11,60	225 544 334
Trapezoids	1 404	15,87	22 283
XY-Repeated Rectangles	43 368 928	23,89	1 036 030 503
Total	62 815 477		1 261 597 120

Table 6.4 Profile of file: 2_output

Record type	Records	Bytes/record /(byte)	Bytes /(byte)
Rectangles	26 744 184	11,71	313 220 207
Trapezoids	1 376	15,87	21 835
XY-Repeated Rectangles	30 430 302	23,70	721 243 704
Total	57 175 862		1 034 485 747

A noticeable effect is that the number of repeated records increase and the number of single records decrease. This can be explained by the fact that one of CFRAC´s main tasks is to limit the maximum geometrical size of a feature. This is for instance done by cutting up a large rectangle into an array of smaller rectangles. An already repeated rectangle would in the same way result in an array of an array of a rectangle. This is not shown specifically in the simplified presentations in Table 6.3 and Table 6.4, but included in the number of repeated rectangles.

The number of rectangle records in the file is however not changed noticeably. Thus not changing the file size in any drastic ways. This also verifies the equal bandwidths in and out of CFRAC specified in Section 3.1. It should in fact, according to studies done by Anders Thurén at Micronic Laser Systems AB, be possible to decrease the output file size with approximately 20 % relative the input file size. This is done by aggregating groups of rectangles/trapezoids and thereby being able to describe them with fewer bytes.

A number of assumptions were done when analyzing these files:

- All figure types are evenly distributed over the file both in geometry and address space.
- It is considered that some noticeable locality occurs in an input file. Some relation between geometrical position and address is inherited from the CAD program, or other preprocessing stages.
- The only difference from larger files is the density. A more dense design would probably mean smaller features, thus reducing the need to split large features. This effect is however omitted in this study.

6.3.1 Conclusions

The purpose is to design a model to simulate a near worst-case scenario, meaning the scenario that takes the longest time to process. Based on previous knowledge this assumes that the largest input and output files correspond to the longest processing time. Large files means typically files without high levels of hierarchical repetitions.

6.4 Summary

The current preprocessing program (CFRAC) is exclusively designed for a specific type of SMP machine. A new structure designed for clustered computation has been presented. The new design distributes the pattern data to the processing nodes using a combination of round robin and bag of tasks. A new feature called healing has also been introduced.

After having analyzed two pattern data files provided by a major semiconductor manufacturer some conclusions could be made. First of all, a typical file does not include general polygons, but rather a collection of repeated and unrepeated rectangles and trapezoids. No major hierarchical repeated structures occur. A likely worst-case scenario would typically be a very large file with only a large number of unrepeated rectangles and a small number of trapezoids.

7 The workload simulator

There are two main reasons for making this workload simulator. The first reason is that it can be used to benchmark and compare different types of clustered computer systems. The second reason is that it can be used to try out new techniques. There are also a number of other benefits of having an easily ported workload simulator of a program. First it buys time if the real program was not thought to be portable. Secondly the code does not need to be protected in the same sense as the real program. It might be possible to mail the source code to other companies that are offering new hardware solutions and in this way speed up the evaluation process.

The amount of work invested in such a workload simulator depends largely on what the goal is. If the simulator is made too complex the benefits of using it disappears, an excessively complex simulator takes much too long to write and the time is better spent rewriting the real program. On the other hand an over simplified simulator is not more useful than any other already existing benchmarking program. In this case the goal is to reproduce approximately the same load on the CPUs, memory and disks. Since it only needs to present the relative difference between different platforms it does not need to be a perfect match.

The plan is to implement the simulator by using MPI since the primary task is to verify a clustered system. At a later stage the simulator could also be used to try out different program architectures.

7.1 Modeling the real preprocessing program

The model of the program used in the workload simulator could roughly be illustrated as everything that is shown in Figure 6.2. The distribution of pattern data from the reading process and to the writing processes is done with a combination of round robin and bag of tasks using MPI.

The writing process initiates the transmission of a pattern data block by sending a pattern data request and then setting up an asynchronous receive operation. The pattern data request is an empty message but with a predefined *request* tag. If necessary this message could in the real program include some status information; allowing the reading process to do some adjustments in load balancing. When the reading process has the corresponding swing buffer ready and receives a request message it will initiate an asynchronous transmission marked with a predefined *pattern data* tag. If all data in the input file has been read the reading process will answer each pattern data request message with an empty message marked with a predefined *finished* tag.

A slightly purified version of the implemented code in the reading process can be seen in Figure 7.1. Error handling and other special cases are removed to enhance clarity.

```
1  while (finishedProcesses < numberOfWritingProcesses) {
2      process = (process + 1) % numberOfWritingProcesses;
3      if (buffCount[process] == mpiSendBlockSizeInBytes ||  ftell (fd) == inputFileLength) {
4          MPI_Iprobe (process+1, REQUEST_TAG, MPI_COMM_WORLD, &flag, &status);
5          if (flag == TRUE) {
6              MPI_Recv (NULL, 0, MPI_CHAR, process+1, REQUEST_TAG,
                       MPI_COMM_WORLD, &status);
7              if (buffCount[process] > 0) {
8                  SWING_BUFFERS (&readBuffer[process], &sendBuffer[process]);
9                  MPI_Isend (sendBuffer[process], buffCount[process], MPI_CHAR,
                           process+1, DATA_TAG, MPI_COMM_WORLD,
                           &request[process]);
```

```
10                  buffCount[process] = 0;
11              }
12              else {
13                  finishedProcesses++;
14                  MPI_Isend (NULL, 0, MPI_CHAR, process+1, FINISHED_TAG,
                                MPI_COMM_WORLD, &request[process]);
15              }
16          }
17      }
18      else {
19          fread (&readBuffer[process][buffCount[process]],
                    averageRecordSize, healingRecords, fd);
20      }
21  }
```

Figure 7.1 The distribution using a combination of round robin and bag of tasks.

Line 2 cycles which buffer/process to currently access, hence leading to a round robin distribution. Line 19 is only executed if the current buffer is not full, hence limiting the round robin to only distribute the data to available buffers. Lines 4 and 5 checks if a request has been posted from the current writing process. If so, the request is received. If the end of the input file is reached and there is no data in the swing buffer the reading process sends an empty message with a finished tag, see line 14. If data is available the swing buffer is swung and an asynchronous send command is started. If no request has been posted, the algorithm skips to the next buffer/process, hence behaving from the outside as a bag of tasks where each writing process can request pattern data in its own pace.

Figure 7.2 shows an equally purified version of the requesting code from a writing process. This code is executed every time the writing process runs out of pattern data. Similarly to Figure 7.1 error handling and other special cases are removed to enhance clarity. The result after having executed this code is that *bytes* number of bytes of pattern data is available in the *processBuffer* buffer and a new asynchronous receive command is set up.

```
22  If (firstTime == TRUE) {
23      firstTime = FALSE;
24      MPI_Isend (NULL, 0, MPI_CHAR, 0, REQUEST_TAG, MPI_COMM_WORLD,
                    &sendRequest);
25      MPI_Irecv (receiveBuffer, mpiSendBlockSizeInBytes, MPI_CHAR, 0,
                    MPI_ANY_TAG, MPI_COMM_WORLD, &receiveRequest);
26  }
27  MPI_Wait (&receiveRequest, &status);
28  switch (status.MPI_TAG) {
29      case DATA_TAG:
30          SWING_BUFFERS (&receiveBuffer, &processBuffer);
31          MPI_Get_count (&status, MPI_CHAR,  &bytes);
32          MPI_Isend (NULL, 0, MPI_CHAR, 0, REQUEST_TAG, MPI_COMM_WORLD,
                        &sendRequest);
33          MPI_Irecv (receiveBuffer, a_param->mpiSendBlockSizeInBytes, MPI_CHAR, 0,
                        MPI_ANY_TAG, MPI_COMM_WORLD, &receiveRequest);
```

```
34              break;
35          case FINISHED_TAG:
36              bytes = 0;
37              break;
38  }
```

Figure 7.2 Requesting data in the writing process.

The first time the code is executed a request is sent and an asynchronous receive command is set up, see lines 22-26. Line 27 waits for the previous receive command to finish. Line 28 determines the tag of the received message. This allows for separate handling and is currently used only to distinguish between data and when the processing is finished. Line 30 swings the swing buffer and lines 32 and 33 sets up a new asynchronous request. Line 31 is needed in order to determine how much data is actually received.

The model of the pattern data used is designed to be a near worst-case scenario in terms of execution time. This has been simplified to be a single input file that is read only one time. The file is considered to be completely flat, meaning no repetitions of any kind. The pattern data only consists of rectangles and trapezoids. More complex features like polygons are not included.

The reading process creates the temporary source file needed to read from in an initializing stage. The writing processes are at this stage waiting at a synchronization barrier. All performance measuring is reset at the time that the reading processes arrive at the barrier and the processes can continue.

Modeling the CPU workload should be considered experimental. Even if the number of lines of C-code is the same and the type of instructions is the same, it is still difficult to guarantee that the compiler would not produce different results due to variable dependencies etc. In some cases, such as healing, the real code does not even exist. The main reason for simulating the CPU workload is to get some results other than just I/O. There is for example no need for asynchronous disk operations unless the CPU could work with something else in the mean time. Modeling the CPU workload in the workload simulator is low-tech, simply counting lines of relevant C-code in the current preprocessing program (CFRAC). This code is then replicated with dummy operations in the workload simulator.

One of the most difficult parts to model is the healing algorithm. Since the algorithm is not yet finalized and far from being written in code there is practically nothing to analyze. The only information to build the model from is that which is presented in Section 6.1.3. The model is implemented using a more artistic approach. The basic structure in the design is as a sequence of a double loop, an O(N*log(N)) sorting algorithm followed by a single loop.

The bucket buffers are slightly simplified compared to what the real preprocessing program would look like. The number of bucket buffers is freely adjustable and so is the size of the bucket buffers. The pattern data is distributed over the bucket buffers at random on a single record level. This means that the geometrical locality in the input files is considered to be absolute. This assumption should be acceptable if the number of bucket buffers is not too low.

Asynchronous disk operations are due to lack of time currently not implemented in the workload simulator. This should be considered high priority if the intention is to use the CPU workload simulations. However for testing I/O performance with the CPU workload simulation disabled the benefits of asynchronous disk operations are limited.

7.1.1 Parameters

An example of a parameter file for the workload simulator is shown in Figure 7.3. Note that the parameters in this example are not to be considered to have relevant values.

```
#Parameters for PrepSim
inputFilename              /wap/bjolun/prepsim/prepsim_src.tmp
outputFilename             /wapbig/tmp/prepsim_tgt.tmp
enableReading              TRUE
enableWriting              TRUE
enableProcessing           TRUE
numberOfRectangles         8192000
numberOfTrapezoids         1024
rectangleSizeInBytes       14
trapezoidSizeInBytes       20
byteExpansionFactor        1.1
readBlockSizeInBytes       40960
writeBlockSizeInBytes      1024000
healingRecords             20
bucketBuffers              100
bucketBufferSize           1024000
mpiSendBlockSizeInBytes    40960
end
```

Figure 7.3 A parameter file for the workload simulator.

The first parameters *inputFilename* and *outputFilename* enables the user to specify where the temporary files are stored and how they should be named. The output files will be named according to the *outputFilename* parameter and an appended number corresponding to the process number.

The second set of parameters *enableReading*, *enableWriting* and *enableProcessing* allow the user to analyze the impact of different parts of the program. If *enableReading* is set to FALSE all other operations except reading from disk will occur. The parameter *enableProcessing* enables or disables the simulated CPU workload. This means that if the parameter is set to FALSE the data will still be read and written to disk as well as sent via message passing and buffered in bucket buffers.

The parameters *numberOfRectangles*, *numberOfTrapezoids*, *rectangleSizeInBytes* and *trapezoidSizeInBytes* are used to calculate the size of the input file. The size of the output file is calculated using the size of the average input record multiplied by the parameter *byteExpansionFactor* and the total number of input records.

Disk access is regulated independently from other parameters. The parameters *readBlockSizeInBytes* and *writeBlockSizeInBytes* are used when creating and reading the input file and writing the output files. Note that there is no relation between bucket buffer memory usage and the *writeBlockSizeInBytes* parameter.

The parameter *healingRecords* specifies how many records the healing algorithm should work on. This also affects the size of each pattern data block that the input process handles as it does the round robin distribution among the output processes.

BucketBuffers specifies how many bucket buffers should be used. This is defined as the number of bucket buffers per output process and not the total number. The related parameter *bucketBufferSize* adjusts the amount of memory that is allocated for each bucket buffer in each output process. This means that the total amount of memory consumed by bucket buffers is the number of output processes times *bucketBuffers* times *bucketBufferSize*

MpiSendBlockSizeInBytes specifies the size of the pattern data messages sent from the reading process to each of the writing processes. This is not dependent on the number of healing records. However the number of healing records could be limited by *mpiSendBlockSizeInBytes*.

7.2 System and programming notes

The workload program is written in C and developed on an Ultrasparc/Solaris 5.7 system using GCC 2.95.2 and MPICH 1.2.5.2. This might not be the fastest combination and GCC is actually not the compiler currently used to compile CFRAC. The most important reason for using the combination of GCC and MPICH is availability, both in terms of compatibility and licenses.

In order to control the read and write block size a disk I/O package called FIO is used from CFRAC. This is a thin abstraction layer that gives an adjustable disk cache. The cache has only one cell, meaning that it can only cache a single area of the file at a time.

7.3 Summary

The preprocessing workload simulator includes:

- Adjustable reading and writing blocks for disk access.
- Moving pattern data using asynchronous message passing.
- Requesting pattern data using bag of tasks.
- Round robin distribution on available swing buffers.
- Dummy loops as a model to simulate healing. (Experimental)
- Dummy loops as a model to simulate fracturing, transformations and translations. (Experimental)
- Adjustable pattern data expansion.

The preprocessing workload simulator does not include:

- Modeling of individual pattern data records.
- Splitting of polygons.
- Variations in pattern density.
- Hierarchical repetition structures.

8 Testing the workload simulator

No real tests have been possible due to the unfortunate lack of a closely clustered system. The only system available at the time was an Ultrasparc/Solaris SMP machine. Thanks to the versatility of MPICH it is still possible to test the workload simulator. However the tests should be considered more to be functionality tests and a guide for future tests rather than a performance evaluation.

8.1 Test setup

The Ultrasparc/Solaris system used for the tests is a Sun Fire V880 which is a SMP machine with four Ultrasparc III 750 MHz CPUs and 8 Gbyte of shared memory (150 MHz and 8-way interleaved). This means that the use of message passing is far from ideal. The benefits and implementation of asynchronous message passing in MPICH on a SMP machine without any supporting hardware could be questioned. The data is read from and written to the same internal RAID-0 device with just two physical disks. This is not the ideal way to do it; it would be more beneficial to use more disk devices in order to spread the workload. The fact that the RAID-0 device is 95% full (8 Gbyte free) and most likely heavily fragmented will not improve the performance either.

The test batch consists of three tests. The first test is done without any disk access or simulated processing, see column (a) in Table 8.1. This is done in order to evaluate the maximum capacity of the message passing. The second test is done with disk access but without simulated processing, see column (b) in Table 8.1. This will give a reference to how much capacity the disks can deliver, provided that the message passing is not the limiting factor. The final test is the complete full test with disk access and simulated processing, see column (c) in Table 8.1.

Table 8.1 The three parameter files used in the workload simulator test

Parameters	(a)	(b)	(c)
inputFilename	prepsim_src.tmp	prepsim_src.tmp	prepsim_src.tmp
outputFilename	prepsim_tgt.tmp	prepsim_tgt.tmp	prepsim_tgt.tmp
enableReading	FALSE	TRUE	TRUE
enableWriting	FALSE	TRUE	TRUE
enableProcessing	FALSE	FALSE	TRUE
numberOfRectangles	81920000	81920000	81920000
numberOfTrapezoids	1024	1024	1024
rectangleSizeInBytes	14	14	14
trapezoidSizeInBytes	20	20	20
byteExpansionFactor	1.1	1.1	1.1
readBlockSizeInBytes	40960	40960	40960
writeBlockSizeInBytes	1000000	1000000	1000000
healingRecords	20	20	20
bucketBuffers	100	100	100
bucketBufferSize	1024000	1024000	1024000
mpiSendBlockSizeInBytes	524288	524288	524288

All four CPUs are utilized by using one reading process and three writing processes. This gives a total memory consumption of approximately 300 Mbyte for all writing processes combined. The parameters in Table 8.1 give an input file size of 1093.77 Mbyte and a total output of 1171,9 Mbyte distributed over the writing processes.

The parameters that are possible to measure with the workload simulator are the throughput of each process measured at the point where it is read from/written to disk. The size of each output file is presented and shows in some sense the result of the workload distribution. The execution time is presented for the complete program since the program does not quit until all processes are done. The creation of the temporary source file is however not included in the presented execution time. This stage is measured separately and pre-

sented as a throughput. This result can be useful as a reference of a single uninterrupted series of disk operations.

8.2 Tests

All of the subtests are done twice to increase the chance of detecting non-typical results. The first run within a test is presented without parentheses in the tables below and the second run is presented within parentheses. At the time for the tests there were no other major programs running on the system. So the results achieved should be considered to be without interference.

Table 8.2 shows the results from the first subtest. No temporary source file is created since it is not used. This test is mainly aimed to test the capacity of the message passing between the processes.

Table 8.2 The results from the first subtest, testing mainly message passing

Process	Throughput /(Mbyte/s)	Execution time /(s)	File size /(Mbyte)
Creating file	-	-	-
Reading process	68.61 (68.58)	15.94 (15.95)	1093.77 (1093.77)
Writing process 1	24.66 (24.63)	-	392.98 (392.84)
Writing process 2	24.60 (24.49)	-	392.18 (390.68)
Writing process 3	24.27 (24.34)	-	386.74 (388.38)

Table 8.3 shows the results from the second subtest. A temporary source file is created and the writing processes actually write data to the disk. The only thing missing in this test is the simulated CPU workload.

Table 8.3 The results from the second subtest, testing mainly disk capacity

Process	Throughput /(Mbyte/s)	Execution time /(s)	File size /(Mbyte)
Creating file	40.96 (41.73)	-	1093.77 (1093.77)
Reading process	15.82 (17.15)	69.13 (63.77)	1093.77 (1093.77)
Writing process 1	5.35 (6.17)	-	390.53 (395.19)
Writing process 2	5.32 (5.61)	-	390.83 (388.04)
Writing process 3	5.30 (5.60)	-	390.53 (388.67)

Table 8.4 shows the results from the final subtest. Everything in the workload simulator is enabled in this test.

Table 8.4 The results from the final subtest, testing the complete workload simulator

Process	Throughput /(Mbyte/s)	Execution time /(s)	File size /(Mbyte)
Creating file	41.68 (41.16)	-	1093.77 (1093.77)
Reading process	10.89 (10.77)	100.41 (101.59)	1093.77 (1093.77)
Writing process 1	3.95 (3.76)	-	392.14 (391.07)
Writing process 2	3.79 (3.70)	-	390.53 (390.53)
Writing process 3	3.81 (3.72)	-	389.22 (390.29)

8.3 Analysis

The absolute performance achieved in these tests is not that important but the relative difference between the tests is more interesting. It is apparent from Table 8.2 that message passing on a SMP machine is not the best solution. A comparison with the results in Table 8.3 shows that the program is, on this system, more limited by disk than message passing.

As could be seen in Table 8.4 the time has increased by approximately 50% compared to the case in Table 8.3. This might be due to the fact that there are no asynchronous disk operations used in the workload simulator. Hence all processing time is being added directly to the total execution time. My guess is that much of the processing time is spent in the healing function. For each 20 figures the pattern data is processed in a loop 20 x 20 =

400 times. This can, if needed, later be verified by using a profiling tool or by simply decreasing the parameter healingRecords to 1.

The distribution among the output files is pleasantly even. However nothing else would be expected since there is nothing uneven in the distribution process.

8.4 Conclusions

The test results are not that important for this project. The most important conclusion is that the workload simulator worked and behaved as expected. These tests are more important as a demonstration on how a possible test scenario would be conducted. Other combinations could be interesting to study. For instance a series of tests with different sized MPI messages.

The program does not really test the ability of the distribution algorithm since the pattern data is treated evenly. One way of testing its functionality is to use different byte expansion factors in each writing process. This would mean that some processes would request data more often than others. The size of all output files should however be somewhat similar or at least more equally distributed than if a pure round robin distribution had been used.

9 Conclusions

Due to the unfortunate lack of hardware this thesis has been reshaped as more of a toolbox for future evaluations. It describes different types of common off the shelf products and techniques, as well as providing a benchmarking tool useful for evaluating a clustered preprocessing system.

A performance test using the workload simulator is documented in Section 8. The specific test results are not that relevant for this project however it is more relevant as an example of how a performance evaluation of a preprocessing system could be done.

The main focus of this thesis is the proposed idea of a Linux/x86 cluster or NUMA. The reason that I have dismissed the idea of a NUMA architecture is that it is a more expensive and an overkill in the sense that it is more dynamic than necessary. The preprocessing program does not need to be fine grained and the pattern data can be processed in a pipelined style. The idea of building a Linux/x86 cluster gives better possibilities to specifically adapt the hardware for the exact needs. It is my recommendation that all the nodes in the cluster should be specialized and not identical. It has not been possible to verify how much hardware is needed for the idea to work. The general conclusion is however that the idea seems feasible.

A real low cost solution for a computer cluster would be to use reasonably low cost servers with IA-32 Xeon CPUs and Gigabit Ethernet as interconnect. I suspect that it will be difficult to sustain an average of 90 Mbyte/s from the input node with a single Gigabit Ethernet connection so my recommendation is to use a host adapter with two or four load balanced ports and an effective switch. The processing nodes will be equipped with single port host adapters. This is my main recommendation for this application. This solution is yet to be verified, it is only based on the requirements presented in Section 3 and the information that the current preprocessing program is mainly limited by disk-I/O and not CPU. The required number of processing nodes has not been decided yet and is subject for evaluation, preferably using the workload simulator as a benchmarking tool. My estimation is that it is only a matter of a handful of nodes. A more scalable system can if necessary be achieved by using IA-64 Itanium 2 CPUs and InfiniBand or 10 Gigabit Ethernet as interconnect. This solution would be more expensive but would also be able to handle more RAM. If even more performance is needed the whole preprocessing concept has to be redesigned in some way to allow for more than one input node.

The only hardware that has been tested and verified is the JBOD disk solution presented by VMETRO, see Section 4.4.4. It is perhaps the most inexpensive solution and it has proved to deliver more performance than foreseen. In the tests in Section 5 we showed that a single JBOD could deliver 220 Mbyte/s to the real-time processing unit and that it is possible to write 50 Mbyte/s to the JBOD. This concludes that the requirements of 90 Mbyte/s writing and 360 Mbyte/s reading can easily be satisfied with three JBODs or perhaps even with two. This solution also solves a problem that is not mentioned in this project and that is how to extract the pattern data from the output disk and into the following real-time processing unit. However the solution is very low level and does not even support a fragmenting file system. The JBOD is also very sensitive to the size of the disk access blocks and how many disk operations that are executed asynchronously in parallel. This is so vital that it is an absolute necessity in order for the whole concept to work.

Another interesting alternative is to use at most 3-4 DELL|EMC CX200 disk arrays in a similar fashion as the JBODs in solution presented by VMETRO. There are two main reasons for choosing a number of smaller disk arrays instead of a single large one. The first is that it is easier and more predictable to scale. The second reason is that it might in fact be cheaper to buy a number of small disk arrays. This is however not based on any real price information but rather the common notion that price for computer hardware often increase almost exponentially relative performance. This solution has not been verified and is only based on the information given by DELL|EMC that a CX200 can deliver 90 Mbyte/s while simultaneously receive 90 Mbyte/s.

The choice of Linux distribution is mostly dependent on the needed level of support and not as much an issue of performance. The new Linux 2.6 kernel might however have a significant effect on the performance. The new kernel includes some important improvements concerning HPC applications. The choice of compiler can be based on the same type of questions as when choosing the Linux distribution. For instance, a compiler and development environment from Borland gives better possibilities for support than the GNU Compiler Collection.

My suggestion for communication between the processes is the MPI standard. It is a reasonably high level API that allows for code to be developed on a SMP or single CPU machine and later be compiled and executed on a clustered machine. MPI also support asynchronous message passing which is important to hide I/O time. Using larger messages might reduce the inevitable effect of the overhead introduced by MPI.

10 Future work

The remaining work is mainly to continue to test and evaluate parts of the preprocessing system. Building one of the suggested systems as complete as possible and evaluate it using the workload simulator as a benchmarking tool.

A decision has to be made of which Linux distribution should be used and more importantly which kernel. A more thorough evaluation of the available compilers and development environments should be done. This also includes development tools for profiling MPI applications.

An important factor for the price of the preprocessing system is the type of CPU used. A more thorough evaluation of the IA-32 and IA-64 CPUs would tell if the performance reached with IA-32 is enough.

Some representative performance tests have to be done on the other proposed disk solutions. Preferably the performance achievable by a single DELL|EMC CX600 and verifying how many CX200s would be needed to ensure enough throughput capacity.

The exact prices for the different common off the shelf products are still missing. For instance the total cost of an InfiniBand switch, fibers and host adapters. A closer calculation of increased development costs that the VMETRO disk solution would bring and compare it to the price of a solution based on standard disk arrays from DELL|EMC.

Some useful improvements concerning the workload simulator could also be done. My general recommendation is to keep the workload simulator as simple as possible and perhaps primarily use it to test I/O performance. Some new functionality related to I/O performance can preferably be implemented and tested using the workload simulator:

- It would be useful to be able to split each writing process into two threads. Note that the current implementation using MPICH is not thread safe.
- It should be investigated as well if the reading process could benefit from being divided into two threads.
- The whole program would perhaps benefit from asynchronous disk operations. Either if the JBOD from VMETRO is used or using functions such as aiowrite(), aioread() and aiowait() in Solaris or corresponding functions in Linux. If such functions are not natively available they should be written using threads.
- If the decision is made to go for a NUMA the workload simulator should be converted into using Pthreads and communicate through shared memory instead. It might in fact be useful to be able to choose between MPI and Pthreads. It is not necessary to be able to do this choice at execution time but rather as an option while compiling. The swing buffers would in that case be replaced by shared memory FIFOs.

11 References

[1] Micronic Laser Systems AB, *Micronic Laser Systems AB*, September 2003. http://www.micronic.se/index.html

[2] T Sandstrom, P Askebjer, J Sallander, R Zerne, A Karawajczyk, *Pattern Generation with SLM Imaging*, Micronic Laser Systems AB, Given at the BACUS symposium 2001

[3] Micronic Laser Systems AB, Marketing department.

[4] Fraunhofer-Gesellschaft, *IMS Fraunhofer - Institut für mikroelektronische Schaltungen und Systeme*, October 2003 http://www.ims.fhg.de/index.htm

[5] Micronic Laser Systems AB, *Annual Report 2002*

[6] Björn Lundberg, *Discussion with Rigmor Remahl and Anders Thurén about the history behind Micronic Laser Systems AB*, October 2003

[7] Minec Systems, *Minec - A Datalogic group company*, October 2003 http://www.minec.com/index.html

[8] SEMI, *SEMI S2-0200E Environmental, health, and safety guideline for semiconductor manufacturing equipment*, 2001 SEMI

[9] SEMI, *SEMI S8-0999 Safety guidelines for ergonomics engineering of semiconductor manufacturing equipment*, 1999 SEMI

[10] Red Hat Inc, *Linux, Embedded Linux and Open Source Solutions*, February 2004 http://www.redhat.com/index.html

[11] SUSE LINUX AG, *Welcome to SUSE LINUX*, February 2004 http://www.suse.de/en/index.html

[12] MandrakeSoft Inc, *MandrakeSoft home*, February 2004 http://www.mandrakesoft.com

[13] J Pranevich, *The Wonderful World of Linux 2.6*, February 2004. http://www.kniggit.net/wwol26.html

[14] Pallas, *High Performance Products - Vampir / Vampirtrace*, February 2004 http://www.pallas.com/e/products/vampir/index.htm

[15] Free Software Foundation, *GCC Home Page*, February 2004 http://gcc.gnu.org/index.html

[16] Borland Software Corporation, *C++ Builder X*, February 2004 http://www.borland.com/cbuilderx/

[17] Intel Corporation, *Intel C++ Compiler for Linux*, February 2004 http://www.intel.com/software/products/compilers/clin/

[18] Dell Inc, *Client & Enterprise Solutions, Software, Peripherals, Services*, February 2004 http://www.dell.com

[19] NEXCOM International Co. LTD, *NEXCOM*, February 2004 http://www.nexcom.com/index.html

[20] HP, *Integrity server family overview*, February 2004 http://www.hp.com/products1/servers/integrity/index.html

[21] InfiniBand® Trade Association, *InfiniBand® Trade Association: Home*, September 2003. http://www.infinibandta.org

[22] Mellanox Technologies, *InfiniBand Architecture – Mellanox Technologies, September 2003* http://www.mellanox.com/products/hpc.html

[23] K Voruganti, and P Sarkar, *An analysis of Three Gigabit Networking Protocols for Storage Area Networks*, 2001 IEEE.

[24] Myricom Inc, *Myricom Home Page*, October 2003
http://www.myricom.com/index.html

[25] Quadrics Ltd, *Quadrics – Homepage*, October 2003
http://www.quadrics.com

[26] Petrini, Feng, Hoisie, Coll and Frachtenberg, *The Quadrics Network (QsNet): High-Performance Cluster Technology*, 2001 IEEE

[27] Argonne National Laboratory and Mississippi State University, *MPI - The Message Passing Interface Standard*, October 2003
http://www-unix.mcs.anl.gov/mpi/index.html

[28] D Gustavson, *SCIzzL Overview*, October 2003
http://www.scizzl.com/index.html

[29] Dolphin Interconnect Solutions Inc, *Dolphin Interconnect Solutions Inc*, October 2003
http://www.dolphinics.com/index.html

[30] 10 Gigabit Ethernet Alliance, *10 Gigabit Ethernet Alliance*, October 2003
http://www.10gea.org/index.htm

[31] Rapid IO, *RapidIO: Home*, January 2004
http://www.rapidio.org

[32] Mercury Computer Systems, Inc., *Mercury Computer Systems-RACE++-Technology corner*, January 2004
http://www.mc.com/technology_corner/race_plus_plus.cfm

[33] Advanced Computer & Network Corporation, *AC&NC - RAID.edu - RAID Tutorial and Benchmarks Collection*, October 2003
http://www.acnc.com/04_00.html

[34] Fibre Channel Industry Association, *Fibre Channel Industry Association (FCIA)*, March 2004
http://www.fibrechannel.org

[35] EMC Corporation, *EMC^2 where information lives*, October 2003
http://www.emc.com

[36] VMETRO, *VMETRO*, October 2003
http://www.vmetro.com

[37] Edward K. Lee, Chandramohan A. Thekkath, Chris Whitaker, Jim Hogg, *A Comparison of Two Distributed Disk Systems*, SRC Research Report 155, 1998 digital Systems Research Center

[38] A Thurén, F Ihrén, *MIC pattern file format 1.8 rev 3,* Micronic Internal, August 2003

12 Abbreviations

API	Application Programming Interface.
CFRAC	The name of the current preprocessing program. Generally described in Section 2.3 and more thoroughly analyzed in Section 6.1.
CP-MDR	Custom Programmable – MIDAS Data Recorder. A product from VMETRO described in Section 4.4.4.
FC	Fibre Channel. A high-speed link primarily used for building SANs. Uses a serialized version of the SCSI protocol.
FIFO	First In First Out.
FMB	File Memory Buffer. Used for making parallel processing possible, further defined in Section 2.3.
HPC	High performance computer.
IA-32	Intel Architecture-32 (used in the Pentium CPUs).
IA-64	Intel Architecture-64 (used in Intel's latest 64 bit CPUs).
JBOD	Just a Bunch Of Disks. A collection of disks connected together, typically in a Fibre Channel loop or a SCSI chain.
MPI	Message Passing Interface. A library specification for message passing between computer nodes.
MPICH	An implemented version of MPI by Argonne National Laboratory and Mississippi University.
MPP	Massively Parallel Processing (each CPU uses its own individual memory).
NUMA	Non-Uniform Memory Access (easily described as a MPP behaving like a SMP, except that it is often much more closely connected).
PCI	Peripheral Component Interconnect. An I/O bus standard.
PCI-X	PCI – eXtended. An enhanced version of the PCI bus.
PMC	PCI Mezzanine Card.
RAID	Redundant Array of Independent Disks. Described in Section 4.4.1.
SAN	Storage Area Network.
S-FPDP	Serial – Front Panel Data Port.
SMP	Symmetric Multiprocessing (with shared resources like memory).
SNMP	Simple Network Management Protocol.
u	Unit. A unit specified for measuring the height of a rack-mounted component. (1u = 1.75")
VME	VersaModule Eurocard bus. An I/O bus standard.
x86	Refers to Intel's series of 16 bit and 32 bit CPUs, from 8086 to Pentium 4. In this report it is used as a collective name for all the x86 binary compatible CPUs including the new 64 bit versions.