Experiencing the Message Passing Interface standard and Parallel Image Processing

E.M.P.A. Mahieu

Catholic University Nijmegen
Faculty of Mathematics and Informatics
Section Experimental Informatics for Technical Applications
Mentor: dr. Th. E. Schouten
Thesis number: 364
18th January 1996

Experiencing the Message Passing Interface standard and Parallel Image Processing

Doctoraalscriptie: 364
ter aflegging van het doctoraalexamen, in de Technische Gerichte Informatica aan de Katholieke Universiteit Nijmegen, faculteit Wiskunde en Informatica, op 19 januari 1996

Door:
Edward Markus Petrus Albertus Mahieu
geboren op 9 februari 1968 te Einhoven (NL)

Mentor:
dr. Th. E. Schouten
Afdeling Experimentele Informatica voor Technische Applicaties

Preface

Nijmegen, 17 January 1996.

Back in 1992 I decided to study Informatics at the university of Nijmegen. At that time I thought the nominal period of two years were easy to do. In the beginning I travelled between Eindhoven and Nijmegen. I thought this would not be a problem. But I was wrong so I moved to Nijmegen. Because I wanted to get to know people in this town I also changed my hockey club.

In the beginning my study had an average progress due to the way of spending my spare time (sports and working).

Before I could start rounding off my study which involved writing this thesis I needed a change in my studying attitude. Among the people who supported me in this change are Erik-Jan, Jornand, Paul and Ramon. I would like to thank them for their interest and support.

Especially I would like to thank my girlfriend Jojanneke. She gave me a lot of goodness, love and English help.

Other people who supported me in this are my mother, my father and my brothers but also the parents of Jojanneke.

I would also like to thank my mentor Theo for his help and good advice and the people who work at the section.

Edward.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is written at the Computing Science Institute (CSI) section Experimental Informatics for Technical Applications (EITA), where research is carried out in the area of processing satellite images, in order to develop remote sensing applications in agriculture statistics. The primary goal of these applications is to distinguish, identify and measure the area of crops of significant importance. A secondary goal is to make production estimates in time to support decisions made concerning agriculture development.

Segmentation is a technique recommended for the processing of satellite images in order to get good classifications. In this area research has been carried out by Schoenmakers [Sch95] who developed an integrated method for segmentation and Schreurs [Sch93] who developed and tested the efficiency of this method.

Janssen [Jan95] carried out some research in the area of the exploitation of computing resources. The main objective of this research was to obtain a reasonable performance on processing large satellite images. He developed a parallel algorithm for the segmentation of images.

This work continues the above line of research. The use of the emerging Message Passing Interface (MPI) standard and load balancing issues have been studied especially.

Experimental evaluation is performed on a network of workstations.

1.1 Satellite images

The satellite images used in this research are acquired by the Thematic Mapper (TM), see [edc], of the Landsat-5 satellite. The TM sensor detects reflected radiation from the surface of the earth in the visible and near infrared (IR) wavelengths. The wavelength range for TM starts from visible (blue),

Band	Wavelength (μm)	Nominal spectral location
1	0.45-0.52	Blue
2	0.52-0.60	Green
3	0.63-0.69	Red
4	0.76-0.90	Near-infrared
5	1.55-1.75	Mid-infrared
6	10.4-12.5	Thermal-infrared
7	2.08-2.35	Mid-infrared

Table 1.1: Spectral bands of the Landsat TM sensor

through mid IR, into the thermal IR wavelength of the electro magnetic spectrum. The TM sensor has a spatial resolution of 30 meters for the visible, near IR, and mid IR wavelengths and spatial resolution of 120 meters for thermal IR band.

The satellite images of the Landsat TM have a size of 6700×5980 pixels. Each pixel contains 6 spectral bands. Table 1.1 summarizes the spectral bands of the Landsat TM sensor. The pictures that are used during the research do not contain the sixth band because the resolution of band 6 is four times lower than the other bands. The satellite image *subjune*, which represents the area of Lisboa (Portugal), is shown in appendix A, figure A.1. *Subjune* is used to extract some smaller subimages that are easier to handle in terms of execution time, e.g. the sequential segmentation of subjune takes more than 10 hours on a SUN 10/30.

The following subimages used are suba, subb and sube, they are shown in appendix A. All subimages are of size 256×256 pixels, there origins are at locations $(860, 300)$, $(0, 0)$ and $(1500, 1500)$ respectively. The average field size is different, well suited for studying the load balancing issues.

Suba is the result of taking 256×256 cells from origin $(860, 300)$, it is shown in A.2. Subb is the result of taking 256×256 cells from origin $(0, 0)$, it is shown in A.5. A new subimage sube, with one big region, is taken from origin $(1500, 1500)$ and again 256×256 pixels. The latter image is shown in A.8.

Performing segmentation on subjune, containing approximately 240 Mbytes, together with the capacity required to store and process sub results, requires optimal use of disk, memory, processor capacity and the network bandwidth.

1.2 Classification

The classification problem is to identify crops in areas of the satellite images, resulting in maps indicating the areas. The methods used to classify should have an acceptable low percentage of miss classifications. Each pixel is associated with some spectral class with a classifier, a class corresponds with a different kind of land cover. This mapping can be implemented in various ways, e.g. a one to one mapping or a mapping which delivers a probability score per class for all land cover types. The used technique operates on a per pixel basis and thus delivers a per pixel classification, e.g. maximum likelihood. Pixel classifiers have the disadvantage to be suspective to irregularities within the spectral domain. Classification performed on raw satellite images could lead to noisy images with many isolated pixels. Then the miss classification percentage would be unacceptable high. However classification is usually executed in a *chain* of algorithms, which can be represented in a general model.

1.3 General model of image processing chains

According to Gonzalez and Woods [GW93] image processing consists of several fundamental stages. First of all they mention the stage of image acquisition. The images used in this research have been acquired by the Landsat-TM satellite. Other satellites are SPOT [1] and ERS [2]. In the second stage, preprocessing is performed, which means that e.g. the noise of an image is removed. The basic goal is to gain a better result of the following stages. The next stage is called segmentation, in which the image is partioned in its objects and constituted parts. In this stage algorithms like edge detection and region growing are performed. The next stages are representation and description and image recognition and interpretation. Figure 1.1 shows the relation between the different phases. The arrows represent data exchange between the different phases.

1.3.1 Preprocessing

As mentioned in the previous section the function of preprocessing is to improve the image in ways that increase the success of the other processing methods. The used images are already corrected for geometrical distortions and for the amount of in falling sun light.

[1] Système Probabtoire pour l'Observation de la Terre
[2] European Remote Sensing satellite

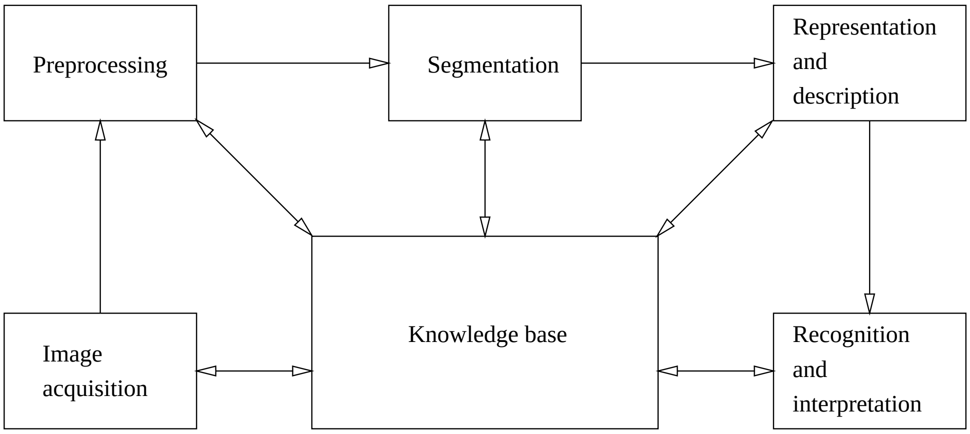


Figure 1.1: Fundamental steps in digital image processing

Filtering could be performed to remove noise from the image. One method is described by Schoenmakers, in [Sch95]. However it is difficult to distinguish between noise and not noise pixels, due to the fact that many objects on the earth surface have sizes in the order of or smaller than the resolution of the satellites. This is the *mixed pixel* problem.

1.3.2 Segmentation

The goal of segmentation is to partition the image in segments or regions, which have a high probability to represent the relevant objects in the image. In principle there are two starting strategies for segmentation:

- edge detection
- region growing

Edge detection tries to detect boundaries of the segments of the image. This is done by detecting discontinuities between a pixel and its immediate neighbor pixels. When the discontinuity between a pixel and its immediate neighbor pixels exceeds some threshold, that pixel is marked as an edge pixel. In a post processing stage the found edges could be thinned, where after edges are one pixel thick, and could be closed, where after images consist of closed polygons. Difficulties in edge detection methods are finding good threshold values. Other difficulties are the loss of information due to the edges. Even when edges are one pixel thick, they still hide information. Take into account the resolution of a satellite image which can be 30 meters. Section 1.1 contains more information about satellite images. As mentioned

w_1	w_2	w_3		z_1	z_2	z_3
w_4	w_5	w_6		z_4	z_5	z_6
w_7	w_8	w_9		z_7	z_8	z_9

Table 1.2: General 3×3 mask and corresponding 3×3 image window

before edge detection is based on discontinuities in the intensity of the grey scale. Point and line are other types of discontinuities who aren't any further explained. The result of the edge detection and polygon construction used is shown in appendix A, figures A.3, A.6 and A.9, for images suba, subb and sube respectively.

In general discontinuities are detected by comparing all pixels with their neighbor pixels. The idea behind the latter is the difference of an 8 connected pixel from its neighbor pixels in gray level or intensity. For this occasion an image window is displayed in 1.2. Some arbitrary masks, see table 1.2, are introduced to calculate the weighted sum of the window, see equation 1.1. This is done by taking the summation of all the gray scale values, of a pixel, z_i, multiplied by, its associated mask value, w_i. This is called the response value, denoted as R_w, of the mask.

$$R_w = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i \tag{1.1}$$

When R_w is above some threshold the center pixel w_5 is a discontinuity, when R_W is below the threshold it is not. Usually more than one window w is used and the resulting R_w's are compared to decide whether an edge is present and to determine its magnitude and direction.

Region growing tries to find segments in an image, in contrast with edge detection. A region growing method searches the image for groups of pixels who are almost similar. A segment or region contains one or more pixels. The segment characteristics are the mean spectral value of the pixels and the number of pixels, denoted as the size. In this method the image is scanned in some pre determined way. The value of a pixel is compared to the mean of an already existing but not necessarily completed neighbor segment. If the pixel value and segments mean are close enough the pixel is added to the segment and segments mean is updated. If two segments are close enough then the closest is taken. When two segments have precisely the same mean they are merged to one segment and the pixel is added to the new merged region. If there is no region close enough a new region will be started. This process is iterated until there are no segments that could be merged. A region growing algorithm that merges only the segments with the highest similarity value is called *best merge* algorithm. An advantage of region growing above edge detection is the information conservation, which means no information is lost between adjacent segments indicating an edge.

1.3.3 Representation and description

The partioning produced by segmentation can be used to improve the quality of the final representation. In principle two strategies can be employed:

- Using the characteristics[3] of each segment for the representation process to classify the complete segment.
- Inside each segment a pixel classification is performed. Then the segment is associated to the class having the highest frequency within that segment.

The representation can be performed on the individual pixels without using segmentations. Standard statistical methods (e.g. maximum likelihood, KMM, etc) can be used as well as neural network based classifiers.

1.4 Used segmentation method

Both edge detection and region growing segmentation have their advantages and disadvantages.

Both methods for segmentation edge detection and region growing need some threshold value. This threshold indicates how much values may differ in order to be similar or not. The problem is to find a good threshold. Because when taking the threshold to high a segmentation method would find to few edges or regions, or even worse the image remains unprocessed. When the threshold is set to low to many edges and regions may be found.

The Ph.D. thesis [Sch95] combines edge detection and region growing into a hybrid method. First his method performs edge detection and edge linking which results in closed polygons. Then region growing is performed inside the polygons. The result is a sequential implementation of the *best merge* algorithm. Besides improving the quality of the segmentation, this method also improves the execution time so that segmentation of large images is feasible.

The result of this best merge region growing is shown in appendix A, figures A.4, A.7 and A.10, for images suba, subb and sube respectively.

[3]e.g. Mean spectral values, size, etc.

The region growing inside a polygon is independent of the region growing in the other polygons thus in principle all region growing can be performed in parallel.

The master thesis of Janssen [Jan95] describes a parallel implementation of *best merge* on a cluster of workstations. In this work the network is indicated as bottle neck for the performance of the computation. The Parallel Virtual Machine (PVM) software was used as Parallel Programming Environment (PPE). The thesis describes several other PPE's, one of them was the Message Passage Interface (MPI) standard. When Janssen's thesis research started a MPI implementation wasn't available. The result of the work of Janssen is a description of parallel programming techniques and a parallel implementation of the *best merge* program on a cluster of workstations. Because MPI intents to become standard, it is a necessary research subject.

1.5 Parallel environments

Hardware systems that deal with parallel objects are called multi processor systems. A multi processor system consists of several computers connected through a network. The computers may all differ in architecture, e.g. a transputer system may be linked with conventional computers. When the computers all have the same architecture, it is called a homogeneous cluster. When the architecture differs, then it is called a heterogeneous cluster. In [DGMS93], a heterogeneous computing environment is the result of linking some high-performance workstations together with conventional supercomputers and multiprocessor architectures.

My work slightly focuses on cluster computing. The cluster used contains 12 SUN-IPC(4/40) workstations with 12 Mbytes on board memory, and 100 Mbytes swapping space on a local disk, and 12 SUN-SLC(4/20) workstations with 8 Mbytes on board memory, no local disk, swapping via the Network File System (NFS). The workstations are connected using Ethernet as Local Area Network (LAN), without bridges or other gateways.

In the near future a 16 node PowerXplorer, see [Gmb94], becomes available at CSI. It will have an MPI implementation so the experimental work of this thesis can be performed on this system with no or little software changes.

1.6 Message passing paradigm

A significant aspect of parallel computers, see [BT89], is the way in which processors exchange information. There are two extreme alternatives *shared memory* and *message passing*.

The first alternative uses a global memory that is accessible to all processors. It is preferable to use processes since a processor may run more than one process. A process may communicate by writing into the global memory and then having another process reading the same location. This paradigm solves interprocess communication, but introduces the problem of simultaneous accessing the same memory location.

In the second alternative, processors do not share memory, but have their own local memory. The processors communicate through an interconnected network consisting of direct communication links. The connection of processors is an important design choice. The best option is to connect all processors with each order, but this is not feasible. In many case the processors communicate with each other with a shared bus, which lead to access delays if much data is involved.

Another description of a message passing paradigm is adopted from [WJ89]. Here a computation is modeled in a hierarchy of objects, they communicate with each other by directing packets of data, the messages. The receipt of that message by an object stimulates that object to perform some computational steps, usually resulting in the transmission of additional messages to other objects. Although the activity of each object is typically sequential, the message passing model stresses parallelism, since many objects can be active simultaneously. MPI is based on the message passing paradigm and is used on distributed memory machines, MIMD [4] machines. The topics ready, buffered and synchronously send or receive are implementation depended. This will be explained in section 2.2 which deals with the functionalities provided by MPI.

In order to exploit a multicomputer several PPE's, such as PVM, Linda, Express and P4, have been developed and implemented. Dongarra and others [DGMS93] compares them with PVM. Another comparison is made in the book of Almasi and Gottlieb, see [AG94], where PVM, Linda and Express are introduced as widely used PPE's for writing parallel programs on distributed memory hardware. Until now the PPE's are developed for small specific purposes but their use increases because of the fact that every research institute or university owns a couple of powerful computers connected through a network. Until now the problem of the PPE's is that they use distinct hardware

[4] Multiple Instruction Multiple Data

platforms and use different paradigms. For example, I would like to mention that Linda is based on tuple space communication, PVM on message passing, P4 also on message passing. In order to get a more converge PPE the MPI forum was founded in 1992. MPI has the intention to become the standard message passing interface for parallel programming environments. In general it is expected that when a development environment contains a standard, which MPI intents to be, the developed applications will be portable. There are several MPI implementations, LAM [5], MPICH, Unify. In [NN94] these MPI implementations are discussed. For this research the LAM implementation is chosen because of its robustness. The motivation for this choice is described in section 2.3.

1.7 Problem specification

This section describes the problem I deal with. I concentrate this thesis on MPI and multi computer exploitation. The area of image processing is a very good subject for this research because of the large amount of data involved and the resulting need for optimal processing. The different processors in the multi computer perform the region growing algorithm on the polygons provided by edge detection and linking. Because of "local operations" this is a good subject for parallelization. One processor (the master) reads from a disk file and distributes them, together, with the relevant part of the satellite image, to the other (the slaves) processors. The results of region growing are send back to the master which orders them and writes them onto a disk file.

An important problem is: how can the time spent for communicating data and computing be minimized? Another important problem is load balancing: how can the needed and useful work (the region growing) be distributed over slave so that all of them are using the highest possible percentage of cpu cycles (utilization)?

The intention is to advise for further research in the area of using the future MPI2 integrated with a load balancing based on the available information from MPI and the obtained metrics. Secondary researching is carried out on LAM MPI and the use of it as a scalable multi computer and LAM MPI is also compared to PVM. Also is studied the problem of parallel region growing on the whole image.

1.7.1 Research issues

- Porting the existing parallel best merge algorithm implemented using

[5] Local Area Multi computer

PVM to an environment that uses the MPI standard for message passing and a MPI program environment, LAM MPI library.

- Description of the advantages and disadvantages of MPI in relation to PVM. Topics: communication facilities, memory management, standard versus non-standard, PVM facilities adopted in the MPI standard, ease of use and translating PVM to MPI.
- LAM MPI performance evaluation in an image processing environment.
- Researching the influence of the order of providing the polygons to the processors. More specific: providing a large polygon early in the computation should reduce the response time.
- Study about the parallel region growing algorithm itself in the MPI environment. The answered question is whether there are any new features in MPI that could increase the performance of the algorithm.
- Another interesting parallel region growing approach is scanning the whole image instead of best merging the polygons. In this case processors receive a static part of the image and communication if necessary when best merging of two regions is needed and the regions are stated on a different processor. But this is only interesting when short communication links are available e.g. the links in a transputer.
- Description of MPI and MPI2 features.

1.8 Thesis lay out

In chapter 2 the message passing interface is described. It focuses on the LAM MPI implementation and mentions others.

Chapter 3 deals with the implemented best merge algorithm from a mathematical point of view.

Chapter 4 describes the model of the parallel implemented best merge algorithm, as described in chapter 3 using the message passing interface as described in chapter 2.

In chapter 5 the measurements are described which are used in the implementation to trace the performance. The measurements are described in a general way, in order to use it at run time, but also in detail to explain how it is used. Also is described the consequence of the integration of load balancing in the algorithm.

The evaluation of the measurements are described in chapter 6.

Finally in chapter 7 some conclusions are stated and ways to use this work for further research.

Chapter 2

Message Passing Interface

This chapter describes MPI, the functionalities of MPI and the MPI implementations. It focuses on the LAM MPI implementation. A comparison is also made between LAM MPI and PVM, in case of dealing with the environment, otherwise the MPI standard is used. PVM was chosen because of the parallel region growing application described in [Jan95].

2.1 Introduction to MPI

In the last several years much research in fully exploiting a network environment for high performance computations has been carried out. In many research centers and universities networks are available that connect workstations, or hosts. Several PPE's were designed to make use of this potential computing power. These PPE's are not compatible, as was also mentioned in section 1.6. A possible option is using MPI. The promised advantages of MPI are its intention to become standard, many implementations, good description of the interface, and the techniques already accepted in other PPE's, see [For94], e.g. PVM.

2.1.1 History of MPI

Since March 1994 there has been standard for message passing interfaces. The MPI forum, which meets regularly since January 1993, is the founder of this standard. This forum is formed by over 40 institutes and more than 60 people are involved in the research of this forum. The MPI forum has the intention to meet several goals, see [For94]:

- Implementation on a wide range of computers.

- Portability.
- Definition of syntax and semantics.
- Implementation of a core of library routines.

To reach many implementors the MPI is binded to C and FORTRAN. MPI is the first standard for programming parallel processors, at least the first that is based on message passing.

The MPI standard is a description of the message passing interface. The produced document from the MPI forum is titled *A Message Passing Interface standard* [For94]. During the writing of this thesis MPI has developed even more. The intention of the forum coming to a consensus about new features that have to be implemented in MPI2. From now on there is MPI1, the first version of MPI, and MPI2.

2.1.2 Future improvements: MPI2

The study about new features in the next version of MPI, MPI2, is going on. The discussed items are described in *minutes*, see the latest minutes for more detail [Gro95d, Gro95a, Gro95b, Gro95c, Gro95e, Lus95] of the forum meetings. One of the main topics of MPI was to keep it simple. Today more sophisticated features are described in the standard.

The main discussions about the improvements lay in the following areas:

- dynamic allocation
- real time facilities

For already described features in MPI1 such as groups and communicators this leads to revision or redefinition. Other improved topics are

Dynamic allocation

The meaning of dynamic allocation is dynamic process and processor configuration. Dynamic process configuration can be accomplished with a spawn routine which allows to start processes at a specified node or processor. Deleting was already implemented in MPI1, e.g. `MPI_Finalize()`.

Dynamic processor configuration is accomplished by providing routines to add and delete nodes from the multi computer. The nodes, subject to adding, must be present in some host list file, see also table 2.2 for such an example.

Realtime facilities

It is expected that some real time aspects, such as priority messages and channels, are added to MPI. These could of some use in the algorithm e.g. for sending management information.

2.2 MPI functionalities

This section describes the MPI features as discussed by the MPI forum. It describes how message passing is implemented in MPI, what the different communication modes are and what computation topologies can be used.

2.2.1 Point-to-point communication

In point-to-point communication two processes are involved, a sender and a receiver. Functions that deal with point-to-point communication are `MPI_Send` and `MPI_Recv`. Their use is illustrated with an example in table 2.1.

These functions can be called in the modes *blocking* or *non-blocking* and can be combined with the following communication initiations *ready*, *buffer* and *synchronous*. Figure 2.1 represents the modes of communication, the dashed lines mean the communication initiations. The standard send call is blocking, this means, it returns when a *matching* receive call is posted. The *non-blocking* standard communication procedures are `MPI_Isend` and `MPI_Irecv`. A non-blocking send immediately returns, it only succeeds when a matching receive is already posted at the time it is called.

A **ready** send may only be started when the matching receive is already posted.

A **buffered** send can be started whether or not a matching receive has been posted. It may complete before a matching receive is posted. If no matching receive is posted the outgoing message is buffered.

A **synchronous** send can be started whether or not a matching receive was posted. The send completes successfully when a matching receive is posted, and the receive operation has started to receive the messages sent by the synchronous send. If the sender and the receiver are both blocking a synchronous mode semantic, the rendezvous, is achieved.

Matching communication

In table 2.1 is an example of C program which uses the MPI send and receive routines. The routines are matching because they have the same tag (99), the same communicator (`MPI_COMM_WORLD`), the sender sends the message to

```
#include{mpi.h}
main(int argc, char **argv)
{ char message[20];
  int myrank;
  MPI_Status status;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
  if (myrank=0) /* process 0 */
  { strcpy(message,"First MPI message");
    MPI_Send(message,strlen(message),MPI_CHAR,1,99,MPI_COMM_WORLD);
    /* MPI_CHAR message type
       number of MPI_CHAR's is strlen(message)
       1 is destination
       99 is message tag
       MPI_COMM_WORLD communicator
    */
  }
  else /* process 1 */
  { MPI_Recv(message,strlen(message),MPI_CHAR,0,99,MPI_COMM_WORLD,&status);
    /* MPI_CHAR message type
       number of MPI_CHAR's is strlen(message)
       0 is destination
       99 is message tag
       MPI_COMM_WORLD communicator
     */
    printf("received :%s\n",message);
  }
  MPI_Finalize();
}
```

Table 2.1: Example of a C program using MPI

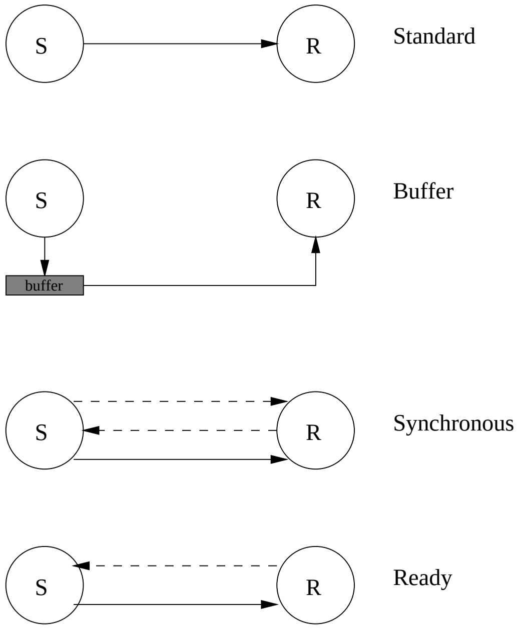


Figure 2.1: Communication modes

the process with communication rank 1 (which is the rank of the receiving process) and the receiver expects from the sender with rank 0 and also the message type is the same. This is an example in which the receive call is limited to only handling this message. MPI also provides less restricted features such as `MPI_ANY_SOURCE` or `MPI_ANY_TAG`, all in the context of the communicator. See section 2.2.3 for more details about the communication context.

2.2.2 Collective communication routines

Collective communication deals with message exchange between many (more than two) processes, e.g. a process collects messages from all other processes in the same context. As soon as a calling process has ended its operation it returns, this means i.e. the gathering operation has to wait until all messages have arrived.

broadcast provides the possibility to send a message to all the processes in a specified communication group. It has to be called by all the processes in that group. On return, the buffer at the sending process (root), has been copied to all. It could be used in the parallel program to provide all processes at once with e.g. the involved image size.

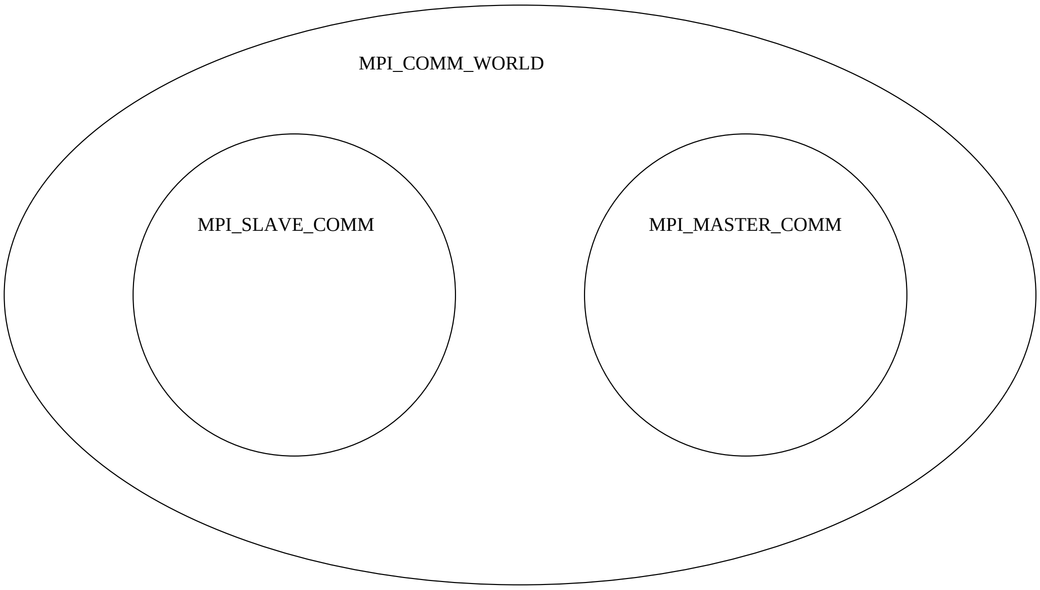


Figure 2.2: Communicators in MPI

scatter and gather the root process, when scattering, sends a message to all the processes in the communication group. Gathering does the opposite of scatter, it receives messages.

all-gather/scatter is the same as gather except that all processes receive the messages from all other processes instead of only the root process.

alltoall is the integration of all gather and all scatter

2.2.3 Communicators, context, groups

Figure 2.2 shows three *communicators*, where `MPI_COMM_WORLD` is the initial communicator. Each process in a group is associated to some integer rank. The other two `MPI_SLAVE_COMM` and `MPI_MASTER_COMM` are user defined communicators. A process in one of the user defined communicators has two contexts, the user defined and the initial communicator. A message send to a process in a context could not be received by a process in another context.

A group is an ordered set of process identifiers. Processes are implementation depended objects.

2.3 Portable MPI implementations

In an article about MPI implementations Nupairoj and Ni [NN94] made a comparison of several MPI implementations: LAM MPI, Unify, MPICH and CHIMP. This section describes the history and background of the implementations in a short way. The implementations based on the MPI standard of the MPI1 forum. Outside the scope of this research were the vendor implementations of MPI.

LAM MPI [Bur89] is a parallel software environment implemented by the Ohio Super computing Center and a subset of the TROLLIUS [1] operating system. This implies that LAM has its own message passing routines, partly described in [Cen94], but these are not a subject in this research. Interesting about LAM are its libraries for both MPI and PVM, see [BDV94], which are implemented on top of the LAM message passing routines described earlier in this subsection. This is also of great importance for migrating to MPI. The PVM routines in LAM PVM are a subset of the PVM routines provided by PVM 3.x. LAM PVM leaves out the routines for dynamically adaption of the configuration: spawning processes, adding and deleting hosts. The experimental results of compiling the original parallel best merge program developed under PVM 3.3.0. with LAM PVM libraries failed. There wasn't much time for more experiments on this subject.

Unify is an MPI implementation that is developed to migrate PVM programs to MPI. The unify system approach evolutes in several stages. The first calls to MPI functions make use of an MPI implementation on top of PVM. This makes it possible to have MPI and PVM routines and functionalities in one program. Later stages provide the full translation of PVM to MPI in which there are no PVM remnants and in which the MPI functions are directly called, not by PVM environment. In the first phase some extra latency will appear for passing the arguments to the PVM environment. This will lead to a lower performance.

MPICH is developed by the Argon National Laboratories and Mississippi State University. It is portable to all parallel machines that are running Chameleon, P4 or PVM, see also section 1.6. Its implementation is the basis of IBM's MPI and Intel's MPI, and is used by Meiko.

[1] Operating system designed for multi computer purposes

```
atlas.cs.kun.nl     edwardm
studs2.sci.kun.nl   edwardm
dionysos.cs.kun.nl  edwardm
zeus.cs.kun.nl      edwardm
```

Table 2.2: Example of an boot schema

CHIMP is implemented by the Edinburgh Parallel Computing Center. Their issue is creating a portable programmable and efficient library for their cluster message passing applications. CHIMP MPI is the basis of Cray's MPI.

Concluding remarks

The LAM MPI implementation is further used in this research, because at the time the choice for an implementation was made this was the most stable, it was easy to install, easy to use and there were no bugs to be encountered. Unify isn't chosen because disregarding its strong performance because of the implementation on top of PVM and the fact that it wasn't a full MPI implementation. The other two MPI implementations MPICH and CHIMP showed some memory management problems.

2.4 LAM MPI

In this section LAM MPI is described in more detail because it is chosen, from the in the previous section mentioned portable implementations, to exploit the multi computer environment. Figure 2.3 provides an overview of the LAM environment.

The starting of an MPI program has to be preceded by booting the hosts that are desired for the computation. This is accomplished by the *lamboot* program, which is delivered with the LAM MPI system and starts the LAM daemons at the specified hosts. The *lamboot* requires a file with the hosts that have to appear in the multi computer, which is given on the command line, table 2.2 represents an example boot schema. The routines containing the total algorithm have to be started up by a sort of *launcher* called *mpirun*. The *mpirun* program gives the possibility to start one job at several hosts. The algorithm that was implemented in this research makes use of the possibility to start an application schema which contains several processes and the places where they have to be started.

In section 4.1.2 the startup of the system is described in more detail.

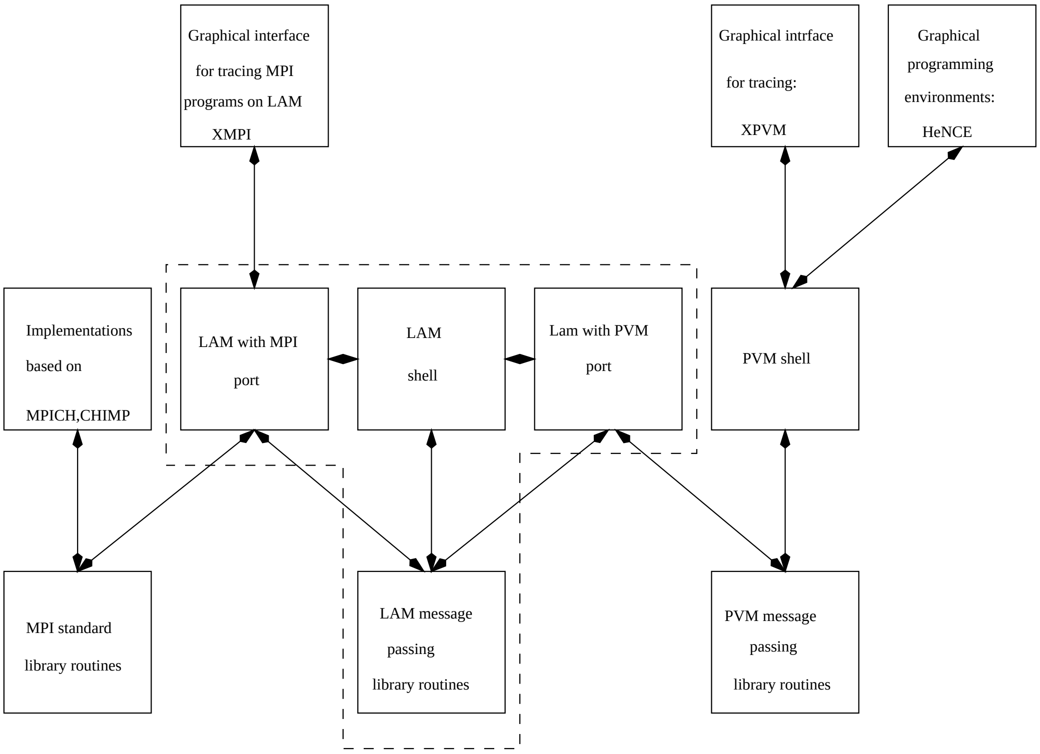


Figure 2.3: Overview of LAM

It is not possible to use debug tools such as *xdbx*. The developed program is debugged in a conventional[2] way.

There is a trace tool available, called *XMPI*, but this tool uses the graphical user interface (GUI) *OSF/Motif*, which is not available at the CSI machines.

2.5 PVM and MPI

As mentioned before in section 2.3 the LAM MPI and Unify MPI implementations contain both a PVM and a MPI interface. It would be easy to migrate a parallel algorithm implemented in PVM from PVM to MPI when using one of these MPI implementations.

[2] using `printf` routines

2.5.1 PVM description

As was mentioned PVM is another PPE that provides the possibility to exploit a network of workstations. This PPE is used in the parallel region growing algorithm, described in [Jan95] for clusters of workstations. PVM is the integration of a description of library routines for message passing (what the MPI document describes) and also an implementation of the PVM routines, and provides the PVM console, see [SJGM94]. The main difference between PVM and MPI at the level of functionality is that PVM provides the user with functions that make it possible to add and delete hosts from multi computer environment. This is necessary when implementing a parallel algorithm which uses dynamic allocation.

Contributions to parallel programming interfaces

Other interesting evolutions are parallel programming tools e.g. HeNCE [3], it is implemented on top of PVM see [BDG+94], it provides the possibility to write parallel programs in a graphical way. In this tool the computation is modeled in a task graph, where the tasks in the graph are the computation objects. The computing network has to be modeled also in a graph. HeNCE maps the task graph on the computing network graph and uses arbitrary numbers in a performance matrix indicating the performance of a task on a computing node. This isn't dynamical because during execution the load may change.

An opportunity to PVM is its trace tools. The LAM MPI implementation also has a trace tool XMPI. Unfortunately, this tool uses OSF/MOTIF library routines which are not supported by the university.

It might also be worthwhile to use load balancing strategies to fully exploit the multi computer. The goal of load balancing is dynamically adapting to workstation load, the problems of load balancing are partly described.

2.6 Translation from PVM to MPI

The translation of a parallel program, which makes use of PVM, to a parallel program which makes use of MPI, with the same semantics, two levels can be distinguished. These levels have also been encountered in [PLVC94]. The first level deals with the one-to-one mapping of PVM functions onto the MPI functions. At this level little knowledge of MPI is required, only the syntax of the different functions has to be learned, because of the same semantics.

[3] Heterogeneous Network Computing Environment

The functions ment are point-to-point communication, some collective communication, broadcast and the environmental functions. At the second level more semantic knowledge of PVM and MPI is required due to functions that can't be mapped directly from PVM to MPI. The functions ment are communicator functions and the collective communication functions, except for broadcast.

2.6.1 Process identification

A PVM program is a collection of several hosts which contain zero, one or more processes. A process has a unique identification by its PID[4] and a TID[5]. Every MPI process has a unique communication rank number in the global communicator `MPI_COMM_WORLD`. The first process entering `MPI_COMM_WORLD` gets number 0, the second 1 and so on. MPI has the capability to create new communicators. In figure 2.2 the levels in the communicators are shown. So a process could have several communication ranks in different intra communicators. Intra communicators serve the purpose for communication between processes in a single group of processes. The task of the inter communicators is the communication between two groups of processes. Groups can be manipulated separately and the communicators are used to invoke communication operations.

[4] PVM processor identification number

[5] PVM task identification number

Chapter 3

Best Merge Algorithm

This chapter deals with the mathematical background of the best merge algorithm as described in Schoenmakers [Sch95], Schreurs [Sch93] and Janssen [Jan95] and improved by Schoenmakers and Schreurs.

3.1 Introduction

The functions of the region growing algorithm are described in section 1.3.2. A problem (another problem is described in section 1.3.2) of conventional region growing methods is the order dependence problem. This problem is solved by the simultaneous order independent iterative merging algorithm (SOIM), which was developed by Schoenmakers and is originally described in[Sch95]. Schreurs developed a version of this algorithm in which region growing is performed in the polygons detected in earlier preprocessing stages. This algorithm is called consistent iterative parallel merging (CIPM) and is described by Schreurs [Sch93].

3.2 Simultaneous Order Independent Iterative Merging

A description of the follows the SOIM best merge algorithm, as developed in Schoenmakers PhD thesis [Sch95], is given.

3.2.1 The similarity & stopping criterion

The similarity criterion used in SOIM differs from criteria used in methods developed by Tilton and Beaulieu & Goldberg. The main difference is not being dependent of the size of the segment. The regions are merged only based on their spectral values. The merging criterion simply takes the Euclidean spectral distance between the regions i and j.

$$C_{i,j} = \sqrt{(u_{1i} - u_{1j})^2 + \cdots + (u_{ci} - u_{cj})^2} \tag{3.1}$$

Where c is the number of channels and u the spectral value. The Schoenmakers cost function is represented in equation 3.1. In this function is c the number of channels and u the spectral value.

Some threshold value is simply taken for the stopping criterion.

3.2.2 Initialization

Schoenmakers algorithm merges two pairs with the same similarity values simultaneously. This is the main difference between this algorithm and the algorithm developed by Beaulieu & Goldberg. Another pre is the space saving due to not creating a neighbor list. The algorithm that is given in 3.2.4 starts with an initial picture partition $S = \{S_1, S_2, \cdots, S_n\}$. At each iteration two or segments with an equal similarity criterion value are merged. The initial picture contains as much segments as the number of pixels. The variables used in the algorithm are defined as follows:

1. M_i, mean intensity value of segment S_i
2. N_i, the size of S_i (number of pixels involved)
3. $P_i = \{e_1, e_2, \cdots, e_n\}$, set of pixels contained in S_i
4. $C_{i,j}$, the cost of merging segment S_i and S_j, where S_i and S_j are adjacent
5. T, merging threshold depending on function C

3.2.3 SOIM: Algorithm in words

This section is describes the SOIM algorithm, starting with the steps contained in part 2, *Iteration of merging the most similar segments*, as stated in section 3.2.4. Step 2a calculates the set of pairs S_u, S_v having minimal cost C_u, C_v. Hereafter step 2b determines for each pair S_u, S_v their neighbors and their costs, if such neighbor is member of CS. In step 2c all segments in CS_{min} and CS_{nbr} are removed from CS to create a new set CS'. In step 2d all pairs having minimum cost are merged in parallel. If more than one segment pair share a common segment they will be merged into one. In step 2e the segments which are included in the segment pairs in CS_{min} are removed from S, since these have been merged. In step 2f the segments that are created and are member of S_{new} are added to $S^{'}$. In step 2g for all members S_i included in S_{new} the mean spectral value is calculated. In step 2h the new cost values are calculated for the segments S_i in S_{new} for merging with there neighbor segments S_j. The steps 2i – 2k the new set of segment pairs is created where each pair has at least one element in S_{new}, the pairs are adjacent and the cost of merging is below the threshold T. The new segment pairs found in steps 2i – 2k are added to $S^{'}$ in order to update $CS^{''}$, in step 2l. Step 2m updates CS to contain segment pairs that are within the threshold. The final iteration step 2n updates the set of segments that makes the image.

3.2.4 SOIM: Algorithm

1. Initialization

   (a) $S = \{S_1, S_2, \cdots, S_n\}$

   (b) set value of threshold T

   (c) merge S_i and S_j if $M_i = M_j$ and S_i and S_j are adjacent

   (d) $CS = \{\langle S_i, S_j\rangle \mid S_i \ and \ S_j \ adjacent \ and \ i < j \ and \ C_{i,j} < T\}$

2. Iteration of merging the most similar segments

   (a) $CS_{min} = \{\langle S_u, S_v\rangle \mid \langle S_u, S_v\rangle \in CS \ and \ C_{u,v} = \ MINIMUM\}$

   (b) $CS_{nbr} \ \{\langle S_u, S_k\rangle, \langle S_v, S_l\rangle \mid \langle S_u, S_k\rangle \in CS \ or \ \langle S_v, S_l\rangle \in CS\}$

   (c) $CS' = CS - CS_{min} - CS_{nbr}$

   (d) simultaneously create a new set S_{new} of all possible new segments S_m, each from K segment pairs $\langle S_x, S_y\rangle$ such that:

   $$P_m : \{e \mid e \in \bigcup_1^K [P_x \cup P_y] \ such \ that \ \langle S_x, S_y\rangle \in CS_{min}\}$$

   and the K segment pairs are spatially contiguous

   (e) $S' := S - S_i$ where S_i is one component of pair $\langle S_p, S_q\rangle$ and $\langle S_p, S_q\rangle \in CS_{min}$

   (f) $S'' := S' \cup S_{new}$

   (g) $\forall S_i \in S_{new}$: calculate M_i

   (h) $\forall S_i \in S_{new}$: calculate $C_{i,j}$ where $S_j \in S''$ and $S_{i,j}$ adjacent

   (i) $CS_{new1} = \{\langle S_i, S_j\rangle \mid \forall S_i \in S_{new} : S_j \in S'', \ i < j, \ C_{i,j} < T \ and \ S_i, S_j \ adjacent \ \}$

   (j) $CS_{new2} = \{\langle S_i, S_j\rangle \mid \forall S_j \in S_{new} : S_j \in S'', \ i < j, \ C_{i,j} < T \ and \ S_i, S_j \ adjacent \ \}$

   (k) $CS_{new} = CS_{new1} \cup CS_{new2}$

   (l) $CS'' = CS' \cup CS_{new}$

   (m) $CS := CS''$

   (n) $S := S''$;

   (o) Go back to 2a unless stopping criterion 3a fulfilled.

3. Stopping condition

   (a) $CS = \emptyset$

Chapter 4

Parallel Image Processing

This chapter deals with two parallel image processing algorithms.

Designed and implemented is performing SOIM region growing on the closed polygons of an satellite image in parallel using the LAM MPI programming environment.

It is shown in Janssen's thesis [Jan95] that the influence of communication overhead is significant for the overall performance, when implemented on a cluster, but for later implementations on a transputer system with faster communication facilities it will be useful.

Further a design is given on performing SOIM itself parallel on a whole satellite image.

4.1 Parallel SOIM within closed polygons

This section describes the parallel segmentation algorithm within closed polygons, implemented using MPI library routines of LAM as PPE. This algorithm is based on the master/slave model. The master process is divided into a master provider and a master receiver. This is a remnant of the PVM implementation of this algorithm in which the influence of the communication overhead on the computation was also tested. It was expected that two processors, containing both one master process, would have enough time to do their work. The current MPI implementation doesn't provide routines to start processes on a specified host. The processes have to be present initially. They preferably are started from a shell script which contains the processes and their computing nodes. Figure 4.1 gives an overview of the parallel region growing algorithm within the closed polygons.

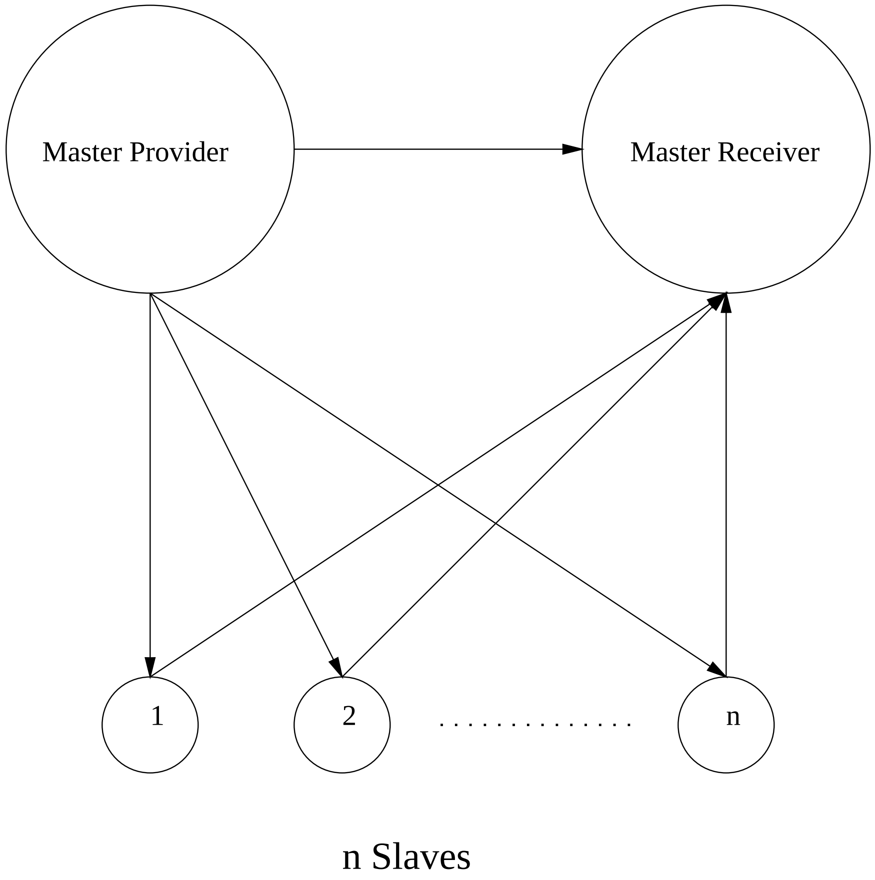


Figure 4.1: Process model of parallel region growing

4.1.1 Process description

The algorithm is divided into several processes, which have been described in this section. But each process consists also of the parts:

- SOIM algorithm, the best merge program
- MPI part, or management, which performs the communication and program setup.

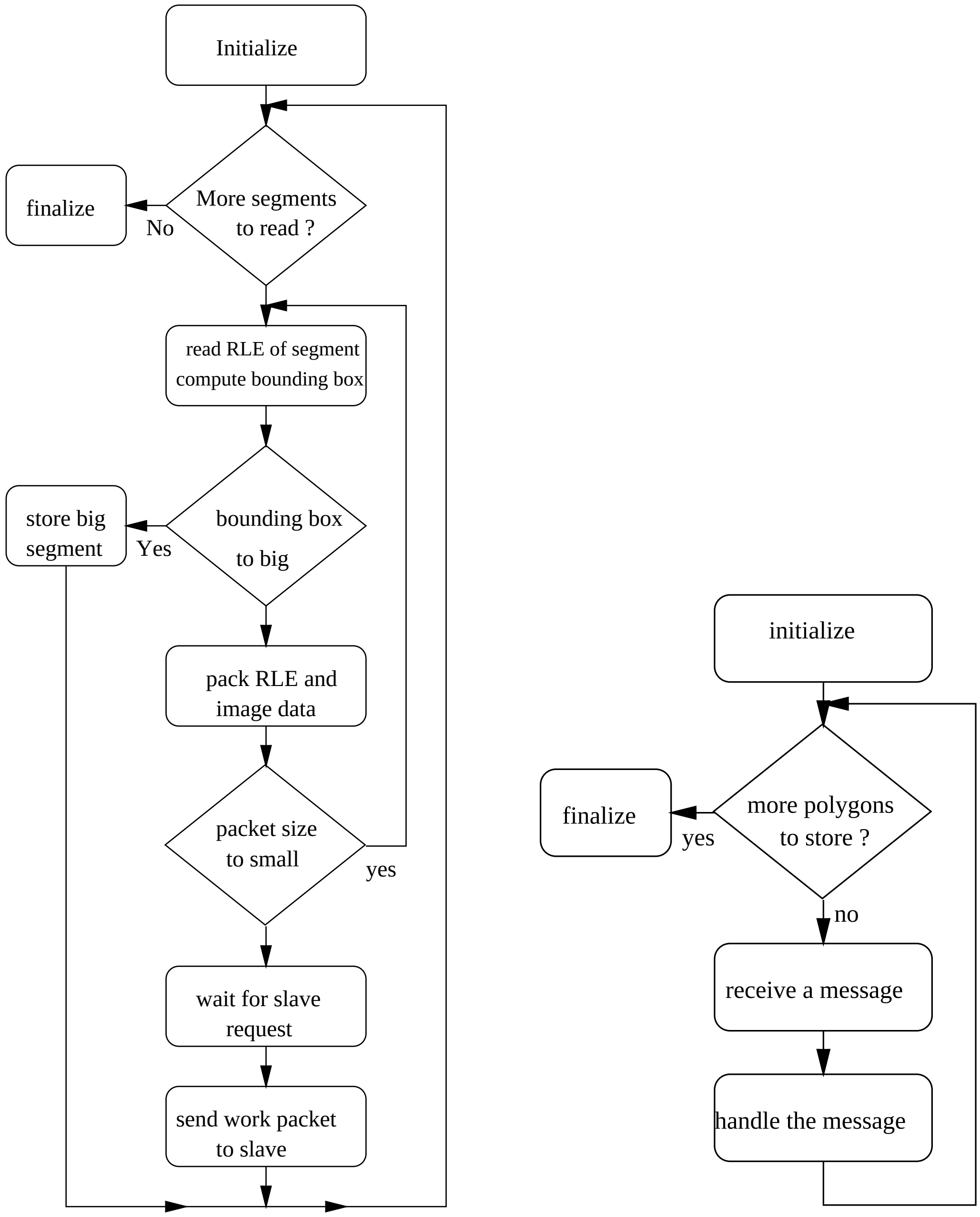
Master provider process

The implementation of the parallel best merge algorithm allows only one master provider process. The master extracts information about the amount of image data from the image data file and this information is send to the master receiver. The provider reads the next image segment and calculates the bounding box. If the bounding box is too[1] big, it is immediately sent to the receiver to store. Otherwise if the current work packet size is too small the next segment is packed in the same message. The next requesting slave gets the packed information. When all segments have been read the master provider kills the slaves and quits. Figure 4.2(a) shows the steps that will be taken in the provider process.

Master receiver process

The configuration of this program allows only one master receiver process stated at the same node as the master provider. The master receiver process receives the results of the slaves and stores them in the output file. To start with, the master receiver process gets information from the master provider about the expected size of the results. During execution, information is received from the provider, if the bounding box is too big and processed image data is received from the slaves. When the expected information is received, the master receiver process kills itself. Figure 4.2(b) shows all the steps involved in the master receiving process.

[1]there is an parameter indicating the maximum number of cells that a region may contain



(a) Provider (b) Receiver

Figure 4.2: Master processes

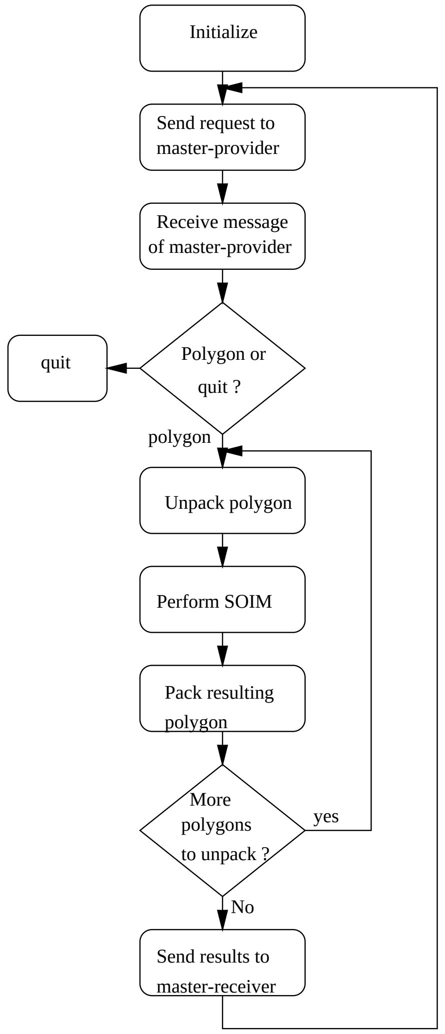


Figure 4.3: Control flow of slave process

Slave process

After initializing, see figure 4.3, the slave process sends the master a request message. The slave now receives a quit message or the next polygon. When the polygon message is received, the next polygon is unpacked and the SOIM algorithm is performed. When this is the last polygon of the packet the results are packed and sent to the master receiver process. When more polygons have to be unpacked this is done first followed by another SOIM performance.

```
lamboot bm.hosts.2slaves
mpirun -w bm.application.schema.2slaves
wipe bm.hosts.2slaves
```

Table 4.1: Shell script to start up the system

```
oker.sci.kun.nl   edwardm
indigo.sci.kun.nl edwardm
roze.sci.kun.nl   edwardm
```

Table 4.2: Host file

4.1.2 Startup in LAM MPI environment

To avoid complications when starting the system from the command line, it is preferable to start up from a shell script as stated in table 4.1. This has been mentioned also in section 2.4. The lamboot program actually starts the lam daemon on the indicated hosts in the host file. The host file `bm.hosts.2slaves` is listed in table 4.2. It contains the computing nodes, in this the three IP host names, which internally will be called $n0$, $n1$ and $n2$. The application schema contains the processes of the system, see 4.1.1, and some parameters. An example of three node application schema is given in table 4.3. There are parameters for the system and parameters for the processes, distinguished by "`--`". In this example the *bm_master* and *mast_rcv* processes are placed on node *n0* and the two *bm_slave* processes on nodes *n1 ... n2* (in this case *n1* and *n2* respectively). The provided parameters to the processes are respectively: $suba32.lan$ name of the image, 8.00 threshold, $suba32.pinf$ polygon information file, y means yes on the question applying knowledge rules, s Schoenmakers cost function, see also 3.1, 2 the number of slaves, 5000 the number of cells which may be packed in one message and 1000000 maximum number of cells in one region. The *wipe* program, see table 4.1, shuts down the multi computer, it deallocates the memory used by pending buffers and removes all daemons from the used nodes.

```
bm_master n0 -s n0 -- suba32.lan 8.00 suba32.pinf y s 2 5000 1000000
mast_rcv n0 -s n0 -- suba32.lan 8.00 y s
bm_slave n1-2 -s n0 -- 8.00 y s 2 5000 1000000
```

Table 4.3: Application schema

4.1.3 Actual used MPI routines

From figure 4.1 may be concluded that collective communication could be used. This is partly true but it wouldn't lead to revolutionary performance increases or nicer source code. Collective communication is useful when the same message has to be send to all other processes (in the same context). The routines that MPI provides are described in section 2.2.2. There are two program stages who could be subject of collective communication. The first is the stage in which the information about the image is send from the master provider to the other processes. The second is for collecting the timing information from all the processes by the master provider. In the first stage the information has to be adapted so that all the other processes can handle all information. Now the slaves get only information about the number of channels and the rank of the master provider in the `MPI_COMM_WORLD` communicator, while the master receiver gets information about the number of lines and elements per line as well. Anyway it would not lead to a significant bettering because too little data is involved.

The second stage is the one in which the master provider collects timing information from all slaves. Also in this case little data communication reduction is achieved because there is not much.

The used routines from MPI are the one for program initiating e.g. `MPI_Init()`. For communication the standard send and receive operations are used. In the cases where a process expects several messages a probe routine is used, which accomplishes the right handling of messages, according to a specified tag or type.

4.1.4 Runtime problems and solutions

This section will describe the problems which were encountered at runtime.

- Trade off between RAM available and number of users.
- Trade off between RAM usage and the number of I/O operations.
- In the current implementation of the algorithm the master waits for all slaves to be ready. Due to errors it is possible that a slave doesn't finish its task. This results in a forever waiting master.
- The algorithm collapses when the polygon area is too large to fit in the RAM available.

Because of the large amount of data involved in the tests that where carried out some extra problems arose. The tests are described in section 6.

Some parts of the program make use of dynamic memory allocation, by calling the `malloc` function from the standard C library, to allocate contiguous blocks of memory of a specified size. When large buffers are used to sent and receive the messages containing image information it can happen that not enough memory is available. This problem is encountered only in the slave processes. To "solve" this problem the slaves where used to kill itself, which leads to a forever waiting master. In fact the master waits for the timing results of the slaves which don't exist any more. Two solutions to reduce the occurrence of the described problems are:

- reducing the number of buffers
- killing all the tasks in `MPI_COMM_WORLD` when a slave could not allocate enough memory

The symptoms of this problem were encountered very early in the developing and testing phase of the research. But due to the influence of workload of the used processors, this hard to determine. It is also possible that it will happen again.

4.2 Parallel SOIM onto the whole image

This section contains a theoretical setup to perform of the SOIM algorithm onto the whole image, instead of performing SOIM onto regions, as described in the previous section. The advantage of this region growing approach is the conservation of the image data where otherwise edge information is stored. A disadvantage is the communication intensity.

The idea isn't new, because the sequential version of the best merge program has got an option for region grow onto the whole image already, but the parallel approach needs a new processing structure. Initially the algorithm will divide the image is divided in equally sized parts, depending on the number processors of the multi computer. Restrictions could be set on the number of processors, $K,\ where\ K = i^2\ and\ i \in N$, this forces the image parts to be equal in the direction x and y.

Because the order independence of SOIM it can parallel perform on each image part, when a segment has to be compared with an adjacent image part, that is stored at local memory of another processor, communication is needed to exchange information. Two types of communication can be distinguished:

- Initialization
    - spawning new processes

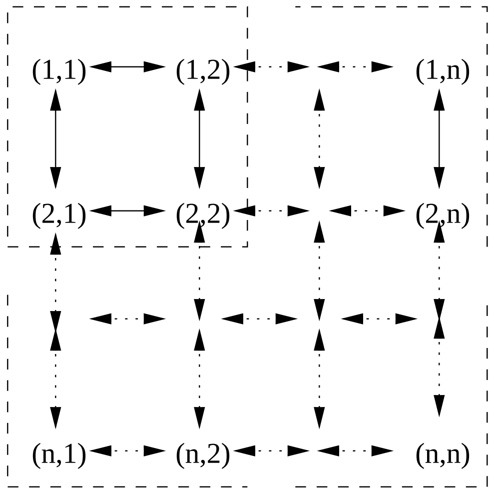


Figure 4.4: Transputer communication links

- initialization of new processors
- data distribution

- interprocess communication at runtime, because of data exchange

- shutdown for recollection

Due to the intense communication, the algorithm could not be implemented on the Ethernet network (many collisions of messages will disturb the performance). It may be expected that the dedicated communication links of 16 node PowerXplorer, see [Gmb94], is well suited to handle this communication problem. Figure 4.4 shows the conceptual communication links of the algorithm and processors n involved. When a vendor implementation of MPI is available the routines which may be used to communicate or to define topologies are interesting research subjects. Process have the possibility to derive the coordinates from the neighbor processor. MPI routines which could be used *MPI_Graph_create()* and *MPI_Cart_create()*, see [For94].

The image processing toolkit, see [JMS95], describes a set of library routines for coarse grained parallelism using MPI as PPE on workstation clusters. They describe the algorithms for image processing in a manager/worker model in which the manager process reads an image file from disk, partitions it in equally sized pieces and sends the pieces to the workers. After the workers have processed their pieces they send the pieces back to the master, which collects the processed pieces and re assembles them into a new image. The problem of the parallel region growing onto a whole image is communication, especially between the workers. This last communication isn't implemented in this libraries.

The latter seem to be the reason why they have accomplished a well performance with this toolkit.

4.2.1 Setup algorithm

The processes model is best called manager/worker. The manager could be placed at some conventional node (ment is a one processor workstation) or on a node in the parallel hardware. When the latter is the case the manager could possibly perform region growing also.

Manager The manager process reads the image data from a disk file and divides them in equally sized parts depended on the number of workers/processors. It starts up the workers at their nodes in the multi computer and sends the image data to them. After the image parts are processed at the worker processes the manager collects them and places them in a disk file.

Worker The worker processes performs the region growing on the image part, which it has in its local memory. When information is needed from a neighbor process communication is needed to get that information. When the region growing is finished the worker sends the processed data back to the manager process.

Implementation issues Some implementation issues are the adaption of the communication and of the storage of data. During region growing a segment might want to check if a neighbor computing node contains an adjacent image part which has a segment with the same characteristics, see 1.3.2, and therefore could be merged, into one segment. This will lead to some communication which isn't expensive on the parallel hardware and some adaption of the administration to indicate that a pixel already belongs to a segment. Another possibility is to leave the merging of adjacent segments (placed at different neighboring nodes) to the manager process placed at a conventional workstation.

Chapter 5

Measurement and Load Balancing

This chapter deals with the measurement, and the use of it to control the performance, and load balancing, taking communication time into account. Communication is an important issue on distributed memory machines (cluster) because of its high cost. Therefore during the development of algorithms in such environments one should keep track on the communication overhead and distances among processors. This section will provide more information about the integration of load balancing in the algorithm. In order to take decisions on workstation cluster adaption an overview of the metrics area is given.

5.1 Evaluation metrics

Before the evaluation will be discussed, first an overview of this area is given. This is derived from Kant's book [Kan92]. When starting performance evaluation of a computer system one should know the type of metrics that exists, the metrics will be used for performance indication. A computer system usually fulfills certain expectations i.e. in terms of the response time or has to operate near its highest operation power under the given resource constraints.

The implementation of parallel best merge algorithm belongs to the system category where the measurements are in terms of responsiveness as a performance indicator. Due to the fact that the operation has to be as fast as possible the measuring is in terms of utilization and response time. Other not explained categories are the usage level, mission-ability, dependability and productivity.

5.1.1 Techniques for metrics

Several techniques can be used to evaluate the system performance in some of the categories as described in the previous section. The metrics can be used in several stages of the development and/or maintenance of the system. For example a technique can be used before the implementation of an algorithm to compare the behavior of the system with the expectations, a technique can be used when the implementation needs some fine tuning or techniques can be used at run time. Techniques for metrics are a way to compare performance with expectations:

measurement It is the most fundamental technique used for metrics. It is used together with statistical analysis to obtain accurate data.

simulation It involves constructing a model for the behaves of the system driving it with an appropriate abstract of the workload.

analytical Tries to solve a model of the system. Gives good insight of the behavior of the system. But isn't rather detailed in its results. Fast manner of analyzing the system performance.

5.1.2 Measuring

In this research the measurement technique is used to evaluate the parallel best merge algorithm. There are two variations in measurement, software and hardware. The software measurement makes use of a programmable timer used for measuring the time duration of certain processes in the system. The timer access can be embedded in the existing program, which is preferable in this case because of later use in dynamic adaption of the multi computer, or can be implemented as separate sets of routines e.g. in a trace tool. The hardware measurement has the advantage of not interfering with the functioning of the program, due to the availability of dedicated hardware.

5.2 Metrics in detail

The scope of responsiveness, mentioned in section 5.1, is evaluating how quickly the total process can be accomplished. Possible measures are processing time, waiting time, queue length and conditional waiting time (waiting time for tasks that require a specified amount of processing time). In this research the following measures are used:

Utilization This is a *derived* metric, it is computed dividing the cpu time, T_{cpu}, by the response time, T_R. It shows the processor utilization of a process during computation.

$$U = \frac{T_{cpu}}{T_R}$$

Response time For this metric the maximum elapsed wall clock time of the involved processes in the computation is taken. This will usually be the *master-provider* process. But taking the problem into account that the other processes are not started by the master but from a MPI application schema, as described in section 2.4 and 4.1.2, another process could start earlier, but the master-provider will always be the last process, due to the fact of waiting before all response times are collected.

Speedup This metric is derived from the response time. The speedup is calculated to indicate the scalability of the algorithm.

Scalability can be indicated in two ways, assuming no communication time and a fully parallelizable code:

1. The speedup should remain constant when the fraction of computing data is in the same fraction as the involved processes.
2. When the involved data remains constant, the speedup has to grow in fractions of the involved processors in the computation.

The calculation of the speedup is based on a two processor model, with one slave and one master, which has speed one. When two slaves are used the algorithm should be twice as fast. The response time of the two processor model is taken as scale factor for the other measures.

The ratio for calculating the speedup is:

$$S_n = \frac{T_2}{T_n},\ where\ n \geq 1$$

where S_n represents the speedup ratio for n processors, T_2 the response time for the 2 processor model and T_n the response time for a n processor model.

5.3 Performance control

In this section performance control is introduced because the measurements are required to minimize the response time. The next step is using the measurement for performance control, until now the performance had been analysed post mortem. But the metrics as described in the previous section can also be used for real time control. Possible metrics are response time, memory usage or percentile measure such as processor utilization. One of these metrics is referred to as the performance index, which leads, compared with the desired level at run time, to an adaption of a control parameter (CP). Control parameters (command line options) are:

Number of cells per slave The maximum number of cells (memory usage) that are packed in a slave message in order to be processed

Number of processors Number of processes involved in the computation can be adapted by the migration of processes from a source node to a destination node and deleting and adding nodes from the multi computer environment

Because of the performance there are some restrictions on the performance index parameter:

- the measuring should be continue without significant overhead
- long measurement interval cannot be used because frequent measurements provides the possibility for an accurate computer configuration
- the measurements should be used directly

5.4 Communication overhead and ideal number of processors

In [AG94] a model is given for the computation-communication overhead on the speedup and the derivation of the ideal number of processors. This model needs some adaption will it be of use for this research. Namely initially two processors are used in computation.

The number of processors is a well usable parameter for performance increasing, but in cluster computing man should be careful, because communication overhead at least linear two the number of processors. Which implies it isn't always ideal, in terms of response time, to add as much processors as are available in the computation.

This provides an introduction to use the number of processors as a parameter to adapt the performance.

Assumptions:

1. processes cannot compute and communicate at the same time
2. master process have a desired low utilization and slave processes have a desired high utilization. To explain the right of the 2 processor model to determine the speedup.
3. the communication rate is very low. In order to accept the two processor model as one fully parallelizable process.

From assumption 1 is derived that the time needed to compute T_n can be written as:

$$T_n = T_{comp_n} + T_{comm_n} + other\ terms \tag{5.1}$$

, where T_{comp_n} is the computing of T_n, T_{comm_n} the communication time T_n and *other terms* will be neglectible low.

Assumption 2 is stressed to use the response time for the two process model for computation of the ideal response time for n processors, which is right from assumption 3. So the ideal response time T for n workers becomes,

$$T_{comp_n} = \frac{T_2}{n}$$

. And

$$n \times T_{comp_n} = T_2$$

.

The ideal speedup

$$S_n = \frac{T_2}{T_n}$$

is rewritten as:

$$S_n = \frac{n \times T_{comp_n}}{T_{comp_n} + T_{comm_n}}$$

. The speedup ratio is:

$$\frac{1}{S_n} = \frac{T_{comp_n} + T_{comm_n}}{n \times T_{comp_n}} \Longleftrightarrow \frac{1}{S_n} = \frac{1}{n} + \frac{T_{comm_n}}{T_2}$$

.

The computation-communication ration is:

$$r_{cc} = \frac{T_2}{T_{comm_n}}$$

, the larger it is the better.

The rewritten speedup ratio becomes:

$$\frac{1}{S_n} = \frac{1}{n} + \frac{1}{r_{cc}}$$

. Since the assumption that the communication time is proportional to to the number of workers n, the communication time for

$$T_{comm_n} = n \times T_{comm_2}$$

.

Again the speedup ratio is rewritten now as follows:

$$\frac{1}{S_n} = \frac{1}{n} + n \times \frac{T_{comm_2}}{T_2} \Longleftrightarrow$$

$$\frac{1}{S_n} = \frac{1}{n} + \frac{n}{r_2} \tag{5.2}$$

with r_2 the computation-communication ratio for the two processor model where:

$$r_2 = \frac{T_2}{T_{comm_2}}$$

.

The derivation of the best possible configuration in terms of n, the number of workers, is calculated from equation 5.2 taking the first derivative as follows:

$$\frac{d(\frac{1}{S_n})}{d(n)} = \frac{-1}{n^2} + \frac{1}{r_2}$$

To find any extreme values the latter equation is set to zero:

$$\frac{-1}{n^2} + \frac{1}{r_2} = 0 \Rightarrow$$

$$n_{opt} = \sqrt{r_2} \Longleftrightarrow n = \sqrt{\frac{T_2}{T_{comm_2}}} \tag{5.3}$$

The substitution of the n from equation 5.3 in equation 5.2 gives:

$$\frac{1}{S_n} = \frac{1}{\sqrt{r_2}} + \frac{\sqrt{r_2}}{r_2} \Rightarrow$$

$$S_{n_{opt}} = \frac{\sqrt{r_2}}{2} \tag{5.4}$$

The subscription *opt* to n and S_n indicates a maximum extrema, which will be indicate in the next chapter.

The next section contains a framework for performance control.

5.5 MPI load balancing approach

In this section load balancing issues will be introduced. The goal of load balancing is to distribute work load evenly among the processors at run time. A load balancing strategy could improve the performance of systems which frequently change in performance.

The features that a PPE needs to implement it are spawning processes at other nodes and adding and deleting nodes, see also section 2.1.2.

There are two methods to implement load balancing: in the parallel algorithm itself or in the PPE, e.g. MPI2.

Because PVM exists longer than MPI there is more research done in order to implement or integrate load balancing. In [LDS94] a dynamic PVM approach, with some extra features, is described. DynamicPVM is based on the assumption that 80% of the workstation in a cluster remains idle, which means there is enough computing time available for load balancing. The system works together Condor (a hunter for idle workstations). In this article some features of load balancing algorithms are described: process checkpoint and restart mechanisms.

The parallel SOIM algorithm uses all computing power for execution, which means the introduction of load balancing would have too much influence on the results. Also the communication cost of the algorithm is too expensive for load balancing.

Although integration of load balancing might be more efficient when the following constraints are fulfilled:

- The used MPI implementation is more optimized
- Some restrictions are send on the maximum number of processors in order to minimize the communication intensity
- The used MPI implementation uses the MPI2 standard

As already mentioned the introduction and the influence of load balancing on the performance of an algorithm in a distributed memory environment depend heavily on communication cost. When developing such an algorithm some attention should be given to communication. Some usable model is described in the article *A uniform framework for dynamic load balancing strategies in distributed processing systems*, see [ZM94] for further reading. The basis of this framework is the calculation of a load contention number (LCN) that accounts the load of the processors, the communication cost and distances among processors. The performance of a load balancing depends on migration techniques and all other issues are integrated.

Chapter 6

Evaluation of the results

This chapter deals with the performance of the SOIM algorithm on several satellite images, as described in section 3.2.4. The test runs were held during night in order to avoid as much influence of other processes as possible. Although it is still possible that other processes have had some influence[1] on the results. The goal of the tests is to derive information from the metrics about the influence of the parameters, work packed size, number of processors, internal regions and providing order on the performance, in order to future adaption of the multi computer configuration at run time.

6.1 Image characteristics

In order to acquire a good evaluation of the parallel algorithm, several images are being used, all subimages were taken from the original Landsat-TM satellite image *subjune*, which is taken from the Lisboa area in Portugal. This image was already mentioned in section 1.1. The following subimages are used: *suba*, *subb* and *sube*. All images are shown in appendix A figures A.2, A.5 and A.8.

6.1.1 Characteristics subimages

Because of the big original image *subjune* and in order to get images with homogeneous characteristics as well, subimages *suba* and *subb* were created. Where *subb* is taken from an area with many small internal regions and *suba* is taken from an area with large internal regions. They both have six spectral bands and 256×256 pixels. These subimages were used to create larger

[1]showed by the peaks in the utilization and response time figures

images, containing 8 times and 32 times the original subimage. Two compositions of the subimages *suba* and *subb* were used as well to investigate the influence of the region providing order on the performance. The first composition consists of four times *suba* followed by four times *subb*. In appendix A figure A.11 this image is shown after performing best merge. The second composed image consists of four times *subb* followed by four times *suba*. Another image is a composition of *sube* and *suba*, although the image *sube* wasn't analysed in detail. The *sube* contains one big region which will lead to a higher response time.

6.2 Measurement results

In this section the measurement results are described and analysed. It contains aggregated graphs from the derived metrics as described in section 5.2. All the individual graphs from the other metrics, response time and utilization, are represented in appendices B.1, B.2 and B.3.

The following control parameters are tested: work packed size, number of processors, internal regions and providing order on the performance.

Because the algorithm didn't have the possibility to adapt the configuration at run time the metrics are analysed post mortem and later used to adapt the parameters for performance increasing.

The main goal is to keep the average slave utilization high and master utilization low in order to respond as quick as possible to slave data requests. Difficult is that low master utilization could also imply that more slaves are needed.

All test runs were performed three times in order to get accurate results.

6.2.1 Image with large internal regions: *suba*

In this section the influence of large internal regions is evaluated, which will lead to a relatively low communication rate, and the number of processors on the speedup and the utilization. The images used are: *suba* (0,75 Mb), *8* × *suba* (3 Mb) and *32* × *suba* (12 Mb). In all the test runs the threshold is set to 8 and the number of cells per slave is set to 5000.

In figure 6.1 the linear speedup is shown. It refers to the highest speedup rate in a homogeneous computer system without disturbing of communication, buffers and other users. In figure 6.1 can be seen that between 2 and 10 involved processors a speedup is accomplished of near linear. After 10 involved processors the speedup of the largest 2 images starts to flatten and of the smallest decreases. The speedup of the larger ones will also decrease but

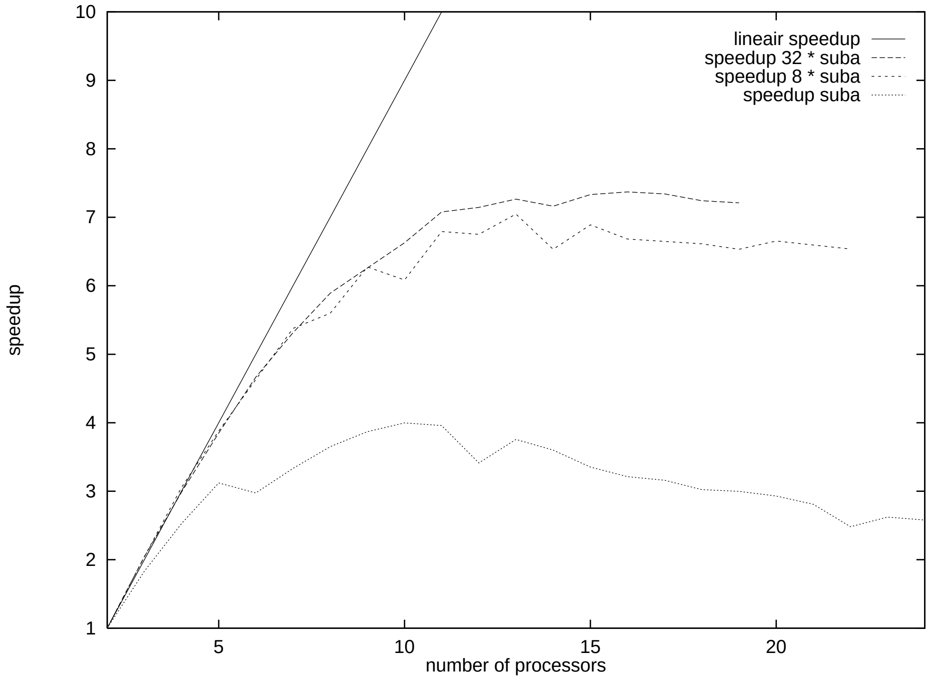


Figure 6.1: Speedup *suba* images

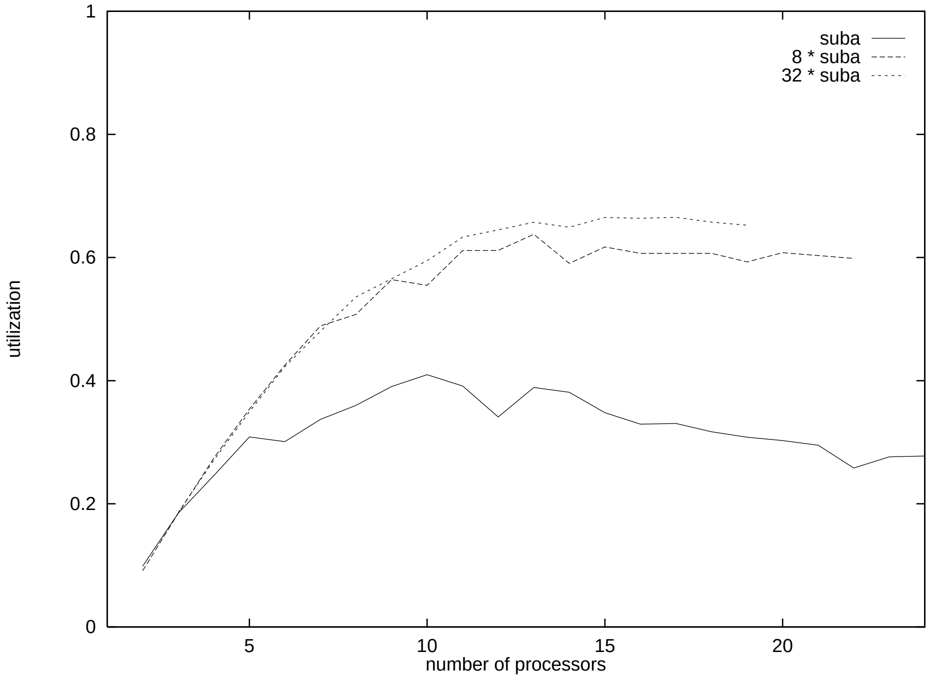


Figure 6.2: Master process utilization *suba* images

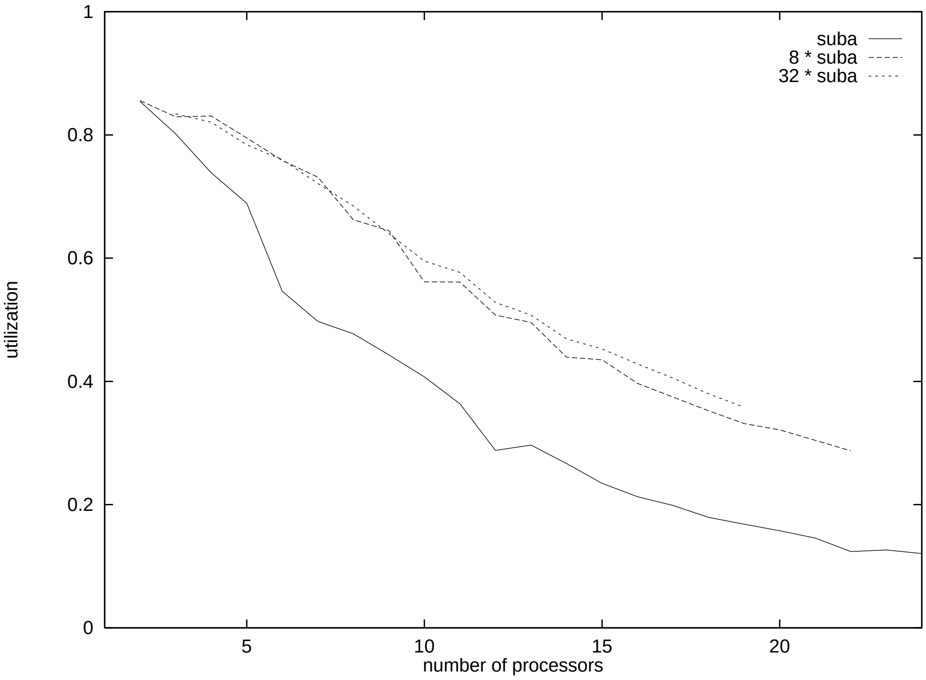


Figure 6.3: Slave process utilization *suba* images

there aren't enough processors (maximum 24) to measure that. The speedup accomplished on the two larger images is almost identical because the limits of the underlying network are reached.

In figure 6.2 it can be seen that the master process utilization is increasing. This is due to the involvement of more slaves in the computation, which have to be served by it.

Figure 6.3 shows the increasing average slave utilization on all images.

Preliminary it can be concluded that it's hard to find a good trade off between data and number of processors in order to get the highest possible speedup. It can also be concluded that a relatively low master utilization and a high slave utilization gives the best speedup range.

Appendix B.1, figures B.1, B.2, B.3, B.4, B.5 and B.6, represents the other measurements on the image with large internal regions.

6.2.2 Image with small internal regions: *subb*

In this section the influence of small internal regions is evaluated, which will lead to relative high communication rate, and the number of processors on the speedup and the utilization. The images used are: *subb* (0,75 Mb), *8* $\times$ *subb* (3 Mb) and *32* $\times$ *subb* (12 Mb). In figure 6.4 it can be seen that

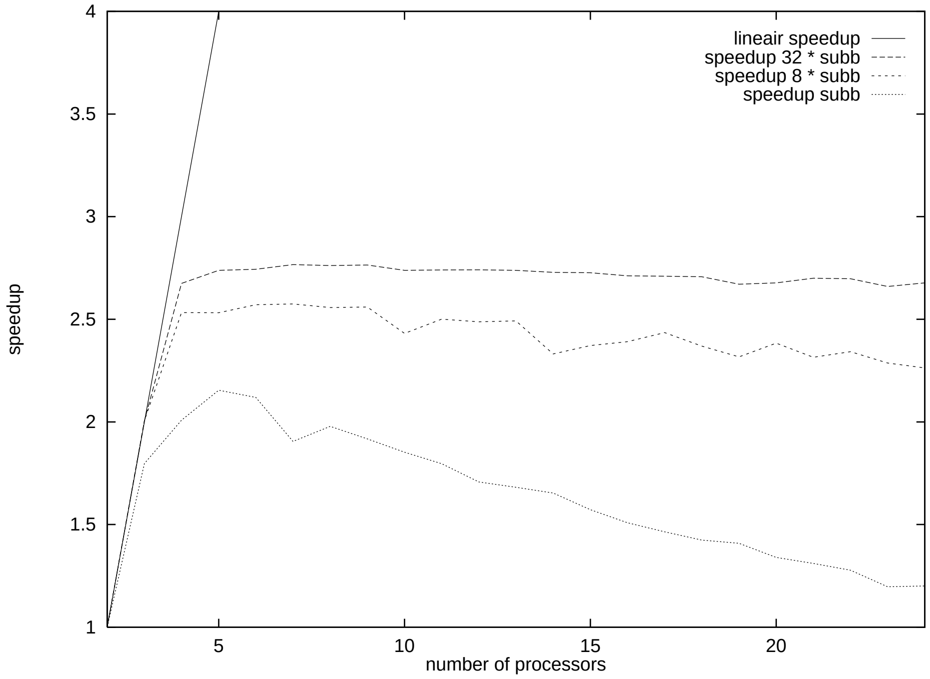


Figure 6.4: Speedup *subb* images

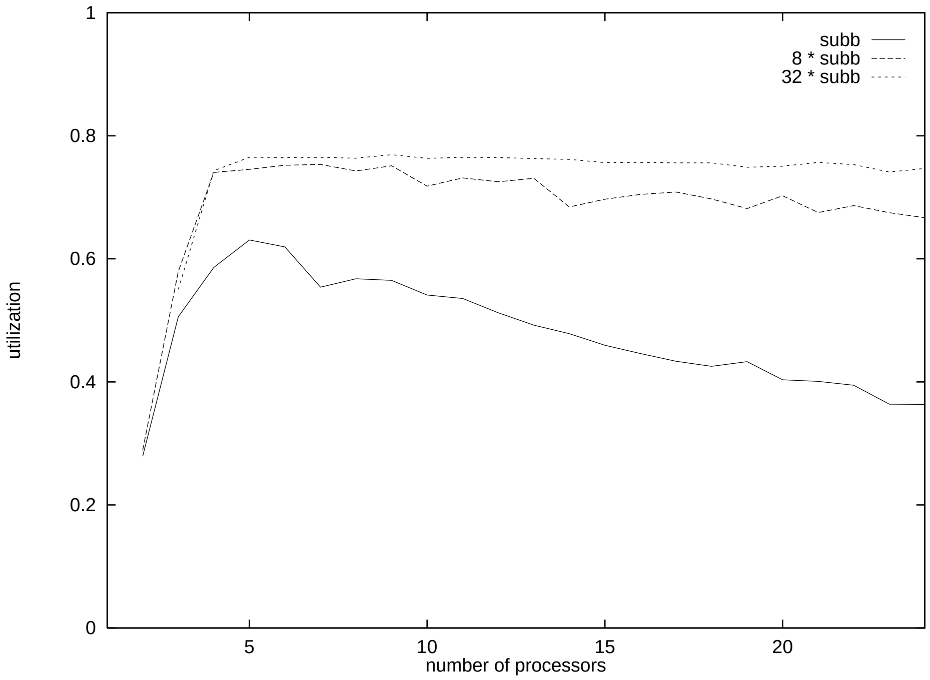


Figure 6.5: Master process utilization *subb* images

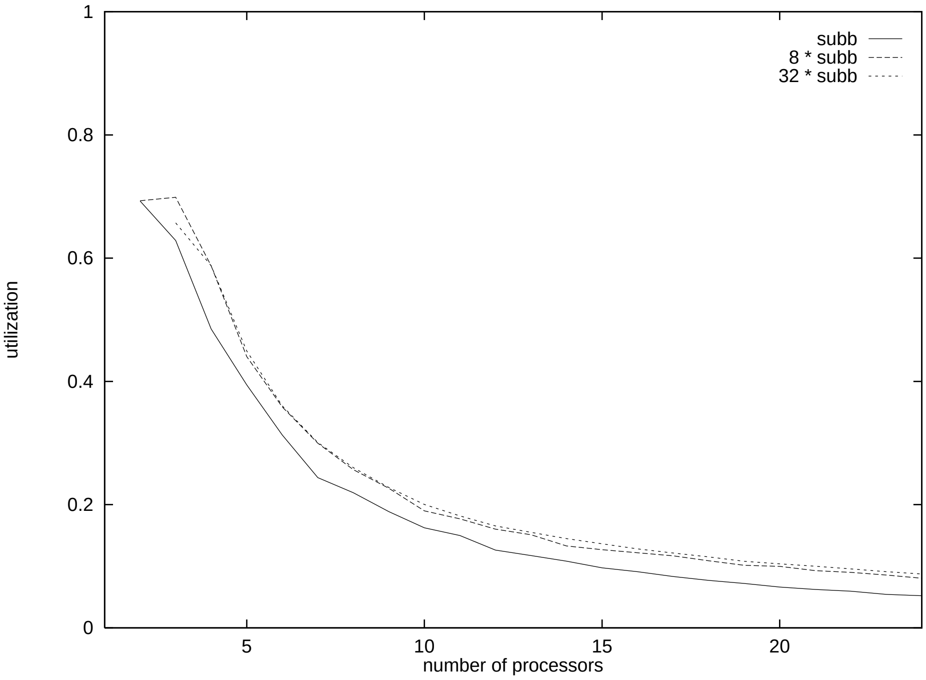


Figure 6.6: Slave process utilization *subb* images

the *subb* images have a decreasing speedup when more than approximately 5 processors are involved due to the trade off between intense communication, as result of many small regions, and the involved data. This can also be seen in the figures 6.5 and 6.6 where the latter figure shows a lesser utilization compared to the average slave utilization of the suba images represented in figure 6.3.

The scalability of the image with many small region isn't very good because the limits of the underlying network.

In figure 6.7 the speedup is shown of the two largest images with small and large internal regions.

A preliminary conclusion can be that the speedup stays stable after approximately 7 processors involved. Again it is shown that the utilization of the master process increases too fast in order to get a reliable speedup, while the slave utilization decreases.

Appendix B.1, figures B.7, B.8, B.9, B.10, B.11 and B.12, represents the other measurements on the image with small internal regions.

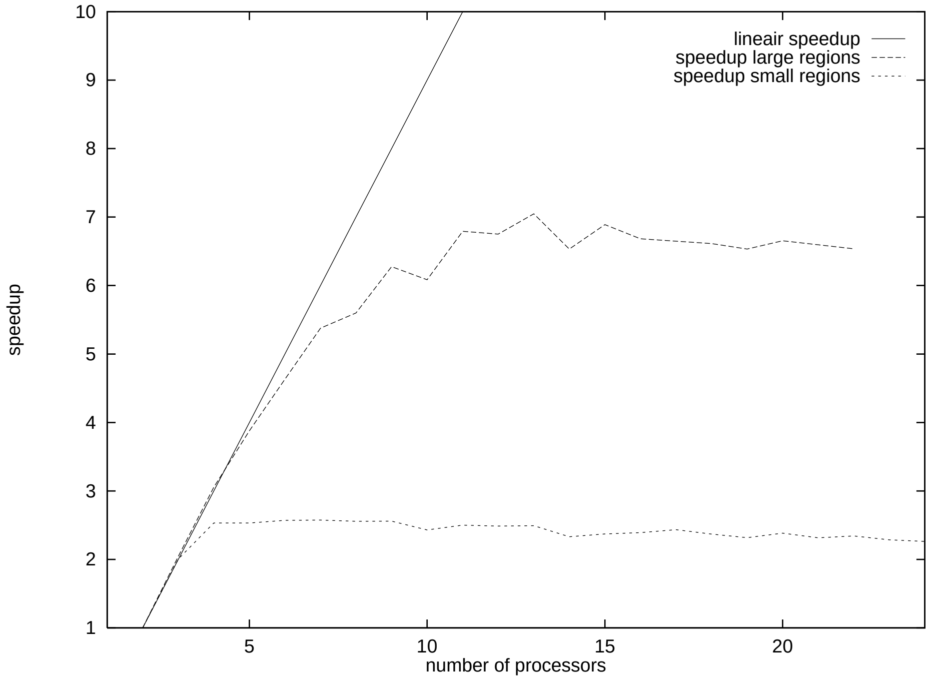


Figure 6.7: Speedup comparison *8 * suba* & *8 * subb*

6.2.3 Work packet size analyses: *suba* & *subb*

This section shows the influence of the work packed size on the performance. The result is shown figure 6.8, representing measurements on 32 times *suba* and 32 times *subb*. The used multi computer is configured of 7 processors. The resulting algorithm configuration is 6 slave processes, all on a different processor and a master provider and receiver process, situated at one processor.

In order to show the influence of the work packet size on the performance a variation on the speedup representation is used. During these tests the number of processors remains constantly 7, the scalability of the speedup is now calculated as the ratio:

$$S_{cells} = \frac{T_{5000}}{T_{cells}}$$

, where $500 \leq cells \leq 10000$. The speedup for S_{5000} becomes becomes 1, this is chosen because the tests described in the previous section where held with a constant cell size of 5000.

Figure 6.8 shows the results of the work packet size on the speedup for the largest images with large and small internal regions. It is shown that indeed the cell size has some influence on the speedup. A recommendation is

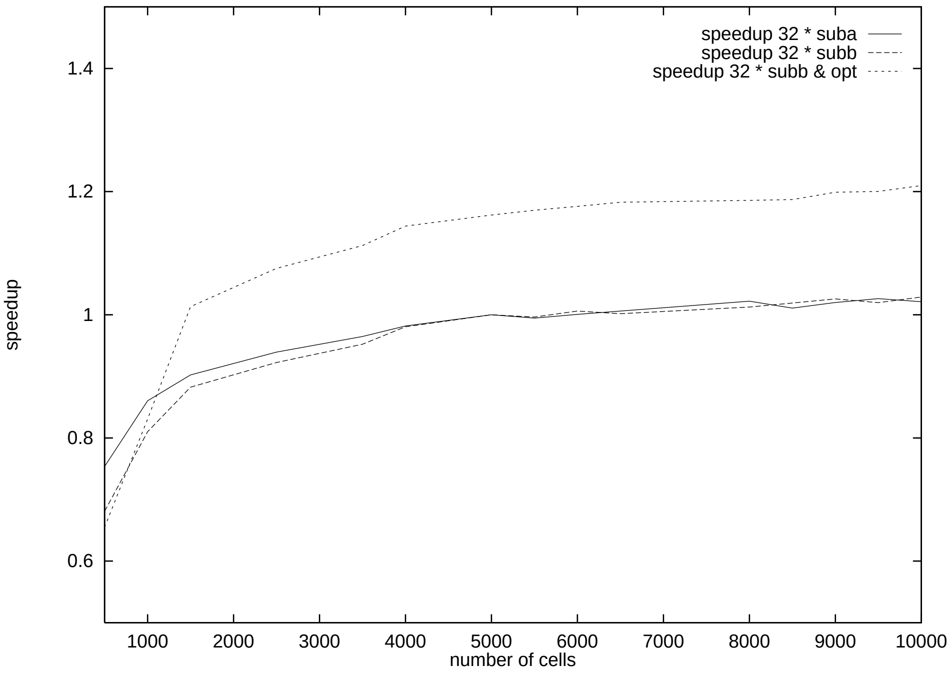


Figure 6.8: Speedup varying cell size

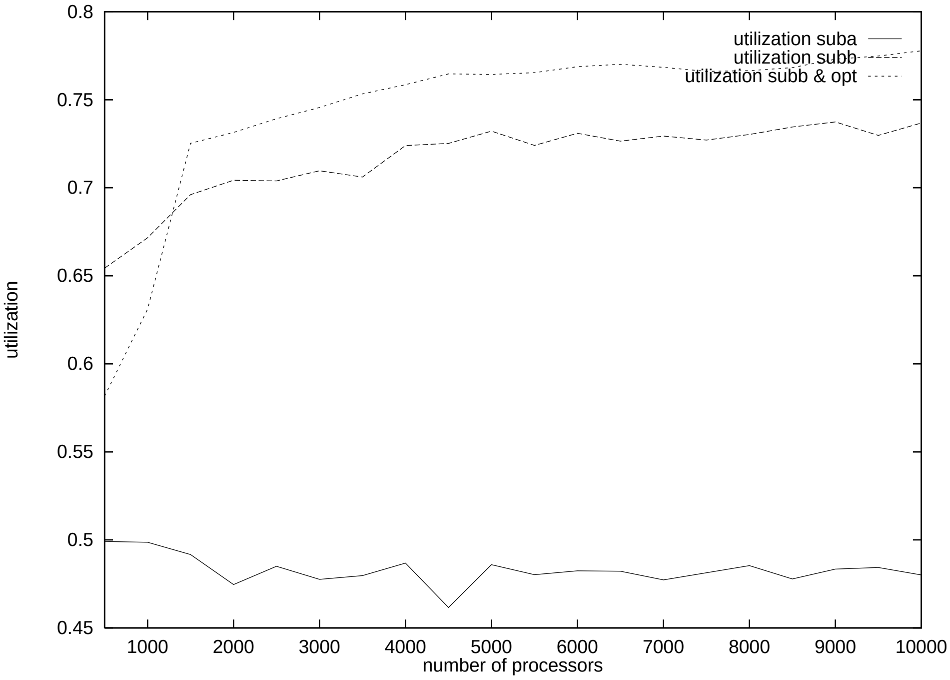


Figure 6.9: Master process utilization varying cell size

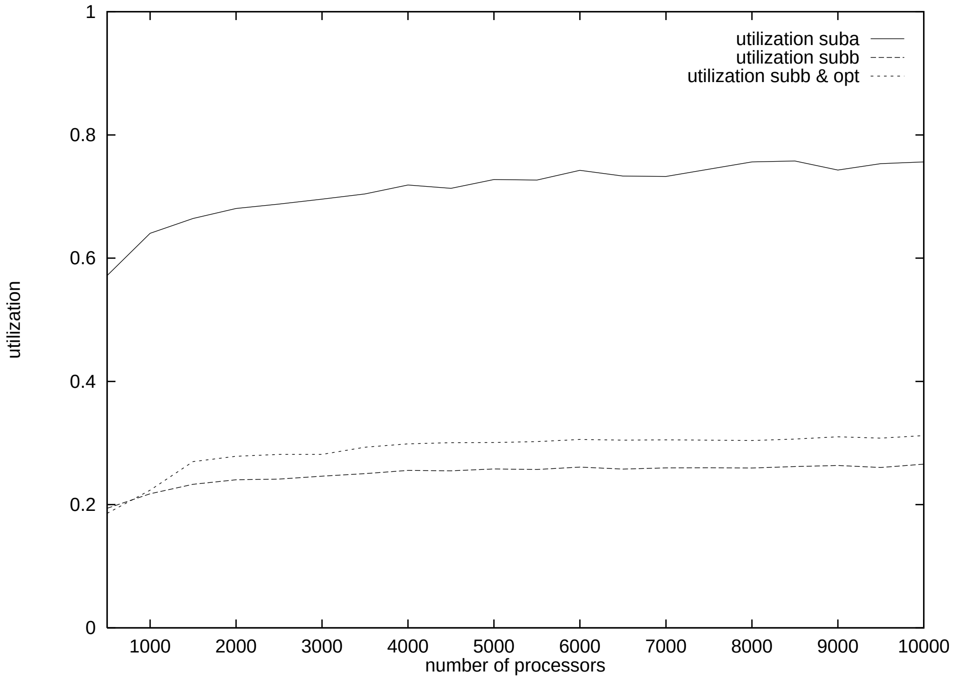


Figure 6.10: Slave process utilization varying cell size

to set the cell size not below 4000 cells per slave because it reduce speedup significantly.

It is also shown that the speedup of suba and *subb* is almost identical but the master process utilization of suba is very low where *subb* is high and the opposite occurs with the slave processor utilization.

In appendix B.2 the other measurement results are shown. The utilization scores of the subimages are stated in figures B.13, B.15, B.17, B.19, B.21 and B.23. The response time scores are shown in figures B.14, B.16, B.18, B.20, B.22 and B.24,.

6.2.4 Influence of the optimization option

The *lamboot* program, for starting up the multi computer, as mentioned in section 4.1.2 and 2.4, provides a parameter to indicate a homogeneous multi computer, which practically means no data conversion before data exchange between nodes in the multi computer takes place. This will also lead to performance increasing. For the evaluation the same test is used as described in 6.2.3.

The influence of the optimization was tested on the 32 times *subb* image, the results are also shown in figure 6.8 because this figure also represents the results with the optimization parameter set. This figure shows a significant influence of the parameter on the speedup.

Figures 6.9 and 6.10 show the influence of work packet size on the utilization of the master and slave respectively. The master process utilization is significantly higher compared to the master process utilization without the parameter set because the master doesn't have to pack the data and it doesn't have any influence on the average utilization of the slave processes.

6.3 Analysis of composed images

In this section the influence of the providing order of image data is analysed. This is done by composing images in some order. The results could lead to an integration in the algorithm. The idea is to provide large regions early in the computation which might lead to response time reduction.

In order to compose new lan files a utility program has been written which is used as follows:

```
complan <file1> <file2> <dfactor>.
```

It multiplies the files `file1` and `file2` with `dfactor` where after they are appended in one new file. In appendix A, figure A.11 the resulting image is shown for:

```
complan sube.lan suba.lan 2.
```

6.3.1 Analyses of composition of *suba* and *subb*

Appendix A figure A.11 shows the image which is composed of 2 times *suba* followed by 2 times *subb*. It is expected that this image has a better performance, in measures of response time, compared to the opposite image (*subb* followed by *suba*). This is based on the expectation that providing slaves earlier with large regions would reduce the response time. Figure 6.11 displays the comparison of four tests on the two images, while the image data remains the same. From this figure is concluded the order isn't significantly influencing the performance. Also providing more cells, the work packet size of cells was set to 8000, per slave hasn't got as well any influence on the performance.

This is probably the result of a too low response time to perform the algorithm on 2 times *subb* and 2 times *suba*. It will be necessary to evaluate a larger composed image of 4 times *suba* and 4 times *subb*.

In appendix B.3, figure B.25 the response time is shown of a larger composition of *suba* and *subb*. Again the order of providing shows no significant

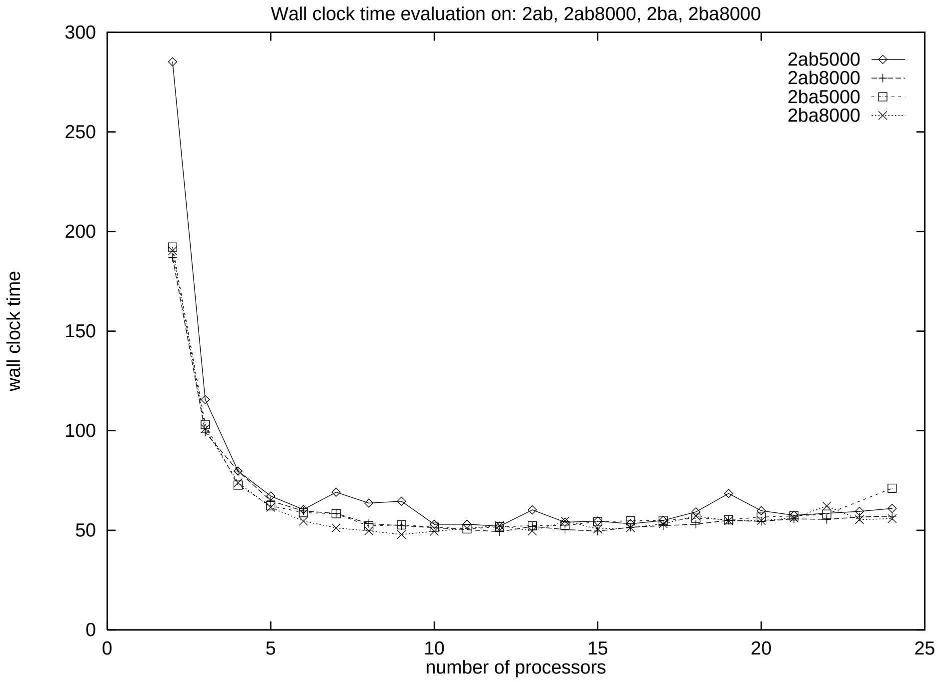


Figure 6.11: Response times on 2 × *suba* and 2 × *subb* and 2 × *subb* and 2 × *suba*, with 5000 and 8000 cells respectively

influence. It is concluded that the regions are too small in order to show that providing the regions in some sorted order could have any significant influence on the response time.

Due to the fact that the response time figure doesn't show any difference the speedup hasn't been calculated for the image composed of *suba* and *subb*.

So in order to show the influence a new subimage *sube* was created, with one big region. The image *sube* is shown in appendix A, figure A.10.

6.3.2 Analyses of composition of *sube* and *suba*

To prove significant influence of the order of providing on the speedup of the algorithm, a new sub image (*sube*) is created. The image *sube* is shown in figure A.10 and described in section 1.1. Two compositions are made, one consisting of 2 times *suba* and 2 times *sube* and one consisting of 2 times *sube* followed by 2 times *suba*. The results on the speed up is shown in figure 6.12, the speedup accomplished by providing first the *sube* image before *suba* speeds up the computation by an extra factor of a half. The work packet size was set to 8000 cells which is nearly optimal concluded from the results described in section 6.2.3. Figure 6.13 shows the increasing utilization of the

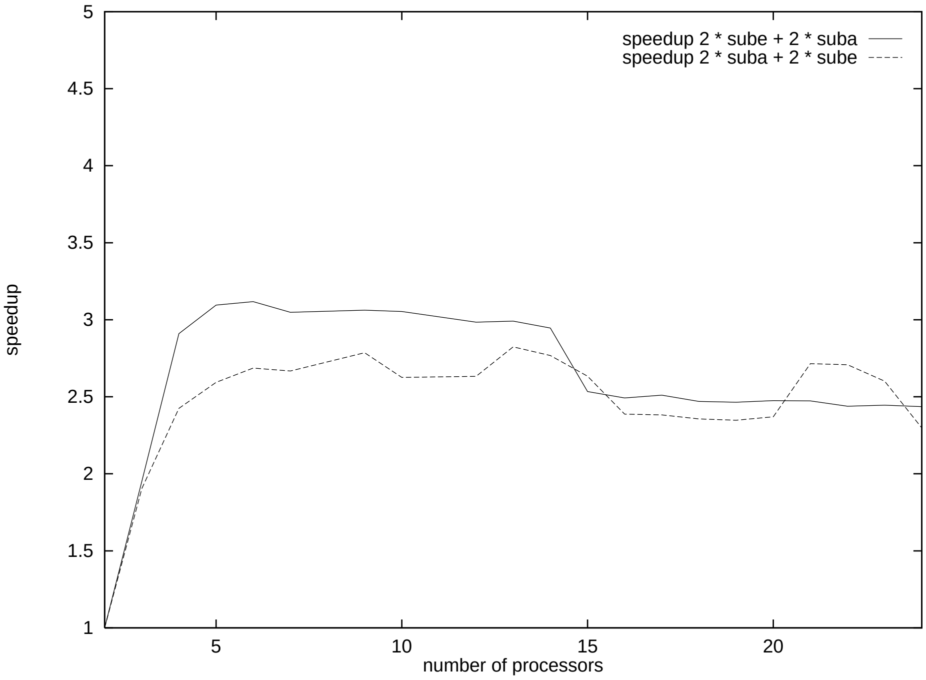


Figure 6.12: Speedup *suba* & *sube*, with cell size 8000

slave processes but also the slave utilization of *sube* followed by *suba* is higher than the opposite. This explains the speed up.

Because figure 6.14 shows no difference in utilization for the master processes.

The conclusion is that the order providing is influencing the speedup but one has to be careful when the difference between the regions is too small in terms of size, the results are not overwhelming.

Appendix B.3 represents the other measurements in figures, B.26, B.27, B.28 and B.29.

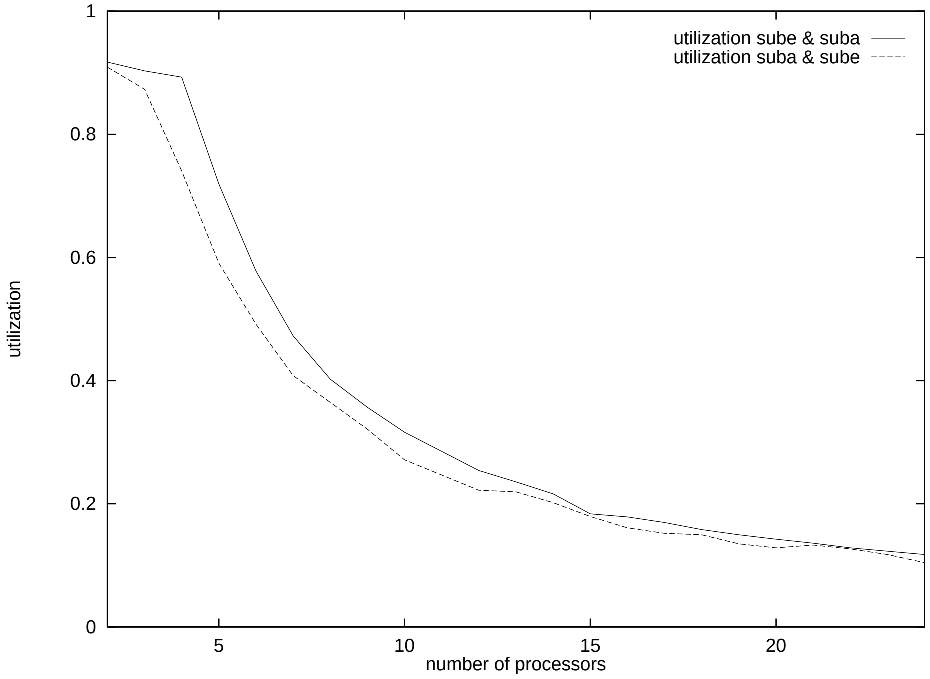


Figure 6.13: Utilization of slave processes *suba* & *sube*

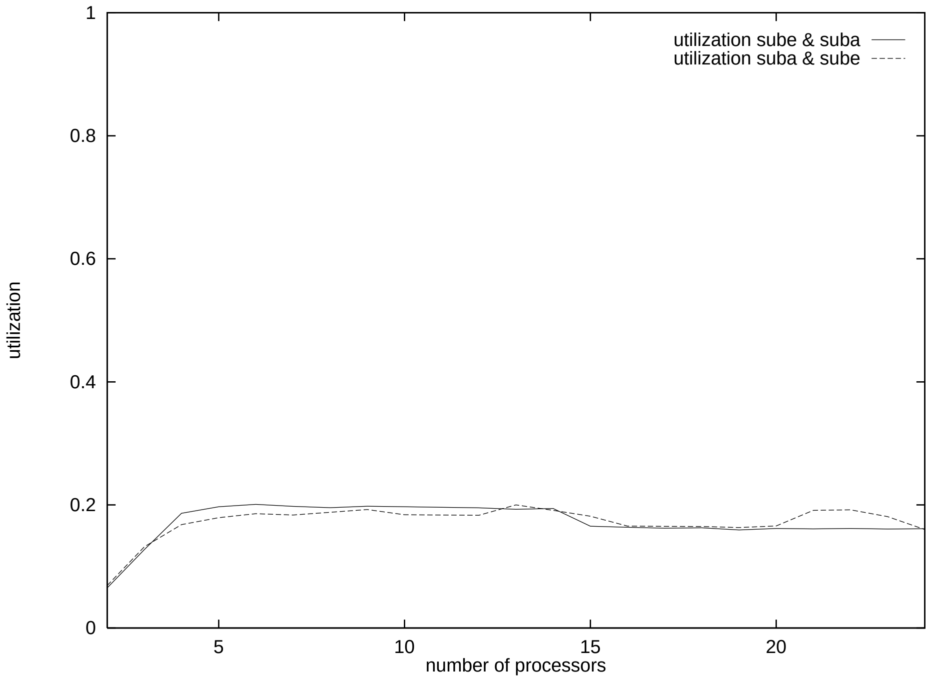


Figure 6.14: Utilization of master processes *suba* & *sube*

Chapter 7

Conclusions and Further Research

7.1 Conclusions

LAM MPI The use of the LAM MPI implementations shows little efficiency when more than approximately 5–7 processors are involved in the computation. The multi computer has shown bad scalability. However it is expected that the used environment evolves further, also with the expected outcome of the MPI2 definitions. Compared to PVM it is less efficient and there are lesser literature references.

The LAM environment wasn't used in this research to migrate a PVM program to a MPI program because PVM shown less portability indicating nodes.

MPI The portable MPI implementations are each others concurrencies in the area of efficient use of memory and network. This increases the performance of the different implementations.

More vendor implementations become available so MPI will be used in more parallel implementations.

During the use of MPI in this research the forwarded MPI goals, as described in section 2.1.1 have been dealt with. The portable implementations in fact realizing the use of MPI on a wide range of computers. Also the portability is guaranteed.

However there are many evolutions in MPI, MPI2, vendor implementations and more algorithms makes use of portable MPI implementations.

Adaption The run time adaption of the multi computer won't show performance increasing as a result of the low scalability factor.

The sorting of the order of providing will show no performance increasing in reality because it is only of use when really large regions are provided. Such large regions belong usually to homogeneous regions such as water and will be no subject to region growing.

The work packet size is a useful parameter together with type optimization (due to used MPI implementation).

7.2 Further research

When developing a transputer implementation it is necessary to do some research in the available implementations for such multi computer systems. More research should take place in the area of dynamically adapting the work environment, when MPI2 is available, following the original PVM implementation of best merge and the now available implementation in LAM MPI and the expectation that the future LAM MPI will show more scalability and efficiency.

Before developing an algorithm for region growing on the whole image some extra research about the topology of the processes should take place.

When an MPI2 implementation becomes available a better performance comparison, e.g. with PVM, could be made because then all processes could be started from the master process.

Appendix A

Images Used

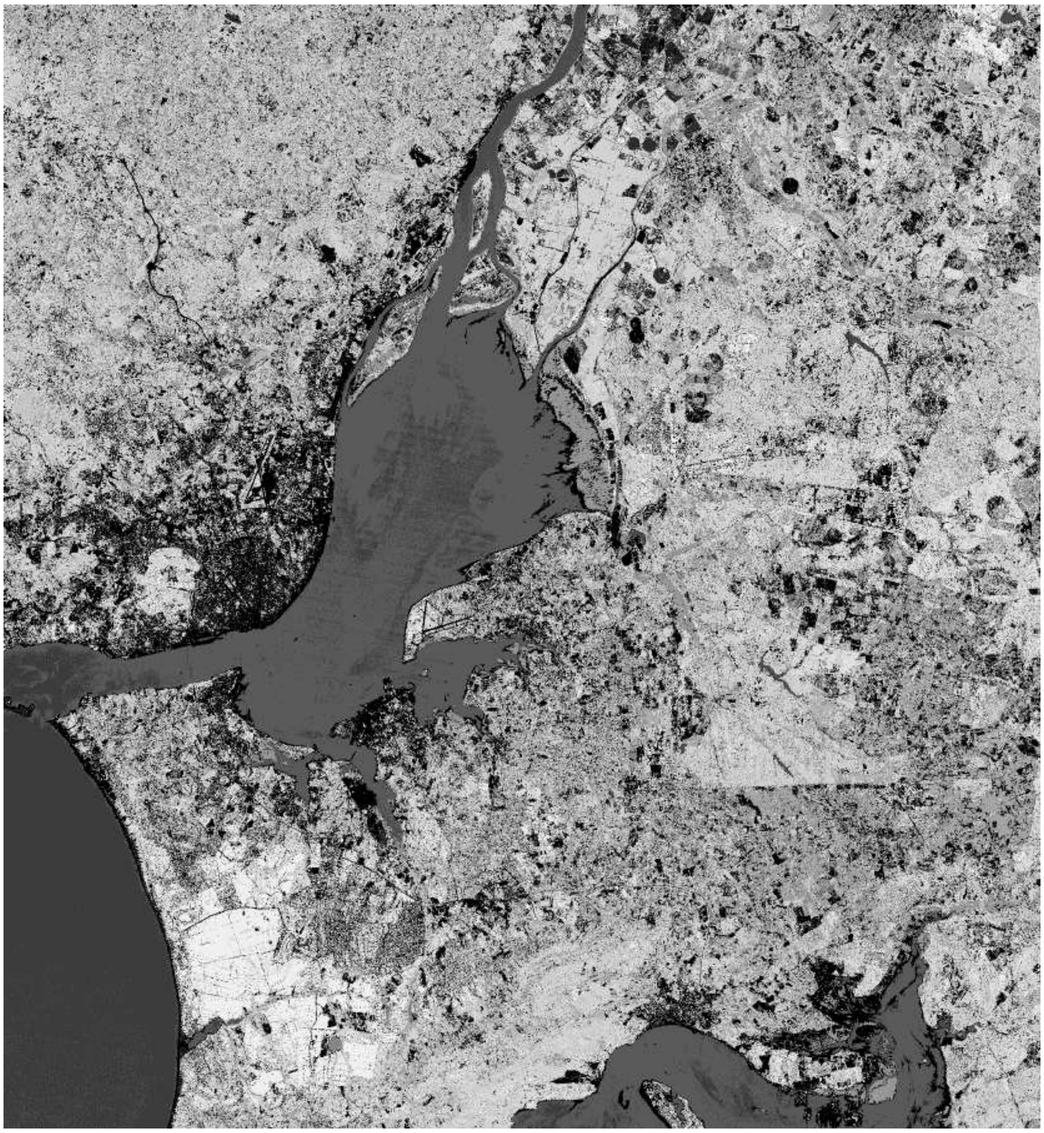


Figure A.1: Subjune image provided by the LANDSAT-TM satellite

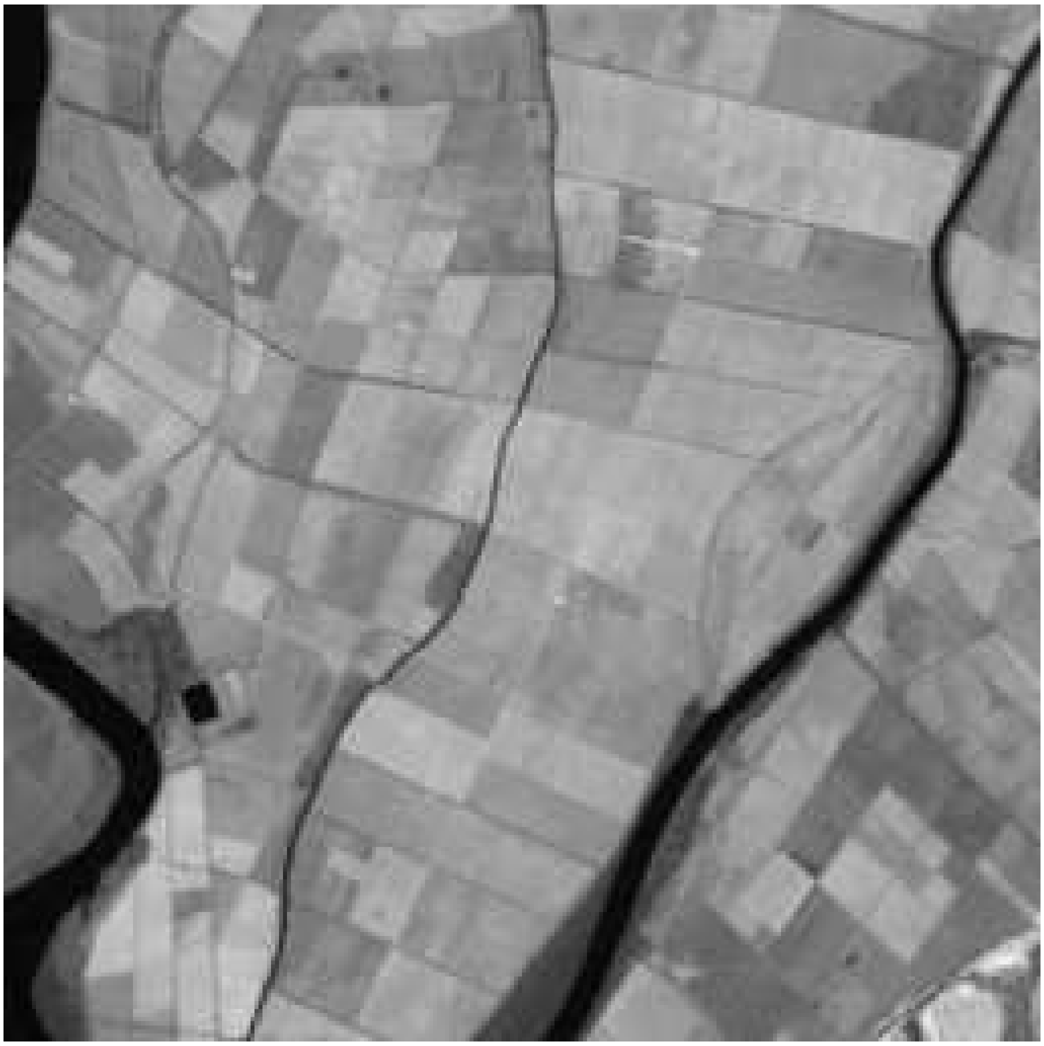


Figure A.2: Image with large internal regions *suba, band 5*

Figure A.3: Suba after edge detection and polygon construction

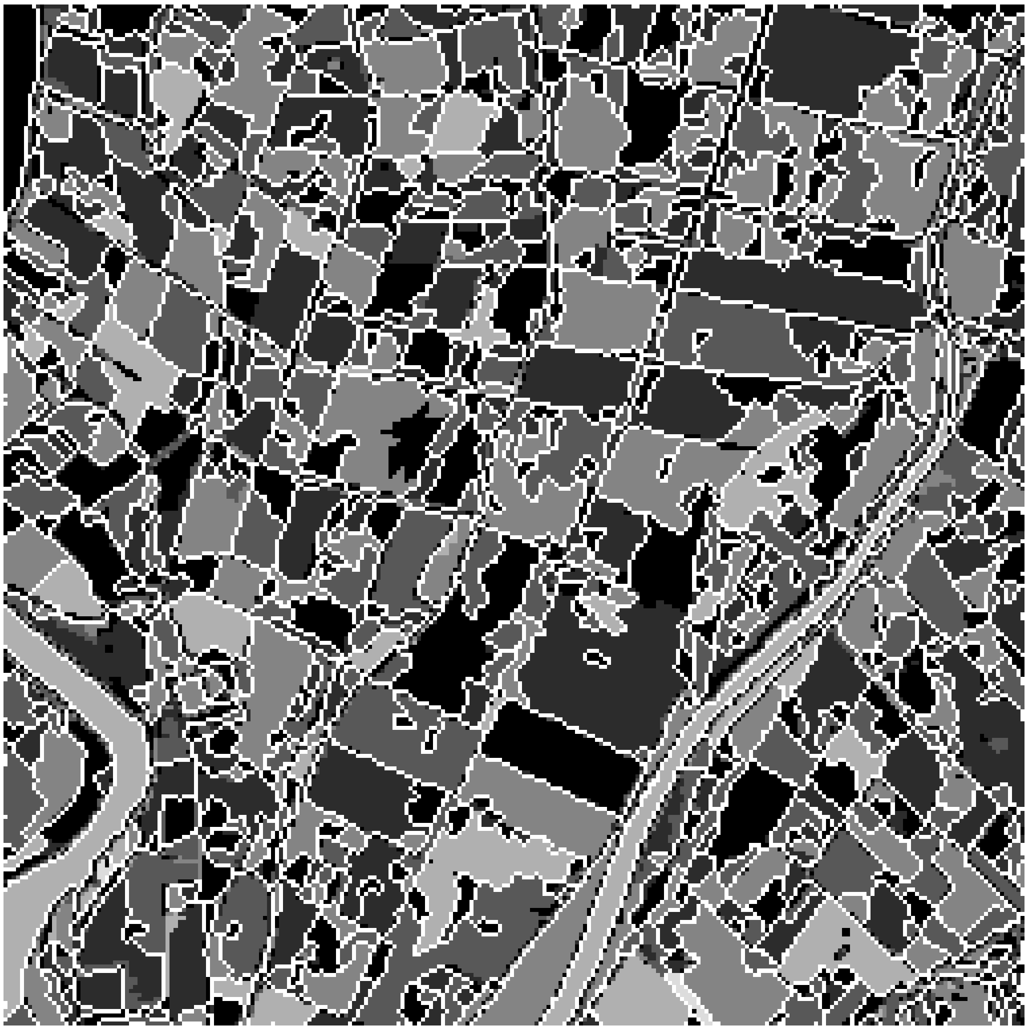


Figure A.4: Image with large internal regions after *best merging* per polygon, with threshold set to 8

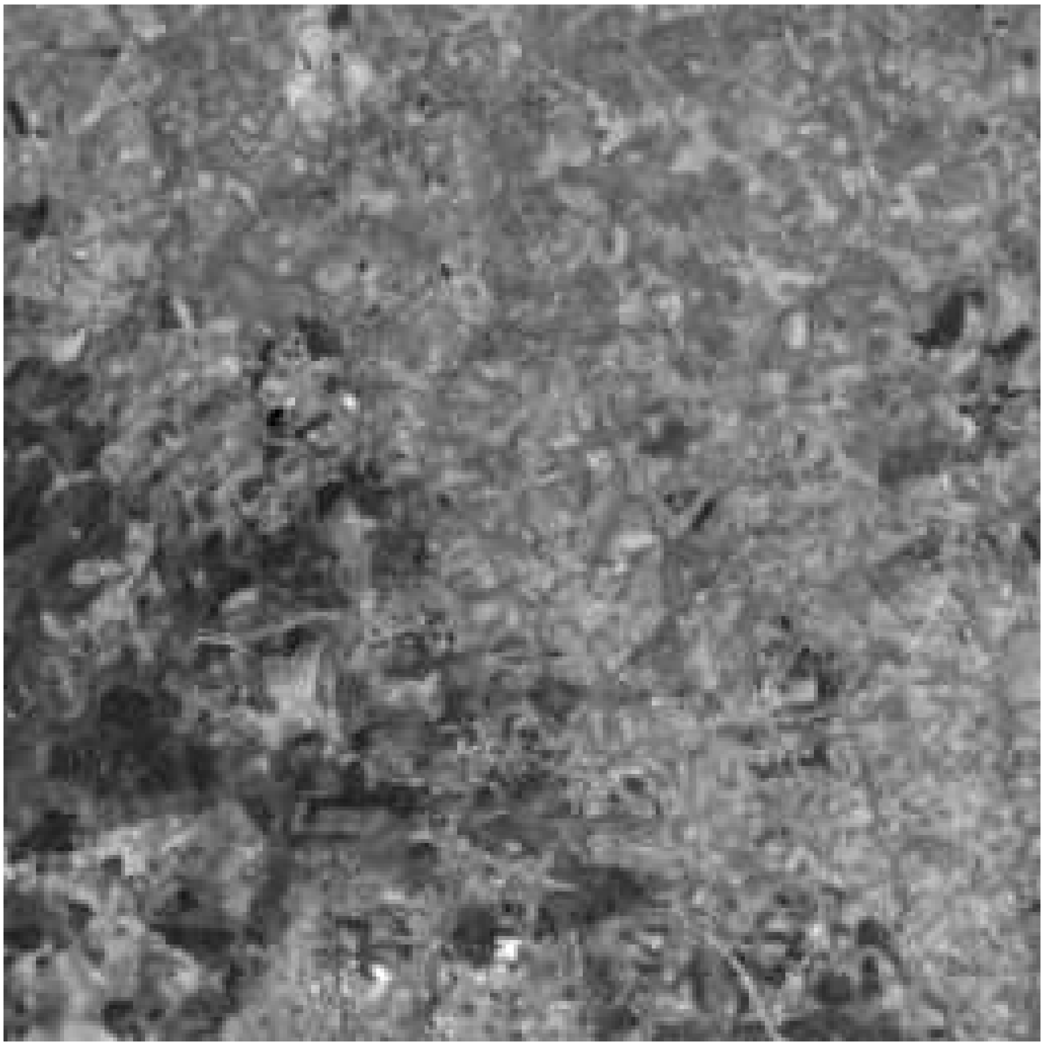


Figure A.5: Image with small internal regions *subb, band 5*

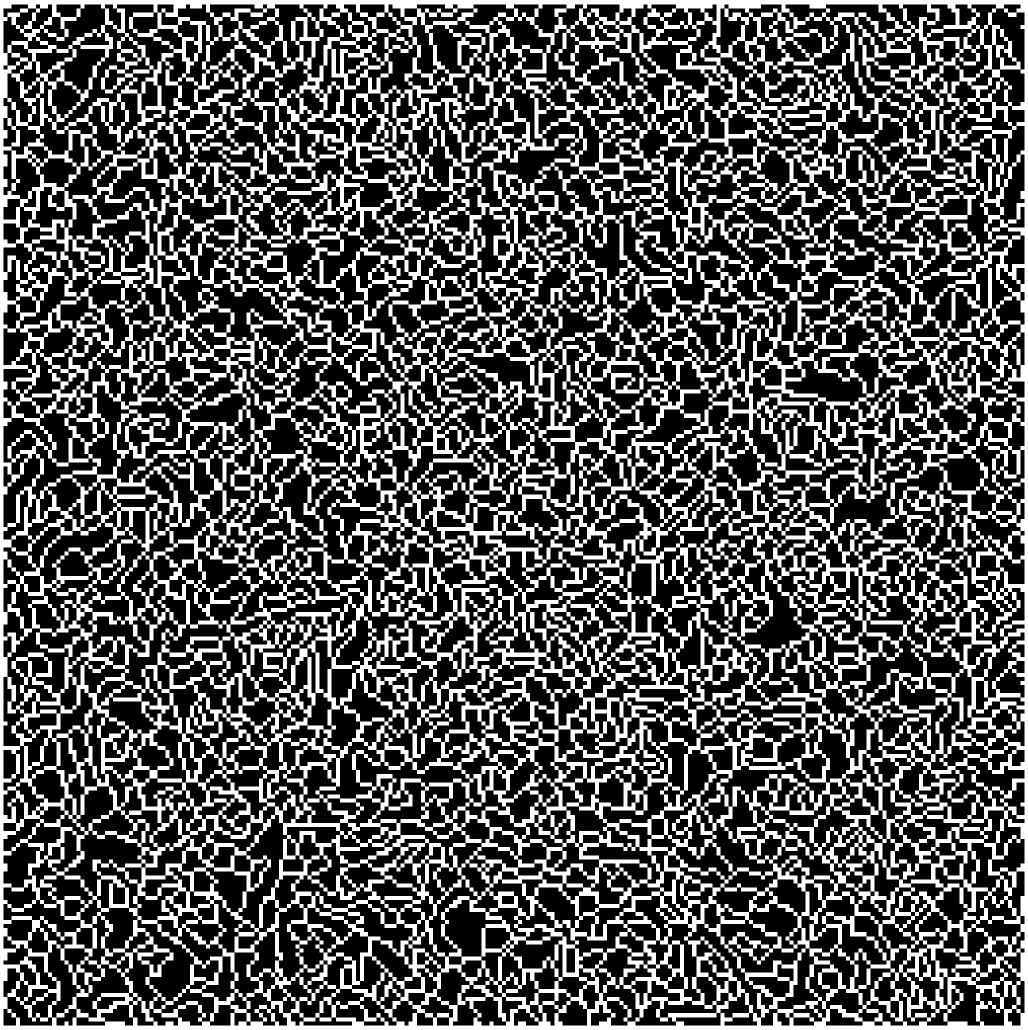


Figure A.6: Subb after edge detection and polygon construction

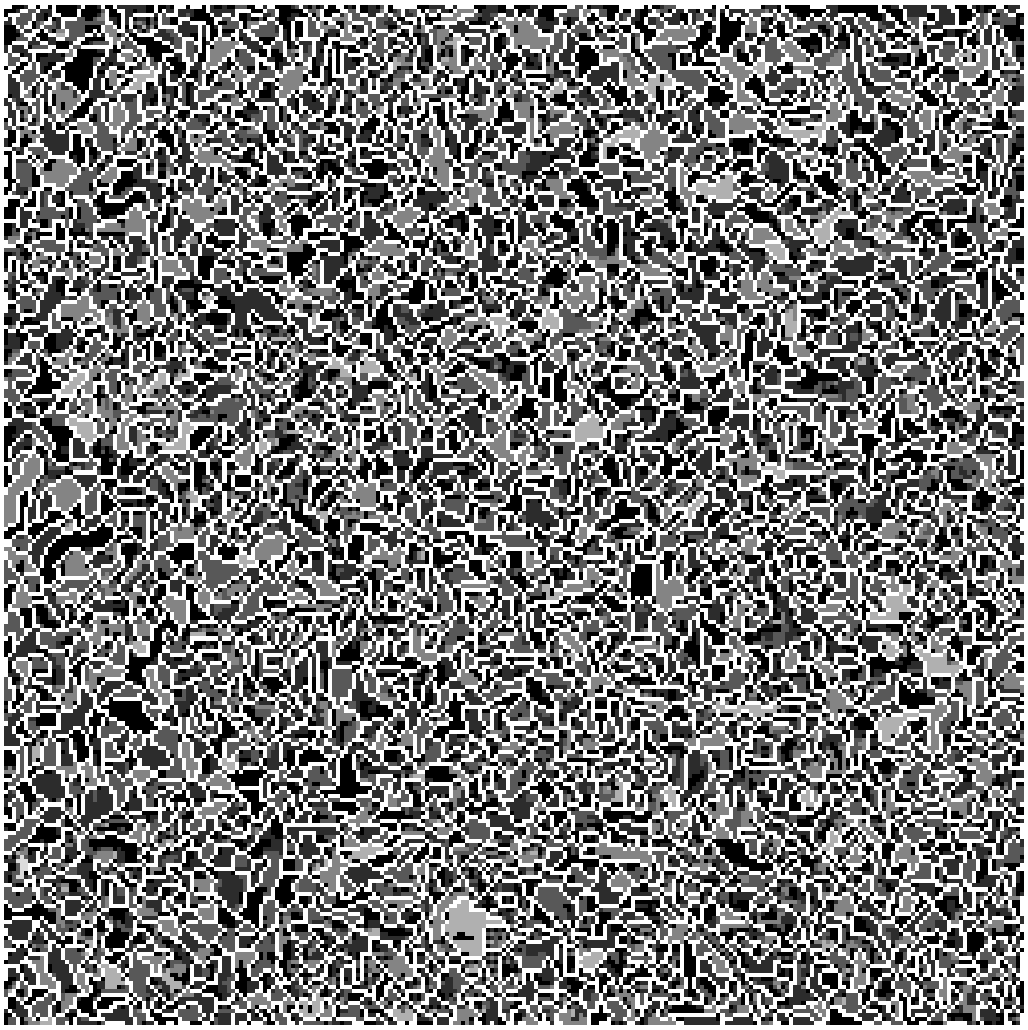


Figure A.7: Image with small internal regions after *best merging* per polygon, with threshold set to 8

Figure A.8: Image with one big internal region *sube, band 5*

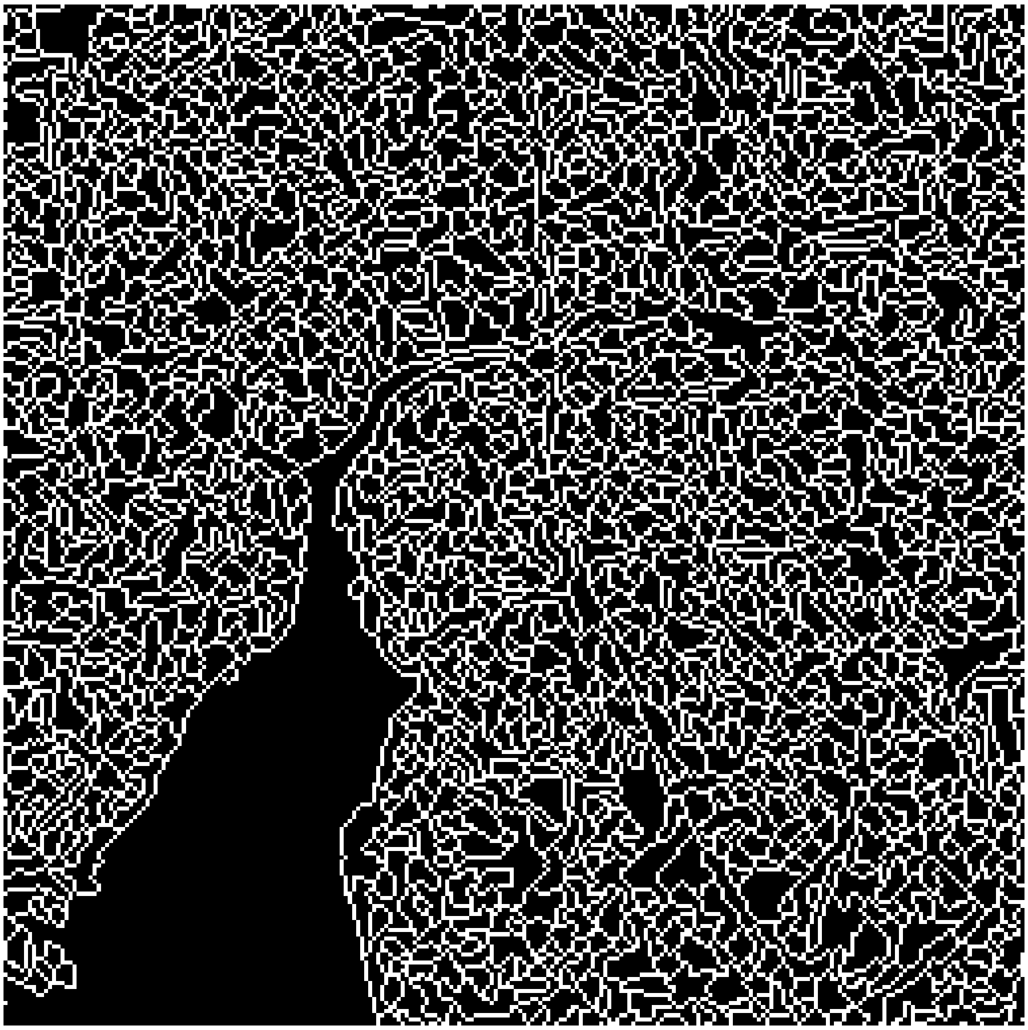


Figure A.9: Sube after edge detection and polygon construction

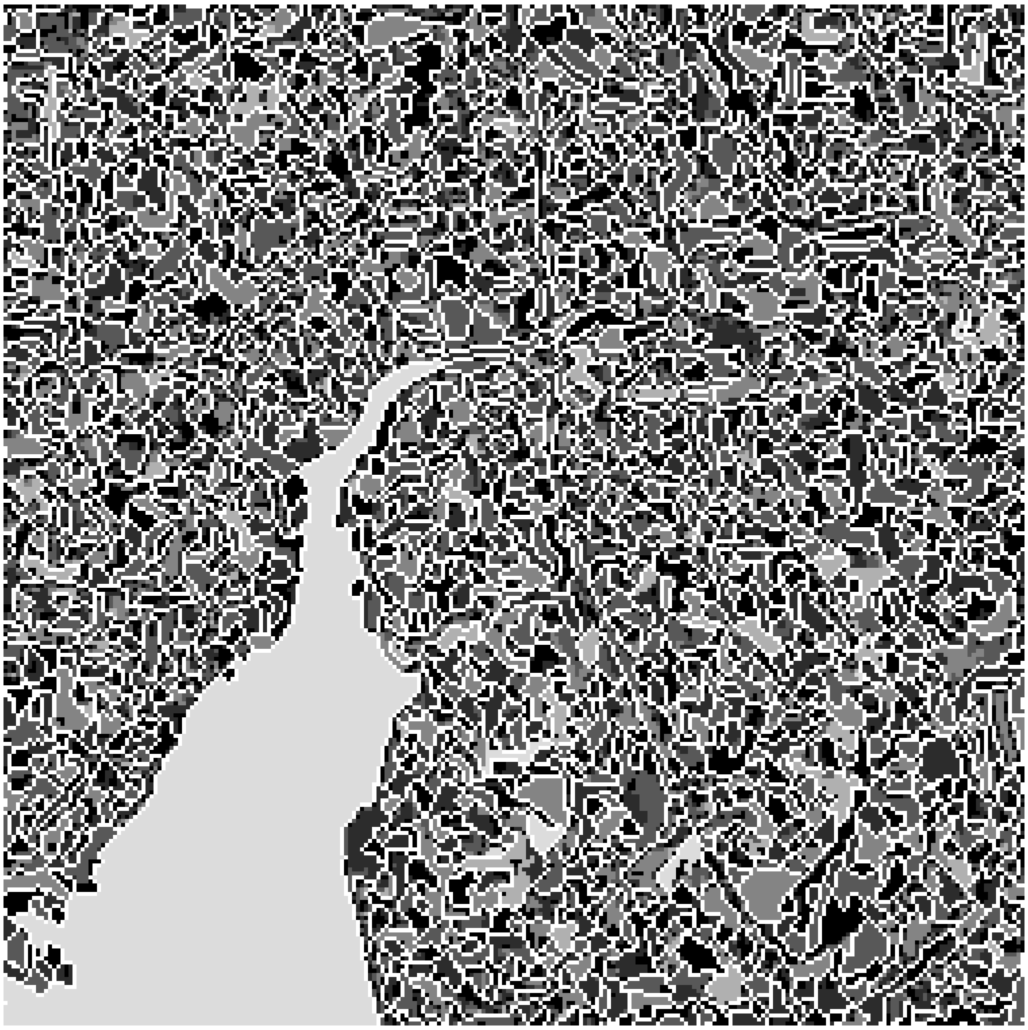


Figure A.10: Image with one big internal region after *best merging* per polygon, with threshold set to 8

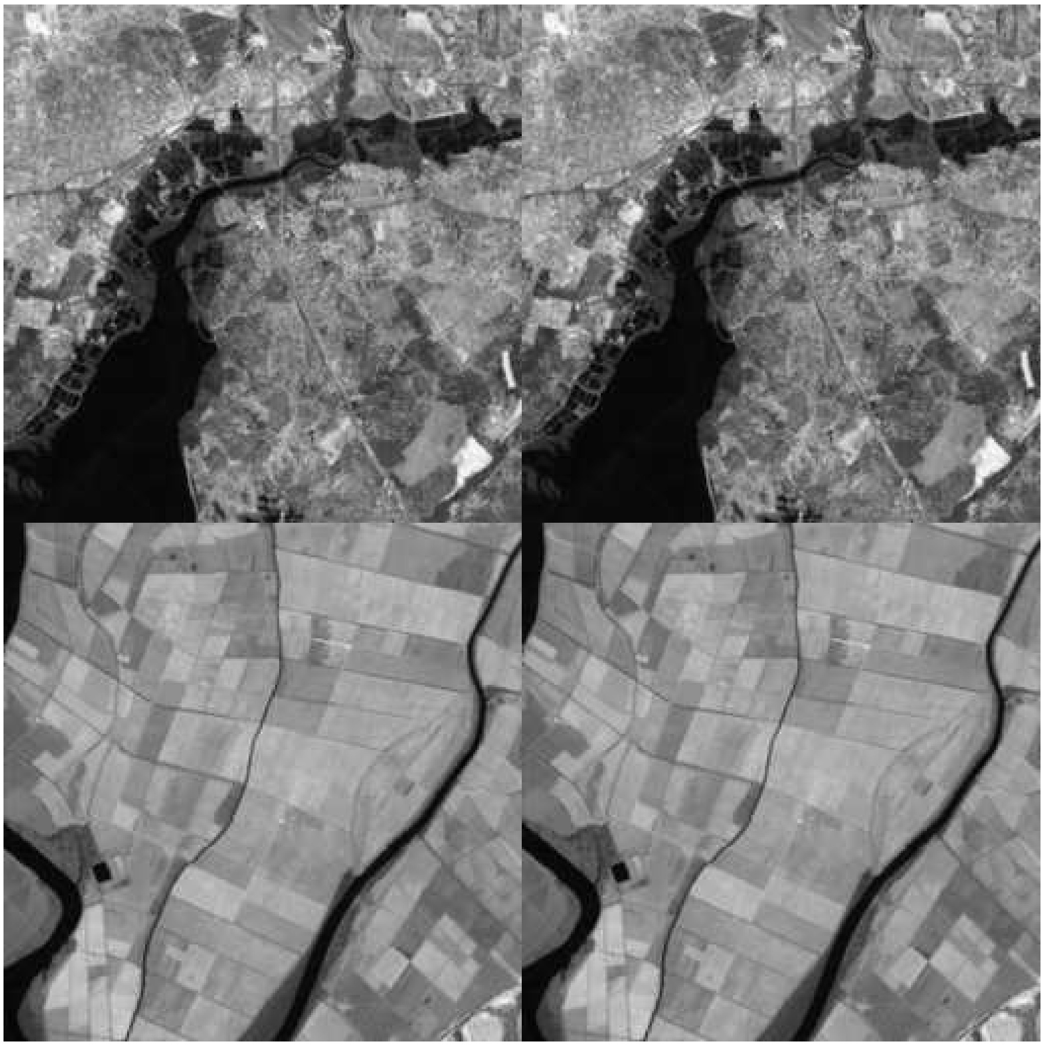


Figure A.11: Composition of 2 × sube and 2 × suba

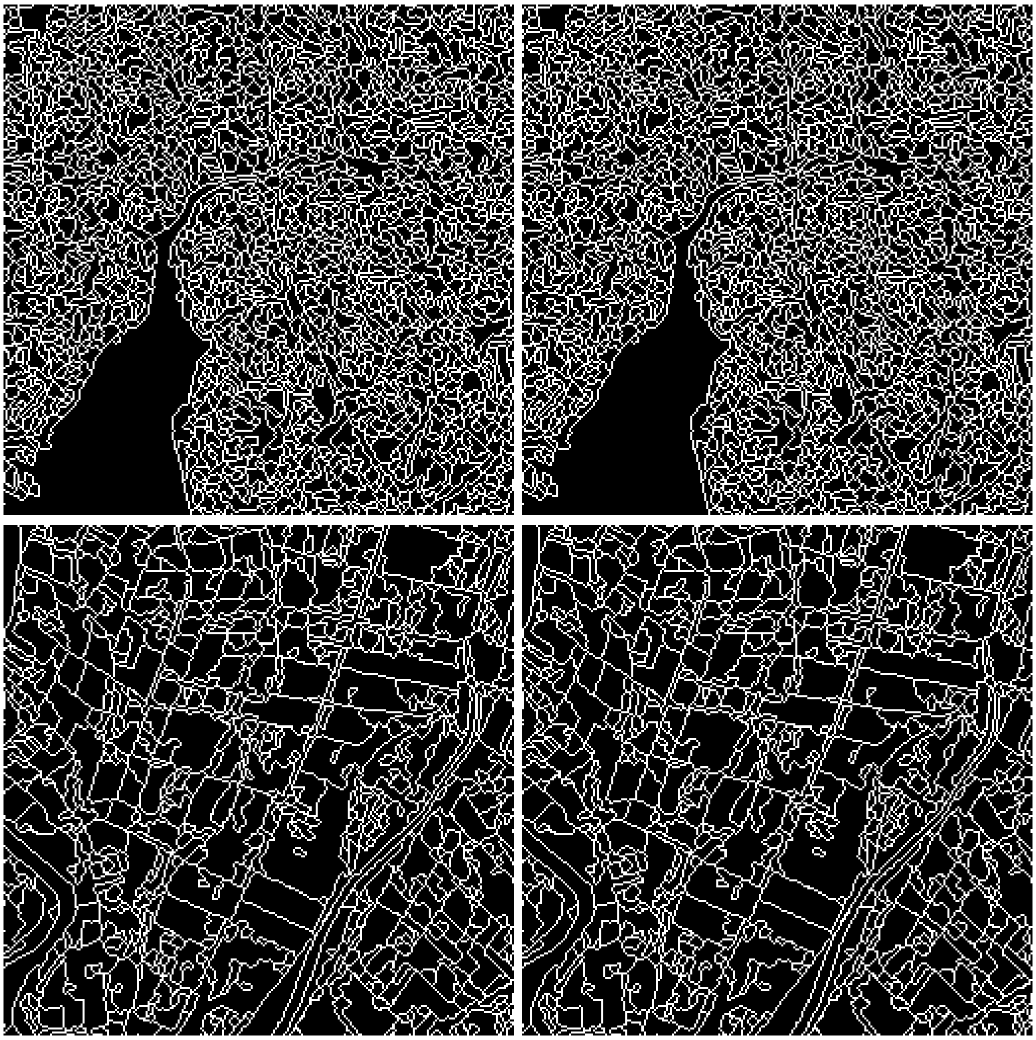


Figure A.12: Composition of 2 $\times$ sube and 2 $\times$ suba, after edge detection and polygon construction

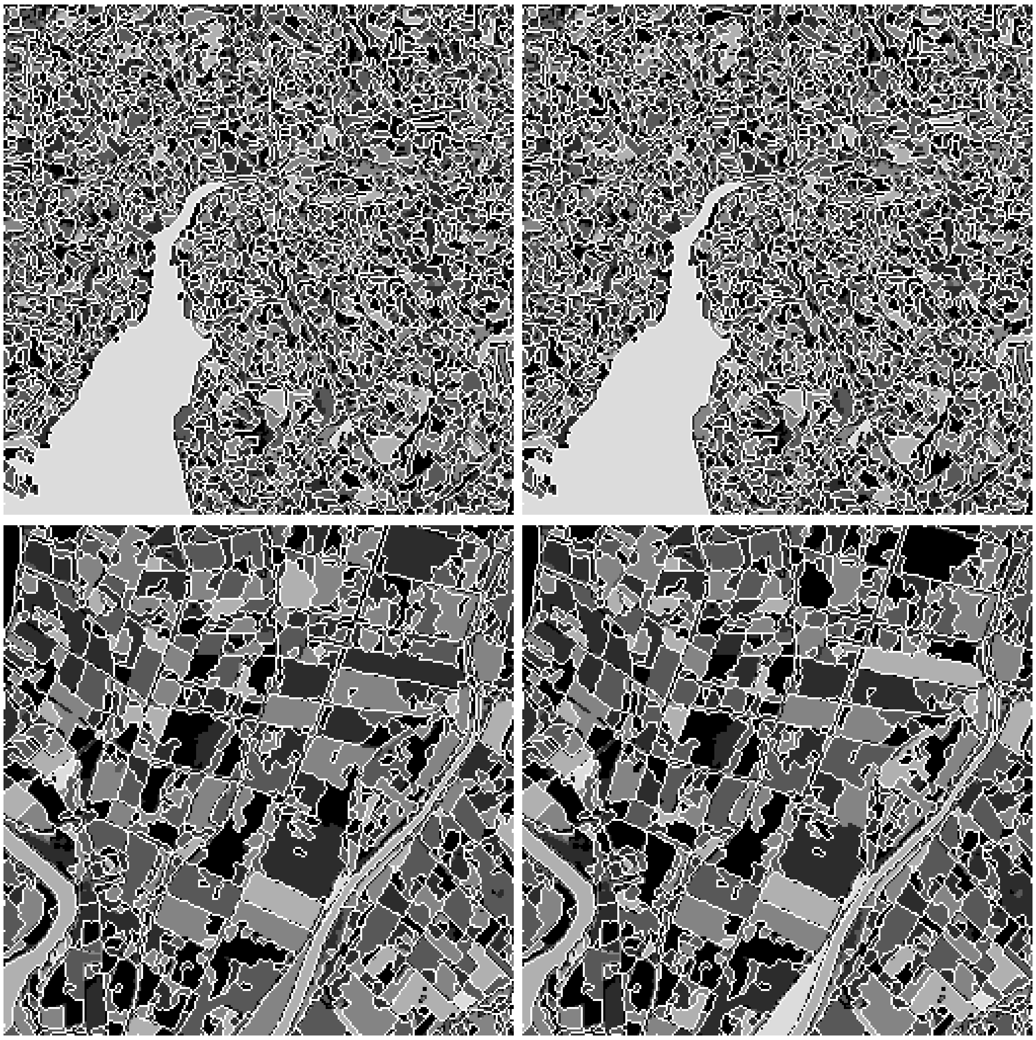


Figure A.13: Composition of 2 × sube and 2 × suba after *best merging* per polygon, with threshold set to 8

Appendix B

All Graphs

B.1 Host graphs

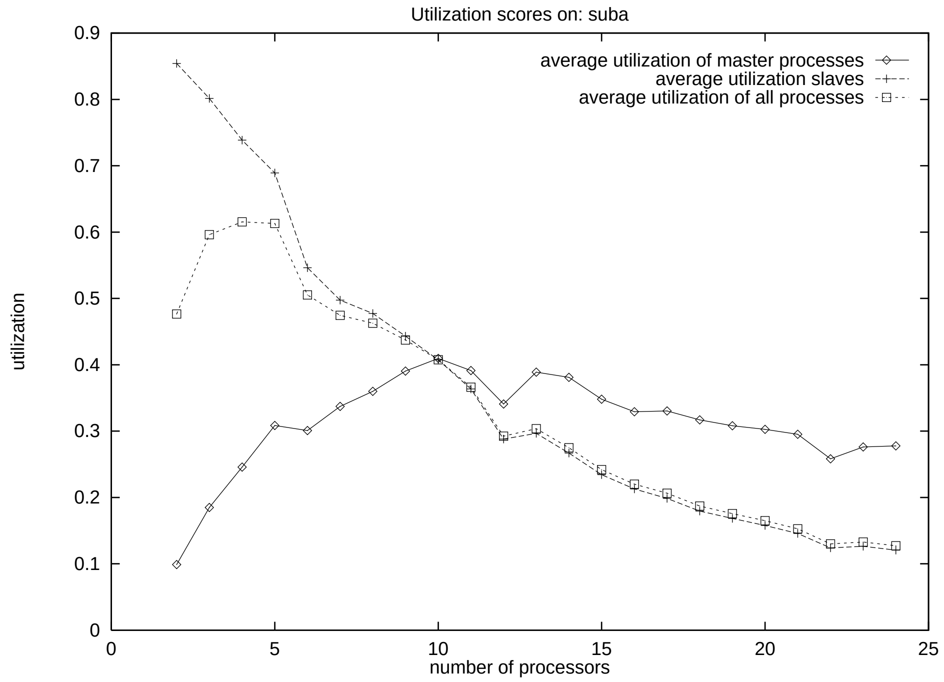


Figure B.1: Utilization scores on suba

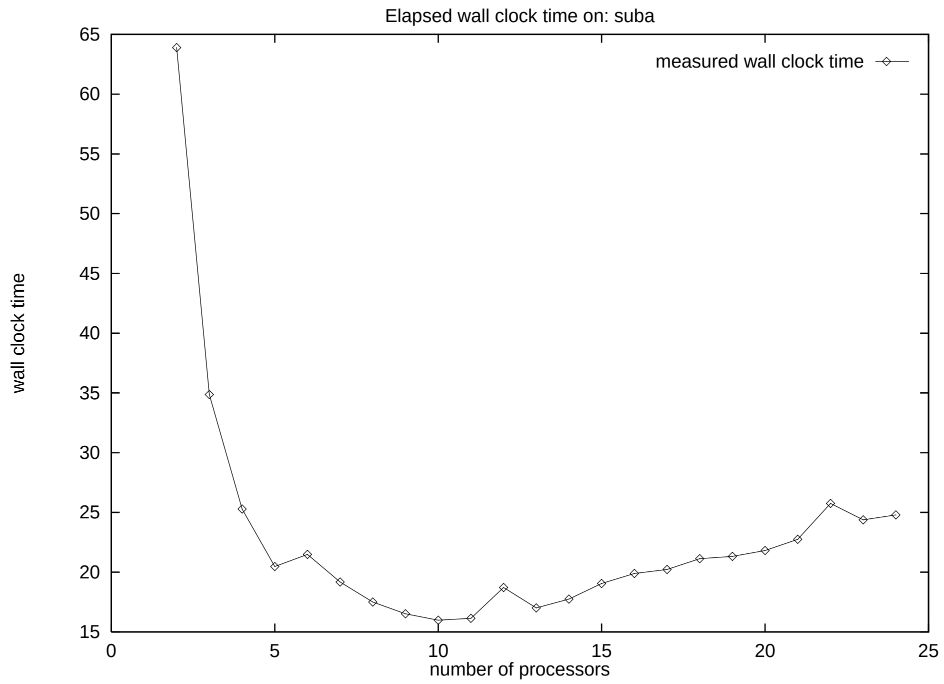


Figure B.2: Response times on suba

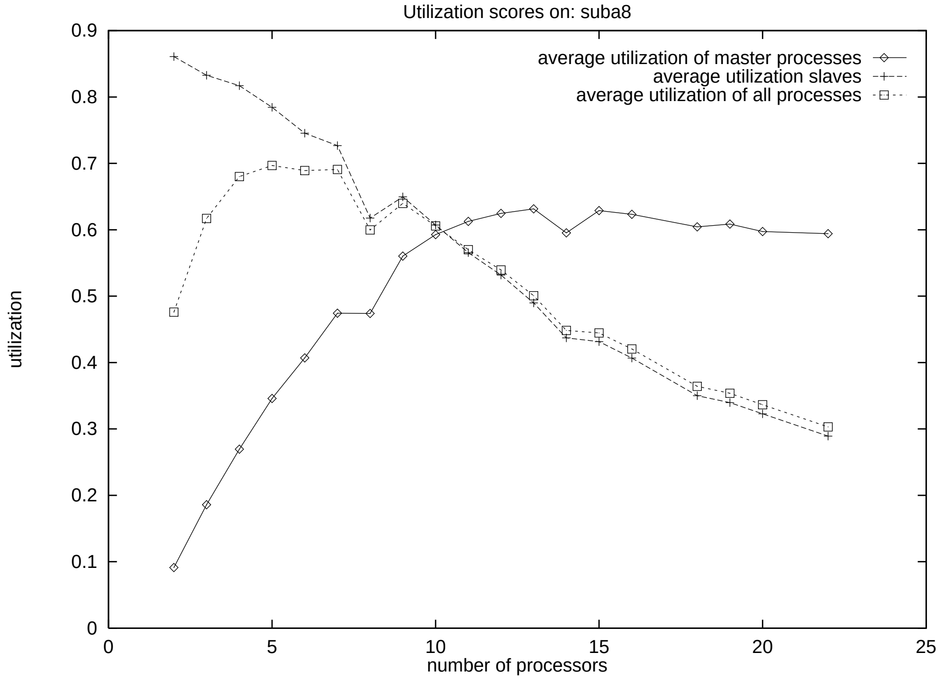


Figure B.3: Utilization scores on 8 × *suba*

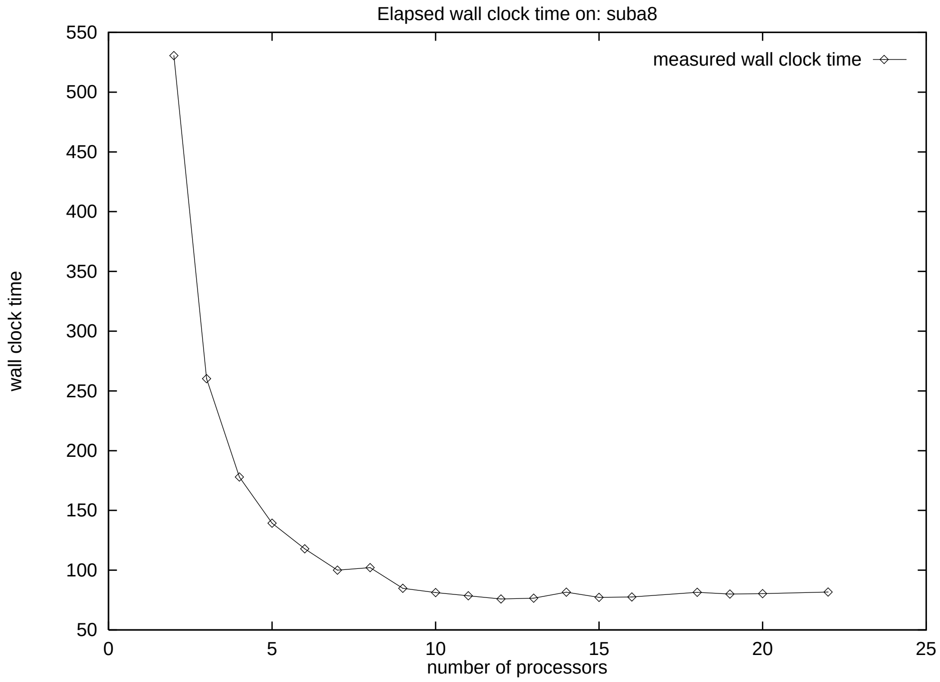


Figure B.4: Response times on 8 × *suba*

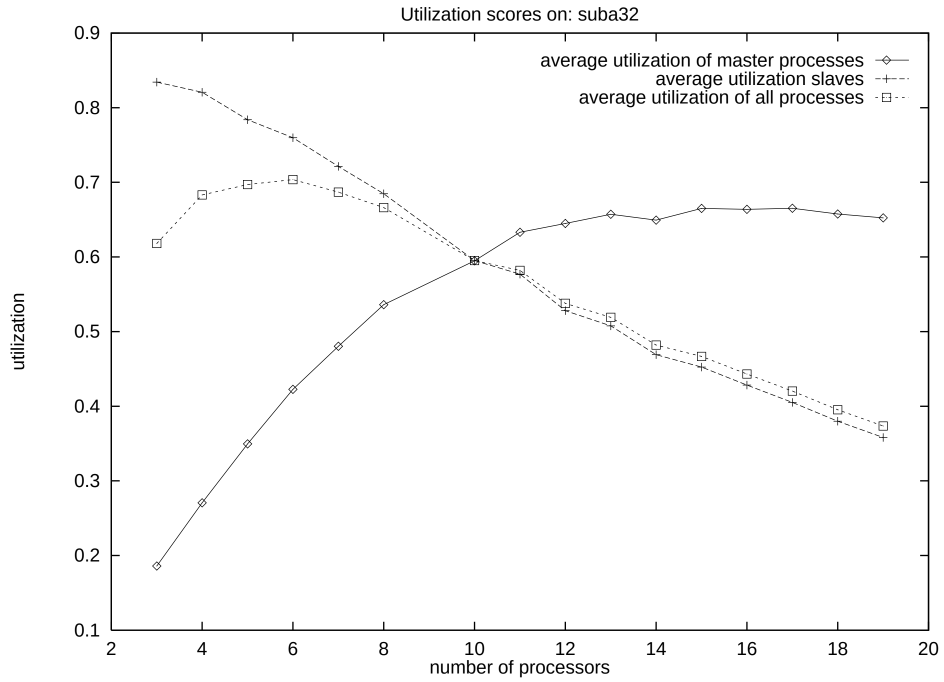


Figure B.5: Utilization scores on 32 × *suba*

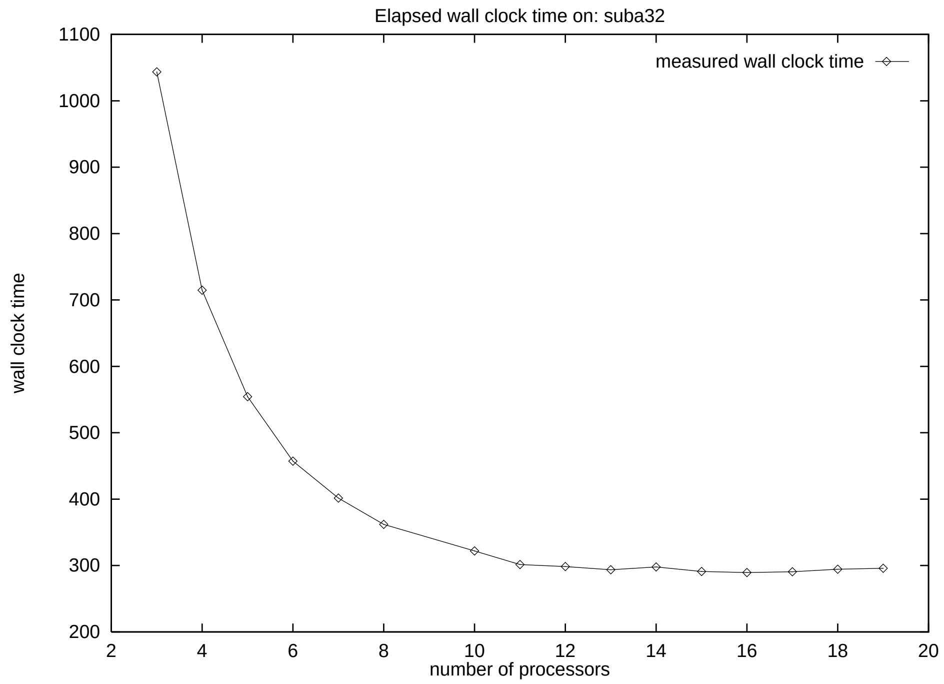


Figure B.6: Response times on 32 × *suba*

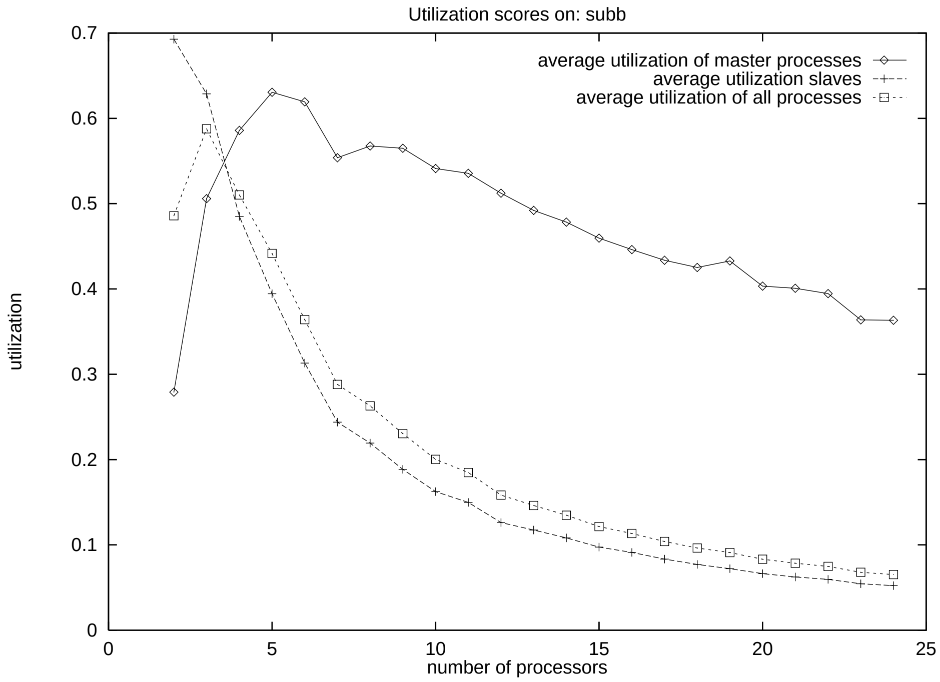


Figure B.7: Utilization scores on *subb*

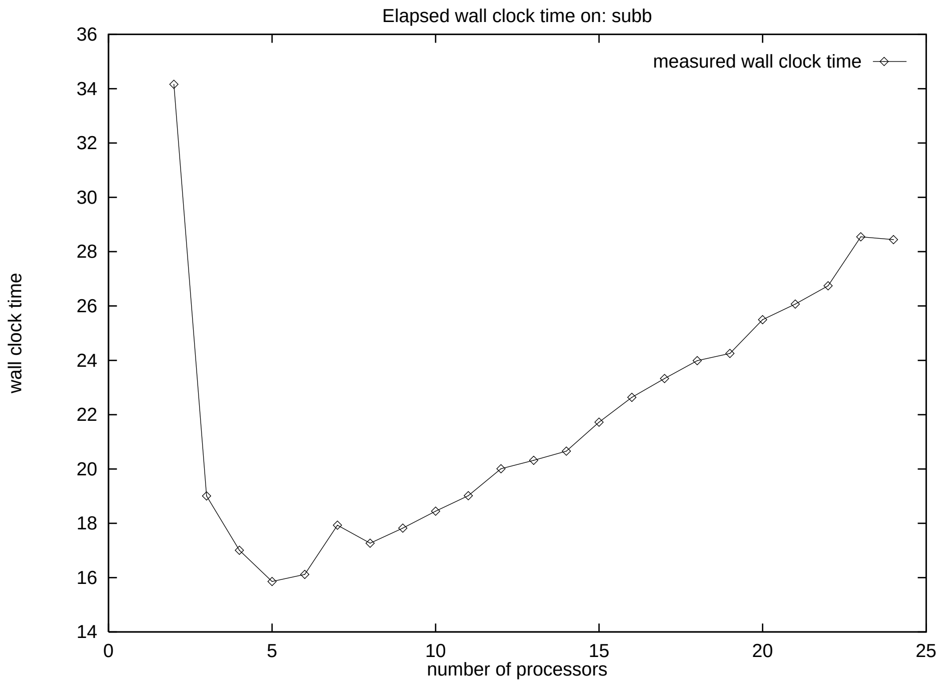


Figure B.8: Response times on *subb*

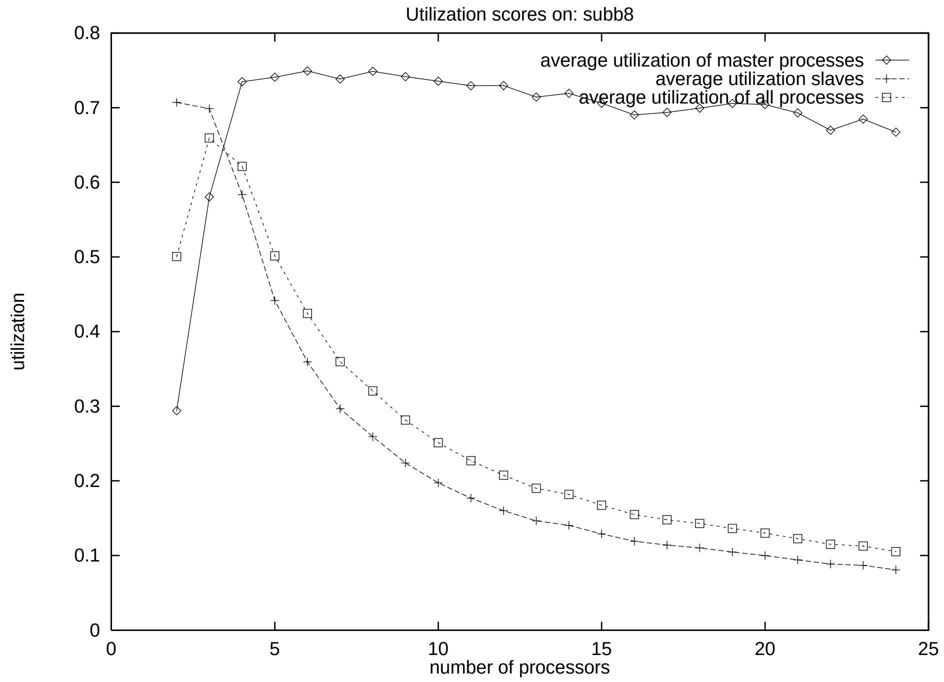


Figure B.9: Utilization scores on 8 × *subb*

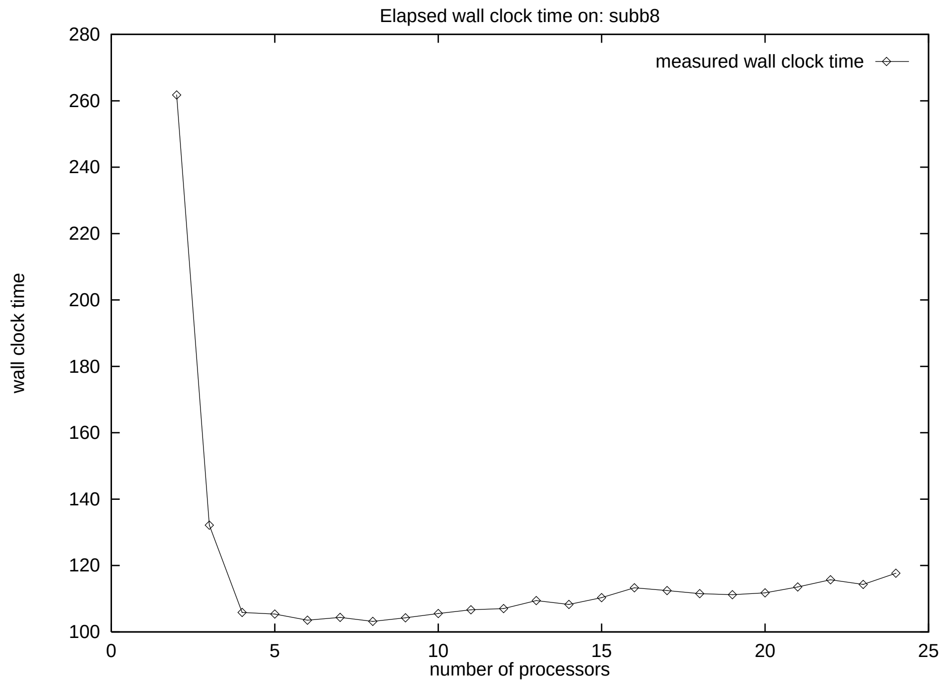


Figure B.10: Response times on 8 × *subb*

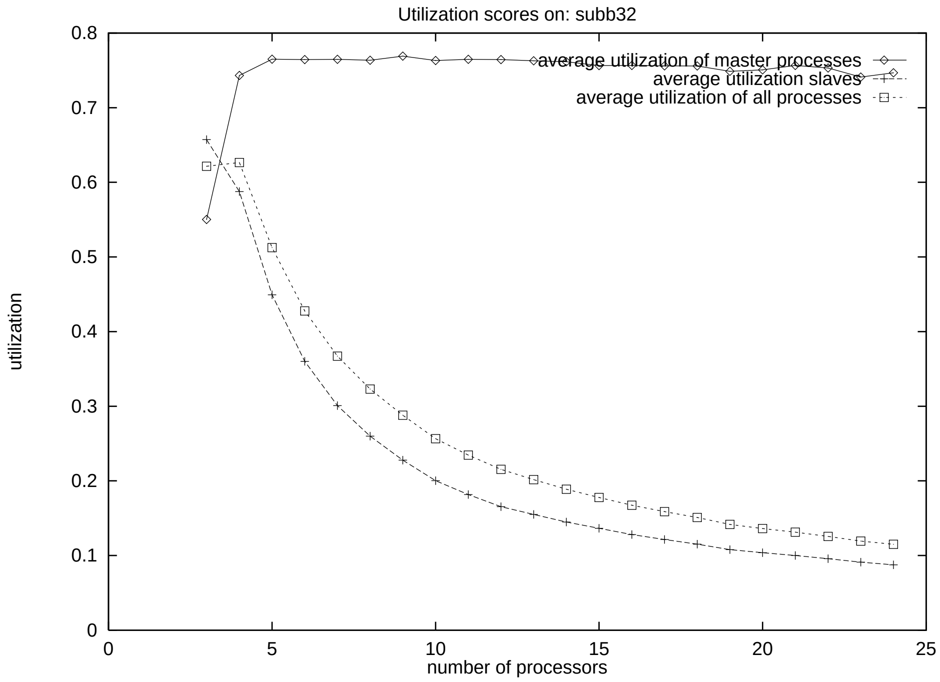


Figure B.11: Utilization scores on 32 × *subb*

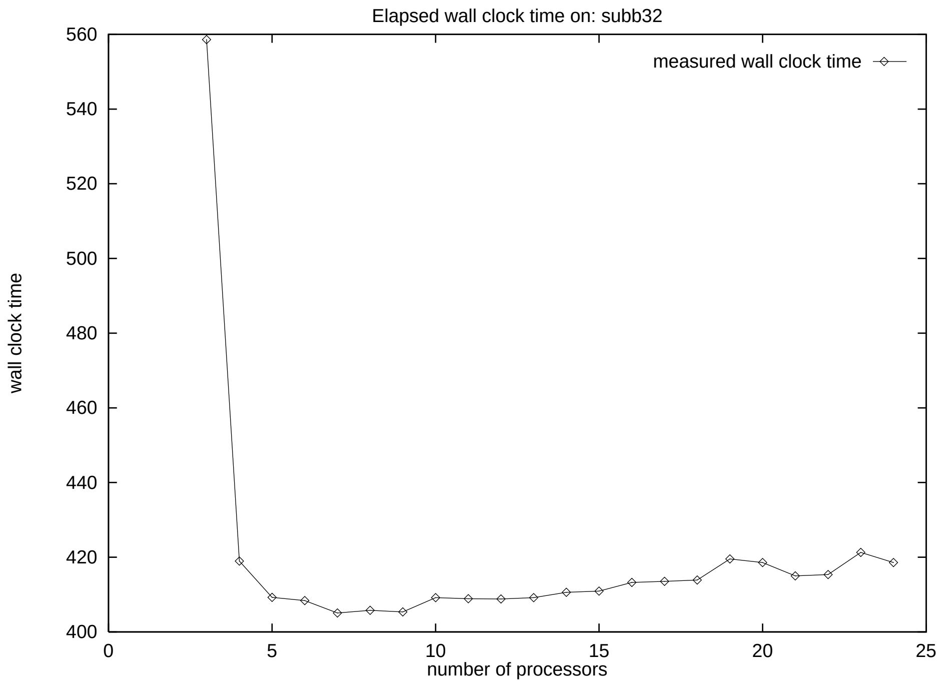


Figure B.12: Response times on 32 × *subb*

B.2 Cell graphs

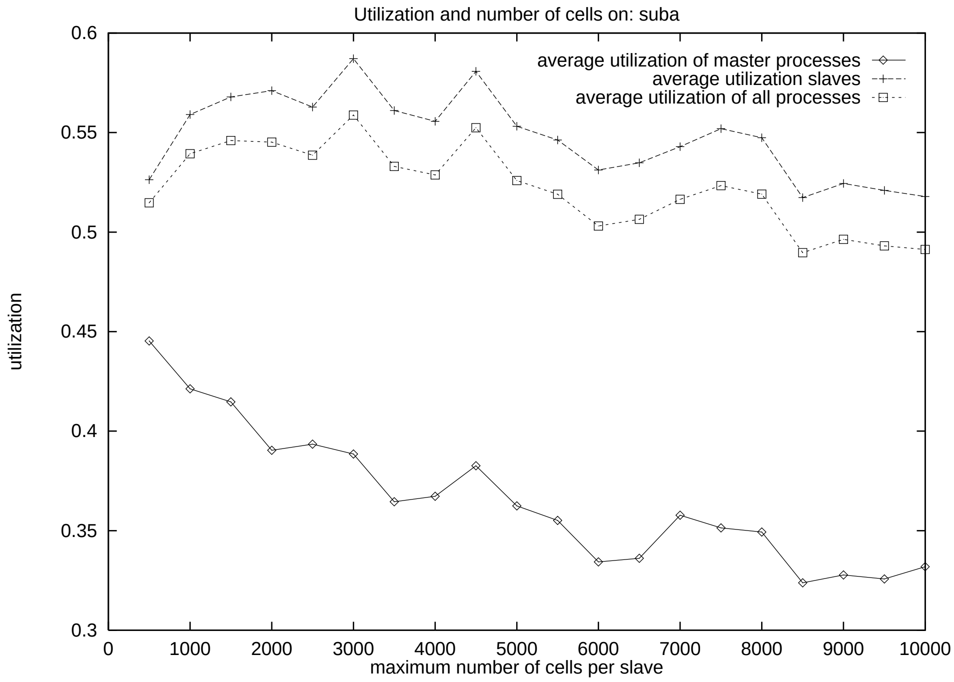


Figure B.13: Utilization scores on *suba* with variable cells

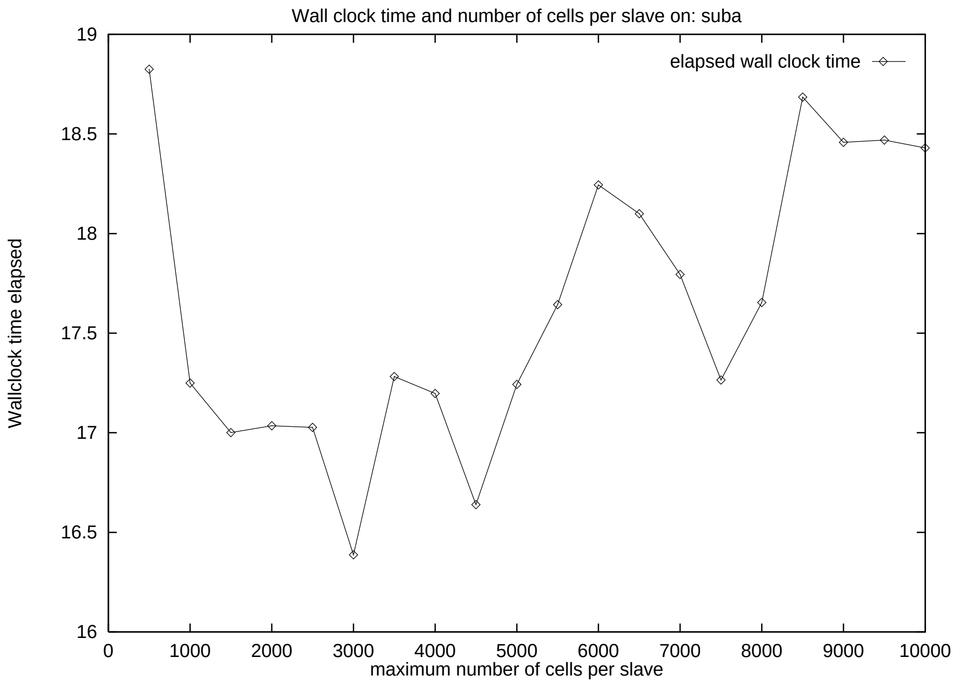


Figure B.14: Response times on *suba* with variable cells

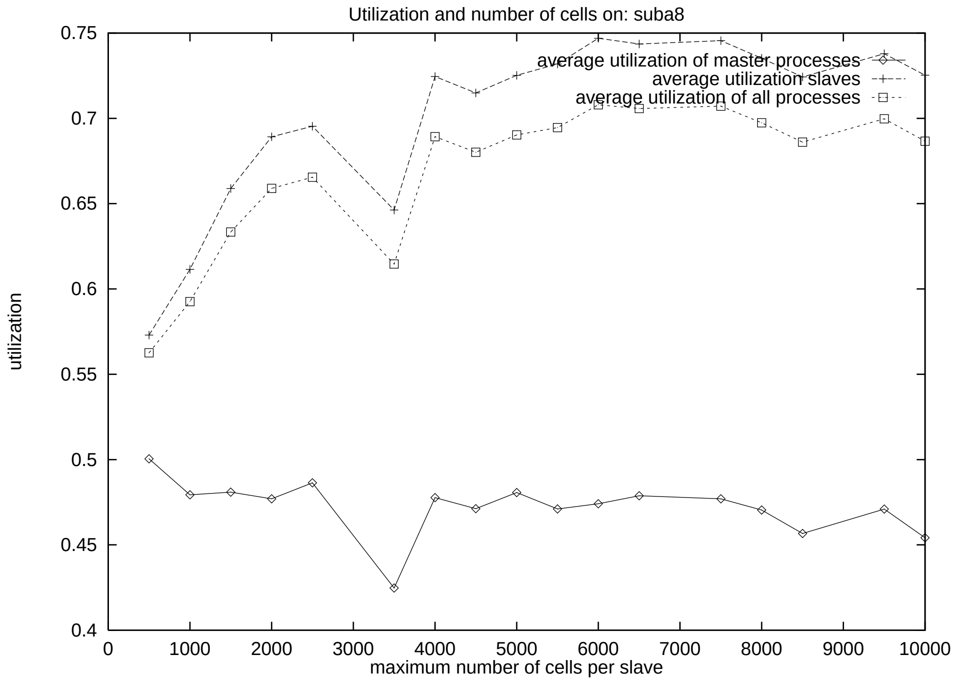


Figure B.15: Utilization scores on 8 × *suba* with variable cells

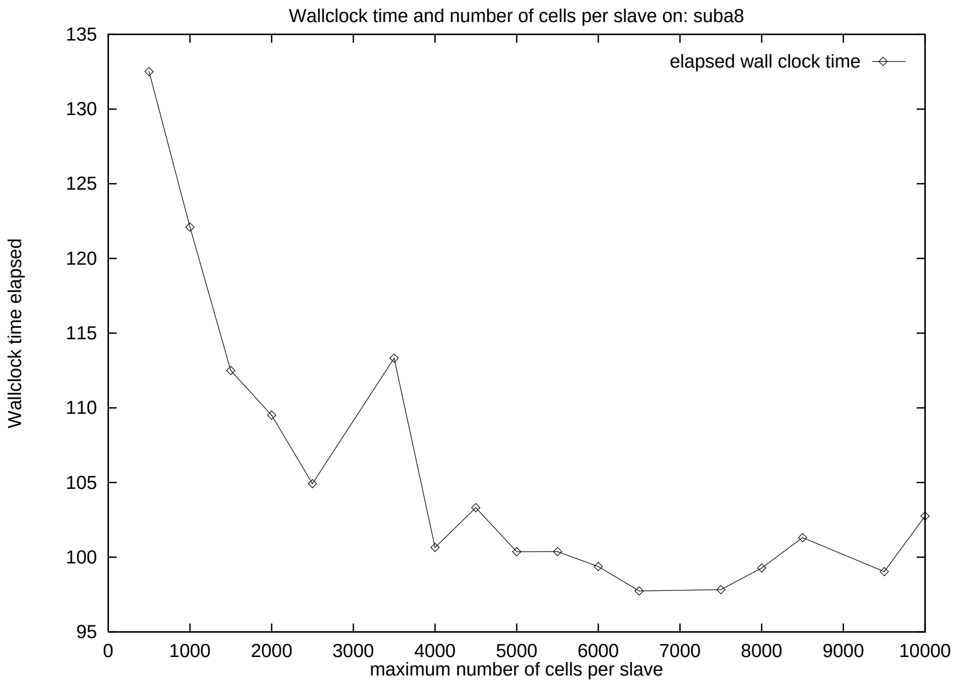


Figure B.16: Response times on 8 × *suba* with variable cells

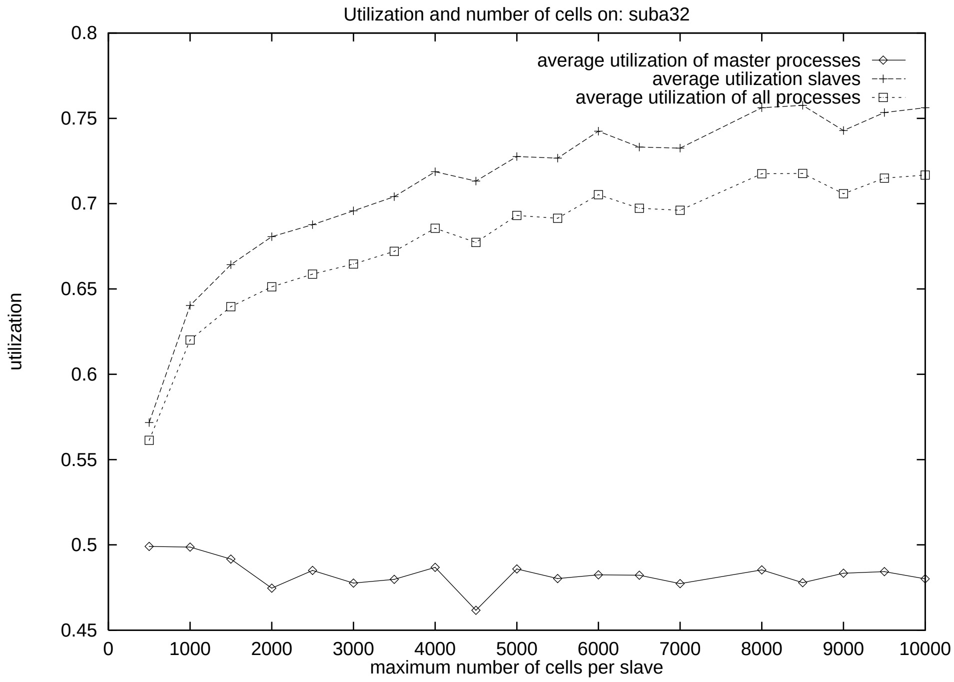


Figure B.17: Utilization scores on 32 $\times$ *suba* with variable cells

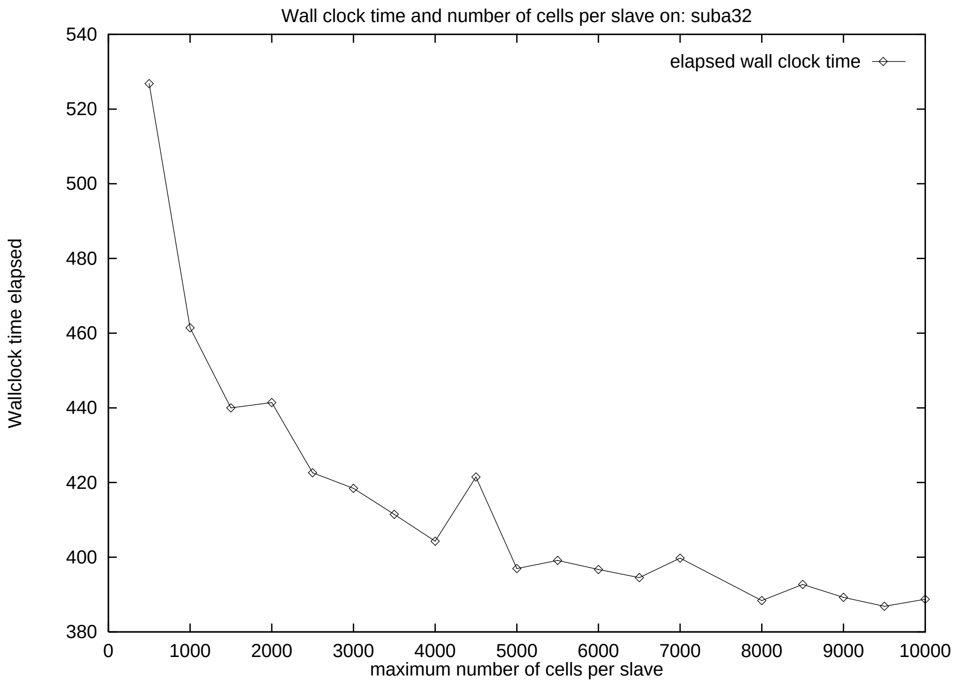


Figure B.18: Response times on 32 $\times$ *suba* with variable cells

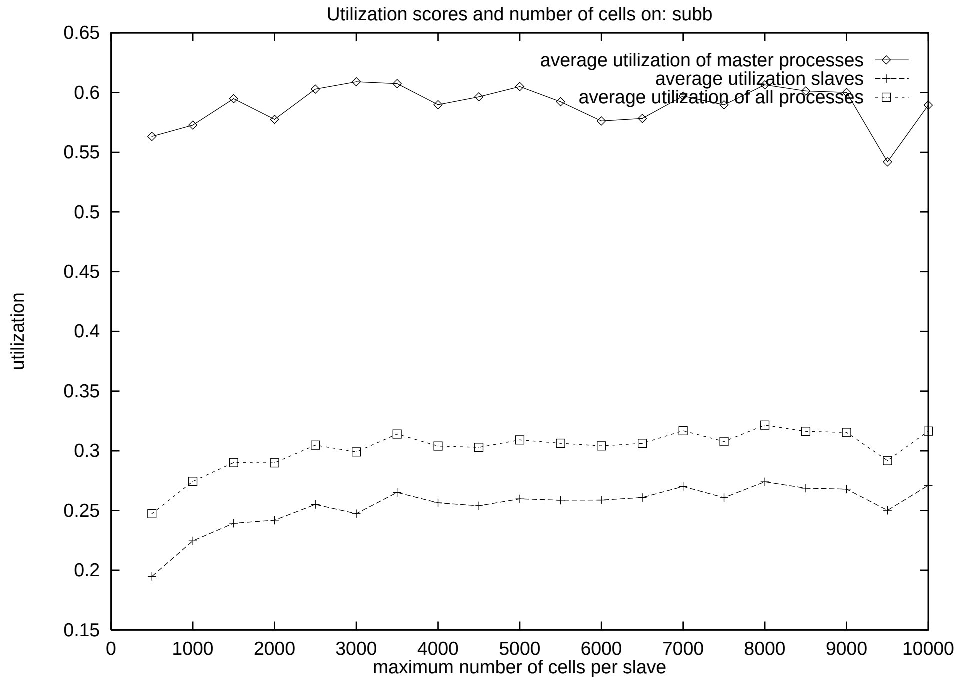


Figure B.19: Utilization scores on *subb* with variable cells

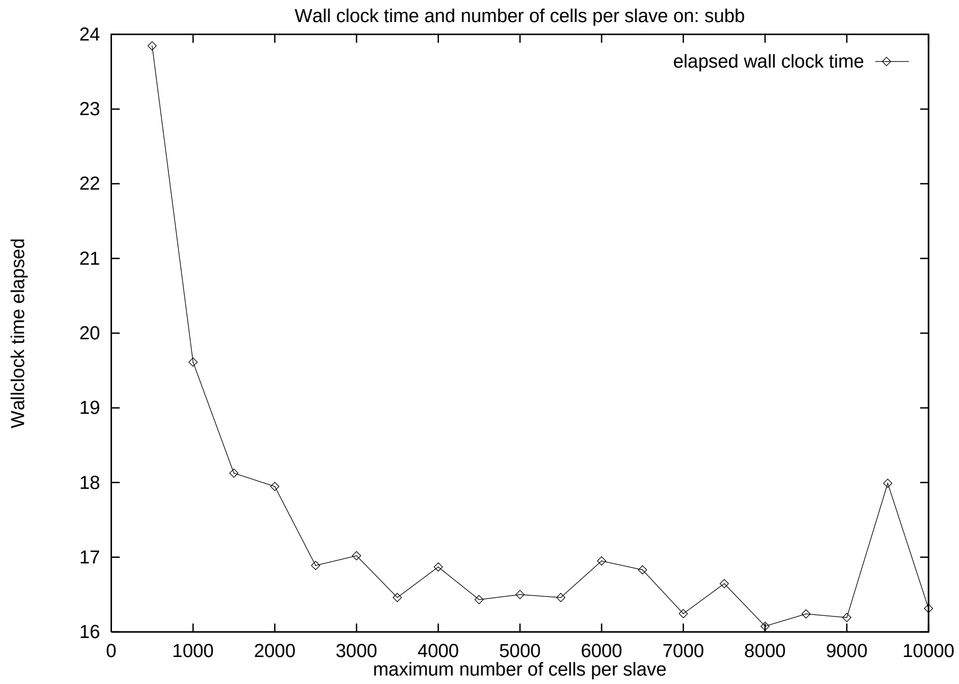


Figure B.20: Response times on *subb* with variable cells

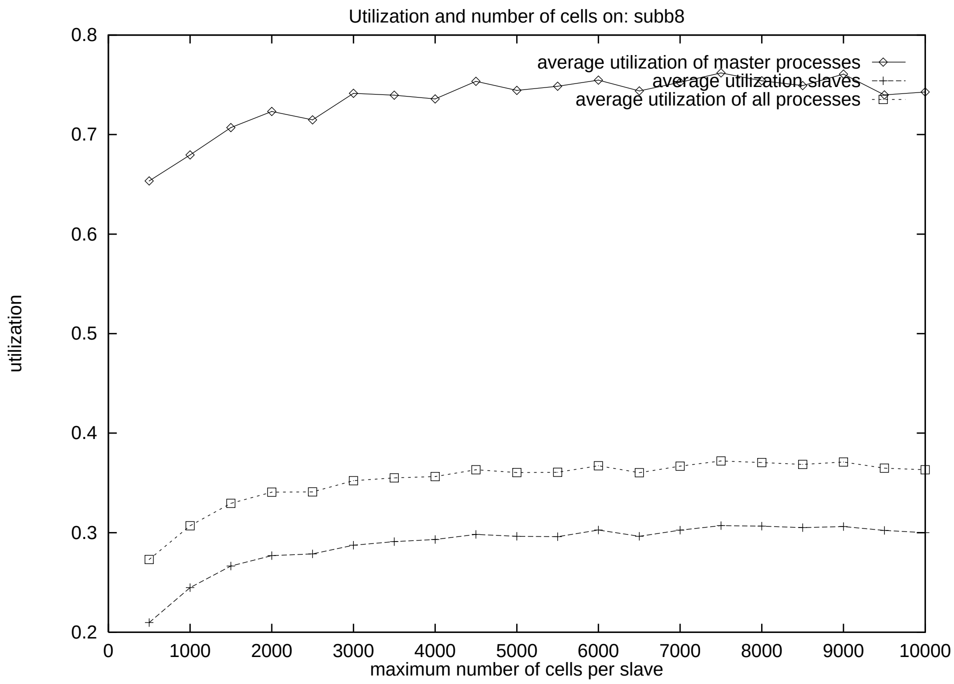


Figure B.21: Utilization scores on 8 × *subb* with variable cells

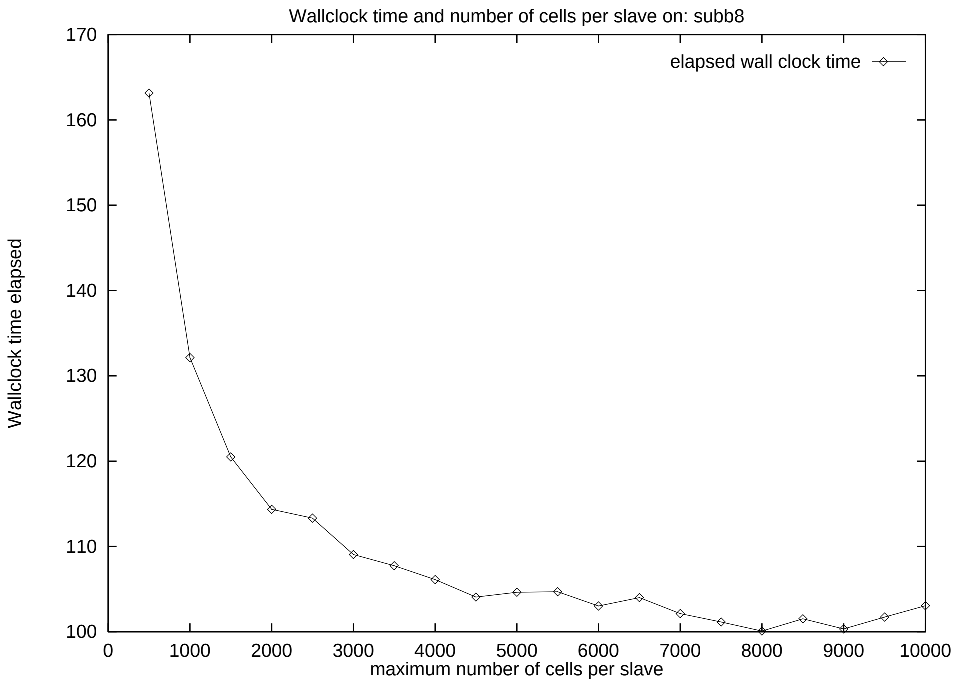


Figure B.22: Response times on 8 × *subb* with variable cells

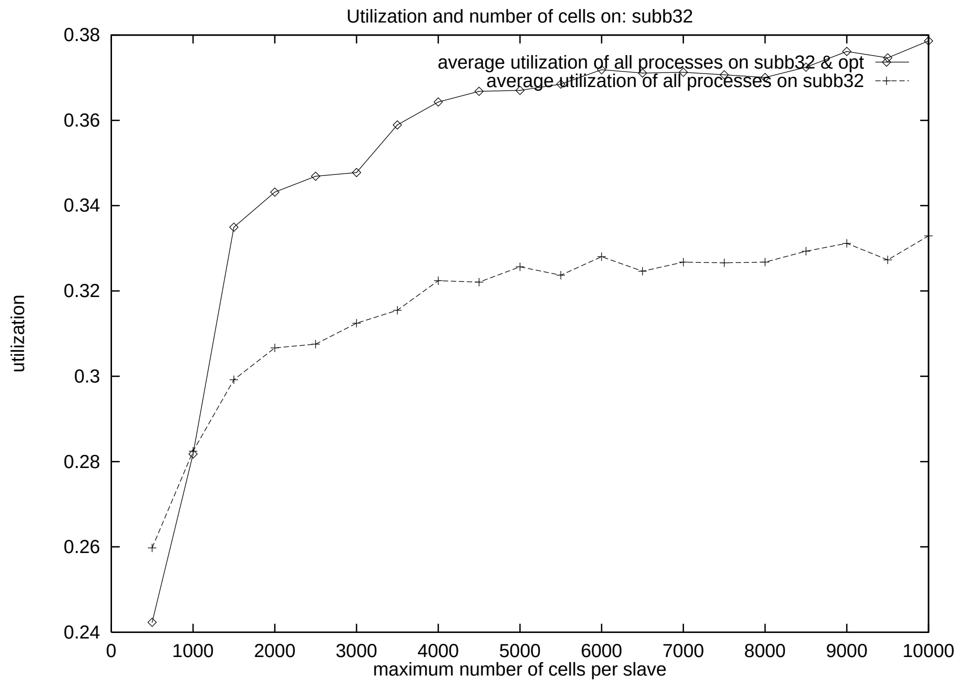


Figure B.23: Utilization scores on 32 × *subb* with variable cells

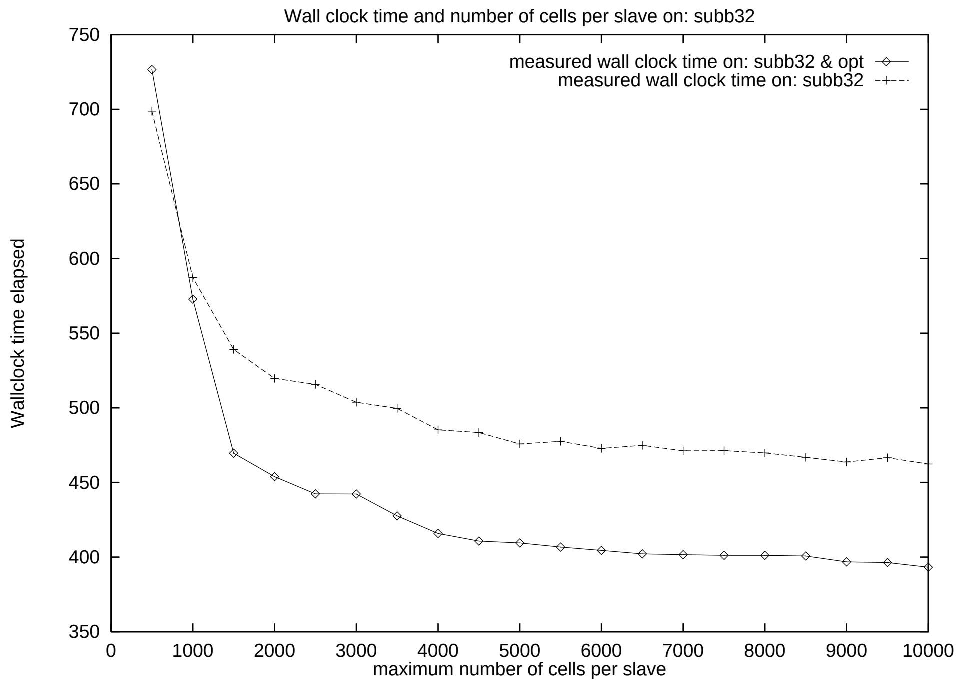


Figure B.24: Response times on 32 × *subb* with variable cells

B.3 Providing order graphs

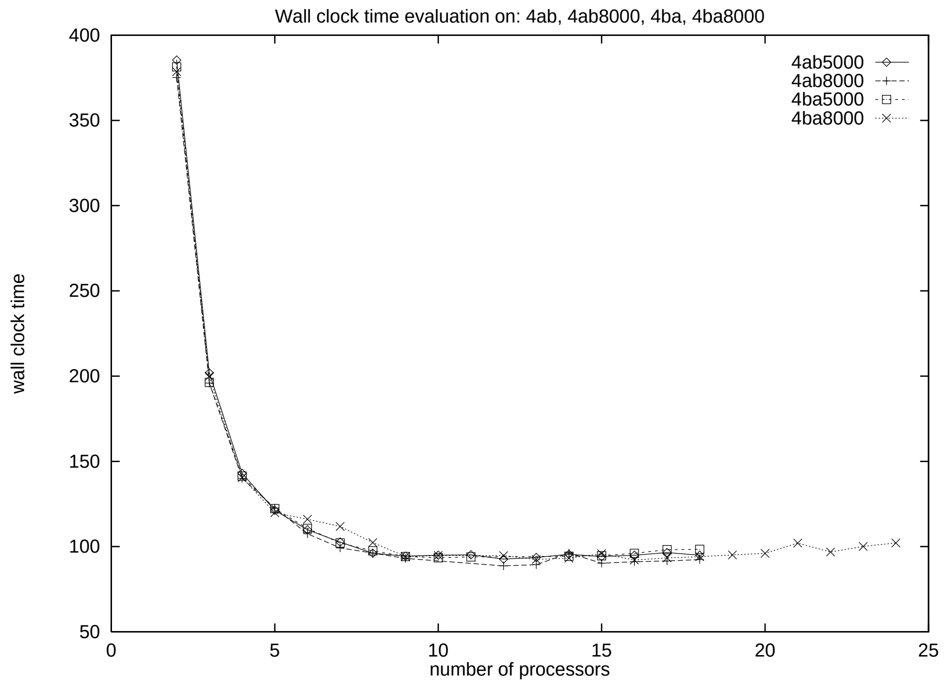


Figure B.25: Response times on $4 \times$ *suba* and $4 \times$ *subb* and $4 \times$ *subb* and $4 \times$ *suba*, with 5000 and 8000 cells respectively

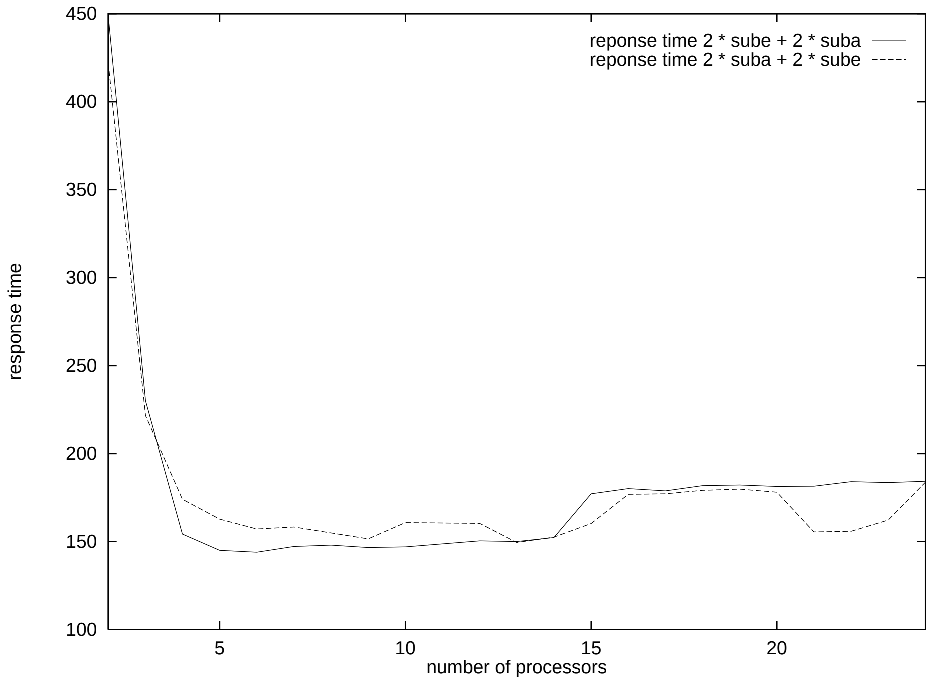


Figure B.26: Response times on $2 \times$ *suba* and $2 \times$ *sube* and $2 \times$ *sube* and $2 \times$ *suba*, with 8000 cells

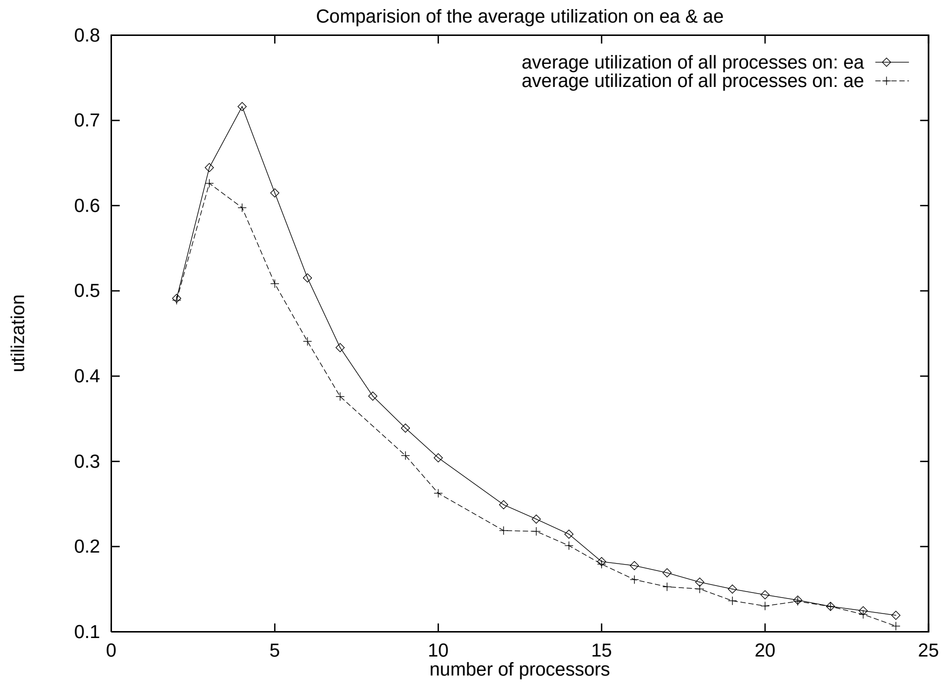


Figure B.27: Utilization scores on 2 × *suba* and 2 × *sube* and 2 × *sube* and 2 × *suba*, with 8000 cells

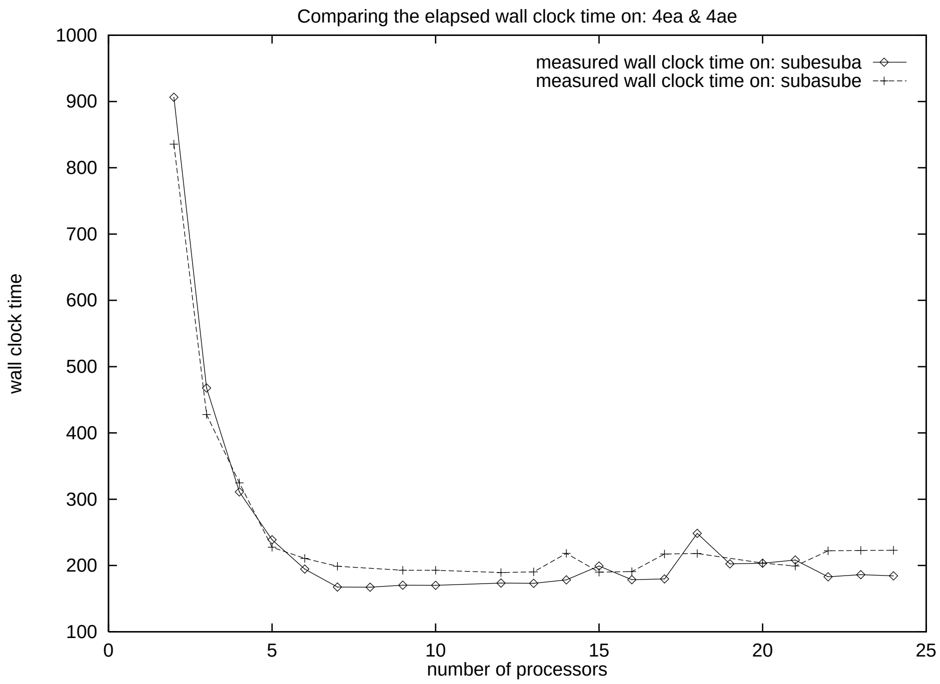


Figure B.28: Response times on 4 × *suba* and 4 × *sube* and 4 × *sube* and 4 × *suba*, with 8000 cells

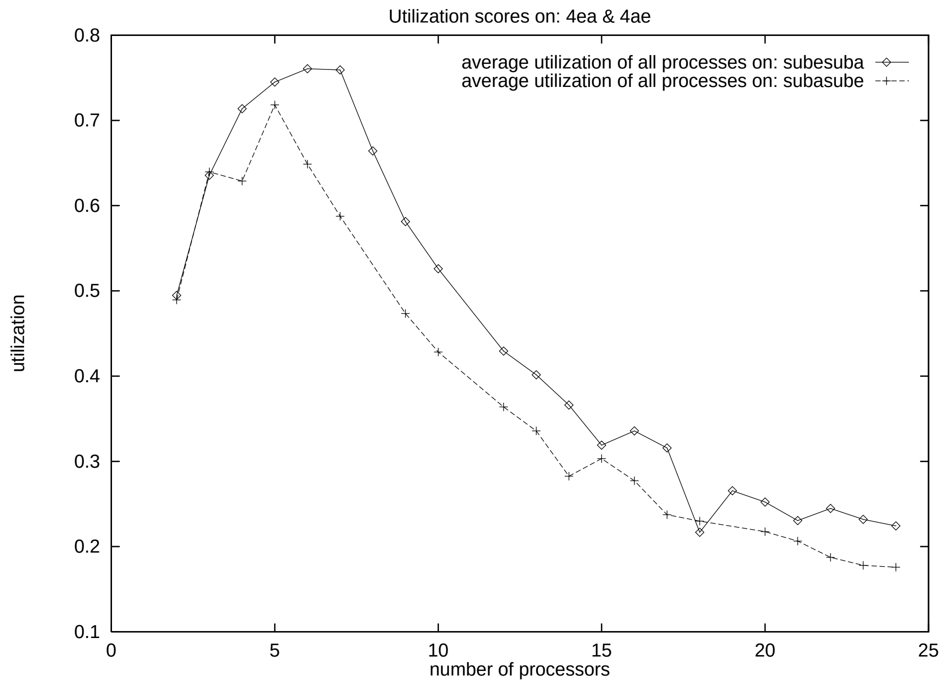


Figure B.29: Utilization scores on 4 × *suba* and 4 × *sube* and 4 × *sube* and 4 × *suba*, with 8000 cells

Bibliography

[AG94] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing*. Benjamin Cummings Publisher Company Inc., 1994.

[BDG+94] Adam Beguelin, Jack Dongarra, G.A. Geist, Robert Manchek, Keith Moore, Peter Newton, and Vaidy Sunderam. *HeNCE: A Users'Guide*, 2.0 edition, June 1994.

[BDV94] Greg Burns, Raja Daoud, and James Vaigl. Lam: An open cluster environment for mpi. Technical report, Ohio Supercomputer Center Columbus, Ohio, 1994.

[BT89] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation*. Prentice Hall, 1989.

[Bur89] Gregory D. Burns. A local area multi computer. In *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, 1989.

[Cen94] Ohio Supercomputer Center. Lam for c programmers. Technical report, The Ohio State University, July 1994.

[DGMS93] J.J. Dongarra, G.A. Geist, R. Manchek, and V.S. Sunderam. Integrated pvm framework supports heterogeneous network computing, January 3 1993.

[edc] edcweb@edcwww.cr.usgs.gov. Thematic mapper landsat data. It was published on a web page. The web page adres is: http://sun1.cr.usgs.gov/glis/hyper/guide/landsat_tm.

[For94] Message Passing Interface Forum. Mpi: A message-passing interface standard. Technical report, ARPA and NSF under grant ASC-930330, the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, and by the Commission of the European Community through Esprit project P6643, March 1994.

[Gmb94] Parsytec GmbH. Parsytec powerxplorer. Technical Report 1.1, Parsytec GmbH, may 1994. Hardware User Guide.

[Gro95a] William Gropp. Minutes of mpi meeting april 24-26 chicago il, April 26 1995.

[Gro95b] William Gropp. Minutes of mpi meeting july 26-28 chicago il, July 28 1995.

[Gro95c] William Gropp. Minutes of mpi meeting june 5-7 chicago il, June 7 1995.

[Gro95d] William Gropp. Minutes of mpi meeting march 13-15 chicago il, March 23 1995.

[Gro95e] William Gropp. Minutes of mpi meeting september 6-8 chicago il, September 15 1995.

[GW93] Rafael C. Gonzales and Richard E. Woods. *Digital Image Processing*. Addison-Wessley, September 1993.

[Jan95] Paul Janssen. Parallel region growing and parallel programming environments for workstations clusters. Master's thesis, Computing Science Department, Faculty of Computing Science, Catholic University of Nijmegen, June 1995.

[JMS95] Robert L.Stevenson Jeffrey M. Squyres, Andrew Lumsdaine. A cluster-based parallel image processing toolkit. Technical report, Laboratory for Scientific Computing, Deprartment of Computer Science and Engineering, and Laboratory for Image and Signal Analysis, Department of Electrical Engineering, University of Notre Dame, 1995.

[Kan92] K. Kant. *Introduction to Computer System Performance Evaluation*. McGraww-Hill, 1992.

[LDS94] Joep Vesseur Leen Dikken, Frank van der Linden and Peter Sloot. Dynamicpvm: Dynamic load balancing on parallel systems. Technical report, Shell Nederland Informatieverwerking Rijswijk and Faculty of Mathematics and Computer Science, University of Amsterdam, 1994.

[Lus95] Rusty Lusk. Minutes of mpi meeting october 23-25 denver co, November 10 1995.

[NN94] Natawut Nupairoj and Lionel M. Ni. Performance evaluation of some mpi implementations on workstation clusters. Technical report, NSF grants CDA-9121641 and MPI-9204066, and DOE grant DE-FG02-93ER25167, 1994.

[PLVC94] Donna S. Reese Paula L. Vaughan, Anthony Skjellum and Fei-Chen Cheng. Migrating from pvm to mpi, part i: The *unify* system. Technical report, NSF Engineering Research Center for Computational Field Simulation & Department of Computer Science Mississipi State University, July 15 1994.

[Sch93] Rene Schreurs. Correctnes and efficiency of the best-merge's region-growing. Master's thesis, Computing Science Department, Faculty of Computing Science, Catholic University of Nijmegen, November 1993.

[Sch95] Ron Schoenmakers. *Integrated Methodology For Segmentation Of Large Optical Satellite Images In Land Applications Of Remote Sensing*. PhD thesis, Computing Science Department, Faculty of Computing Science, Catholic University of Nijmegen, September 1995.

[SJGM94] V.S. Sunderam, Weicheng Jiang, G.A. Geist, and J.J. Dongarra & R. Manchek. Pvm 3 user's guide and reference manual. Technical report, University of Tennesee, Oak Ridge National Laboratory, Emory University, September 1994.

[WJ89] Stephen A. Ward and Robert H. Halstead Jr. *Computations Structures*. The MIT Press and Mc Graw-Hill Book Company, 1989.

[ZM94] Taieb F. Znati and Rami G. Melhem. A uniform framework for dynamic load balancing strategies in distributed processing systems. In *Journal of Parallel and Distributed Computing*, 1994.