Universitat Autònoma de Barcelona

Universitat Autònoma de Barcelona
Escola d'Enginyeria

Computer Architecture and Operating Systems Department

M.Sc. in High Performance Computing, Information Theory and Security

Master Thesis

Parallelizing remote sensing image geometric correction

by

Gerard Bernabeu

Date: July 3, 2012

Directed by: Ana Cortés Fité
Lluís Pesquer Mayos

Master Thesis

M.Sc. in High Performance Computing, Information Theory and Security

Course 2011-12

Parallelizing remote sensing image geometric correction

Author: Gerard Bernabeu Altayó

Directors: Ana Cortés Fité
Lluís Pesquer Mayos

Computer Architecture and Operating Systems
Escola d'Enginyeria
Universitat Autònoma de Barcelona

Signatures

Author Directors

Abstract

Remote sensing spatial, spectral, and temporal resolutions of images, acquired over a reasonably sized image extent, result in imagery that can be processed to represent land cover over large areas with an amount of spatial detail that is very attractive for monitoring, management, and scientific activities.

With Moore's Law alive and well, more and more parallelism is introduced into all computing platforms, at all levels of integration and programming to achieve higher performance and energy efficiency.

Being the geometric calibration process one of the most time consuming processes when using remote sensing images, the aim of this work is to accelerate this process by taking advantage of new computing architectures and technologies, specially focusing in exploiting computation over shared memory multi-threading hardware.

A parallel implementation of the most time consuming process in the remote sensing geometric correction has been implemented using OpenMP directives. This work compares the performance of the original serial binary versus the parallelized implementation, using several multi-threaded modern CPU architectures, discussing about the approach to find the optimum hardware for a cost-effective execution.

Keywords

Parallel Computing, HPC, OpenMP, co-processor, GIS, Landsat, Geometric Correction

Resumen

Las resoluciones espaciales, espectrales y temporales de imágenes de teledetección, adquiridas a un tamaño razonable, dan como resultado imágenes que se pueden procesar para representar grandes áreas de terreno con un nivel de detalle espacial que es muy atractivo para la observación y la gestión, así como para actividades científicas.

Con la ley de Moore aún vigente, más y más paralelismo es introducido a cada nueva generación para todas las plataformas de computación. El paralelismo está presente en todos los niveles de integracion y en la programación, con el fin de obtener un mayor rendimiento y eficiencia energética.

Siendo el proceso de calibración geométrica uno de los procesos más costosos computacional y temporalmente cuando utilizamos imágenes de teledetección, el objetivo de este trabajo es acelerar este proceso mediante el aprovechamiento de las nuevas tecnologías y arquitecturas de computación, haciendo especial hincapié en la explotación de hardware paralelo con memoria compartida.

Mediante el uso de directivas OpenMP se ha paralelizado la etapa más costosa y lenta del proceso de corrección geométrica de imágenes de teledetección. Este trabajo compara el rendimiento de la aplicación original serie con la versión parelelizada mediante pruebas en distintos sistema multiprocesador, proponiendo varios enfoques para escoger el hardware más adequado para una ejecución óptima.

Palabras claves

Computación paralela, HPC, OpenMP, co-procesador, GIS, Landsat, corrección geométrica

Resum

Les resolucions espacials, espectrals i temporals d'imatges de teledetecció, adquirides a una mida raonable, donen com a resultat imatges que es poden processar per a representar grans àrees de terreny amb un nivell de detall espacial que és molt atractiu per a l'observació i la gestió, així com per a les activitats i recerca científiques.

Amb la llei de Moore encara vigent, més i més paral·lelisme és introduït a cada nova generació per a totes les plataformes de computació. El paral·lelisme és present en tots els nivells d'integració i també en la programació, amb la finalitat d'obtenir un millor rendiment i eficiència energètica.

Éssent el procés de calibració geomètrica un dels processos més costosos computacional i temporalment quan utilitzem imatges de teledetecció, l'objectiu d'aquest treball és accelerar aquest procés mitjançant l'aprofitament de les noves tecnologies i arquitectures de computació, fent especial èmfasi en l'explotació de maquinari paral·lel amb memòria compartida.

Mitjançant l'ús de directives OpenMP s'ha paral·lelitzat l'etapa més costosa i lenta del procés de correcció geomètrica d'imatges de teledetecció. Aquest treball compara el rendiment de l'aplicació original sèrie amb la versió parelelizada mitjançant proves en diferents sistemes multiprocessador, proposant diversos enfocaments per tal d'escollir el maquinari més adequat per a una execució òptima.

Paraules claus

Computació paralela, HPC, OpenMP, co-processador, GIS, Landsat, correcció geomètrica

Contents

Chapter 1

Introduction

1.1 Motivation

Started in 1972, the Landsat program has been providing a continuous record of Earth Observation. Spatial, spectral, and temporal resolutions of Landsat, acquired over a reasonably sized image extent, result in imagery that can be processed to represent land cover over large areas with an amount of spatial detail that is very attractive for monitoring, management, and scientific activities. Since 2008 USGS distributes Landsat images for free[1].

U.S. Geological Survey (USGS) and other image providers offer several pre-processed images but further image processing is required to achieve the degree of image refinement required by many use cases. To this purpose CREAF has developed MiraMon[1], a modular software tool-kit that is used to improve and analyse remote sensing images.

One of the most time consuming processes when using remote sensing images is the geometric calibration process. The aim of this work is to accelerate this process by taking advantage of new computing architectures and technologies.

1.2 Automatic matching of orthorectified imagery

Automated remote sensing image geometric correction to orthorectified[2] images, as described by X. Pons et al. [3], is a calculation intensive operation which can take from many minutes to a few hours, depending on the image complexity (clouds, cities, mountains, etc).

[1] `http://landsat.usgs.gov/products_data_at_no_charge.php`

The geometric correction is a process which needs to be applied image by image, therefore big terrain areas need of processing scalability. Being able to rapidly correct images is critical to enable certain applications like natural disaster simulation and forecast.

Within a wider project and in order to ensure continuity and a wide range of potential users, CREAF (Center for Ecological Research and Forestry Applications) is developing a Web Processing Service[4] (WPS) within the standard protocols of Open Geospatial Consortium (OGC) which allows to apply the automated remote sensing image geometric correction.

The objective of this work is to accelerate CREAF's remote sensing image geometric correction WPS based Geographic Information Systems (GIS).

The process will be implemented initially using Landsat 7 NASA images, each one covering an area of 185 km X 172 km, which can be later generalizable to other image sources (e.g. Satellites SPOT 4, SPOT 5, etc).

1.3 High Performance Computing architecture trends

With Moore's Law alive and well, more and more parallelism is introduced into all computing platforms at all levels of integration and programming to achieve higher performance and energy efficiency.

In 2012, a regular desktop PC typically has 4 CPU cores or more, as well as a graphics accelerator (GPU) which might be also able to aid in some specific computing tasks (e.g. with CUDA). With new server CPU architectures it is possible to acquire a four socket 64 core shared memory server (e.g. 4x16 core CPU) with the same budget that a few years ago was required to buy a two socket 2 core machine. Even some high-end smartphones are equipped nowadays with multi-core CPUs. The multi-core architecture has been around for a few years already and is here to stay.

The incursion of multi and many-core architectures in all market segments is also impacting clusters, where it is now more important than ever to be able to take advantage of all kinds of parallelisms and accelerators.

1.4 Software parallelization approaches

In order to take advantage of HPC architecture trends, many software solutions have been, and are, in active development. Especially in the area of High Performance Computing (HPC) users can entertain a combination of different hardware and software parallel architectures and programming environments. Those technologies range from vectorization and SIMD computation over shared memory multi-threading (e.g. OpenMP) to distributed memory message passing (e.g. MPI) on cluster systems.

With the purpose to use GPU's increasing computing power, some non-graphics oriented programming frameworks for GPU are emerging, like Nvidia's CUDA. This frameworks are still under heavy development and different CPU generations often require software adaptations. Under ideal conditions it is now possible to get many GigaFLOPS[2] from a single GPU with a very competitive flop/price ratio. On the other hand, heavy software redesign and recoding processes are still required to take full advantage of GPU architectures.

1.5 Contributions and Outline

The present work contributes to the optimisation of a widely used real world application whose enhancements helps to accelerate and improve research in many fields which require of properly orthorectified images.

Within the scope of the present work I suggest a more robust and above all scalable approach to store and retrieve the stored map image files, which is independent of the used orthorectification algorithm. With a better shared storage system it would be possible to access data faster and therefore improve the overall geometric correction process. A distributed storage system, like HadoopFS, might be a good solution providing both high availability and high performance at a reasonable cost.

The remaining work is organised as follows: The subsequent chapter 2 gives an introduction to the algorithm used for automatic matching of orthorectified images. The CREAF's MiraMon software is presented, followed by an in depth description of some AfinaPC algorithm specific concepts, which is the basic framework for this thesis.

[2]In computing, FLOPS (or flops or flop/s, for floating-point operations per second) is a measure of a computer's performance, especially in fields of scientific calculations that make heavy use of floating-point calculations, similar to the older, simpler, instructions per second. One GigaFLOPS is 10^9 FLOPS.

Chapter 3 briefly describes an outline of current mainstream CPUs, covering the different layers of parallelization implemented in them, as well as some key architectural aspects like cache hierarchy and effective memory bandwidth.

Chapter 4 analyses the AfinaPC algorithm from the computing point of view and discusses about different approaches for several parallelization techniques i.e. MPI, GPU/CUDA and OpenMP.

Precisely this last approach is explored in detail in chapter 5. The conducted experiments to back up the assumptions and expectations of the prior chapters are presented. Besides the experimentation with different CPU architectures to measure several performance metrics, it comprises an analysis of the detected bottlenecks. Finally the work is terminated with conclusions and some proposals for future enhancements in chapter 6.

Chapter 2

Autotmatic matching of orthorectified images

2.1 Overview

CREAF has developed a protocol to match remote sensing images series with orthorectified imagery. As described in [3] the objective of the matching is to automatically find ground control points (GCP) that will be used in a geometric correction model.

The whole process is composed by three main stages: the creation of a GCP bank, the matching between the candidate image and the reference image for each GCP and, finally, the correction stage, which includes the model fitting and validation with different GCP. Each of the stages is performed by one or more independent applications, from the MiraMon software tool-kit, that should work in a pipelined way so that the image is finally orthorectified.

This work means to accelerate the automatic matching process, for this reason the most time consuming stage was selected as main candidate for the optimization process; this stage is the second one: the matching between the candidate image and the reference image for each GCP.

The matching of the GCPs is a crucial and complex step which scales linearly with the number of GCPs. The application in charge of this process within the MiraMon toolkit is called AfinaPC.

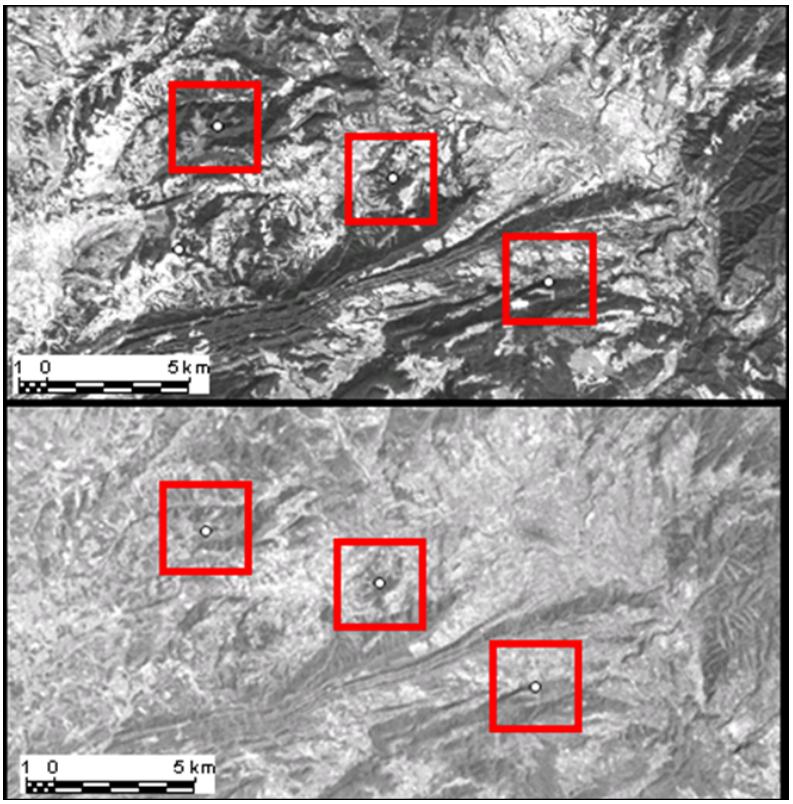

Figure 2.1: On top: reference image (2008). On bottom: Landsat 1 MSS acquired on 18/8/1972. GCP are represented by white points centred in a red square in order to facilitate the visualization.

2.2 AfinaPC

The AfinaPC program precises the recognition of homologue ground control points (GCP).

Given two images with similar geometry, AfinaPC determines the location of the provided GCP from a pattern reference image (pattern image) in the image which needs to be orthorectified (candidate image). The search is performed by analysing each GCP approximate surrounding areas in both images in order to find the maximum correlation between two given groups of adjacent points, also called sub-window matching. The search area size is determined by the search window size and the size of the GCP is determined by correlation window. In figure 2.1 matching GCP are shown for two images of the same area, the task of automatically finding the most approximate match for each point is performed by the AfinaPC application.

Once the required major global image calibration movement has been determined, fine-grained pixel level shifts are tested to improve the maximum correlation between the two images and, therefore, obtain a better homologue points identification.

Finally the global required orthorectification displacement is calculated using

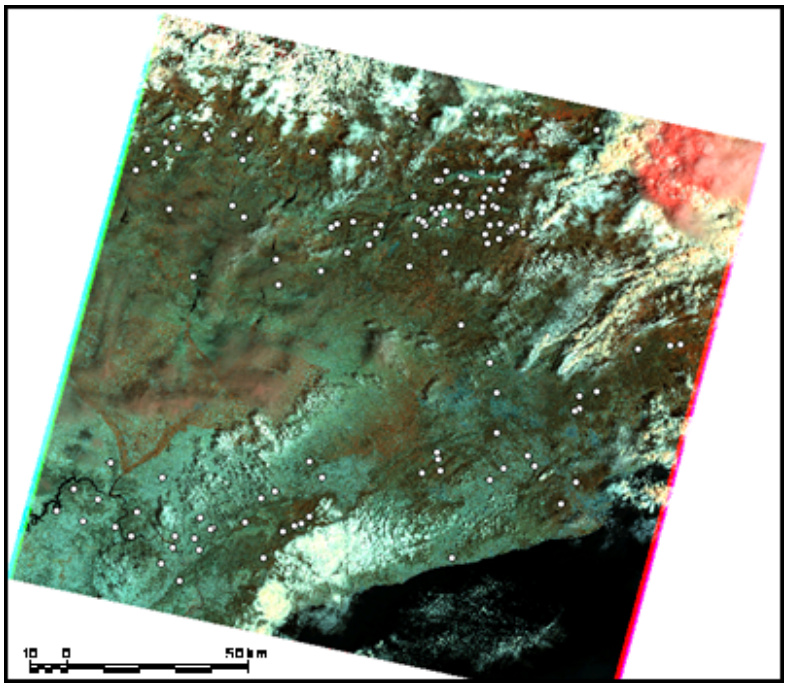

Figure 2.2: 4+5+7 composite of Landsat 5 TM acquired on 3/6/1987. GCPs are represented as white points

the desired mode (median or mode) taking the better fitting GCPs found, i.e. the ones with greater correlation values.

In case some rotation is detected, it is necessary to use another application from the MiraMon toolkit in order to determine it (CercaRot) and then redefine the new GCPs (with TraiPC). The AfinaPC software is not able to find correct correlations with escalated images either.

2.3 Ground Control Points

Traditionally the Ground control Points (GCPs) selection in an image is a manual process where only a few number of GCP is able to be accurately located, mainly due to the limitations of human eye to locate GCPs in regions without conspicuous morphologies.

CREAF's procedure selects GCPs in the pattern image automatically, this leads to the selection of hundreds or even thousands of GCPs which in return provide more accurate image corrections, allowing the system to properly correct images with significant differences due to many reasons like weather conditions (e.g. snow, clouds) or very separated in time; i.e. the area has suffered considerable changes like human buildings. Figure 2.2 show a typical distribution of GCP in a map image.

2.4 SLC-off imagery

The current Thematic Mapper (TM) class of Landsat sensors began with Landsat-4, which was launched in 1982[5]. This series continued with a similar sensor on Landsat-5, launched in 1984. The latest sensor in the series is the Landsat-7 Enhanced Thematic Mapper Plus (ETM+), which was carried into orbit in 1999.

As of June 2012, both the Landsat-5 TM and the Landsat-7 ETM+ are operational and providing data. Despite 20+ years of operation, the TM on Landsat-5 is still functional, although downlinks for the data are limited and since November 2011 it's in a degraded state. Landsat-7 ETM+ experienced a failure of its Scan Line Corrector (SLC) mechanism in May 2003 and has had some further issues since then. The status of the Landsat satellite series can be followed at the Landsat USGS website[1].

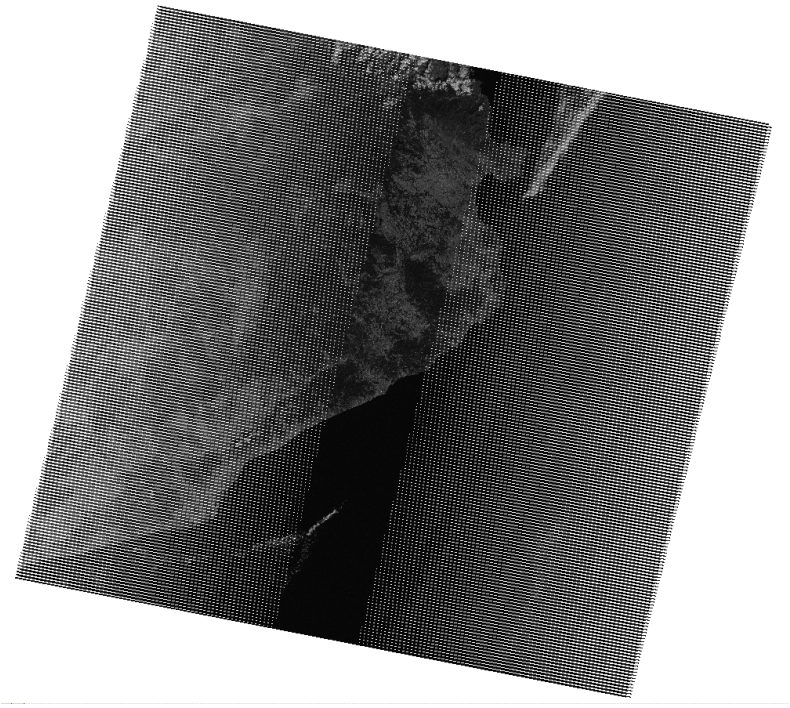

Figure 2.3: SLC-off effect in Landsat-7 EMT+ images

The consequence of the SLC failure (or SLC-off) is shown in figure 2.3; about 20% of the pixels in an ETM+ image are not scanned. Although there are gaps in the data coverage, with proper postprocessing the data remain of equivalent quality to pre-failure data.

[1] http://landsat.usgs.gov/

In the AfinaPC software, a mode for fine finding GCP correlations in images with the SLC mechanism broken (also known as SLC-off) has been implemented. The use of the SLC-off mode, called NODATA in the AfinaPC application, significantly increases the computational needs of the process, which can last more than twice the time to search for GCP in the affected areas, thus greatly unbalancing the computational needs for some GCP. This fact is specially relevant when it comes to select a proper thread scheduling algorithm, which could be critical in some parallel architectures like in CUDA GPUs.

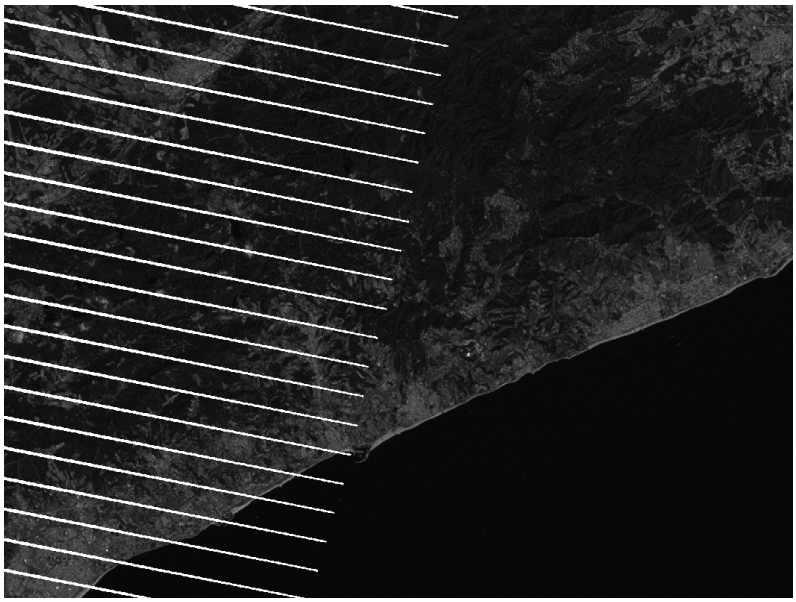

Figure 2.4: Detail of SLC-off effect in Landsat-7 EMT+ images

If the NODATA feature is not used in images with SLC-off, then all correlation windows where data is missing due to SLC-off are discarded, significantly hardening the GCP matching search and, in most cases, limiting the successful GCP search to restricted areas that might not be enough for a correct geometric correction. Please see figure 2.4 for a detailed zoomed view of how SLC-off impacts in the image information.

As an example of the impact of processing the areas with invalid data, the 4-thread parallel execution without NODATA takes about 900 seconds to run on this work's benchmarking reference pair of images but leads to zero GCP matches and, therefore, no valid calibration data. On the other hand, running with the NODATA mode (i.e. analysing areas affected by the SLC-off missing data issue) takes about 78% more (1600s aprox) but leads to valid results.

Chapter 3

Modern CPU architecture

Computers can be roughly divided in four different categories: (1) personal mobile devices (PMD), (2) desktops, (3) servers and clusters/warehouse-scale computers and (4) embedded systems. Parallelism at multiple levels is now the driving force of computer design across all four classes of computers, with energy and cost being the primary constraints. There are basically to kinds of parallelism in applications:

1. Data-Level Parallelism (DLP): this is when there are many data items that can be operated on at the same time.
2. Task-Level Parallelism (TLP): is when tasks of work are created that can operate independently and in parallel.

The Hennessy and Patterson Computer Architecture[6] book editions is a complete guide that has evolved over the time, giving a current and with perspective view of modern computer architectures. After industry left the single-core frequency ramp up a few years ago (around 2001) to focus in the multi-core architecture, the emphasis on instruction-level parallelism (ILP) started moving to the thread-level parallelism which lately has been enriched with enhanced data-level parallelism.

Even though we are still in the middle of the multi-core technology shift, cloud computing is showing up as what appears to be the next major step, leaded by the GRID in the scientific computing area. It is important to keep in mind the new paradigms whenever designing software.

The two kinds of application parallelism can be exploited by hardware in four major ways:

1. Instruction Level Parallelism: implemented in forms of pipelining and speculative execution. Present in microprocessor architectures since many years, long before dual core processors became a reality.

2. Request Level Parallelism: mainly exploited by the programmer or the operating system consists in running largely decoupled tasks in parallel (i.e. two different applications). This is the only parallelism available when running the serial version of AfinaPC, where parallelism could be achieved by running two different runs of the application in a multi-core processor.

3. Vector Architectures and GPUs: apply a single instruction to a collection of data in parallel. The use of GPUs within the AfinaPC application might lead to great performance improvements, further discussion about this comes in Chapter 4.

4. Thread Level Parallelism: exploits data-level and task-level parallelism in a tightly coupled hardware model, allowing interaction among parallel threads, usually in a shared memory space. Thread level parallelism has been widely exploited in this work's AfinaPC parallel implementation, for a deep analysis please refer to Chapter 4.

In the following sections an outline of the current CPU panorama for the two mainstream major desktop and server microprocessor vendors will be described.

3.1 Intel

In 2012 all laptop, desktop and server CPUs manufactured by Intel®provide multiple cores[7]. Laptop and desktop CPUs usually range from 2 to 4 cores while the server branch is packing from 2 (E3 series) to 10 cores (E7 series) in each processor. Even on the mobility market, with the Atom™family, Intel is implementing multiple cores in the same chip.

On top of multiple cores, Intel implements Hyper-Threading Technology (HT or HTT) in the cores of some of their CPUs. Intel's proprietary HT Technology[8] is used to improve parallelization of computations performed on PC microprocessors by doing multiple tasks at once. For each processor core that is physically present, the operating system addresses two virtual or logical cores, and shares the workload between them when possible. The main function of hyper-threading is to decrease the number of dependent instructions on the pipeline.

Hyper-threading works by duplicating certain sections of the processor -those that store the architectural state- but not duplicating the main execution resources. This allows a hyper-threading processor to appear as two "logical" processors to the host operating system, allowing the operating system to schedule two threads or processes simultaneously. When execution resources would not be used by the current task in a processor without hyper-threading, and especially when the

processor is stalled[1], a hyper-threading equipped processor can use those execution resources to execute another scheduled task.

With HT technologies the latest Intel CPU Xeon E7 series provide up to 20 threads to the system, that the OS usually represent as 20 cores in the server. Teaming up to eight of this processors in a 8-way socket leads to a server with up to 80 cores and 160 threads.

At the same time that core performance has increased thanks to improved instruction level parallelism and frequency, many efforts have been devoted to improve the memory bandwidth to keep pace and feed the execution units. According to David Kanter[10], the new Xeon family (Sandy Bridge) Intel is using an improved 3 level cache-coherent memory hierarchy similar to the one used in the previous core generation (Nehalem), with a core private L1 and L2 caches and a shared L3 cache. For more detail please see figure 3.1.

Intel recently announced new Intel Many Integrated Core (MIC)[9] architecture for highly-parallel workloads and general purpose, energy efficient TFLOPS performance. This new architecture is an extension of multi-cores focused on an increased number of cores while reducing the frequency of each core in order to be highly energy efficient. As shown in figure 3.2 with this new processor architecture Intel intends to greatly improve the performance per chip for highly parallel workloads. The system is designed so that current multi-core software tools and technologies can be reused with MIC, providing access to a cache coherent shared memory.

The first production MIC, the KnightsCorner is a 22nm +50 core energy efficient system that will provide about 1TFlop and have two different operating modes; as an offload co-processor or as a native Linux node programming that will be able to natively execute C/C++ and Fortran code compiled with the Intel compiler.

3.2 Advanced Micro Devices (AMD)

After the merger between AMD and ATI the company has worked in a product named accelerated processing unit (APU), a processor where some of the computing tasks originally done on the CPU moved to a shared in-chip GPU, which is better optimized for calculations such as floating-point unit calculations.

Bulldozer is the latest AMD core achitecture for server and desktop processors. As happens with all new Intel processors, the AMD Bulldozer based processors contain multiple cores; actually AMD processors include even more cores than the

[1] the processor may stall due to a cache miss, branch misprediction, or data dependency

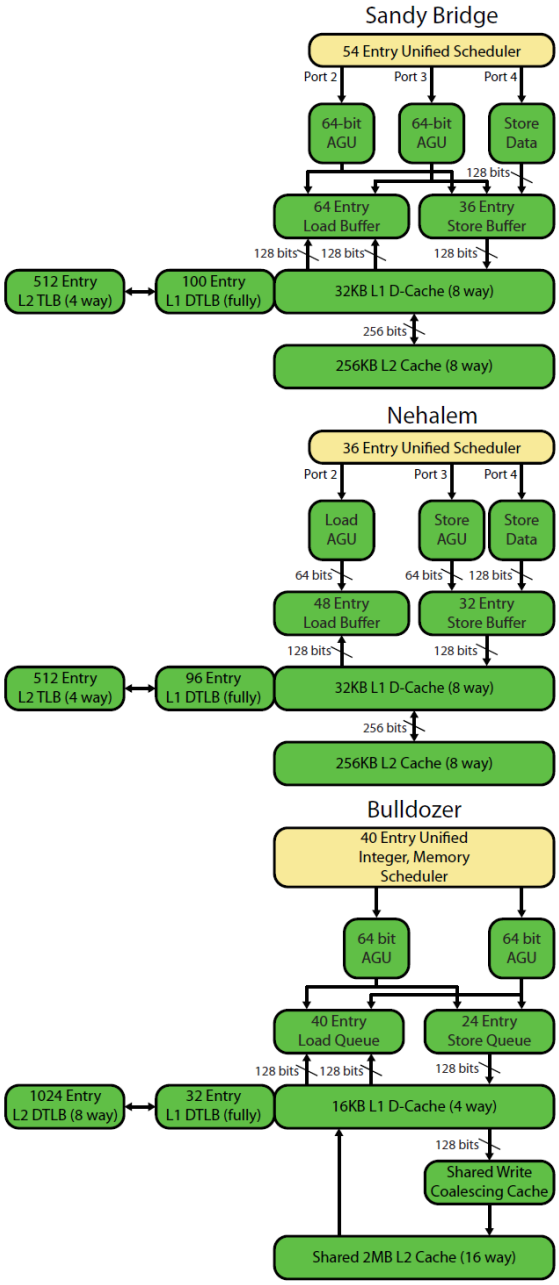

Figure 3.1: *Sandy Bridge* memory subsystem and comparison with Intel's previous generation *Nehalem* and AMD's *Bulldozer* core. Image from realworldtech.com[10]

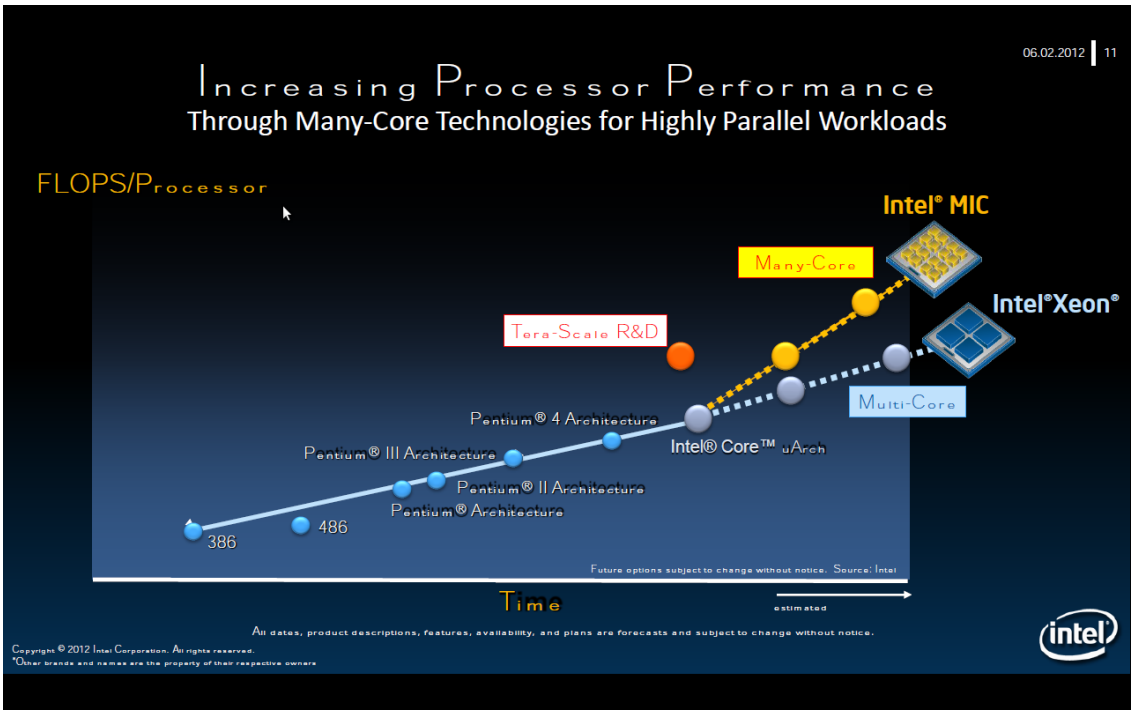

Figure 3.2: Intel processor performance estimated evolution

Intel's, scaling up to the 16 cores of the Interlagos CPU in the AMD Opteron^{TM}6200 series.

As shown in figure 3.1, and unlike Intel cores, the Bulldozer has a shared L2 cache. Despite of this memory hierarchy difference, the declared memory bandwidth for the AMD Bulldozer based Opteron 6200 family achieves the same 51.2GB/s as the Intel Sandy-bridge based E5 CPUs, more technical specifications about this and other processors is shown in Chapter 5.

Chapter 4

Parallelizing AfinaPC

Serial algorithms are one of the main burdens to achieve efficient parallel computations. Therefore a good understanding of the serial parts in a given piece of code sets the basis for performant parallel programming.

The AfinaPC program precises the recognition of homologue ground control points (GCP). The matching of the GCP is a crucial complex step which scales linearly with the number of GCPs. The application in charge of this process within the MiraMon toolkit is called AfinaPC. For further information on the AfinaPC internals please refer to Chapter 2.

4.1 Port to linux

The first step of the process has been porting the application from its original environment, a Windows 32 bit Borland C compiler, to a GCC free and open Linux friendly framework.

Moving the application from it's original Windows32 environment to a UNIX like has been also useful to make it more independent, removing many dependencies on the AfinaPC application with the rest of the MiraMon software tool-kit. This process eased the software profiling and optimization process but also restricted the amount of supported image formats, i.e. only decompressed images are supported in the ported version. The porting process was also necessary because only the AfinaPC code of the MiraMon code was available, as well as some other required functions upon request.

Having a UNIX capable software enables many possibilities like running it in non-proprietary OS (e.g. Linux) or in the GRID[16], where extensive resources are available to the scientific community, enabling the use of thousands of computers to do in a few minutes all the computing that otherwise could take weeks to do

with in-house resources.

The work required to eliminate Windows dependencies was reasonably simple since only a few windows library functions used to parse some *ini* files where used. Where some hard debugging and coding efforts necessary was to remove many of the existing dependencies with code from the MiraMon tool-kit that was not available for the project like message dictionary functions, metadata read functions or many *defines* and *typedef* which required several iterations with the original software developers at CREAF. All this work was also very useful to understand the internals of the application, easing the future analysis of parallelization alternatives.

Finally, after the serial code compiled and worked as expected, substantial performance differences between different GCC versions were observed. Taking as reference the original Windows 32 bit execution time, just recompiling with GCC in 64 bits improved approximately 10 times the execution, i.e. finding the matching GCP in the same pair of images, with the same search and calibration parameters, requires 10 times less computing time with the 64 bit GCC binary.

This work has been developed manly with GCC version *4.4.6 20110731 (Red Hat 4.4.6-3)*. For portability testing purposes and to see how more modern compiler can still improve CPU utilization, lowering execution time even more, a test with one of the CPUs (Intel i5-2400) was repeated with GCC version *4.6.3 (Ubuntu/Linaro 4.6.3-1ubuntu5)*. With the newer GCC version the execution speedup with regards to the original Windows32 execution time is approximately 12, this is about 20% improvement compared with the 4.4.6 GCC version, and this speedUP comes at zero cost; the very same code was used with both GCC versions, as well as the same optimization level (3).

4.2 Profiling serial AfinaPC implementation

After having the first operational serial version of the AfinaPC code, having studied it analytically and before starting any parallel implementation, a profiling task was carried out.

The main target when profiling the application is to determine in which functions and operations CPU time is spent mostly. Finding out the memory requirements, as well as the memory access pattern, is also very rellevant since it can represent additional constrains that should be taken in account when looking for a suitable hardware architecture, and/or parallelizing the application.

4.2.1 CPU utilization

Analysing the AfinaPC code it is possible to divide the application in three major stages:

1. Memory allocation and data load stage
   this includes reading the map images and some consistency checks on both the parameters and the images.

2. Ground Control Points (GCP) search
   this stage is mainly driven by a for loop that, for each given GCP, searches in two images (memory arrays) for the best correlation window. This search process consists of several loops that read data from the images, calculate correlations and, finally, compare and order the best matching positions using a quicksort.

   At the end of the main for loop the matched GCP; as well as the required displacement to fit the match, are saved in a unique array of successfully correlated GCPs.

3. Calculation of the global required orthorectification

   the global required orthorectification displacement is calculated using the desired mode (median or mode) taking the better fitting GCPs found in the previous stage. This is, the unique array filled at the end of the loop from the last stage is accessed and some calculations are performed.

The *gprof*[18] tool tells where a program spends its time and which functions called which other functions during the execution. This tool was used to obtain some empiric measurements profiling the AfinaPC application. The run-time figures that *gprof* provide are based on a sampling process, so they are subject to statistical inaccuracy. By contrast, the number-of-calls figures are derived by counting, not sampling. They are completely accurate and will not vary from run to run, if the profiled program is deterministic.

```
...
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 95.05     36.25    36.25  4438978     0.00     0.00  compara_correlacions
  4.93     38.13     1.88       13     0.14     2.93  SaltaDoc
  0.03     38.14     0.01  6959950     0.00     0.00  InterpolVeiMesProperByte
  0.00     38.14     0.00     2477     0.00     0.00  InterpolDoubleBilinealByte
..
```

Listing 4.1: AfinaPC serial execution gprof profiling with a reduced sample

Analysing the linux ported serial implementation running on a pair of test images with *gprof*, as shown in listing 4.1 clearly shows that there is a function called *compara_correlacions* (correlation comparison) which takes most of the execution time.

```
int compara_correlacions(const void *a, const void *b)
{
        if (((struct RESULTATS_CORRELACIO *)a)->correlacio < ((struct
            RESULTATS_CORRELACIO *)b)->correlacio)
        return 1;
        if (((struct RESULTATS_CORRELACIO *)a)->correlacio > ((struct
            RESULTATS_CORRELACIO *)b)->correlacio)
        return -1;
    return 0;
}
```

Listing 4.2: Most CPU consuming function from AfinaPC: compara_correlacions

Inspecting the code, as well as the function call graph generated by *gprof*, it is possible to see how the most CPU consuming, shown in listing 4.2, is a simple and not much optimizable function that is called many times from a quicksort function within the Ground Control Points (GCP) search main *for* loop.

There is also a piece of code at the end of the main *for* loop where the matched GCP as well as the required displacement to fit the match, are saved in a unique array of successfully correlated GCPs. This means that within the main loop there is an array that needs to be updated by all loop executions, and this update depends on former updates performed by the same loop. Fortunately the order in which the array is filled is not critical to obtain the proper AfinaPC results; thanks to this it is possible to overcome this potential parallelization difficulty by updating the array in an atomic way.

4.2.2 Memory Requirements

Memory requirements in AfinaPC are very dynamic and the best way to estimate them is with an analytic study of the source code, as well as some tests to verify the analysis.

```
valors_patro=calloc(dens_subpixel*dens_subpixel*mida_finestra_patro*
    mida_finestra_patro, sizeof(*valors_patro)))
copia_valors_patro=malloc(mida_finestra_patro*mida_finestra_patro*sizeof(*
    copia_valors_patro)))
sub_matriu_candidat=calloc(mida_finestra_patro*mida_finestra_patro, sizeof(*
    sub_matriu_candidat)))
valors_candidat=calloc(mida_finestra_candidat*mida_finestra_candidat, sizeof(*
    valors_candidat)))
```

Listing 4.3: Per thread major memory allocations

Memory needs depend on the selected images, the calibration window sizes and, in the parallelized case, also in the number of threads. Memory requirements for the serial part of the code are dominated by the memory allocated images; since images are loaded and stored in memory as a float array, the uncompressed images size should be taken as reference (instead of the compressed disk size). A rough approximation to the memory requirements of each thread is shown in listing 4.3. With the parameters used in our real case benchmarking test this means that each thread requires 2,47 MB[1].

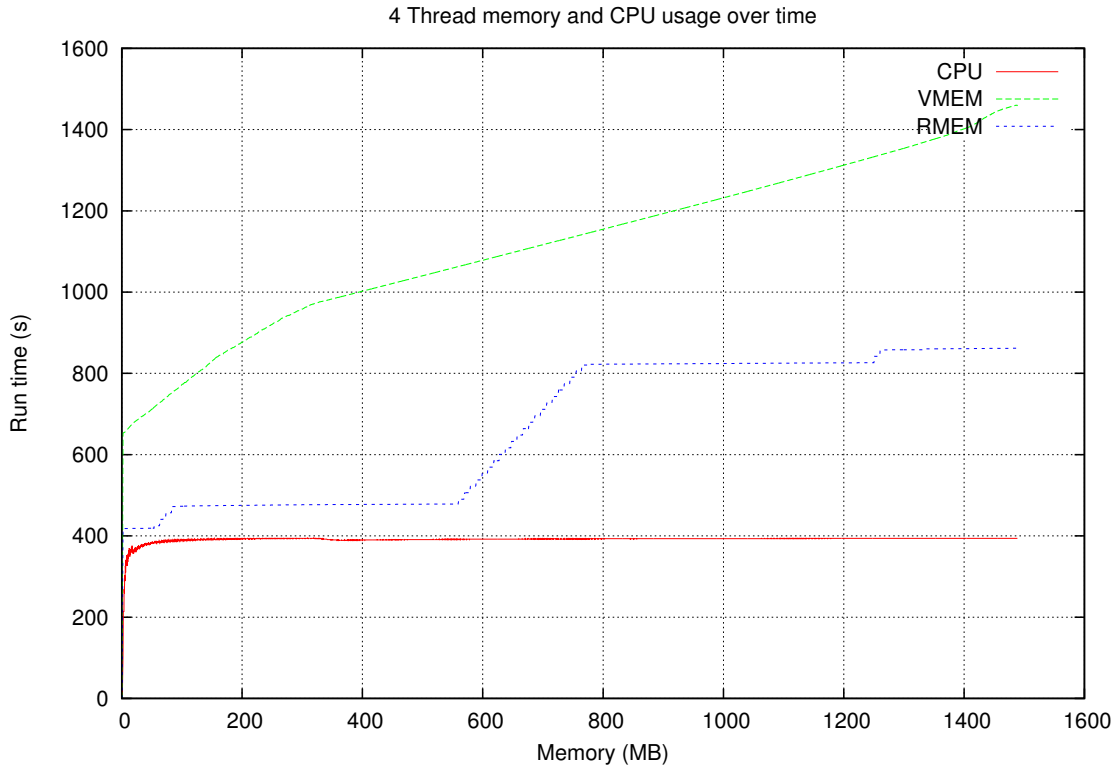

Figure 4.1: Physical memory, virtual memory and CPU usage by this work's OpenMP parallelized version of AfinaPC with the benchmarking real example on a quad-core server (2*2 AMD Opteron 290). When GCP matches are found one can see how RMEM (physical memory) increases. A CPU usage of 400 means that all 4 cores are fully used.

It is crucial to take in account that depending on if a correct match for the CGP if found or not, the physical memory required by the execution grows as the GCP search moves forward; this is because at each run of the main for loop (there is one run execution per GCP) results are stored in a dynamically allocated memory structure that needs up to 251000 bytes per GCP with standard search

[1] $(5 * 5 * 41 * 251 * 8) + (41 * 41 * 8) + (41 * 41 * 8) + (251 * 251 * 8) = 2589104$ bytes

parameters[2]. This means that, as shown in figure 4.1, if there are approximately 3500 GGP (like in the benchmarking real example) at the end of the execution the AfinaPC program memory requirements will grow up to 837,8 MB plus the size of the pattern and candidate image, which sum 414MB in this case. Therefore the final memory requirements for the benchmarking real example, if all GCP where properly matched would grow from an initial 414MB up to 1251MB just with a single thread, for a 64 thread execution up to 1409 MB of physical memory might be necessary if GCPs are found.

The total amount of virtual memory used by the task is proportional to the non-swaped physical memory requirements, it is roughly 60-70% greater than the physical memory used by the task all over the execution but, again, this depends on the number of properly matched GCPs. Virtual memory includes all code, data and shared libraries plus pages that have been swapped out and pages that have been mapped but not used.

4.3 Automated Paralelization

As an academic work, and due to the high complexity of the software (roughly 6500 lines), the first approach for parallelizing the software was tried using automated parallelization tools. This initial effort determined that none of the tested tools were able to provide real parallelization improvements with the linux ported serial version of AfinaPC.

Automating the parallelization of applications from the source code at a high level is still a challenge. Some of the tested tools were unable to work with C ISO/IEC 9899:1999 code enhacement (also known as C99), but the main issue for the many tested parallelization tools was the manual memory management[3] present in the serial version of the program. This is, the tools could not properly determine which variables should be private or shared neither could reserve memory (i.e. malloc et al.) for those that required so in private threads.

After a few unsuccessful attempts to obtain a parallelized version of AfinaPC using automated parallelization tools the efforts shifted to a manual approach, discussed in the following sections.

[2]It depends on two arguments of the application: dens_subpixel (5 in the benchmarking real example) and mida_finestra_candidat (251 in the benchmarking real example), the formula is: 200*dens_subpixel*dens_subpixel*mida_finestra_candidat

[3]by the use of the malloc, calloc, realloc and recalloc functions

4.4 MPI

In order to efficiently solve the problem presented by the AfinaPC algorithm, the main issue of MPI is the intensive data communication required by the application; at least one portion of the pattern and the candidate map image should be delivered to each MPI node through the network before any computing could start.

With current many-core architectures, scaling up to many tens of cores in the same shared memory space, it is possible to overcome the communication overheads while using a high number of cores to calibrate the same set of images; therefore the complexity of an MPI implementation of AfinaPC versus the potential obtained gains needs is analytically not convincing enough.

Under the need of calibrating a series of images, by using other parallelization techniques and a batch system one could beat the performance gains of an AfinaPC MPI implementation at a fraction of the cost for both the application (re)design and the computing platform infrastructure.

4.5 GPU - CUDA

Along with multi-core processors, we have seen the rise of many-core processors (having 32 or more cores) in the consumer market. GPUs from nVIDIA and AMD plus the forthcoming MIC line from Intel all fit this category.

The computational needs explored by the AfinaPC profiling show a great usage of decoupled tasks (each CGP search) acting in a shared memory area (the candidate and pattern images). This is a computing model that fits great with current GPU architectures although might present high coding cost of reimplementing part of the AfinaPC algorithm in CUDA[11] or a CUDA-like language. Reimplementing part of the code in a different language might compromise portability and be against a general trend of using highly standard programming languages to keep developers with a short learning curve.

An alternate option to recoding part of the AfinaPC algorithm could be to take advantage of already implemented GPU quicksort library[12] or any other efficient sorting algorithms for manycore GPUs[13]. In order to overcome some limitations inherent in the GPU architecture, like the memory limitations, an image partitioning system should be implemented so that only a restricted number of GCPs, allocated in certain GPU memory loaded area of the maps, are processed at a time; this could also show some throughput limitations that could become a potential bottleneck for the AfinaPC application performance.

Although taking advantage of GPU highly parallel architecture performance could be greatly beneficial, it would also add some hardware constrains in the platforms where the GPU (e.g. CUDA) parallel version of AfinaPC could run. Besides running in dedicated servers, AfinaPC has currently many users running the application in desktops, laptops and general purpose servers. AfinaPC requiring any specific GPU hardware would undesirably restrict the range of potential users and therefore should be avoided.

Another disadvantage that a GPU implementation might show is portability among GPU versions, which could restrict the efficient execution of the application to a narrow range of compatible GPUs. Also moving a GPU implementation to the GRID, as it is today, would be a challenge due to the lack of highly performant GPUs in most of the GRID computing elements.

4.6 OpenMP

About ten years ago, multi-core processors were just on the horizon. Today they are mainstream and have become the impetus for a revolution in computer programming that can make best use of the minimum two cores any new processor has up to the 16 cores the latest[4] AMD Opteron 6200 CPU series offer.

The OpenMP[14] Application Program Interface (API) supports multi-platform shared-memory parallel programming in C/C++ and Fortran on all architectures, including Unix platforms and Windows NT platforms. Jointly defined by a group of major computer hardware and software vendors, OpenMP is a portable, scalable model that gives shared-memory parallel programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer. The changes applied to the code while parallelizing AfinaPC under a Linux environment should be essentially the same for any other environment (e.g. Windows).

Another point in favour for parallelizing using OpenMP, instead of adapting the software to run in a GPU, is the incoming Intel MIC architecture that, as pointed in chapter 3.1, promises to provide many cores within a x86 software compatible model.

OpenMP simplifies the complex task of code parallelization, letting even beginners move gradually from serial programming styles to parallel programming. OpenMP extends serial code by using compiler directives. A programmer familiar with a language (such as C/C++) needs to learn only a small set of directives. Adding them doesn't change the serial code's logical behaviour. It tells the compiler

[4] as of June 2012 general market availability

only which piece of code to parallelize and how to do it, and the compiler handles the entire multi-threaded task. As opposed to a GPU implementation, OpenMP allows a wide range of compatible hardware; it just requires a shared memory multi-threaded architecture.

Message passing interfaces (e.g. MPI) use multiple processes that run in parallel, and communicate via special I/O channels; their memory is entirely separate and local to where the process runs, which is why that is called distributed-memory processing. On the other hand, multi-threaded shared memory programs (like OpenMP) have a single process, with mutiple threads of execution that run in parallel, and all the threads have access to all of the process's memory. In contrast with MPI potential implementations of the application, with OpenMP it is not necessary to specify all communication between the parallel tasks, which in return means a less invasive approach when parallelizing the application. This is simpler in some ways, but more complex in others.

4.6.1 Exploiting AfinaPC concurrency with OpenMP

When moving from the serial to the OpenMP parallel paradigm there are some aspects that must be taken into account, shared memory programming is generally not about the syntax and coding (i.e. getting a program to compile, link and run simple tests), but is far more about knowing what to do and what not to do. While moving AfinaPC to an OpenMP parallel implementation special efforts have been devoted in the following topics:

1. Memory handling in serial versus parallel regions of the application. In AfinaPC many arrays use dynamic memory allocation with malloc-like primitives, this implies that memory allocation for non-shared variables must be handled in the parallel region, as well as the memory freeing process.

   Is is also important to understand what the algorithm does in order to properly define the data sharing attributes[15] for the parallel region. AfinaPC uses many variables, with both static and dynamic memory allocation.

```
#pragma omp parallel default(shared) private(valors_patro,q,valors_candidat,
    stad_candidat,despXY,sub_matriu_candidat,i_banda,i_banda2, fil_col_PC,k,l,
    i,j,x,y,x0,y0,x2,y2) firstprivate(copia_valors_patro,nodata)
...
#pragma omp for schedule(runtime) lastprivate(resultats_reservats,stad_patro,
    p)
```

Listing 4.4: AfinaPC parallel region data sharing attributes

As one can see in listing 4.4 the data sharing attributes must be defined at the beginning of the parallel region, in the AfinaPC application all variables are shared by all threads by default, some are private which means that should be initialized in each thread independently and some others are firstprivate which means that they are independent in each thread but are initialized with the common value that comes from the serial region. More data sharing attributes are defined when the main for loop starts (in the parallel region), here lastprivate attributes are used so that the final value of the variable after the parallel loop matches the one from the last run of the loop in a serial equivalent execution.

2. Error handling required some adaptations within the parallel loop region. Due to the lack of synchronization in the OpenMP parallel execution it is necessary to adapt the code to properly throw an exception out of a parallel loop in OpenMP[5]. The code has been changed so that exceptions are not really thrown from within the parallel loop region anymore but avoid the loop execution for all threads and after it the exception is thrown. This need arised to properly handle memory allocation errors.

3. Some variable type issues arised with some size_t variables when compiling with the openmp flag under certain environments. This errors are fairly easy to solve by moving from size_t to int or any other "more standard" data type. This might also be a non-issue under certain environments, it's all about the compiler version and the libraries in the system. Catching this errors is not too difficult because AfinaPC never produces valid results when the issue arises.

4. Aliasing refers to the case when two variables overlap, either in whole or in part. The most common form is two names for the same location, but there are many others once one uses compound objects like C structures, which are widely used in the AfinaPC code. Aliasing bugs are illegal in serial code, but often show up only when the code is run in parallel.

   The number one approach for ensuring correctness is to avoid even correct aliasing as much as possible – i.e. avoid two threads accessing the same location, except when all of those accesses are read-only. To do this, minimise the update of global objects, which includes anything not passed as arguments; this includes global variables, any pointers and so on. And, most of all, never access anything both globally and via arguments unless you can guarantee

[5]http://www.thinkingparallel.com/2007/06/29/breaking-out-of-loops-in-openmp/

both accesses are read-only to the whole object.

5. Atomic operations. Atomic means that an action does not overlap with another atomic action; it does not always imply consistency, and the rules for how atomic actions interact with non-atomic ones are complicated. A data race is caused when two non-atomic actions overlap, or often when an atomic one overlaps with a non-atomic one. The effect is completely undefined, and often lead to invalid results.

   In AfinaPC initially an atomic approach for certain operations (e.g. global counter update) was tried but finally a critical area within the main parallel for loop was introduced. This is implemented with the *#pragma omp critical* OpenMP directive when updating the shared array of matched GCPs and since it is quite reduced does not significantly slow down the parallel execution of AfinaPC; figure 4.1 show a complete 4-thread execution of this work's OpenMP parallel implementation of AfinaPC and, after the initial data load stage, the CPU usage is almost 400 until the end, completely using all server's CPU cores. Without the critical area race conditions showed up whenever a GCP was found at the same time in different threads, this could not be easily spotted since many times this situation does not happen but whenever it does *despF* and *despC* arrays end up with inconsistent information that leads to incorrect calibrations.

6. The compiler version is relevant when compiling the AfinaPC code so that all used OpenMP directives and C primitives are supported. With GCC version 4.1.2 the code does not compile but it does in GCC version 4.4.5 and higher.

   In order to compile an application with the OpenMP directives taken into account the flag *-fopenmp* should be enabled. The same code without the flag will compile in most cases, but the OpenMP directives, also known and code-referred as pragmas, will not be taken into account and therefore the execution will be serial. For instance, the exact GCC options used to compile AfinaPC parallel implementation are: *gcc AfinaPC.c fcorauto-omp.c -o AfinaPC2-omp.exe -fopenmp -DOMP=2 -lm -O3*, without -fopenmp -DOMP=2 the serial version is obtained.

```
real 1624.27
user 6418.51
sys 1.36
AMB NODATA
[root@dc065 OrtoPNOA_15m]# gprof ../../code/AfinaPC2-omp.exe
Flat profile:
```

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ks/call  Ks/call  name
 65.44   4111.23  4111.23 99709368     0.00     0.00  CalculaCorrelacioAmbNodata
 34.52   6279.96  2168.74        1     2.17     6.28  moda_dwords
  0.05   6282.99     3.03     8916     0.00     0.00  InverMat2Xerraire
  0.00   6283.01     0.02     2314     0.00     0.00
      OmpleEstadistiquesNumeriquesFiltrantNoData
  0.00   6283.03     0.02                             CalculaCorrelacio
  0.00   6283.04     0.01   632593     0.00     0.00  compara_correlacions
  0.00   6283.05     0.01     3386     0.00     0.00  QuickSort
  0.00   6283.05     0.00    93921     0.00     0.00  Swap
  0.00   6283.05     0.00    11463     0.00     0.00  Partition3
  0.00   6283.05     0.00      178     0.00     0.00  EscriuCoordXYZEntitatVec
  0.00   6283.05     0.00       89     0.00     0.00  Qsort
  0.00   6283.05     0.00       13     0.00     0.00  SaltaDoc
...
```

Listing 4.5: AfinaPC execution time and gprof profiling with *NODATA* calibration

With the OpenMP parallel version of AfinaPC it is not possible to properly profile with *gprof* any more. In the code from listing 4.5 the *gprof* profiling output for a parallel execution can be analysed. Knowing the code and execution times of the parallel version it is impossible that the function *moda_dwords*, which runs in a serial section at the end of the application, can lead to a linear speedup if it really required almost 34% of the execution time.

A relevant performance aspect to take in account when optimizing OpenMP parallelized applications is the thread scheduling[17] (e.g. dynamic, auto, static, guided). Details depend on each compiler implementation but a general explanation of each thread scheduling technique can be checked at Wikipedia[6] or at the OpenMP consortium website[7].

Some OpenMP parameters can be defined at execution time, like the number of threads (*export OMP_NUM_THREADS=$nthreads*) and the thread scheduler (*export OMP_SCHEDULE="$schedule"*). A deep analysis on the performance of the OpenMP parallelized AfinaPC application is presented in the following chapter.

[6] `http://en.wikipedia.org/wiki/OpenMP`
[7] `http://www.openmp.org`

Chapter 5

Measured Results

This chapter gives an overview on the performance tests carried on in order to identify the best fit processor architecture for the parallelized AfinaPC application.

In total five different servers[1] have been chosen for the performance tests, mixing different architectures, vendors and generations in order to quantify the major differences when running the parallelized AfinaPC algorithm.

The same pattern and candidate image, as well as GCPs have been used in all execution runs. For each server all OpenMP built-in thread scheduling algorithms have been tested, as well as several granularity values for those algorithms supporting the feature. Each CPU as been tested with different degrees of parallelism, ranging from the single threaded execution up to four times the number of cores recognized by the server (usually it is the number of logic cores).

Both pattern and candidate images, as well as the AfinaPC correlation search parameters have been provided by CREAF. Although being a 100% real use case of AfinaPC, parameters and images have been selected so that the AfinaPC process requires high computing resources.

```
1 byte_197031_OrtoPNOA_15m.IMG byte_L72197031_03120110910_B70_HPFAv1.img
    MND_L72197031_03120110910_B70_Afina_Ajust.vec MND_resultat1.vec 41 251 0.6 5 2
    5 MND_resultat2.vec Equacions.ini 15
```

Listing 5.1: Parameters (arguments) used all along this work with the AfinaPC application.

The parameters shown by listing 5.1 have been used all along this work and have the following meaning:

[1] resources gently provided by Port d'Informació Científica (PIC) `http://www.pic.es`

Figure 5.1: Image used as pattern for the AfinaPC correlation search (byte_197031_OrtoPNOA_15m.IMG).

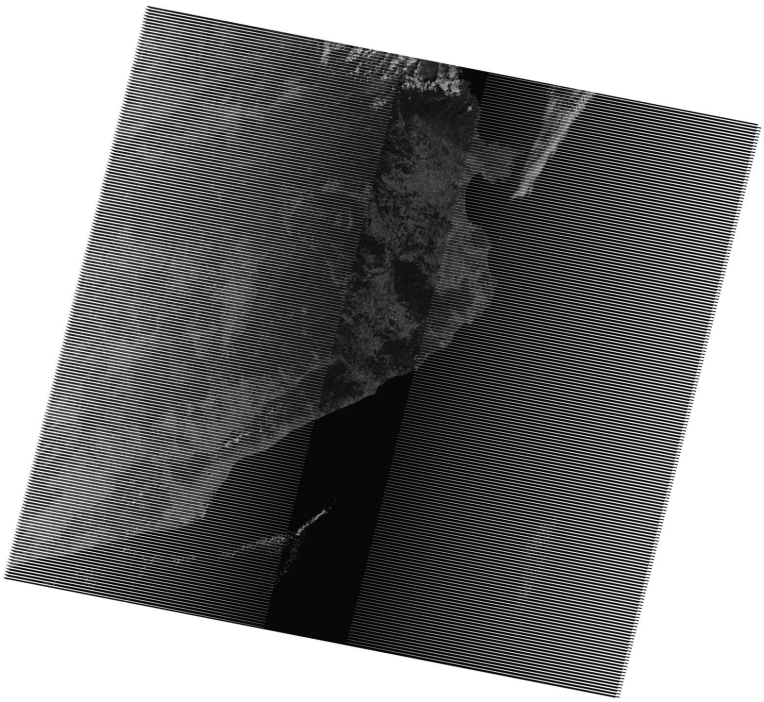

Figure 5.2: Image used as candidate (problem) for the AfinaPC correlation search (byte_L72197031_03120110910_B70_H

- Global required orthorectification shift is calculated using the median of the shifts from the matched GCPs.
- The 189MB image byte_197031_OrtoPNOA_15m.IMG is used as pattern image, please see figure 5.1 for its graphic representation.
- The 225MB image byte_L72197031_03120110910_B70_HPFAv1.img is used as candidate image, please see figure 5.2 for its graphic representation.
- The file MND_L72197031_03120110910_B70_Afina_Ajust.vec contains the location for 3338 GCPs.
- The original location (from the previous *.vec* file) of all matched GCPs will be stored in the file MND_resultat1.vec
- The pattern search window is 41*41 points.
- The candidate search window is 251*251 points, the pattern search window is moved within this window so it must be bigger.
- The minimum correlation for a certain GCP window to be taken as a match is 0.6 (it ranges from 0 to 1)
- The maximum number of possible match locations is 5
- The tolerance to group individual shifts is 2
- For the fine-grain GCP matching each position is subdivided in 5 pixels
- The file MND_resultat2.vec will be generated by AfinaPC and all GCPs found will be stored here together with it's position in the candidate image. This file is necessary to make a complete geometric correction of the candidate image.
- The file Equacions.ini contains the parameters of a first general correction done by a previous step with *TriaPC*[2].
- Size of the pixels for the candidate image (byte_L72197031_03120110910_B70_HPFAv1.img) is 15.

5.1 Performance metrics

In order to analyse the behaviour of each CPU under the wide range of test runs the following information has been collected for each execution:

[2]TriaPC is an application from the MiraMon software that generates GCPs for an image.

- Real execution time
  is the time that the execution of the application requires since the very same second of it's instantiation until it returns to shell.

- CPU time
  is the total amount of CPU time used by the application. If more than one CPU has been used, the CPU time contains the sum of the utilization time of all CPUs.

- System time
  is the total amount of CPU time used by system calls triggered from the application.

- SpeedUP vs single thread
  is the number of times the execution is faster with regard to the single threaded execution.

- SpeedUP vs original Win32 execution in i5-2400 [3]
  is the number of times the execution is faster compared to the single threaded execution of the original Windows 32 bit Borland compiled application running in the Intel i5-2400 CPU server described in section 5.3.

- Efficiency
  is a performance metric defined as SpeedUP divided by the number of cores or processors. It is a value, typically between zero and one, estimating how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization. Algorithms with linear speedup and algorithms running on a single processor have an efficiency of 1. In this work a similar figure is used where the SpeedUP is divided by the number of threads in order to better detect efficiency loss.

5.2 Benchmarking hardware

In order to have a wide point of view of the user experience with the parallelized version of AfinaPC, four different servers and one desktop computer have been used for the performance benchmarking. The following table contains a description of each one of this computers, identified by the most distinctive element, its processor (CPU):

[3] Due to technical and temporal limitations it has not been possible to perform measurements with the original binaries with each CPU architecture.
The Intel i5-2400 has been selected to run the original Windows binary because of it's modern microarchitecture and remarcable single core performance.

Processor	Intel i5-2400	Intel E5-2643	Intel L5630	AMD Opteron 290	AMD Opteron 6276
Server name	wl-gerard.pic.es	pbs04.pic.es	dc021.pic.es	dc065.pic.es	td801.pic.es
Product name	Dell OptiPlex 790	Dell PowerEdge R620	Supermicro X8DTH	Sun Fire X4500	Dell PowerEdge C6145
# of Processors	1	1	2	2	4
Total # of Cores	4	4	8	4	64
Total # of Threads	4	8	16	4	64
Memory type	DDR3-1333 (0.8ns)	DDR3-1600 (0.6ns)	DDR3-1066 (0.9ns)	DDR2-400 (5ns)	DDR3-1333 (0.8ns)
Memory size	8 GB	16GB	48GB	16GB	64GB
Used memory banks	2	2	6	8	16
ECC Memory	No	Yes	Yes	Yes	Yes
Operative System	Ubuntu 12.04	SL 6.1	SL 6.1	SL 6.1	SL 6.1

The information from the servers was collected via visual inspection as well as using the Linux *lshw* command on each server. More detailed information for each processor[4][5] can be found in the table below. Pricing information list prices[6] in 1000 unit sales. The maximum clock speed row is due to Intel's Turbo Boost or AMD's Turbo Core technologies which only apply under certain circumstances like when just one or a few cores are active.

Processor	Intel i5-2400	Intel E5-2643	Intel L5630	AMD Opteron 290	AMD Opteron 6276
Launch Date	Q1'11	Q1'12	Q1'10	2006	Q4'11
# of Cores	4	4	4	2	16
# of Threads	4	8	8	2	16
Clock Speed	3.1 Ghz	3.3 Ghz	2.13 Ghz	2.8 Ghz	2.3 Ghz
Max. Clock Speed	3.4 Ghz	3.5 Ghz	2.4 Ghz	-	3.2 Ghz
L1 Cache	256KiB	128KiB	256KiB	128KiB	768KiB
L2 Cache	1MiB	1MiB	1MiB	1MiB	16MiB
L3 Cache	6MiB	10MiB	12MiB	-	16MiB
Litography	32 nm	32 nm	32 nm	90 nm	32 nm
Max TDP	95W	130W	40W	95W	115W
Recommended price	284$	885$	551$	711$	788$
Memory Channels	2	4	3	2	4
Max Memory Bandwidth	21GB/s	51.2GB/s	25.6GB/s	6.4 GB/s	51.2GB/s

Processor	Intel i5-2400	Intel E5-2643	2x Intel L5630	2x AMD Opteron 290	4x AMD Opteron 6276
Launch Date	Q1'11	Q1'12	Q1'10	2006	Q4'11
Clock Speed	3.1 Ghz	3.3 Ghz	2.13 Ghz	2.8 Ghz	2.3 Ghz
Total # of Cores	4	4	8	4	64
Total # of Threads	4	8	16	4	64
Max Mem Bandwidth	21GB/s	51.2GB/s	25.6GB/s	6.4 GB/s	51.2GB/s

The following sections show some of the relevant performance metrics measured in the different servers (identified by their processor). As we go through the different CPUs the parallel implementation of AfinaPC is analysed, as well as some architectural specific remarks.

[4]Source for all Intel processors: `http://ark.intel.com`

[5]Sources for AMD processors: `http://www.amd.com/us/products/Pages/products.aspx`, `http://www.avadirect.com/product_print.asp?PRID=7599`, `http://www.cpu-world.com/CPUs/Bulldozer/AMD-Opteron\%206276.html`, `http://www.anandtech.com/show/5058/amds-opteron-interlagos-6200`

[6]Prices as of 14/6/12. Sources: Intel prices from `http://ark.intel.com`. AMD prices from `http://www.amd.com/us/products/pricing/Pages/server-opteron.aspx` and `http://www.dailytech.com/AMD+Cuts+Prices+on+Opteron+Processors/article8324.htm`

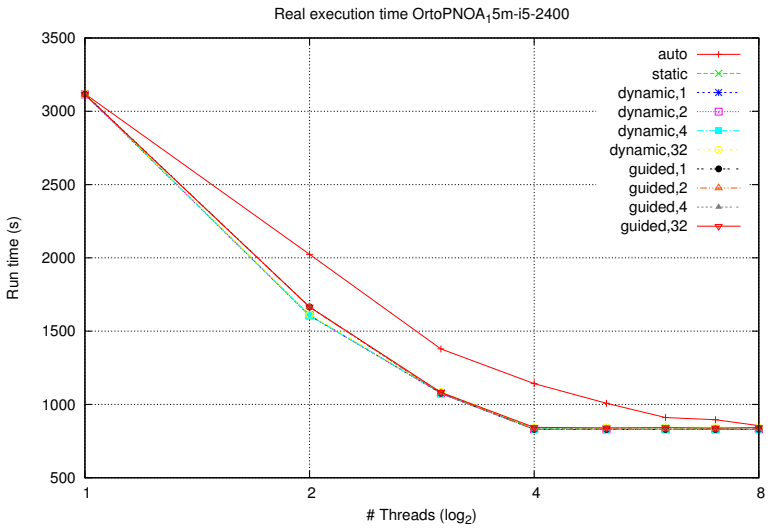

Figure 5.3: Real time used by a single execution of the multi-threaded AfinaPC on a Intel i5-2400 CPU using different OpenMP thread task scheduling techniques.

5.3 Intel Core i5-2400

This section focuses on the performance of a single-socket PC featuring the Intel i5-2400 processor, which is a desktop widely used mid-range CPU based in the SandyBridge Intel core architecture.

Figure 5.3 points out a common trend observed with all processors where consistent differences are observed using different OpenMP thread scheduling; surprisingly the auto scheduling is clearly worse than any other policy but it joins all others as many threads are used. The reason for this is that when more threads than cores are available, a non-efficient thread scheduling policy is masked by the OS task scheduler that gives CPU time to those threads in need while leaving the idle threads waiting.

Figure 5.4 shows an initial SpeedUP of 12 with just a single thread, which acts exactly like the non-multi-threaded version (i.e. the execution 64 bit Linux is 12 times faster than the original binary). As more threads are used the application scales almost linearly until the number of cores in the server (4) is matched.

GCC 4.6.3

The exact same OpenMP parallel implementation code was recompiled with a more modern version of GCC: 4.6.3. In every sense the behaviour is the same but, as can be observed in figure 5.5, the SpeedUP obtained with regards to the original execution using the newer GCC version is slightly higher. Although interesting and important to take in account, this shouldn't sound as breaking news since many groups around the world actively work to improve the GCC compiler at each

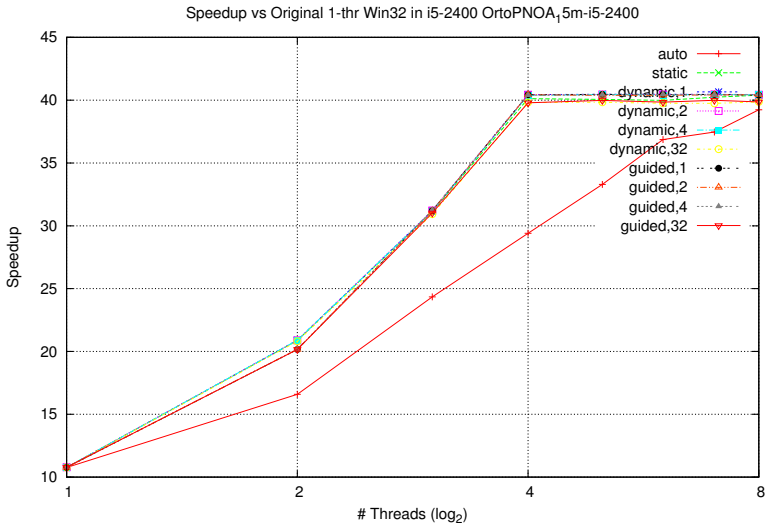

Figure 5.4: Real time speedUP of the Linux 64 bit GCC 4.4.6 compiled multithreaded AfinaPC implementation versus the Win32 borland C compiled original binary. The same Intel i5-2400 CPU server is used to execute the Win32 run.

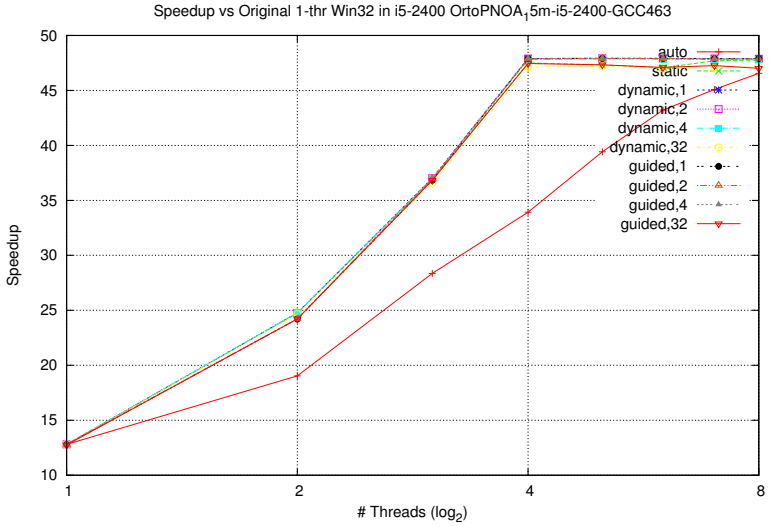

Figure 5.5: Real time speedUP of the Linux 64 bit GCC 4.6.3 compiled multi-threaded AfinaPC implementation versus the Win32 borland C compiled original binary. The same Intel i5-2400 CPU server is used to execute the Win32 run.

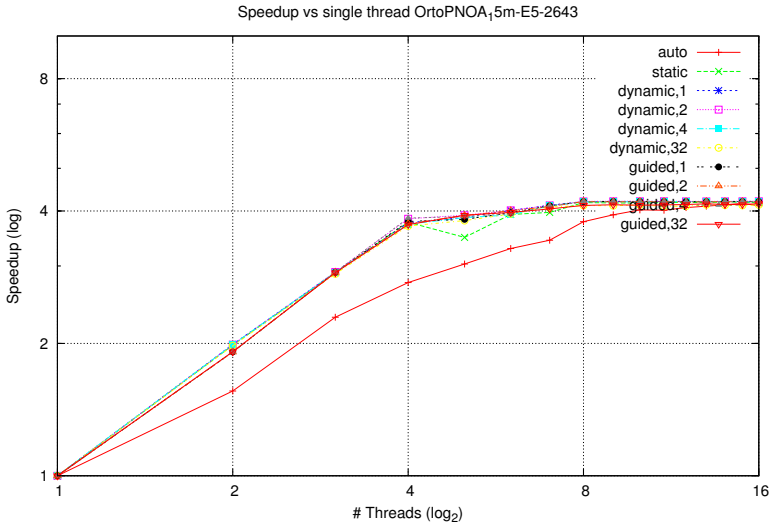

Figure 5.6: Real time speedUP of the multithreaded AfinaPC implementation versus single threaded execution in a single socket Intel Xeon E5-2643 server.

version and, as we can see, it is worth the effort.

5.4 Intel Xeon E5-2643

With the Intel E5-2643 another Sandy Bridge core based processor is evaluated, this time it is on the server segment. With regards to the i5-2400 in this CPU we've half of the L1 cache but more L3 cache, also the E5 doubles the maximum memory bandwidth and provides hyperthreading capabilities to the cores, so with 4 physical cores we get 8 logical cores/threads. In terms of clock speed, this E5 is slightly faster than the tested i5 (3.3 vs 3.1 base clock speed).

In figure 5.6 we can see how hyperthreading helps slightly improving performance after 4 threads, when the non-multi-threaded cores of the i5 could not provide any further benefits. This is, with the E5 we're obtaining a speedUP slightly better than 4, thus going beyond the number of cores in the processor; we get at a steady state at 8 threads though, when the number of logical cores is reached. Unlike the desktop series i5 CPU, the E5 family supports dual socket server setups, so it is possible to build a shared cache coherent memory system with twice the cores and that would potentially double the performance. Another advantage of using a processor from the server segment is the memory integrity check provided by ECC protected RAM.

5.5 Intel Xeon L5630

The latest Intel benchmarked in this work is another server segment processor, the Intel Xeon L5630. Although having a 32nm lithography this is an older CPU than

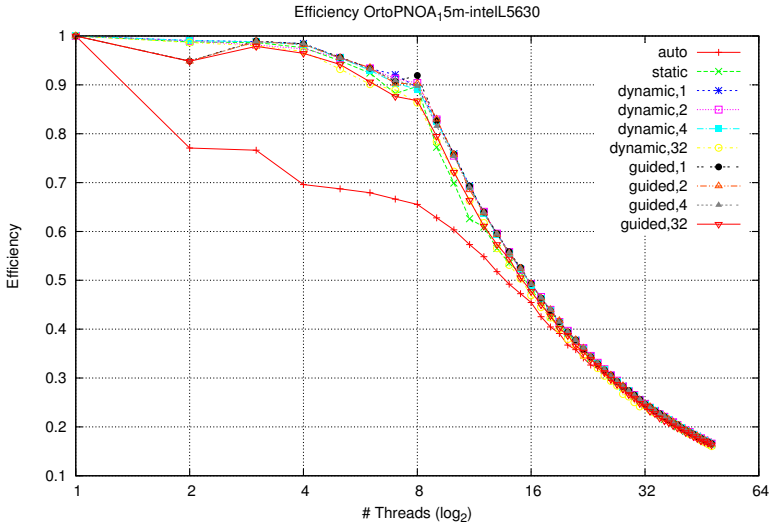

Figure 5.7: Efficiency of the multi-threaded AfinaPC implementation in a dual socket Intel Xeon L5630 processor server.

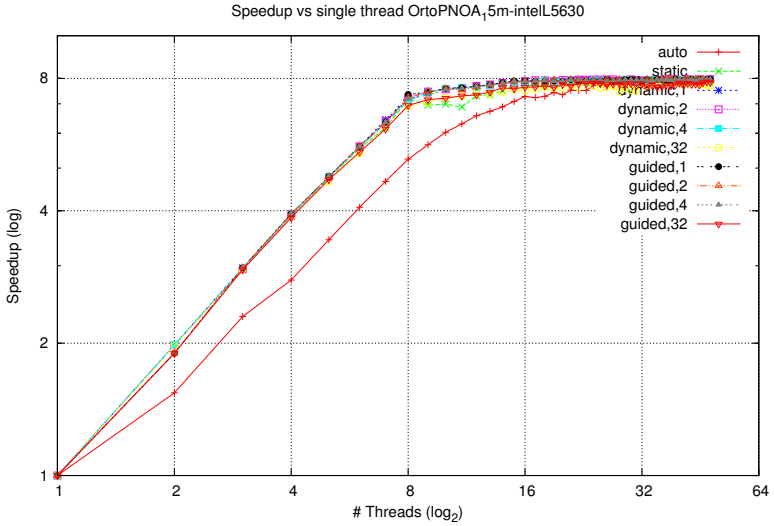

Figure 5.8: Real time speedUP of the multi-threaded AfinaPC execution in a dual socket Intel Xeon L5630 processor server.

the ones tested before, from Q1'2010, and it is not based in the Sandy Bridge core but in the previous Westmere (formerly Nehalem-C) generation.

In this occasion the server is under a dual socket setup, so 2 identical L5630 CPUs are used, providing a total of 8 physical cores. Memory banks are distributed across both CPUs and the Intel QuickPath Interconnect (QPI) is used when one processor needs to access a memory position stored by memory banks from the other processor.

Figure 5.7 shows how efficiency while only one CPU is used remains very close to 1, which is the ideal case. After 4 threads cores from the second CPU start working and we can see how efficency decreases down to 90% when all 8 cores are in use. This is mainly due to the higher memory latency bottleneck. Fortunatelly

the L5630 cores are multi-threaded, and this helps by hiding the memory latency when more threads are used; in figure 5.8 we can see that when 16 or more threads are used the speedUP versus the single threaded execution is 8, so all the physical cores are fully active.

5.6 AMD Opteron 290

The AMD Opteron 290 CPU is the older CPU analysed in this work. The server used for the benchmarking has a dual-socket setup so a total of 4 cores are available. As an older generation system, the RAM memory latency is significantly higher than in all other processors (5ns vs <1ns). Anyway it is interesting to have benchmarks using older hardware because not all potential users of AfinaPC have the latest processors in their production servers (or personal computers).

More details on the behaviour of this server are shown latter in this chapter, in the CPU measurement benchmarking comparison section.

5.7 AMD Opteron Model 6276

The AMD 6276 processor is a 16 Bulldozer based core Interlagos CPU. The server used for benchmarking has a quad-socket setup, so a total of 64 cores are available in the system. Unfortunately not all memory banks on the server are full, only half of them, so only 2 out of 4 memory channels are active and therefore the maximum memory bandwidth per processor is 25.6 GB/s instead of the nominal 51.2GB/s.

In figure 5.9 we can see now more behavioural differences in the thread scheduling decision than in any other architecture, this is due to the differences between the computing needs for the different GCP searches that unbalance the load, leading to some inefficient executions with certain schedulers when many cores are present but the number of threads is not big enough. As in all other cases, the best scheduler for the AfinaPC workload is *dynamic,1* and *guided,1*, which means that only one task is assigned at a time to each thread.

Looking at figure 5.10 we can see how an almost linear speedUP is achieved until 64 threads are used, then the speedUP keeps steady at almost 42, that is the 65% efficiency by figure 5.9, and means that what was shown as a tendency by other architectures with greater memory bandwidth per core; memory is not able to feed the cores with enough data.

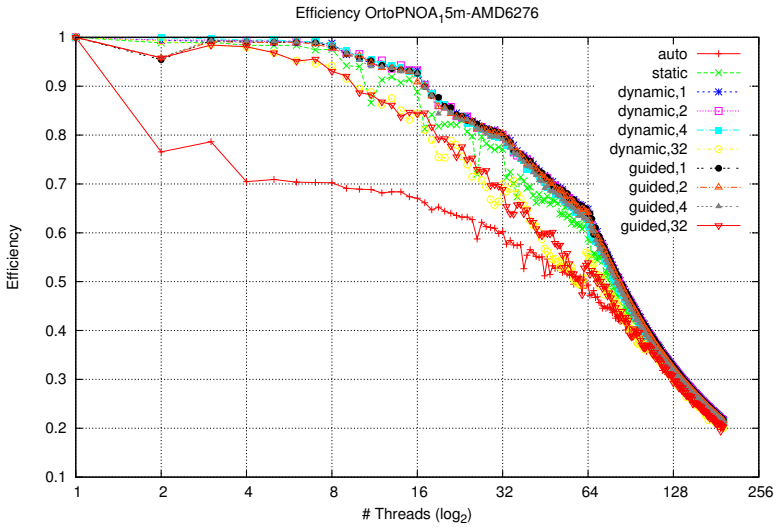

Figure 5.9: Efficiency of the multithreaded AfinaPC implementation in a 64 core quad socket AMD Opteron 6276 processor server.

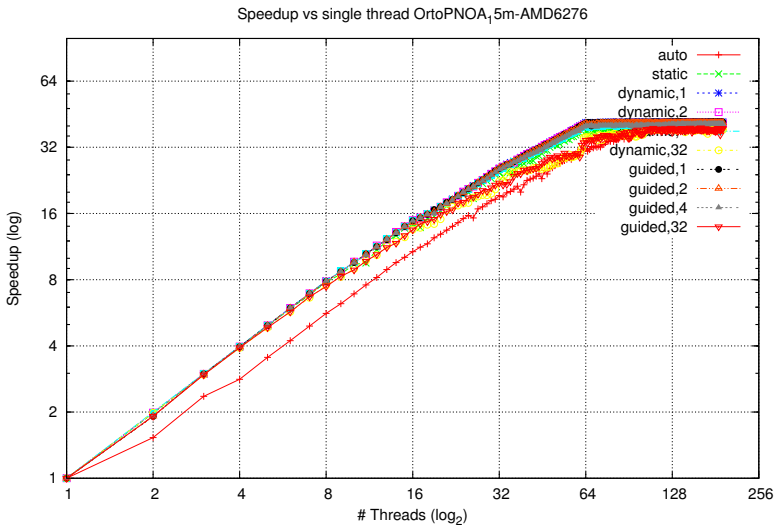

Figure 5.10: Real time speedUP of the multithreaded AfinaPC execution in a 64 core quad socket AMD Opteron 6276 processor server.

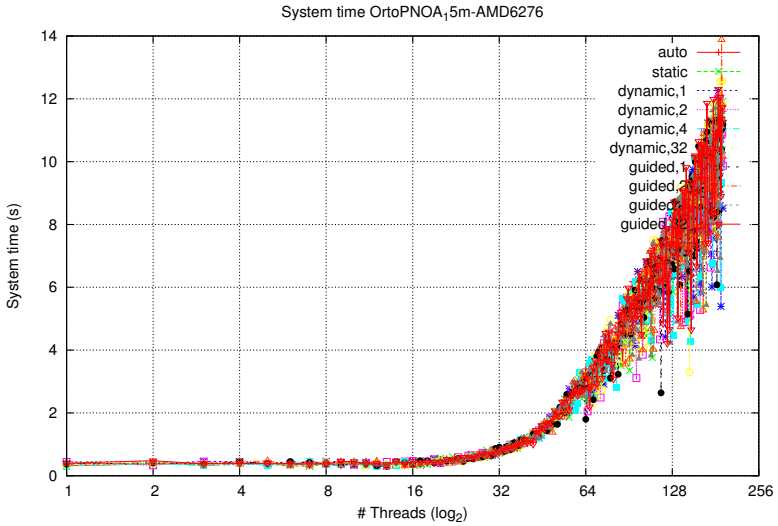

Figure 5.11: System time required by the multi-threaded AfinaPC execution in a 64 core quad socket AMD Opteron 6276 processor server.

Despite the observed hints it is not possible to be completely sure that the actual bottleneck is the memory bandwidth without a fine-grain profiling on the memory bandwidth requirements of the AfinaPC application and a deep analysis on the Bulldozer architecture. Each Interlagos processor offering 16 Bulldozer cores is actually composed by 8 dual threaded modules, this is clear when looking at the efficiency drop from 1 to 8 threads is about 0,005675 but from 8 to 9 cores it is four times greater (0,023213); the technology is completely different from the hyperthreading used by Intel's cores, though. Inspecting figure 5.10 we can see that when the maximum number of modules (32) is used, the speedup is about 26 and this is not consistent with Intel's hyperthreading effect when later we see that using 64 threads the speedup grows up to 42.

A similar inter-CPU communication issue like seen in the 2 socket Intel L5630 server is observed also in the quad-socket AMD 6276 server: when moving from 16 to 17 threads the efficiency loss is 0,028 while going from 17 to 18 cores we're just loosing 0,019.

After analysing the performance for the many studied architectures one could be mistakenly driven to think that the more threads used the better. Actually no performance improvement has been observed in any case after the number of logical cores (or threads) is reached and figure 5.11 shows that having more threads require more system CPU time, so keeping the number of threads in balance is definitely a good practice.

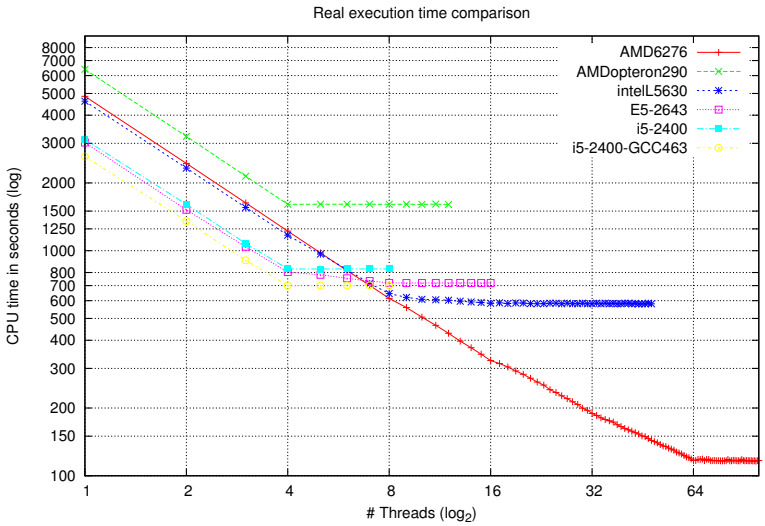

Figure 5.12: Real time execution comparison between the different benchmarked servers. This is the metric that the end user cares more about.

5.8 CPU benchmarking comparison

After looking at each system's behaviour it is time now to put all measurements together.

Figure 5.12 shows that as one would expect "bigger" and newer CPUs provide lower execution times, thus deliver greater performance. Looking at the single-threaded execution time we see how the AMD cores are significantly slower than the Intel counterparts. Although it is true that Intel cores are faster and the newer the better when we compare core by core performance, thanks to the parallelization of the AfinaPC application when more cores are active the situation changes, and the 64 core AMD 6276 based server, that was the second most slow processor with one thread ends up being significantly faster than any other. A similar situation happens with the octacore Intel L5630, despite being older than the Sandy Bridge based CPUs, it ends up beating them thanks to having twice the cores, which sum up more computing capacity with a lower energy footprint.

Even when the original binary has been executed in the second most powerful core available[7], figure 5.13 points out a great common fact in all architectures: the OpenMP multi-threaded Linux implementation greatly improves the performance of the original Windows 32 bit Borland C original binary,

The efficiency comparison presented by figure 5.14 is consistent with the memory bandwidth and processor capacities offered by the benchmarked computers. It is tightly related with figure 5.13 and demonstrates that even being more expensive, server oriented systems are in general better balanced, therefore more efficient in

[7]the i5-2400 server, slightly less powerful than the E5-2643

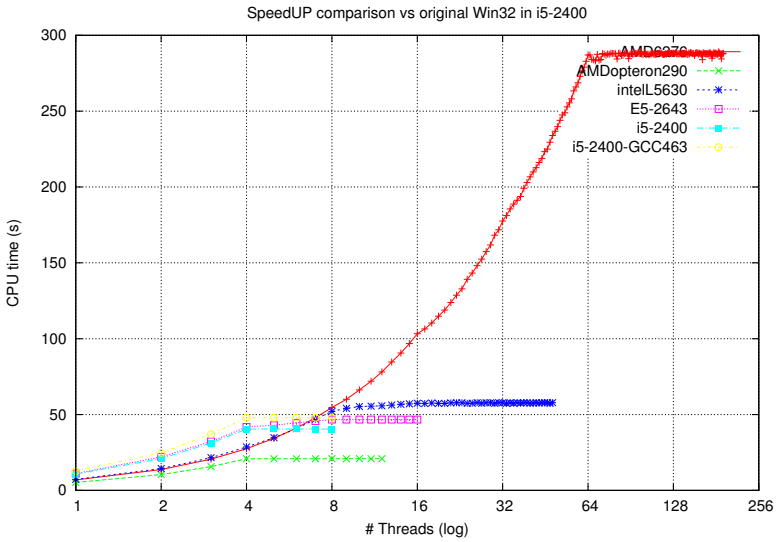

Figure 5.13: CPU speedUP comparison vs original AfinaPC execution in a Intel i5-2400@3.1Ghz

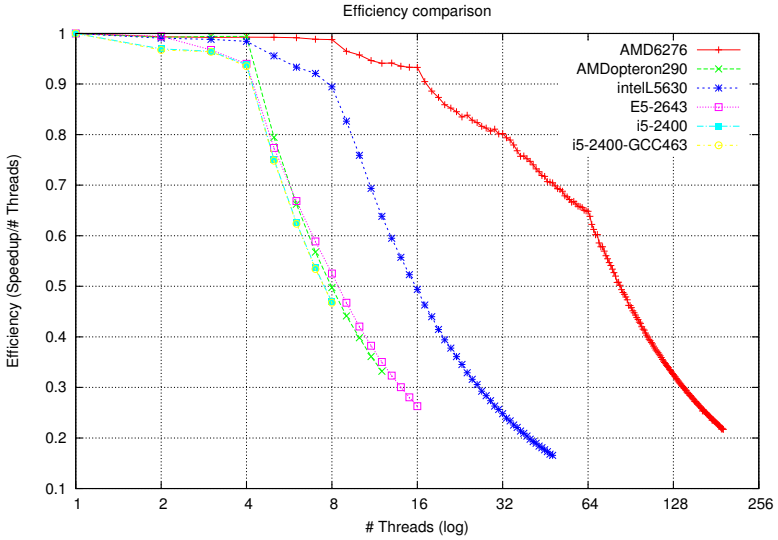

Figure 5.14: CPU efficiency comparison while increasing the number of parallel threads for the AfinaPC parallel execution.

computational and energy efficiency terms.

Chapter 6

Conclusions and future work

6.1 Conclusions

This work demonstrates how by using the OpenMP widely used parallelization API it is possible to obtain significant performance improvements on the AfinaPC application using standard computers that range from old (2006) to the latest multi-socket multi-core processor servers. The most time consuming application[1] of a remote sensing image geometric correction process has been parallelized, significantly reducing the time required to process the image as well as cutting the hardware cost and power needs; now it is possible to take advantage of all the cores in existing systems, this will slightly increase the power consumption of the servers, but the task finishes well in advance thus reducing the overall costs by increasing power, budget and time efficiency.

Guidelines to ease the parallelization of programs alike (i.e. from the MiraMon tool-kit) are provided in chapter 4; the efforts that trained developers should invest to parallelize the application following the suggested approach will be greatly rewarded. As shown in chapter 5, a significant acceleration of the AfinaPC application has been achieved, reaching a speedUP of almost 300 times with regards to the original binary, using the proper general purpose hardware architecture. In user measurable metrics this means that a process that usually takes over nine hours can now be completed in less than two minutes while keeping, if not improving, the accuracy[2].

As a matter of which is the most suitable hardware platform for an optimum execution of the parallelized AfinaPC application, it is highly dependant on the market trends, but in general it is possible to take advantage of multi-core architectures with low clock frequencies and high number of cores, which in return

[1] within the MiraMon software

[2] moving from the original 32 bits binary to 64 bits also improves floating point accuracy

lead to an energy efficient scenario.

On the desktop processor segment very good results have been obtained with mainstream processors, having very similar performance of equivalent server class CPUs at a fraction of the cost. On the other hand, ECC memory to ensure data integrity and the multi-socket feature required for higher scalability and lower execution times is compromised by the desktop segment.

The analysis in chapter 5 points out how the optimum number of threads should be equal or slightly higher than the number of logical cores in the system, not lower neither much higher in any case. A key issue for the AfinaPC application is the memory bandwidth and latency, therefore it is advisable to fill all RAM banks in those computers that should run the OpenMP parallel version of AfinaPC.

6.2 Future work

Should greater efficiency or greater execution time reductions be pursued, OmpP[19] and PAPI[20] performance analisys could still show some room for improvement in the algorithm, as well as demonstrate the conclusions from analysis about the detected bottlenecks in the studied systems from chapter 5.

Moving from Quicksort to a more efficient ordering algorithm would help in reducing the computational needs. Some OpenMP enhancements like transforming the OMP critical area to OMP atomic areas could also provide greater efficiency in the parallel execution.

On the hardware side, adding more memory to the servers, like the Quad-socket AMD 6200 hexacore used in the benchmarks, by filling all memory banks to use all memory channels, would increase effective memory bandwidth. This would probably enable greater efficiency than showed in the tests from section 5.7.

When the need for many geometric correction processes arises, like when analysing the evolution over the last 30 years of a designed area, the GRID or Cloud environments could provide a good solution as a scalable pay-as-you-grow model. The GRID approach could also benefit from the current research facilities with very little effort for massive calibration processes; the software could be used as is and just the workflows should be properly defined (e.g. where to store the images, how to submit the jobs for a rapid execution).

Due to lack of time, GPU parallelization techniques could not be developed as part of this work. The AfinaPC application shows potential for great speedUPs

using GPU architectures, I believe it is worth exploiting these technologies to lower execution times in a cost effective way.

Besides the AfinaPC application, some more components of the MiraMon software could take advantage of widely available multicore computer architecture.

Appendix A

Abbreviations

CREAF	Center for Ecological Research and Forestry Applications
CUDA	Compute Unified Device Architecture
HPC	High Performance Computing
GCP	Ground Control Point
GIS	Geographic Information Systems
CPU	Central Processing Unit
OGC	Open Geospatial Consortium
RAM	Random Access Memory
SIMD	Single Instruction Multiple Data
SLC-off	Scan Line Corrector off
USGS	United States Geological Survey
WPS	Web Processing Service

Bibliography

[1] Pons X., 2000. *MiraMon. Geographical Information System and Remote Sensing Software*, CREAF, ISBN: 84-931323-4-9. `http://www.creaf.uab.es/MiraMon`

[2] Toutin T., 2004 *Review article: Geometric processing of remote sensing images: models, algorithms and method*, International Journal of Remote Sensing. 25.10: 1893–1924.

[3] Pons X., Moré G. and Pesquer L., 2010, *Automatic matching of Landsat image series to high resolution orthorectified imagery*, Proc. ESA Living Planet Symposium SP-686

[4] Schut P. 2007, *OGC Web Processing Service (WPS), Version 1.0.0*, OGC 05-007r7: `http://portal.opengeospatial.org/files/?artifact_id=24151`

[5] Markham, B.L. December 2004. *Landsat sensor performance: history and current status. IEEE transactions on Geoscience and remote sensing. Web Page,* `http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1369366`

[6] John L. Hennesy, David A. Patterson. 2012. *Computer Architecture*. Elsevier, Fifth Edition

[7] *Intel Product Information. Web Page,* `http://ark.intel.com/`, Visited June 2012

[8] *Hyper-threading technology* - Wikipedia, the free encyclopedia. *Web Page,* `http://en.wikipedia.org/wiki/Hyper-threading`, Visited June 2012

[9] Cornelius, Herbert (speaker) (Intel Corp.). February 6th 2012. *Many-core technologies: The move to energy-efficient, high-throughput x86 computing (TFLOPS on a chip). Web Page,* `http://cdsweb.cern.ch/record/1421960`, CERN, Geneva (Switzerland)

[10] David Kanter. *Intel's Sandy Bridge Microarchitecture. Web Page,* `http://www.realworldtech.com/page.cfm?ArticleID=RWT091810191937&p=7`, Visited June 2012

[11] Shuai Che, Michael Boyer et al. October 2008. *A performance study of general-purpose applications on graphics processors using CUDA*. Elsevier Journal of Parallel and Distributed Computing Volume 68, Issue 10, October 2008, Pages 1370–1380

[12] Daniel Cederman and Philippas Tsigas. 2009. *GPU-Quicksort: A Practical Quicksort Algorithm for Graphics Processors*. In the ACM Journal of Experimental Algorithmics (JEA), Vol. 14, No. 4, ACM press 2009. *Web Page,* `http://www.cse.chalmers.se/research/group/dcs/gpuqsortdcs.html`

[13] Harris M., Garland M., May 2009. *Designing efficient sorting algorithms for manycore GPUs*. Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium.

[14] Dagum L., Menon R. 1998. *OpenMP: an industry standard API for shared-memory programming*. Computational Science & Engineering, IEEE. Jan-Mar 1998 Volume: 5 , Issue: 1 (Pages 46 - 55).

[15] Blaise Barney, LBNL. *OpenMP tutorial. Web Page,* `https://computing.llnl.gov/tutorials/openMP/`

[16] I.Foster, C.Kesselman, Morgan Kaufmann. 1998. *The grid: blueprint for a new computing infrastructure*. ISBN-13: 978-1558604759. Elsevier.

[17] JM Bull. 1999. *Measuring synchronisation and scheduling overheads in OpenMP. Web Page,* `http://cs.anu.edu.au/student/comp4300/labwork/bullewomp1999final.ps.gz`

[18] Susan L. Graham, Peter B. Kessler, Marshall K. Mckusick. 1982. *Gprof: A call graph execution profiler*. University of California. SIGPLAN '82 Proceedings of the 1982 SIGPLAN symposium on Compiler construction (pages 120-126).

[19] Karl Fürlinger, Michael Gerndt. 2008. *OmpP: A profiling tool for OpenMP*. Lecture Notes in Computer Science, 2008, Volume 4315/2008, 15-23, DOI: `10.1007/978-3-540-68555-5_2`

[20] S. Browne, J. Dongarra, N. Garner, G. Ho, P. Mucci. 2000. *A Portable Programming Interface for Performance Evaluation on Modern Processors*. International Journal of High Performance Computing Applications Fall 2000 vol. 14 no. 3 189-204