Scalable Computing:
Practice and Experience

Scientific International Journal
for Parallel and Distributed Computing

ISSN: 1895-1767

Universitatea de Vest din Timişoara

Volume 17(1) March 2016

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 17, Number 1, March 2016

TABLE OF CONTENTS

DOI 10.12694/scpe.v17i1.1145
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 1, pp. iii–iv. http://www.scpe.org

INTRODUCTION TO THE SPECIAL ISSUE ON HIGH PERFORMANCE COMPUTING SOLUTIONS FOR COMPLEX PROBLEMS

In the last decades, the complexity of the current and upcoming scientific/engineering problems has increased considerably. Computations involved in numerical simulations, molecular dynamics, computational fluid dynamics, bio-informatics, image processing, linear algebra, deep-learning, information retrieval, or big-data computing are just a few examples of such problems.

At the same time, improvements in high performance computing (HPC) systems are mainly associated with an important increasing in the complexity of computer architectures, making difficult the code optimization. This results in a greater gap between the general scientific / engineering user community (in need of easy access to efficient high performance computations) and the HPC programmers community (who design codes for narrow sub-classes of problems). The development of user-friendly codes for non-HPC-trained user community becomes a big challenge. As consequence, effective use of HPC centers requires specialized / individual training for each user group.

Today, programmers of HPC or/and scientific applications have to deal with numerous details regarding computer architectures at low level to take advantage of the last features of the current and upcoming computing systems. It makes difficult the efficient and optimized software development. Thus, strategies and tools that can help us to adapt our codes over different computing architectures is of vital importance. However, previously, we must know and identify what are most efficient programming strategies and architectonic features. This is a difficult task as both, strategies and features, depend on the particular problem to be dealt. Portability is other important issue today. Multiples kind of processors have arisen in the last years. Most of these current platforms use their own compilers, languages, etc., being a very complex task to implement portable codes.

This special issue provides several studies, which involve different applications and strategies to improve performance and to achieve better usage of current computing systems. It is composed by three works.

The work "Performance Optimizations for an Automatic Target Generation Process in Hyperspectral Analysis" by Fernando Sierra-Pajuelo, Abel Paz-Gallardo, and Antonio Plaza presents several optimizations for hyperspectral image processing algorithms intended to detect targets in hyperspectral images. The algorithm used is the automated target generation process (ATGP) and the optimizations comprise parallel versions of the algorithm developed using open multi-processing (OpenMP, including Intel Xeon Phi) and message passing interface (MPI).

The study "Using Computational Geometry to Improve Process Rescheduling on Round-Based Parallel Applications" by R. da Rosa-Righi, V. Magalhaes-Guerreiro, G. Rostirolla, V. Facco-Rodrigues, C. Andre da Costa, and L. Dagnino-Chiwiacowsky, proposes two novel heuristics applied to process rescheduling, named MigCube and MigHull, to choose the candidate processes for migration and their destination. Both heuristics consider the use of computational geometry for plotting computation, communication and migration costs metrics in a 3D graph without any user intervention.

Finally, the work "Many-Task Computing on Many-Core Architectures" by Pedro Valero-Lara, Poorima Nookala, Fernando L. Pelayo, Johan Jansoon, Serapheim Dimitropoulos, and Ioan Raicu, studies what are the Many-Task Computing (MTC) programming mechanisms to take advantages of the massively parallel features of current hardware accelerators for the particular target of MTC. Also, the hardware features of the two dominant many-core platforms (NVIDIAs GPUs and Intel Xeon Phi) are also analyzed for our specific framework. This study consisted of comparing the time consumed for computing in parallel several tasks one by one (the whole computational resources are used just to compute one task at a time) with the time consumed for computing in parallel the same set of tasks simultaneously (the whole computational resources are used for computing the set of tasks at very same time). Finally, both software-hardware scenarios were compared to identify the most relevant computer features in each of our many-core architectures.

We would like to thank the editorial board of SCPE and reviewers for their effort and time, which is very appreciated.

Dr. Pedro Valero Lara, The University of Manchester, UK.
Prof. Dr. Fernando L. Pelayo, University of Castilla-La Mancha, Spain.
Prof. Dr. Johan Jansson, Basque Center for Applied Mathematics (BCAM), Bilbao, Spain and KTH, Royal Institute of Technology, Stockholm, Sweden.

DOI 10.12694/scpe.v17i1.1146
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 1, pp. 1–11. http://www.scpe.org

PERFORMANCE OPTIMIZATIONS FOR AN AUTOMATIC TARGET GENERATION PROCESS IN HYPERSPECTRAL ANALYSIS

FERNANDO SIERRA-PAJUELO, ABEL PAZ-GALLARDO* AND ANTONIO PLAZA†

Abstract. Hyperspectral sensors acquire images with hundreds of spectral channels. These images have a lot of information in both spectral and spatial domain, and with this kind of information different research studies can be accomplished. In this work, we present several optimizations for hyperspectral image processing algorithms intended to detect targets in hyperspectral images. The hyperspectral image selected for our study was collected by the NASAs Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) over the World Trade Center (WTC) in New York, five days after September 11th attack. The algorithm used in our experiments is the automated target generation process (ATGP) and our optimizations comprise parallel versions of the algorithm developed using open multi-processing (OpenMP) and message passing interface (MPI). Our experiments indicate that the ATGP can be successfully implemented in parallel in multicore and cluster computing architectures, including Intel Xeon Phi.

Key words: Hyperspectral imaging, automatic target generation process (ATGP), open multi-processing (OpenMP), message passing interface (MPI), Xeon Phi

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction. Hyperspectral imaging [1] is concerned with the analysis and interpretation of spectra acquired form a given scene (or specific object) by an airborne or satellite sensor [2]. Instruments such as the Airborne Visible Infrared Imaging Spectrometer (AVIRIS) [3] are able to record the visible and near-infrared spectrum of the reflected light using 224 spectral bands. As shown in Fig. 1.1, the resulting "image cube" is a stack of images in which each pixel has an associated spectral signature or fingerprint that uniquely characterizes the underlying objects [4]. The resulting data volume typically comprises several GBs per flight [5].

The special properties of hyperspectral data have significantly expanded the domain of many analysis techniques, including (supervised and unsupervised) classification, spectral unmixing, compression, target, and anomaly detection [6, 7, 8, 9, 10]. Specifically, the automatic detection of targets and anomalies is highly relevant in many application domains, like fire control in forests or detect deposit of minerals, including those addressed in Fig. 1.2 [11, 12, 13].

The automatic detection of targets and anomalies in hyperspectral images is highly relevant in many applications and it is particularly important for defense and security applications [14, 15], as well as for rare mineral detection in geology [16] or location of infected trees in forestry. In this paper, we developed and compared several efficient parallel versions of the automatic target generation process (ATGP) algorithm [4]. This algorithm was designed to find spectral signatures with orthogonal projections. The considered method includes the spectral angle distance (SAD) and the parallel versions are developed with open multi-processing (OpenMP) and message passing interface (MPI). They are focused on identifying thermal hot spots in a complex urban background, using AVIRIS hyperspectral data collected over the World Trade Center in New York just five days after the terrorist attack of September 11th, 2001.

2. Methods. In this section, we will describe the target detection algorithm that will be efficiently implemented in parallel: the ATGP algorithm [4], it was created to find spectral signatures using orthogonal projections. The starting point of the algorithm is the brightest pixel in the image, similar to other existing measures, it is possible to use different starting points instead of the brightest pixel. But, in these cases, it has been experimentally verified that the pixel is always detected in a small number of iterations if not chosen as a point starting [17]. Therefore, it seems reasonable to use as a starting condition. Next, we show a detailed algorithmic description of the classical version of this algorithm. It begins by an orthogonal subspace projector specified by the following expression:

$$P_{\mathbf{U}}^{\perp} = \mathbf{I} - \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T \tag{2.1}$$

*Centro Extremeño de Tecnologías Avanzadas, Trujillo, Cáceres, Spain (fernando.sierra@externos.ciemat.es, abelfrancisco.paz@ ciemat.es).

†Hyperspectral Computing Laboratory, University of Extremadura, Politécnica de Cáceres, Cáceres, Spain (aplaza@unex.es)

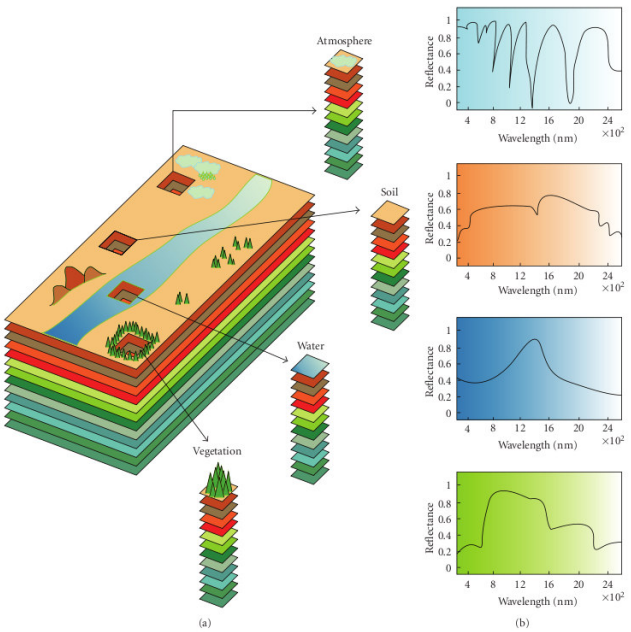

Fig. 1.1: Hyperspectral imaging concept. The reflectance and wavelength of pure pixels and mixed pixels are described in the figure. They are different and unique for each kind of pixel.

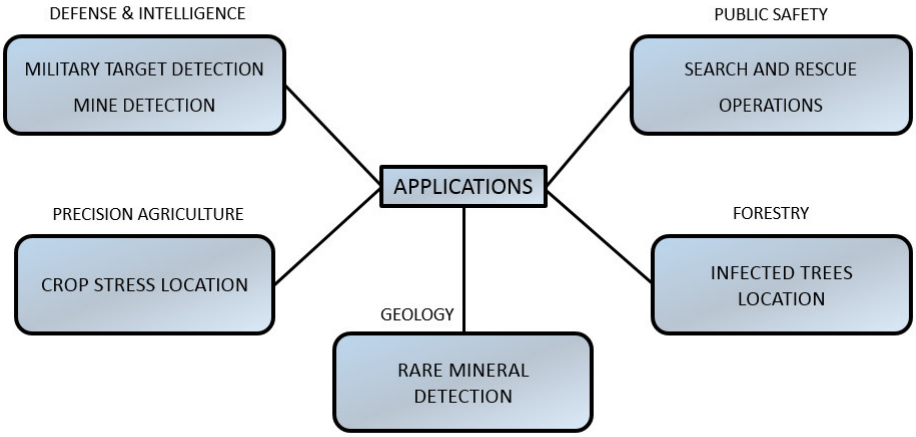

Fig. 1.2: Applications of target and anomaly detection. Detect targets in a war, humanity missions or you could use this to find mines and deactivate it.

where $\mathbf{U}$ is a matrix of spectral signatures, $\mathbf{U}^T$ is the transpose of the matrix and $\mathbf{I}$ is the identity matrix. ATGP algorithm uses the orthogonal projection of the equation 2.1 in each iteration to find a number of pixels or bands vectors from an initial pixel that is passed to the algorithm as ATGP value and which is usually the brightest pixel. This algorithm performs the following steps:

1. Calculate t_0, the brightest pixel of the hyperspectral image, using equation 2.2, where $\mathbf{F}_{(x,y)}$ is the pixel (vector) at coordinates (x, y) in the image. The brightest pixel is that with greater value performing

the vector product between the associated vector with that pixel and its transposed $\mathbf{F}(x,y)^T$.

$$\mathbf{t}_0 = arg\{max_{(x,y)}[\mathbf{F}(x,y)^T \cdot \mathbf{F}(x,y)]\} \tag{2.2}$$

2. Apply an orthogonal projection operator tagged as $P_{\mathbf{U}}^{\perp}$, using the expression 2.1, with $\mathbf{U} = \mathbf{t}_0$. This operator is applied to all pixels of the hyperspectral image.
3. Then, the algorithm finds a new target named as $\mathbf{t}_1$ with the greater value in the complementary space $< \mathbf{t}_0 >^{\perp}$, orthogonal to $\mathbf{t}_0$, using equation 2.3. In other words, the algorithm finds the pixel with higher orthogonality respect to $\mathbf{t}_0$.

$$\mathbf{t}_1 = arg\{max_{(x,y)}[P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]^T \cdot [P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]\} \tag{2.3}$$

4. The next step is to modify the $\mathbf{U}$ matrix and adding the new target found, that is $\mathbf{U} = [t_0 t_1]$.
5. The algorithm finds a new target named $\mathbf{t}_2$ with the highest complementary space $< \mathbf{t}_0, \mathbf{t}_1 >^{\perp}$, orthogonal to $\mathbf{t}_0$ and $\mathbf{t}_1$, using the expression 2.4. At this point, the orthogonal projector is based on a matrix $\mathbf{U} = [\mathbf{t}_0 \mathbf{t}_1]$ and the orthogonally concept is different.

$$\mathbf{t}_2 = arg\{max_{(x,y)}[P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]^T \cdot [P_{\mathbf{U}}^{\perp} \cdot \mathbf{F}(x,y)]\} \tag{2.4}$$

6. The process is repeated iteratively, to find a third target, $\mathbf{t}_3$, a fourth target $\mathbf{t}_4$, until a certain condition satisfies the termination for the algorithm. The termination condition considered in this paper is to achieve a number of targets p that is determined as an input parameter to the algorithm.

3. Parallel Implementations. Partitioning or data division prior to processing of the hyperspectral image can be done essentially by using two different strategies [18]:

- Spectral partitioning considers that different parallel architecture processors may contain non-overlapping parts of the same spectral signature (pixel). This schema has the disadvantage that, considering the spectral signature (vector) as a minimum unit for processing algorithms, it would be necessary to include more communication operations for each calculation of the metric that is used. From the viewpoint of the parallelization of the algorithm, which is based on applying repetitive computations, this type of partitioning means a huge cost in terms of communication operations. Clusters of computers are made up of different processing units interconnected via a communication network [19]. In previous works, it has been reported that data-parallel approaches, in which the hyperspectral data is partitioned among different processing units, are particularly effective for parallel processing in this type of high-performance computing systems [5, 20].
- Spatial partitioning considers that the same spectral signature or pixel cannot be partitioned in different units of the parallel processing architecture. We can work locally with the image on each processor, eliminating much of the communication load of the algorithm. In this way, we just need to make global communications to synchronize processes or get results in each iteration of the algorithm [25].

Our parallel implementation uses spatial partitioning so that each node carries a certain portion of the image, which can be managed easily indicating each participant node from where to start reading and the number of lines associated with the node. Besides, another reason for selecting spatial versus spectral partitioning is that, with spatial partitioning, each pixel vector (spectral signature) remains in the same processing unit. As the spectral signature is the minimum unit of computation in most hyperspectral imaging algorithms, keeping the spectral signatures in the same processor reduces drastically the amount of inter-processor communications. A comparison between spatial versus spectral partitioning for parallel hyperspectral algorithms has been reported in [21]. The parallelization by spatial decomposition adopted by our implementation is described graphically in Fig. 3.1. We opted for a spatial partitioning for that reason and other that we describe below:

- The spatial partitioning is a natural alternative to parallelize algorithms based on a processing by window, such as algorithms for detect anomalies and target detection.
- Another reason to select the spatial partitioning versus spectral is that the spatial partitioning allows to reduce communications between processors of the parallel architecture significantly. The most part of hyperspectral analysis algorithms consider the spectral signature as the minimum processing unit and, therefore, spectral partitioning involve inter-processor communications on pixel level increasing

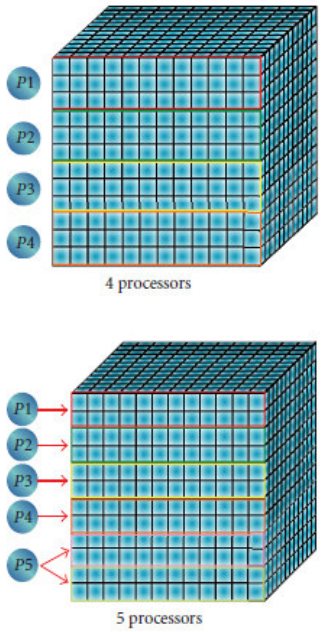

Fig. 3.1: Spatial-domain decomposition of a hyperspectral data set.

Table 4.1: Hyperspectral Image Features that we use.

Lines	614
Samples	512
Spectral Bands	224
Spectral Range	0,4 - 2,5 μm
Spatial Resolution	1.7 metres/pixel

the associated communications costs. The result of this is a lower parallel performance and scalability issues when the processor number increases.

- The last reason to select spatial partitioning is the ability to reuse code and improve the portability of parallel algorithms developed to different architectures. Is highly desirable to reuse serial code when we are developing parallel version due to the complexity of some analysis techniques. The spatial partitioning allows parallelism of fine-grained to make easier to use a parallel algorithm to different portions of data in which all spectral information is saved, allowing the transformation of the serial code to parallel code easier than applying a spectral partitioning.

This parallel scheme preserves in any case the sequential algorithm functionality of ATGP, except that in this case a matrix of intermediate values is calculated in each of the nodes and then an update is performed globally to share which node has the maximum value.

4. Experimental Results. In this section, we evaluate the parallel performance of the implementation introduced in the previous section.

Hyperspectral image considered. The image to be processed (AVIRIS World Trade Center) was taken five days after September 11th attacks. The main features of the image are in the Table 4.1. Note that the spatial resolution of the image is very high for what is usually in AVIRIS. This is because the image corresponds to a low altitude flight in which this flight pretended to obtain the highest possible spatial resolution. Therefore,the impact of mixed pixels is much smaller than we would expect and it is possible to do an study more focused on

Table 4.2: RGB Spectral Bands used for Fig. 4.1.

Color	Band
Red	1682
Green	1107
Blue	655

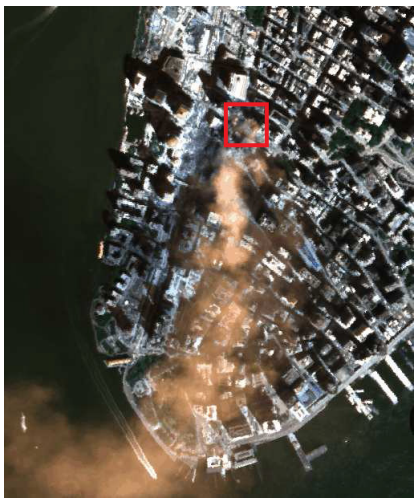

Fig. 4.1: WTC Hyperspectral image with RGB Colors.

detecting anomalies (fires).

We make a fake color composition in RGB with three bands (see Table 4.2) and we could see the vegetation in green color and the fires in gray hue. The smoke has the source in the red square (WTC Area) and its going to the south of the island, with a blue hue because the smoke has a high reflectance in the $655\mu m$ wavelength.

As we could see in the Fig. 4.1, it should be pointed out that the automatic detection of fires in the WTC is a very complex problem, due to the diversity of the urban environment in which fires are located. This complicates the discrimination between points of interest (fires) and background due to the complexity of the background, that has many different spectral substances as expected in a urban landscape.

4.1. Sequential. For comparison we use the ATGP sequential version and run this code on a computer with an 2x Intel Xeon processors model E5649 at 2.53 GHz with 6 cores and 24 GB of DDR3 memory. The experimental results show the time to load and process the image completely. We calculated the average of five executions for each result. The time spent by the sequential version of the algorithm in the considered platform was 18.42 seconds with a standard deviation of 0.21 seconds.

4.2. Performance optimizations with OpenMP. In this section, we evaluate the performance optimizations with OpenMP. The test-bed was performed in two different platforms:

- s6030: NUMA (Non-Uniform Memory Access) shared memory platform with 64 cores, 8x Intel Xeon X7550 at 2.00 GHz and 1 Terabyte of memory.
- Computational node: 2x Intel Xeon processors model E5649 at 2.53 GHz with 6 cores and 24 GB of DDR3 memory.

The most important part of the algorithm is the ATGP method. As we have seen in Section 2, it's a highly iterative algorithm with three *for* loops to scroll the image and perform operations. In these loops we will do the study using OpenMP. Before we start to get the optimal results, we have performed various tests to find the optimal state of OpenMP code.To make easier the understanding, we include 2 different pseudo-codes

Algorithm 1 ATGP algorithm pseudo-code

Input: ImageFile ImageHeader Targets

Read header information.
Read_Header(*ImageHeader*) →bands,lines,samples

Inicialize Matrix and Vectors
iVector[*targets*][2] ←0

Load image into an array
Load_Image(ImageFile, iVector)

Pixel more brilliant to start.
Get_Max_Bright() →iVector

Calculate targets
for *i* =0,*i* <targets **do**
  max_atgp =ATGP(iVector,lines,samples)
  Save the coordinates P[i][0] and P[i][1]
end for
Return the results P[targets][2]

Algorithm 2 ATGP method pseudo-code

Input: iVector imageLines imageSamples
Output: maxValue

Compare the image pixels
Where **n** and **n2** are values for lines and samples, respectively.
pragma omp parallel for num_threads(n) private(i)
for *i* =0,*i* <lines **do**
  pragma omp parallel for num_threads(n2) private(Values,k)
  for *k* =0,*k* <samples **do**
    Calculate de maximum distance
    for *j* =0,*j* <bands **do**
      Get vector for compare
      vector[*j*] ←iVector[j*lines]
    end for
    Calulate de Spectral Angle Distance
    maxValue ←Distance(vector[j],bands)
  end for
end for
Return maxValue

(Algorithm 1 and 2). It can help to reproduce the code. In Algorithm 2 we use two openmp *pragma* to distribute block of data on cores and nodes. While **n** is used to select the lines, **n2** is used for the samples. According to our experiments, the optimal **n** and **n2** are *16* and *32*, respectively.

After performing various executions we reached the optimal solution for each machine, s6030 and computational node. We run our optimized OpenMP code in the computational node with 2, 4 and 8 cores. We always try to use the thread-to-core binding with 1-1 ratio to be more efficient. If we evaluate the results obtained (see Table 4.3), we could see that the s6030 environment is faster than the computational environment, as Fig. 4.2 shows.

Table 4.3: Multicore scalability study with OpenMP code optimizations (Time in seconds) with five executions.In s6030 with same cores is faster because is a shared memory machine and OpenMP code take advantage of this.

	Computational Node	Speed-up	s6030	Speed-up
2 Cores	2,4868 s	7,4071	1,5246 s	12,0818
4 Cores	2,3790 s	7,7427	1,4953 s	12,3185
8 Cores	2,1709 s	8,4849	1,4926 s	12.3408

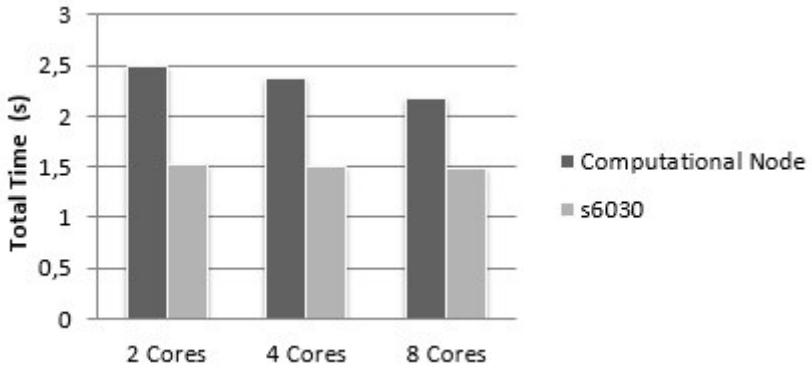

Fig. 4.2: Multicore scalability study in the two environments (Time in seconds) with five executions.It's clearly how the multicore scalability is faster in a shared memory machine (s6030).

Besides the ATGP method, we parallelized using OpenMP other loops into the code and we always make a study of the optimal number of threads for each loop. We are compiling it with using the icc compiler (version 14.0.2) with -openmp and -O3 flags.

The obtained results are expected, because the s6030 environment is a system designed for computing codes like this one. The next performance results were obtained using 16, 32 and 64 cores on s6030 environment and the best result were obtained using 64 cores (see Table 4.4), as we can see in Fig. 4.3.

4.3. Performance optimizations with MPI. We performed several optimizations in our sequential code using MPI. In this subsection, we considered up to 16 computational nodes composed by the resources previously described each, where we do many test until find the best option for this algorithm. For all executions we used the maximum time in each node and then calculate the average. We will select the worst time from each test to calculate the average time.

All our tests were conducted using MPI with 2, 4, 8 and 10 nodes and 1 thread in each node. Besides, we have tested using MPI with 12, 14 and 16 nodes, the maximum number of nodes that we could use at cluster, but the results that we obtained are worse and we see no need to include these in this results.

So we tried to find which are the best results by combining any number of nodes with any number of threads. Specifically, we calculated the results, and after different tests we could guarantee that the best result is achieved using one node and ten threads (see Table 4.5). In that case, we have tried various combinations with nodes and threads and we dont have used these results because the values obtained are worse than others.

We considered important to get two new times for the MPI implementation, in order to obtain a more reliable comparative evaluating the time spent in communication. We calculate the average send time (Broadcast) with the worst time that we get in total send communications. Moreover, we estimate the time of receive (Gather) and calculate the average using the worst time that we get in total receive communications.

We believe that the best result in that case is using one node because when running the code within the same node does not suffer delay by MPIBroadcast or MPIGather calls. We use MPI variables within the code

Table 4.4: Multicore scalability on a shared memory machine (Time in seconds) with five executions. The total time in a shared memory machine is improvement when we use more cores,the scalability is normal and the time improvement is small because the algorithm is highly iterative with data dependencies.

OpenMP	Total time(s)	Speed-up
16 Cores	1,4834	12,4174
32 Cores	1,4502	12,7016
64 Cores	1,3858	13,2919

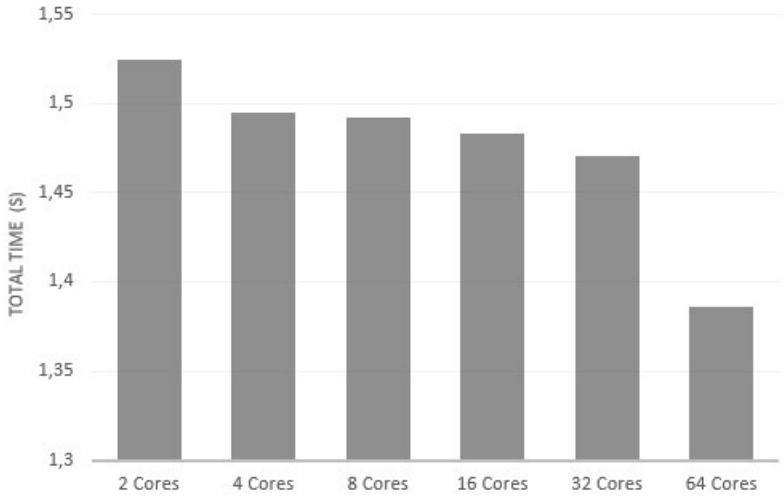

Fig. 4.3: Multicore scalability on a shared memory machine with five executions. The progress is bigger with 64cores because OpenMp use the strength of the shared memory machine.

and we are compiling it using the mpicc compiler(version 14.0.2) and -lpmi and -O3 compilation flags. It can be seen that the scalability study with the best results in Fig. 4.4. Also, we obtained send and receive times from all tests and compared them in Fig. 4.5.

As we could see, with 4, 8 and 10 nodes the results that we obtain are highly similar and this is because the ATGP algorithm is iterative and we think that we need to work with a larger volume of data. Maybe, the size of our problem (this particular hyperspectral image) is small for a such distributed memory platform.

4.4. Performance optimizations over Intel Xeon Phi™. We performed several optimizations in our OpenMP code for Xeon Phi [22, 23]. In this subsection, we considered an Intel Xeon Phi node:

- 2x Intel Xeon CPU E5-2620 v2 @ 2.10 GHz with 6 cores and 32 GB of DDR3 memory.
- Xeon Phi Card 5110P with 8 GB of DDR3 memory and 60 cores.

The performance of the OpenMP runtime can be essential for the overall scalability of OpenMP codes.

In the experiments using Intel Xeon Phi we could distinguish two modes [24]:

- *Offload Mode*: The application starts the execution on the host. As the computation proceeds it can decide to send data to the coprocessor and let that work on it and the host and the coprocessor may or may not work in parallel.
- *Native Mode*: This execution environment allows the users to view the coprocessor as another compute node. In order to run the code natively, an application has to be cross compiled for Phi operating environment.

Many experiments were done using OpenMP with Xeon Phi directives until finding the best option for this algorithm. All experiments were executed five times and the results are the average of all of them. The

Table 4.5: Total time using MPI in a cluster environment (Time in seconds) with five executions. Try to find the harmony between send time and receive time.We suspect that the best time is with 1Node and 10Threads because the receive time is minor.

MPI	Total Send (s)	Total Receive (s)	Total Time (s)	Speed-up
2 Nodes	0,0531	0,0404	4,0052	4,5990
4 Nodes	0,1050	0,0977	2,6626	6,9180
8 Nodes	0,1902	0,1645	2,5050	7,3532
10 nodes	0,1400	0,1161	2,3372	7,8812
1 Node 10 Threads	0,1558	0,0343	1,1002	16,7424

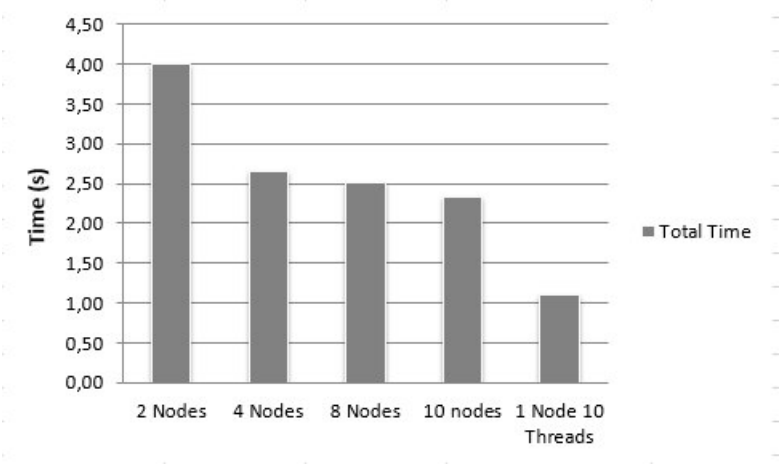

Fig. 4.4: Scalability study in a cluster environment using MPI with five executions. The total time is better in a single node because the communications spend a lot of time.

executions were done using Xeon Phi offload and native mode and results are shown in Table 4.6.

As we can see, using offload the results that we obtain are faster than using native mode. This is obvious because in native mode you need to communicate the results with the other coprocessor and work in parallel with the host, and for nested parallel regions, the overhead is much larger on the Xeon Phi system. We use OpenMP variables with Xeon Phi support within the code and we are compiling it using the mpicc compiler (version 14.0.2) and -mmic, -openmp and -O3 compilation flags.

5. Conclusions. In this paper we did several performance optimizations for the automatic target detection process (ATGP) algorithm for hyperspectral imaging. The results and parallel performance of the proposed parallel implementations, conducted using OpenMP and MPI [26], have been presented and thoroughly discussed in the context of a real defense and security application: the analysis of hyperspectral data collected by NASAs AVIRIS instrument over the World Trade Center (WTC) in New York, five days after the terrorist attacks that collapsed the two main towers in the WTC complex. From the results obtained we can conclude that the best performance is obtained with 1 node and 10 threads, using MPI. This is also the result that we expected because the ATGP algorithm is highly iterative and with high data dependency between iterations.

6. Future Research. Although the results reported in this work are very encouraging, further experiments should be conducted in order to increase the parallel performance of other versions of the proposed parallel algorithms and also optimizing the parallel design of the algorithms. We will do some research comparing this results with accelerators like GPUs (using CUDA).

In addition, we could use a heavier image and compare if the results over MPI are better than with the

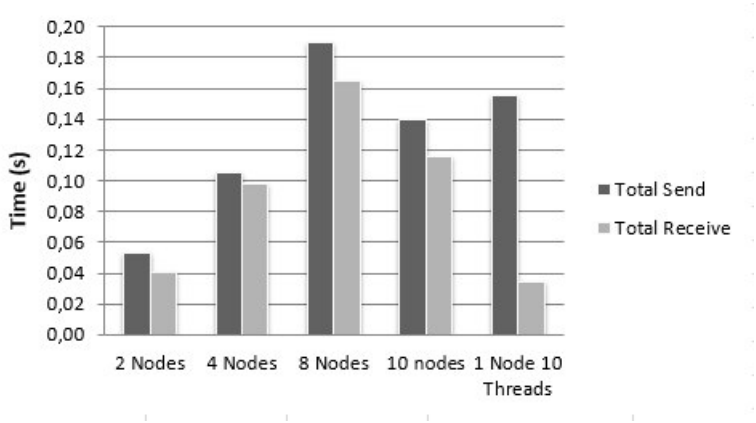

Fig. 4.5: Comparative between send and receive times in each test using MPI with five executions.In this figure we could see how the receive time is better in the same node and this is too relevant for the final total time.

Table 4.6: Total time using OpenMP over Intel Xeon Phi (Time in seconds) with five executions.

MODE	Total Time (s)	Speed-up
OFFLOAD	4,1322	4,4576
NATIVE	7,3886	2,4930

actual image.

Finally, new target detection and endmember extraction algorithms could be optimized and evaluated using different multi-core and many-core architectures.

Acknowledgment. This work was partially supported by the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

REFERENCES

[1] A. F. H. Goetz, G. Vane, J.E. Solomon and B.N. Rock,*Imaging spectrometry for Earth remote sensing*, Science 228, pp. 1147-1153, 1985.
[2] A. Plaza, J.A. Benediktsson, J.Boardman, J.Brazile, L.Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, J.Gualtieri, M. Marconcini, J.C Tilton and G. Trianni, *Recent advances in techniques for hyperspectral image processing*, Remote Sensing of Environment 113, pp. 110-122, 2009.
[3] R. O. Green, M.L Eastwwod, C.M. Sarture, T.G. Chrien, M. Aronsson, J.Chippendale, J.A. Faust, B.E. Pavri, C.J. Chovit, M. Solis, M.R. Olah and O. Williams, *Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)*, Remote Sensing of Environment 65, pp. 227-248, 1998.
[4] C.-I. Chang, *Hyperspectral imaging: Techniques for spectral detection and classification*, Kluwer Academic/Plenum Publishers: New York, 2003.
[5] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*, CRC Press: Boca Raton FL. Chapman & Hall, 2007.
[6] R. A. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing*, Academic Press: London, 1997.
[7] D. A. Landgrebe, *Signal Theory Methods in Multispectral Remote Sensing*, John Wiley & Sons, New York, NY, USA, 2003.
[8] J. A. Richards and X. Jia, *Remote Sensing Digital Image Analysis: An Introduction*, Springer, 2006.
[9] C.-I.Chang, *Recent Advances in Hyperspectral Signal and Image Processing*, John Wiley & Sons, 2007.
[10] C.-I.Chang, *Hyperspectral Data Exploitation: Theory and Applications*, John Wiley & Sons, 2007.
[11] C.-I. Chang and H. Ren, *An experiment-based quantitative and comparative analysis of target detection and image classification algorithms for hyperspectral imagery*, IEEE Transactions on Geoscience and Remote Sensing 38.2, pp. 1044-1063, 2000.

[12] H. Ren and C-I.Chang, *Automatic Spectral Target Recognition in Hyperespectral Imagery*, IEEE Transactions on Aerospace and Electronic Systems 39.4, pp. 1232-1249, 2003.
[13] D. Manolakis, D. Marden and G. A. Shaw, *Hyperspectral image processing for automatic target detection applications*, MIT Lincoln Laboratory 14, pp. 79-116, 2003.
[14] A. Paz, A. Plaza and S. Blazquez, *Parallel Implementation of Target Detection Algorithms for Hyperspectral Imagery*, International Geoscience and Remote Sensing Symposium 2, pp. 589-592, 2008.
[15] Y. Tarabalka, T.V. Haavardsholm, I. Kasen and T.Skauli, *Real-time anomaly detection in hyperspectral images using multivariate normal mixture models and gpu processing*, Journal of Real-Time Image Processing 4, pp. 1-14, 2009.
[16] S. Sanchez, A.Plaza, G.Martin and A.Plaza, *Parallel Unmixing of Remotely Sensed Hyperspectral Images on Commodity Graphics Processing Units*, Concurrency and Computation: Practice & Experience 23.13, pp. 1538-1557, 2011.
[17] A. Plaza and C.-I. Chang, *Impact of initialization on design of endmember extraction algorithms*, IEEE Transactions on Geoscience and Remote Sensing 43.11, pp. 3397-3407, 2006.
[18] A. Plaza, J. Plaza, A. Paz and S. Sanchez, *Parallel Hyperspectral Image and Signal Processing*, IEEE Signal Processing Magazine 28, pp. 119-126, 2011.
[19] R. Brightwell, L.Fisk, D. Greenberg, T. Hudson, M.Levenhagen, A. Maccabe and R. Riesen, *Massively parallel computing using commodity components*, Parallel Computing 26, pp. 243-266. 2000.
[20] A.Plaza, J.Plaza and D.Valencia, *Impact of Platform Heterogeneity on the Design of Parallel Algorithms for Morphological Processing of High-Dimensional Image Data*, The Journal of Supercomputing 40.1, pp. 81-107, 2007.
[21] A.Plaza, D.Valencia, J.Plaza and P.Martinez, *Commodity Cluster-Based Parallel Processing of Hyperspectral Imagery*, Journal of Parallel and Distributed Computing 66.3, pp. 345-358, 2006.
[22] Schmidl et al. ,*Assessing the Performance of OpenMP Programs on the Intel Xeon Phi*, Euro-Par 2013 Parallel Processing, p.12, 2013.
[23] C. Heirman et al., *Automatic SMT threading for OpenMP applications on the Intel Xeon Phi co-processor*, 4th International Workshop on Runtime and Operating Systems for Supercomputers, Article No.7, ACM New York, NY, USA, 2014.
[24] M. Koesterke et al., *Tutorial: Programming for the Intel Xeon Phi (MIC)*, IEEE Cluster 2013.
[25] A. Paz and A. Plaza, *Clusters vs. GPUs for Parallel Automatic Target Detection in Hyperspectral Images*, EURASIP Journal of Advances in Signal Processing, 2010, Article ID 915639, 18 pages doi:10.1155/2010/915639.
[26] Y. He and Q.Ding, *MPI and OpenMP Paradigms on Cluster of SMP Architectures: The Vacancy Tracking Algorithm for Multi-Dimensional Array Transposition*, Scalable Computing: Practice and Experience, Volume 5, No 2, 2002.

Edited by: Pedro Valero Lara
Received: Sept 28, 2015
Accepted: Febr 25, 2016

DOI 10.12694/scpe.v17i1.1147
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 1, pp. 13–32. http://www.scpe.org

USING COMPUTATIONAL GEOMETRY TO IMPROVE PROCESS RESCHEDULING ON ROUND-BASED PARALLEL APPLICATIONS

RODRIGO DA ROSA RIGHI, VLADIMIR MAGALHÃES GUERREIRO, GUSTAVO ROSTIROLLA, VINICIUS FACCO RODRIGUES, CRISTIANO ANDRÉ DA COSTA AND LEONARDO DAGNINO CHIWIACOWSKY *

Abstract. Process rescheduling is a known technique to face with system heterogeneity and dynamism, being especially pertinent on Bulk Synchronous Parallel (BSP) programs. These programs are organized in a set of round-based supersteps, in which the slowest process determines the moment of synchronization. This approach motivated us to develop a first model called MigBSP, which combines computation, communication and migration costs metrics for process rescheduling decisions. MigBSP originally employed an heuristic that could select either a single or a collection of process to migrate at each load balancing invocation. The first proposal is not reactive, so you should manually setup a percentage of processes to be migrated as input parameter for the load balancing model. In this work, two novel heuristics, named MigCube and MigHull, are proposed to choose the candidate processes for migration and their destination. Both heuristics consider the use of computational geometry for plotting computation, communication and migration costs metrics in a 3D graph, so both 'which' and 'where' load balancing questions can be answered without any user intervention. We believe that the contribution is not only in the MigBSP landscape, but also for the BSP community, who is trying to enhance performance in round-based applications in an effortless way. In addition to the description of MigCube and MigHull, this article also presents their evaluations with performance gains of up to 42% when enabling process migration over a subset of the Grid5000 infrastructure.

Key words: Computational Geometry, Process Migration, Performance, Dynamism, Grid Computing

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction. Process migration is a useful mechanism to offer runtime load balancing, mainly in dynamic, complex and heterogeneous environments. Generally, process migration requires explicit rescheduling calls within the application [11]. A different migration approach happens at middleware level, where changes in the application code and previous knowledge about the system are usually not required. Considering this, we have developed a process rescheduling model for grid computing architectures called MigBSP [28]. We decided to work with round-based applications, such as those that follow the BSP (Bulk-Synchronous Parallel) programming model [33]. Concerning the choose of migration processes, MigBSP creates a priority list based on the highest Potential of Migration (PM) of each process. PM is a decision function that combines the migration costs with data from computation and communication phases in order to create a unified scheduling metric.

Taking profit from the highest PM of each process, MigBSP could originally employ one of two methods to select the candidate processes for migration. As illustrated in Figure 1.1, MigBSP can select one or a group of processes located on the top of the list. The second case is viable thanks to a predefined percentage that acts over the highest PM value. Although we achieved good results particularly with this second approach [28], we agree that the use of another percentage value could eventually determine better migration results. Consequently, a question arises: *Using the PM idea, how can one reach an optimized percentage of migratable candidates on dynamic environments?* A solution involves the testing of several hand-tuned parameters for each new BSP application and a comparison among the results.

After developing the first version of MigBSP, we focused our research on investigating new heuristics and metaheuristics in order to fill the aforementioned gap. We followed this rationally because both scheduling and rescheduling techniques are classified as NP-hard problems [15]. Taking into account metaheuristics, Genetic Algorithms [14, 22, 26], Simulated Annealing [13, 34], Artificial Bee Algorithms [16, 3], Pareto Search [32] and Hybrid Schemes [18] are commonly used for these tasks. Considering their iterative nature, they are known by reaching high-quality solutions meanwhile paying a high-computational time for achieving optimal or near-optimal solutions. On the other hand, heuristics are faster than metaheuristics, since they operate with mental shortcuts to ease the cognitive load of making a decision [9]. Thus, heuristics such as min-min and max-min operate by trading optimistically, completeness, accuracy, or precision for speed. When analyzing the state-of-the-art on migration-aware BSP communication libraries [8, 19, 21, 24, 25, 28, 35], we still observe that both

*Applied Computing Graduate Program, Universidade do Vale do Rio dos Sinos, São Leopoldo - Rio Grande do Sul - Postal Code 93022-000 - Brazil (rrrighi@unisinos.br).

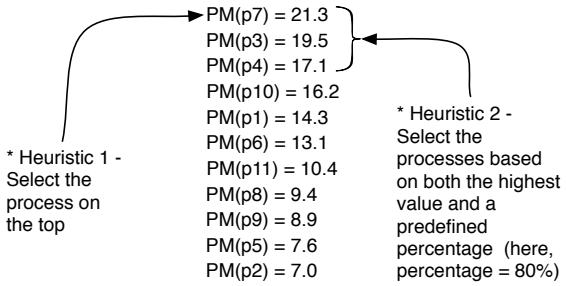

Fig. 1.1: Current MigBSP's methods for choosing the candidate processes for migration based on a decision function named Potential of Migration (PM).

heuristics and metaheuristics techniques are not employed to offer rescheduling under the following constraints: (i) combination of multiple metrics; (ii) automatic selection of candidate processes for migration without user intervention.

When process (re)scheduling is considered, two timers are involved: calculus complexity and quality of the mapping. Both measures are used in heuristics for optimizing the MigBSP's initial approaches. In this regard, we developed two novel heuristics named **MigCube** and **MigHull** for automatically selecting one or more candidates for migration at each rescheduling attempt. They solve a 3D geometric query problem taking profit from the computation, communication and migration costs metrics of the PM as the values for the x, y and z axes. So, the scientific contribution of the article consists in exploring **computational geometry** concepts to select the most suitable points arranged in a three-dimensional space, consequently indicating the processes for migration, without needing any intervention when considering the user viewpoint. MigCube explores the Euclidean distance [12] among the points while MigHull extends the idea of Convex Hull for a 3D setting [4].

This article presents the algorithms of MigCube and MigHull in detail, followed by their evaluation when using two BSP scientific applications over a subset of the Grid5000 infrastructure[1]. Besides not needing a particular parameter in the model at compilation time, the results also show the benefits of selecting a more appropriate number of migratable processes instead of selecting just one or a percentage of them. The contribution of both proposed heuristics does not appear only in the MigBSP scope, but also for the BSP community who is interested in efficient migration process at middleware level in an effortless way.

The remainder of this article will first introduce the fundamental concepts in Section 2, explaining how MigBSP works in detail. The main part of the paper belongs to Section 3, where both MigCube and MigHull algorithms are proposed. Sections 4 and 5 show the employed methodology and the results, respectively. Related work is discussed in Section 6. Finally, Section 7 emphasizes the scientific contribution of the work and notes challenges that we can address in the future.

2. Fundamental Concepts. This section explains the functioning of MigBSP, emphasizing its rationales and parameters. MigBSP is a rescheduling model that works over heterogeneous resources, joining the power of clusters, supercomputers and local networks. The heterogeneity issue considers the processor's clock (all processors have the same set of instructions), as well as the network bandwidth. Such an architecture is assembled with Sets (sites or clusters) and Set Managers. Set Managers are responsible for scheduling, capturing data from a Set and exchanging it among other managers [28].

The decision for process remapping is taken at the end of a superstep. A BSP program has an arbitrary number of supersteps, each one composed by a local computation phase on each process, a global and arbitrary communication phase among the processes and a synchronization barrier [33]. Aiming at not trying to test process rescheduling at each conclusion of superstep, we designed a parameter named α to control the interval of supersteps between two consecutive attempts for process rescheduling. Thus, we applied two adaptations

[1] https://www.grid5000.fr/

that control the value of the α ($\alpha \in \mathbb{N}$) in order to reduce the scheduling model intrusiveness: (i) to postpone the rescheduling call if the processes are balanced or to turn it more frequent, otherwise; (ii) to delay this call if a pattern without migrations on ω past calls is observed. Thus, α is automatically updated at each rescheduling call and will indicate the interval for the next one (more details in [28]). A shorter initial value of α will bring better reactivity on application reorganization, since process rescheduling will be evaluated as soon reaching the α^{th} superstep. So, this configuration implies on reconfiguring the application sooner, benefiting the remaining of the execution with an optimized process-resources mapping. However, if process migration is inviable (due to the large number of bytes to be transferred or a prohibitive network latency overhead, for example), a shorter α will cause more overhead in the normal execution of the application. In this last case, process rescheduling is tested more frequently, but no migrations take place actually.

The answer for 'Which' is solved through our decision function called Potential of Migration (PM). Each process i computes n functions $PM(i,j)$, where n is the number of Sets and j means a particular Set. The key rationale consists in performing only a subset of the processes-resources tests at the rescheduling moment. The value of $PM(i,j)$ is found using Computation, Communication and Memory metrics as presented in Equations 2.1–2.4. A previous paper describes them in detail [28]. The greater the value of $PM(i,j)$, the more prone the processes will be to migrate.

$$Comp(i,j) = P_{comp}(i) \times CTP_{k+\alpha-1}(i) \times ISet_{k+\alpha-1}(j); \tag{2.1}$$

$$Comm(i,j) = P_{comm}(i,j) \times BTP_{k+\alpha-1}(i,j); \tag{2.2}$$

$$Mem(i,j) = M(i) \times T(i,j) + Mig(i,j); \tag{2.3}$$

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j). \tag{2.4}$$

Computation metric $Comp(i,j)$ considers a Computation Pattern $Pcomp(i)$ that measures the stability of a process i regarding the number of instructions at each superstep. This value is close to 1 if the process is regular and close to 0 otherwise. Furthermore, we also have a computation time prediction $CTP(i)$ for process i based on all computation phases between two rescheduling activation. In this way, here k refers to the index of the last call for process rescheduling and $k + \alpha - 1$ means the interval of supersteps from the last to the current rescheduling attempt. The metric $Comp(i,j)$ also presents an index $ISet(j)$ which informs the average computation capacity of Set j. In the same way, Communication metric $Comm(i,j)$ computes the Communication Pattern $Pcomm(i,j)$ between processes and Sets. Furthermore, this metric uses communication time prediction $BTP(i,j)$ considering data between two re-balancing activation. $Comm(i,j)$ increases if process i has a regular communication with processes from Set j and performs slower communication actions to this Set. The metric $Mem(i,j)$ considers process memory $M(i)$, transferring rate $T(i,j)$ between considered process i and the manager of target Set j, as well as migration costs $Mig(i,j)$. These costs are dependent of the operating system, as well as the migration tool [28].

At each rescheduling call, each process passes its highest $PM(i,j)$ to its Set Manager. This last entity exchanges the PM of the processes with other managers. As described earlier, each manager creates a decreasing-sorted list and selects either the process on the top or a percentage of them for testing the migration viability. Here, besides using the abstraction of Set, this test also considers the following data: (i) the external load on source and destination processors; (ii) the processes that both processors are executing; (iii) the simulation of considered process running on a destination processor; (iv) the time of communication actions considering local and destination processors; (v) migration costs. We used these five information to compute the migration viability of each process through a relationship between two timers: $t1$ and $t2$. $t1$ means the superstep time of process i in the current processor, while $t2$ encompasses its execution on the other processor and it includes the migration costs. Process migration takes place if $t1 > t2$.

3. MigCube and MigHull: Proposal of Novel Heuristics to Select the Candidates for Migration. This article proposes MigCube and MigHull heuristics to improve efficiency on selecting process of BSP

applications at the rescheduling moment. The main idea is to outperform the current MigBSP strategies for this task, particularly without user intervention at editing or launching time to set model parameter for selection purposes. At the application perspective, the use of a particular selection policy or even process rescheduling facility is totally hidden from the user. Usually, for the submission of a BSP application in a grid, it is previously compiled with a rescheduling-aware BSP library, informing an initial processes-nodes scheduling [38]. Figure 3.1 depicts the software stack when using MigBSP. The gray boxes represent the scope of this article.

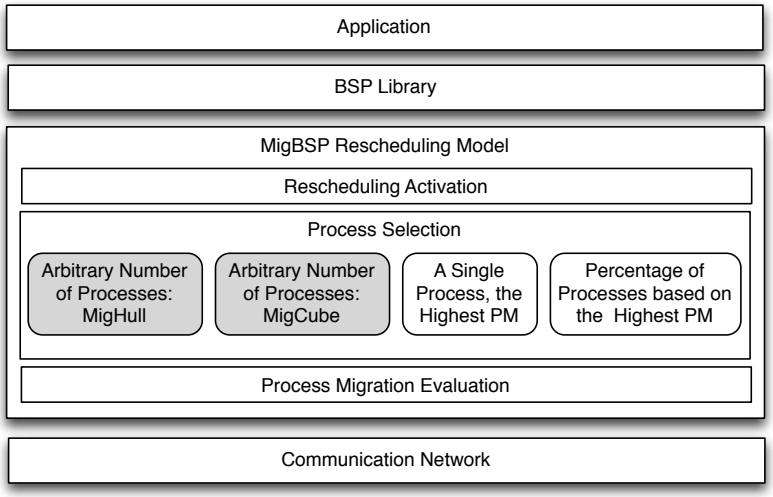

Fig. 3.1: Software stack when using the novel heuristics for BSP process rescheduling

BSP applications have their performance always driven by the slowest process, so both heuristics try to optimize the number and the selection of processes to eventually migrate so that the remaining supersteps may run faster. Unlike previous approaches, MigCube and MigHull select an arbitrary number of processes but also considering the list of the highest PM of each process. Figure 3.2 illustrates the rescheduling in a BSP application. The mapping quality of MigCube or MigHull will impact the next value of α parameter. If the system is classified as balanced, the value of α is increased in order to postpone the next call for process rescheduling.

Both MigCube and MigHull take profit from the list of the highest PM of each process. Considering that each PM identifies a process and a target Set, and since all the three assumed metrics are expressed in the same data unit, we may plot them as a single point in a 3D setting. In this way, the proposed heuristics must answer the following answer: *Which points should be selected at each rescheduling call?* To accomplish this, MigCube and MigHull use computational geometry to analyze a set of points, in order to efficiently find which points are close to the input query. At model level, each Set Manager compute the selected points locally, each one referring to a particular process i that presents a $PM(i, j)$. After that, only the source and the target Sets (represented by j in the PM notation) are involved to transfer a process i actually. The destination Set informs the source Set about which is the most suitable processor under its responsibility to receive the process.

Heuristic methods are employed because they are generally used to find a solution of a specific-domain problem without exhaustively searching the entire solution space [36]. Thus, such algorithms can usually achieve good solutions in a small computational time. This is special pertinent on our case, since we have to pay the overhead inherent to the migration process. Nevertheless, we could use metaheuristic methods rather than heuristic ones. However, metaheuristics represent more general approximate algorithms which are defined as upper level techniques that guide strategies underlying heuristics to solve specific optimization problems [31]. For this reason, metaheuristics sometimes require high processing time to attain near-optimal solutions, especially for large-size problems. Anyway, both single solution-based or population-based metaheuristics could be used,

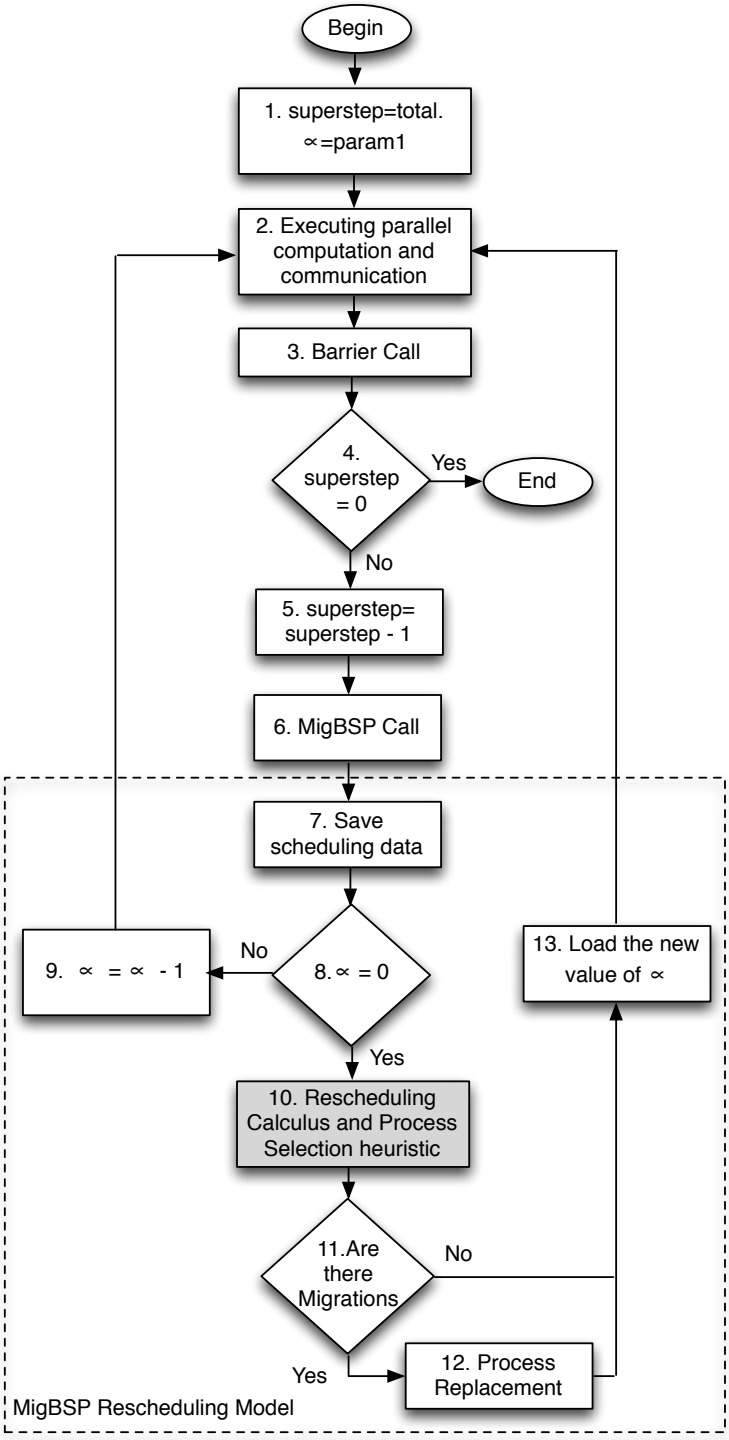

Fig. 3.2: MigBSP flowchart, where the gray box represents the work of MigCube or MigHull.

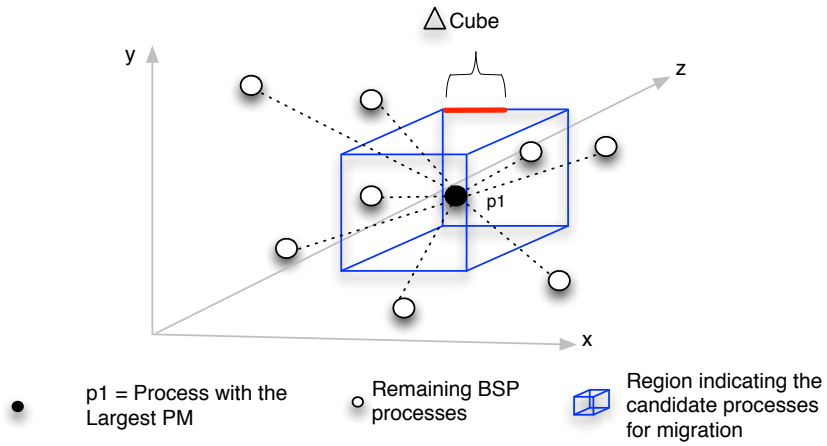

Fig. 3.3: Processes and cube representation in MigCube heuristic. Those processes that are located inside the cube will be appointed as candidate processes for migration.

such as Simulated Annealing, Tabu Search, Genetic Algorithms and Swarm Intelligence Schemes, needing only a given quality measurement to be performed.

3.1. Terminologies. Both MigCube and MigHull plot each process p_i $(i = 1, 2, \ldots, n)$ as a point in the 3D Cartesian coordinate system where x_i, y_i and z_i represent the coordinates of p_i on each of the graph axes. Here, x_i, y_i and z_i also represent respectively, the computation, communication and memory metrics from the largest PM of p_i. In addition, a process p_i can be also represented as $p_i = (x_i, y_i, z_i)$.

3.2. MigCube Heuristic. MigCube uses the processes' location to create a cube, so selecting as candidates those processes inside it. The algorithm starts by selecting the central point of the cube, which refers to the point that has the largest PM. Its notation is p_1 and it represents the best candidate for migration. After that, parameter $\triangle_{cube}$ is computed in accordance with Equation 3.1 as an average of the distances from the aforementioned point to the others. Equation 3.2 computes the distance between p_1 and any point p_i in the 3D coordinate system. Finally, $\triangle_{cube}$ is used to situate the cube edges as defined in Equation 3.3.

$$\triangle_{cube} = \frac{1}{n-1} \sum_{i=2}^{n} D(p_1, p_i)\,; \tag{3.1}$$

$$D(p_1, p_i) = \sqrt[2]{(x_1 - x_i)^2 + (y_1 - y_i)^2 + (z_1 - z_i)^2}\,; \tag{3.2}$$

$$edge = 2\,\triangle_{cube}\,. \tag{3.3}$$

Figure 3.3 depicts an example of the points and the cube, where p_1 has the largest PM. Algorithm 1 presents the pseudocode of MigCube heuristic for selecting the candidate process for migration. As mentioned earlier, the idea is to select as candidates all processes inside the cube. MigCube will always select at least one process, the one with the largest PM. After the heuristic sets which processes could migrate, the model follows its normal processing, migrating or not the processes in accordance with the destination Set of each candidate process. Each process p_i has a $PM(i, j)$ where i means the process index and j a target Set. So, the Manager of $j-th$ Set is asked about a resource and migration viability is computed as explained in Section 2 (more details in [28]).

Algorithm 1: MigCube heuristic for selecting the candidate processes for migration.

Input: *pm_list* receives a decreasing-sorted list of the n processes based on the PM values.
Output: *candidate_list* with the candidate processes for migration

Set process p_1 as the first element of *pm_list*, being represented by (x_1, y_1, z_1);
$minorX = x_1 - \triangle_{cube}$;
$majorX = x_1 + \triangle_{cube}$;
$minorY = y_1 - \triangle_{cube}$;
$majorY = y_1 + \triangle_{cube}$;
$minorZ = z_1 - \triangle_{cube}$;
$majorZ = z_1 + \triangle_{cube}$;
$candidate\_list = p_1$;
for $i = 2$ **to** n **do**
  if $x_i \geq minorX$ and $x_i \leq majorX$ **then**
    if $y_i \geq minorY$ and $y_i \leq majorY$ **then**
      if $z_i \geq minorZ$ and $z_i \leq majorZ$ **then**
        $candidate\_list \mathrel{+}= p_i$;
      end if
    end if
  end if
end for

3.3. MigHull Heuristic. MigHull heuristic is a Convex Hull adaptation. In brief, the Convex Hull or Convex Envelope of a set S of points in the Euclidean plane is the smallest convex set that contains S [6, 4]. It can be seen as a convex polygon whose vertices are some of the points in the input set. MigHull employs the Convex Hull ideas, but providing two adaptations: (i) three-dimensional space is split in three two-dimensional planes; (ii) despite of selecting all processes, MigHull chooses only a part of them based on the two processes with the highest PM values.

We are calculating three 2D hulls, considering a pair of coordinates of each point i at a time, as follows: (i) x_i and y_i; (ii) x_i and z_i; and (iii) y_i and z_i. Figure 3.4 (a) illustrates this idea. Here, each process that is inside each plane concomitantly is then selected as a candidate for migration. For the standard 2D Convex Hull, the problem consists of finding the smallest convex polyhedron/polygon containing all the points. Thus, the native Convex Hull always selects all the points, which would not make sense for migration decision-making. In this way, we are adapting the QuickHull algorithm [5] to select processes. By default, QuickHull finds the points with the minimum and maximum x coordinates and creates a line between them. The next step in the QuickHull algorithm is the selection of the point with the maximum distance from the aforesaid line, so the two points found before along with this one form a triangle. The points lying inside of that triangle cannot be part of the convex hull and can therefore be ignored in the next steps. Discarding the tested point and the previous ignored ones, the algorithm selects the next point with the maximum distance from the line and proceeds the same calculus again.

MigHull changes QuickHull as follows. Considering the plane $a - b$, where a means the abscissa and b the ordinate, we are considering the a-coordinate of the two points with the highest PM to draw a line segment between them. After that, we calculate $\triangle_{Hull}$ as the maximum distance of coverage from this line segment to the other processes, so the processes inside this region are candidates to migrate in the scope of $a - b$ plane. Figure 3.4 (b) illustrates an example of this procedure for the $x - y$ plane, but the same is evaluated for other two planes: $x - z$ and $y - z$. In other words, by substituting $x - y$ plane by $x - z$ and $y - z$ the distance $\triangle_{Hull}$ is also calculated in the $x - z$ and $y - z$ planes, respectively. Finally, the processes that appear as candidates concomitantly in the $x - y$, $x - z$ and $y - z$ planes are selected as final candidates to migrate according to the MigHull algorithm.

$$\triangle_{Hull} = \sigma(x, y) = Max(\sigma(x), \sigma(y)) \,. \tag{3.4}$$

Equation 3.4 shows how we are computing $\triangle_{Hull}$ for the $x - y$ plane. Each plane has its own value for this metric. In this equation, $\sigma(a)$ for a specific axis a is the standard deviation of all points (*i.e.* processes) when

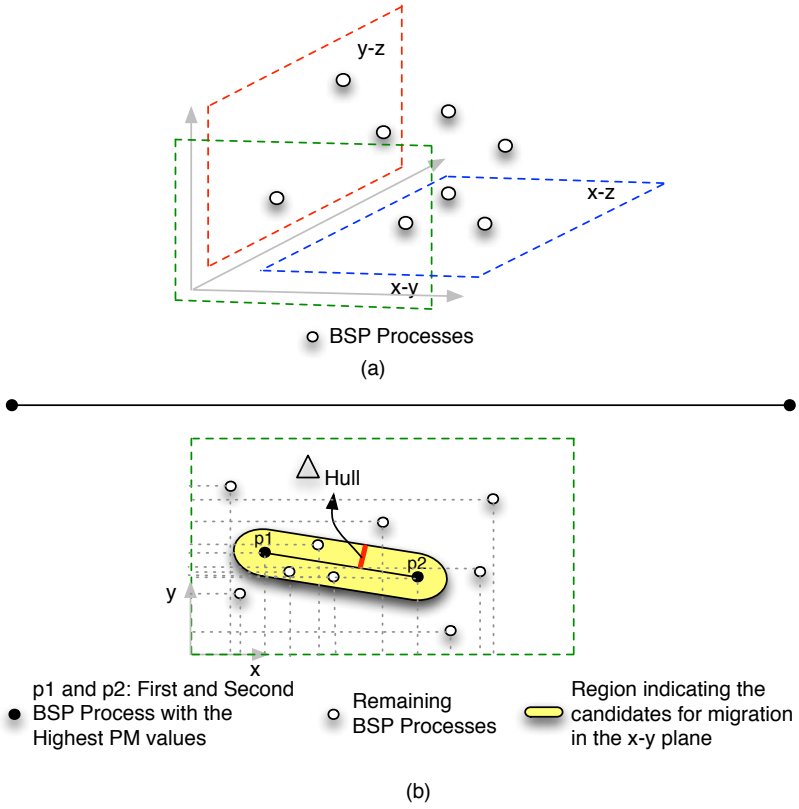

Fig. 3.4: Selection of candidate process for migration with MigHull: (a) Creating three planes (x-y, x-z and y-z) from the three-dimensional space; (b) partially selecting the candidate process in the x-y plane. Those processes that appear concomitantly in the yellow region of the three planes are chosen for the next rescheduling step: the tests of migration viability.

considering the coordinate a. Using Figure 3.4 (b) as an example, we firstly take the value of the coordinate x of all the 11 points, computing the standard deviation $\sigma(x)$ of these respective values. The same calculation is performed with respect to the y axis, so the greatest standard deviation is selected as $\triangle_{Hull}$ for the $x-y$ plane. In order to identify the candidate processes for migration, the distance of any point p_m to the line determined by the points p_1 and p_2 over a specific plane (see Algorithm 2) is computed and denoted as $d(p_m, p_1, p_2, plane)$. These last two points represent the processes with the highest PM. If x-coordinate of p_m is lower or greater than the x-coordinate of points used as limits of the line segment, we are computing the Euclidean distance given by the Pythagorean formula [12]. Otherwise, we are using the perpendicular distance from a point to a line determined by p_1 and p_2. Although Algorithm 2 was developed for the x-y plane, its use for x-z and y-z is trivial and not explained here.

Figure 3.4 (b) depicts the MigHull ideas to create a region of candidate processes for migration. Contrary to MigCube, MigHull always selects at least two processes as candidates for migration: p_1 and p_2. Independently of the evaluated plane, these points always have the largest PM, so being always selected according to the MigHull algorithm. Algorithm 3 shows all steps to compute MigHull, where the processes that appear as candidates concomitantly in the $x-y$, $x-z$ and $y-z$ are candidates to be rescheduled. After MigHull presents the candidates, MigBSP continues its normal execution investigating the migration feasibility for each candidate through an interaction between the source and target Set Managers. MigBSP was presented in Section 2 and detailed in [28].

Algorithm 2: Calculating the distance $d(p_m, p_1, p_2, plane)$ from the point p_m to the line segment created by the points p_1 and p_2 in the $x - y$ plane.

Input: p_1 (x_1, y_1, z_1) and p_2 (x_2, y_2, z_2) denote the two processes with the highest PM values. The point p_m (x_m, y_m, z_m) refers to one of the remaining processes, where $3 \leq m \leq n$.
Output: Distance $d(p_m, p_1, p_2, plane)$ from the point p_m to the line created by p_1 and p_2 in the plane denoted by *plane*.

Denote $ax + by + c$ as the line equation formed by the points p_1 and p_2, where the coefficients are defined as:
$a = (y_1 - y_2)$, $b = (x_2 - x_1)$ and $c = (x_1 y_2 - x_2 y_1)$;
if $x_m < x_1$ **then**
  $d(p_m, p_1, p_2, \text{“}x - y\text{”}) = \sqrt{(x_1 - x_m)^2 + (y_1 - y_m)^2}$
end if
else if $x_m > x_2$ **then**
  $d(p_m, p_1, p_2, \text{“}x - y\text{”}) = \sqrt{(x_m - x_2)^2 + (y_m - y_2)^2}$
end if
else
  $d(p_m, p_1, p_2, \text{“}x - y\text{”}) = \dfrac{ax_m + by_m + c}{\sqrt{a^2 + b^2}}$
end if

Algorithm 3: MigHull heuristic for selecting the candidate processes for migration.

Input: *pm_list* receives a decreasing-sorted list of the n processes based on the PM values.
Output: *candidate_list* with the candidate processes for migration.

Set processes p_1 and p_2 as the first and the second elements of *pm_list*, being represented by (x_1, y_1, z_1) and (x_2, y_2, z_2), respectively;
$candidate\_list = p_1$;
$candidate\_list \mathrel{+}= p_2$;
$candidate_{\text{x-y}} = \text{null}$;
$candidate_{\text{x-z}} = \text{null}$;
$candidate_{\text{y-z}} = \text{null}$;
for $i = 3$ **to** n **do**
  if $d(p_i, p_1, p_2, \text{“}x - y\text{”}) \leq \triangle_{Hull}$ **then**
    $candidate_{\text{x-y}} \mathrel{+}= p_i$;
  end if
end for
for $i = 3$ **to** n **do**
  if $d(p_i, p_1, p_2, \text{“}x - z\text{”}) \leq \triangle_{Hull}$ **then**
    $candidate_{\text{x-z}} \mathrel{+}= p_i$;
  end if
end for
for $i = 3$ **to** n **do**
  if $d(p_i, p_1, p_2, \text{“}y - z\text{”}) \leq \triangle_{Hull}$ **then**
    $candidate_{\text{y-z}} \mathrel{+}= p_i$;
  end if
end for
$candidate\_list \mathrel{+}= \{candidate_{\text{x-y}} \cap candidate_{\text{x-z}} \cap candidate_{\text{y-z}}\}$

4. Evaluation Methodology. This section describes the evaluation methodology, presenting data about the evaluation technique, MigBSP parameters, execution environment and tested application. Firstly, we are using the SimGrid [2] simulator to assembly a grid computing infrastructure, because of it offers a framework to evaluate message-passing applications with different scheduling algorithms and execution platforms. We did not developed any extension to SimGrid, but only applications that use its native API (Application Program Interface). Considering that SimGrid is deterministic, a single execution of each set of parameters was done. Moreover, the number of supersteps is variable, as follows: 20, 40, 60, 80 and 100. The initial value of α is selected among three numbers: 4, 8 and 16. We selected them because these values were used when evaluating the first version of MigBSP [28], where significant impacts on performance and overhead were perceived when changing from one value to another. Furthermore, as will be discussed in Subsection 5.4, we will present a

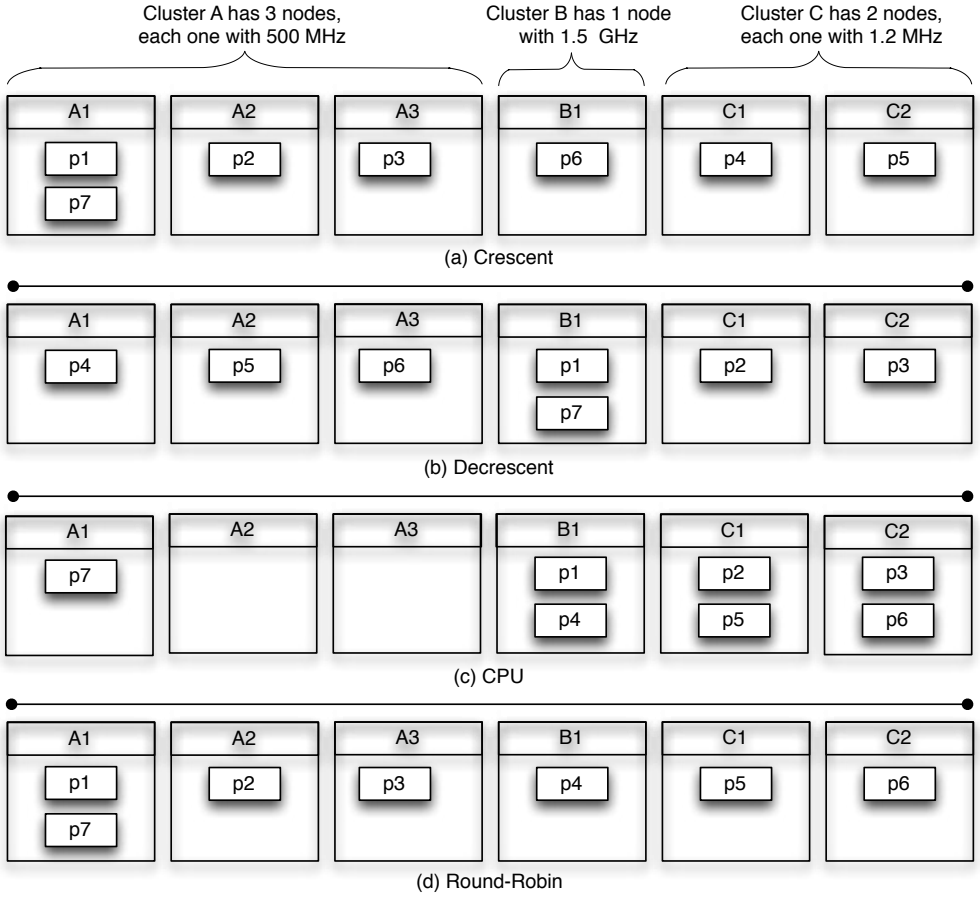

Fig. 4.1: Example of the four initial processes-resources scheduling employed in the tests using a hypothetical grid infrastrucuture.

comparison study among MigCube, MigHull and the originals heuristics of MigBSP, so we can analyze the impact of these values of α on different algorithms for process migration.

Since the MigBSP was designed for grid environments, we are testing it with the proposed heuristics over the Grid5000 platform[2]. In fact, this platform is an XML file used by SimGrid, denoting machines, clusters and network configurations. Besides the platform file, SimGrid also receives as input another XML file informing the first scheduling (deployment). Particularly, we are using 45 nodes, distributed in 3 distinct sites, each one offering here a single cluster. We are using the 10 nodes from cluster Chicon, 15 from cluster Capricorne and 15 nodes from cluster Suno. The hardware information is described as follow: (i) Chicon has AMD Opteron 2.6 GHz processors with 4GB of memory and a Gigabit Ethernet card; (ii) Capricorne has AMD Opteron 2.0GHz processors, with 2GB of memory and a Myrinet network card; (iii) Suno has Intel Xeon E5520 2.26GHz processors with 32GB of memory and 2 Gigabit Ethernet cards. Considering the deployment file, we are working with 60 processes that are launched in accordance with an initial process-scheduling mapping. We are considering four of them, explained below and detailed in the example of Figure 4.1:

(a) Ascending: Processes are scheduled cyclically in Ascending order of nodes' processing power;

(b) Descending: Same idea of the Ascending mapping, but in reverse order, where the nodes with the higher

[2] http://lists.gforge.inria.fr

capacities are the first to receive processes;

(c) CPU: This mapping allocates each process to the resource that has the largest amount of free processing power on that moment;

(d) Round-Robin: It allocates the processes cyclically without taking into account any characteristics of the resources.

The initial mapping will influence the execution time directly, also influencing the rescheduling model in the same way. For example, the CPU mapping implies on using load balancing in accordance with the CPU power of the nodes, so this idea of equilibrium from the beginning tends to reduce the number of migrations at runtime. On the other hand, for example, the scheduling of all process in a single node could compromise the performance, imposing more rescheduling actions afterwards to spread them in the resource pool. Besides the initial mappings and MigBSP parameters, the tests also consider three scenarios: (i) execution of the native application, without MigBSP or proposed heuristics; (ii) the application runs with MigBSP, which performs the heuristics calculus and message-passing, but does not migrate any processes actually; (iii) the application runs with MigBSP and an heuristic to select the candidate processes for migration, enabling then any migration if it was evaluated as viable. The main idea is to show the overhead impact of the heuristic execution (comparison between scenarios (i) and (ii)) and performance impact when enabling migrations (comparison between scenarios (i) and (iii)).

Regarding the BSP application, we developed an implementation of the Lattice-Boltzmann method [29] to compute fluid dynamics. Technically, this method considers a typical volume of fluids composed of a collection of particles, where a particle is represented by a distribution function for each fluid component at each grid point. The data volume is divided into continuous blocks of equal size in accordance with the number of processes. Each block is copied and runs in a BSP process. After the computation phase, each process sends data to its right-sided neighbor. Finally, a synchronization barrier takes place and other superstep is computed afterwards.

5. Discussion of Results. The results consider the performance of MigCube and MigHull heuristics in terms of application processing time in Subsections 5.1 and 5.2. In addition, we also present two subsections, 5.3 and 5.4, for comparison purposes; the first one analyzes MigCube against MigHull and the second one compares both approaches with the standard process selection heuristics from MigBSP. We are using a BSP implementation of the fluid dynamics application with variations in the following configurations: number of supersteps; initial process-processor scheduling; the MigBSP's parameter denoted α; and the aforementioned evaluation scenarios.

5.1. MigCube Evaluation. Table 5.1 shows the test results with MigCube. Scenario (ii) always produces a time larger than scenario (i), since the first adds the heuristic calculus and message passing. This overhead can be considered as part of the heuristic execution cost. The mean overhead of MigCube is 3.21%. This overhead also takes place when migration are enabled but any process replacement is viable during the application execution. The effectiveness of MigCube appears when comparing scenarios (iii) and (i). The larger the number of supersteps, the larger the gains with process migration. In other words, an application that migrates the processes in the first supersteps presents better performance because of both it has more time to amortize the penalties involved in process migration and more time to execute with an optimized configuration. The highlighted fields show that only one migration happens when using 20 supersteps and α equal to 16. Although achieving better results than scenario (i), the system remains unbalanced. This situation is only solved when 60 supersteps are performed.

Figure 5.1 shows the percentage of gain in execution time when analyzing scenarios (iii) and (i). It has been calculated using the following equation:

$$\text{Gain} = \frac{Scenario\ (i) - Scenario\ (iii)}{Scenario\ (i)} \times 100\,. \tag{5.1}$$

The parameter α equal to 8 was the responsible for the best results. A lower value of α implies in a greater number of process rescheduling calls (recalling that each call encompasses scheduling calculus and message-passing), while a larger value of α parameter postpones the calls being so less reactive for process migration. When running a short number of supersteps, the configuration with α equal to 16 outperforms the other values

Table 5.1: MigCube evaluation. The times are expressed in seconds. We are highlighting the execution of 20 supersteps with α=16, where a single process migration takes place.

Supersteps	Scenarios						
	i	ii	iii	ii	iii	ii	iii
		$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
Ascending							
20	16.10	17.39	11.79	16.57	10.85	16.33	14.33
40	32.19	34.59	23.41	33.42	20.00	32.66	20.66
60	48.28	51.86	36.29	49.93	28.32	48.99	27.42
80	64.37	67.00	47.97	66.78	36.66	65.60	38.00
100	80.47	84.90	60.85	83.28	46.20	81.88	44.61
Descending							
20	16.27	16.68	11.67	16.55	11.50	16.39	14.41
40	32.94	33.09	21.47	33.28	21.36	33.20	21.99
60	48.82	49.50	32.41	49.38	29.91	49.28	29.88
80	65.09	65.80	42.21	65.78	38.58	65.63	40.82
100	81.37	81.40	53.14	81.75	48.40	81.68	47.67
CPU							
20	20.08	20.33	13.78	20.30	16.97	20.13	19.50
40	40.15	40.45	25.94	40.38	27.91	40.27	33.26
60	60.22	60.56	39.16	60.50	38.28	60.47	41.28
80	80.29	80.50	51.31	81.20	48.83	81.01	53.96
100	100.36	100.87	64.50	100.63	60.27	100.51	62.12
Round-Robin							
20	16.16	17.11	10.73	16.42	10.95	16.30	14.28
40	32.33	34.12	20.30	33.01	20.45	32.49	20.89
60	48.46	51.13	30.94	49.29	29.38	48.68	27.73
80	65.56	66.20	39.94	65.89	38.19	65.73	40.00
100	80.66	83.50	49.99	82.17	48.00	81.20	47.09

of α: with α=4 or α=8 we have a higher number of migrations (with time penalization on each migration activity) but not enough number of supersteps to amortize the investment in migrations. Figure 5.1 presents a linear behavior when considering the execution with α equal to 4. In this case, process reorganization happens earlier and then, the execution can proceed with an optimized process-resource mapping after passing the first supersteps.

Figure 5.2 illustrates the number of migrations at each rescheduling call when considering α equal to 8. Considering the CPU strategy for initial scheduling, we can observe that MigCube selects a large number of processes to migrate at each attempt. The PM of the processes are closed to the largest PM, showing a large number of migrations at each rescheduling call. After analyzing the log of operations, we can observe a principle of hysteresis, *i.e*, several consecutive migrations in order to stabilize the behavior of the system. Moreover, in the current implementation, the migration test of a candidate process does not take into account the previous migration of other process to the same target (node or cluster), so contributing for the large number of observed migrations. These ideas explain the performance of the CPU strategy when compared to the remaining ones. Although obtaining good performance rates when comparing scenarios (i) and (iii), the CPU technique for initial scheduling achieves the worst performance among the other ones.

5.2. MigHull Evaluation. Table 5.2 shows the MigHull results. A mean overhead of 3.45% in the execution time was observed when comparing scenarios (i) and (ii). Considering scenario (iii), the same MigCube performance panorama appears here in MigHull, where larger gains appear when enlarging the number of supersteps. In particular, as presented in MigCube, the use of α equal to 16 was responsible for the best performance values when executing a short number of supersteps. Table 5.2 highlights the large difference in time when comparing the CPU initial mapping against the other three initial scheduling strategies. Although

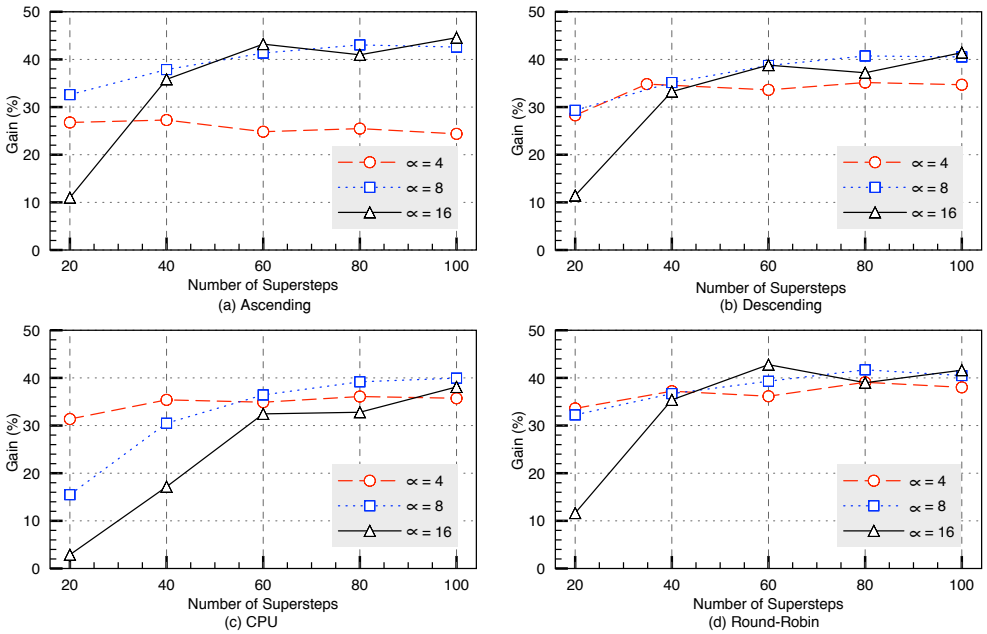

Fig. 5.1: Percentage of gain in the execution time with MigCube-driven process rescheduling

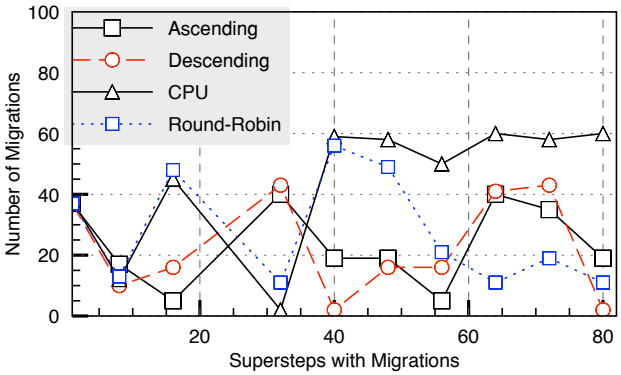

Fig. 5.2: Number of migrations at each rescheduling call when using MigCube and $\alpha = 8$.

efficient in the CPU perspective, the CPU initial mapping causes communication penalties because there are a large number of inter-cluster communication.

Figure 5.3 shows the percentage of gain on the execution time when MigHull-driven migrations take place. The results were calculated considering Equation 5.1 and data from scenarios (i) and (iii). Analyzing Table 5.2 and the graphs in Figure 5.3 using the MigHull, it is possible to verify that a value of α equal to 8 is the most stable when considering the time gain. However, different from MigCube, a value of α equal to 16 does not show a tendency of gain in performance. The use of MigHull tends to be more complex and increases the cost of execution as increases the number of clusters, because each BSP process need to calculate the probability of migration in each cluster; in this way, increasing the computation cost. Figure 5.4 depicts the number of migrations at each rescheduling intervention when using α equal to 8 and the MigHull heuristic. Clearly, the

Table 5.2: MigHull evaluation. The times are expressed in seconds. We are highlighting the performance of scenario (i), where the CPU strategy for initial mapping presents large disparities

Supersteps	Scenarios						
	i	ii	iii	ii	iii	ii	iii
		$\alpha = 4$		$\alpha = 8$		$\alpha = 16$	
Ascending							
20	16.10	17.33	11.75	16.57	11.89	16.33	14.56
40	32.19	34.59	21.52	33.42	21.95	32.66	23.04
60	48.28	51.56	32.20	49.93	30.93	48.99	31.57
80	64.37	67.04	42.07	66.78	39.91	64.60	42.39
100	80.47	84.48	52.85	83.28	49.90	81.88	49.96
Descending							
20	16.27	16.68	12.15	16.60	11.99	16.39	14.31
40	32.54	33.09	21.53	33.01	22.15	32.95	23.32
60	48.82	49.50	32.27	49.38	31.12	49.25	32.12
80	65.09	65.38	41.97	65.25	40.21	65.15	42.71
100	81.37	81.70	53.11	81.65	50.22	81.53	51.37
CPU							
20	20.08	20.33	14.47	20.31	15.72	20.13	17.46
40	40.15	40.45	27.04	40.37	27.05	40.29	28.46
60	60.22	60.56	38.87	60.51	37.07	60.49	40.91
80	80.29	81.10	51.84	81.02	47.37	80.95	55.03
100	100.36	101.13	63.67	100.94	56.57	100.77	67.51
Round-Robin							
20	16.16	17.11	11.61	16.42	11.81	16.30	14.50
40	32.33	34.12	21.20	33.01	21.93	32.49	23.04
60	48.46	51.13	31.85	49.29	30.87	48.68	31.97
80	64.56	65.15	41.44	65.89	39.87	65.09	43.20
100	80.60	83.60	52.10	82.17	49.85	81.20	50.70

MigHull strategy of using an intersection of the three 2D planes is responsible for reducing the number of migratable processes when compared to MigCube. Particularly, Figure 5.5 illustrates three moments of the execution for the Ascending strategy, showing the division of the processes among the clusters. We can observe the movement of the processes to take profit from the most powerful clusters, Chicon and Suno (see Section 4 for details regarding the subset of the Grid5000 infrastructure used in the tests).

5.3. MigCube and MigHull Comparative. Both MigCube and MigHull heuristics have the same objective and make use of the same idea: computational geometry to select a portion of process to migrate. Figure 5.6 illustrates the gains considering each initial scheduling and heuristic. The graph shows the mean value of gain of scenario (iii) over scenario (i) when considering all set of supersteps and α values. MigCube with the Ascending scheduling and α value equal to 4 achieved a gain of 25%, diverging significantly from the other results. This divergence occurs due to a low α, which makes many processes to migrate, increasing the communication between process and approximating metrics.

MigHull achieved up to 35% of gain in application execution time with process migration, while MigCube obtained 42%. Figure 5.7 presents an analysis of the execution time of the supersteps at each migration call. The time presented in the graph refers to the interval between two supersteps in which a migration call took place. Particularly, we are considering 60 processes, 80 supersteps, α equal to 8 and the initial scheduling as being Round-Robin. In this way, migrations are evaluated at supersteps 1, 8, 16, 24, 32, 40, 48, 56, 64, 72 and 80. In this figure, the label 'Without Migration' represents the execution of the application without any migration, so the time is stable since the application performs the same number of computation activities at each superstep. We can observe that the time on both MigCube and MigHull tends to stabilize after crossing the

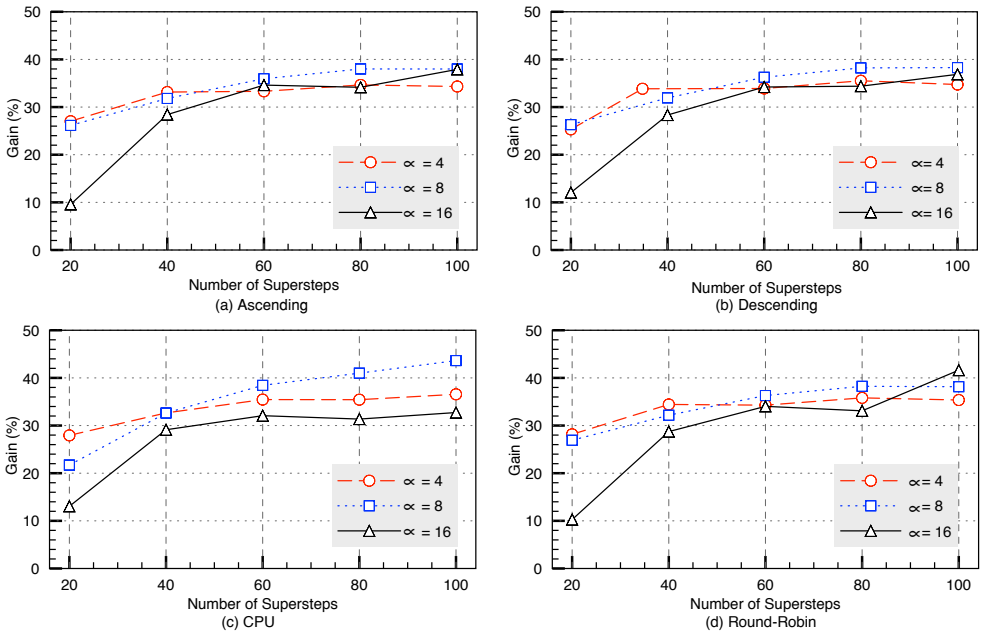

Fig. 5.3: Percentage of gain in the execution time with MigHull-driven process rescheduling

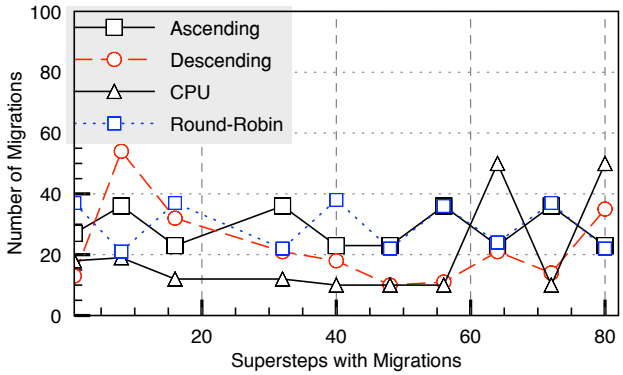

Fig. 5.4: Number of migrations at each rescheduling call when using MigHull and $\alpha = 8$.

fourth migration call. Furthermore, we can observe that they achieved the main idea with process migration: to reduce the time of a superstep, so minimizing the application time as a whole.

5.4. Comparing MigCube and MigHull Against the Original Heuristics of MigBSP. Here, we intend to compare the proposed heuristics with the original ones, all developed for the scope of MigBSP. Up to the moment of MigCube and MigHull proposals, MigBSP offers two heuristics to select the candidate processes for migration, both of them based on the descending-sorted list of the highest PM of each process: (i) we can select the top of the list or; (ii) use a percentage to select a number of processes based on the value belonging to the top. While the first is not reactive, the second needs the user intervention to set a particular percentage for the application and execution environment duet. This last task is not trivial, mainly when dealing with heterogeneous and/or dynamic applications or parallel machines. Concerning this panorama, MigCube and

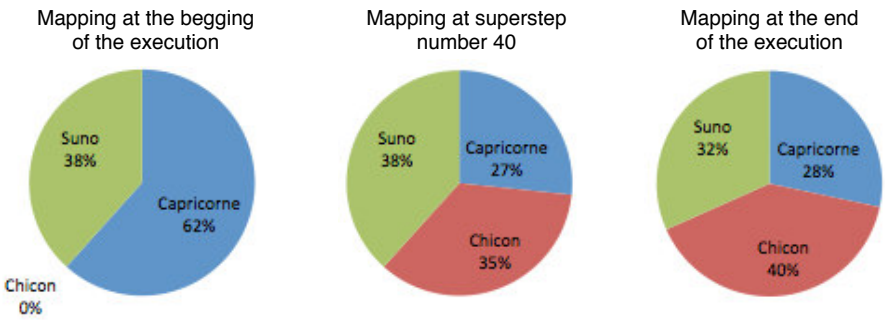

Fig. 5.5: Different moments of processes-clusters mappings when executing MigCube with the Ascending strategy for initial scheduling, 80 supersteps and $\alpha = 8$.

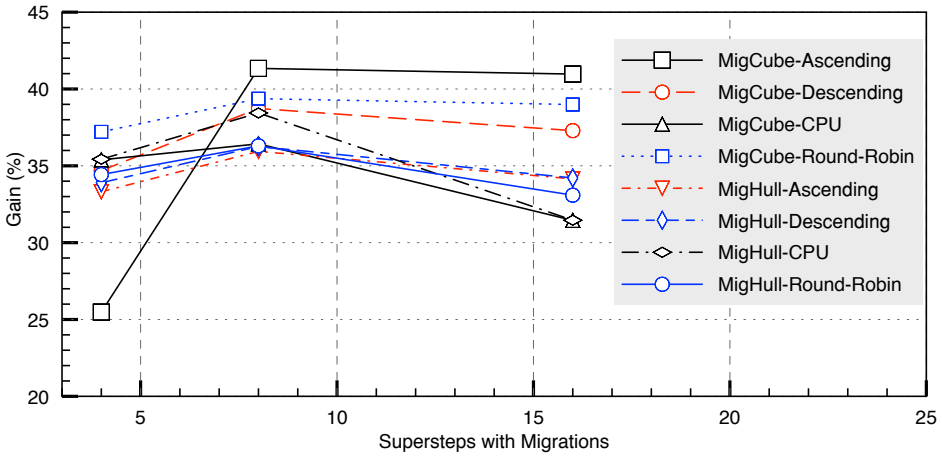

Fig. 5.6: Comparative involving MigCube and MigHull when varying the value of α

MigHull come to fill the gap on process selection re-activity, not needing any intervention from the user nor previous knowledge about the BSP application.

Figure 5.8 shows a performance graph when considering the four aforementioned heuristics. This graph depicts, for each value of α, a mean value of the executions with the four initial scheduling. The gain refers to the performance of scenarios (i) and (iii). As expected, the heuristic that selects only one process obtained the worst results. The heuristic of percentage selection, that is using a 20% selection from the top PM has similar results to MigCube and MigHull. This happens because the percentage heuristic can select more process at each execution, providing a fast rescheduling of processes. The MigCube and MigHull achieve the best results due its analysis of each metric and the use of geometrical space.

6. Related Work. Today, BSP represents the most used programming model to write successful parallel programs that exhibit phase-based computational behaviors. Thus, despite being proposed more than two decades ago by Leslie Valiant [33], several initiatives offer this model together with load balancing techniques and/or to treat particular parallel platforms [1, 8, 7, 37, 19, 21, 24, 27]. HAMA [1] is a cluster-driven library, particularly suitable for heterogeneous systems. It runs on top of the HDFS (Hadoop Distributed File System) in order to integrate BSP and iterative Map-Reduce applications. PUB [8, 7] is a C library that offers both

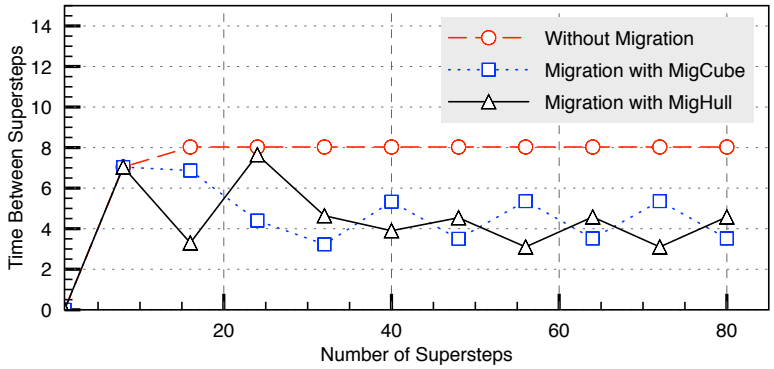

Fig. 5.7: Time between two supersteps in which migration calls took place.

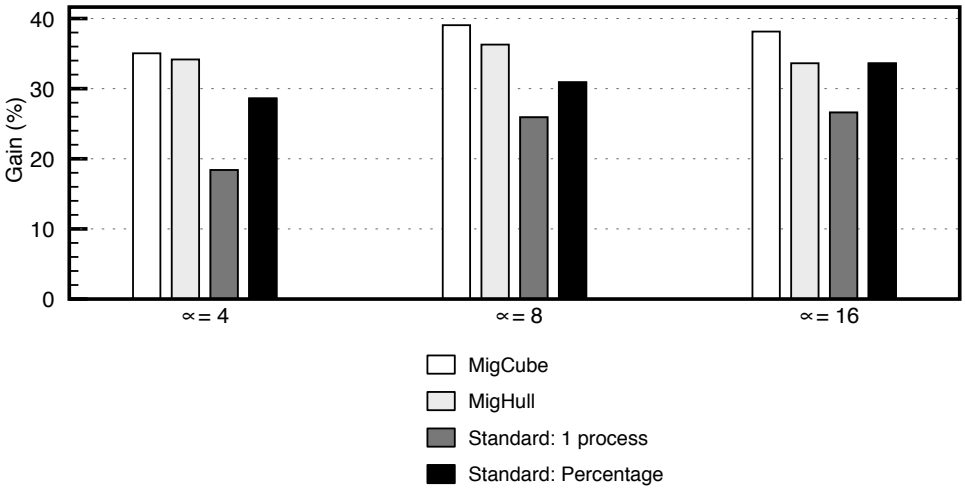

Fig. 5.8: Comparing MigCube and MigHull with the standard MigBSP (approaches to select the migratable processes: only the process in the top of the PM list and a percentage of processes based on the top value of this list).

centralized and distributed strategies for load balancing. In the first one, all nodes send data about their CPU power and load to a master node. The master verifies the least and the most loaded node and migrates one process between them. In distributed approaches, every node chooses c (PUB parameter) other nodes randomly and asks them for their load. One process is migrated if the minimum load of c analyzed nodes is smaller than own load of the node that is performing the test.

Mizan [19] monitors run-time characteristics of all processes (*i.e.*, their execution time and incoming and outgoing messages). Using these measurements, at the end of every superstep, Mizan constructs a migration plan that minimizes the variations across workers by identifying which vertices to migrate and where to migrate them. BSPCloud [21] can make full use of multi-core clusters and has the advantage of performance predictability. Its target architecture are clusters, which are offered by cloud computing virtual machines. Pregel.NET [27] is based on Google's Pregel [23], offering distributed graph programming on the Azure Cloud using Bulk Synchronous Parallel model. It works with partitioning and scheduling of activities to workers in a Cloud environment, making use of the elasticity of virtual machines. Mansouri et al. [24] proposed task migration of a DSP (Digital

Table 6.1: Related work comparison: F1 - Changing the application code; F2 - Platform: Grid or Cluster; F3 - Automatic selection of migratable processes, *i.e.*, without user intervention; F4 - Use of computation metric (CPU load, CPU time or processing time) for load balancing; F5 - Use of communication data for load balancing; F6 - Use of migration costs for load balancing; F7 - Combination of metrics for load balancing purposes; F8 - Support for BSP applications; F9 - Support of any kind of adaptivity on dynamic environments; F10 - Support for heterogeneous systems; F11 - Process migration capability

Proposal	Features										
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
MigBSP [28]	No	All	No•	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HAMA [1]	HDFS	Cluster	No	No	No	No	No	Yes	No	Yes	Yes
PUB [7]	No	All	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes
MulticoreBSP [37]	Yes	No	NA†	No	No	NA†	No	Yes	No⋆	No⋆	No
Mizan [19]	No	Cluster	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes
BSPCloud [21]	No	Cluster	No	No	No	No	Yes	Yes	Yes	No	Yes
DistPM [20]	No	All	Yes	No	No	No	Yes	No	Yes	Yes	Yes
Pregel.NET [27]	Yes	Cloud	Yes	No	No	No	No	Yes	Yes	Yes	Yes
CPU-GPU cluster [24]	Yes	Cluster	Yes	No	No	No	No	Yes	Yes	Yes	Yes

References: • Depends on user definition at the beginning of the application; ⋆ Unknown; † Not Applicable.

Signal Processing) application implemented with the BSP computing model on a CPU-GPU cluster. During the processing phase of a BSP superstep, instead of moving the heavily loaded processes to another CPU, part of the load is divided to run in different GPUs. In this way, this middleware avoids network interaction, saving time on such operation. Unlike distributed systems, MulticoreBSP [37] library targets shared-memory computing employing thread-based parallelization. Finally, DistPM [20] is a library particularly developed to support process migration in grid computing. DistPM manages the network communication to avoid high data interaction between different clusters.

Table 6.1 summarizes the analysis of the aforementioned systems. We observe that our previous work named MigBSP is competitive among the BSP libraries regarding the load balancing perspective. Only MigBSP combines computation, communication and migration costs metrics for migration decision-making. Although having a process running in a slow processor that has a communication consistent pattern with a specific cluster, the migration penalties can act against migration viability, being dependent of process' size and network characteristics. The MigBSP's drawback considers how it selects the migratable processes, where now needs the intervention of user. In this way, both MigCube and MigHull proposed in this article seek to fill the MigBSP's gap, which is being used today to run BSP-based weather forecast and oil prospection applications in the south of Brazil [30].

7. Conclusion. Considering that the bulk synchronous style is a common organization on writing successful parallel programs [7, 10, 17], MigCube and MigHull emerge as alternatives for selecting their processes for running on more suitable resources without interference from the users. The key contribution of the proposed heuristics is the efficient use of computation, communication and migration costs metrics as axes values in the computational geometry for process migration decision-making. As mentioned above, MigCube and MigHull are not restricted to the MigBSP's scope, being employed to manage both heterogeneity and dynamism with process migration effortlessly at middleware level. Many data analysis techniques, such as machine learning and graph algorithms, require iterative computations and this is where Bulk Synchronous Parallel model can be more effective than MapReduce or Divide-and-Conquer strategies. The results showed gains larger than 40% when using MigCube or MigHull to decide process rescheduling in a subset of the Grid5000 environment. In addition, we also demonstrated a mean overhead close to 3% when employing the heuristics, but not performing any migrations. The evaluation emphasized the capacity of both proposed heuristics with different initial processes-processors mappings over an heterogeneous cluster-based grid.

Thus, future work includes the use of dynamism at resource and network usage levels to analyze MigCube and MigHull reactivity and overhead. The Lattice-Boltzmann application was very useful to present the benefits of the aforementioned heuristics in front of the originals presented in MigBSP, but we plan to evaluate the new

proposals on new complex applications including weather prediction and DNA sequencing [30]. Moreover, the use of a simulator was very convenient to evaluate the MigCube and MigHull feasibility. In this way, also as future work, we are analyzing communication libraries such as ProActive[3] and AMPI[4] to implement MigBSP and the proposed heuristics. Consequently, real tests in the Grid5000 infrastructure will be conducted and compared with data obtained at simulation level.

Acknowledgements. This work was partially supported by the following Brazilian Agencies: CAPES, FAPERGS and CNPq.

REFERENCES

[1] *Hama*, June 2013. Available at: http://hama.apache.org/. Access: Jun. 2013.
[2] *Simgrid*, June 2013. Available at: http://simgrid.gforge.inria.fr/. Access: Jun. 2013.
[3] V. Arabnejad, A. Moeini, and N. Moghadam, *Using bee colony optimization to solve the task scheduling problem in homogenous systems*, International Journal of Computer Science Issues, 8 (2011), pp. 348–353.
[4] S. Bandyapadhyay, S. Bhowmick, and K. Varadarajan, *Approximation schemes for partitioning: Convex decomposition and surface approximation*, in Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '15, SIAM, 2015, pp. 1457–1470.
[5] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, *The quickhull algorithm for convex hulls*, ACM Trans. Math. Softw., 22 (1996), pp. 469–483.
[6] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed., 2008.
[7] O. Bonorden, *Load Balancing in the Bulk-Synchronous-Parallel Setting using Process Migrations*, 2007 IEEE International Parallel and Distributed Processing Symposium, (2007), pp. 1–9.
[8] O. Bonorden, B. Juurlink, I. von Otte, and I. Rieping, *The paderborn university bsp (pub) library*, Parallel Comput., 29 (2003), pp. 187–207.
[9] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, *A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems*, J. Parallel Distrib. Comput., 61 (2001), pp. 810–837.
[10] R. E. De Grande and A. Boukerche, *Dynamic balancing of communication and computation load for hla-based simulations on large-scale distributed systems*, J. Parallel Distrib. Comput., 71 (2011), pp. 40–52.
[11] G. El Kabbany, N. Wanas, N. Hegazi, and S. Shaheen, *A dynamic load balancing framework for real-time applications in message passing systems*, International Journal of Parallel Programming, 39 (2011), pp. 143–182.
[12] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno, *2d euclidean distance transform algorithms: A comparative survey*, ACM Comput. Surv., 40 (2008), pp. 2:1–2:44.
[13] Z. Fan, H. Shen, Y. Wu, and Y. Li, *Simulated-annealing load balancing for resource allocation in cloud environments*, in Proceedings of the 14th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'13), New York, NY, USA, 2013, IEEE, pp. 1–6.
[14] V. Gaba and A. Prashar, *Comparison of processor scheduling algorithms using genetic approach*, International Journal of Advanced Research in Computer Science and Software Engineering, 2 (2012), pp. 37–45.
[15] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1990.
[16] S. Hashemi and A. Hanani, *Solving the scheduling problem in computational grid using artificial bee colony algorithm*, Advances in Computer Science: an International Journal, 2 (2013), pp. 37–41.
[17] B. Hendrickson, *Computational science: Emerging opportunities and challenges*, Journal of Physics: Conference Series, 180 (2009), p. 012013.
[18] S. Kardani-Moghaddam, F. Khodadadi, R. Entezari-Maleki, and A. Movagha, *A hybrid genetic algorithm and variable neighborhood search for task scheduling problem in grid environment*, Procedia Engineering, 29 (2012), pp. 3808–3814.
[19] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, *Mizan: a system for dynamic load balancing in large-scale graph processing*, in Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys '13, New York, NY, USA, 2013, ACM, pp. 169–182.
[20] Y. Li and Z. Lan, *A novel workload migration scheme for heterogeneous distributed computing*, in Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on, vol. 2, 2005, pp. 1055–1062.
[21] X. Liu, W. Tong, and Y. Hou, *BSPCloud: A Programming Model for Cloud Computing*, 2012 IEEE 12th International Conference on Computer and Information Technology, (2012), pp. 1109–1113.
[22] A. Madureira, F. Santos, and I. Pereira, *Self-managing agents for dynamic scheduling in manufacturing*, in GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation, New York, NY, USA, 2008, ACM, pp. 2187–2192.
[23] G. Malewicz, M. Austern, and A. Bik, *Pregel: a system for large-scale graph processing*, Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, (2010), pp. 135–145.

[3] http://proactive.activeeon.com
[4] http://charm.cs.illinois.edu/research/ampi

[24] F. Mansouri, S. Huet, V. Fristot, and D. Houzet, *Task migration of DSP application specified with a DFG and implemented with the BSP computing model on a CPU-GPU cluster*, Proccedings of the 2013 Conf. on Design and Architectures for Signal and Image Processing (DASIP), IEEE, 2013, pp. 326-333.

[25] M. F. Pace, *BSP vs mapreduce*, Procedia Computer Science, 9 (2012), pp. 246 – 255. Proceedings of the International Conference on Computational Science, ICCS 2012.

[26] J. Pecero and P. Bouvry, *An improved genetic algorithm for efficient scheduling on distributed memory parallel systems*, in Proceedings of the International Conference on Computer Systems and Applications (AICCSA), New York, NY, USA, 2010, IEEE, pp. 1–8.

[27] M. Redekopp, Y. Simmhan, and V. Prasanna, *Optimizations and analysis of bsp graph processing models on public clouds*, in Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013, pp. 203–214.

[28] R. d. R. Righi, L. Graebin, and C. A. da Costa, *On the replacement of objects from round-based applications over heterogeneous environments*, Software: Practice and Experience, (2015), v. 45, n. 5, pp. 633-656.

[29] C. Schepke and N. Maillard, *Performance improvement of the parallel lattice boltzmann method through blocked data distributions*, in Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on, Oct 2007, pp. 71–78.

[30] J. Schneider, J. Gehr, H.-U. Heiss, T. Ferreto, C. De Rose, R. Righi, E. Rodrigues, N. Maillard, and P. Navaux, *Design of a grid workflow for a climate application*, in Computers and Communications, 2009. ISCC 2009. IEEE Symposium on, pp. 793–799.

[31] E.-G. Talbi, *Metaheuristics : from design to implementation*, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2009.

[32] H. Tang, Y. Zhou, X. Huang, and G. Rong, *Does pareto's law apply to evidence distribution in software engineering? an initial report*, in Proceedings of the 3rd International Workshop on Evidential Assessment of Software Technologies, EAST 2014, New York, NY, USA, 2014, ACM, pp. 9–16.

[33] L. G. Valiant, *A bridging model for parallel computation*, Commun. ACM, 33 (1990), pp. 103–111.

[34] H. Wu and C. Nie, *An overview of search based combinatorial testing*, in Proceedings of the 7th International Workshop on Search-Based Software Testing, SBST 2014, New York, NY, USA, 2014, ACM, pp. 27–30.

[35] A. Yzelman, R. Bisseling, D. Roose, and K. Meerbergen, *Multicorebsp for c: A high-performance library for shared-memory parallel programming*, International Journal of Parallel Programming, (2013), pp. 1–24.

[36] K. Ponnavaikko and J. Dharanipragada, *Wide Area Distributed Filesystems - A Scalability and Performance Survey*, Scalable Computing: Practice and Experience (SCPE), v.11, n.3, (2010), pp. 305-325.

[37] A. Yzelman and R. H. Bisseling, *An object-oriented bulk synchronous parallel library for multicore programming*, Concurrency and Computation: Practice and Experience, 24 (2012), pp. 533–553.

[38] E. Cesario and D. Talia, *Using Grids for Exploiting the Abundance of Data in Science*, Scalable Computing: Practice and Experience (SCPE), v.11, n.3, (2010), pp. 251-261.

Edited by: Pedro Valero Lara
Received: Sept 9, 2015
Accepted: March 2, 2016

DOI 10.12694/scpe.v17i1.1148
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 1, pp. 33–46. http://www.scpe.org

MANY-TASK COMPUTING ON MANY-CORE ARCHITECTURES

PEDRO VALERO-LARA,* POORNIMA NOOKALA,† FERNANDO L. PELAYO,‡ JOHAN JANSSON § SERAPHEIM DIMITROPOULOS,¶ AND IOAN RAICU‖

Abstract. Many-Task Computing (MTC) is a common scenario for multiple parallel systems, such as cluster, grids, cloud and supercomputers, but it is not so popular in shared memory parallel processors. In this sense and given the spectacular growth in performance and in number of cores integrated in many-core architectures, the study of MTC on such architectures is becoming more and more relevant. In this paper, authors present what are those programming mechanisms to take advantages of such massively parallel features for the particular target of MTC. Also, the hardware features of the two dominant many-core platforms (NVIDIA's GPUs and Intel Xeon Phi) are also analyzed for our specific framework. Given the important differences in terms of hardware and software in our two many-core platforms, we have considered different strategies based on CUDA (for GPUs) and OpenMP (for Intel Xeon Phi). We carried out several test cases based on an appropriate and widely studied problem for benchmarking as matrix multiplication. Essentially, this study consisted of comparing the time consumed for computing in parallel several tasks one by one (the whole computational resources are used just to compute one task at a time) with the time consumed for computing in parallel the same set of tasks simultaneously (the whole computational resources are used for computing the set of tasks at very same time). Finally, we compared both software-hardware scenarios to identify the most relevant computer features in each of our many-core architectures.

Key words: Parallel Computing, Multi-Task Computing, Many-Core, GPU, Intel Xeon Phi, CUDA, OpenMP

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction.

Many-Task Computing (MTC), the execution of multiple tasks on one particular parallel platform at very same time, is historically dominated by some parallel platforms such as clusters, grids, and supercomputers. However, the advances in hardware, in particular in many-core architectures, for MTC applications is a relevant topic. Again, the main problem is at the software side. Programmers need to address the challenge of analyzing and studying the different hardware features to efficiently map the MTC applications in order to achieve the best performance on such architectures.

The main contribution of the present work is twofold. While, on one hand the first motivation consists of presenting those approaches and mechanisms to efficiently exploit Multi-Task Computing applications on current many-core architectures. Secondly, but not least important, authors provide a study to clarify what are the most amenable features of the two dominant many-core systems today, these are NVIDIA GPUs and Intel Xeon Phi, for the specific target of MTC.

In the last years, the use of scheduler based on many-core or heterogeneous architectures for general or for specific applications has been widely studied [26, 48]. S. Yamagiwa et al. [48] propose a GPGPU streaming based on distributed computing environment; S. Nakagawa et al. [26] provide a new middleware capable of out-of-order execution of works and data transfers using stream processing. Other works [12, 46] follow a similar strategy based on streaming to minimize data transfers overhead. S. Kato et al. [21] introduce *TimeGraph*, a GPU scheduler composed by two different GPU scheduling policies which allow to interrupt the low priority tasks execution in order to execute higher priority tasks within a real-time multi-tasking environments for video applications. Similar to the previously mentioned works and considering that the GPUs in a cluster are not usually fully utilized, Duato et al. [9] present their *rCUDA*, a middleware that enables CUDA remoting over a commodity network by allowing to use CUDA-compatible GPUs installed in a remote computer, as, they were installed in the computer where the application is being executed. Also, V. J. Jiménez et al. [19] present a sort of predictive runtime scheduling which supports several scheduling algorithms in order to choose the appropriate platform (Multicore, GPU, etc.) in which the algorithm would be better executed, resulting in

*Univeristy of Manchester, UK, and Basque Center for Applied Mathematics (BCAM), Bilbao, Spain (pvalero@bcamath.org).
†Illinois Institute of Technology (IIT), Chicago, USA (pnookala@hawk.iit.edu.)
‡University of Castilla-La Mancha (UCLM), Albacete, Spain (fernandol.pelayo@uclm.es.)
§Basque Center for Applied Mathematics (BCAM), Bilbao, Spain, and KTH Royal Institute of Technology, Stockholm, Sweden (jjansson@bcamath.org).
¶Illinois Institute of Technology (IIT), Chicago, USA (sdimitro@hawk.iit.edu.)
‖Illinois Institute of Technology (IIT), Chicago, USA (iraicu@cs.iit.edu.)

almost fully usage of CPU/GPU-like systems, with a peak time reduction of 40% with respect to only using the GPU. Basically most the aforementioned works take advantage of overlapping memory transfers among CPU and GPU memories with single kernel executions.

With the aim of exploiting MTC on many-core, other authors [23, 25] have studied the efficiency of this new feature. *Batched task*, maybe the first MTC approach on GPUs, allows us to run several independent kernels over the same GPU simultaneously. It was presented by M. Guevara et al. [14] and P. Valero-Lara et al. [41]. Posteriorly, C. Gregg et al. [13] and K. Zhang et al. [49] included a scheduler which can select the best matching among tasks before running. Additionally, P. Valero-Lara et al. [42] applied this strategy to different GPU architectures to obtain the most convenient architectural features for running concurrent kernels. After that, in [40], it is proposed a new heterogeneous (CPU-GPU) scheduler in which groups of independent blocks of tasks were efficiently managed to fully use CPU-GPU and reduce the overhead of memory transfers. More recently, S. Krieder et al. [24] presented *GeMTC*, a CUDA based framework which allows MTC workloads to run efficiently on NVIDIA's GPUs. Posteriorly P. Nookala et al. [27] adapted this framework (*GeMTC*) to efficiently use the particular features of Intel Xeon Phi and evaluate MTC applications on Intel accelerators. In fact, we can now find some applications which takes advantage of MTC on hardware accelerators. One of these applications was presented by P. Valero-Lara et al. [43, 44], in which multiples tridiagonal problems are efficiently executed on the same NVIDIA's GPU simultaneously. Other examples consist of computing several relatively small Linear Algebra problems [16, 15, 17], multiple range queries in metric spaces [2, 1] or multiple string matching [22].

This work is structured as follows: Section 2 introduces the main features of many-core architectures considered (NVIDIA GPUs and Intel Xeon Phi), then in Section 3, authors briefly outline the different mechanisms for MTC on both many-core architectures. After that, both platforms and the MTC mechanisms are deeply analyzed and studied. Finally, Section 5 concludes summarizing the most relevant results.

2. Many-Core Architectures. Today, the increase in performance for single-threaded processor has come to an end due to the limitation of the current Very Large Scale Integration (VLSI) technology. In response, most hardware companies are designing and developing new parallel architectures [11]. Programs will only increase in performance if they use and exploit the new parallel characteristics of new architectures. On the other hand, multicore designs are also encountering scaling problems, notably the "Dark Silicon" phenomenon [10]. Power and cooling concerns suggest the number of dynamically active transistors on a single die may be greatly constrained in the near future. In other words, even if the number of transistors per chip continues to follow Moore's law, we will not be able to use all of them simultaneously. This problem may lead to scenarios in which only a small percentage of the chip's transistors can be "on" at a time [34]. Given the limitations of current CMOS technology and the excessive power consumption reached by current platforms, it is necessary a renewal of hardware design.

In this context, many-core architectures may be an answer to these challenges. These new massively parallel platforms offer a high ratio performance/cost and an efficient power consumption design [39, 38, 37]. They are also widely used on high performance computing, including systems ranging from cluster of personal computers, to large scale supercomputers.

Most processors for high-performance computing (HPC) are still multi-core. However, as we can see in Top 500 list [36], many of the most powerful supercomputers today are based on platforms that combine multicore processors with data parallel accelerators. The fastest system, which is currenlty the Tianhe-2 supercomputer from China, uses Intel's Xeon Phi coprocessors and its runnerup, which is the Titan supercomputer from the Oak Ridge National Laboratory, uses NVIDIA GPUs.

2.1. Graphic Processing Units (GPUs). GPUs are traditionally used for interactive applications, and are designed to achieve high rasterization performance however, their characteristics have allowed the opportunity to other more general applications to be accelerated in GPU-based platforms. This trend is now called General Purpose Computing on GPU (GPGPU) [31], or what is the same, the usage of GPUs for applications for which they were not originally designed. These general applications must have parallel characteristics and an intense computational load to obtain a good performance.

The main feature of these devices is a large number of processing elements integrated into a single chip at the expense of a significant reduction in cache memory. These processing elements are arranged on memory

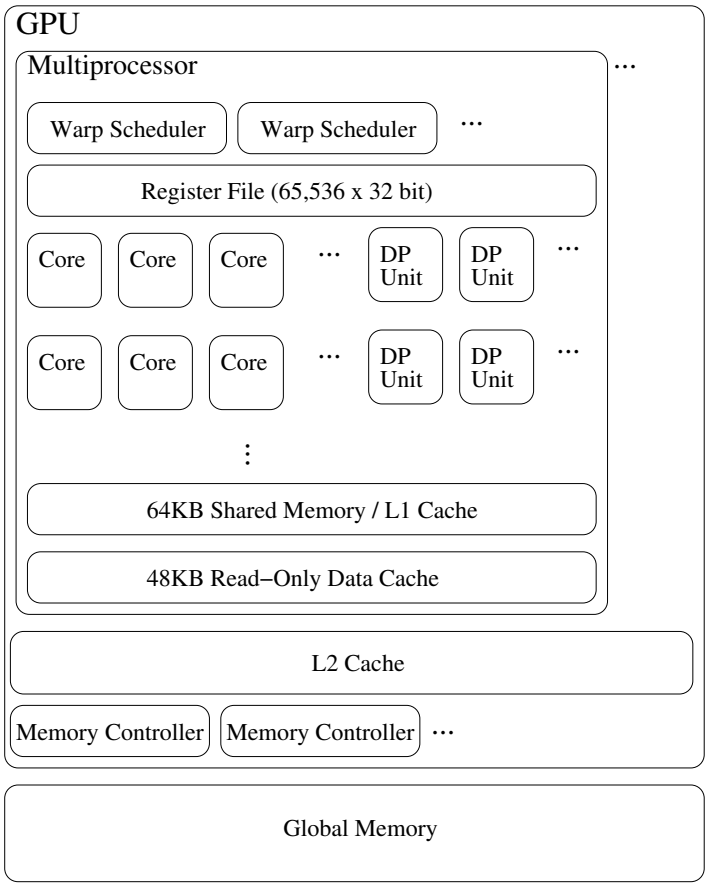

Fig. 2.1: NVIDIA GPU (Kepler) architecture [45].

cards that have a local high-speed external DRAM and are connected to the computer through a high-speed I/O interface (PCI Express).

Figure 2.1 shows an abstract block diagram of NVIDIA's (Kepler) GPU [45]. The GPU is organized into several multiprocessors, which in turn are composed of various simple processors (cores) that operates in SIMD fashion. The multiprocessors have fine grain multithreading capabilities, which means that they support hundreds of threads in-fly. Every multiprocessor switches to a different set of threads every clock cycle, which helps to maximize computational resources and hide the long latency memory accesses to a share GPU main memory.

The GPU main memory, usually called "global memory", is banked, which allows the hardware to coalesce several simultaneous memory accesses to adjacent positions into a single memory transaction. In addition, each multiprocessor contains a large set of registers and an on-chip SRAM scratchpad memory, i.e., a software controlled cache, to speed up data access. In more recent GPUs (starting from NVIDIA's Fermi architecture) the SRAM can be configured either as scratchpad or as cache memory and the user decide, with certain restrictions, the size of both memories. These newer GPUs also incorporate a L2 cache common to all multiprocessors. The access to the global memory can also be performed through special read-only two level hierarchy of so called texture caches, that are optimized to capture 2D access patterns [47].

2.2. Intel Xeon Phi. GPUs have a very restrictive programming model, but provide at least an order of magnitude better throughput for applications painstakingly coded to that model. To program GPUs, typically there is a need to learn another programming language such as CUDA (NVIDIA) or OpenCL (AMD). As

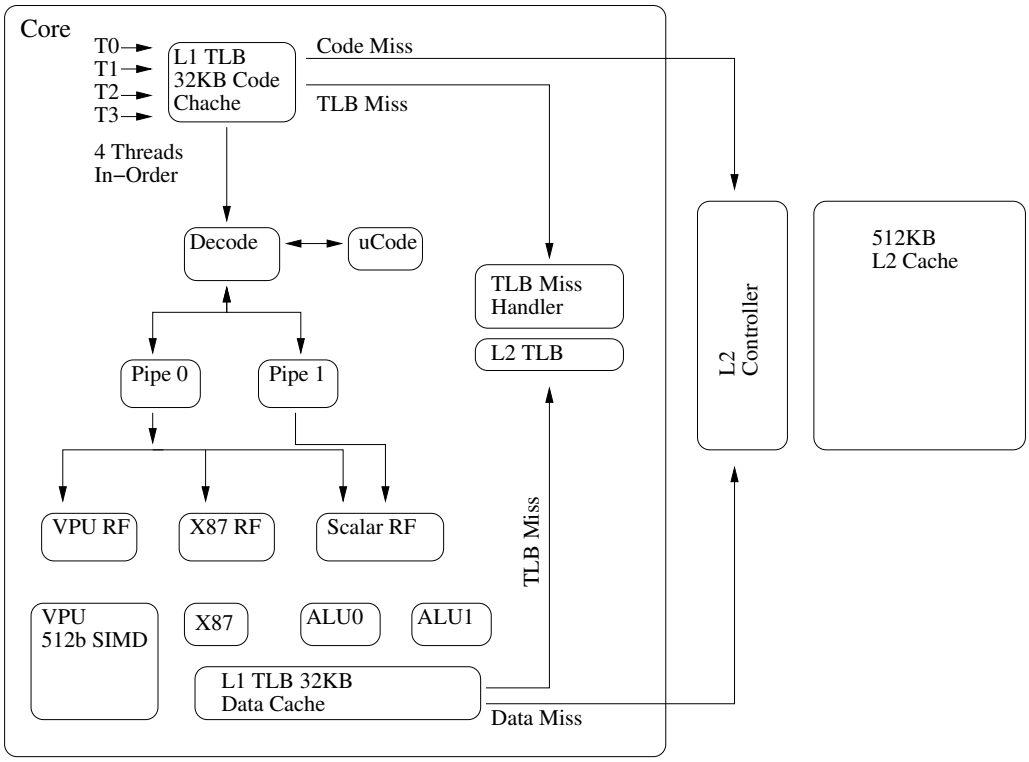

Fig. 2.2: Architecture of a single Intel Xeon Phi Core [18].

a result, existing vendors must spend extra time and effort to modify or rewrite parts of their codebase to take advantage of the new capabilities provided by General Purpose GPUs (GPGPUs). Besides that, barely rewriting an application just to offload computations to a GPU rarely works well. Because of the architecture of most GPUs out there, applications must be tailored from the ground up to follow the rules of the restrictive programming model of GPUs, otherwise they may suffer from severe performance penalties. Because of that, interested vendors cannot afford to go through the effort involved. Finally, while GPUs are great for massively parallel applications with thread- switching that comes almost at no cost, their performance can take a large hit when executing programs with complex logic (like complicated branching and looping for example). Therefore they may be unsuitable for certain applications of MTC. The Intel Xeon Phi is a new family of processors based on the Intel MIC Architecture [18] that incorporates earlier work on the Larrabee architecture [33]. It follows an alternative programming model that, although may not provide the same level of parallelism, provides more flexibility and therefore can be more suitable for certain application of MTC that GPUs are not suited for. The reason is that the Xeon Phi has x86 cores that are more capable (can handle complex branching and looping) than most GPU cores. Another advantage of having x86 cores is that programming the coprocessor minimizes the amount of work that needs to be done in order to integrate a Xeon Phi to an existing system. That is because the Phi does not require being programmed in any specific framework and it can natively run applications written in C with Pthreads or OpenMP. All of the above facts were enough to motivate us to work on this project. We have used the 22nm Knights Corner chip graphically described in Figures 2.2 and 2.3, which was the first commercial product from this family.

The Corner is a PCIe vector co-processor with integrates up to 61 in-order dual issue x86 cores, which trace some history to the original Pentium core, like the Larrabee predecessor. Among other enhancements, the Corner's cores are augmented with 64-bit support, 4 hardware threads per core (resulting in more than 200 hardware threads available on a single device) and 512-bit SIMD instructions [18]. Each core has a 512KB

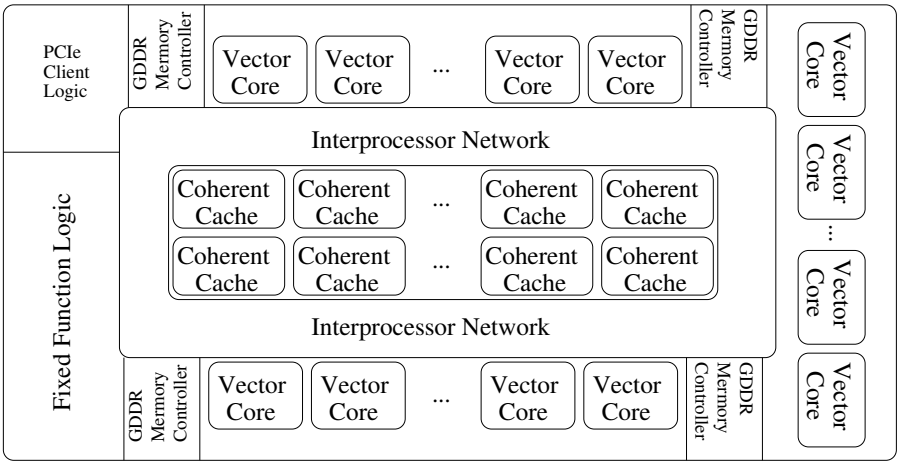

Fig. 2.3: Micro-architecture of the Entire MIC coprocessor [18].

L2 cache locally but has also access to all other L2 caches in the system through a high-speed bidirectional ring [18]. Unlike previous GPUs, the L2 cache is kept fully coherent by a global-distributed tag directory.

The performance achieved by Knight Corner chips is usually outperformed by NVIDIA's counterparts [29]. However, last year Intel announced the Knight Landing processor [35] that should significantly improve MIC performance.

3. Multi-Task Computing (MTC). Although, the dominant choice to compute MTC problems continues being distributed memory architectures, the impressive growth in performance of current parallel shared memory architectures makes possible to compute a considerable high number of independent tasks over this kind of computational platforms.

In this regard, this section introduces some of the most extended and known approaches for computing MTC over many-core architectures (NVIDIA GPUs and Intel Xeon Phi). Essentially, each of these approaches share the same major steps (Figure 3.1). These consist of performing memory transfers (communication) sequentially, then the set of independent tasks are executed on many-core platform.

3.1. MTC on GPU. This subsection introduces 3 different approaches for MTC running on NVIDIA's GPUs. To clarify, we include some pseudocodes which can help us to understand the differences among them and the particular features (advantages and disadvantages) of every of them.

Batched Task, several authors [13, 14, 41, 42] proposed this strategy to fully utilize the GPU by running multiple tasks (kernels) simultaneously on the same GPU. Basically, all of them include a single pass compiler which is able to create the batched task source code by renaming the variables, by adding the if-else control flow, and by adding indexing in the independent task that is executed by blocks that are offset from *blockId*. The set of tasks are mapped either on one or on a set of blocks of threads. In this case, the number of threads launched must be equal than the sum of all threads required by all tasks. Also, all parameters must be included in the same call. We would like to point out that this strategy can be carried out on all CUDA GPUs architectures. Algorithm 1 illustrates a simple scheme of this strategy.

NVIDIA proposed a new approach [5, 40] called **Concurrent Kernels** which allows to execute a set of independent tasks (kernels) on the same GPU by means of streams. It only can be used over *FERMI* architecture forward. Using different streams for CUDA kernels makes the concurrent execution being possible. Therefore n kernels on n streams could theoretically run concurrently if they are fitted into the hardware. This approach allows to execute up to 16 different kernels at the very same time. A scheme of this approach is shown in Algorithm 2, where two kernels are used in such way.

Recently, NVIDIA has introduced a new feature compatible for the *KEPLER* CUDA architecture (**Dy-**

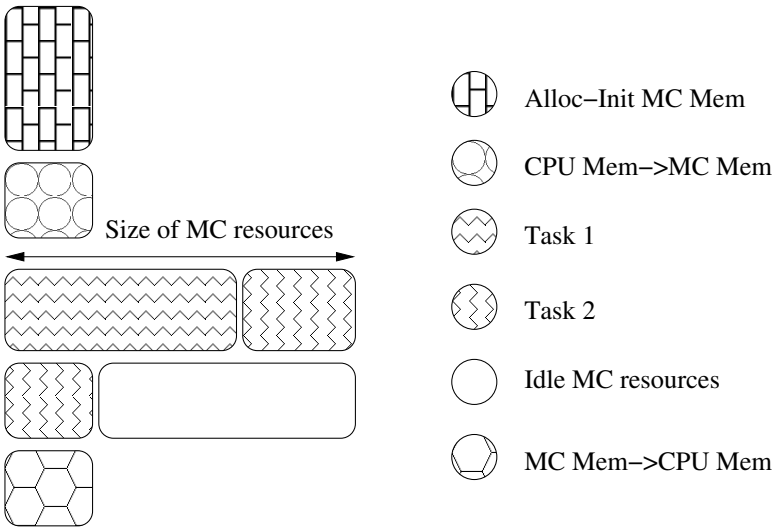

Fig. 3.1: Basic scheme for MTC on Many-Core (MC) architectures. Allocation and Initialization of MC memory (Alloc-Init MC Mem). Data transfer (input) from CPU memory to MC memory (CPU Mem → MC mem). MC executions (Task 1 and Task 2). Idle cores in MC (Idle MC resources). Data transfer (output) from MC memory to CPU mem (MC Mem → CPU Mem).

Algorithm 1 Batched Task.

`BatchedTaskCPU`
1: CPUMemAllocate(A_{CPU},B_{CPU},C_{CPU},D_{CPU})
2: GPUMemAllocate(A_{GPU},B_{GPU},C_{GPU},D_{GPU})
3: CPU->GPUMemTransfer(A_{CPU},A_{GPU})
4: CPU->GPUMemTransfer(C_{CPU},C_{GPU})
5: CPU->GPUMemTransfer(D_{CPU},D_{GPU})
6: BatchedTask<THREADS1+THREADS2> (A_{GPU},B_{GPU},C_{GPU},D_{GPU})
7: GPU->CPUMemTransfer(B_{GPU},B_{CPU})
8: GPU->CPUMemTransfer(D_{GPU},D_{CPU})
`BatchedTaskGPU(A,B,C,D)`
9: i = index of thread
10: j = index of block
11: **if** j = 0 **then** ▷ kernel1
12: B[i] = A[i] + 100
13: **else if** j = 1 **then** ▷ kernel2
14: D[i] = C[i] × D[i]
15: **end if**

namic Parallelism [6]) which allows to manage multiple tasks inside GPU. It is supported via an extension of the CUDA programming model that enables a CUDA kernel to create and to synchronize new nested work. CUDA kernel can consume the output from the other kernels (childs) without CPU involvement. It requires at least two-level task running (parent-child).

3.2. MTC on Intel Xeon Phi. Due to the foundations of Intel architecture, the coprocessor can be programmed in several different ways [32]. Here we introduce two different approaches, one using *OpenMP* and one using *SCIF* (Intel's Symmetric Communication Interface). *OpenMP* implementation uses offloading approach for offloading computations from host to the accelerator. The *SCIF* implementation runs natively

Algorithm 2 Concurrent kernels.

```
   ConcurrentKernelsCPU
 1: CUDAStream Stream[2]
 2: CPUMemAllocate(A_CPU,B_CPU,C_CPU,D_CPU)
 3: CPU->GPUMemTransfer(A_CPU,A_GPU)
 4: CPU->GPUMemTransfer(C_CPU,C_GPU)
 5: CPU->GPUMemTransfer(D_CPU,D_GPU)
 6: for i = 1 → 2 do
 7:     StreamCreate(Stream[i])
 8: end for
 9: Kernel1<THREADS1>(A_GPU,B_GPU,Stream[1])
10: Kernel2<THREADS2>(C_GPU,D_GPU,Stream[2])
11: for i = 1 → 2 do
12:     StreamDestroy(Stream[i])
13: end for
14: GPU->CPUMemTransfer(B_GPU,B_CPU)
15: GPU->CPUMemTransfer(D_GPU,D_CPU)
```

Algorithm 3 Dynamic parallelism.

```
   DynamicParallelismCPU
 1: CPUMemAllocate(A_CPU,B_CPU,C_CPU,D_CPU)
 2: GPUMemAllocate(A_GPU,B_GPU,C_GPU,D_GPU)
 3: CPU->GPUMemTransfer(A_CPU,A_GPU)
 4: CPU->GPUMemTransfer(C_CPU,C_GPU)
 5: CPU->GPUMemTransfer(D_CPU,D_GPU)
 6: DynamicParallelismKernel<1> (A_GPU,B_GPU,C_GPU,D_GPU)
 7: GPU->CPUMemTransfer(B_GPU,B_CPU)
 8: GPU->CPUMemTransfer(D_GPU,D_CPU)
   DynamicParallelismKernel(A,B,C,D)
 9: Kernel1<THREADS1>(A_GPU,B_GPU)
10: Kernel2<THREADS2>(C_GPU,D_GPU)
11: SyncronizeThreads
```

on the accelerator and accepts jobs from Clients running on the host. There are several advantages and disadvantages between the two methods. The major advantage of native execution coupled with *SCIF* over offloading is that the developer gets more control overall in the configuration and the architecture of their design in order to maximize performance. Computation does not necessarily have to be transferred back to to the CPU. In addition, different MIC cards can communicate directly with each other basically making certain designs more efficient.

Finally, frameworks that use offloading mode (*OpenMP*), do not necessarily take advantage of the DMA-features of the hardware they run on while on *SCIF* you are guaranteed that if you are using Remote Memory Access (RMA). That is not to say that *OpenMP* does not come with any advantages over *SCIF*. Quite the opposite, the advantages of offloading are pretty significant for the framework that was implemented for this project. The low-level C code needed for the *SCIF* implementation is relatively a lot more complex when compared with *pragma* directives provided by *OpenMP*. In addition, using *SCIF* implies that the framework must have at least one of its parts running natively on the Phi as the endpoint. In order to do that the developer needs to set up an application to run natively on the Phi and involves a lot of configuration. Using *OpenMP* with the offloading capabilities provided by the MIC, all this configuration is taken care of.

OpenMP version (Figure 3.2) uses asynchronous offloading capabilities. We have employed a Producer-Consumer architecture which communicates using shared memory for IPC (InterProcess Communication). The

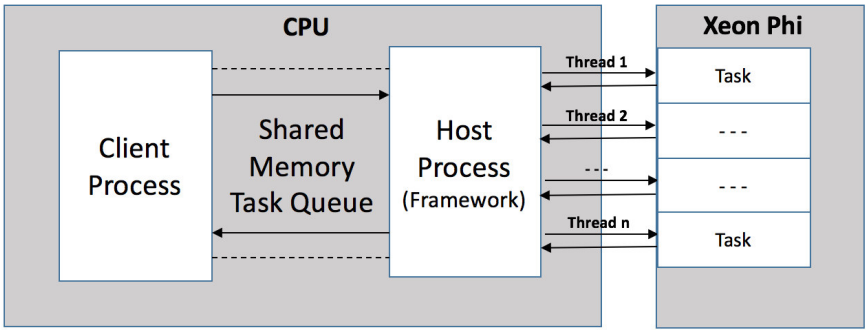

Fig. 3.2: OpenMP-Phi diagram.

Consumer side hosts the framework which runs as a single thread and launches as many master processes on the Xeon Phi as specified by the user [28]. The master processes use the shared memory space as a queue structure, continuously accepting new tasks from producer processes. Likewise, the producer acts as a client process which submits tasks to the queue via this shared memory pool. For testing this framework, we have implemented two different types of applications: Sleep and Matrix Multiplication. Both were developed and tested using the OpenMP approach of offloading tasks on to Xeon Phi. The master processes in our framework read the tasks from shared memory location and based on the type of task, offload the computation part to Xeon Phi. Asynchronous offloading is used to allow the framework to continue accepting tasks while other tasks are running. The Phi sends a signal back to the master processes after job execution has completed. At this point the output is sent back to the Client.

SCIF implementation employs a Client Server architecture [4] where clients send their tasks to the Phi from the host and the server, which runs natively on the Phi, accepts the jobs. After submitting the job, the clients can request the result and the server will deliver it to them when the task has finished processing and is placed on the results queue of the framework. The whole procedure is non-blocking for the server who can handle multiple requests and submissions at the same time. That functionality is implemented with *epoll()* [30] for handling connections that are later passed to threads [20, 24] that push or dequeue tasks from the queues. The *SCIF* socket-like API is used for communications between the server and the clients. It comes as a shared library named **libmtcq* [8]. This library includes all the functionality that handles incoming and outgoing queues of tasks, pushing jobs and distributing tasks to workers. It is also completely parametrizable in terms of queue sizes, worker threads, and application threads. Since the Xeon Phi does not have the hierarchical architecture of SMXs and Warps nor the concept of application kernels that you generally see in GPGPUs, everything is implemented with standard Pthreads. There is a parametrizable number of master threads that dequeues tasks from the incoming queue. If the task is a parallel application, which is the case most of the time, then the master thread will assign the task to the specified number of worker threads. Else if it is sequential only one thread will be assigned and the master thread will go back to dequeue more jobs. Each queue is implemented as a finite buffer from the Producer-Consumer model which means that it uses a single mutex and two semaphores to ensure that no deadlocks or data-races arise.

4. Performance Evaluation. This section presents a performance study to test and obtain what are the most relevant programming and hardware features for MTC. Each of the programming approaches and many-core architectures are analysed in deep.

4.1. NVIDIA GPUs. Taking into account that most many-core accelerators, especially GPUs, reach their optimum performance over problems having a high level of data parallelism together with a fine granularity, we have planned a set of tests over one suitable problem (in the previous sense) as matrices multiplication. The implementation used is an optimization of SGEMM method on GPU presented in [3] which incorporates

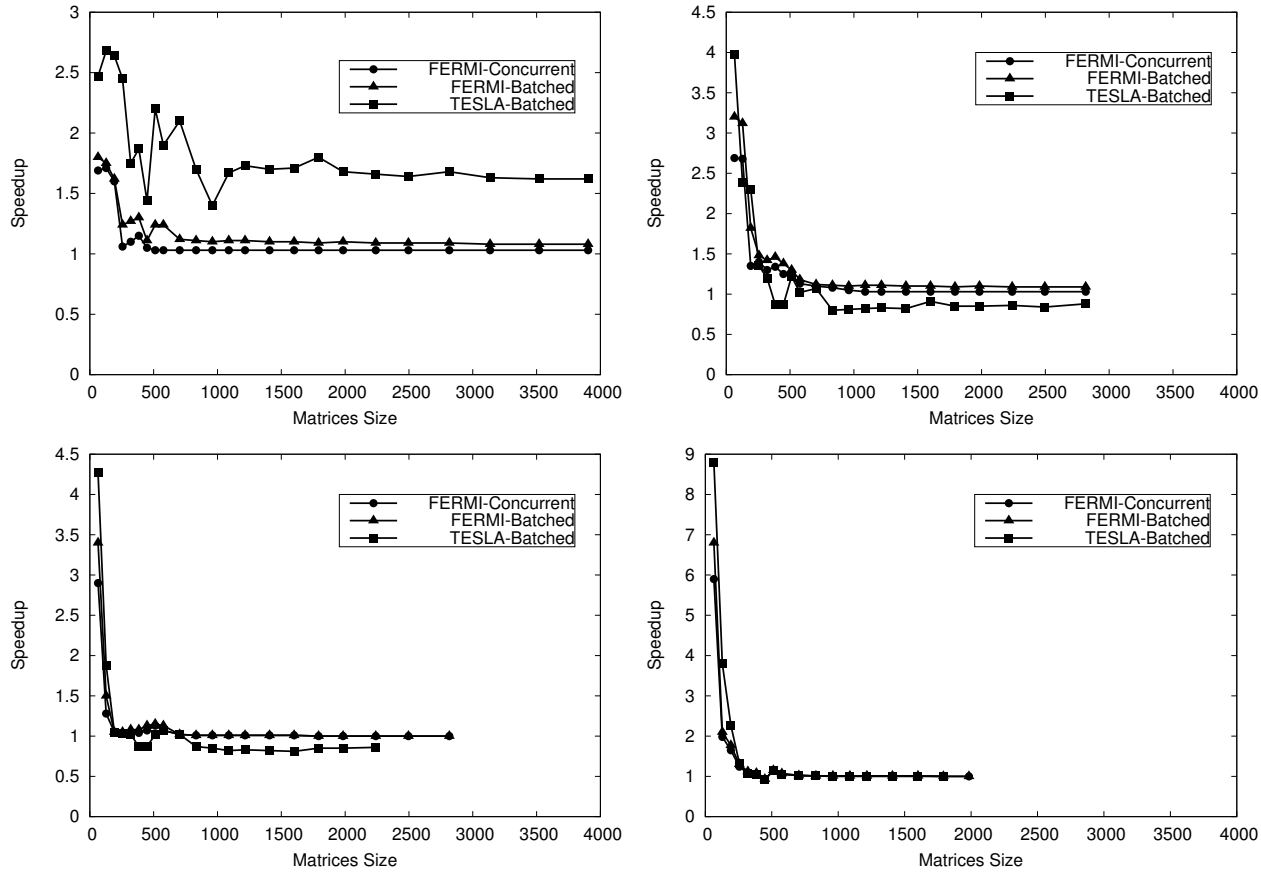

Fig. 4.1: Performance (speedup) achieved by the *Batched* (TESLA and FERMI architectures) and *Concurrent* (FERMI architecture) approaches.

several optimizations for NVIDA's GPUs, such as coalescing memory accesses and shared memory exploitation. Besides, we have considered the best choice concern with the size of threads blocks.

Four different test cases have been carried out which consist of computing 2, 4, 8 and 16 tasks (maximum number of tasks for FERMI architecture) in parallel on the same GPU. The size of matrices is increased in order to show the impact on performance by increasing memory requirements. Results are shown in terms of speedup (Figures 4.1 and 4.2), which is the ratio between the execution time when running (one by one) several tasks (matrix multiplication), and, the execution time when computing all multiplications (MTC approach) on the same GPU in parallel (Figure 3.1) according to each approach, *Dynamic, Concurrent* and *Batched*. Due to the memory capacity of TESLA and FERMI, some tests carried out in the first graph (Figure 3.1-Top) could not be included in the rest.

We have considered three different NVIDIA GPU architectures which we have re-called as TESLA (GT 200), FERMI (GF 100) and KEPLER (GK 110). Although, all of them share the major components, we can find important differences (Table 4.1). As we see later, these differences have important consequences in performance. Each of the CUDA-compatible MTC approaches (subsection 3.1) have been tested on our three GPU architectures, when it is possible. The study carried out in this subsection is an extension of the work previously presented in [41]. We include additional results using new MTC approaches on new GPU architectures. We evaluate the *Batched* and *Concurrent* approaches on our FERMI, as the *Dynamic* one is not compatible with this architecture. Otherwise, we can analyze all approaches on KEPLER GPU, as it is one of

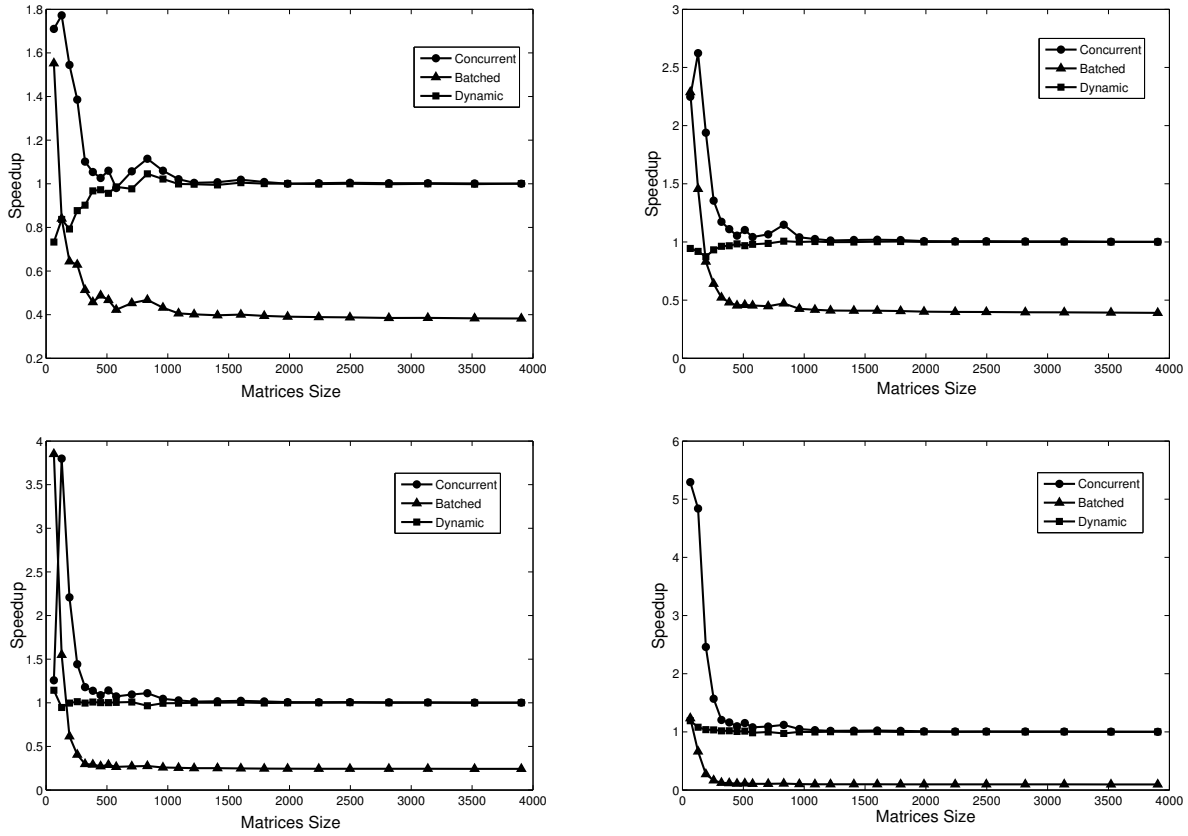

Fig. 4.2: Performance (speedup) achieved by each of the approaches (*Batched*, *Concurrent* and *Dynamic*) over KEPLER GPU architecture.

the newest NVIDIA GPUs. However, our TESLA GPU is only compatible with the *Batched* approach, as it is oldest architecture included in our study.

From the results obtained (Figures 4.1 and 4.2), we highlight several conclusions comparing each of the MTC approaches over each of the architectures. On FERMI both approaches, *Batched* and *Concurrent*, achieve a similar result, being faster the *Concurrent* one. Although the *Dynamic* scheduler is an easy to implement approach from programmer point of view, it does not show any benefit on KEPLER. In contrast, a better scalability is reached by using the other two approaches. The *Batched* approach presents a good scaling for small matrix sizes, however, it turns to be inefficient for bigger sizes. Otherwise, the best scaling is obtained by the *Concurrent* approach, even for medium matrix sizes.

Both approaches, *Batched* and *Concurrent*, share a similar trend on each of the architectures. First, we focus on the impact of matrix size in performance. In this regard, the best performance is reached in the first tests (small matrices). Obviously, the performance achieved on the first tests is degraded by increasing matrix what implies to increase also the number of threads per CUDA block. As consequence, the GPU resources (multiprocessors) are saturated so that a higher degree of parallelism can not be efficiently exploited. Second, we focus on analysing the trend in performance by increasing the number of tasks. Unlike the results achieved by increasing the matrix size, in which we appreciate an important fall in performance, we see the opposite scenario, that is the performance achieved is higher by increasing the number of tasks, at least for small matrix sizes.

Although all architectures are similar and share the major components, the most relevant variance among

Table 4.1: GTX 285, GTX 480 and K 20c hardware.

	GTX 285	GTX 480	K 20c
Code Name	GT 200(TESLA)	GF 100(FERMI)	GK 110 (KEPLER)
# Multiprocess. (MP)	30	15	13
# Cores/MP	8	32	192
# Cores	240	480	2496
Core Clock	648 Mhz	1401 Mhz	706 Mhz
Mem. Clock	1242 Mhz	1848 Mhz	2600 Mhz
Mem. Capacity	1 GB	1.5 GB	5 GB
On-chip Mem.			
SM (per MP)	-	16/48 KB	16/48 KB
L1 (per MP)	-	48/16 KB	48/16 KB
L2 (unified)	-	768 KB	768 KB
Mem. Bus	512 bits	384 bits	320 bits
Bandwidth	159 GB/s	177.4 GB/s	208 GB/s
Gigaflops (SP)	708	1344.96	4577

Table 4.2: Intel Xeon Phi hardware.

	Intel Xeon Phi
Code Name	5110P (PHI)
# Cores	60
Core Clock	1053 Mhz
Mem. Capacity	8 GB
On-chip Mem.	
L1 (per core)	32 KB
L2 (per core)	256 KB
L2 (coherent)	30 MB
Bandwidth	320 GB/s
Gigaflops (DP)	2022

them is found in the number of multiprocessors (Table 4.1). While KEPLER is composed by 13 multiprocessors, TESLA is composed by 30. Given these results, we can assume that this factor has a great influence in performance. Although the number of cores in KEPLER is more than 10× than in TESLA, it is remarkable, that the speedup reached by TESLA is up to almost 2× bigger than the speedup obtained by KEPLER. In this regard, the number of cores is not as relevant as the number of multiprocessors, at least in a MTC framework, since every core into one multiprocessor shares the same control component. As consequence, a greater number of small multiprocessors (TESLA) allows us to reach a better performance than a lower number of big multiprocessors (FERMI and KEPLER).

4.2. Intel Xeon Phi. All of our experiments were ran on the MidWay High- Performance Computing Cluster at University of Chicago. Our testing host is an Intel SandyBridge with 16 cores at 2.6 Ghz and 32 GB of RAM. It has 2 Xeon Phis attached to it (Table 4.2). In this subsection, we focused on analyzing the Matrix Multiplication tasks by using the *OpenMP* approach (Figure 3.2), as the *SCIF* implementation is under development and work is being carried out to run some experiments using Sleep and Matrix Multiplication tasks to measure the performance of the framework.

In order to assess the real-world performance of the Xeon Phi [7], we developed a matrix multiplication application to show how well it performed for various task sizes and levels of concurrency. It should be noted that the work performed is exponentially greater than the matrix size, since a naive matrix multiplication

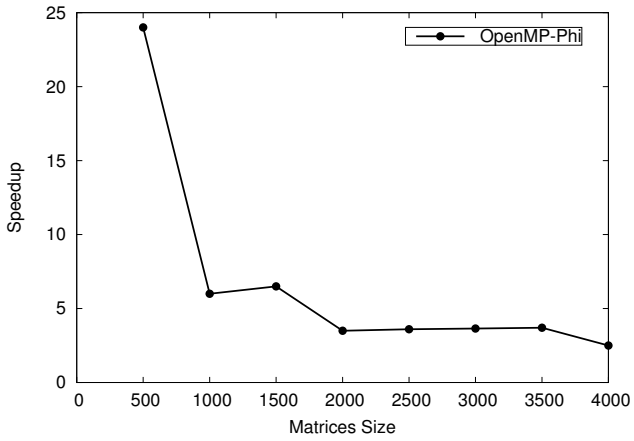

Fig. 4.3: Performance (speedup) achieved by each Intel Xeon Phi by using the *OpenMP* approach.

algorithm of $O(n^3)$ was used.

We used the same speedup used previously for the GPUs analysis, time consumed computing a set of tasks one by one over the time consumed by computing the same set of tasks in parallel at very same time. In particular for our MTC approach and ensuring full utilization of Xeon Phi, we use blocks of 60 tasks. Each of the tasks is mapped on one Phi-core in which we use 4 hardware-threads, $60 \times 4 = 240$ total threads). When one task completes, we send another until 600 tasks are computed. Speedup was calculated by varying the matrix sizes, number of threads and also by varying the level of concurrency of tasks. It was observed (Figure 4.3) that higher performance is achieved with very granular tasks, but the gain reduces as problem scales up to higher matrix sizes. This clearly shows that overhead of data transfer from CPU to MIC is high, which can be mitigated by employing techniques such as allocation a block of memory during the initialization of the framework and reusing the memory blocks for data transfer. Also, performance of sleep jobs was analyzed to assess the ideal performance of Xeon Phi with very short length tasks. It was observed that efficiency in higher 90s could be achieved with tasks lasting as short as 640 microseconds.

5. Conclusions. At the beginning of this work, we described a set of approaches for dealing with MTC over two different many-core architectures, NVIDIA's GPU and Intel Xeon Phi. Also, the main features of both hardware-accelerators were briefly introduced. After that, we analyzed each of the software-hardware approaches individually. In particular, for NVIDIA's GPUs three programming approaches, *Batched*, *Concurrent* and *Dynamic*, were tested on three GPU architectures re-called as TESLA, FERMI and KEPLER. *Batched* and *Concurrent* approaches shown the highest scaling. The overall performance suffers a dramatic fall in performance by increasing the memory requirements and the number of threads, reaching only a good performance over those scenarios with a small demand of memory. Regarding GPUs architecture, we proven that the number of multiprocessor is more relevant that the number of cores to reach a good scaling, at least for our target problem. In this regard, the TESLA architecture (30 multiprocessor and 240 cores) shown a better performance against the KEPLER architecture (13 multiprocessor and 2496 cores). Unlike NVIDIA's GPUs, Intel Xeon Phi turned to be a more appropriate many-core architecture for MTC using an *OpenMP* approach. We obtain a similar trend than obtained in GPUs, the peak performance is reached on very granular tasks, the gain reduces as problem scales up to higher matrix sizes. However, the fall in performance is not so dramatic as in GPUs, and the number of tasks to be efficiently executed is considerable higher than GPUs. Also the peak in performance is much higher, 24 against 9.

Acknowledgments. This research has been supported by EU-FET grant EUNISON 308874, the Basque Excellence Research Center (BERC 2014-2017) program by the Basque Government, the Spanish Ministry of Economy and Competitiveness MINECO: BCAM Severo Ochoa accreditation SEV-2013-0323 and the Project of

the Spanish Ministry of Economy and Competitiveness with reference MTM2013-40824.We also thank the support of the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT) and NVIDIA GPU Research Center program for the provided resources.

REFERENCES

[1] R.J. Barrientos, J.I. Gómez, C. Tenllado, M.Piedro-Matias, and P. Zezula, *Multi-level Clustering on Metric Spaces Using a Multi-GPU Platform*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 216–228.

[2] R.J. Barrientos, J. I. Gómez, C. Tenllado, M. Piedro-Matias, and M. Marin, *knn query processing in metric spaces using gpus*, in Euro-Par 2011 Parallel Processing, Springer Berlin Heidelberg, 2011, pp. 380–392.

[3] S. Chien, *Hand-tuned sgemm on gt200 gpu*, Technical Report, Tsing Hua university, R.O.C. (Taiwan), (2010).

[4] Intel Corp., *Intel many integrated core symmetric communications interface (scif) user guide*, (2012).

[5] NVIDIA Corp., *Nvidia cuda compute unified device architecture-programming guide, version 5*, (2012).

[6] Nvidia Corp., *Dynamic parallelism in cuda - nvidia technical report*, (2013).

[7] T. Cramer, D. Schmidl, M. Klemm, and D. an Mey, *Openmp programming on intel xeon phi coprocessors: An early performance comparison*, in Proceedings of the Many-core Applications Research Community (MARC) Symposium at RWTH Aachen University, November 2012, pp. 38–44.

[8] S. Dimitropoulos, *Gemtc-scif source code repository, https://github.com/sdimitro/scif-modules/tree/ master/scif-sc.*

[9] J. Duato, J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Orti, *Performance of cuda virtualized remote gpus in high performance cluster*, the 40st International Conference on Parallel Processing (ICPP), (2011), pp. 365–374.

[10] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, *Dark silicon and the end of multicore scaling*, in Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA '11, New York, NY, USA, 2011, ACM, pp. 365–376.

[11] D. Geer, *Industry trends: Chip makers turn to multicore processors*, Computer, 38 (2005), pp. 11–13.

[12] J. Gómez-Luna, J.M. González-Linares, J. I. Benavides, and N. Guil, *Performance models for cuda streams on nvidia geforce series*, J. Parallel Distrib. Comput., 72 (2012), pp. 1117–1126.

[13] C. Gregg, J. Dorn, K. Hazelwood, and K. Skadron, *Fine-grained resource sharing for concurrent gpgpu kernels*, In Proceedings of the 4th USENIX Workshop on Hot Topics in Parallelism (HotPar), (2012).

[14] M. A. Guevera, C. Gregg, K. Hazelwood, and K. Skadron, *Enabling task parallelism in the cuda scheduler*, In Proceedings of the Workshop on Programming Models for Emerging Architectures (PMEA), in conjunction with the ACM/IEEE/IFIP International Conference on Parallel Architectures and Compilation Techniques (PACT), (2009).

[15] A. Haidar, T. Dong, P. Luszczek, S. Tomov, and J.J. Dongarra, *Optimization for performance and energy for batched matrix computations on gpus*, in Proceedings of the 8th Workshop on General Purpose Processing using GPUs, GPGPU@PPoPP 2015, San Francisco, CA, USA, February 7, 2015, 2015, pp. 59–69.

[16] A. Haidar, T. Dong, P. Luszczek, S. Tomov, and J. J. Dongarra, *Batched matrix computations on hardware accelerators based on gpus*, IJHPCA, 29 (2015), pp. 193–208.

[17] ———, *Towards batched linear solvers on accelerated hardware platforms*, in Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, San Francisco, CA, USA, February 7-11, 2015, 2015, pp. 261–262.

[18] J. Jeffers and J. Reinders, *Intel Xeon Phi Coprocessor High Performance Programming*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st ed., 2013.

[19] V. J. Jiménez, LL. Vilanova, I. Gelado, G. Fursin M. Gil, and N. Navarro, *Predictive runtime code scheduling for heterogeneous architectures*, the 4th International Conference on High Performance Embedded Architectures and Compilers (HiPEAC), (2009), pp. 19–33.

[20] Jeffrey Johnson, Scott J Krieder, Benjamin Grimmer, Justin M Wozniak, Michael Wilde, and Ioan Raicu, *Understanding the costs of many-task computing workloads on intel xeon phi coprocessors*, 2nd Greater Chicago Area System Research Workshop (GCASR), (2013).

[21] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa, *Timegraph: Gpu scheduling for real-time multi-tasking environments*, In Proceedings of the 2011 USENIX Annual Technical Conference (USENIX ATC'11), (2011).

[22] C. S. Kouzinopoulos, P. D. Michailidis, and K. G. Margaritis, *Multiple string matching on a GPU using cudas*, Scalable Computing: Practice and Experience, 16 (2015).

[23] J. Kreutz, *Dgemm-tiled matrix multiplication with cuda*, Jülich Forchungszentrum, (2013).

[24] S. J. Krieder, J.M. Wozniak, T. Armstrong, M. Wildel, D. S. Katz, B. Grimmer, I.T. Foster, and I. Raicu, *Design and evaluation of the gemtc framework for gpu-enabled many-task computing*, in Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, HPDC '14, New York, NY, USA, 2014, ACM, pp. 153–164.

[25] J. Lima, T. Gautier, N. Maillard, and V. Danjean, *Exploiting concurrent gpu operations for efficient work stealing on multi-gpus*, 24rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), (2012), pp. 75–82.

[26] S. Nakagawa, F. Ino, and K. Hagihara, *A middleware for efficient stream processing in cuda*, Computer Science - Research and Development, 16 (2010), pp. 197–204.

[27] P. Nookala, S. Dimitropoulos, K. Stough, and I. Raicu, *Evaluating the support of mtc applications on intel xeon phi many-core accelerators*, in International Conference on Cluster Computing, CLUSTER '15, 2015.

[28] P. Nookala and K. Stough, *Gemtc-openmp source code repository, https://github.com/pnookala/ mic openmp gemtc.*
[29] nVidia Corp., *Just the facts*, Nvidia. Retrieved, (2013).
[30] B. O'Hallaron, *Using blocking to increase temporal locality, http://csapp.cs.cmu.edu/2e/waside/wasideblocking.pdf*, (2013).
[31] GPGPU. General purpose computation using graphics hardware, *http://www.gpgpu.org*.
[32] D. Schmidl, T. Cramer, S. Wienke, C. Terboven, and M. S. Müller, *Assessing the performance of openmp programs on the intel xeon phi*, in Euro-Par 2013 Parallel Processing, Felix Wolf, Bernd Mohr, and Dieter an Mey, eds., vol. 8097 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 547–558.
[33] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, *Larrabee: A many-core x86 architecture for visual computing*, in ACM SIGGRAPH 2008 Papers, SIGGRAPH '08, New York, NY, USA, 2008, ACM, pp. 18:1–18:15.
[34] M. Själander, M. Martonosi, and S. Kaxiras, *Power-Efficient Computer Architectures: Recent Advances*, Synthesis Lectures on Computer Architecture, Morgan and Claypool Publishers, Dec. 2014.
[35] R. Smith, *Intel's knights landing xeon phi coprocessor detailed*, June 2014.
[36] TOP500.org, *TOP500 List June 2015*.
[37] P. Valero, J.L. Sánchez, D. Cazorla, and E. Arias, *A gpu-based implementation of the mrf algorithm in itk package*, The Journal of Supercomputing, 58 (2011), pp. 403–410.
[38] P. Valero-Lara, *A gpu approach for accelerating 3d deformable registration (dartel) on brain biomedical images*, in Proceedings of the 20th European MPI Users' Group Meeting, EuroMPI '13, New York, NY, USA, 2013, ACM, pp. 187–192.
[39] P. Valero-Lara, *Multi-gpu acceleration of dartel (early detection of alzheimer)*, in Cluster Computing (CLUSTER), 2014 IEEE International Conference on, Sept 2014, pp. 346–354.
[40] P. Valero-Lara and F.L. Pelayo, *Full-overlapped concurrent kernels*, in Architecture of Computing Systems. Proceedings, ARCS 2015-The 28th International Conference on, VDE, 2015, pp. 1–8.
[41] P. Valero-Lara and F. L. Pelayo, *Towards a more efficient use of gpus*, Computational Science and Its Applications (ICCSA) Workshops, (2011), pp. 3–9.
[42] P. Valero-Lara and Fernando L. Pelayo, *Analysis in performance and new model for multiple kernels executions on many-core architectures*, IEEE International Conference on Cognitive Informatics (ICCI*CC), (2013), pp. 189–194.
[43] P. Valero-Lara, A. Pinelli, J. Favier, and M. Piedro-Matias, *Block tridiagonal solvers on heterogeneous architectures*, in Proceedings of the 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, ISPA '12, Washington, DC, USA, 2012, IEEE Computer Society, pp. 609–616.
[44] P. Valero-Lara, A. Pinelli, and M. Prieto-Matias, *Fast finite difference poisson solvers on heterogeneous architectures*, Computer Physics Communications, 185 (2014), pp. 1265 – 1272.
[45] J. van Oosten, *Introduction to cuda 5.0*, nVidia, (2014).
[46] B. van Werkhoven, J. Maassen, F.J. Seinstra, and H.E. Bal, *Performance models for cpu-gpu data transfers*, CCGRID, (2014), pp. 11–20.
[47] S. Verdoolaege, J. C. Juega, A. Cohen, J. I. Gómez, C. Tenllado, and F. Catthoor, *Polyhedral parallel code generation for cuda*, ACM Trans. Archit. Code Optim., 9 (2013), pp. 54:1–54:23.
[48] S. Yamagiva and L. Sousa, *Design and implementation of a stream-based distributed computing platform using graphics processing units*, 4th Int. Conf. Computing Frontiers (CF'07), (2007), pp. 197–204.
[49] K. Zhang and B. Wu, *Task scheduling greedy heuristic for gpu heterogeneous cluster involving the weights of the processor*, International Symposium o Parallel & Distributed Processing Workshops (IPDPSW), (2012), pp. 1817–1827.

Edited by: Dana Petcu
Received: Sept 30, 2016
Accepted: March 9, 2016

DOI 10.12694/scpe.v17i1.1149
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 1, pp. 47–60. http://www.scpe.org

SENSITIVITY STUDY OF INPUT PARAMETERS FOR SEEPAGE FLOW SIMULATIONS USING PARALLEL COMPUTERS*

FRED T. TRACY†, LUCAS A. WALSHIRE‡, AND MAUREEN K. CORCORAN§

Abstract. This paper describes a comprehensive sensitivity study that was performed using high performance parallel computers to understand the importance of input parameters to a transient partially saturated finite element seepage analysis for a levee with separate soil layers of sand, silty sand, and clay. Seepage flow in this paper refers to the type of flow of water that occurred through the failed levees in New Orleans, Louisiana, USA, as a result of Hurricane Katrina. The input parameters tested were saturated hydraulic conductivity, volumetric compressibility, residual moisture content, saturated moisture content, and two van Genuchten unsaturated flow parameters. The output data compiled to show the sensitivity of the input parameters were the simulation times (days) to achieve 25%, 50%, and 75% of the steady-state values of pore pressure at the toe of the levee beneath its blanket, flow rate through the landside flux section, and the level of saturation in the levee. The use of high performance parallel computers enabled the running of thousands of scenarios using different values for the input variables. A sensitivity investigation of this magnitude has not been previously performed.

The results of this investigation indicated that the more sensitive soil parameters were the saturated hydraulic conductivity and the volumetric compressibility. The unsaturated van Genuchten parameters of the landside blanket had a larger than anticipated impact on the duration of time to achieve steady state. This practical example is an excellent success story for high performance computing in that running a given simulation for a couple of hours on thousands of processors in parallel replaced over a year's work using a PC.

Key words: high performance computing, sensitivity analysis, finite element method, seepage flow modeling

AMS subject classifications. 35J66, 65Y05, 76S05

1. Introduction.

Historically, the majority of practicing engineers have used two-dimensional (2-D) steady state analyses to design and analyze levees. Finite element programs such as SEEP2D [1,2] in the Groundwater Modeling System (GMS) [3,4] and SEEP/W [5] with excellent graphical user interfaces have greatly aided these design and analysis processes. However, using only steady-state analyses leads to the most conservative and therefore, the most expensive design. With the ability to now do a transient seepage analysis on a computer, key questions are when should a design be based on a transient analysis instead of a steady-state analysis, what soil parameters are important in the transient analysis, and how reliable are the transient simulation results.

One traditional analytical tool used in determining the relative importance of the various input parameters is a sensitivity study. Some sensitivity analyses have been historically too compute-intensive to perform. However, with the availability of high performance, parallel computing, thousands of scenarios can be run at once, allowing for the testing of a greater number of input parameters than was previously possible.

The purpose of this research was to perform a comprehensive sensitivity study of input soil parameters needed for a transient finite element seepage analysis as measured by the response of key output variables. A generic levee common to the southeastern United States was selected for the analysis. To perform the analysis, a parallel program was created such that from a set of data describing the levee cross section, a finite element mesh was generated, initial and boundary conditions were applied, both steady-state and transient seepage analyses were performed, and key data were stored for future analysis. This was all done in the context of parallel computing where thousands of scenarios could be computed simultaneously. A feature of the groundwater modeling program used in the study is that the time needed to achieve a certain percentage of the steady-state value of a given output variable can be computed and stored for future analysis. This is possible because a steady-state solution is computed before the transient solution is performed. This modified parallel groundwater modeling program made this research possible. Because of all the obstacles, very few sensitivity studies of this magnitude have been completed with no known studies performed on this magnitude for transient seepage.

*This work was supported in part by a grant of computer time from the Department of Defense High Performance Computing Modernization Program (HPCMP).

†Information Technology Laboratory, Engineer Research and Development Center (ERDC), Vicksburg, MS, USA.
‡Geotechnical and Structures Laboratory (GSL), ERDC, Vicksburg, MS, USA.
§GSL, ERDC, Vicksburg, MS, USA.

2. Measuring sensitivity. There are many ways to measure sensitivity of an output variable to an input parameter, and an excellent description of them is given in [6]. This information will not be repeated here but rather some key methods of doing sensitivity studies for numerically intensive applications are highlighted. The method of slopes and the method of ranges were used in this investigation.

2.1. Method of slopes. Changing only one input parameter while holding all the others constant and measuring the output to obtain the slope of the output variable versus the varied input parameter curve is the simplest way of doing a sensitivity analysis [6]. The slope, m, for a given output value (Y) versus input parameter (X) curve is simply the partial derivative,

$$m = \frac{\partial Y}{\partial X} \tag{2.1}$$

This curve can be determined by using different values of X, running the transient seepage program, and then recording the resulting Y output values. Because this slope was determined numerically, it was approximated as follows:

$$m \approx \frac{\Delta Y}{\Delta X} \tag{2.2}$$

where ΔX is a small increment of input parameter, and ΔY is the resulting small change in the output value.

Only one type of output parameter was considered in this study, but several input parameters varying over several orders of magnitude were considered. Therefore, a sensitivity coefficient, s_m, fashioned after [7] was implemented. Defining $Y(X)$ as the output, Y, as a function of the input parameter, X, s_m is

$$s_m = \frac{Y\left(X + \frac{p}{100}X\right) - Y(X)}{p} \tag{2.3}$$

where p is a percentage of a given input parameter. Because of the highly nonlinear nature of the governing partial differential equation where repeated Picard or Newton iterations [8] are needed, a value of $p = 10$ was selected. The sensitivity coefficient, s_m, can be computed for different values of the input parameters to obtain an overall view of the nonlinear behavior.

2.2. Method of ranges. The method of ranges is useful when the output can be sampled over the entire range of the respective input parameters. This method compares how much the output variable changes when the different input parameters are varied. A sensitivity coefficient based on ranges is now defined, which is an extension of previous work [9]. The case where there are $i = 1, 2, 3, \ldots, M$ scenarios of the $j = 1, 2, 3, \ldots, N$ input parameters was considered. Defining K_j as the number of different values of the j^{th} input parameter, M in this research becomes

$$M = K_1 K_2 K_3 \cdots K_N = \prod_{j=1}^{N} K_j \tag{2.4}$$

Eq. 2.4 is rearranged to isolate the j^{th} input parameter as follows:

$$M = \left(\prod_{i=1, i \neq j}^{N} K_i \right) K_j = M_j K_j \tag{2.5}$$

where M_j is the number of combinations of all the input parameters except the j^{th} one. For each of these $m = 1, 2, 3, \ldots, M_j$ combinations, the results of varying the j^{th} input parameter over its range while holding the other input parameters constant are used to define a sensitivity coefficient based on the range of the output values. The sensitivity coefficient for this m^{th} scenario while changing only the j^{th} input parameter is defined as follows:

$$s_{rm}^{j} = \frac{(Y_{max})_{mj} - (Y_{min})_{mj}}{Y_{max} - Y_{min}} \tag{2.6}$$

where

$(Y_{min})_{mj}$ = minimum value of output, Y, when varying the j^{th} input parameter and holding the others constant for the m^{th} combination

$(Y_{max})_{mj}$ = maximum value of output, Y, when varying the j^{th} input parameter and holding the others constant for the m^{th} combination

Y_{min} = overall minimum value of the output variable, Y

Y_{max} = overall maximum value of the output variable, Y

The overall sensitivity coefficient for the j^{th} input parameter is computed by taking the maximum value of the s^j_{rm} values over the M_j combinations. That is,

$$s^j_r = \max_{m=1}^{M_j} \left(s^j_{rm}\right) \tag{2.7}$$

2.3. Statistical methods. There are many statistical methods for calculating sensitivity. One of the simplest methods that uses simple correlation coefficients was derived from Monte Carlo simulations [10]. Here, Pearson's product moment correlation coefficient [11] was employed. The case that is now considered is where M output values are computed from M values of an input parameter, X. Pearson's product moment correlation coefficient is given by

$$r = \frac{\sum_{i=1}^{M} \left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{\sqrt{\left[\sum_{i=1}^{M} \left(X_i - \bar{X}\right)^2\right]\left[\sum_{i=1}^{M} \left(Y_i - \bar{Y}\right)^2\right]}} \tag{2.8}$$

$$\bar{X} = \frac{1}{M}\sum_{i=1}^{M} X_i \tag{2.9}$$

$$\bar{Y} = \frac{1}{M}\sum_{i=1}^{M} Y_i \tag{2.10}$$

The larger the r value, the stronger the reaction of the output to the input [12]. This works best when the relationship between input parameter and output variable is linear. However, the application presented in this paper is highly nonlinear.

3. Levee. Eqs. 2.3 and 2.7 were applied to a generic levee. The geometry, material properties, initial conditions, boundary conditions, and hydrograph of the river are provided in the following subsections. Detail is provided for those practicing geotechnical engineers who find such information important.

3.1. Generic levee cross section. A 2-D finite element model of a generic levee cross section representative of the southeastern United States is shown in Fig. 3.1 (this section is further described in [13]). This model was used to perform computations and to do the sensitivity study described in this paper.

3.2. Riverside and landside water elevation. The initial river elevation was set to -5 ft. An initial water elevation below the ground surface is the most interesting case because the water must rise through two of the three soil layers as it reaches its maximum height. The river elevation advanced at 2 ft/day until it reached 17.5 ft where it was held constant indefinitely. This is illustrated in the hydrograph shown in Fig. 3.2 where the time for the hydrograph is plotted for only 30 days but extends indefinitely.

4. Input parameters selected for the sensitivity study. Table 4.1 gives the input parameters and respective descriptions selected for variation in the sensitivity study. The first case considered was a homogeneous levee and foundation section where the levee and foundation are all assigned one set of values. In the next case, different material properties were assigned to three layers: levee, blanket, and aquifer. With reference to Fig. 3.1, the levee material would be a lean clay, the blanket would be the silty sand, and the aquifer would be the clean sand. The geometry of the cross section remained constant throughout the study.

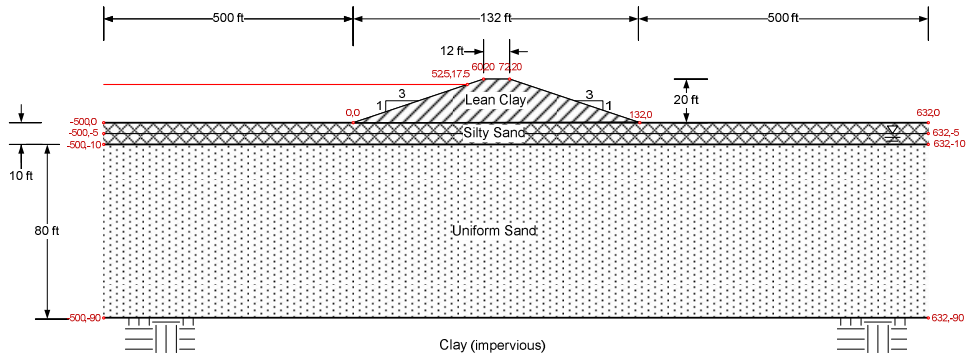

Fig. 3.1: Generic levee cross section showing three separate layers of sand, silty sand, and clay.

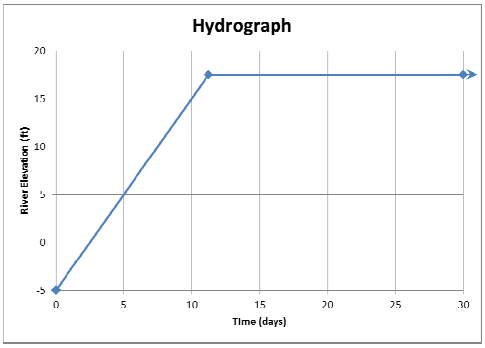

Fig. 3.2: Hydrograph that starts at -5 ft and then goes up 2 ft/day until 17.5 ft is reached and then remains constant indefinitely.

Table 4.1: Parameters used in the levee system study.

Name	Symbol
Saturated hydraulic conductivity (ft/day)	k_s
Residual volumetric water content (unitless)	θ_r
Saturated volumetric water content (unitless)	θ_s
First van Genuchten parameter (1/ft)	α
Second van Genuchten parameter (unitless)	n
Volumetric compressibility(1/psf)	m_v

5. Output variables selected for the sensitivity study. The output variables selected for this research are the simulation time in days to achieve 25%, 50%, and 75% of the steady-state values of the following three quantities:

- The pore pressure (psf) at the toe of the levee beneath the blanket (coordinates 132, -10 in Fig. 3.1).
- The flow rate per unit length (ft^3/day/ft) of water leaving the flux section, (132, 0) to (632, 0) to (632, -90) in Fig. 3.1.
- Levee saturation coefficient.

The levee saturation coefficient is defined as

$$S_L = \frac{\int\int_A \theta\,(x,y,t)\,dxdy - \int\int_A \theta\,(x,y,0)\,dxdy}{\int\int_A \theta\,(x,y,\infty)\,dxdy - \int\int_A \theta\,(x,y,0)\,dxdy} \tag{5.1}$$

Table 6.1: Values of input parameters used in the homogeneous case.

Parameter	Value
θ_r	0.034
θ_s	0.46
α	0.488 ft^{-1}
n	1.37
m_v	1.0×10^{-5} psf^{-1}

where

A = area of the levee
$\theta(x, y, t)$ = volumetric moisture content of an (x, y) point in the levee at time, t
$\theta(x, y, 0)$ = volumetric moisture content of an (x, y) point in the levee at $t = 0$
$\theta(x, y, \infty)$ = volumetric moisture content of an (x, y) point in the levee at steady state

It is important to note that $0 \leq S_L \leq 1$ since $S_L = 0$ at initial conditions, and $S_L = 1$ at steady state.

6. Homogeneous case. In the first case, the aquifer, confining blanket, and levee materials were assigned identical soil properties.

6.1. Varying only the saturated hydraulic conductivity. The approach used in this research was to start with the simplest case and build on that effort with more complicated scenarios. In this initial run, only saturated hydraulic conductivity was varied with values, 0.001, 0.01, 0.1, 1.0, 10.0, and 100.0 ft/day (3.528×10^{-7}, 3.528×10^{-6}, 3.528×10^{-5}, 3.528×10^{-4}, 3.528×10^{-3}, and 3.528×10^{-2} cm/sec, respectively). The hydraulic conductivity values correspond to the following material types: clay, silt, silty sand, fine sand, and coarse sand, respectively [14]. Table 6.1 gives the values of the other input parameters used for these runs. Figs. 6.1, 6.2, and 6.3 show the times to achieve 25, 50, and 75% of the respective steady-state values of the pore pressure at the toe of the levee beneath the blanket, the flow rate through the flux section, and the levee saturation coefficient, respectively. The computer runs were terminated after 1000 days, so in the cases where more than 1000 days are needed to achieve the given percentages of steady state, no values are plotted. Observations are as follows:

- The saturated hydraulic conductivity has a significant impact on results.
- For hydraulic conductivity values of 0.001 ft/day and 0.01 ft/day, none of the given percentages of the steady-state values of the pore pressure beneath the blanket at the toe and the flow rate through the flux section can develop within 1000 days.
- The same basic trend occurs in each plot with only the separation among curves varying.

6.2. Varying the saturated hydraulic conductivity and volumetric compressibility. Once it was found that the magnitude of the hydraulic conductivity value had a significant impact on the results of a transient analysis, other parameters were varied in conjunction with the hydraulic conductivity. Here, both k_s and m_v were varied and results collected for the same times to percentage of steady state as before. m_v values of 1.0×10^{-3}, 1.0×10^{-5}, and 1.0×10^{-7} psf^{-1} were used with the same k_s values as before. The results are given in Figs. 6.4, 6.5, and 6.6. It is important to note that all combinations of k_s and m_v are not what a practicing engineer may choose. However, to maintain continuity of trends, they are kept in the plots. The method of ranges (Eq. 2.7) was applied to these results, and the result of this computation is provided in Table 6.2. For all computations in this paper where Eq. 2.7 is used, $Y_{max} = 1000$. Observations from these results are:

- m_v has a significant impact on results but slightly less influence than k_s.
- Pore pressure, flow rate, and levee saturation coefficient have the same basic trend.
- The greater the hydraulic conductivity, the less time it takes to achieve steady state.
- The greater the volumetric compressibility, the more time it takes to achieve steady state.

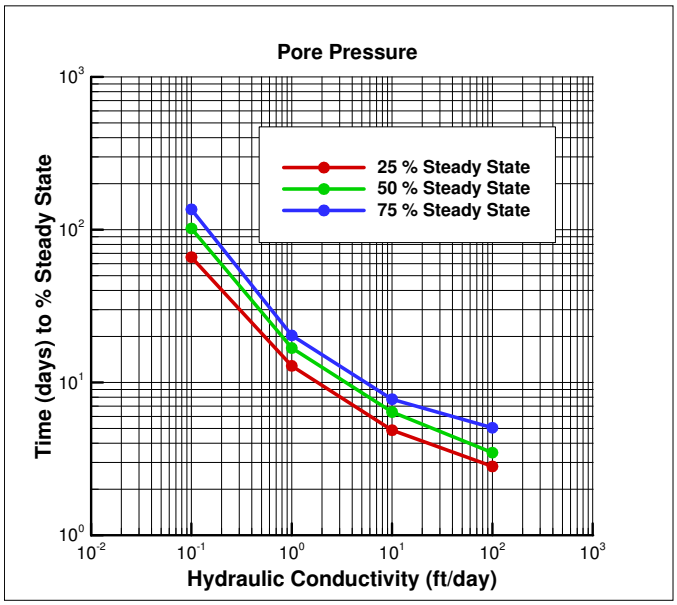

Fig. 6.1: Plot of times to percentage of steady state for pore pressure.

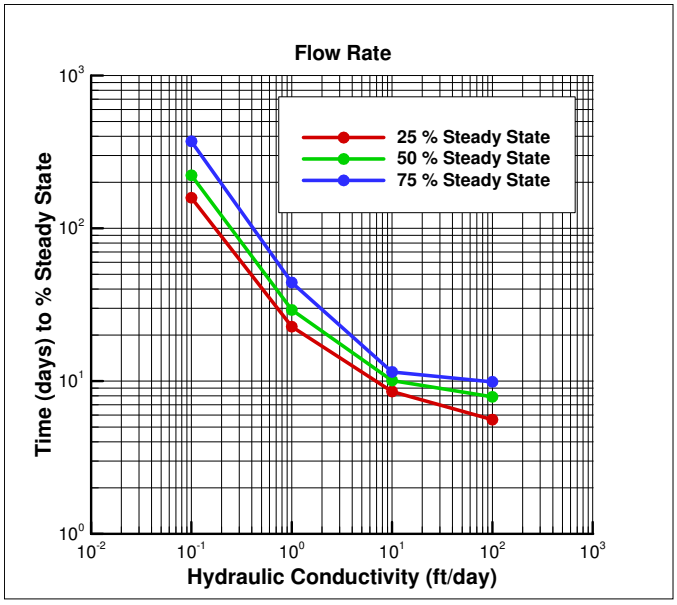

Fig. 6.2: Plot of times to percentage of steady state for flow rate through the flux section.

6.3. Varying all input parameters except volumetric compressibility. The next analysis was performed with all input parameters being varied except m_v, and s_r was computed as before using Eq. 2.7. The values of the parameters used in this analysis are given in Table 6.3. There are two reasons why only $m_v = 0.00001$ psf^{-1} was used, and they are as follows: (1) When m_v was set to 0.001, 0.00001, and 0.0000001 psf^{-1} as before, all the s_r values were 0.99. Thus, no separation of importance was possible. (2) $m_v = 0.00001$ psf^{-1} is often selected by practicing engineers as a default option. Observations are as follows:

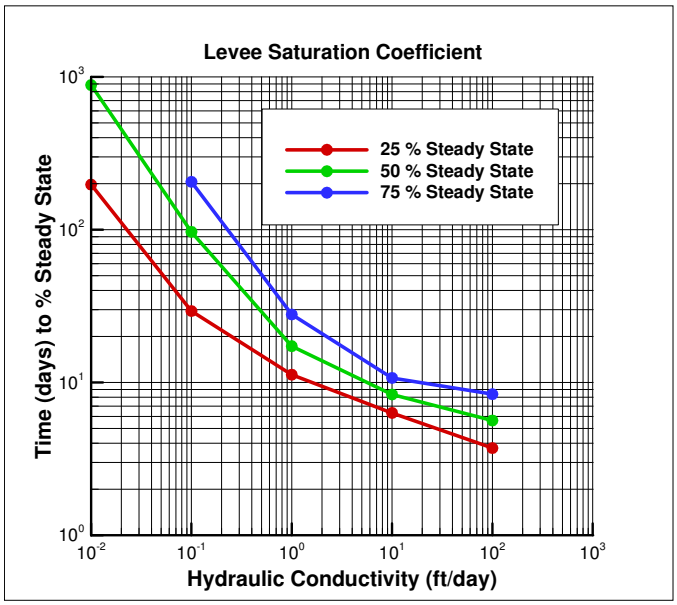

Fig. 6.3: Plot of times to percentage of steady state for levee saturation coefficient.

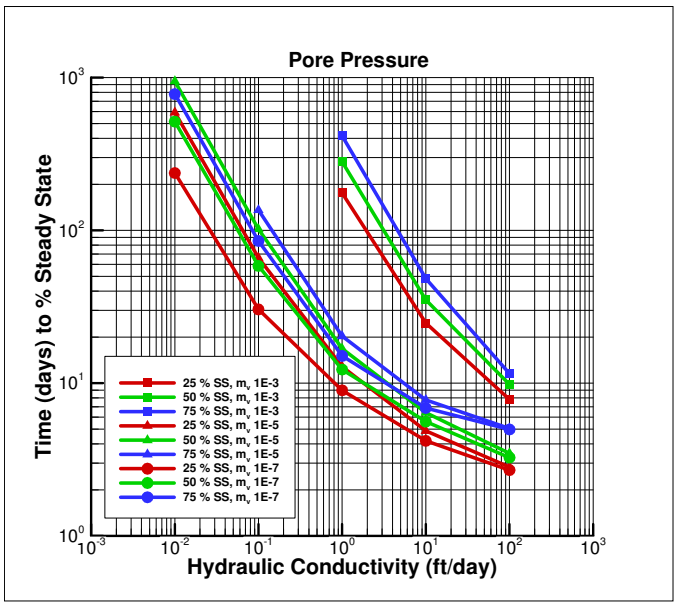

Fig. 6.4: Plot of times to percentage of steady state for pore pressure for k_s and m_v combinations.

- The saturated hydraulic conductivity was shown to be the most sensitive input variable.
- The other input parameters of θ_r, θ_s, α, and n all have significant influence on results.
- α and n are the most sensitive of the unsaturated flow parameters.

7. Heterogeneous Case. A final experiment involved a more realistic scenario in which the three different layers of levee, blanket, and aquifer in Fig. 3.1 were each assigned different material properties. Table 7.1 gives

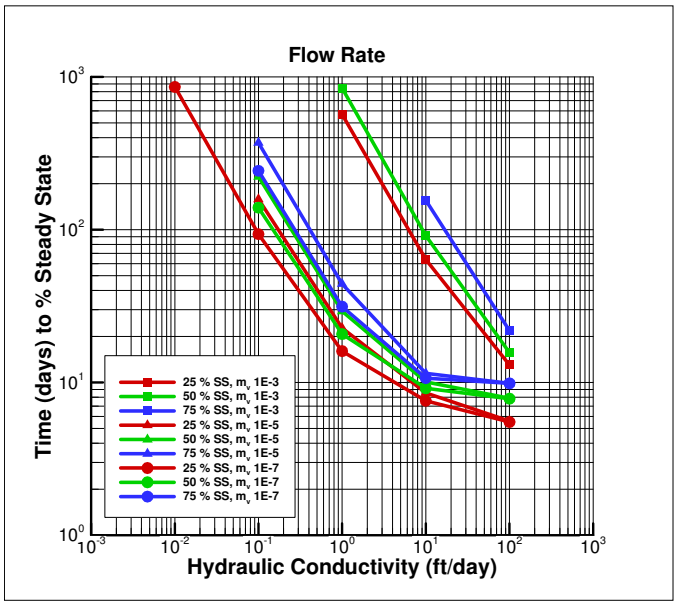

Fig. 6.5: Plot of times to percentage of steady state for flow rate through the flux section for k_s and m_v combinations.

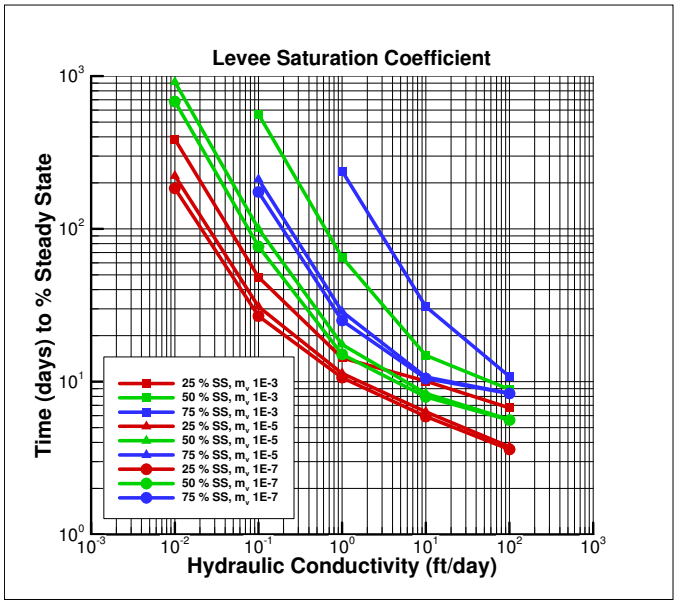

Fig. 6.6: Plot of times to percentage of steady state for levee saturation coefficient for k_s and m_v combinations.

the variation of soil types in the three layers, and Table 7.2 gives the values of soil properties used. The 14 allowable combinations of saturated hydraulic conductivity for the levee and blanket are given in Table 7.3. Saturated hydraulic conductivity values considered for the aquifer are $k_{s,aquifer} = 10$ and 100 ft/day.

Because k_s, m_v, θ_r, θ_s, α, and n are considered for sensitivity for both the levee and blanket, over one million scenarios would be needed to use the sensitivity method of ranges. Thus, the sensitivity method of

Table 6.2: Sensitivity coefficient, s_r, values for k_s and m_v for the times to different percentages of steady state for pore pressure, flow rate, and levee saturation coefficient.

s_r for pore pressure			
	Percent of steady state		
Input parameter	25	50	75
k_s	0.9973	0.9967	0.9950
m_v	0.9696	0.9415	0.9148
s_r for flow rate			
	Percent of steady state		
Input parameter	25	50	75
k_s	0.9945	0.9922	0.9901
m_v	0.9065	0.8605	0.9686
s_r for levee saturation coefficient			
	Percent of steady state		
Input parameter	25	50	75
k_s	0.9964	0.9944	0.9916
m_v	0.2010	0.4863	0.8255

Table 6.3: Range of input parameters.

Material Property	Range of Values Investigated
k_s (ft/day)	0.001, 0.01, 0.1, 1, 10, 100
m_v (1/psf)	1.0×10^{-5}
θ_r	0.05, 0.1, 0.15
θ_s	0.40, 0.45, 0.5
α (1/ft)	0.2, 0.4, 0.6
n	1.25, 1.75, 2.25

Table 6.4: Sensitivity coefficient, s_r, values for k_s, θ_r, θ_s, α, and n for the times to different percentages of steady state for pore pressure, flow rate, and levee saturation coefficient.

s_r for pore pressure			
	Percent of steady state		
Input parameter	25	50	75
k_s	0.9969	0.9969	0.9951
θ_r	0.1459	0.1486	0.1287
θ_s	0.1459	0.1486	0.1287
α	0.3154	0.3048	0.2591
n	0.4503	0.4046	0.3186
s_r for flow rate			
	Percent of steady state		
Input parameter	25	50	75
k_s	0.9945	0.9922	0.9901
θ_r	0.0961	0.0672	0.1146
θ_s	0.0961	0.0672	0.1146
α	0.1874	0.1637	0.3257
n	0.1874	0.1767	0.3072
s_r for levee saturation coefficient			
	Percent of steady state		
Input parameter	25	50	75
k_s	0.9968	0.9957	0.9926
θ_r	0.2585	0.2206	0.1620
θ_s	0.2585	0.2206	0.1620
α	0.4692	0.5659	0.2966
n	0.2263	0.3417	0.2905

Table 7.1: Description of soils in levee layers.

Layer	Material variation
Levee	Clay to gravelly sand
Blanket	Clay to sandy silt
Aquifer	Sand to gravelly sand

Table 7.2: Soil property data for selected soils.

Soil classification	k_s (ft/day)	θ_r	θ_s	α (1/ft)	n	m_v (1/psf)
Clay	0.001	0.05	0.50	0.076, 0.137	1.05	5×10^{-5}
Lean clay	0.01	0.045	0.45	0.152, 0.228	1.1, 1.15	1×10^{-5}
Silt	0.1	0.04	0.40	0.243, 0.610	1.2, 1.35	5×10^{-6}
Silty sand/sandy silt	1	0.035	0.35	0.762, 1.829	1.4, 1.6	1×10^{-6}
Sand	10	0.03	0.30	2.286, 4.420	1.9, 2.7	5×10^{-7}
Gravelly sand	100	Not used				1×10^{-7}

Table 7.3: Hydraulic conductivity (ft/day) combinations for levee, blanket, and aquifer layers. The check marks indicate the allowable combinations.

$k_{s,blanket}$	$k_{s,levee}$				
	0.001	0.01	0.1	1	10
0.001	√	√	√		
0.01	√	√	√	√	
0.1		√	√	√	√
1			√	√	√

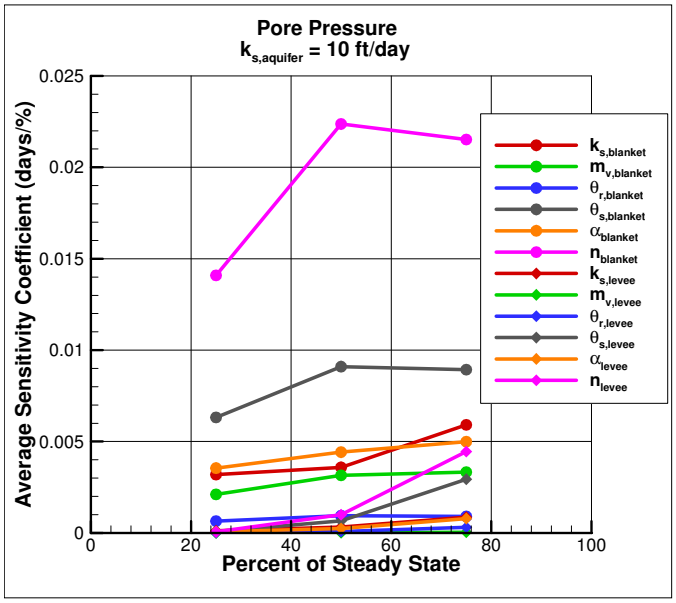

Fig. 7.1: Plot of average s_m for pore pressure for $k_{s,aquifer} = 10$ ft/day.

slopes given by Eq. 2.3 was used. Another disadvantage of the method of ranges approach used thus far is that some combinations of the input parameters would not be used by practicing engineers. Thus, the material property data were restricted to the allowable combinations given in Table 7.3 and values given in Table 7.2.

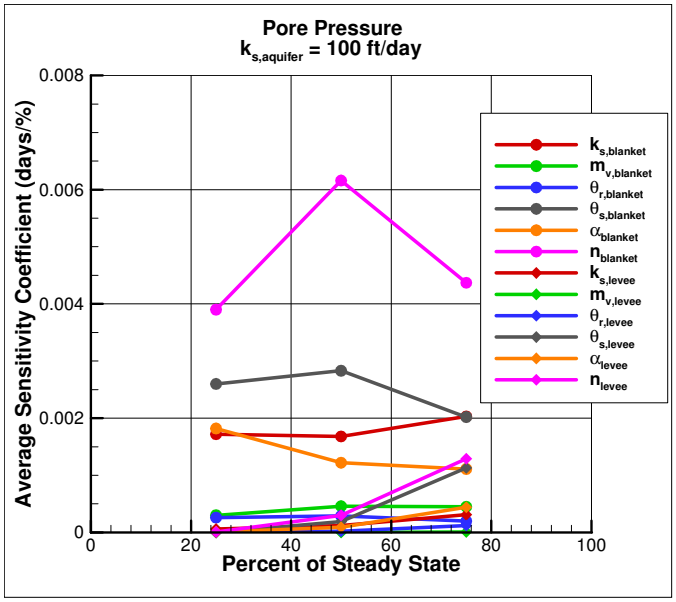

Fig. 7.2: Plot of average s_m for pore pressure for $k_{s,aquifer} = 100$ ft/day.

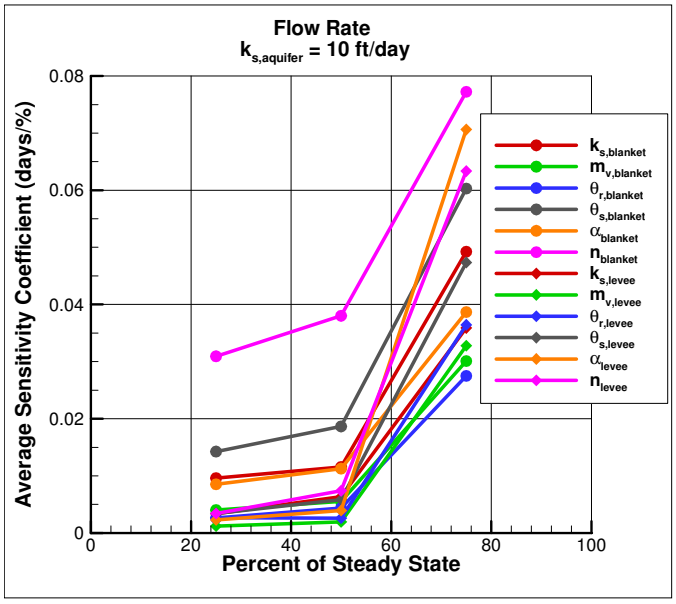

Fig. 7.3: Plot of average s_m for flow rate for $k_{s,aquifer} = 10$ ft/day.

When Eq. 2.3 was used in the sensitivity analysis, the allowable combinations described above were all considered. For a given scenario, the six soil parameters, k_s, m_v, θ_r, θ_s, α, and n, for both the levee and blanket (12 parameters) were varied slightly in succession, leaving all the other input parameters constant when a given parameter was varied. This required that for each valid set of soil values or scenario, 13 simulations were needed (the original run plus 12 variations). Two separate parallel high performance computing runs were made, one

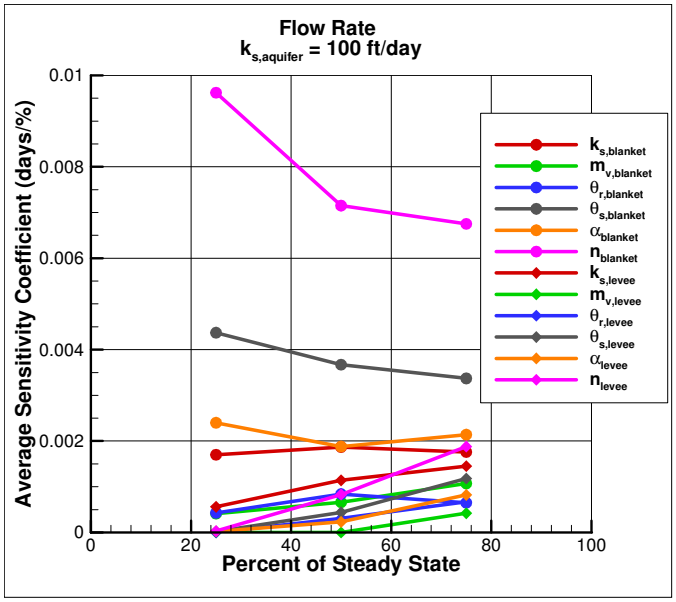

Fig. 7.4: Plot of average s_m for flow rate for $k_{s,aquifer} = 100$ ft/day.

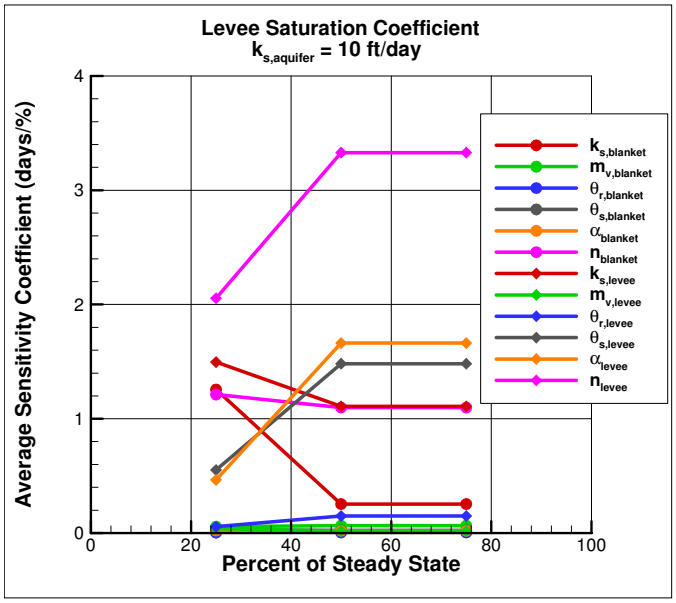

Fig. 7.5: Plot of average s_m for levee saturation coefficient for $k_{s,aquifer} = 10$ ft/day.

with $k_{s,aquifer} = 10$ ft/day, and another with $k_{s,aquifer} = 100$ ft/day. Thus, two sets of 2444 simulations each were needed. All 2444 simulations for a given $k_{s,aquifer}$ value were accomplished with a parallel MPI job using 2444 processes and taking approximately 1.5 hours.

The sensitivity coefficient, s_m, from Eq. 2.3 was computed for all the valid combinations for time to achieve 25, 50, and 75% of the respective steady-state value of pore pressure, flow rate, and levee saturation coefficient,

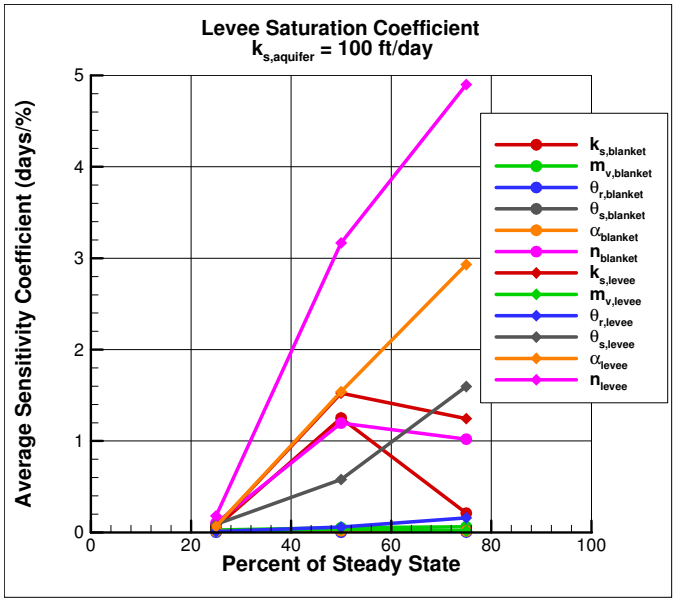

Fig. 7.6: Plot of average s_m for levee saturation coefficient for $k_{s,aquifer} = 100$ ft/day.

and the average value of s_m over these valid combinations for each input parameter was tabulated. Both runs for $k_{s,aquifer} = 10$ and 100 ft/day were done, and the results are given in Figs. 7.1, 7.2, 7.3, 7.4, 7.5, and 7.6. Observations are as follows:

- Pore pressure at the toe of the levee and flow rate are 2-3 orders of magnitude less sensitive to the input parameters than is the levee saturation coefficient.
- The input parameters for both the blanket and the levee are important.
- The second van Genuchten unsaturated flow parameter, n, is the most sensitive input variable for most of the scenarios analyzed.
- The sensitivity coefficients for pore pressure and flow rate are approximately an order of magnitude less when $k_{s,aquifer} = 100$ ft/day as compared to $k_{s,aquifer} = 10$ ft/day.

8. Conclusions. This sensitivity study considered a large number of scenarios and was made feasible only through the use of high performance, parallel computers. This is especially true because the scenarios can be run independently without any communication among MPI processes. The results of the study indicate the need to obtain all the input parameters (saturated hydraulic conductivity, volumetric compressibility, residual moisture content, saturated moisture content, and the two van Genuchten parameters) as accurately as possible since the output quantities of interest show a significant sensitivity to each parameter for at least some of the scenarios analyzed. The volumetric compressibility had a dominant effect on output values in the homogeneous case but ranked near the bottom of the list in the case where all three layers had different material properties. The second van Genuchten parameter ranked low in sensitivity in the homogeneous case for pore pressure and flow rate but ranked very high when all three layers had different material properties. When all three layers had different material properties, the van Genuchten unsaturated flow parameters for the blanket often dominated for pore pressure and flow rate, whereas the unsaturated flow parameters for the levee dominated for levee saturation coefficient.

Many of the sensitivity analyses assume that the relationship between input parameters and output variables is linear. The application of unsaturated transient flow is highly nonlinear, so these analyses are of limited value. The method of slopes is suitable for a nonlinear application, but it is very compute-intensive. However, with high performance, parallel computing, this obstacle can be easily overcome since the computation involved for different scenarios is an embarrassingly parallel task. A disadvantage of the method of slopes is that it gives

a local value that varies greatly over the different scenarios. A simple average of the results over the different scenarios as presented here could be improved with a more sophisticated analysis. This is a topic of future research.

The method of ranges is acceptable for the application presented in this paper, but it has the disadvantage that many scenarios are needed to explore a full range of input parameters and thus to obtain comprehensive results. However, the number of combinations of input parameters can be reduced by limiting the scenarios to only those of interest to practicing engineers.

REFERENCES

[1] F. T. Tracy, *User's guide for a plane and axisymmetric finite element program for steady state seepage problems*, Instruction Report IR K-83-4, U.S. Army Engineer Research and Development Center, Vicksburg, MS, 1983.
[2] N. L. Jones, *SEEP2D primer*, GMS documentation, Environmental Modeling Research Laboratory, Brigham Young University, Provo, Utah, 1999.
[3] GMS, *Groundwater Modeling System, commercial version*, www.aquaveo.com/GMS, 2016.
[4] GMS, *Groundwater Modeling System, government version*, http://chl.erdc.usace.army.mil/gms, 2016.
[5] Geo-Slope, *Seepage modeling with SEEP/W*, Calgary, Alberta, Canada, 2012.
[6] D. M. Hamby, *A review of techniques for parameter sensitivity analysis of environmental models*, Environmental Monitoring and Assessment, 32 (1994), pp. 135-154.
[7] R. W. Atherton, R. B. Schainker, and E. R. Ducot, *On the statistical sensitivity analysis of models for chemical kinetics*, AIChE, 21 (1975), pp. 441-448.
[8] S. Mehl, *Use of Picard and Newton iteration for solving nonlinear ground water flow equations*, Ground Water, 44 (2006), pp. 583-594.
[9] F. O. Hoffman and R. H. Gardner, *Evaluation of uncertainties in environmental radiological assessment models*, In: J. E. Till and H. R. Meyer (eds.), Radiological Assessments: a Textbook on Environmental Dose Assessment, Report No. NUREG/CR-3332, U.S. Nuclear Regulatory Commission, Washington, DC, 1983.
[10] R. H. Gardner, R. V. O'Neill, J. B. Mankin, and J. H. Carney, *A comparison of sensitivity analysis and error analysis based on a stream ecosystem model*, Ecol. Modelling, 12 (1981), pp. 173-190.
[11] W. J. Conover, *Practical Nonparametric Statistics, 2nd edition*, Oxford University Press, John Wiley & Sons, New York, 1980.
[12] International Atomic Energy Agency (IAEA), *Evaluating the reliability of predictions made using environmental transfer models*, Safety Series No. 100, Report No. STI/PUB/835, Vienna, Austria, pp. 1-106, 1989.
[13] F. T. Tracy, T. L. Brandon, and M. K. Corcoran, *Transient seepage analyses in levee engineering practice*, In review, U.S. Army Engineer Research and Development Center, Vicksburg, MS, 2016.
[14] K. Terzaghi, B. Peck, and G. Mesr, *Soil Mechanics in Engineering Practice*, John Wiley & Sons, New York, 1996.

Edited by: Dana Petcu
Received: Dec 18, 2015
Accepted: Mar 8, 2016

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2_ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.