Evaluating 3-D Stencil codes on Intel Xeon Phi: Limitations and Trade-offs

Mario Hernández[1] Juan M. Cebrián[2] José M. Cecilia[3] José M. García[4]

Abstract—**Accelerators like Intel Xeon Phi aim to fulfill the computational requirements of modern applications. A particular interest to us are those applications that are based on *Stencil Computations*. Stencils are finite-difference algorithms used in many scientific and engineering applications for solving large-scale and high-dimension partial differential equations. Programmability on massively parallel architectures of such kernels is still a challenge for inexperienced developers.**

This paper evaluates three stencil-based kernels that are widely applied to simulate heat and acoustic diffusion as well as isotropic seismic wave equation. We focus on key issues that should be considered in order to achieve optimal performance on the Xeon Phi architecture. Among them, we highlight trade-offs between scalability and affinity, blocking and effect of grid shape. Our experimental results yield small performance gains using scatter affinity, showing that the blocking size strongly affects the kernel performance. In addition it reveales that grid shape has minimal impact in performance as long as the best block size is selected.

Keywords— **3-D stencil codes, Xeon Phi evaluation, scalability, affinity, blocking size, shape.**

I. Introduction

In the last decade, there has been a technological shift for both hardware and software towards massively parallel architectures (accelerators). Intel Many Integrated Core (MIC) [1] [2] and Graphics Processing Units (GPUs) [3] clearly show the potential of these architectures, especially in terms of performance and energy efficiency. The most powerful supercomputers in the world are currently based on accelerators [4]. Concurrently, there has been a quick evolution on programming models for co-processors and GPUs. However, porting applications to these systems is still not a straight-forward task. In order to maximize performance and energy efficiency of their systems, software developers need to use the latest breakthroughs in both high performance computing and the specific field of interest (e.g., image processing, modeling of acoustic or heat diffusion, etc). This vertical approach enables remarkable advances in computer-driven scientific simulations (the so-called hardware-software co-design).

As stated in [5], many applications are developed using an algorithmic method that captures a pattern of computation and communication (so-called *Dwarf*). These patterns are repeated in different applications and thus, hardware-software solutions can be extrapolated to many scientific areas. Applications can be either: (1) computationally bounded, if they perform many operations over the loaded data, or (2) memory bounded, if they perform a few simple computations over the loaded data. The latter is a serious performance limiting factor, since the computation of a single output value may require loading almost as many values from memory as operations are performed.

Stencil codes comprise a family of iterative kernels that operate over an N-dimensional data structure that changes over time, given a fixed computational pattern (stencil). The design of efficient stencil codes has proven to be a challenging task, and has been widely studied by both academia and industry.

In this paper, we evaluate the Xeon Phi architecture using three 3-D stencil kernels computing double precision floating point values (8 bytes). Our kernels are selected from different fields of research: 1) 3-D heat diffusion stencil (11-point), 2) 3-D acoustic diffusion stencil (7-point) and 3) 3-D isotropic seismic wave stencil (25-point). We describe the vectorization and parallelization process from a simple sequential version. Then we show how to achieve the maximum performance by tuning some specific inherent parameters relative to both the kernel and the hardware architecture. The main lessons learned are the following:

- Scalability: As our 3-D stencil kernels are mostly memory-bound, we first evaluate the kernel's scalability by varying the number of threads in the execution. Our empirical results reveal that the minimum number of threads per node that maximizes performance is 122 (i.e. two-threads per core). However, the memory hierarchy is unable to feed the amount of data required by those threads.
- Affinity: We test three different affinity modes available for OpenMP on Xeon Phi. Our evaluation shows that the *compact* mode is the worst in terms of performance, while the *scatter* mode gives the same or better results that the *balanced* mode.
- Blocking size: The Intel Xeon Phi architecture requires a proper usage of caches to achieve good performance when running stencil applications. We have implemented a blocking technique carefully tuned experimentally to achieve the best performance (i.e., adjusting blocking size). After an analysis of different block sizes, we empirically found that the blocking technique works best on the Y and Z dimensions, but not on X. Specifically, a block size of *by=2* and *bz=40* offers good results for the *Acoustic* and *Heat* kernels, whereas a block size of *by=2* and *bz=8* is the best option Seismic.
- Input data layout shape: Finally, we have evaluated the influence of the input data shape on the performance of these kernels. We analyze four different shapes: the cuboid one and three rectangular cubes. We have noticed that, after adjusting the block sizes to the best ones for each of these shapes, the impact

[1]Department of Computer Engineering, University of Murcia, 30100, Murcia. Academic Unit of Engineering, Autonomous University of Guerrero, Chilpancingo, México, e-mail: mario.hernandez4@um.es

[2]Department of Computer Engineering, University of Murcia, 30100, Murcia, e-mail: jcebrian@ditec.um.es

[3]Computer Science Department, Universidad Católica San Antonio de Murcia, Spain, e-mail: jmcecilia@ucam.edu

[4]Department of Computer Engineering, University of Murcia, 30100, Murcia, e-mail: jmgarcia@ditec.um.es

on performance is around 20% for the seismic and heat codes, and 10% for the acoustic code.

The paper is structured as follows. The next section gives some fundamentals and related work about stencil computations and the MIC architecture. The main parameters considered for 3-D stencil codes in our evaluation are introduced in Section III. Section IV shows our evaluation results. Finally, we summarize our conclusions and future work in Section V.

II. Background and Related Work

This section provides a brief background on 3-D Stencil computations, the Intel Xeon Phi architecture, and related work regarding several stencil optimization for the Xeon Phi.

A. 3-D Stencil Computations

Stencil codes [6], [7], [8], [9] are a type of iterative kernels which update data elements according to some fixed predetermined set or pattern. The stencil can be used to compute the value of various elements in an array at a given time-step based on its neighbors values computed from previous time-steps (including the element itself). Stencils are the base of finite-difference algorithms used for solving large-scale and high-dimension partial differential equations (PDEs). PDEs provide numerical approximations to computational expensive problems, being widely used in many scientific and engineering fields [10], [11], [12], [13]. This allows scientists to accurately model phenomena such as scalar-wave propagation modeling [14], heat conduction, acoustic diffusion, etc.

Algorithm 1 shows the pseudo-code of a generic 3-D stencil solver kernel for spatial loops. It is implemented as a triple nested loop traversing the complete data structure while updating each grid point. The computation of every output element usually requires: a) the weighted contribution of some near neighbors in each direction defined by the physics of the problem, b) the previous value of that element in a time *t-1* (for second order in-time stencils) and, c) a single corresponding point from other input arrays. The code normally uses two copies of the spatial grid (at time steps *t* and *t+1*) swapping their roles as source and destination on alternate time steps, as shown in the Algorithm 1.

Algorithm 1 The 3-D stencil solver kernel. *width, height, depth* are the dimensions of the data set including border (*halo*) points.

```
1:  for time = 0; time < TimeMax; time++ do
2:      for z = 1; z < depth − BorderSize; z++ do
3:          for y = 1; y < height − BorderSize; y++ do
4:              for x = 1; x < width − BorderSize; x++ do
5:                  stencil_solver_kernel();
6:              end for
7:          end for
8:      end for
9:      tmp = Input_Grid; Input_Grid = Output_Grid; Output_Grid = tmp;
10: end for
```

An important feature of these algorithms is that 3-D stencil kernels usually suffer from a high cache miss rate and poor data locality. The reason is that, for input sizes that exceed the cache capacity, by the time we reuse an entry from the dataset it has already been replaced from the cache. Moreover, the non-linear memory access pattern of 3D based implementations creates additional memory stalls. As a result, standard implementations of the 3D stencil solvers typically reach a small fraction of the hardware's peak performance [15].

B. Intel Xeon Phi Architecture

The Intel Xeon Phi (Knights Corner) coprocessor is the first commercial product of the Intel MIC family. The design is purely throughput oriented, featuring a high number of simple cores (60+) with support for 512-bit wide vector processing units (VPU). The VPU can be used to process 16 single-precision or 8 double-precision elements per instruction. To keep power dissipation per unit area under control, these cores execute instructions in-order and run at a low frequency (<1.2Ghz). The architecture is backed by large caches and high memory bandwidth. Xeon Phi is based on the x86 ISA, allowing a certain degree of compatibility with conventional x86 processors (but not binary).

The architecture is tailored to run four independent threads per core, where each in-order core can execute up to two instructions per cycle. Unlike latency oriented architectures, the MIC architecture assumes that applications running on the system will be highly parallel and scalable. In order to hide the cache/memory latency caused by the in-order nature of the cores, the scheduling policy swaps threads on each cycle. When an application runs a single thread per core, the scheduler switches to a special *null thread* before going back to the application thread. Suffice it to say, Intel recommends at least two threads per core, although the optimal may range from 2 to 4. Running a single thread per core will reduce the peak capacity of the system by half.

C. Related Work

Multi-core systems [16] provide good opportunities for parallelizing stencil applications. Authors in [17] present a thorough methodology to evaluate and predict stencil code performance on complex HPC architectures. The authors in [18] introduce a methodology that directs programmer efforts toward the regions of code most likely to benefit from porting to the Xeon Phi as well as providing speedup estimates. Other researchers [19] investigate the porting and optimization of the test problem basic N-body simulation for the Intel Xeon Phi coprocessor [20], which is also the foundation of a number of applications in computational astrophysics and biophysics.

Many proposals have been focused on improving cache reuse. Tiling is a program transformation that can be applied to capture data reuse when data does not fit in cache. In [21], [22] the authors focus on exploiting data locality by applying tiling techniques. On the other hand, works like [23], [24], [25], [26] considered locality and parallelism issues. Kamil *et al.* [23] examine several optimizations targeted to improve cache reuse across stencil sweeps. Their work includes both an implicit cache oblivious approach and a cache-aware algorithm blocked to match the cache structure. This enables multiple iterations of the stencil to be performed on each cache-resident portion of the grid. Authors in [24] developed

an approach for automatic parallelization of stencil codes that explicitly addresses the issue of load-balanced execution of tiles. Finally, recent contributions on Intel Xeon Phi coprocessor revealed its high compute capabilities for many HPC applications [27] [2].

III. Parameters Evaluated

One of the key design features of the Intel Xeon Phi architecture is the use of wide SIMD registers and vector functional units to achieve the best performance out of this architecture. This is usually automatically done by the compiler, with some help of the programmer.

There are several issues that need to be addressed to allow for the automatic vectorization of the code: (1) To allocate all rows of the 3-D arrays consecutively in memory (i.e., row major order); (2) To perform data alignment with an alignment factor of 64 bytes; (3) To apply the "padding" technique by adding some elements (if needed) to ensure that the first element of each row is on the desired address boundary (64 bytes in Xeon Phi); (4) To put the `#pragma ivdep` before the inner loop in the codes to notify the compiler that it can assume that the array pointers point to the disjoint location. Without all these hints, the compiler may not be able to correctly identify the inner loop as vectorizable and could fail in vectorizing the code.

Our base codes consider all these points related to vectorization. In this Section, we show other key important issues that should be considered to achieve optimal performance on the Intel Xeon Phi architecture. The trade-offs between different parameters for the stencil codes and the Xeon Phi are disclosed to help the designer make an informed decision.

A. Scalability and Affinity

3-D stencil kernels operate over an input data represented as a three-dimensional array of elements (double precision floating point in our experiments). The parallelization process consists of dividing these kernels in different "threads" or "tasks" to run in parallel on the target architecture. Our implementations have been developed using the C language with the OpenMP extensions. OpenMP development is based on *#pragma* statements that are captured by the compiler, validated and translated to the appropriate function calls to the OpenMP library and runtime system.

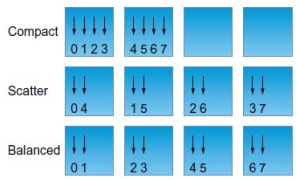


Fig. 1
KMP affinity-type distributions.

In this work, we have evaluated some parallelization strategies for the three *for* loop statements (see Algorithm 1 in Section II-A). Our best results are obtained when the two-most outer loops are parallelized using the OMP pragma construct *#pragma omp parallel for collapse (2)*. The use of the *collapse (2)* clause is useful to merge loop iterations, increasing the total work units that will be partitioned across the available threads. The inner loop remains sequentially to ease the vectorization process by the compiler.

For scalability experiments, we set the number of threads to the desired number by using the environment variable `OMP_NUM_THREADS`, or using the function `omp_set_num_thread` in the source code.

We can test the effect of the thread affinity by changing the environment variable `KMP_AFFINITY` to *compact*, *balanced*, or *scatter*. This variable controls how threads are assigned to cores. When using *compact*, threads are placed as close as possible, filling up the physical cores one after another, whereas with *scatter* all threads are distributed across the entire system with non-unit stride regarding the logical core IDs. *Balanced* is an intermediate option, when the run-time tries to select the best place where to put a thread in a "balanced" way. Figure 1 shows an example of 8 threads and 4 cores.

B. Blocking.

Stencil codes with an input size that does not fit on the higher cache levels of the processor will experience a significant performance degradation due to cache capacity misses. Code transformations that improve data locality can be useful to hide the complexities of the memory hierarchy, improving overall performance of 3-D stencil codes.

Algorithm 2 Blocking technique applied to the 3-D stencil solver.

```
for bz = 1; bz < depth − BorderSize; bz += depth_Tblock do
    for by = 1; by < height − BorderSize; by += height_Tblock do
        for bx = 1; bx < width − BorderSize; bx += width_Tblock do
            for z = bz; z < MIN(bz + depth_Tblock, depth − BorderSize); z++ do
                for y = by; y < MIN(by + height_Tblock, height − BorderSize); y++ do
                    for x = 1; x < MIN(width_Tblock, width − BorderSize − bx); x++ do
                        stencil_solver_kernel();
                    end for
                end for
            end for
        end for
    end for
end for
```

Blocking is a transformation which groups loop iterations into subsets of size N (or tiles). The size of the tiles needs to be adjusted to fit in the cache in order to obtain maximum performance gains by exploiting data locality. In this way, cache misses can be minimized by bringing data blocks into cache once for all necessary accesses.

In our 3-D stencil codes the goal is to exploit data locality, focusing on increasing the reuse of the elements of the plane (*X-Y*) as the code works along the column of the Z-axis. The first step is to create tiles of reduced sizes *bz*, *by* and *bx*. Next, three additional loops are created over the three existing loops to traverse the dataset in tiles of the selected sizes. A blocking version of a generic 3-D stencil is shown in algorithm 2.

C. Input Matrix Shape.

Technical computing applications often need to discretize the dataset into grids as part of the numerical representation of the problem. The grid size can sometimes drastically affect the performance of these algorithms on vector-based machine architectures, such as Intel Xeon Phi, as it dictates the memory access patterns.

Depending of the Z length and the memory access stride, the compiler will be able to do a lot of vectorizing in the inner loop, allowing maximum memory access bandwidth in Xeon Phis cache-based system.

Often a developer or user has flexibility in picking the block shapes for simulation if they are aware of the advantage of one shape over the other.

We have evaluated four different shapes for the input matrix size: one cuboid shape and three rectangular cuboid shapes (width x height x depth). Every one of these rectangular shapes has one dimension larger than the other two dimensions (that are equal in size).

IV. Evaluation

This section shows our experimental environment and evaluation results on Intel Xeon Phi for the three stencil kernels previously introduced.

A. Target Platforms

The experiments were conducted on a system contains two Intel Xeon E5-2650 CPUs that hold an Intel Xeon Phi coprocessor 7120P connected to the host via a PCIe connection. The Xeon Phi 7120P has 61 cores working at 1.238 GHz, 32KB of the L1 data and instruction caches and 512 KB of L2 cache. Each coprocessor core contains a wide 512-bit single instruction multiple data (SIMD) vector processing unit (VPU), which implements fused multiply-add operations. The architecture provides a theoretical peak computation of 1210 gigaflop per second (GFlop/s) for double precision floating point values (64 bits).

Another important feature of Intel Xeon Phi coprocessors is its high memory bandwidth. The Xeon Phi 7120P has 16 memory channels, each 32-bits wide. At up to 5.0 GT/s transfer speed, it provides a theoretical bandwidth of 352 GB/s. We have used the Intel's `icc` compiler (version 14.0.2), Linux CentOS 6.5 with kernel 2.6.32 and Intel MPSS 3.4.3.

B. Target Kernels

We have evaluated three stencil solvers from different scientific fields. These solves cover a wide research area and have distinct computational features. The most common stencil code is represented by the 3-D acoustic diffusion stencil, which uses a stencil of 7-point spatial neighbors and second order in time. It uses three different matrices of the same size for the kernel calculation. Our next kernel is the 3-D isotropic seismic wave stencil of 25-point spatial neighbors and also second order in time. Finally, we have evaluated the simplest solver, i.e., the 3-D heat diffusion stencil of 11-point spatial neighbors and first order in time, which only uses two matrices for the stencil calculations.

These choices impact the arithmetic intensity (AI) (number of floating-point operations (flops) per byte of memory transferred) [15] of our different stencil kernels. A given AI can be easily linked to performance expectations when using the Roofline model methodology. This technique helps when measuring the performance level of a given implementation with respect to its achievable performance on a particular hardware system. For any given computer, the hardware specifications define a peak capacity for computing flops and transferring data to and from memory (memory bandwidth).

For the Xeon Phi architecture using double precision data, the theoretical AI is around 5.5 Flop/Byte. This means that, for this architecture, we can characterize a given compute kernel as compute bounded if its AI is greater that 5.5 Flop/Byte, or memory bounded in the opposite case. The acoustic diffusion stencil has a low arithmetic intensity (slightly more than 0.5). The seismic stencil has greater arithmetic intensity than the previous one, although as it uses a different spatial matrix for storing physical characteristics (four matrices in total), and its arithmetic intensity is only slightly greater than 1. Finally, the arithmetic intensity of the Heat kernel is close to 1.5. As we realized, all of them are memory-bounded kernels.

C. Evaluation Scalability and Affinity

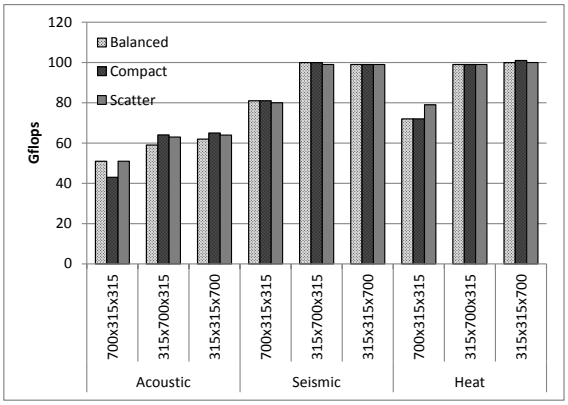
Figure 2 shows our affinity and scalability experimental results with the blocking vectorized double-precision code for the 3-D heat stencil, 3-D acoustic stencil and 3-D seismic stencil. More specifically, Figure 2(a) shows results when running 244 threads and Figure 2(b) when running 122 threads using the balanced, compact and scatter affinity policies.

Increasing the number of threads to 3 and 4 did not have a huge impact on performance, meaning that either we already saturated the double-precision execution units or the memory bandwidth of the core. Nevertheless, we found the best performance was achieved using KMP_AFFINITY=scatter and OMP_NUM_THREADS=183 (three threads from each core). Moreover, the choice of "KMP_AFFINITY" had minimal performance effects when all 244 threads were utilized.

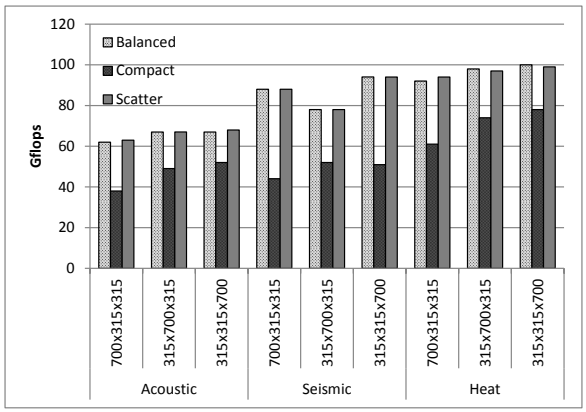
We conclude that the KMP_AFFINITY to *scatter* is the best suited to distribute the threads across the Xeon Phi cores, maximizing the usage of the cache storage space.

D. Achieving the Best Block Size

In this Section we show the experimental results when looking for the best block size in ours evaluated kernels. Figure 3 shows the blocking size evaluation for the vectorized double-precision code for all three kernels using 244 threads and scatter affinity. The analysis of different block sizes shows that the blocking technique works best on the Y and Z dimensions, but not on the X axis. More specifically, Figure 3(a) shows that the best blocking size achieves significant (up to 20%) performance benefits for the 3-D acoustic diffusion stencil kernel. Figure 3(b) shows similar results, with a performance increase of up to 30% for the 3-D isotropic seismic wave stencil kernel.



(a) KMP affinity-type distributions with 244 threads



(b) KMP affinity-type distributions with 122 threads

Fig. 2
AFFINITY RESULTS IN OURS KERNELS.

Finally, the 3-D heat diffusion stencil kernel (Figure 3(c)) achieves performance benefits of up to 10% when using blocking mechanisms. Specifically, a block size of *by=2* and *bz=40* offers good results for the *Acoustic* and *Heat* kernel, whereas a block size of *by=2* and *bz=8* is the best option for the Seismic code.

E. Matrix Shape Evaluation

In this section we present the experimental results based on the double-precision code for our three kernels running with 244 threads, scatter affinity, the best blocking for each kernel while varying the shape of the dataset matrix.

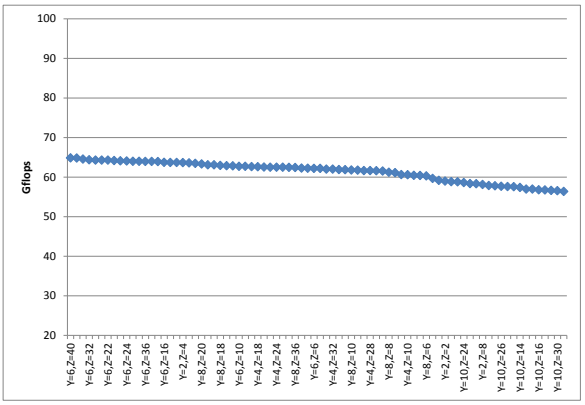
Our evaluation includes a cuboid shape dataset block (e.g. 420x420x420) and several non-cuboid shape dataset blocks (700x315x315, 315x700x315 and 315x315x700). Figure 4 shows that, with the best blocking configuration, changing the shape of the dataset has an impact on performance, around 20% for the seismic and heat codes, and 10% for the acoustic one.

V. CONCLUSIONS AND FUTURE WORK

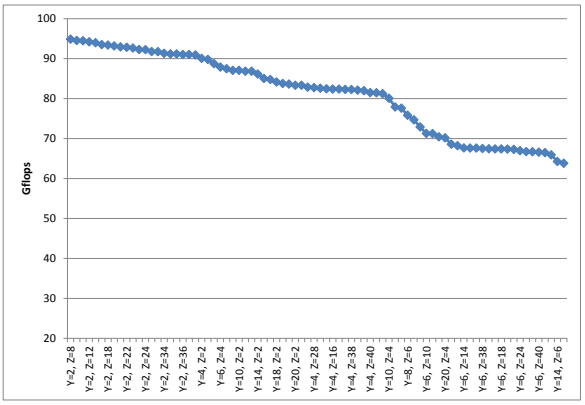
Real world applications based on Stencil computations have a real impact on the society, enabling scientists and indirect stakeholders such as end users or service providers benefit from improvements in their codes. These improvements not only provide faster results, but also enhance accuracy by allowing more detailed simulations of different phenomena.

Our paper evaluates three stencil-based kernels that are widely applied to different scientific fields. We have evaluated some key issues that have a positive impact to achieve optimal performance on the Xeon Phi. Among them, we highlight trade-offs between scalability and affinity, blocking and the effect of input matrix shape.

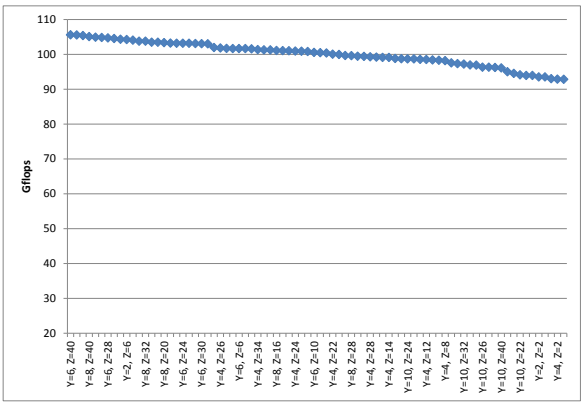
Our experimental results yield small performance gains using scatter affinity. We have also found a variability of up to 30% of performance in those codes de-



(a) Acoustic



(b) Seismic



(c) Heat

Fig. 3
BLOCKING (Y AXIS, Z AXIS).

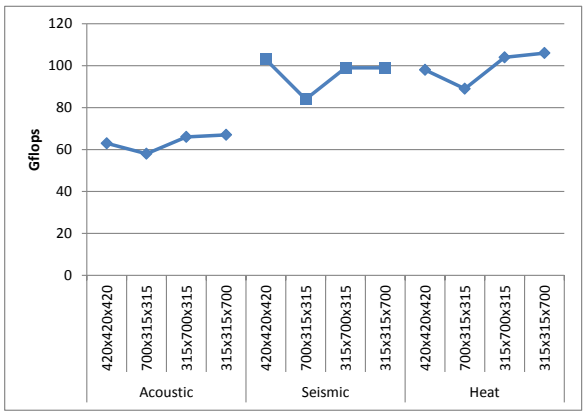


Fig. 4
SHAPE DATASET BLOCK.

pending of the block size, working the blocking better on the Y and Z dimensions, but not on the X axis. Finally, we have showed that grid shape has little impact on performance as long as the best block size is selected.

As for future work, we are interesting to extend our evaluation to larger datasets. To properly handle big sizes, we plan to split input data among different Xeon Phi cards, analyzing the communication effects on the performance.

ACKNOWLEDGEMENTS

This work is jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 15290/PI/2010, and the Spanish MINECO and Spanish MEC, as well as European Commission FEDER funds under grant number TIN2012-31345. This work has been also funded by the Nils Coordinated Mobility under grant 012-ABEL-CM-2014A, in part financed by the European Regional Development Fund (ERDF). Mario Hernández was supported by a research grant from the PROMEP under the Teacher Improvement Program (UAGro-197) México.

REFERENCES

[1] Jim Jeffers and James Reinders, *Intel Xeon Phi coprocessor high-performance programming*, Elsevier Waltham (Mass.), Amsterdam, Boston (Mass.), 2013.

[2] Rezaur Rahman, *Intel Xeon Phi Coprocessor Architecture and Tools: The Guide for Application Developers*, Apress, Berkely, CA, USA, 1st edition, 2013.

[3] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Yao Zhang, and V. Volkov, "Parallel Computing Experiences with CUDA," *IEEE, Micro*, vol. 28, no. 4, pp. 13–27, July 2008.

[4] "Top 500 supercomputer site," [last access 15 May 2015], `http://www.top500.org/`.

[5] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick, "The landscape of parallel computing research: A view from berkeley," Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[6] Yuan Tang, Rezaul Alam Chowdhury, Bradley C Kuszmaul, Chi-Keung Luk, and Charles E Leiserson, "The pochoir stencil compiler," in *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2011, pp. 117–128.

[7] Matthias Christen, Olaf Schenk, and Helmar Burkhart, "Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures," in *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 2011, pp. 676–687.

[8] Lukasz Szustak, Krzysztof Rojek, Roman Wyrzykowski, and Pawel Gepner, "Toward efficient distribution of mpdata stencil computation on intel mic architecture," *Proce. HiStencils*, vol. 14, pp. 51–56, 2014.

[9] Shoaib Kamil, Cy Chan, Leonid Oliker, John Shalf, and Samuel Williams, "An auto-tuning framework for parallel multicore stencil computations," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[10] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller, *Multigrid*, Academic press, 2000.

[11] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick, "Stencil Computation Optimization and Autotuning on State-of-the-art Multicore Architectures," in *Proc. of the 2008 ACM/IEEE Conference on Supercomputing*. 2008, SC '08, p. 4, IEEE Press.

[12] Matteo Frigo and Volker Strumpen, "Cache oblivious stencil computations," in *Proceedings of the 19th Annual International Conference on Supercomputing*, New York, NY, USA, 2005, ICS '05, pp. 361–366, ACM.

[13] V.T. Zhukov, M.M. Krasnov, N.D. Novikova, and O.B. Feodoritova, "Multigrid effectiveness on modern computing architectures," *Programming and Computer Software*, vol. 41, no. 1, pp. 14–22, 2015.

[14] Dimitri Komatitsch, Gordon Erlebacher, Dominik Göddeke, and David Michéa, "High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster," *Journal of computational physics*, vol. 229, no. 20, pp. 7692–7714, 2010.

[15] James Reinders and James Jeffers, *High Performance Parallelism Pearls, Multicore and Many-core Programming Approaches*, chapter Characterization and Auto-tuning of 3DFD, pp. 377–396, Number 23. Morgan Kaufmann, 1st edition, june 2014.

[16] Sreeram Potluri, Akshay Venkatesh, Devendar Bureddy, Krishna Kandalla, and Dhabaleswar K Panda, "Efficient intra-node communication on intel-mic clusters," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 128–135.

[17] Raúl de la Cruz and Mauricio Araya-Polo, "Modeling stencil computations on modern hpc architectures," 2014.

[18] J. Peraza, A. Tiwari, M. Laurenzano, L. Carrington, W.A. Ward, and R. Campbell, "Understanding the performance of stencil computations on Intel's Xeon Phi," in *Cluster Computing (CLUSTER), 2013 IEEE Intern. Conference on*, Sept 2013, pp. 1–5.

[19] Jianbin Fang, Henk Sips, LiLun Zhang, Chuanfu Xu, Yonggang Che, and Ana Lucia Varbanescu, "Test-driving intel xeon phi," in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA, 2014, ICPE '14, pp. 137–148, ACM.

[20] Andrey Vladimirov and Vadim Karpusenko, "Test-driving intel xeon phi coprocessors with a basic n-body simulation," *Coflax International*, 2013.

[21] G. Rivera and Chau-Wen Tseng, "Tiling optimizations for 3d scientific computations," in *Supercomputing, ACM/IEEE 2000 Conference*, Nov 2000, pp. 32–32.

[22] Yonghong Song, Rong Xu, Cheng Wang, and Zhiyuan Li, "Data locality enhancement by memory reduction," in *Proceedings of the 15th international conference on Supercomputing*. ACM, 2001, pp. 50–64.

[23] Shoaib Kamil, Kaushik Datta, Samuel Williams, Leonid Oliker, John Shalf, and Katherine Yelick, "Implicit and explicit optimizations for stencil computations," in *Proceedings of the 2006 Workshop on Memory System Performance and Correctness*, New York, NY, USA, 2006, MSPC '06, pp. 51–60, ACM.

[24] Sriram Krishnamoorthy, Muthu Baskaran, Uday Bondhugula, J. Ramanujam, Atanas Rountev, and P Sadayappan, "Effective automatic parallelization of stencil computations," in *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY, USA, 2007, PLDI '07, pp. 235–244, ACM.

[25] Zhiyuan Li and Yonghong Song, "Automatic tiling of iterative stencil loops," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 26, no. 6, pp. 975–1028, 2004.

[26] Yun Zou and Sanjay Rajopadhye, "Cache efficient parallelizations for uniform dependence computations," , no. TR/CS-14-101, May 2014.

[27] Jianbin Fang, Ana Lucia Varbanescu, Henk Sips, Lilun Zhang, Yonggang Che, and Chuanfu Xu, "An empirical study of Intel Xeon Phi," *arXiv preprint arXiv:1310.5842*, 2013.