Historic Learning Approach for Auto-tuning OpenACC Accelerated Scientific Applications

Shahzeb Siddiqui[1], Saber Feki[1]

[1] King Abdullah University of Science and Technology, Kingdom of Saudi Arabia
{Shahzeb.Siddiqui, Saber.Feki}@kaust.edu.sa

Abstract. The performance optimization of scientific applications usually requires an in-depth knowledge of the hardware and software. A performance tuning mechanism is suggested to automatically tune OpenACC parameters to adapt to the execution environment on a given system. A historic learning based methodology is suggested to prune the parameter search space for a more efficient auto-tuning process. This approach is used to tune the OpenACC gang and vector clauses for a better mapping of the compute kernels onto the underlying architecture. Our experiments show a significant performance improvement against the default compiler parameters and drastic reduction in tuning time compared to a brute force search-based approach.

1 Introduction

Accelerators are gradually becoming mainstream in supercomputing as their capability to significantly accelerate a large spectrum of scientific applications at a higher power efficiency has been clearly identified and proven. Moreover, with the introduction of high level programming models such as OpenACC [1] and OpenMP 4.0 [2], these devices are becoming more accessible and practical to use by a larger scientific community. OpenACC was announced in the ACM/IEEE Supercomputing Conference 2011 as a new standard for parallel programming targeting hardware accelerators. Although the goal of the standard is to increase programmer productivity by using compiler directives, getting the best performance of the target device is still tedious and requires a significant effort by the developer to tune some of the OpenACC annotations and the corresponding parameters. As a matter of fact, without tuning, a suboptimal performance is recorded on different applications while using the latest implementations of OpenACC compilers. In this work, we propose a new methodology for empirical tuning of OpenACC accelerated scientific applications to relieve the end user from this burden. The OpenACC *gang* and *vector* clauses are used to map nested loops to the underlying hardware architecture. The auto-tuning engine performs a search on the space of possible uses of these clauses and their corresponding attributes, for a better mapping and thus to improve the performance of the offloaded kernel. However, when the number of parameters is quite big, the search space becomes significantly large, and the tuning procedure becomes very expensive and unpractical. We suggest a performance tuning strategy that reduces the

cost of tuning by pruning the search space using the historic knowledge of previous tuning operations performed on similar problem sizes. This approach was presented and its effectiveness was demonstrated on auto-tuning MPI communication operations in the Abstract Data and Communication Library (ADCL) [3], [4].

The remainder of the paper is organized as follows. Section 2 introduces our methodology to efficiently tune OpenACC clauses using a historic learning approach. In section 3, the experimental setup is described and the performance results of the proposed tuning strategy are reported. Section 4 presents the related work in this research area. Finally, Section 5 summarizes our findings and future work.

2 Performance Tuning Methodology

The OpenACC standard offers flexibility to the developer to further tune the *loop pragma* with the *gang* and *vector* clauses and therefore control the mapping of the nested loops to the underlying hardware specification; that is the threads partitioning in the accelerator. The *gang* clause specifies in how many groups to aggregate the parallel threads generated by the parallelization of the given loop, which corresponds to the shape and size of the grid of blocks in the GPU environment. The vector clause specifies the granularity of the parallel threads per gang, which translates to the dimensionality and the size per dimension of each of the thread blocks in NVIDIA hardware. The latest compiler technology relies on heuristics to specify these parameters for a given application. Our analysis showed that a significant improvement could be further obtained by spending more effort in tuning these parameters. Our methodology of tackling this particular aspect is described next.

2.1 Tuning Methodology of OpenACC loop clauses

The strategy for auto-tuning proposed here is based on empirical evaluation of different uses of the OpenACC clauses *gang* and *vector* in nested loops as depicted in Fig. 1. The performance tuning procedure is in two steps. In the first phase, the performance of different placements of the *gang* and *vector* clauses within the nested loops is evaluated and the best performing one is selected. At this initial phase, we keep the compiler choice of the numbers of gangs and vectors by omitting any specific values. Once the optimal placement of these clauses is determined, we explore in the second phase different numbers of gangs and vectors other than the ones chosen by the compiler. This tuning methodology is referred as the brute force tuning in the experimental results section. It is worth noting that the set of meaningful configurations is constrained by the specification of the accelerator. Despite this restriction, this parameter space can be very huge, and using an exhaustive search on all possible combinations could be considerably time-consuming and unreasonable. A historic learning based approach able to shrink the search space and accelerate the tuning process is detailed next.

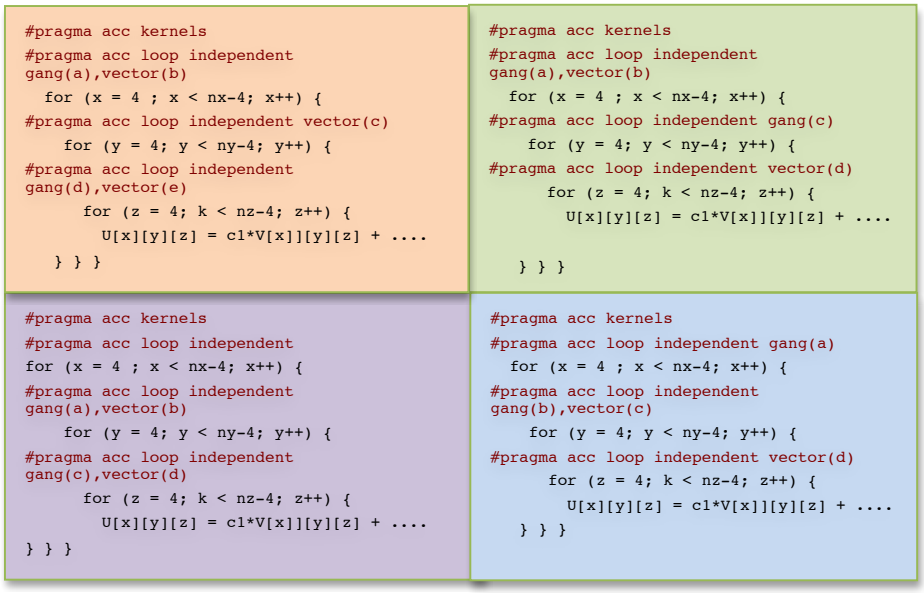

Fig. 1. Different placements of the gang and vector clauses in three nested loops

2.2 Historic Learning Approach

Tuning OpenACC gang and vector clauses with an exhaustive search of the full parameter space is time consuming. We suggest here a tuning methodology based on the previous tuning results of the same application, for different problem sizes, on the same hardware. Practically, a learning phase is needed for building a knowledge database out of the best tuning parameters for various input sizes. Given a new problem size PS_{new} for the same application, the closest problem size in the knowledge base is then identified using the Euclidian distance for example. This problem size is used as a reference for the suggested tuning approach and is referred as PS_{ref}. The tuning parameters of PS_{ref} are used to define a subset of the parameters space to be explored by the tuning engine for PS_{new}. This subset consists in a smaller range of gang and vector values, the closest to the optimal parameters for PS_{ref}. The search space of possible parameters combinations is then drastically reduced and the tuning procedure becomes much faster and thus more attractive. The tuning results of PS_{new} are then included to enrich the tuning knowledge database for future reuse. This tuning methodology is referred as the historic learning tuning in the experimental results section.

3 Experimental Results

The test bed hardware and software specifications and the test application used for this performance analysis are first described. Following that, we showcase the

importance of the gang and vector clauses placement within nested loops to the performance tuning of OpenACC applications. The performance gain of applying the suggested tuning methodology on the test application as well as the tuning time reduction by using the historic learning approach is reported.

3.1 Test Bed Specifications and Test Application

The test bed used for performance evaluation consists in a dual socket CPU system hosting four NVIDIA Kepler K20c GPU cards. Each socket is an eight-core Sandy Bridge Intel(R) Xeon(R) CPU E5-2650, running at a clock speed of 2.00GHz. The software stack consists in the PGI compiler version 12.9, and the NVIDIA CUDA driver 5.0.
In our experiments, we used the isotropic finite difference kernel, which constitutes the building block for the Reverse Time Migration (RTM) and the Full Waveform Inversion (FWI) applications, extensively used by the oil and gas exploration industry for the velocity model building and seismic imaging of the sub-surface. The Reverse Time Migration application consists in a forward modeling and backward migration using a finite difference kernel that solves the acoustic wave equation.

$$\frac{1}{c^2}\frac{\partial^2 P}{\partial t^2} = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2}$$

where c is the velocity of the propagated wave and P is the wavefield pressure. The 3D finite difference stencil scheme is 8^{th} order in space and 2^{nd} order in time.

3.2 Importance of Gang and Vector Placement

The application of the brute force tuning methodology to a set of ten different problem sizes resulted in a performance improvement of up to 30% while compared to the base code version tuned by the compiler. It is emphasized here that the optimal gang and vector clauses placement varies from one problem size to another. Table 1 shows for each 3D problem size, the chosen grid and block sizes by the compiler and the corresponding tuned parameters. The color code of each row corresponds to a different placement of the gang and vector clauses in the three nested loops as depicted in Fig 1. We can conclude here that the first phase in the proposed tuning methodology is crucial as the clauses placement in the nested loops has a significant importance in the performance tuning of the OpenACC code.

Table 1: Compiler choices versus tuning results for gang and vector placement and values for different problem sizes

3D Domain Size	Grid/block sizes chosen by PGI	Tuned grid/block sizes
128x128x128	grid: [2x30] block: [64x4]	grid: [30x120] block: [64x4]
256x256x256	grid: [4x62] block: [64x4]	grid: [248x6] block: [64x6]
512x512x512	grid: [8x126] block: [64x4]	grid[504x63] block: [32x8]
640x640x640	grid: [10x158] block: [64x4]	grid: [10x316] block: [64x4x2]
128x128x640	grid: [10x30] block: [64x4]	grid: [10x64] block: [64x4]
128x640x128	grid: [2x158] block: [64x4]	grid: [4x256] block: [64x4]
640x128x128	grid: [2x30] block: [64x4]	grid: [2x316] block: [64x4x2]
640x640x128	grid: [2x158] block: [64x4]	grid: [2x316] block: [64x4x2]
640x128x640	grid: [10x30] block: [64x4]	grid: [10x316] block: [64x4x2]
28x640x640	grid: [10x158] block: [64x4]	grid: [10x256] block: [128x4]

3.3 Performance Tuning Results

In the brute force search-based tuning methodology, the performance of the compute kernel is evaluated with all possible gang and vector values allowed by the hardware specification of the K20c GPU. In our experiment, the gang values were chosen at increments of 2, starting from 2 to 1,024. The vector values were chosen in multiples of 32 (warp size), starting from 32 to 1,024. The total search space consists of 16,384 combinations. The brute force tuning method is a very time consuming process yet very simple in finding the best possible gang and vector tuple. The historic learning algorithm is used to identify a reference problem size PS_{ref} with a known solution (i.e. gang and vector tuple). The search space is then reduced by selecting a subset range of parameters keeping only the closest ten values of gang and vector with regard to the reference problem size solution. Therefore, the parameters space to be explored and evaluated is significantly reduced to only a 100 combination. The brute force search tuning method was first applied to 25 problem sizes and the best tuning parameters are stored in a knowledge base. Then, another set of 8 different problem sizes are tuned using both the brute force search and the historic learning tuning approaches. The performance speedups in comparison to the compiler tuned code version as well as the tuning time are recorded while using either of the tuning approaches for each of the eight new investigated problem sizes. As shown in Fig. 1, the tuning procedure using either the brute force search or the historic learning tuning method resulted in a better performance than the compiler default tuning. Indeed, a performance increase of up to 80% is recorded against the performance of the base code. Moreover, the performance of the code while tuned with the historic learning approach is within less than 1.5% of the best possible performance recorded while using a brute force search. The two cases where the historic learning tuned code was not performing as well as the brute force search tuned version are with the problem size 1500x400x400 and 800x600x400. The analysis of the data shows that the main reason for that is the lack of a close enough problem size in the database used for the prediction of a problem size of reference. Indeed, the two problem sizes have the

highest Euclidian distance to the problem size of reference used. Fig. 2 shows the required time for tuning a given problem size with the brute force and the historic learning based tuning methods. Our experiment shows that the tuning time is reduced dramatically by a factor of 18 to 52 times while using the new proposed tuning approach. At the same time, a comparable performance to the brute force search approach is achieved as detailed before.

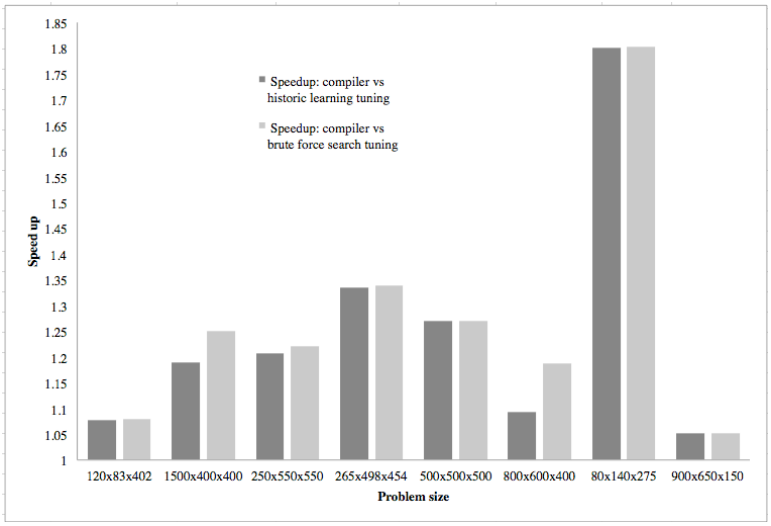

Fig. 3. Performance speedup analysis using the different tuning methodologies in comparison to the compiler tuning performance

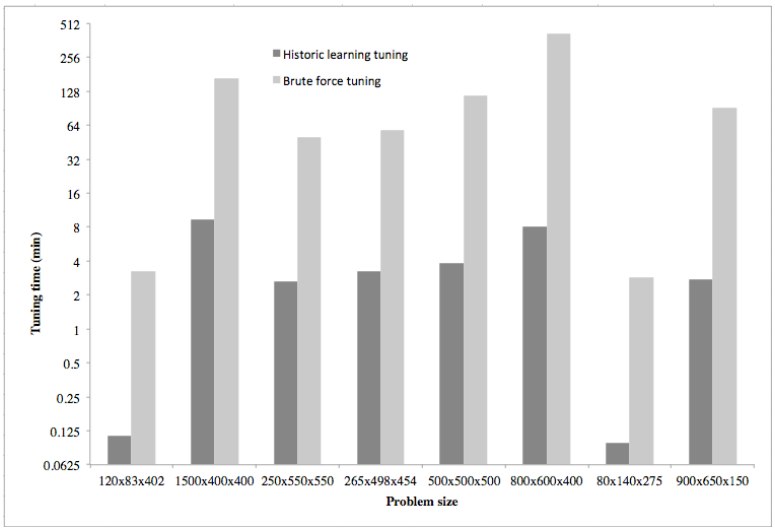

Fig. 4. Tuning time using the brute force and the historic learning tuning approach

4 Related Work

A significant research has been conducted in tuning applications written for GPUs. At the compiler level, another directive-based programming models called HMPP [6], is presented along with its tuning methodology. The CAPS OpenACC compiler also includes an auto-tuning driver that can explore the optimization space to tune kernel regions [15]. At the application level, researchers applied various tuning mechanisms to GPU codes such as sparse matrix-vector multiply [5], stencil computations [11], [7], and computational electromagnetics [14]. Vuduc proposed in [12] a statistical approach for automatic performance tuning of matrix-matrix multiply operation. AtuneRT [13] is an application-independent auto-tuner, which optimizes GPU-specific parameters such as block size and loop-unrolling degree.

The historic learning approach was applied to the runtime tuning of MPI communications in the abstract data and communication library (ADCL) [8], [9]. The notion of historic learning is also implemented to a limited extent in FFTW [10], namely with a feature called *Wisdom*. The user can export experiences gathered in previous runs into a file, and reload it at subsequent executions. However, the wisdom concept in FFTW lacks any notion of related problems, i.e. wisdom information can only be reused for exactly the same problem size that was used to generate it.

5 Conclusions and Future Work

A historic learning-based performance tuning of OpenACC accelerated applications is presented. The performance results obtained by the proposed tuning methodology on a finite difference kernel showed a significant performance gain against the compiler-tuned code and close to the optimal performance that can be obtained with an exhaustive search. Furthermore, the time needed for tuning is reduced drastically compared to the brute force tuning technique. Nevertheless, the main limitation of this approach is its dependency on the historic data that is crucial for a good prediction and therefore for achieving a performance close to the tuned code with a brute force search.
Our future work includes the automation of the tuning process including a code generator tool along with a tuning engine and its application to a larger spectrum of scientific applications and on a variety of accelerator architectures including different NVIDIA GPU generations and Intel's Xeon Phi coprocessors. Other machine learning algorithms such as Bayesian classifiers and support vector machines will be explored for a better prediction of the most similar problem size to use as a reference to shrink the search space.

Acknowledgments. The authors would like to thank NVIDIA for the hardware donation to King Abdullah University of Science and Technology in the context of the CUDA Research Center award.

References

1. OpenACC Standard specification, www.openacc-standard.org
2. OpenMP 4.0 specification, www.openmp.org/mp-documents/OpenMP4.0.0.pdf
3. Gabriel, E., Feki, S., Benkert, K., Chaarawi, M.: The abstract data and communication library. Journal of Algorithms and Computational Technology, 2(4):581–600, 2008
4. Gabriel, E., Feki, S., Benkert, K., and Resch, M.: Towards performance and portability through runtime adaption for high performance computing applications. In International Supercomputing Conference, Dresden, Germany, June 2008
5. Choi, J.W., Singh, A., Vuduc, R.W.: Model-driven autotuning of sparse matrix-vector multiply on gpus. In Proceedings of the 15th Symposium on Principles
6. Dolbeau, R., Bihan, S. and Bodin, F.: HMPP: a hybrid multi-core parallel programming environment. In the1st Workshop on General Purpose Processing on Graphics Processing Units, GPGPU, 2007.
7. Feki, S., Siddiqui, S.: Towards Automatic Performance Tuning of OpenACC Accelerated Scientific Applications. In GPU Technology Conference, San Jose, California, USA, 2003
8. Feki, S., Gabriel E.: A Historic Knowledge Based Approach for Dynamic Optimization. In proceedings of the International Conference on Parallel Computing, P. 389-396, 2009
9. Feki S., Gabriel, E.: Incorporating Historic Knowledge into a Communication Library for Self-Optimizing High Performance Computing Applications. In second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Venice, Italy, 2008
10. Frigo, M., Johnson, S.: The design and implementation of FFTW3. Proceedings of IEEE, 93(2):216–231, 2005
11. Mametjanov, A., Lowell, M.C., Norris, B.: Autotuning Stencil-Based Computations on GPUs, Cluster Conference, Beijing, China, 2012
12. R. Vuduc, J. W. Demmel, and J. A. Bilmes. Statistical models for empirical search-based performance tuning. International Journal for High Performance Computing Applications, 18(1):65–94, 2004.
13. Tillmann, M., Karcher, T., Dachsbacher, C., Tichy, W.F.: Application-independent Autotuning for GPUs, In International Conference on Parallel Computing, Munich, Germany, 2013
14. Feki, S., Al-Jarro, A, Bagci, H.: Multi-GPU-based Acceleration of the Explicit Time Domain Volume Integral Equation Solver Using MPI-OpenACC. IEEE International Symposium on Antennas and Propagation and USNC/URSI National Radio Science, Lake Buena Vista, Florida, USA, 2013.
15. Bodin, F.: Using CAPS Compiler on NVIDIA Kepler and CARMA Systems, Supercomputing, Salt Lake City, Utah, USA, 2012