Proceedings of the 17th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2017
4–8 July, 2017.

Analysis of OpenACC Performance Using Different Block Geometries

Daniel Barba[1], Arturo Gonzalez-Escribano[1] and Diego R. Llanos[1]

[1] *Departamento de Informatica, Universidad de Valladolid, Spain*

emails: daniel@infor.uva.es, arturo@infor.uva.es, diego@infor.uva.es

Abstract

OpenACC is a parallel programming model for automatic parallelization of sequential code using compiler directives or pragmas. OpenACC is intended to be used with accelerators such as GPUs and Xeon Phi. The different implementations of the standard, although still in early development, are primarily focused on GPU execution. In this study, we analyze how the different OpenACC compilers available under certain premises behave when the clauses affecting the underlying block geometry implementation are modified. These clauses are the Gang number, Worker number, and Vector Size defined by the standard.

Key words: OpenACC, GPU, block geometry, thread geometry

1 Introduction

OpenACC is an open standard intended to automatically parallelize sequential code and manage its execution in accelerators like GPUs or Xeon Phi coprocessors. It defines a number of compiler directives, also called pragmas. The main goal of OpenACC is to reduce both learning and coding time in a portable way [1]. The version of the OpenACC standard at the time of writing is the 2.5 [2].

The OpenACC standard was founded by Nvidia, CRAY, CAPS and PGI. The number of members now is larger, including both academic institutions and companies like the Oak Ridge National Laboratory, the University of Houston, AMD, and the Edinburgh Parallel Computing Centre (EPCC), among others.

There are several compilers that implement the OpenACC standard. The PGI compiler, developed by the Portland Group (subsidiary of Nvidia) is being distributed as part of

 ISBN: 978-84-617-8694-7

the Nvidia OpenACC Toolkit under a free 90-day license. Cray Inc. has its own OpenACC compiler, only available for use with their supercomputers. Pathscale Inc., a software developer for compilers and multicore software, also has an OpenACC implementation, the ENZO compiler.

Among the many academic or open-source alternatives there are the OpenUH compiler [3] by the University of Houston and accULL [4] from Universidad de La Laguna (Spain).

This work presents a study on the impact of different values for the clauses that affect the underlying block geometry of OpenACC-generated code. GPUs are very sensitive to the geometry of the thread-block chosen [5], and OpenACC makes use of the terms "gang", "worker" and "vector" in order to define different levels of parallelism. According to [6], the specification is ambiguous and this functionality depends directly on how each compiler is implemented. In this work, we measure the impact of the choice of an appropriate *thread-block geometry* when running a representative benchmark. By default, the geometry is decided by the compiler unless the specific clause is used inside the OpenACC directive. Our aim is to compare the resulting behaviour among different compilers and options. For thread-block geometry testing, we will modify the most representative of the benchmarks, testing several combinations of values for the clauses to specify gang, workers and vectors, and analyzing the differences in execution time for each compiler. This will offer some insight on the implementation of these clauses on each compiler.

Our contribution shows that the decisions made by each compiler is not always optimal, but manual tuning of the different values is not always possible for every compiler.

The rest of this paper is organized as follows. Section 2 describes the selected compilers. Section 3 shows our selected microbenchmark for testing the behaviour of the generated code when modifying the block geometry. Section 4 contains the result of our analysis about the impact on performance when changing the underlying block geometry. Finally, Section 5 concludes our paper.

2 Available Compilers

We mentioned several compilers in the Introduction. In this section we describe with more detail the compilers we were able to use for this study.

2.1 PGI Compiler

The PGI Compiler [7] is being developed by The Portland Group, being owned by Nvidia. This compiler is frequently presented in webinars, workshops, and conferences.

At the time of writing this paper, the PGI compiler is available for download as part of the OpenACC Toolkit from Nvidia. This toolkit includes a 90-day free trial, the possibility of acquiring an academic license for a whole year, or buying a commercial license.

 ISBN: 978-84-617-8694-7

2.2 accULL

The accULL [4] compiler developed by Universidad of La Laguna (Spain) is an open-source initiative. accULL consists on a structure of two layers containing YaCF [8] (Yet another Compiler Framework) and Frangollo [9], a runtime library. YaCF acts as a source-to-source translatori, while Frangollo works as an interface that provides the most common operations found in accelerators.

2.3 OpenUH

The OpenUH [3] compiler, developed by the University of Houston (USA) is another open-source initiative. It makes use of Open64, a discontinued open-source optimizing compiler.

3 Microbenchmark Description

In OpenACC, block size is defined by gangs, workers and vectors.Their choices affect the performance on memory-bound applications. To study this issue, we are going to use a very simple matrix addition implemented both in CUDA and OpenACC. Our decision is made by the fact that the problem is embarrassingly parallel, memory acceses are perfectly coalesced, and the computational load per global memory access is low (memory-bound application).

The standard only defines the different levels of parallelism, but it is up to each implementation to decide how are these levels exploited in the actual architecture. In order to obtain comparable results, some details should be taken into account. The CUDA version needs to be implemented using elastic kernels, using a fixed number of blocks, which equals the gang number in the OpenACC code. Also, since the OpenACC standard establishes that grid dimensions depend on the use of the collapse clause with nested loops, we have decided to make a one-dimensional grid. We evaluate a number of blocks in the grid ranging from one to 2048 (using only powers of two). The sequential code can be seen in Fig. 1 and the CUDA kernel in Fig. 2.

In OpenACC the X dimension of the CUDA block translates to vector length, whereas the Y dimension equals the worker number. We have also decided to use 512 threads per block, trying each possible combination of X and Y dimension values using powers of two.

4 Evaluation

In this section we analyze the impact of different choices for the geometry of the underlying thread-blocks in the OpenACC generated code.

 ISBN: 978-84-617-8694-7

```
#pragma acc data copyin(p_A[0:Size*Size],p_B[0:Size*Size]), copyout(p_C[0:Size*Size])
{
    int j;
    #pragma acc kernels
    #pragma acc loop independent gang(GANG), worker(WORKER), vector(VECTOR)
    for (j = 0; j < Size*Size; ++j)
    {
         p_C[j] = ALPHA*p_A[j] + p_B[j];
    }
} //End data region
```

Figure 1: Sequential Code for the Matrix Addition

```
__global__ void matrixKernel(float* p_A, float* p_B, float* p_C)
{
    int iterations = ((SIZE/blockDim.x)*(SIZE/blockDim.y)) / GANG;

    int iter;
    for (iter = 0; iter < iterations; ++iter)
    {
        int tx = (blockIdx.x + iter*GANG)*blockDim.x + threadIdx.x;
        int ty = threadIdx.y;
        int i = (tx/SIZE)*blockDim.y + ty;
        int j = (tx%SIZE);
        int offset = i*SIZE + j;
        if (offset < SIZE*SIZE) p_C[offset] = ALPHA*p_A[offset] + p_B[offset];
    }
}
```

Figure 2: CUDA Kernel for the Matrix Addition

 ISBN: 978-84-617-8694-7

4.1 Experimental Setup

We used a Nvidia GTX Titan Black to run the experiments. This GPU contains 2 880 CUDA cores with a clock rate of 980MHz and 15 SMs. It has 6GB of RAM, and Compute Capability 3.5. The host is a Xeon E5-2690v3 with 12 cores at a clock rate of 1.9GHz, and 64GB in four 12GB modules.

The PGI compiler is the one contained in the Nvidia OpenACC Toolkit, version 15.7-0, published in Jul 13, 2015. We used OpenUH version 3.1.0 (published in November 4, 2015), based on Open64 version 5.0 and using GCC 4.2.0, prebuilt, downloaded from the High Performance Computing Tools group website [10]. accULL is version 0.4alpha (published in November 28, 2013), downloaded from Universidad de La Laguna's research group "Computación de Altas Prestaciones" [11].

4.2 Block Geometry Sensibility of Generated Code

Observing the results obtained using the CUDA code in Fig. 3, we can see that the best results are obtained when the block number is high enough to make the GPU reach proper levels of occupation. The X dimension plays a huge role, but we expected to see an improvement in performance when the X dimension was at least 16. We suspect this is due to several factors, being the most important the behaviour of the cache when elastic kernels are used.

The results of the OpenACC code generated by PGI (Fig. 4) show that the only factor affecting the performance is the gangs number, whereas the variation of workers number and vector length does not play a significant role except when a vector length of one is used. In this case, performance is severely affected, which indicates a poor use of the cache. Overall, PGI's behaviour is the closest one to CUDA for this example.

When using OpenUH as the compiler for the OpenACC code (Fig. 5), all three parameters affect the performance of the generated code, which means that the parameters are actually being used to map the computation to the architecture resources. There is an exception when any of the three parameters is set to one. In this case, OpenUH seems to assume direct control and choose what it considers an adequate set of parameters for gangs number, workers number and vector length.

Finally, the results obtained by using accULL (Fig. 6) as our OpenACC compiler show that none of the parameters have any effect on the performance of the generated code, and it is the compiler itself who decides the value to be used.

5 Conclusions

During this work, we have realized that the OpenACC standard is very unspecific about how the different compilers should implement the three levels of parallelism. This allows

 ISBN: 978-84-617-8694-7

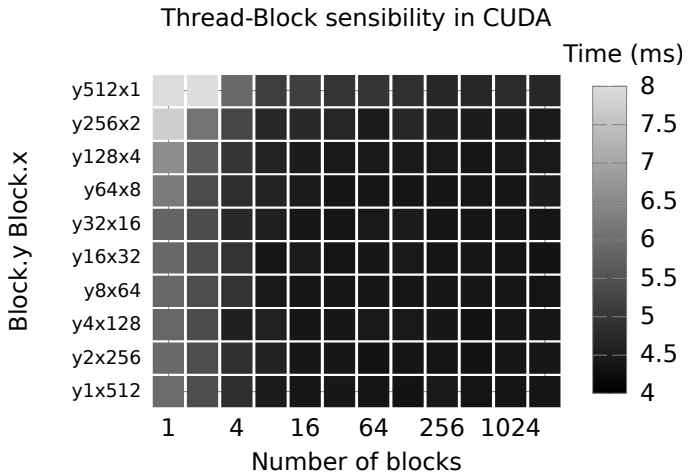

Figure 3: Effects of the Gang Number, Worker and Vector Length in the measured execution time using CUDA. Execution time is in milliseconds (lower is better). Darker is lower. Brighter is higher.

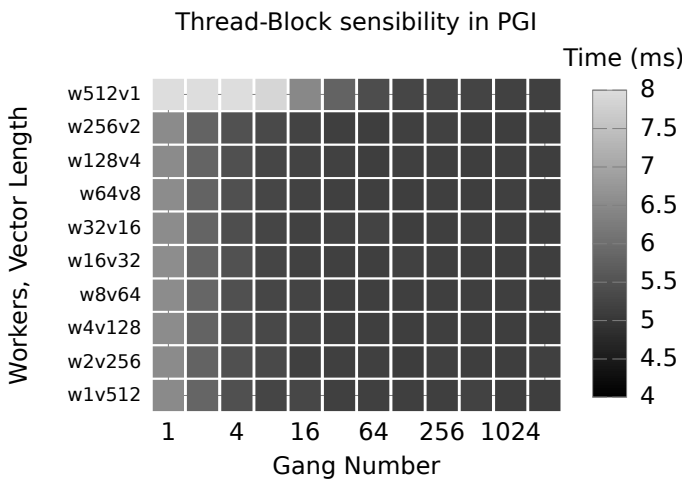

Figure 4: Effects of the Gang Number, Worker and Vector Length in the measured execution time using PGI compiler. Execution time is in milliseconds (lower is better). Darker is lower. Brighter is higher.

 ISBN: 978-84-617-8694-7

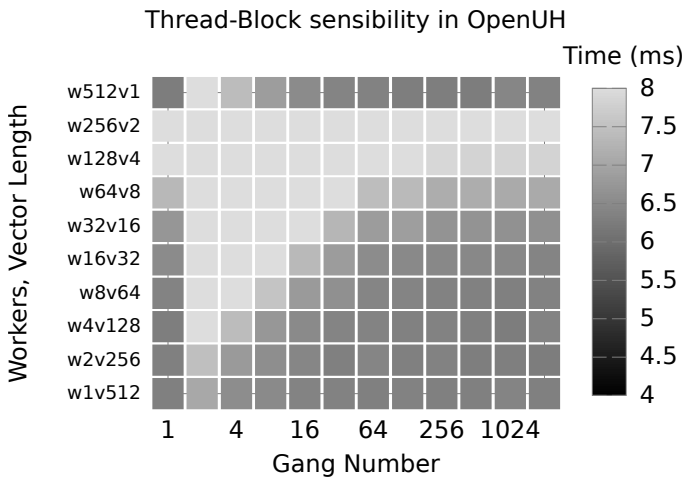

Figure 5: Effects of the Gang Number, Worker and Vector Length in the measured execution time using OpenUH compiler. Execution time is in milliseconds (lower is better). Darker is lower. Brighter is higher.

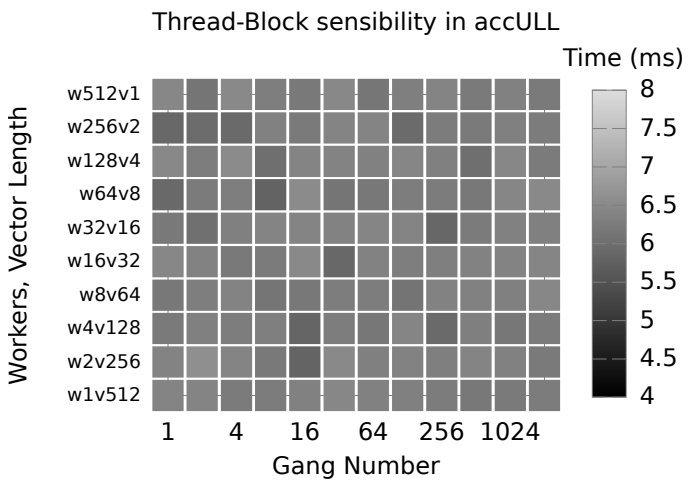

Figure 6: Effects of the Gang Number, Worker and Vector Length in the measured execution time using accULL compiler. Execution time is in milliseconds (lower is better). Darker is lower. Brighter is higher.

 ISBN: 978-84-617-8694-7

very different behaviours while unifying the basic concepts of automatic parallelization for both GPUs and Xeon Phi coprocessors.

Due to the standard leaving freedom for the implementation of the different levels of parallelism to the different compilers, the maturity of the latter directly affects the performance of the OpenACC-generated code. Thus we find that the PGI compiler, being the more mature of the analyzed compilers, generates code that resembles an optimized CUDA implementation. OpenUH shows an implementation that takes the defined levels of parallelism into consideration, but with room for a performance improvement. On the other hand, accULL seems to avoid hand-made changes to the levels of parallelism, choosing always the same configuration.

Although compiler implementations are not very mature yet, the simplicity of our microbenchmark allows us to see the effects of the variations in the different clauses: Gang number, Worker number, Vector length. Our results remark that the performance boost obtained by the tested OpenACC compilers in GPUs is dependant on the implementation of these clauses. However, since the mission of these clauses is to unify different concepts among GPUs and Xeon Phi accelerators, we argue this is complex task for the compiler implementations.

Acknowledgements

This research has been partially supported by MICINN (Spain) and ERDF program of the European Union: HomProg-HetSys project (TIN2014-58876-P), CAPAP-H5 network (TIN2014-53522-REDT), and COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] OpenACC-standard.org, "About OpenACC."

[2] OpenACC-Standard.org, "The OpenACC application programming interface version 2.5," oct 2015.

[3] X. Tian, R. Xu, Y. Yan, Z. Yun, S. Chandrasekaran, and B. Chapman, "Compiling a high-level directive-based programming model for GPGPUs," in *Languages and Compilers for Parallel Computing*, pp. 105–120, Springer, 2014.

[4] R. Reyes, I. López-Rodríguez, J. J. Fumero, and F. de Sande, "accULL: an OpenACC implementation with CUDA and OpenCL support," in *Euro-Par 2012 Parallel Processing*, pp. 871–882, Springer, 2012.

 ISBN: 978-84-617-8694-7

[5] H. Ortega-Arranz, Y. Torres, A. Gonzalez-Escribano, and D. R. Llanos, "Optimizing an apsp implementation for nvidia gpus using kernel characterization criteria," *The Journal of Supercomputing*, vol. 70, no. 2, pp. 786–798, 2014.

[6] C. Wang, R. Xu, S. Chandrasekaran, B. Chapman, and O. Hernandez, "A validation testsuite for OpenACC 1.0," in *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pp. 1407–1416, May 2014.

[7] PGI, "Pgi accelerator compilers with OpenACC directives." `https://www.pgroup.com/resources/accel.htm`, nov 2015.

[8] U. de La Laguna, "YaCF." `https://bitbucket.org/ruyman/llcomp`, nov 2015.

[9] U. de La Laguna, "Frangollo." `https://bitbucket.org/ruyman/frangollo`, nov 2015.

[10] U. of Houston, "Open-source UH compiler." `http://web.cs.uh.edu/~openuh/download/`, nov 2015.

[11] U. de La Laguna, "accULL." `http://cap.pcg.ull.es/es/accULL`, nov 2015.

 ISBN: 978-84-617-8694-7