Analysis of OpenACC Performance Using Different Block Geometries

Daniel Barba Arturo Gonzalez-Escribano Diego R. Llanos

Universidad de Valladolid

Grupo Trasgo
Universidad de Valladolid

International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE), 4-8 July 2017, Cadiz, Spain

Introduction

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.

CUDA kernels are usually heavily affected by the thread block geometry of choice[1] and, in OpenACC, this is defined by the *Gang* number, *Worker* number, and *Vector Length*. In this study we analyze how different choices of these values affect the performance of OpenACC generated code.

Benchmark Proposal

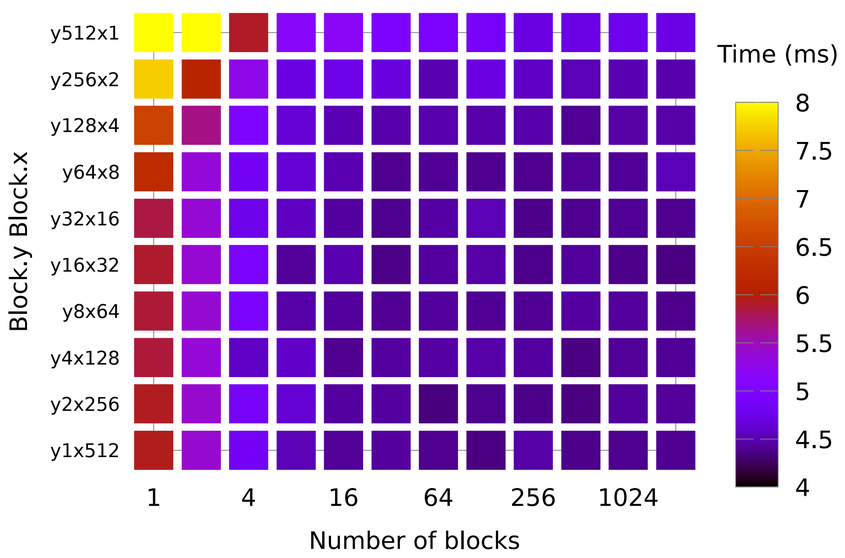
We use a very simple matrix addition implemented both in CUDA and OpenACC. Our decision is driven by the fact that the problem is embarrassingly parallel, memory accesses are perfectly coalesced, and the computational load per global memory access is low (memory-bound application).

The standard only defines the different levels of parallelism, but it is up to each implementation to decide how these levels are exploited in the actual architecture. 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. In order to obtain comparable results in a relative performance study, some details should be taken into account:

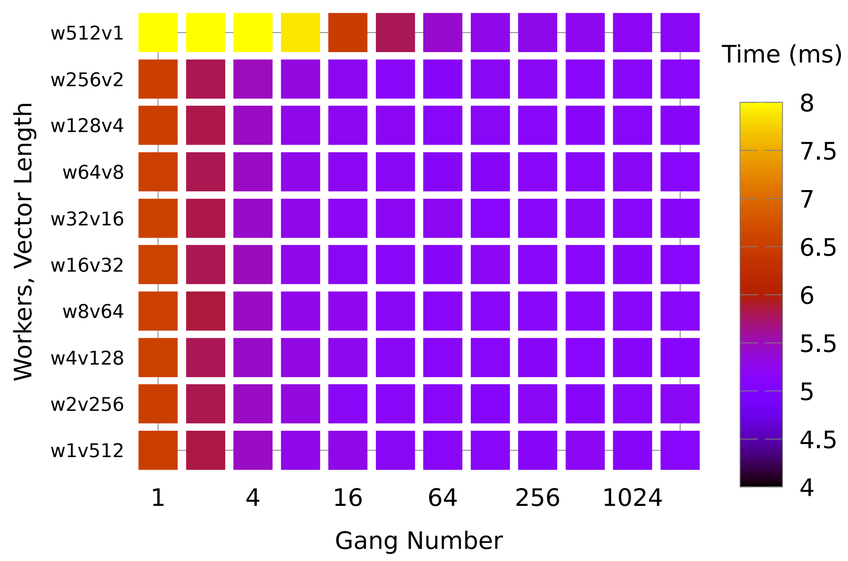
- CUDA version needs to use elastic kernels.
- We use a one-dimensional grid, because grid dimensions depend on the collapse clause.
- We evaluate a number of blocks in the grid ranging from one to 2048 (using only powers of two).

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.

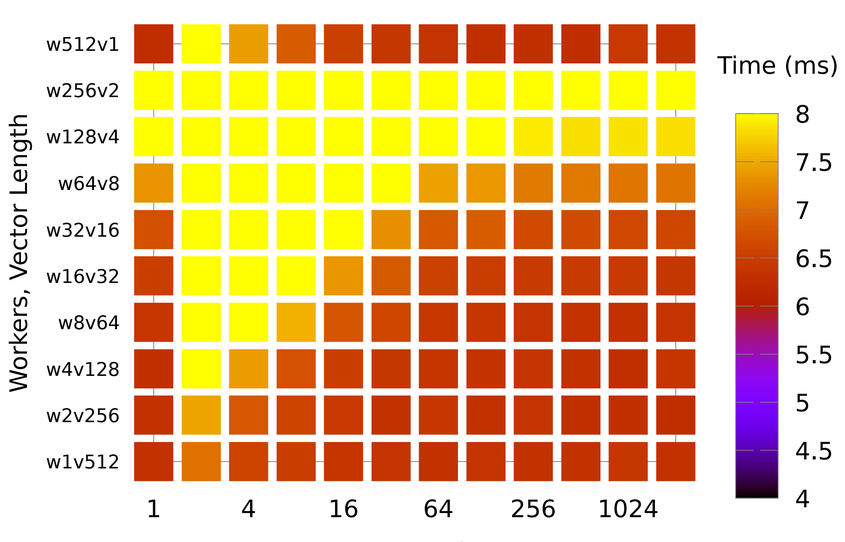
CUDA NVCC Results



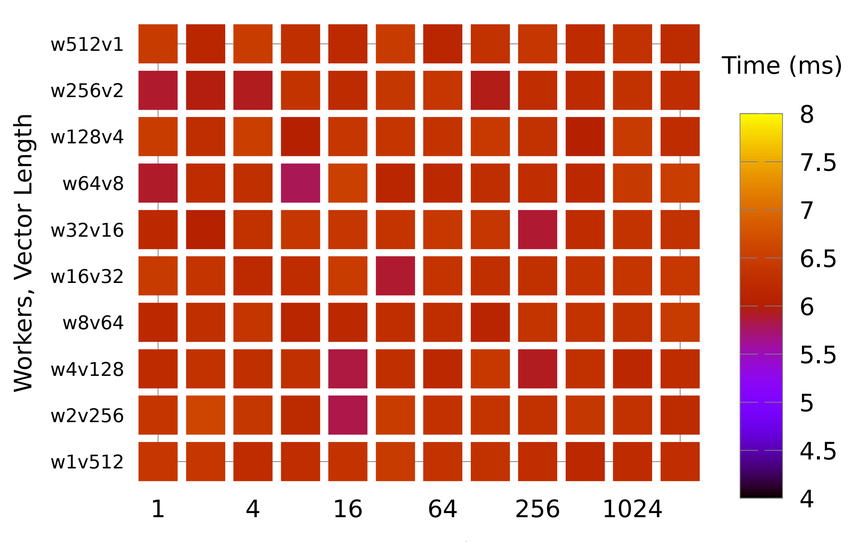
PGI[2] Results



OpenUH[3] Results



accULL[4] Results



The figures show matrices of execution times depending on two variables: Gang number and Worker number - Vector Length combinations (which are tied together in order to obtain 512 thread blocks). Execution time is in milliseconds and it is a LB metric (lower is better). Darker colours means lower execution times while brighter means higher.

Conclusions

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 an important role, but we expected to see an additional improvement in performance when the X dimension was at least 16. We suspect this is due to the behaviour of the cache when elastic kernels are used.

For PGI, the only clause affecting the performance is the gangs number, unless a vector length of one is specified. Overall, PGI's behaviour is the closest one to CUDA for this example.

When using OpenUH all three parameters affect the performance. This means that the parameters are actually being used to map the computation to the architecture resources, unless all three parameters are set to one. In this case, OpenUH seems to choose what it considers adequate.

Finally, the results obtained by using accULL 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.

References

[1] Ortega-Arranz, H., Torres, Y., Gonzalez-Escribano, A., & Llanos, D. R. (2014). Optimizing an APSP implementation for NVIDIA GPUs using kernel characterization criteria. The Journal of Supercomputing, 70(2), 786-798.

[2] PGI, "PGI Accelerator Compilers with OpenACC Directives", available online at https://www.pgroup.com/resources/accel.htm on april 2017.

[3] Tian, X., Xu, R., & Chapman, B. (2014). OpenUH: open source OpenACC compiler. GTC2014, HPCTools Group Computer Science Department University of Houston.

[4] Reyes, R., López-Rodríguez, I., Fumero, J. J., & de Sande, F. (2012, August). accULL: an OpenACC implementation with CUDA and OpenCL support. In European Conference on Parallel Processing (pp. 871-882). Springer Berlin Heidelberg.

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).