TORMENT OpenACC2016: A benchmarking tool for OpenACC compilers

Daniel Barba, Arturo Gonzalez-Escribano, Diego R. Llanos
Dpto. de Informática
Universidad de Valladolid
Valladolid, Spain
daniel@infor.uva.es, arturo@infor.uva.es, diego@infor.uva.es

***Abstract*—OpenACC is a parallel programming model for hardware accelerators, such as GPUs or Xeon Phi, which has been in development for several years by now. During this time, different compilers have appeared, both commercial and open source, which are still on development stage. Due to the fact that both the OpenACC standard and its implementations are relatively recent, we propose a benchmark suite specifically designed to check the performance of the OpenACC features in the code generated by different compilers on different architectures. Our benchmark suite is named *TORMENT OpenACC2016*. Along with this tool we have developed an adequate metric for the comparison of performance among different machine-compiler pairs which we have named *TORMENT_ACC2016 Score*. The version 1 of *TORMENT OpenACC2016* presented in this paper, contains six benchmarks, and is available online.**

***Keywords*-OpenACC, compilers, benchmarking.**

I. Introduction

OpenACC is an open standard which defines a number of compilation directives or pragmas for parallel code fragments execution using hardware accelerators such as GPUs and Xeon Phi. Its objective is to ease parallelization of sequential code using this kind of accelerators and reducing the required time both for learning and coding [1]. The OpenACC specification is currently on its 2.5 version [2].

At the time of writing this paper, there are several benchmarks for OpenACC. For example, the EPCC OpenACC Benchmark Suite [3] has been developed by the Edinburgh Parallel Computing Centre, and consists on both a number of microbenchmarks intended to check the compliance of the standard and measure the overhead of different pragmas implementation. It also measures the general performance of OpenACC generated code for each benchmark, but several compilers lack full support for all the features of this tool yet. This implies that it is not straightforward to do comparative performance analysis of different compilers. On the other hand, there are other benchmark suites not originally designed for OpenACC. For example, Rodinia [4] for OpenACC is a port of the original benchmarks from the suite of the same name [5], developed by *Pathscale*. As it is not specifically designed for OpenACC, most of the benchmarks included in this suite present many compilation problems with the state-of-the-art OpenACC compilers, and they do not cover all of the OpenACC features in a systematic way.

It would be interesting for the community to have a benchmark suite that covers the main features of the OpenACC standard and easily allows to compare the improvements in code generation for different architectures in terms of performance. The contribution of this work includes:

- A benchmark suite designed for OpenACC, that covers the main features of the standard based on programming patterns of different kinds of real applications.
- A proposal for a metric to provide a relative score which allows for the comparison of the performance obtained by the implementation of different compilers on different platforms. The design of this metric allows to classify the results in a global ranking. We named it *TORMENT_ACC2016 Score*.
- A compilation-execution wrapper, along with a systematic way to include in the source code conditional compilation statements to adapt the OpenACC syntax and details to the specific particularities of each compiler. This wrapper allows to easily adapt the benchmarks for different OpenACC compilers and platforms. It also automatizes the generation of a final report including the metric measure and other relevant data for the comparisons.

We present *TORMENT OpenACC2016*. It is a tool that includes the benchmarks and the wrapper that automatizes the compilation and execution. The generated reports are presented in a standard and easy readable way, including information about the execution platform, the compilers, statistics of execution times, and the final values of the metric for each case. The report also offers statistics of execution times for sequential and native CUDA versions of the benchmarks for further reference. The tool, in its current state, has been validated for three different compilers in two different GPU platforms. The support of the used compilers for the generation of Xeon Phi code is not yet mature enough to extend our validation for these kind of device.

The rest of the paper is structured as follows: Section II describes with more detail previous benchmarking tools. Section III discusses the maturity of OpenACC compilers. Section IV describes *TORMENT OpenACC2016*, including its goals, structure of the tool, implemented benchmarks, the proposed metric, and comments the report obtained in

the experimental validation. Finally, section V concludes the paper.

II. Existing Tools

As it was stated in the introduction, at the time of writing this paper there are previous performance evaluation tools for OpenACC. In this section we will describe more thoroughly examples of these tools and the problems detected to use them for the goals of this work.

A. EPCC OpenACC Benchmark Suite

This benchmark suite developed by the *EPCC* has been designed specifically for OpenACC. It is composed of three different parts, named *Level 0*, *Level 1* y *Application Level*.

Level 0 contains several microbenchmarks whose goal is measuring the overhead caused by the implementation of the different pragmas. These microbenchmarks offer a result calculated as the difference between the execution times of two different versions of the use of the OpenACC pragmas. For #pragma acc data directive it simply measures the required time for data transfers.

Level 1 contains a set of BLAS type benchmarks based on *Polybench* and *Polybench/GPU* [6]. The results offered by these benchmarks are the execution time of the different code fragments. This means they are a good performance indicator of the OpenACC generated code. However, in the current state of development of the different compilers some of the benchmarks use unsupported pragmas. Due to this, these benchmarks present compilation or runtime problems with a number of compilers.

Application Level contains three benchmarks of a bigger complexity than those from the previous section. These benchmarks also offer as their results the execution time of the generated code.

The *EPCC OpenACC Benchmark Suite* is well planned and designed. However, the results obtained from the microbenchmarks are not adequate enough for a relative performance analysis. The other benchmarks from *Level 1* and *Application Level* could be adequate. However, since they are not specifically designed to test OpenACC features supported by all of the available compilers, they present problems. In some cases they couldn't be compiled with some compilers, and sometimes they produced runtime errors.

B. Rodinia for OpenACC

Pathscale Inc. has developed an OpenACC version [4] of the Rodinia benchmark suite [5], [7]. The current version at the time of writing this paper has been published on GitHub (April, 24th 2014). The suite is composed of benchmarks that return as their result the execution time. This means that they are a good starting point for a performance analysis. The main issues are the low maturity level of the OpenACC use in the code, and the lack of compatibility with the available compilers. These issues make it impossible to establish a fair performance comparison.

III. OpenACC Compilers

There is a number of choices among compilers with support for OpenACC, both commercial and free. In this work we focus on those compilers which are available for free or with academic or trial licenses. In general, the OpenACC compilers have currently very limited support for any platform other than GPUs. The compilers considered are:

A. PGI Compiler

The PGI compiler[8], developed by *The Portland Group* and Nvidia, is, at the time of writing, the compiler with the highest maturity level. In our studies, the PGI compiler has shown to be the most stable and the highest maturity level among the used compilers. It has a proper implementation of most of the OpenACC specification.

B. OpenUH

OpenUH compiler, developed by the University of Houston, is an open source project. According to our tests and compared to the PGI compiler, its maturity and robustness are lower and it lacks several functionalities.

C. accULL

The accULL compiler [9] has been developed by Universidad de La Laguna, Spain, and it is an open source compiler like OpenUH. Our tests show that compared to the previous compilers, this is the one with a lower robustness, leading to several issues during source to source translation, not supporting several C characteristics, like function pointers. Several OpenACC functionalities are also lacking, like several types of reduction.

IV. TORMENT OpenACC2016

Our proposal is implemented as a tool called *TORMENT OpenACC2016*, an acronym for Trasgo perfORMance and EvaluatioN Tool for OpenACC. The project was born to give response to the need of performance analysis and comparison for OpenACC code generated by the various compilers implementing the standard.

A. Goals

The main goal of *TORMENT OpenACC2016* is to allow a performance analysis of OpenACC code generated by different compilers in an easy way, generating a result summary that can be easily analyzed and which offers the *TORMENT_ACC2016 Score* metrics. This metric is useful to compare machine-compiler pairs in terms of performance.

Our proposal intends to offer a tool specifically prepared for OpenACC, developed taking into account the early development stage of the available compilers. Thus, we try to ensure that compilation or runtime problems are avoided for

the compilers considered, unlike in the previous benchmark suites available. *TORMENT OpenACC2016* is prepared to be compiled and executed with minimum intervention from the user. The scripts accompanying the suite get information from the benchmarking process and finally offer a HTML report with the most relevant data.

Moreover, *TORMENT OpenACC2016* makes use of the GCC and NVCC compilers in order to obtain data from sequential and CUDA-based code executions. With this data, our proposal can offer to the user information about the *speedup* compared with sequential and CUDA code executed in the same machine.

B. Structure of our tool

TORMENT OpenACC2016 is composed of a number of shell scripts which are in charge of the compilation, execution and results gathering processes, removing the burden from the user. Each benchmark has been chosen to represent a relevant class of problems, and has been implemented to be as simple as possible, in order to avoid compilation issues.

The benchmarks are launched ten times, with an extra invocation at the beginning whose results are discarded. The values that correspond to the best of the executions and the arithmetic mean of all of them are recorded to later calculate the proposed metric. These values are named *peak* and *average* respectively.

C. Features Coverage

In order to make a selection of benchmarks for our suite we have analyzed the development status of the different compilers. We have studied their capabilities and their completeness of the OpenACC standard. Our conclussion is that the existing benchmarks, based on Polybench and Rodinia, often require pragmas that are not implemented yet on one of the compilers at least. To achieve full compatibility among the different compilers we need to use only the set of directives available for all of them at the time of developing this version of *TORMENT OpenACC*.

We list below a number of GPU characteristics that we argue are important to be tested for the current status of compilers' maturity:

- Use of shared memory
- Intensive and non-balanced computation
- Computation without memory access
- Data transfers
- Non-perfectly coalescent memory access

These are important characteristics that play a pivotal role in the resulting code of manually written CUDA kernels. The code generated by the OpenACC compilers will also be affected by these characteristics, thus the performance of these generated codes could be measured and compared.

As most of the well-known benchmarks require the use of non-supported directives for one compiler at least, we have decided to follow a bottom-up approach for developing a working suite of benchmarks, trying to cover the characteristics listed above. This decision leaves us with a set of very simple benchmarks, useful for a comparative analysis of the performance of OpenACC generated code using different compilers, but not for a thorough analysis of completeness, robustness and performance for a single compiler. Our work aims to obtain a tool capable of generating a comparative analysis report for a wide number of compilers.

As the support for OpenACC directives improves, this design decision should also be reviewed and adapted in order to use benchmarks of a higher complexity as the ones provided in suites like Polybench (used in the EPCC OpenACC benchmark suite), Rodinia, and the CUDA Toolkit.

D. Implemented Benchmarks

Version 1 of *TORMENT OpenACC2016*, contains a set of six benchmarks. These benchmarks try to cover the different characteristics enumerated in the previous section, as it is shown in Table I.

1) T_MonteCarloPi: This is an embarrassingly parallel, without dependencies, perfectly regular, well balanced with a static partition, and computational bound example. This benchmark consists on an approximation of Pi using the Monte Carlo method. This method is based on the random generation of coordinates in a unit square. For each coordinate, it is checked whether or not that coordinate is located inside a quarter circle and, if that condition is satisfied, then the coordinate is accumulated. Finally, the following formula is applied:

$$\pi \approx \frac{4*P}{T}$$

Where P is the number of coordinates inside the quarter circle and T is the total of generated coordinates.

T_MonteCarloPi is a benchmark that contains no memory transfers and a very simple computation. However, it can be optimized if each thread computes several coordinates and if the *shared memory* in the thread blocks is used in order to avoid global memory access.

The function used for random number generation can be problematic. OpenACC code should use functions that can be offloaded to the GPU and at the same time can be used on the host machine. Thus, we cannot use native CUDA functions like the ones included in the curand library. Our solution has been to replicate the implementation of the srand function in the standard C library.

2) T_StringMatch: This example is coalesced with low memory bandwidth and irregular data-dependant loads per thread. The *T_StringMatch* benchmark is a string matching program. This algorithm is interesting because it combines data transfer with the need of efficient use of the memory, specially the *shared memory*. It consists on the search of the first occurrence of a small string in a larger one, using a *naive* algorithm. In this benchmark, the large string has a

	Shared Memory	Intensive Computation	Computation without memory access	Data Transfer	Non-perfectly coalescent Memory Access
T_MonteCarloPi	X		X		
T_StringMatch	X			X	
T_3DStencil				X	
T_Mandelbrot		X			
T_MatrixMult	X			X	
T_ReverseArray	X			X	X

Table I
GPU CHARACTERISTICS TESTED BY THE BENCHMARKS

size of 10 million characters, that is, approximately 10MB. We search four small strings of 1000 characters.

3) T_3DStencil: This is a regular, well balanced example with dependencies that lead to iterative neighbour synchronization, and low load per thread on each iteration. The *T_3DStencil* benchmark consists on a 6 point 3D stencil. Each point of a 3D matrix is updated with the arithmetic mean of its neighbours. This benchmark is interesting because, traditionally, stencil programs can be very efficient when executed on a GPU. It includes memory transfers only at the beginning and end of the computation.

4) T_Mandelbrot: This example is embarrassingly parallel without dependencies with non-balanced load. The *T_Mandelbrot* implements the most famous of the fractal sets. The implemented algorithm is the escape-time algorithm. This is an embarrassingly parallel problem and there are no dependencies among the computations. Therefore, there is no need to use the *shared memory*. Global memory access is also negligible. On the other hand, the computation is very intensive, depending on the maximum number of iterations. This benchmark should scale very well on GPU.

5) T_MatrixMult: This example has perfectly regular loads with coalesced memory access if shared memory is properly used. Matrix multiplication is widely used in many domains. It has a large margin for optimizations. When using a GPU, the most important optimization is the correct use of the *shared memory*, which requires a correct thread block geometry. As this benchmark has three nested loops, it presents certain complexity for the OpenACC compilers. They need to make choices at the different levels of parallelism, including also the geometry of the thread blocks.

6) T_ReverseArray: This last example has a regular load, but a direct implementation for non-aligned array sizes derives in coalesced reads but non-perfectly coalesced writes. The last of the benchmarks implemented in *TORMENT OpenACC2016* consists on a very simple operation; reversing an integer array. Although this seems to be a trivial operation, running it on a GPU presents a relevant problem. Reversing an array on a GPU can be very inefficient because of non-coalescent global memory access. The correct solution involves using the *shared memory* as an intermediate buffer where a partial reverse is done. This allows for coalescent global memory access.

E. Proposed Metric

For the selection of the metric named *TORMENT_ACC2016 Score*, we use the same methodology as SPEC [10] (*Standard Performance Evaluation Corporation*). It is a widely known reference in terms of benchmarking and performance analysis. One of its strengths is to acknowledge that benchmarks get older as time passes and, in consequence, they must be updated.

SPEC uses the following methodology. First, the program returns its execution time. Then the *SPECratio* is calculated, which consists on the ratio obtained from dividing the reference execution time (supplied by SPEC) and the execution time obtained by the benchmark. Finally, the geometric mean of all the SPECratios is calculated [11].

Our proposal follows a similar idea but with several changes. First, the execution of the benchmarks contained in *TORMENT OpenACC2016* return three values. *Peak Time* is the best execution time obtained, measured in seconds. *Average Time* is the arithmetic mean of all the executions of the same benchmark, also in seconds. Finally, the *Standard Deviation* of the set of times obtained gives an idea of the variability among the several executions of the benchmarks. With the *peak* and *average* times, a ratio is calculated dividing the reference times provided by the tool, which are the execution times of the sequential version of the benchmarks in a reference machine. The characteristics of this reference machine are shown in *https://torment.infor.uva.es*. Finally, it computes the harmonic mean of all the ratios, both for *peak time* as *average time*. These values are the *TORMENT_ACC Peak Score* and *TORMENT_ACC Average Score*.

The main difference between *TORMENT OpenACC2016* and SPEC metrics, apart from the time measurement methodology, consists on the use of the harmonic mean instead of the geometric mean. For the goals of *TORMENT OpenACC2016*, the harmonic mean is more adequate than the geometric mean. First, even if the geometric mean always produces a consistent ordering, it's not necessarily the correct ordering [11]. This is because this mean is not inversely proportional to the execution time. In contrast, the harmonic mean is inversely proportional to the execution time, which makes it an adequate mean for ratios (see e.g. [12]). *TORMENT_ACC2016 Score* is a HB (Higher is Better) metric.

An example of usage can be found on *https://torment.infor.uva.es*. The TORMENT Report Example link shows a report generated for one of our machines. The first part of the report consists on system information. After that section, the result tables for sequential and CUDA code are found. These results are merely informative, and they are not used in the *TORMENT_ACC2016 Score*. However, these results allow the user to compare the performance of the OpenACC code with sequential and CUDA versions executed in the same machine. After the sequential and CUDA results, the OpenACC results are shown for each compiler used. These results consist on the tables, with the execution time statistics, the *TORMENT_ACC2016 Score*, and the speedup values using as reference the sequential and CUDA executions.

In the Report Example available online, the PGI compiler obtains the best results, obtaining a 64% of the performance achieved with CUDA implementations. OpenUH gets a second position, with a 41% of the performance achieved with the optimized CUDA implementations. Finally, the accULL compiler get the third position, with 16% of the performance achieved with CUDA.

The *TORMENT_ACC2016 Score* also allows for a comparison among compiler-machine pairs. For example, the Hydra System in the Report Example available online has Titan Black GPUs, and we also ran the benchmark suit in a laptop which has a Nvidia GTX 850M. The *TORMENT_ACC2016 Peak Score* in Hydra is 28.69, while in the laptop is 8.45. On the other hand the accULL compiler obtains 7.26 in Hydra and 4.98 in the laptop. This indicates the important differences of performance introduced by the code generation and optimization techniques used by both compilers.

V. Conclusions and Future Work

TORMENT OpenACC2016 is a performance analysis and comparison tool for OpenACC generated code, taking into consideration the maturity level of the support of OpenACC in different compilers. *TORMENT OpenACC2016* contains a suite of benchmarks specifically designed for OpenACC and maintaining the maximum portability.

The results offered by *TORMENT OpenACC2016* include the so called *TORMENT_ACC2016 Score*, designed for the comparison of machine-compiler pairs. Moreover, it offers a result report of the benchmarks, including a table with execution times statistics (both for peak and average values) and the standard deviation, and also including execution results from sequential and CUDA code generated by GCC and NVCC compilers. This offers the user a comparative analysis of the OpenACC generated code versus the sequential and CUDA versions of the benchmarks.

Future work consists on the inclusion of new benchmarks, specially when the supported compilers become more mature, including a richer structure with different levels (such as BLAS and real world applications) in addition to the existing synthetic applications. Moreover, an important part of our future work consists on maintaining compatibility among the supported compilers and new additions. We are in the process of adding support for GCC, Omni (University of Tsukuba), OpenARC (Oak Ridge National Laboratory), and RoseACC (University of Delaware).

Acknowledgments

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

References

[1] OpenACC-standard.org. About OpenACC. [Online]. Available: http://www.openacc.org/about-openacc

[2] ——. (2015, oct) OpenACC 2.5 draft for public comment. [Online]. Available: http://www.openacc.org/content/openacc-25-draft-public-comment

[3] EPCC. (2013, sep) Epcc OpenACC benchmark suite. https://github.com/EPCCed/epcc-openacc-benchmarks.

[4] Pathscale. (2014, apr) Rodinia benchmark suite 2.1 with OpenACC port. https://github.com/pathscale/rodinia.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. (IISWC), 2009 IEEE International Symposium on*. IEEE, 2009, pp. 44–54.

[6] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite," *URL: http://www. cs. ucla. edu/~ pouchet/software/polybench/[cited July,]*, 2012.

[7] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–11.

[8] PGI. (2015, nov) Pgi accelerator compilers with OpenACC directives. https://www.pgroup.com/resources/accel.htm.

[9] 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*. Springer, 2012, pp. 871–882.

[10] K. M. Dixit, "The spec benchmarks," *Parallel computing*, vol. 17, no. 10, pp. 1195–1209, 1991.

[11] D. J. Lilja, *Measuring computer performance: a practitioner's guide*. Cambridge University Press, 2005.

[12] J. R. Mashey, "War of the benchmark means: time for a truce," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 4, pp. 1–14, 2004.