11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
July 20–25, 2014, Barcelona, Spain

PERFORMANCE CONSIDERATIONS WHEN USING INTEL® XEON PHI™ COPROCESSORS FOR UNSTEADY DISCRETE ADJOINT CALCULATIONS

Jan C. Hückelheim[1] **and Jens-Dominik Müller**[2]

[1] Queen Mary, University of London,
Mile End Road, E1 4NS, London, UK
j.c.hueckelheim@qmul.ac.uk

[2] Queen Mary, University of London,
Mile End Road, E1 4NS, London, UK
j.mueller@qmul.ac.uk

Key words: *Adjoint, Shape Optimisation, Checkpointing, Intel XeonPhi, Roofline model*

The discrete adjoint method is an established way of computing gradients of a function with respect to its input variables. It is used e.g. in shape optimisation, where a cost function such as lift or drag is optimised by tuning the design. The complexity of the adjoint method is independent of the number of design variables, making it feasible even for industrial-scale applications [1]. In spite of this, the computational cost and memory requirements for gradient calculations are a challenge, especially in unsteady flow optimisation. The use of parallel high performance computing is essential, and latest computing technology can help making this method more affordable for the industry.

Vector-coprocessors like GPUs or the Intel® Xeon Phi™ offer more computational power per power consumption than traditional CPUs and are used increasingly in both academia and industry for cost-effective large scale computations. With the latest revision, the OpenMP 4.0 standard includes directives that allow offloading program parts to a coprocessor, i.e. transfer data to a GPU or Xeon Phi™, run parts of the program there, and copy back the results to the host CPU. This requires fewer code changes compared to approaches like writing a CUDA kernel, while retaining most of the added performance offered by the coprocessor[2].

This project has received funding from the European Unions Seventh Framework Programme for research, technological development and demonstration under grant agreement no [317006]

In our project, we use the new OpenMP directives to parallelise an incompressible unstructured flow solver. The parallel code is reverse-differentiated using the automatic differentiation tool Tapenade[3]. We study the impact of this process on the vector and cache efficiency of the coprocessor. Furthermore, the memory limitations on the coprocessor require more frequent checkpointing with additional data transfer to the host device.

While not all of this can be done automatically by Tapenade, the additional checkpoint transfers in forward and reverse sweeps being a particular problem similar to the additional communication needed in source-transformed MPI-parallel code, we will use a hand-optimised code to answer a few more general questions.

The computations that are required for unstructured meshes and sparse matrices are usually memory bound, memory accesses are not aligned and the vectorisation that coprocessors rely on for their performance can hardly be used. Other groups have faced performance issues with vector coprocessors even on structured meshes [4] and experienced a speedup factor of only around 30%. Unstructured codes have an even lower computational intensity and will thus most likely experience even lower speedups.

We will conduct a detailed performance analysis of our code using the roofline model. This model allows us to compare the performance that our implementation achieves with a theoretical limit that can be reached on a particular machine. We will show that the speedup is not always promising even in the best case. This results in general results about the usefulness of vector coprocessors in our application, and not only a report of our experience so far.

Finally, we will point out alternatives to make use of the coprocessor, such as data checkpoint compression on the Xeon Phi while the host CPU is working on the main CFD computations. In this case, the coprocessor could be easily used *in addition* to the host CPU, resulting in a more efficient use of the available resources.

REFERENCES

[1] M. B. Giles and N. A. Pierce. *Adjoint equations in CFD: duality, boundary conditions and solution behaviour*. Oxford University Computing Laboratory, 1997.

[2] T. Cramer, D. Schmidl, M. Klemm and D. an Mey. *Programming on Intel® Xeon Phi™ Coprocessors: An Early Performance Comparison*. RWTH Aachen University, 2012.

[3] L. Hascoet and V. Pascual. *The Tapenade Automatic Differentiation Tool: Principles, Model, and Specification*. ACM Transactions On Mathematical Software, 2013.

[4] G. Skinner, M. A. Heroux. *Running MiniFE on Intel® Xeon Phi™ Coprocessors* . http://software.intel.com/en-us/articles/running-minife-on-intel-xeon-phi-coprocessors