Einstein Toolkit Kernel Benchmarks on Xeon and Xeon Phi Processors [Draft]

Ian Hinder
Max-Planck-Institut für Gravitationsphysik (Albert-Einstein-Institut)
Am Mühlenberg 1, D-14476 Golm, Germany
ian.hinder@aei.mpg.de

June 22, 2015

Abstract

We present initial benchmark results for solving the vacuum Einstein equations using finite difference methods with the Einstein Toolkit on Xeon and Xeon Phi processors. This is a very superficial study of performance using a simple kernel benchmark, and could be supplemented by a more thorough study in the future. Without modifying the code, we find that a single 61-core Xeon Phi MIC can outperform a single 8-core Xeon CPU by a factor of 1.5, but caution that the benchmark uses an unrealistically large grid, that only a single CPU was utilised in a dual-socket configuration, and that these results are only preliminary.

Code The evolution code is McLachlan [2], a Cactus module in the Einstein Toolkit [5, 1]. It solves the Einstein equations using finite difference methods. The code is the ET_2015_06 release of the Einstein Toolkit, compiled with the Intel compiler version 15, using "-Ofast". Thread parallelism is obtained via OpenMP, and SIMD-parallelism is obtained via explicit vectorisation using intrinsics. The equation code is generated using the Kranc [3, 4] code generation package, which automatically replaces arithmetic operations with SIMD intrinsics.

Benchmarks This is a kernel benchmark operating on a grid of size $n_{\mathrm{gridpoints}} = 128 \times 128 \times 128$. It computes two iterations of the solution to the Einstein equations using a 3rd order Runge Kutta time integrator, requiring 3 substeps. This requires 6 evaluations of the right hand sides of the equations. 6th-order finite differencing is used, with upwind derivatives for the advection terms and centered differences for the other terms. The grid is surrounded by a boundary of width 8 points (only 4 are needed, but 8 are used here for SIMD alignment purposes). Each benchmark is run 3 times and the mean and standard deviation of the evolution times are computed. The reported measurements, t_{gp} are the times spent in the $\mathrm{CCTK\_EVOL}$ time bin, which is dominated by the computation of the right-hand-sides, divided by the number of evolved points and the number of right-hand-side evaluations.

$$t_{\mathrm{gp}} = \frac{t_{\mathrm{evol}}}{n_{\mathrm{substeps}} \times n_{\mathrm{iterations}} \times n_{\mathrm{gridpoints}}} \tag{1}$$

This gives the evolution time per right-hand-side grid point evaluation for the whole processor. We have not yet carefully computed the number of floating point operations for a single gridpoint evaluation for this benchmark, but from similar measurements in the past we expect it to be ~ 5000. Note that the 128^3 grid is unrealistically large, since with adaptive mesh refinement, the grids are typically smaller. We found significantly worse performance per grid point with smaller grids, so for this initial test, we chose a larger grid size.

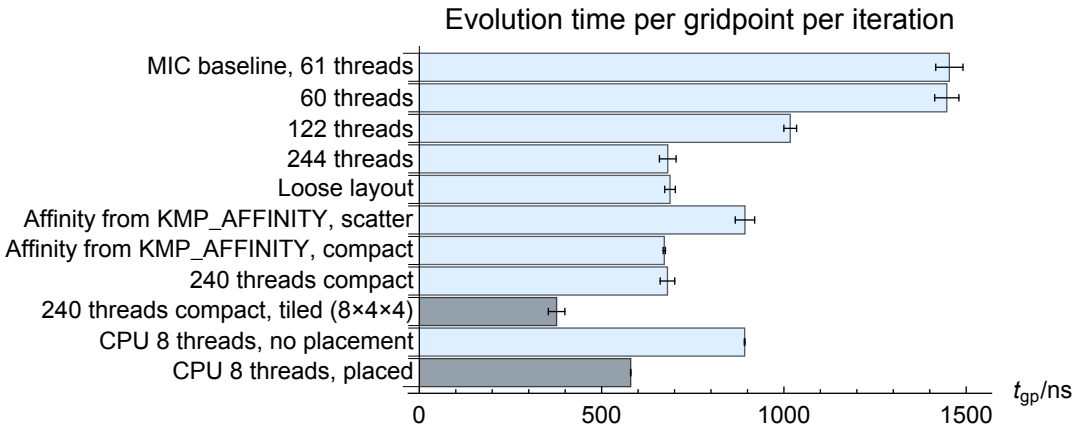

Figure 1: Evolution time per gridpoint (lower is better) for the kernel benchmark on MIC and CPU for various different threading and looping configurations. A more coherent set of benchmarks with more detailed descriptions may be presented in future.

Hardware All results are obtained on a single node of the "bphi" system at Zuse Institute Berlin. The node contains two Xeon E5-2690 "Sandy Bridge" CPUs, each with 8 cores, each core with 2 hardware threads. The node also contains one Intel Xeon Phi 7120P "Knights Corner" MIC coprocessor. The MIC results were obtained on the Xeon Phi, whereas the CPU results were obtained on one of the Xeon processors. Note that in a realistic situation, both CPUs would be utilised, and this will affect the performance due to the reduced memory bandwidth, but for this initial test, we used a single CPU for simplicity.

Results For this initial summary, the exact benchmarks used are not in any coherent framework. Fig. 1 shows the time to evolve one gridpoint for different numbers of threads and looping algorithms using either the MIC or the CPU. The Einstein Toolkit contains a module called "LoopControl" which implements tiling on 3-dimensional loops. This appears to give better results than parallelisation over entire $z = \mathrm{const}$ grid planes.

Conclusions By using 244 threads, with the domain split into tiles of size $8 \times 4 \times 4$ points, and OpenMP threads assigned one per tile as they become available, the MIC was able to outperform the single CPU by a factor of 1.5. The same tiling strategy was used on the CPU, as it has been found to give good performance there in the past. Since we have not yet optimised the code for the MIC architecture, we believe that further speed improvements will be possible, and that solving the Einstein equations on the MIC architecture should be feasible.

References

[1] Cactus computational toolkit. http://www.cactuscode.org.

[2] J. David Brown, Peter Diener, Olivier Sarbach, Erik Schnetter, and Manuel Tiglio. Turduckening black holes: an analytical and computational study. *Phys. Rev. D*, 79:044023, 2009.

[3] Sascha Husa, Ian Hinder, and Christiane Lechner. Kranc: A Mathematica application to generate numerical codes for tensorial evolution equations. *Comput. Phys. Commun.*, 174:983–1004, 2006.

[4] Sascha Husa, Ian Hinder, Christiane Lechner, Erik Schnetter, and Barry Wardell. Kranc: Automatic numerical code generation. http://kranccode.org.

[5] Frank Löffler, Joshua Faber, Eloisa Bentivegna, Tanja Bode, Peter Diener, et al. The Einstein Toolkit: A Community Computational Infrastructure for Relativistic Astrophysics. *Class.Quant.Grav.*, 29:115001, 2012.