OPTIMIZING THE HYBRID PARALLELIZATION OF BHAC

OLIVER PORTH*, HECTOR OLIVARES
ASTROLAB CONTACT: LUIGI IAPICHINO, SALVATORE CIELO
PROJECT PARTNER: FABIO BARUFFA (INTEL), ANUPAM KARMAKAR (LENOVO)

CONTENTS

1. CODE OVERVIEW AND INITIAL PERFORMANCE

`BHAC`[1] is an open-source general relativistic MHD code [1] based on the `MPI-AMRVAC` Toolkit v1.0 [2, 3]. `BHAC` offers a variety of numerical methods and fully adaptive block based (bi-, quad- or oct-) tree mesh refinement using staggered mesh constrained transport in any covariant coordinate system [4]. The high-resolution shock-capturing finite volume scheme uses second order Implicit/Explicit timesteppers combined with second or third order (PPM) spatial reconstruction. `BHAC` uses robust multi-dimensional non-linear solvers (Newton-Raphson and Newton-Krylov) for the conversion from the integrated conserved variables to the primitive variables with well tested backup strategies. This allows to perform simulations of challenging magnetically-dominated regimes as present in pulsar magnetospheres, black hole "funnels jets" and in relativistic reconnection.

`BHAC` is written in `Fortran90` and parallelized using `MPI`. The code has a rudimentary hybrid `MPI/OpenMP` parallelization which has not yet been used in production. Typical use cases so far have utilized up to 8000 cores, e.g. on SuperMUC, Cartesius/Netherlands or Breniac at VSC/Belgium. However, we have observed in previous scaling runs that **the MPI overhead of pure MPI runs becomes prohibitive at** $\sim$ 10 000 **processes**. This is demonstrated in Figure 1 which shows the `BHAC` strong-scaling on SuperMUC-NG (48 cores per node) where the pure MPI run with 9600 processes in fact allocated 400 nodes, loaded with only half the processes to fit memory requirements. The preliminary exploration shown in Figure 1 shows the potential of the hybrid parallelization: already with four or eight threads per task, the code is able to run beyond 400 nodes (tested up to 1600 nodes/76800 cores) and scaling is efficient until 800 nodes. However, in its initial state, the hybrid implementation comes with a severe performance penalty, e.g. at eight threads per task, the runtime is twice the pure MPI case at the same node count.

2. GOALS FOR THE ASTROLAB PROJECT

To efficiently run large problems beyond 2000^3 cells as needed for example in the study of relativistic turbulence, it is necessary to go beyond 10 000 cores accessible with pure MPI parallelization. Goal of this AstroLab project is hence to understand and improve the performance in the hybrid implementation. Since this is the first investigation of its kind with the `MPI-AMRVAC` toolkit, we use a simple uniform grid setup and aim for a fairly complete characterization of the code infrastructure.

We set out to perform the following tasks:

- OpenMP correctness check (Intel® Inspector™)

[1] https://bhac.science

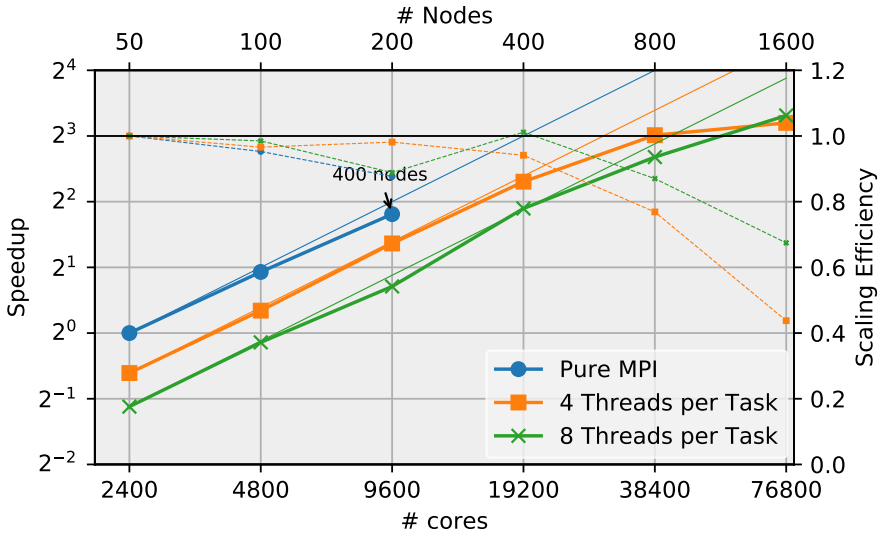

FIGURE 1. Initial scaling on SuperMUC-NG. At 9600 cores, the pure MPI case requires in reality 400 nodes in order not run out of memory, due to the MPI buffer overhead. Hybrid MPI/OpenMP alleviates this problem, running successfully up to 76800 cores. However, there is a severe performance penalty when increasing the number of OpenMP threads per task.

- Pure-MPI profiling and scaling (Intel APS™, Lenovo's lightweight and extended MPI tracing tool)
- Single-node profiling: node level performance, threading (OpenMP), vectorization, roofline model and memory access (Intel VTune™ and Advisor™)

3. ACHIEVEMENTS WITHIN THE ASTROLAB PROJECT

The different tasks require different problem sizes. We target single-node, 16 nodes and 800 nodes such that we obtain 65536 cells per core in each case, in order to keep the workload per node constant. Thus our problems range from $\sim 3 \times 10^6$ to over 2×10^9 computational cells.

3.1. **Overall characterization and correctness, single node.** This task was entirely executed on SuperMUC-NG at LRZ. As the OpenMP implementation was experimental, the correctness check we run yielded several race conditions which all could be fixed. At this point we used Intel Advisor for a general performance characterization: in particular, a roofline model showed a mixture of memory- and compute-bound kernels, overall the setup being memory bound. According to Intel Advisor, the vectorization efficiency with Skylake $AVX-512$ registers is good, but a 32% speedup is still theoretically possible. The hotspot analysis yielded no pressing need for action.

3.2. **MPI profiling of the 16 node test case.** Using the mpi_trace tool of Lenovo, the pure MPI scaling of the 16-node test case was evaluated in a range from 80 to 3200 MPI tasks, this time in a strong-scaling fashion. These runs were performed at the Lenovo Benchmark Cluster Stuttgart ("Lenox") on Lenovo SD530 platform with dual-socket Intel Xeon Gold 6148 based nodes. These nodes have roughly 80 GB available physical memory for the running applications. An Intel Omni-Path (OPA) high-performance fast interconnect provides up to 100G network bandwidth on these nodes. For these baseline measurements Intel fortran compiler 20.2 and Intel MPI 2019.8 were used. The results obtained are summarized in Figure 2 (top-left panel) which shows an excellent strong scaling of up to a factor of 19 from 160 to 3200 processes. The run at 80 processes (2 nodes) significantly underperformed, likely due to caching effects. This experiment shows that the pure MPI implementation using non-blocking `ISEND/IRECV` is quite efficient, requiring no urgent improvements. An analysis of the mpi_trace findings, Figure 2 (bottom-right panel), shows how the relative time spent on MPI communications alters very slowly with the number of MPI tasks, whereas the memory usage rapidly drops due to greater parallelism. This, in a way, explains the good scalability behaviour achieved with node scaling of this use-case.

As an aside, the same setup was also used to compute the frequency scaling and energy efficiency of the code. Using 16 nodes, the frequency was scaled from 1 GHz to 2.4GHz and the time-to-solution

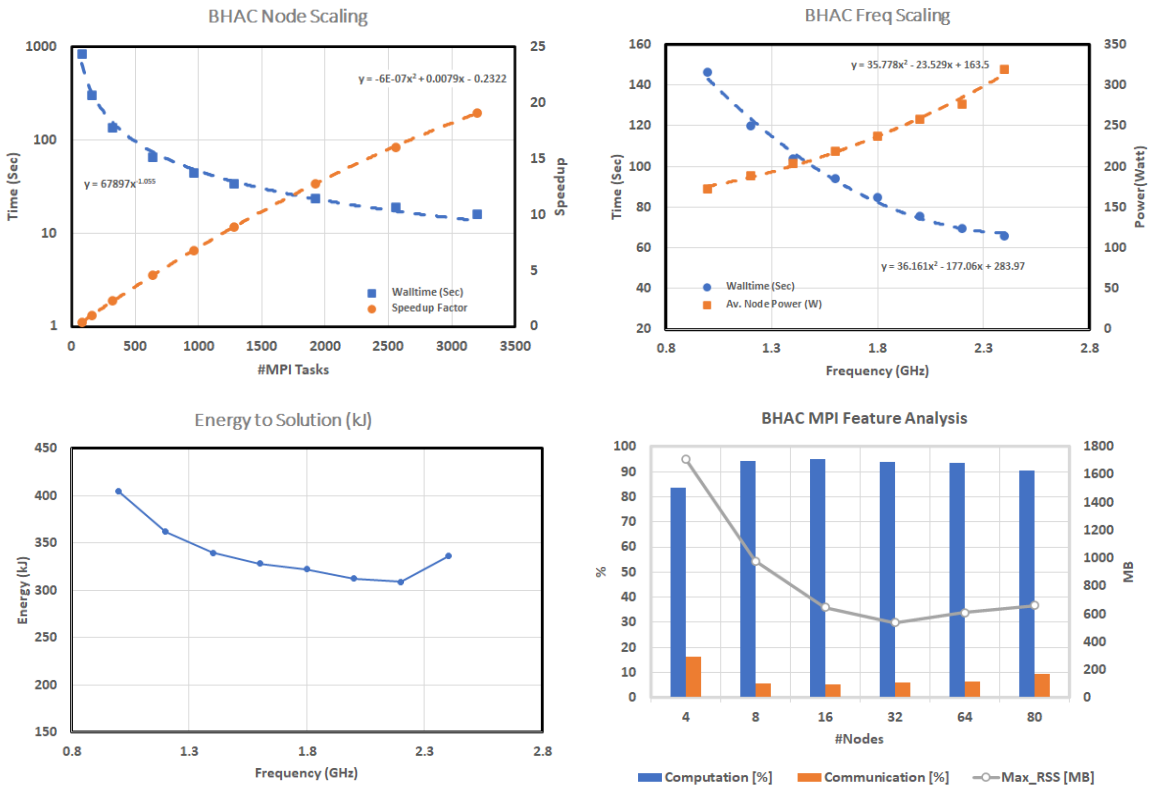

FIGURE 2. Pure MPI and frequency scaling with the 16 node test setup. We also show the average node power yielding a sweet spot at a clock frequency of 2.2GHz. These runs were performed at the Lenovo Benchmark Cluster in Stuttgart ("Lenox").

and average node power was recorded. The results are illustrated in Figure 2 (top-right panel). At an average of 277W per node, the minimum energy-to-solution ($\sim$ 309 kJ) was found for a clock speed of 2.2GHz, this is an optimum trade-off between performance and energy consumption. At a maximum clock speed, the energy-to-solution increases around 9%, however, it slightly lowers ($\sim$ 6%) the time-to-solution performance. Switching on the turbo frequency does not positively impact the overall performance, but only surges the energy to solution requirement [Figure 2 (below left panel)].

3.3. **Optimization, single node and 16 nodes.** Using APS and Advisor, the performance bottlenecks in the hybrid implementation were identified. The tools reported an OpenMP imbalance which could be solved by adding `dynamic` scheduling to the loops with significant workload. This yielded an average performance improvement of $\sim$ 5%. VTune's threading analysis identified the main bottleneck as large serial code blocks in the ghost-cell exchange. Restructuring of the code allowed to fully OpenMP-parallelize this section of the code.

Figure 3 shows the single-node scaling before and after the optimization. While pure MPI scaling (dark blue lines) is close to ideal, the initial pure OpenMP scaling (orange lines) was poor due to the increasing contribution of serial regions (Amdahl's law). The updated OpenMP scaling (light blue) performs as well as pure MPI. The hybrid performance was investigated by varying the OpenMP/MPI contributions. In Figure 3 we exemplify this by either choosing 8 OpenMP threads and increasing MPI tasks, or by fixing two MPI tasks and increasing the OpenMP thread count. This shows a generally acceptable scaling but also points out room for improvement when mixing MPI and OpenMP.

3.4. **Final performance at scale.** Using the updated code version, the initial large scaling run was repeated using 8 threads per task. The result is shown in Figure 4. The data shows that improving the OpenMP parallelism led to an average performance increase of 27% compared to the initial code. This shows a significant reduction of the hybrid penalty. Trace analysis at 16 nodes has indicated that further modifications of the ghost-cell exchange for mixed MPI/OpenMP could improve this even more. Unfortunately a know bug of the default MPI version of SuperMUC-NG at the time of writing made

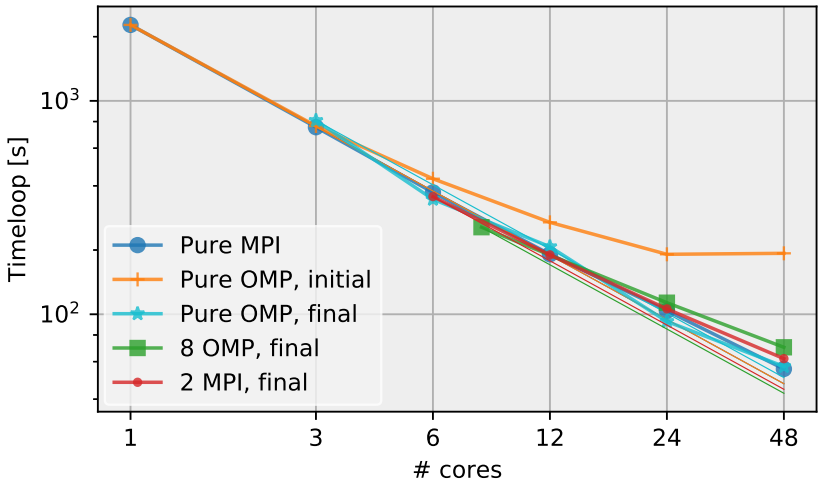

FIGURE 3. Single node scaling with varying contributions of MPI and OpenMP. Code restructuring has yielded a significant improvement in the pure OpenMP scaling, on par with the pure MPI case. The curve labeled as "8 OMP, final" uses the optimized code with fixed eight threads per task, varying the number of tasks. The curve "2 MPI, final" fixes 2 MPI tasks and investigates scaling by increasing the number of threads.

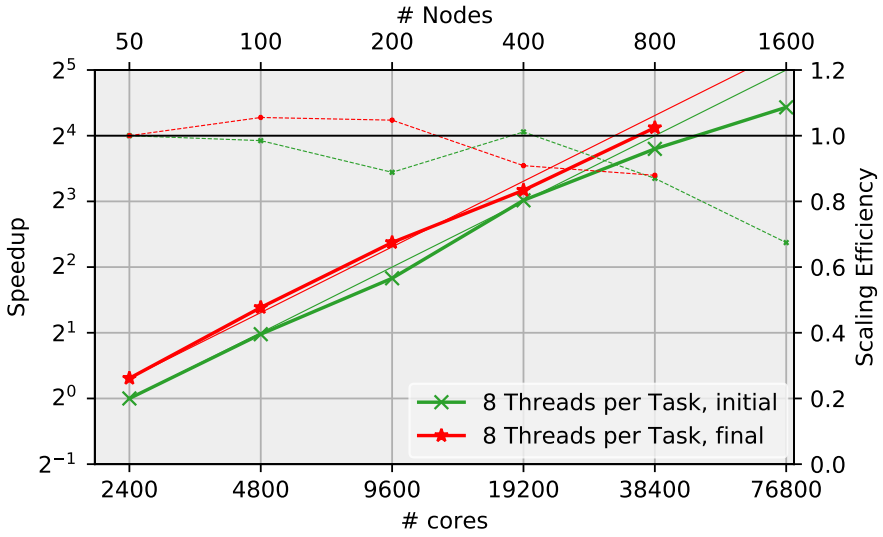

FIGURE 4. Comparison of the initial and final speedup for the large scale setup. The green curve is identical to the one shown in Figure 1.

pursuing a scaling to 1600 nodes and beyond not worth the effort. Yet the data collected so far give already enough directions for future optimizations.

Intel MPI still offers several useful options: switching off *shared-memory intranode communication* by setting `export I_MPI_SHM=off` at runtime, we were able to efficiently port the single-node speedups to a higher node counts (namely, a speedup of a factor 4.5). Plus it offers several switches to reduce the MPI memory footprint, should the initial overhead problem be presented again.

4. CONCLUSIONS AND OUTLOOK

A characterization of the `BHAC` code infrastructure has been obtained using single-node, 16 nodes and 800 node scenarios. The setup used for this initial investigation was deliberately chosen as simple as possible: we have resorted to non-AMR grids and IO has also been excluded. In this simple scenario, OpenMP correctness was addressed and imbalance was solved through `dynamic` scheduling (remaining OpenMP potential gain: 0.5% as reported by VTune).

Guided by VTune's hotspots and threading analyses and APS' communication pattern report, the OpenMP serial fraction was significantly reduced, leading to a performance increase of 27% at scale. While there remains room for improvement, the project has shown that the hybrid implementation is capable to efficiently utilize over 30 000 cores allowing to study large scale problems. The improvements made through in the AstroLab project are already merged into the staging branch of `BHAC` and will become part of the next public release.

There are several potential next steps: *1.* addressing the remaining OpenMP serial fraction in case of mixed OpenMP/MPI parallelization. *2.* Extending the optimizations to the AMR case. *3.* Investigating the performance for a production level setup using AMR. Here, next to the issues due to OpenMP/MPI hybridization, potential load-imbalance on the MPI level will need to be addressed.

REFERENCES

[1] O. Porth et al. "The black hole accretion code". In: *Computational Astrophysics and Cosmology* 4, 1 (May 2017), p. 1. DOI: 10.1186/s40668-017-0020-2. arXiv: 1611.09720 [gr-qc]. URL: http://adsabs.harvard.edu/abs/2017ComAC...4....1P.

[2] R. Keppens, Z. Meliani, A.J. van Marle, P. Delmont, A. Vlasis, and B. van der Holst. "Parallel, grid-adaptive approaches for relativistic hydro and magnetohydrodynamics". In: *Journal of Computational Physics* 231.3 (2012), pp. 718–744. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2011.01.020. URL: http://www.sciencedirect.com/science/article/pii/S0021999111000386.

[3] O. Porth, C. Xia, T. Hendrix, S. P. Moschou, and R. Keppens. "MPI-AMRVAC for Solar and Astrophysics". In: *ApJS* 214, 4 (Sept. 2014), p. 4. DOI: 10.1088/0067-0049/214/1/4. arXiv: 1407.2052 [astro-ph.IM]. URL: http://adsabs.harvard.edu/abs/2014ApJS..214....4P.

[4] Hector Olivares et al. "Constrained transport and adaptive mesh refinement in the Black Hole Accretion Code". In: *A&A* 629, A61 (Sept. 2019), A61. DOI: 10.1051/0004-6361/201935559. arXiv: 1906.10795 [astro-ph.HE].

*API

Email address: o.porth@uva.nl