Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Optimization of PIERNIK for the Multiscale Simulations of High-Redshift Disk Galaxies

Kacper Kowalik[a], Artur Gawryszczak[b], Marcin Lawenda[c]*, Michał Hanasz[a,c], Norbert Meyer[c]

[a]*Nicolaus Copernicus University, Jurija Gagarina 11, 87-100 Toruń, Poland*
[b]*N. Copernicus Astronomical Center, Polish Academy of Sciences, Bartycka 18, 00-716 Warszawa, Poland*
[c]*Poznan Supercomputing and Networking Centre, Dąbrowskiego 79a, 60-529 Poznań, Poland*

Abstract

PIERNIK is an MHD code created in Centre for Astronomy, Nicolaus Copernicus University in Toruń, Poland. The current version of the code uses a simple, conservative numerical scheme, which is known as Relaxing TVD scheme (RTVD). The aim of this project was to increase the performance of the PIERNIK code in a case where the computational domain is decomposed into large number of smaller grids and each concurrent processes is assigned a significant number of those grids. This optimization enable the PIERNIK to efficiently run on Tier-0 machines. In chapter 1 we introduce PIERNIK software more particularly. Next we focus on scientific aspects (chapter 2) and discuss used algorithms (chapter 3) including potential optimization issues. Subsequently we present performance analysis (chapter 4) carried out with Scalasca and Vampir tools. In the final chapter 5 we present optimization results. In the appendix we provided technical information about the installation and test environment.

* Corresponding author. *E-mail address*: lawenda@man.poznan.pl

Table of contents

1 Introduction

PIERNIK [1] is a grid-based MHD code using a simple, conservative numerical scheme, which is known as Relaxing TVD scheme (RTVD). The code relies on a dimensionally split, second order algorithm in space and time. The Relaxing TVD scheme is easily extendable to account for additional fluid components: multiple fluids, dust, cosmic rays, and other physical processes, such as fluid interactions, Ohmic resistivity effects and self-gravity. The simplicity and a small number of floating point operations of the basic algorithm is reflected in a high serial performance.
We have recently implemented two new features: Adaptive Mesh Refinement (AMR) and a Multigrid (MG) solver. The AMR algorithm allows us to reach bigger effective resolutions than it was possible with the uniform grid. It dynamically adds regions of improved resolution (fine grids) where it is required by the refinement criteria. It also can delete grids which are no longer needed to maintain high-quality solution. The MG on the other hand is one of the fastest known methods to solve parabolic and elliptic differential equations, which in our case are used to describe respectively self-gravity of the fluid and diffusion of the cosmic rays. In addition, the isolated external boundaries for self-gravity use multipole expansion of the potential to determine proper boundary values in a fast and efficient manner.
Combination of those two algorithms make PIERNIK an ideal tool for simulations of e.g. isolated galaxy disks. In such cases we need to resolve multiscale environment ranging from parsec scale, gravitationally bound star forming regions, up to tenths of kpc long cosmic ray-driven outflows (see paper). However, our initial large scale tests show that PIERNIK suffers from a few bottlenecks that cripple its performance for a large number (greater than 4096) of processors. We hope that this project will allow us to overcome scarce obstacles that prevent us from reaching maximum potential of our code.
The whole PIERNIK code [2] and test problems are released under GNU GPL terms and thus are not confidential.

2 Scientific aspect

Observation presents that galaxies at the redshift $z \sim 1.5-3$ produce powerful winds which eventually transport a significant fraction of the gas away from the central galaxy making it temporarily unavailable for star formation. The estimated outflow rates can be several times higher than the star formation rates.
We would like to perform simulations of the magnetized interstellar medium (ISM) in isolated models of gas-rich star forming disk galaxies at high-redshift. In our models type II Supernovae locally deposit cosmic rays into the ISM. Our initial work indicates that it leads to the transportation of a significant fraction of gas in a wind perpendicular to the disk plane. The wind speeds can exceed the escape velocity of the galaxies and the global mass loading factor, i.e. the ratio of the gas mass leaving the galactic disk in a wind to the star formation rate, is ~ 10. These values are very similar to values observed for high redshift (z = 2 - 3) star forming galaxies. Therefore cosmic ray driven galactic winds provide a natural and efficient mechanism to explain the low efficiencies for the conversion of gas into stars in galaxies as well as the early enrichment of the intergalactic medium with metals. This mechanism can be of at least as important as the usually considered momentum feedback from massive stars and thermal feedback from supernovae.

3 Algorithms

There are two main grid decomposition approaches in the PIERNIK code. The uniform grid (UG) and the recently developed Adaptive Mesh Refined (AMR).
The UG algorithm divides the grid into smaller pieces, not necessarily of the same size, and assigns one piece to each process. Decomposition is performed in a way that minimizes the total size of internal boundaries between the pieces. Communication between each piece is done via non-blocking MPI communication.
The current implementation of the AMR algorithm uses the Hybrid-Block AMR approach, which means that on each level of refinement the grid is decomposed into relatively small pieces of the same size and shape (typically 16^3 cells) and each grid piece can be covered by some grid pieces at the finer level of refinement. The finer grid pieces do not cover more than one coarse grid piece and the coverage does not have to be complete in order to save computational resources (in contrast to standard Block AMR approaches). The resolution difference between consecutive refinement levels is equal to 2. Typically there are many grid pieces that are associated with a given process. They are kept evenly distributed along a Morton fractal curve to decrease intra-communication and improve load balance.
The multi-grid module creates a stack of coarse grids, each coarsened by a factor of 2 (i.e. compatible with the AMR assumptions) to accelerate approximate solutions of elliptic equations by relaxation. The decomposition of these coarse grids can be performed in a various way, which can influence on internal communication.

3.1 Main performance bottlenecks

There is an overhead caused by the fact that the computational domain is decomposed into smaller grids in order to assign them to the concurrent processes. All those grid pieces require few layers of guardcells to cache fluid data held on the neighbouring grid piece. These guardcells need to be processed and exchanged after each update of the state of the fluid in the MHD solver. The communication part becomes dominant for highly parallel runs (thousands of processes). It is the main factor that limits the scalability of the PIERNIK code.
As the multigrid algorithm is more communication-intensive than the MHD algorithm, the parallelization of the multigrid components depends more strongly on the distribution of grid pieces across the processes. Thus, it is crucial that all communication routines (guardcell exchange and interpolation between grids on neighbouring resolution levels) are independent on the decomposition strategy used on a given grid level. The cost of multigrid communication becomes dominant in highly parallel runs (i.e. with several thousands of concurrent processes).
The MHD solver operates in a directionally-split way so the guardcell-filling can be performed in a similar way and update only the guardcells affected by the recent update. This can reduce the number of messages exchanged between processes.
The complete guardcell exchange currently involves exchanging messages simultaneously with face-, edge- and corner- neighbours (typically 26 neighbours in 3D) followed by a call to MPI_Waitall. It can be replaced by calling exchanges in separate directions (3 stages consist 2 exchanges, each stage followed by an MPI_Waitall). An additional exchange for edge and corner guardcells is required in order to retain compatibility with AMR.
It is assumed that some performance improvements can be achieved by tuning the decomposition of coarse grids in the multigrid solver.
Another optimization possibility is to enable OpenMP parallelism to be used for intra-node communication and leave MPI calls only for performing communication between cluster nodes.

We were considering to perform write operations only from a small number of processes, finally tuned to the level of parallelism available on the local file system. We were also considering writing to multiple files to avoid collective HDF operations.

3.2 I/O strategy

Writing data and restart files often takes more time than performing tens of timestemps. These procedures are considered as bottlenecks which are caused by large number of collective MPI calls originating from HDF5 procedures.

PIERNIK uses parallel HDF5 library. Current implementation allow the two I/O scenarios:

- one process collects data via MPI and writes to single HDF5 file,
- all processes write/read data to/from a single file, using HDF5.

Checkpoints are usually taken once per 12h (runtime parameter) and take 10-100Gb. Output data files are produced usually every 0.5h (few hours maximum). File size ranging from a few Gb to dozens of Gb.

Typical experiment in standard resolution creates 1000 files 3Gb each, additionally a few 15Gb checkpoints. High resolution experiment can take up to 200 files 20Gb each and a few 100Gb checkpoints.

3.3 Optimization procedure

Our goal is to improve PIERNIK code to enable effectively using 8k+ cores on uniform grid. There are three steps we performed:

1. performed speed up analysis – measuring scalability of the MHD solver and detecting bottleneck issues,
2. Implementing improvements for the MHD solver and positive,
3. the multigrid solver is the next target for performance optimization in case the multigrid dominates the calculations.

4 Analysis

4.1 Initial scalability analysis

As a first step, speed up analysis was performed. This analysis has been performed for the gravitational (Jeans) instability test problem called “jeans” engaging both the MHD and Poisson equation solvers. The Jeans instability test problem is representative for a wide class of astrophysical research problems involving selfgravity. The number of cores used for the tests that were performed, varied from 32 to 8192.

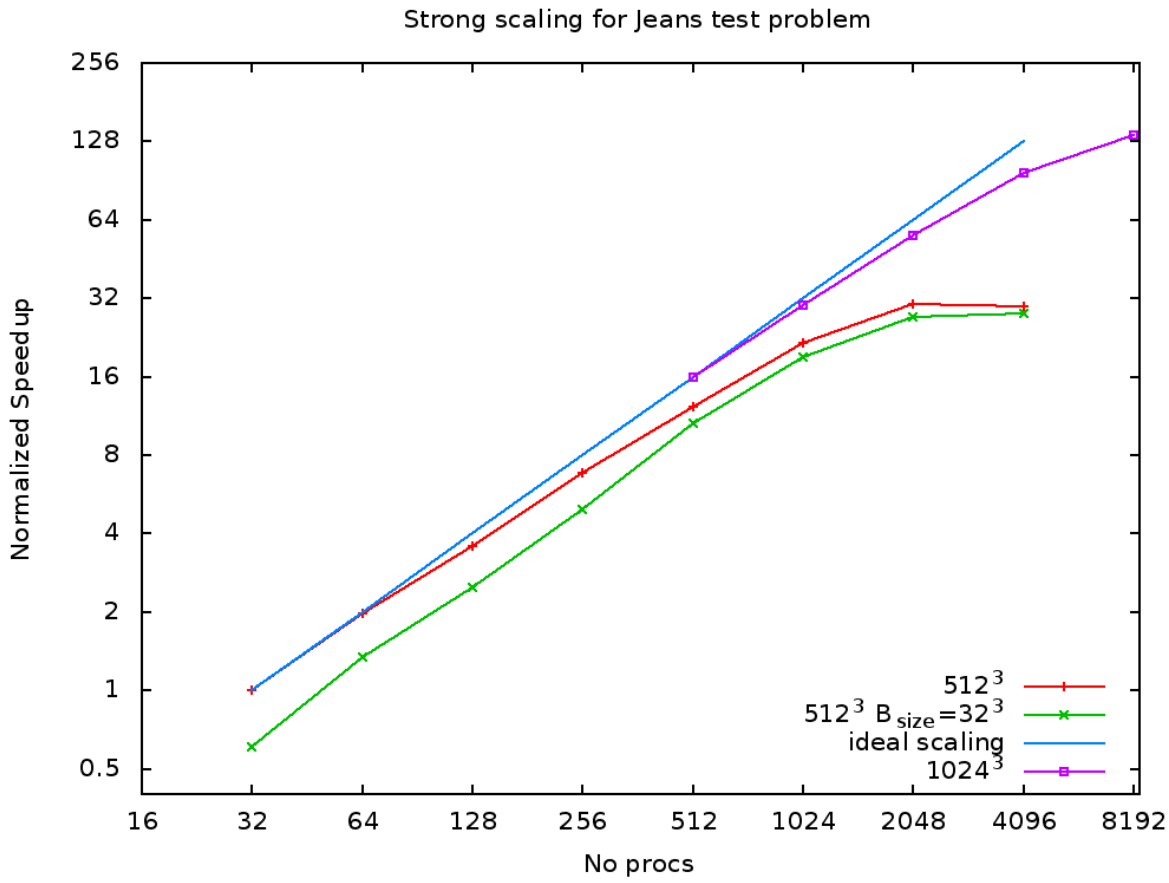

Figure 1. Speed up vs. no of processors for “jeans” problem

The red curve represents a medium size problem (512^3) called "uniform grid" where each MPI process receives a relatively big (maximum available problem size considering the overall grid size and the number of processors), equal part of the grid. In this approach the problem is scalable as long as the ratio of the total volume of boundary cells (four layers of supplementary cells added on external faces of each subgrid designated for the realisation of boundary conditions) to the volume of the proper physical domain is close to zero.

The reason for the smaller computational efficiency follows from the fact that computational costs (per cell) of the MHD algorithm updating boundary cells is the same as in the physical domain. In other words reduced efficiency is compared to the factor ff defined as the ratio of the number of physical to the total (physical+boundary) number of cells. Assuming for simplicity equal block sizes in *x*, *y* and *z* directions one can easily estimate that for block sizes: 32^3 and 64^3 and 4 layers of boundary cells, required by the MHD algorithm, on each face of computational blocks ff equals respectively $(32/(32+2x4))^3 = 0.512$, $(64/(64+2x4))^3 = 0.702$. It is very difficult to reduce the overhead due the additional quantity of boundary cells without a significant complication of the basic MHD algorithm, therefore we do not attempt to do it.

The green curve represents a similar setup with the only difference relying on the replacement of one big MPI block per process by several small blocks attributed to a single MPI process. In this case a more intense comunication (with respect to the case of big blocks, represented by the red curve) between blocks is needed. Both cases represented by red and grean curves converge to the same block size 32^3 at 4096 processes.

We find therefore that the reduction of computational efficiency on the right end of the red and green curves (corresponding to 512^3 mesh divided into 4096 of 32^3 blocks) is partially explained by the overhead of the boundary cells. The remaining part is due to the overhead of MPI communication.

The effect of boundary cell overhead is also visible on violet curve which represents the same problem for a higher grid resolution 1024^3 divided into blocks resulting in ff = 0.68. In this case the code appears scalable up to 8k cores.

4.2 Scalasca optimization analysis

Below we present the screenshot from Scalasca [3] analysis. We can conclude that biggest communication delay appears in “multigrid_gravity_MOD_multigrid_solve_grav” function.

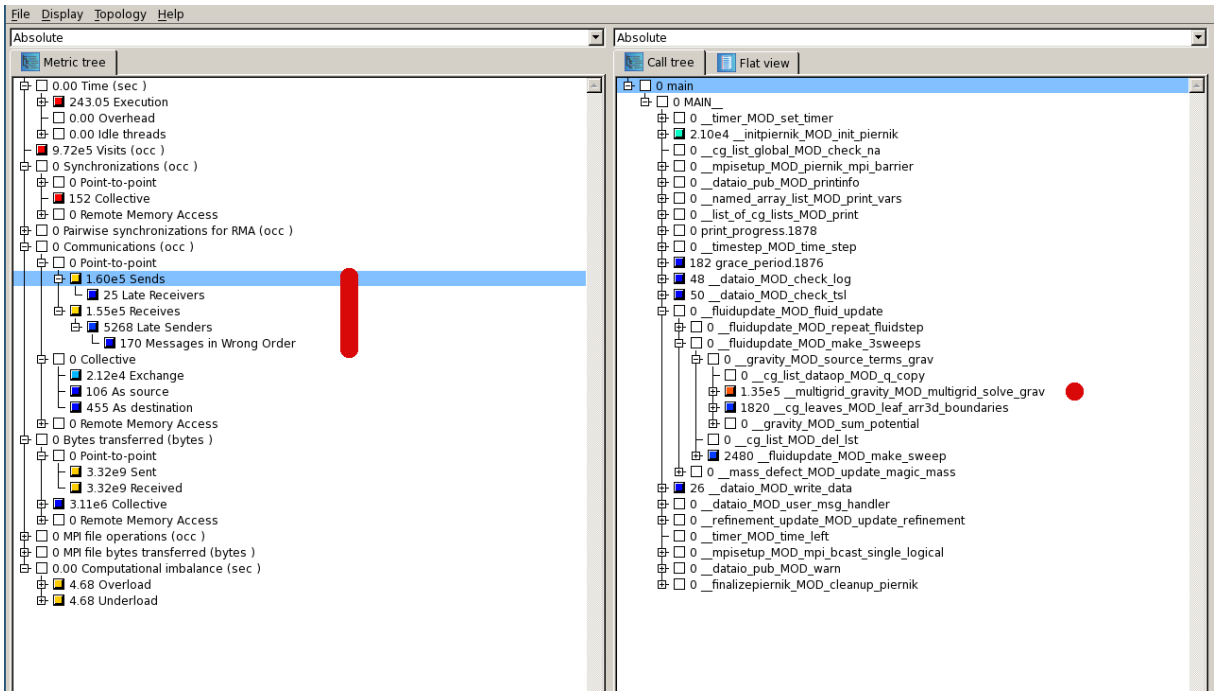

Figure 2. Screenshot from Scalasca optimization analysis

The presented picture shows that MPI communication PIERNIK is problematic in some cases. It is apparent that many communicates are late, some of them are sent in the wrong order.

4.3 Vampir analysis

Results of optimization analysis using Vampir [4] software are presented in figures 3. 4. and 5.
Red colour symbolizes communication delay on specific process.

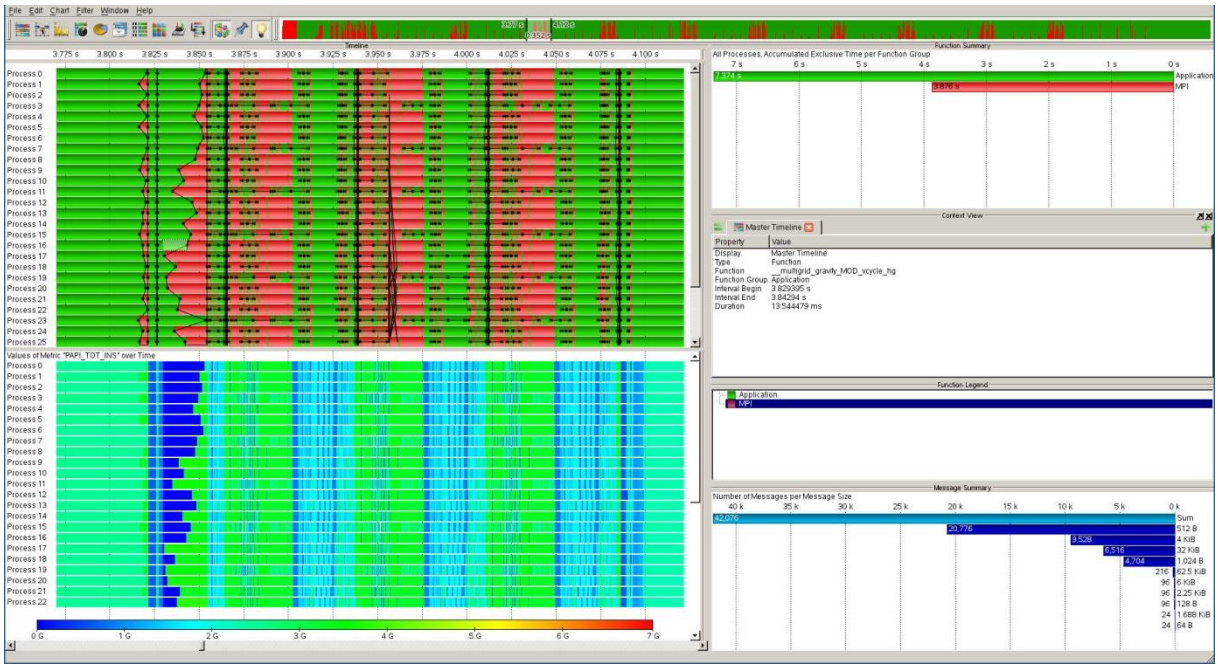

Figure 3. Vampir screenshot – MPI communication analysis – communication lags

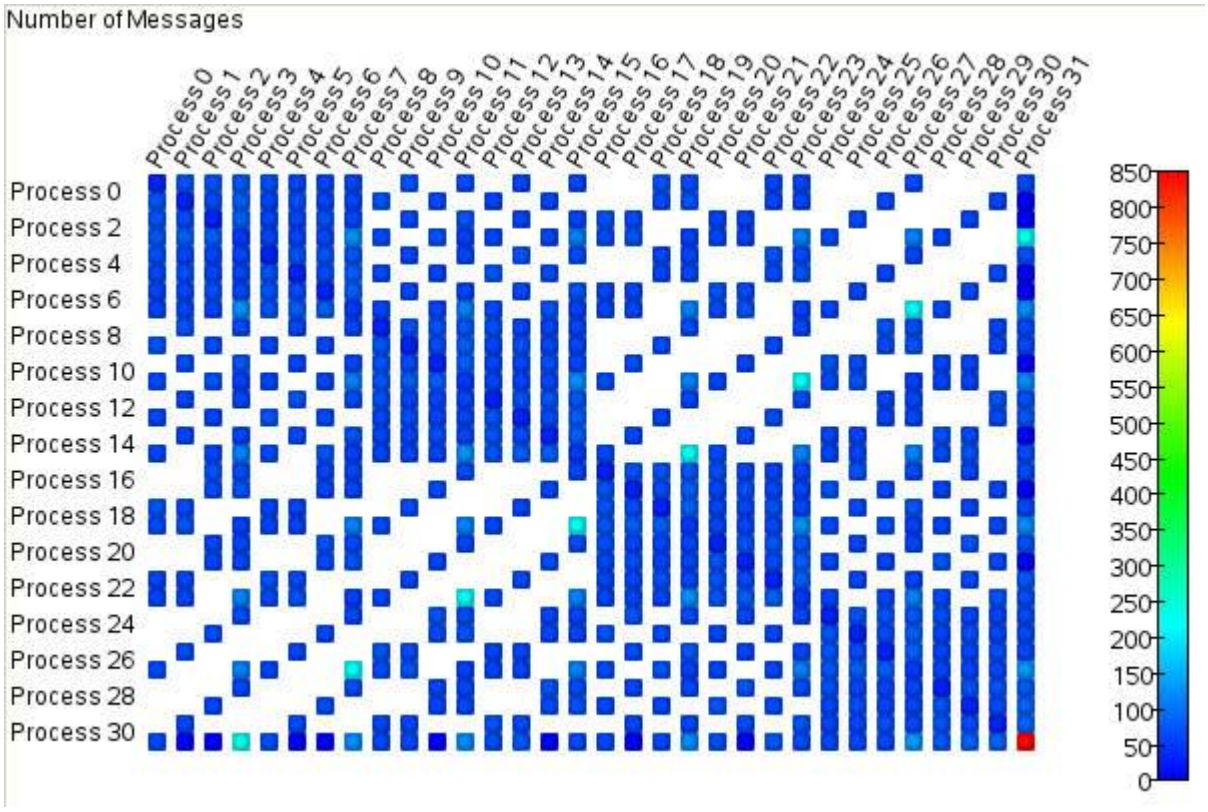

Figure 4. Vampir screenshot – MPI communication analysis.

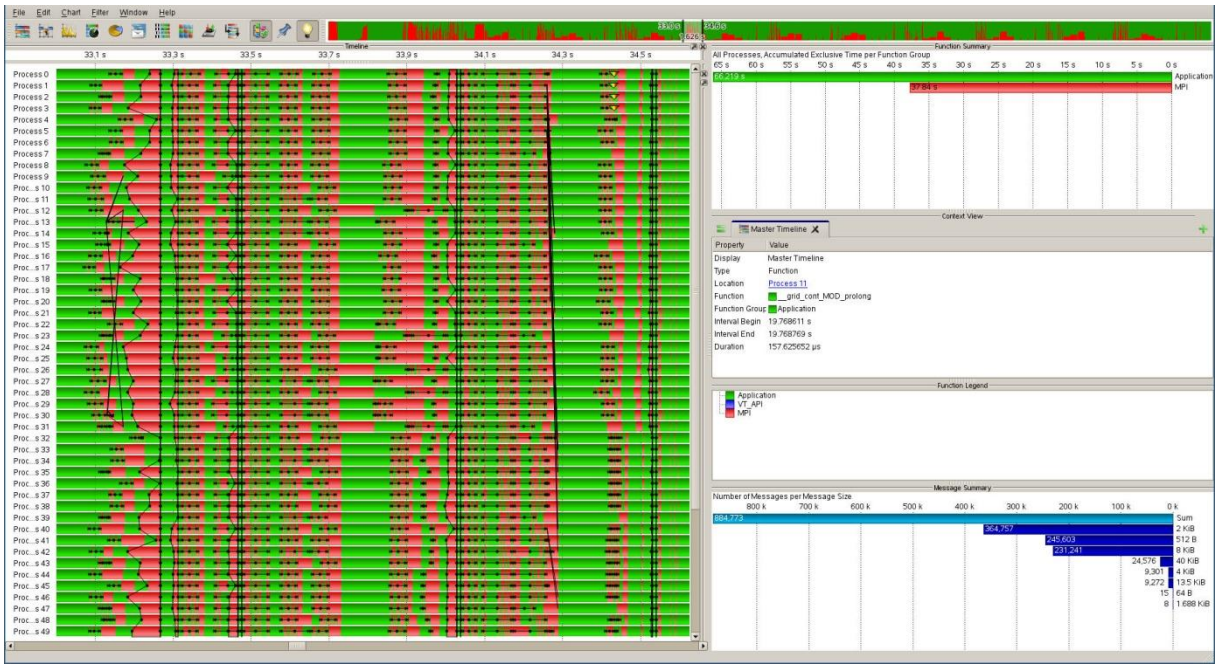

Figure 5. Vampir screenshot – MPI communication analysis – communication lags no. 2

The presented screenshots show that the number of MPI communicates that are exchanged between the processes is very high. Moreover the sizes of messages are very small resulting in a high MPI management overhead (appearing e.g. in a long startup time), what significantly affects the overall processing time.

The majority of the communication is executed by:

- The module responsible for exchanging boundary conditions
- The self-gravitation module where small MPI messages are broadcasted with varying frequency, which makes the optimization difficult.

5 Optimization

The MHD solver in PIERNIK operates in a directionally-split way, so the guardcell-filling can be performed by updating only the guardcells affected by recent 1D MHD step. Communication in separate messages is realized for several arrays used the solver. From analysis performed with Vampir we can conclude that great number of small messages results lags behind the collective.

5.1 Optimization methodology

The following actions were undertaken to optimize the code:

1. The first step in reducing the MPI overhead relies on the identification group of processes running on the same computational node and converting MPI calls into direct memory access.
2. We implemented coalescing of MPI messages wherever it was applicable, i.e. all messages exchanging in one step between a pair of processes are now put into the common buffer and only one message is sent. This implementation significantly decreased fragmentation of the communication.

3. Additionally, in order to decrease the number of MPI messages we have implemented domain decomposition using the Morton space-filling curve (SFC), which provides high "localization", i.e. neighbouring grids are located on the same processes as much as possible. When carefully implemented, the properties of SFC can allow for fast neighbour searching. This is essential for reducing costs of AMR bookkeeping.
4. Finally, we changed AMR so that it became more selective. It doesn't refine the full block at once, but only the required regions are covered by finer grid blocks. This greatly improves the performance of initial iterations of the grid structure and saves some blocks from unnecessary refinements during the regular refinement update.

5.2 Optimization results

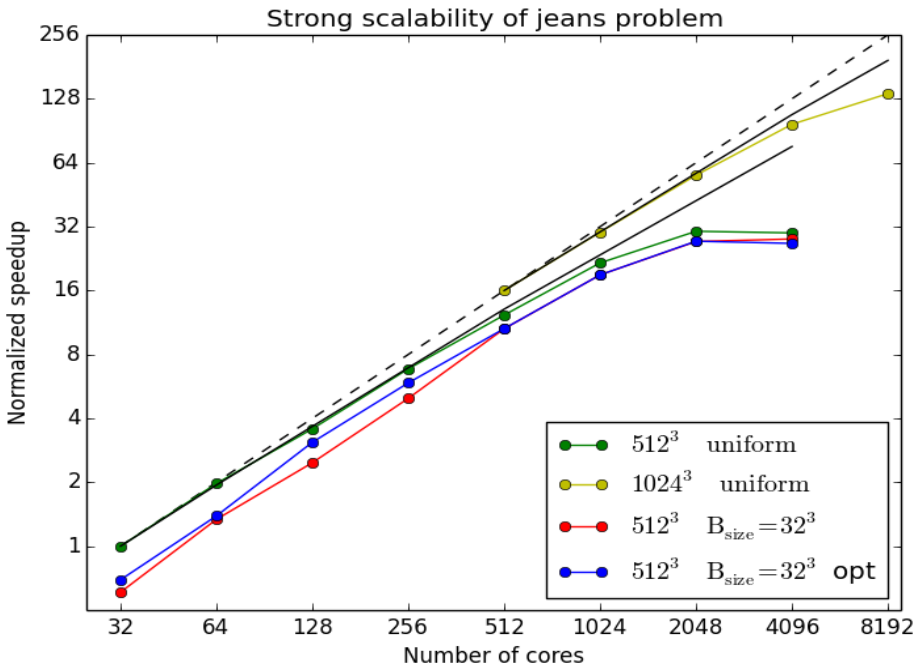

Figure 6. Strong scalability of jeans problem. The red, green and yellow curves represent same data as displayed in Fig. 1. The blue curve shows the effect of our optimization.

Strong scalability curves (Fig. 6) for the uniform grid of moderate sizes (512^3 red, green and blue, 1024^3 yellow) are taken as reference for optimization. Black lines are ideal scaling curves for each of those runs taking into account the ratio of the total number of cells (including guardcell layers) to the number of physically valid cells. Dashed black line shows ideal speed-up. The red curve shows PIERNIK's performance for many grids (of equal cell size without AMR) per computational process before the optimization, the blue curve reflects results of the optimization. The main improvement apparent for the runs on 32 to 256 cores results from the convertion of MPI calls into direct memory access within processes running on the same node. The reduced efficiency of computations on 2048-4096 cores for the 512^3 run should be understood as limitation for the total size of the grid processed by a single core. The present computationally inexpensive MHD algorithm the code scales very well up to for 8192 cores, taking into consideration that one single core processes more than 64^3 cells, even if they are distributed in many blocks.

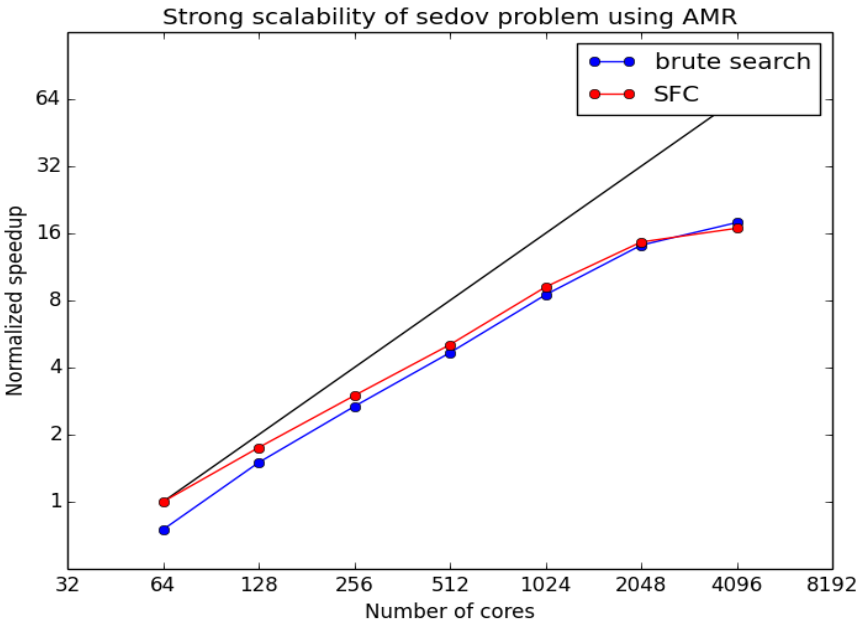

Figure 7. Strong scalability of sedov problem using AMR method.

The strong scaling curve (Fig. 7) in the AMR run showing improvement in performance after implementation of domain decomposition using the Morton space-filling curve. Fig. 8 displays the walltime spent on grid operation (total time minus time spent on hydro algorithm).

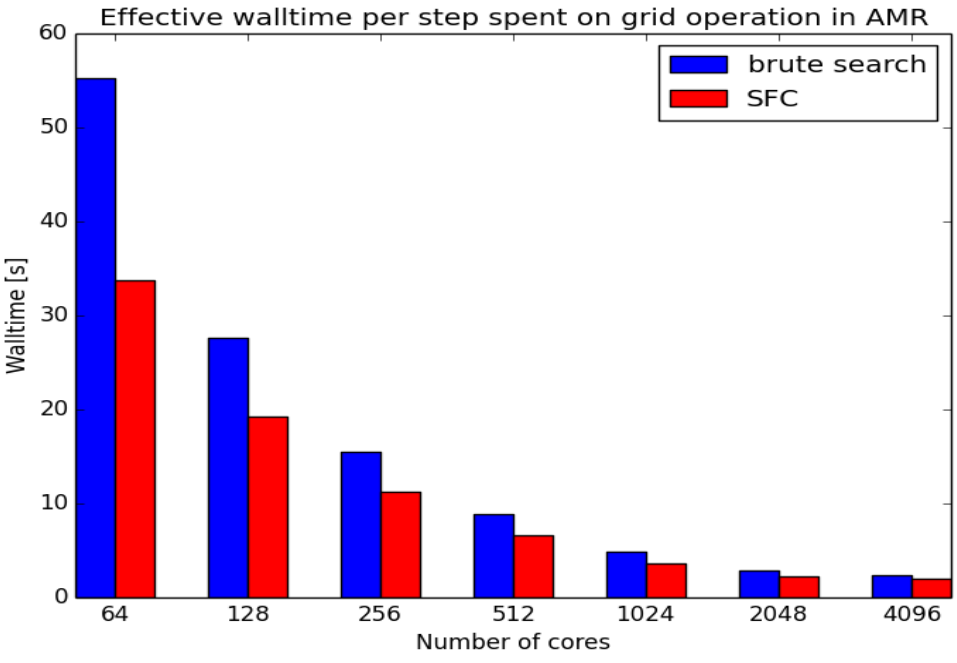

Figure 8. Performance improvement obtained by using SFC for domain decomposition.

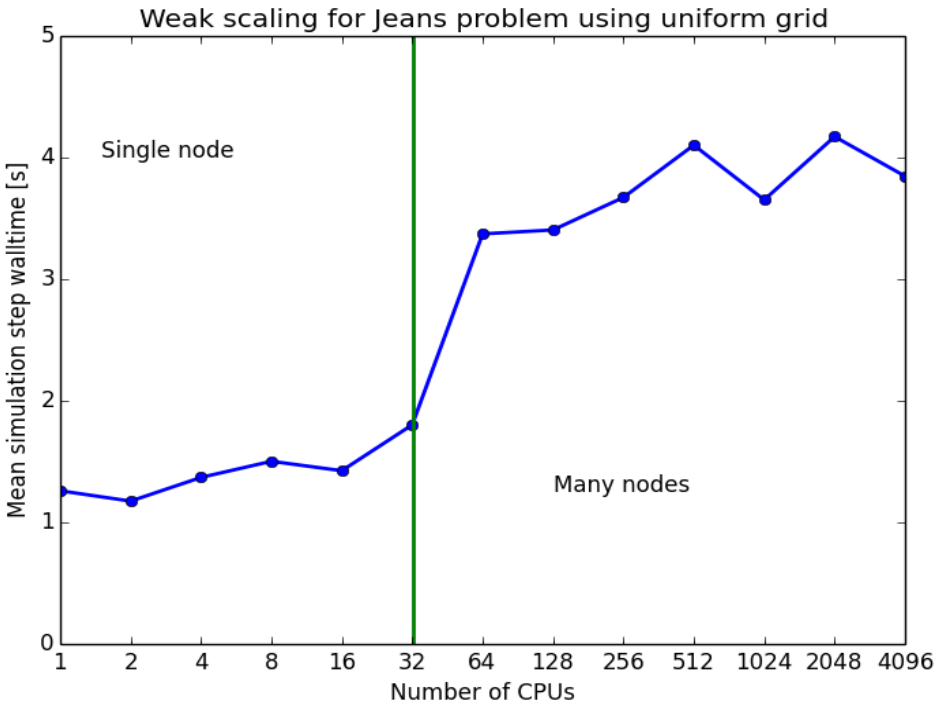

Figure 9. Weak scaling curve for Jeans problem using 64^3 cells per process.

As was shown in figures 6 - 8 several improvements were achieved. Overall the performance of non-uniform and adaptive meshes, that are strictly necessary for the fulfilling scientific goals of the project, was significantly improved. Additionally we have ported the code to the latest Cray compilers which will allow to utilize PIERNIK on the broader range of HPC sites.

5.3 Optimization in numbers

5.3.1 Typical user test cases

Problem size: 512^3, bsize=32^3, with the middle step multiplied by the constant 500.

Number of cores	Wall clock time	Relative Speed-up	Number of Nodes	Number of process
64	12816.5	1	2	64
128	6862	1.87	4	128
256	3665	3.50	8	256
512	1980	6.47	16	512
1024	1060	12.1	32	1024
2048	616	20.8	64	2048
4096	440	29.1	128	4096

5.3.2 Strong scaling curve

Problem size: 512^3 uniform (Fig. 6)

Number of cores	Wall clock time	Relative Speed-up	Number of Nodes	Number of process
32	342.0	1.00	1	32
64	173.3	1.97	2	64
128	95.8	3.57	4	128
256	50.0	6.84	8	256
512	27.9	12.27	16	512
1024	15.8	21.59	32	1024
2048	11.2	30.43	64	2048
4096	11.4	29.90	128	4096

Problem size: 1024^3 (Fig. 6)

Number of cores	Wall clock time	Relative Speed-up	Number of Nodes	Number of process
512	320.0	1.00	16	512
1024	170.0	1.88	32	1024
2048	91.6	3.49	64	2048
4096	52.8	6.07	128	4096
8192	37.8	8.47	256	8192

5.3.3 Weak scaling curve

Source data for figure no. 9.

Number of cores	Wall clock time	Number of Nodes	Number of process
1	1.26	1	1
2	1.175	1	2
4	1.37	1	4
8	1.5025	1	8
16	1.425	1	16
32	1.8025	1	32
64	3.3725	2	64
128	3.405	4	128
256	3.6725	8	256
512	4.1025	16	512
1024	3.6525	32	1024
2048	4.1725	64	2048
4096	3.8475	128	4096

6 Summary

Performed optimization greatly improved scalability of the code nearly reaching the reference performance in the situation when each computational process is assigned only one big chunk of the computational domain. Moreover, utilization of the Morton Space Filling Curve resulted in significant reduction of the time spent on grid operation, which was dominant in simulations using Adaptive Mesh Refinement.
The scalability of the PIERNIK code is predominantly dependent on the number of grid cells attributed to every MPI process. For big meshes of the overall size 10243 the code scales very well with respect to the ideal scaling curve (including the overhead of boundary conditions) up to 4096 CPU cores and shows further speed-up by 50% at 8192 cores. The essential gain in scalability has been achieved for meshes divided into a large number of small blocks, typical for intense use of the AMR technique in multi-scale astrophysical simulations. The strong scalability improvement resulting from the work performed within the current project varies in the range of 10% - 20% for 32-1024 CPU cores.

References

[1] PIERNIK website http://piernik.astri.umk.pl/doku.php
[2] PIERNIK download https://github.com/piernik-dev/piernik.git
[3] Scalasca http://www.scalasca.org/
[4] Vampir http://www.vampir.eu/

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-312763.
The authors would like to thank HLRS for their support and making HERMIT machine available for tests. Moreover we want to thanks reviewers for their valuable comments and suggestions to improve the final document.

7 Appendix 1

7.1 PIERNIK download

To download PIERNIK directly from git repo type: git clone https://github.com/piernik-dev/piernik.git

7.2 System requirements

- git
- python 2.7
- fortran 2003 compiler (>=gfortran-4.7, >=ifort-13.1)
- MPI with ROM-IO (any flavour, tested with mpich2, openmpi, mvapich)
- HDF5 (>=1.8.8, --enable-shared --enable-fortran --enable-fortran2003 --enable-parallel)
- yt for visualization

- FFTW (>=3.0, optional, for selfgravity)
- Lapack (optional, for selfgravity)
- matplotlib (optional, visualization)
- IDL (optional, visualization)

7.3 Test environment

All test were made on **Hermit at HLRS** cluster.

Technical description (installation step 1)	
Peak performance	1.045 PFlops
Cabinets	38 with 96 nodes each
Number of compute nodes	3552
Number of compute cores	per node 2 sockets with 16 cores each: 113 664
Number of service nodes	96
Processor compute nodes	Dual Socket AMD Interlagos @ 2.3GHz 16 cores each
Memory/node	32 GB and 64 GB
Disk capacity	2.7 PB
Node-node interconnect	CRAY Gemini
Special nodes	External Access Nodes, Pre- & Postprocessing Nodes, Remote Visualization Nodes
Power consumption	2 MW maximal