Available on-line at www.prace-ri.eu

Partnership for Advanced Computing in Europe

PLANETESIM: Fourier transformations in the Pencil Code

Joachim Hein[a,*], Anders Johanson[b]

[a]Centre of Mathematical Sciences & Lunarc, Lund University, Box 118, 221 00 Lund, Sweden
[b]Department of Astronomy and Theoretical Physics, Lund University, Box 43, 221 00 Lund, Sweden

Abstract

The simulation of weakly compressible turbulent gas flows with embedded particles is one of the main objectives of the Pencil Code. While the code mostly deploys high order finite difference schemes, portions of the code require the use of Fourier space methods. This report describes an optimisation project to improve the performance of the parallel Fourier transformation in the code. Certain optimisations which significantly improve the performance of the parallel FFT were observed to have a negative impact on other parts of the code, such that the overall performance decreases. Despite this challenge the project managed to improve the performance of the parallel FFT within the Pencil Code by a factor of 2.4 and the overall performance of the application by 8% for a project-relevant benchmark.

Project ID: PLANETESIM

1. Introduction

In this white-paper we report on the enabling work to improve the performance of the Pencil Code [1]. The work was carried out during the livetime of the PRACE DECI 8 project PLANETESIM.

The formation of kilometre sized planetesimals is an important step in the planetary formation process around young stars. The main issue is that small objects of few centimetres do not stick together when they collide. Simulations including drag forces exerted by the gas on particles have shown that pebble sized particles can concentrate in filaments [2]. The mass density in these filaments becomes so large, that the gravitational pull leads to the formation of planetesimals.

The asteroid and Kuiper belt are examples of planetesimal belts unaffected by the later stages of planet formation. The scientific aim of the PLANETESIM project is to use the Pencil Code for high resolution computer simulations aimed to understand the birth-sizes of planetesimals. It is particularly interesting to study whether the simulations reproduce the break in the size distribution for radii of around 50 km, which is observed in the asteroid and Kuiper belt. This is known as the missing intermediate-sized planetesimals problem.

The high resolution simulations required to achieve the scientific aims of the project consume significant amounts of computing resources. Improving the code efficiency and minimising the resource consumption is therefore highly desirable. The project also included enabling effort aiming to improve the performance of the Pencil Code, which is the part of the project this white-paper focuses on. We report on the performance profile of the entire application and the improvements to the routine calculating the gravitational self-potential of the particle density field. The self-potential is calculated in Fourier space. To achieve high performance the routine requires an efficiently implemented parallel fast Fourier transformation.

This white-paper is organised as follows: In Section 2 we give a brief overview on the Pencil Code and discuss its initial performance profile. Section 3 discusses the performance improvement of the routine calculating the self-potential. Our conclusions and an outlook on future work can be found in Section 4.

2. The Pencil Code

2.1. Code overview

The Pencil Code's main aim is to simulate weakly compressible turbulent flows, in particular in a magneto-hydro-dynamic (MHD) context. It is a symmetric finite difference code of high spatial and temporal order. Spatial derivatives in the equations, for example for advection and pressure gradient terms, are found through

*Corresponding author.
tel. +46 46-2224309 fax. +46 46-2224213 e-mail. joachim.hein@lunarc.lu.se

Table 1. Profiling result for the Pencil Code using 32 and 64 MPI tasks. We separately list time spent on executing user code and time spent on MPI calls. Results are given in units of 1000s and were obtained from program executions with 2000 steps.

Routine	CPU time		MPI time	
	32 tasks	64 tasks	32 tasks	64 tasks
total	78.4	80.9	12.2	30.4
`particles_pde_pencil`	27.7	26.8	0	0
`particles_bondconds`	14.0	14.4	1.0	3.7
`calc_selfpotential`	6.2	6.3	6.0	9.9
`particles_pde`	8.7	8.8	0	0
`fold_df`	3.9	7.7	1.9	10.1
`finalize_isendrcv_bdry`	0.7	1.3	2.6	4.8

sixth order polynomial interpolation from the three nearest grid points in both directions. The high order yields high-quality spatial derivatives and hence high effective resolution. A third order Runge-Kutta scheme is used for temporal integration. This is necessary to have a stable scheme (i.e. with positive numerical viscosity). The Pencil Code is optimised for subsonic and transonic compressible turbulence. Particles with independent space and velocity vectors, e.g. representing dust particles in planet formation, can be added as a second component which interacts with the gas via drag force and self-gravity.

The Pencil Code is organised as an open source project with more than 10 active developers and several hundreds of users. An annual meeting, the Pencil Code Meeting, is held to bring core developers and new users together to discuss how to improve the code as well as to present their scientific results. The Pencil Code Meeting 2013 was held at Lund Observatory in Lund from 17th until 20th June 2013.

2.2. Calculating the self-potential in Fourier-space

An important application area for the Pencil Code is astrophysical turbulence. Circumstellar discs around young stars contain both gas and dust particles, the latter being the building blocks of planetary systems. In the early stages of planet formation dust grains collide and stick together to form planetesimals. A necessary ingredient for this to occur may be that particles concentrate in the turbulent gas flow. Overdense regions can become so massive in dust that they contract to form kilometre-scale planetesimals. Hence it is important to have a code which can solve simultaneously for the turbulent flow of gas, the drag force interaction with the dust and the self-gravity of the dust.

The self-gravity of the dust component is found by mapping the particle positions as a continuous density field on the grid. The potential of the particles is then found by solving a Poisson equation on the grid. The most efficient way to solve the Poisson equation on a regular grid is by using the Fast Fourier Transform (FFT). The Poisson equation is directly solvable in Fourier space, under periodic or shear periodic boundary conditions. Hence it is crucial that the Pencil Code can calculate the parallel FFT in an efficient way. Once the gravitational potential of the dust particles is known on the grid, then the acceleration of the particles can be found by interpolation from the nearest grid cells [3].

2.3. Initial performance analysis

To start the enabling project, the Pencil Code was profiled using the parallel profiling tool Scalasca [4]. We used a benchmark configuration describing planet formation using a grid of the size 128^3. The gas is evolved in time on this fixed grid while the dust particles can move freely. The two components are interacting with each other via drag forces. This interaction will eventually result in a streaming instability, concentrating the particles and form planetesimals which are the building blocks of planets [5]. The performance was measured for two different task counts, 32 and 64. Process grids of $1 \times 16 \times 2$ for 32 tasks and $1 \times 32 \times 2$ for 64 tasks were used.

For all the performance measurements of this part of the project we used the Alarik system [6] at Lunarc [7], at Lund University in Sweden. Alarik utilises dual processor nodes with AMD Opteron 6220 processors (Interlagos). Each processor offers 8 cores with a clock frequency of 3 GHz. The nodes are connected with an Infiniband switch network. Each node has a Mellanox 4xQDR Infiniband host channel adapter. The nodes are connected via a network of Mellanox 5030Q 36 port 4xQDR switches. There are three core switches and nine leaf switches.

To obtain runtimes for the instrumented Scalasca runs, which are comparable to the un-instrumented runs, a Scalasca filter file as given in the appendix A had to be used. In addition, the files `general.f90` and `particles_sub.f90` were compiled without instrumentation.

A summary of the profiling results is given in Table 1. The table gives the total runtime and the time spent within the most expensive daughter routines (including decendents) of the routine `pde` inside the module `equ`. There is only a minimal increase in the time spent in user code when the task count is increased from 32 to 64, while there is a considerable increase in the time spent in the MPI library. In line with the original proposal, we

Table 2. Performance of different versions of the subroutine `fourier_transform_shear` including descendants. Measured times are given in units of 1000s. Results have been obtained from benchmark runs of the full application for 2000 steps when utilising 64 MPI tasks.

Code Version				Measured time		
1st transp.	2nd & 3rd transp.	Comm. method	FFT library	CPU	MPI	Total
2 real	2 real	pt-to-pt	FFTPACK	5.8	10.0	15.9
2 real	2 real	all-to-all	FFTPACK	5.7	5.6	11.3
complex	complex	all-to-all	FFTPACK	5.5	5.8	11.3
real	complex	all-to-all	FFTPACK	5.4	5.7	11.1
real	complex	all-to-all	FFTW	4.0	6.3	10.2
real	complex	all-to-all	FFTW, reused exp	1.7	6.4	8.1
real	complex	pt-to-pt	FFTW, reused exp	1.8	7.7	9.4
real	complex	pt-to-pt, no barrier	FFTW, reused exp	1.7	4.9	6.6

focused our effort on the routine `calc_selfpotential`, which is where the Fourier transformations are used. For runs on 64 processors this routine is responsible for about 15% of the total runtime.

3. Fourier transformations and associated data transformations

3.1. Point-to-point and all-to-all communications

In the original code the communications required for the Fourier transformations associated with the calculation of the self-potential are implemented using `MPI_Send` and `MPI_Recv` calls, which are blocking point-to-point communication calls. The Scalasca profile shows a significant load imbalance for the `MPI_Recv` calls of this transposition. Some tasks spend a significant amount of time waiting for their data to arrive. After each transposition the tasks are synchronised in a global barrier before continuing the calculation. As common in parallel FFTs, individual transpositions are happening in subgroups of the processor grid. For the Pencil Code these are the xy-planes or the xz-planes of the process grid.

To avoid any danger of deadlock while also using small send- and receive-buffers, the original code works like a pipe line. In each communication group (a plane of the process grid) the two lowest ranking task will start exchanging data. The lowest ranking process will then move on to the third lowest ranking, fourth lowest ranking and so on, to pair-wise exchange data. The higher ranking processes will initially wait until they exchange data with the lowest ranking. Once a processor has exchanged data with the lowest ranking task, they will wait until they can exchange data with the second lowest ranking, which is followed by a wait for the third lowest ranking and so on.

To remove the initial and final waiting phase of the pipeline, the entire transposition was re-implemented using `MPI_Alltoall` calls inside the communication groups. As shown in the first two rows of Table 2, this change resulted in a significant improvement of the time spent in the MPI library. The 10000s of the original code got reduced to 5600s for a run of 2000 steps using 64 MPI tasks. The timings in the table were determined with Scalasca for the `fourier_transform_shear` routine, incl. decendents. This is the relevant routine inside `calc_selfpotential`. The code was compiled with gfortran version 4.6.2 and the OpenMPI library 1.6.4 was deployed.

3.2. Complex and real data

The next step was to re-implement the relevant communication routine for complex data. While the basic data structures of the Pencil Code are real, the data will become complex after the first partial Fourier transform. Prior to this work the application deployed only real communication routines and separately communicated real and imaginary part, which is denoted as "2 real" in Table 2. Splitting the complex arrays into separate arrays for the real and imaginary parts prior to each communication step requires additional data copies.[1] As shown in Table 2 this step did not yield any significant performance improvement, much to our surprise. The next step was to move the conversion of the real data array into a complex data array until after the 1st transposition, hence only a single real communication is needed - reducing the amount of data in that first exchange. This resulted in a marginal improvement, if at all.

3.3. Exchanging the FFT library

The next step was to exchange the FFT library FFTPACK [8] with the more modern FFTW [9]. This generated a number of issues. So far the Pencil Code adhered to the Fortran 95 standard. However in the most recent

[1] The Pencil Code deploys complex-to-complex FFTs. There is a potential saving to be had by using real-to-complex FFTs and using the symmetries to reduce the array size in the (partial-) Fourier space. During the project it was decided to leave this issue for a future project.

Table 3. Comparison of the overall performance of the Pencil Code for different versions of the subroutine `fourier_transform_shear`. Results have been obtained from 2000 steps benchmark runs utilising 64 MPI tasks. The measurements are expressed in hours of wall clock time. The performance is assessed for the gfortran compiler version 4.6.2 and Intel's ifort compiler version 13.1.

Code Version				Measured time	
1st transp.	2nd & 3rd transp.	Comm. method	FFT library	gfortran	ifort
2 real	2 real	pt-to-pt	FFTPACK	0.491	0.375
real	complex	all-to-all	FFTW, reused exp	0.506	0.396
real	complex	pt-to-pt	FFTW	0.478	0.361
real	complex	pt-to-pt	FFTW, reused exp	0.464	0.360
real	complex	pt-to-pt, no barrier	FFTW, reused exp	0.452	0.346

versions of FFTW the Fortran-interface was declared as deprecated. Users are recommended to call the C-interface using Fortran 2003 functionality. Since this concerns only a single file in the source we decided to use the C-interface instead of adhering to the legacy Fortran bindings.

A portable code utilising FFTW requires control over data alignment, to be able to utilise vector instructions such as SSE and AVX instructions. For this all input and output arrays for the Fourier transformation need to be allocated using the FFTW-provided function `fftw_alloc_complex`. Since all complex data structures used by the code are currently allocated insider a single module, this was not an issue. For real transformations, which are not required for the PLANETESIM project and have not yet been worked on, this needs reassessing in future.

For FFTW the plan creation can be very costly in terms of CPU time. Therefore plans that are to be used repeatedly are best created once and reused many times. The original interface to the FFT library did not anticipate this. It was decided to have each call to the library to check whether a suitable plan exists. If a suitable plan does not exist, which will be the case the first time the routine is called, the plan will be created and stored for future use. Scalasca profiling confirmed the extra overhead for checking whether a plan exists to be negligible in comparison to the cost of the FFT.

As shown in Table 2, the code version deploying FFTW gives a significant performance improvement. It is interesting to note that the change to the FFTW library slowed down the MPI communications using the `MPI_Alltoall`. The reasons could not be understood during the project. Regarding the overall performance of the `fourier_transform_shear` routine, deploying FFTW lead to sizeable performance improvement.

3.4. Shearing

The benchmark used for this investigation requires shearing, which is implemented by multiplying the data with a complex phase. For this complex exponentials have to be calculated. These phases depend are independent of the z-coordinate. As shown in row six of Table 2, calculating the phases once per call, storing and reusing them leads to further significant performance improvement.

3.5. Revisiting the point-to-point routine

It was discovered later during the project, that the transposition routine deploying `MPI_Alltoall`, while performing very well when studied in isolation, would reduce the performance in other parts of the code. The reasons never became fully clear during this project. However we noticed that when using a version of the original point-to-point data exchange modified to transpose complex data together with the improvements discussed in subsections 3.3 and 3.4 a significant overall improvement of the performance can be gained. Removing the barrier synchronisation from the point-to-point routines gave a further improvement to the overall performance.

For the sake of completeness we also studied these versions in isolation, using Scalasca. The results are in the last two rows of Table 2. For both routines the MPI time is significantly shorter than in the original code. The version without barriers is the fastest we observed. For both complex point-to-point versions the time spent in computing part of the code is comparable to the corresponding all-to-all version.

3.6. Effect on the overall performance

Having observed a significant performance improvement for the `fourier_transform_shear` routine, the next step was to study the effect this had on the overall performance of the Pencil Code. Since for this investigation the code was build without any Scalasca instrumentation, we relied on the timers integrated in the application for the performance assessment. We compared the original code to the version of the code that had shown the best performance in the Scalasca studies. This version uses the `MPI_Alltoall`, complex and real transpositions, the FFTW library and reuses the complex phases. However it was observed that the overall performance was inferior to the original code, as shown in Table 3. This holds for the gfortran and Intel's ifort compiler. Looking into the Scalasca profiles confirmed that while the routine `fourier_transform_shear` was better performing other parts of the code got slowed down, such that the overall performance was inferior. This slow down is spread over a large number of routines and could not be understood within the time available to the project.

In an attempt to understand this, we assessed the effect of the individual components of our changes on the overall performance. However the code version using the FFTW library could not be used with the original point-to-point transpositions any longer. Instead of back-fitting this to use old communication routines, it was easier to create a version of the point-to-point transposition suitable for complex data arrays. For both compilers, using these transposition routines, together with the FFTW library yielded a significant performance improvement over the original code. Adding the feature to reuse the complex phases led to an extra performance improvement for the gfortran compiler, while for the ifort compiler this yielded a marginal performance improvement, if at all.

As a last experiment we removed the barrier synchronisation from the transpose routines, which gave a very significant performance improvement for both compilers, see the last row of Table 3. We profiled this code version with Scalasca. The performance of this version of the `fourier_transform_shear` routine is included in Table 2. It is the best performing version the project managed to develop.

4. Conclusions and future work

This white paper describes the enabling work for the Pencil Code performed as part of the PRACE-2IP DECI 8 project PLANETESIM in detail. The project has worked on the parallel Fourier transformations, including their communication-intensive transpositions and introduced parallel I/O into the code.

The work includes an exchange of the FFTPACK library with the FFTW library. It also includes removing synchronisation in the transposition routines and extending them to be able to transpose complex data without splitting them into real and complex data. For the routine `calc_selfpotential` this resulted in a performance improvement by a factor of 2.4. For the entire application we observed an overall performance improvement of around 8%.

For future work it would be interesting to study the performance of the new code for different benchmarks and also on different architectures. There are still open questions regarding the performance of the entire code when using the version based based on `MPI_Alltoall`. The work presented in the white paper was also presented at the Pencil Code developer meeting in Lund. This discussions at the meeting gave new interesting ideas, which have potential to further improve the performance of the communications inside the `calc_selfpotential` routine. This should be implemented and tested in a future project, for which we are currently seeking funding.

Acknowledgements

This work was financially supported by the PRACE-2IP project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-283493. The work was achieved using resources at Lunarc, Lund University in Sweden. The PLANETESIM project used the PRACE Research Infrastructure resources JuRoPa at FZJ in Germany, VIP and Hydra at RZG in Germany and Stokes at ICHEC in Ireland.

References

1. A. Brandenburg, "*Computational aspects of astrophysical MHD and turbulence*", in: Advances in Nonlinear Dynamics. Edited by Antonio Ferriz-Mas and Manuel Núñez. London: Taylor and Francis Group, (2003), p. 269
2. A. Johansen, A. Youdin, M.-M- Mac Low, "*Particle clumping and planetesimal formation depend strongly on metallicity*, Astrophys. J. 704 (2009) L75
3. A. Johansen, J. Oishi, M.-M. Mac Low, H. Klahr, Th. Henning, A. Youdin, "*Rapid planetesimal formation in turbulent circumstellar discs*", Nature (London) 448 (2007) 1022
4. M. Geimer, F. Wolf, B.J.N. Wylie, E. Ábrahám, D. Becker, B. Mohr, "*The Scalasca performance toolset architecture*", Concurrency and Computation: Practice and Experience 22 (2010) 702
5. A. Johansen, A. Youdin, Y. Lithwick, "*Adding particle collisions to the formation of asteroids and Kuiper belt objects via streaming instabilities*", Astron. Astrophys. 537 (2012) A125
6. `http://www.lunarc.lu.se/Systems/AlarikDetails`
7. `http://www.lunarc.lu.se`
8. P.N. Swarztrauber, "*Vectorizing the FFTs*", in: Parallel Computations (G. Rodrigue, ed.), Academic Press, (1982), pp. 51–83.
9. Matteo Frigo, Steven G. Johnson,"*The Design and Implementation of FFTW3*", Proceedings of the IEEE **93** (2005) 216.

A Scalasca filter file

The following filter file, naming routines not to be measured, was used when profiling the Pencil Code with Scalasca. The routine names correspond to version 4.6.2 of the gfortran compiler.

```
__particles_sub_MOD_get_rhopswarm_point
__general_MOD_keep_compiler_quiet_i
__general_MOD_keep_compiler_quiet_r4d
__general_MOD_keep_compiler_quiet_i2d
__general_MOD_random_number_wrapper_0
__general_MOD_random_number_wrapper_1
__general_MOD_ran0
__particles_map_MOD_interpolate_quadratic_spline
__particles_MOD_get_frictiontime
```