Optimizing \textit{Code_Saturne} computations on Petascale systems

Y. Fournier\textsuperscript{a}, J. Bonelle\textsuperscript{a}, C. Moulinec\textsuperscript{b}, Z. Shang\textsuperscript{b}, A.G. Sunderland\textsuperscript{b,*}, J.C. Uribe\textsuperscript{c}

\textsuperscript{a} EDF R&D, MFEF, 6 quai Watier, 78400 Chatou, France  
\textsuperscript{b} STFC Daresbury Laboratory, Warrington WA4 4AD, UK  
\textsuperscript{c} School of MACE, University of Manchester, Manchester M13 9PL, UK

\section{Introduction}

The performance capabilities of high-end computational resources have increased rapidly over recent years. In particular, the introduction of Petascale systems has brought with it massive increases in the number of processing units, where it is now common to have many tens of thousands of cores available for users codes. This development raises a number of significant challenges for the parallel performance of Computational Fluid Dynamics (CFD) applications. Recently, new parallelization and optimization techniques have been introduced to the open source CFD code \textit{Code_Saturne} \cite{1} in order to address these challenges at several different stages of the calculation. The introduction into the code of multigrid iterative solvers, parallel I/O and new parallel partitioning tools is described in Section 3. In Section 4, the impact on scalability and performance of these new features have been analysed on a range of prototype Petascale systems. Under the Partnership for Advanced Computing in Europe (PRACE) project \cite{2}, \textit{Code_Saturne} has been chosen as a CFD applications benchmark to be used for assessing the performance of proposed Petascale systems. The PRACE initiative, \textit{Code_Saturne} and the dataset for benchmarking are introduced in the following section.

\section{\textit{Code_Saturne} in PRACE}

\subsection{The Partnership for Advanced Computing in Europe (PRACE)}

The Partnership for Advanced Computing in Europe (PRACE) \cite{2}, funded in part by the EU’s 7th Framework Programme, has been formed with the objective to create a persistent pan-European High Performance Computing (HPC) service. It is intended that this service will consist of three to five tier-0 centres providing European researchers with access to capability computers and forming the top level of the European HPC ecosystem. The partnership comprises of organizations from 20 European countries, with the UK’s Engineering and Physical Sciences Research Council (EPSRC) \cite{3} involved as one of the six principal partners. The Computational Science and Engineering Department \cite{4} at STFC Daresbury Laboratory \cite{5} has undertaken work in several of the specified PRACE Work Packages during the preparatory phase of the project. The largest work package has been WP6, which has as its objective: to identify and understand the software libraries, tools, benchmarks and skills required by users to ensure that their application can use a Petaflop/s system productively and efficiently. The prevalence of multi-core technologies indicates that future Petascale systems are likely to have hundreds of thousands of cores. A key element of WP6 is ensuring that it enables understanding of how the major application codes used across Europe can exploit these large systems. \textit{Code_Saturne} has been selected as the core application benchmark code in PRACE representing CFD applications. With some of the planned tier-0 Petaflop/s systems now fully
operational in Europe, the PRACE project has recently entered a new implementation phase (PRACE-1IP). It is planned that Code_Saturne will continue to be developed to assess application benchmark performance under the new project.

2.2. Code_Saturne

Code_Saturne has been under development since 1997 by EDF R&D (Electricité de France). The software is based on a co-located Finite Volume Method (FVM) that accepts three-dimensional meshes built with any type of cell (tetrahedral, hexahedral, prismatic, pyramidal, polyhedral) and with any type of grid structure (unstructured, block structured, hybrid). It is able to handle either incompressible or compressible flows with or without heat transfer and turbulence. Parallel code coupling capabilities are provided by EDF's Finite Volume Mesh library (under Lesser General Public License (LGPL)). Since 2007, Code_Saturne has been open-source and available to any user. The main hub in the UK is the University of Manchester, which is part of the UKTC consortium and has been using and co-developing the software since 2001 [6].

2.3. Description of dataset for benchmarking

The flow around the demonstration DARPA2 submarine (see Fig. 1 Left) [7] is considered to be a suitable large-scale benchmark for the code. The ability of Code_Saturne to compute the friction coefficient is first investigated. The Reynolds number based on the length of the hull is $3.89 \times 10^7$, the free stream velocity 9 m s$^{-1}$ and the pressure outlet $2.01 \times 10^5$ Pa. The flow is highly turbulent and the high Reynolds number $k-\epsilon$ model is used. An unstructured mesh made of tetrahedra and prisms is generated in such a way that $y^+ = 30$ and five layers of prisms are built to catch the boundary layer effects around the submarine. Sensitivity of the friction coefficient to the mesh has shown that 0.2 M cells are sufficient for its computation.

The simulation is run until steady state and the friction coefficient is plotted as a function of the distance from the bow. Results are compared to experimental data. Overall Code_Saturne produces a very good estimation of the friction coefficient (see Fig. 1 Right). It is somewhat underestimated near the bow, but very well predicted at the tail of the submarine. This gives confidence in the numerical model which is used.

In order to analyse the performance of the code, two other unstructured meshes are generated, with 5.7 M and 107 M of cells, respectively. The cases of 5.7 M and 107 M cells have been run here without any turbulence model (laminar equations) as, for reasons of throughput, performance test runs have involved only 50–100 timesteps, which is not enough to reach convergence. In the figures and associated discussion 'timestep' refers to the relaxation parameter of the steady state algorithm.

We focused on the Navier–Stokes solver and the pressure equation, where between 60% and 80% of the CPU time is spent, depending on the configuration. For this dataset, $k$ and epsilon turbulence modelling only accounts for about 20% of the whole simulation time. In Code_Saturne this is solved by Conjugate Gradient.

3. Code_Saturne petascaling and optimization techniques

Several advances have been made to Code_Saturne that have improved significantly the Petascale performance of the code. These optimizations have focused on the linear solver, I/O and the pre-processing stage.

3.1. Optimization of the linear solver

During the initial phase of the PRACE project the preconditioned Conjugate Gradient solver in Code_Saturne has been replaced by an algebraic multigrid [8] Conjugate Gradient approach for solving the pressure equation. This is expected to benefit performance by accelerating convergence rates in the iterative solver, i.e. reducing the number of iterations per timestep and reducing the average number of floating-point operations per iteration (due to fewer mesh points in the coarser grids). For multigrid, computational complexity scales quasi-linearly with problem size. In addition to improving performance, this optimization improves the robustness of the code by making convergence to solution more likely.

Fig. 2 compares the solver performance of multigrid variant of Conjugate Gradient (CG_MG) vs. Conjugate Gradient only (CG) on both Cray XT4 and IBM Blue Gene/P systems. Performance evaluations have taken place on each platform for two different problem sizes of 3D tetrahedral meshes around the submarine dataset described in the previous section (illustrated in Fig. 1). The results demonstrate that for the larger 107 M cell problem, even out to extremely large numbers of cores, the multigrid version is significantly faster to solution at each timestep of the calculation. However the Conjugate Gradient CG performs around 30% faster than CG_MG on the smaller 5.7 M cell problem at 4096 cores. The main reason for this behaviour lies with the limitation that mesh coarsening in the multigrid solver does not cross mesh partition boundaries. This avoids undertaking a costly re-partitioning scheme for each level built, but limits the coarsening when we have high surface to volume ratios, such is the case for the 5.7 M cell problem on large core counts. Here for example the 64 core run has 10 levels of multigrid with a coarsest grid of 1488 cells, whilst the 4096 core run creates only seven levels for multigrid

![Fig. 1. Left: mesh around the submarine. Right: friction coefficient as a function of the distance from the bow.](image-url)
that the code undertakes often require the input of huge datasets, particularly relevant for disk and writing data to disk has become increasingly important.

3.2. Optimization of I/O

As a general pattern noticeable from the two charts is the superior scaling of both solvers on the IBM Blue Gene/P. For example on 4096 cores, the CG_MG solver is 4.6 times faster on the Blue Gene/P than the Cray XT4 for the smaller 5.7 M cell case. The principal explanation for this behaviour is that the computational power of the processing cores on the Cray XT4 is much higher: the Cray XT4 has 2.3 GHz quad-core AMD Opteron processors whilst the IBM Blue Gene/P has 850 MHz quad core PowerPC 450 processors. The combination of slower clock rate and fast interconnect on the Blue Gene/P means that communication loads remain more in balance with computational loads on higher core counts. This also explains the relatively good performance of the Cray XT4 on lower processor counts for the 5.7 M cell dataset and the better scaling for the 107 M cell dataset, where the computation/communication loading is likely to be more balanced for this architecture. For the Blue Gene/P, parallel scalability for CG_MG-based solves on the 107 M cell dataset is 75% for core counts of between 2048 and 8192, and 40% for core counts of between 8192 and 32,678. For CG-based solves the corresponding figures are 89% and 62%.

An important feature of multigrid algorithms is that they add numerical robustness to sparse linear solvers. Whilst no indirect method can guarantee convergence for all systems, employing multigrid techniques significantly reduces the likelihood of non-convergence. This important property was evidenced here, as the Conjugate Gradient solver alone tended to breakdown at high Reynolds numbers when attempting the simulation with purely laminar flow (i.e. with no turbulence model). The current version of the algebraic multigrid algorithm could be optimized, if extra cell aggregation would be performed. In the existing version, the coarsest grid size is at least equal to the number of subdomains. Aggregating cells during the restriction step could allow to get coarsest grids smaller than the number of subdomains and therefore to reduce computing time on the coarsest grid.

3.2. Optimization of I/O

The development of efficient methods for reading data from disk and writing data to disk has become increasingly important for codes that use high-end computing resources. This issue is particularly relevant for Code_Saturne, as the large-scale simulations that the code undertakes often require the input of huge datasets from disk and the output of large amounts of results files. For example, a dataset comprised of 1 billion cells requires around 70 GBytes of storage. Moreover, the outputs are usually required at frequent intervals in order to model a system that changes with time. Parallel I/O strategies have been introduced to the code in order to address the I/O bottleneck associated with runs on large numbers of cores. Both serial and parallel I/O implementations are designed to use a common interface. The parallel I/O is fully implemented for reading of preprocessor and partitioner output and restart files. Very recently, Parallel I/O for visualisation data exported to the EnSight gold format has also been implemented.

A two-stage scheme, summarised in Fig. 3, is used:

1. Partitioned data on p processors is redistributed to n blocks, with \( 1 \leq n \leq p \). The block distribution is based on an entity’s global number, and all blocks are of the same size (except for the last); a minimum blocks/cores ratio or block size may optionally be defined.
2. Actual I/O may be done using the block distribution, using collective operations on shared files, so that the files are partitioning-independent.

It is to be noted that this type of pre-distribution may be done internally by MPI-I/O, and we could in many cases avoid the first stage using indexed datatypes. This may be tested in the future, but although using indexed datatypes is recommended in MPI tutorials and should in theory allow for better performance, indexed datatype bugs encountered in Open MPI 1.1 and 1.2 a few years ago have led us to be cautious using those datatypes, and we did not observe any performance gain compared to using additional copies to serialize data (as datatypes were usually used only once, their construction was perhaps not amortised). Also, using a block distribution for data that is itself indexed such as polygon connectivity only requires additional MPI_Bcast operations, while global indexes of indexes would be more difficult to manage.

Predistributing data to blocks will make it easy to use visualisation output APIs based on parallel HDF5, which is more restrictive regarding data distribution than pure MPI-I/O. Finally, Code_Saturne’s parallel I/O layer allows circumventing MPI-I/O altogether, serialising reading or writing of blocks by rank 0 with a simple send/receive scheme. This is a useful fallback mechanism when issues with MPI-I/O are encountered, especially on older systems, and has the added benefit of allowing comparison a serial and parallel I/O performance using almost identical code paths.
Finally, with only minor changes, by ensuring blocks are used only on 1 of setsize cores, it will be possible to use a "coarser" communicator for I/O than for computation, which will provide an additional tuning parameter.

Initial tests using MPI-I/O seem to indicate that performance is quite irregular, and may depend on many parameters, such as block size, alignment, filesystem striping, and data distribution options. We have only scratched the surface of I/O performance benchmarking, but on a Blue Gene/P system with a GPPS filesystem running on 2048–16,384 cores, an improvement factor of 2–7 using MPI-I/O compared to serial I/O is obtained, though it may actually be slower for some files in certain configurations. This is far from optimal, but better than initial tests on a Nehalem cluster using a Lustre filesystem, in which shared-file MPI-I/O seems to perform no better on average than serialised I/O. Judging from other benchmarks such as [9], our initial results are not surprising, and though we expect to be able to be using the many tuning parameters available to improve I/O rates, obtaining consistent gains in a portable and automatic manner will be a challenge. Currently, I/O is not yet a bottleneck for the code’s performance, but the many available tuning options associated with real-world data using the shared file paradigm (so as to make the partitioning more “transparent” to the user) provide interesting benchmarking possibilities in the PRACE context.

3.3. Optimization of the pre-processing stage

A current trend in HPC architectures is for the expansion of memory resources to not match that of CPU resources and therefore memory per core is often reducing on high-end systems. Meanwhile, researchers are attempting to simulate much larger, more complex physical systems. For CFD applications, one major consequence of these trends is that the parallel performance and efficiency of partitioning software is becoming increasingly important. In this paper we investigate the performance (involving both speed and quality) of several parallel partitioning software packages: ParMETIS [10], PT-SCOTCH [11], the application of a Morton-based Space-filling Curve (SFC) [12] algorithm for parallel partitioning and compare them to a serial one, METIS [13]. METIS, ParMETIS and PT-SCOTCH use graph-based methods whereas SFC is geometry-based.

The results in Table 1 compare partitioning performance of the three parallel methods for the 107 M cell submarine dataset undertaken on a Cray XT4 system for up to 131,072 subdomains. For the 131,072 subdomain case, PT-SCOTCH can be run on only 16 cores, whereas ParMETIS requires at least 256 cores. SFC requires one core per domain, therefore the largest case analysed with this method involves 8192 domains. An observed trend is that the quality of the partitioning reduces somewhat as the number of cores increases. The load-balancing of ParMETIS is noticeably poorer throughout, whilst the maximum neighbour count, which gives an indication of communication loads, is comparable for ParMETIS and SFC, but relatively lower for PT-SCOTCH. Overall, partitioning speeds are similar for ParMETIS and SFC, but slower for PT-SCOTCH. At this stage it is difficult to conclude as to which is the preferred partitioner. Halo cells have not been considered here. Code_Saturne is then run on each partition in order to determine the best CPU time/per iteration for the solver. Fig. 4 Left shows them for the 107 M cell submarine dataset up to 32,768 cores on the IBM Blue Gene/P. As expected, METIS induces the best performance for Code_Saturne, while SFC gives the worst. PT-SCOTCH partitioning is better than ParMETIS and represents a very good alternative to METIS for large domains, where serial software would not be able to partition due to memory limitations.

4. Code_Saturne performance on PRACE prototype systems

Several Petaflop/s-class prototype systems have been installed across Europe in order to evaluate next-generation HPC architectures for PRACE. These cover a wide variety of advanced computing architectures that are expected to scale to Petaflop/s performance in the next 1–2 years, including IBM Blue Gene/P [14] and IBM Power 6 [15] series, Cray XT4/XT5 [16] and Nehalem-based cluster systems [17]. In preparation for full-scale Petalop/s systems, these prototypes are being assessed at many levels including performance, scalability, total cost of ownership and energy consumption. Code_Saturne has been developed to C, Fortran and Python language standards and is therefore highly portable software. Consequently there were no significant problems encountered when porting the code to the PRACE prototype systems. A series of experiments was undertaken to determine the most favourable compiler optimization levels on each machine. Fig. 4 Right summarizes the relative performance and parallel scaling of Code_Saturne on these systems for the 107 M cell submarine dataset. It can be seen that the code scales well up to very large core counts on all the platforms investigated, with particularly fast, scalable performance demonstrated on the IBM Power 6 platform.

An overall breakdown of run-time between solver and I/O stages for a typical benchmark run is shown in Fig. 5. Overall I/O costs are minimal (6%) at 2048 cores but are a greater overhead at 32,768 cores (31%). It should be noted that production run simulations would generally involve many hundreds of timesteps, rather than the 50 timesteps used in the benchmark tests, thus reducing the impact of I/O overheads significantly.

5. Concluding remarks—perspectives

Code_Saturne is highly portable, powerful, scalable CFD software and therefore an ideal choice for inclusion in the PRACE application benchmark suite. Due to improvements made to the parallel code over the past couple of years, several of which are detailed in this paper, Code_Saturne has very good scaling properties across all the PRACE prototype systems tested. A series of machine-specific optimizations for IBM platforms has resulted in particularly fast and highly scalable performance on large-scale IBM systems such as the Power series and Blue Gene platforms. It is intended that these low-level specific optimizations will be further tuned for other platforms as the PRACE project progresses. The scaling perfor-
Performance remains good on the IBM Blue Gene/P, right out to 32,768 cores – a feat only matched by a few other codes in the PRACE application benchmark suite. Code_Saturne will continue to be a PRACE application benchmark in the new Prace-1IP implementation project.

Several major optimizations may further improve the code over the coming year:

- IBM has worked on local mesh renumbering in the context of OpenMP enablement, and that work will be integrated into the mainline development tree over the coming months.

- The multigrid algorithm does not coarsen the mesh across partition boundaries. On large processor counts, this means the coarsest matrix has at least as many global lines as there are MPI ranks, with only 1 or 2 coarse cells per rank and solving for this matrix using an iterative algorithms may still require several hundred or thousand iterations, which are completely latency-bound. A planned improvement is to regroup (i.e. gather) coarse mesh data on one out of every 2–8 ranks every few multigrid levels, to allow for a much smaller coarse system, requiring fewer iterations. Ideally, a very coarse system could be handled by a single rank, avoiding communication latency.

- Partitioning could be improved by switching from the Morton SFC to the Peano-Hilbert SFC, with no loss of scalability, and with an expected improvement in partitioning quality which could reduce the gap between PT-SCOTCH and internal (SFC) partitioning. Also, part to block distribution and its reverse currently use MPI_Alltoallv operations, which may become a bottleneck at very high processor counts. Moving to sparse all-to-all communications using algorithms such as the crystal-router or possible future MPI-3 sparse collectives within a few years may be necessary, though using an hybrid MPI-OpenMP approach should already reduce communicator sizes and mitigate this issue in the short term.

### Acknowledgements

The authors would like to thank the PRACE project organizers and partners, without whom much of this work would not have been possible. In particular CSC in Finland [18], SARA in the Netherlands [19], and FZ Jülich in Germany [20] for allowing access to

---

**Table 1**

Comparison of mesh partitioning for 107M cell submarine dataset for ParMETIS 3.1.1 (PM), PT-SCOTCH 5.1 (PT-S) and SFC.

<table>
<thead>
<tr>
<th>Domains</th>
<th>Minimum cores</th>
<th>Partitioning time (s)</th>
<th>Load balance ratio</th>
<th>Maximum neighbours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PM</td>
<td>PT-S</td>
<td>SFC</td>
<td>PM</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>16</td>
<td>32</td>
<td>44.3</td>
</tr>
<tr>
<td>128</td>
<td>32</td>
<td>16</td>
<td>128</td>
<td>44.5</td>
</tr>
<tr>
<td>512</td>
<td>32</td>
<td>16</td>
<td>512</td>
<td>45.0</td>
</tr>
<tr>
<td>2048</td>
<td>32</td>
<td>16</td>
<td>2048</td>
<td>46.5</td>
</tr>
<tr>
<td>8192</td>
<td>32</td>
<td>16</td>
<td>8192</td>
<td>50.5</td>
</tr>
<tr>
<td>16,384</td>
<td>32</td>
<td>16</td>
<td>–</td>
<td>52.5</td>
</tr>
<tr>
<td>32,768</td>
<td>64</td>
<td>16</td>
<td>–</td>
<td>33.4</td>
</tr>
<tr>
<td>65,536</td>
<td>64</td>
<td>16</td>
<td>–</td>
<td>33.4</td>
</tr>
<tr>
<td>131,072</td>
<td>256</td>
<td>16</td>
<td>–</td>
<td>34.6</td>
</tr>
</tbody>
</table>

---

**Fig. 4.** Left: parallel performance of Code_Saturne for 107 M cell case depending on the mesh partitioner. Right: relative parallel performance of Code_Saturne for 107 M cell case on PRACE prototype systems.

**Fig. 5.** Breakdown of runtime for Code_Saturne between solver and I/O for 107 M case.
their systems and providing help and assistance. They also would like to thank The Engineering and Physical Science Research Council (EPSRC) in the UK for their support of Collaborative Computational Project 12 (CCP12).

References