**Proceedings of the 7th GACM Colloquium on Computational Mechanics
for Young Scientists from Academia and Industry
October 11-13, 2017 in Stuttgart, Germany**

Load-balance strategies for CFD-codes on HPC systems

Philipp Offenhäuser[1]*

Micro Abstract
Today's HPC systems generate their performance by facilitating hundreds of thousands of cores. In order to use this computing power efficiently, the computational effort must be distributed evenly across all cores. Techniques for distributing the simulation initially are well-known. Based on numerical and physical phenomena additional computational effort may occur locally, at run-time. Techniques are presented which recognize these additional loads and redistribute the simulation evenly.

[1]Numerical Methods & Libraries, High Performance Computing Center Stuttgart, Stuttgart, Germany
***Corresponding author**: offenhaeuser@hlrs.de

Introduction

Numerical simulation is an indispensable tool in science and research. In the last decades it has strongly profited from the continuing increase of available computing power. However, clock speeds and computing powers of individual processor cores no longer see the significant improvements that made computers more and more powerful in the past. Instead, current supercomputers generate their increase in performance by facilitating more and more cores in parallel, with modern systems already having hundreds of thousands of cores [10].

Computational fluid dynamics (CFD) is one of the research areas that profits most strongly from fast computers. A number of CFD algorithms specifically designed for massive parallel hardware was developed and implemented recently [5, 7]. To use the performance of massive parallel HPC-hardware the communication effort of such parallel CFD-applications has to be small compared to the computational cost. Additionally, the computation must be distributed evenly across all CPU-cores. Over the years very efficient communication patterns have been developed, which enable an overlap of communication and computation [2]. Also, a variety of partitioning algorithms that distribute the work on the processors is now available, such as methods based on space-filling curves [4] or graph based methods [1]. The choice of a partitioning strategy depends strongly on the use-case [11] and for different classes of use-cases different techniques for the initial distribution of computation effort are well-known. The parallel CFD-algorithms together with efficient communication patterns and efficient partitioning algorithms led to very powerful CFD-applications [3, 5, 7].

In a next step, the CFD-applications have been extended to much more complex problems like transient multi-scale and multi-phase-flow problems. To simulate such problems, the numeric has been expanded to stabilize the basic numerical approach and cover all physical phenomena of interest. However, such simulations suffer from the fact that the treatment of certain physical phenomena, such as shockwaves is numerically costly. At the same time, the occurence of such phenomena is hard to predict both concerning the precise location within the simulated volume and concerning the exact time. These irregularities cause a mismatch of computational effort for individual MPI-processes. These load imbalances lead to massive performance decreases and also have a negative impact on communication hidding.

1 Numerical Background

As an example for a powerful numerical method for CFD with time dependent additional computational effort we use the discontinuous Galerkin spectral element method (DG SEM). At this point only a short overview is given, for further reading we refer to [7]. Starting point of DG SEM are the compressible Navier-Stokes equations expressed in conservative form

$$\boldsymbol{u}_t(\boldsymbol{x}) + \boldsymbol{\nabla}_x \cdot \boldsymbol{F}(\boldsymbol{u}(\boldsymbol{x}), \boldsymbol{\nabla}_x \boldsymbol{u}(\boldsymbol{x})) = \boldsymbol{0} \quad \forall \boldsymbol{x} \in \Omega \tag{1}$$

where $\boldsymbol{F}$ contains the physical fluxes, Ω is the computational domain, and $\boldsymbol{u}$ is a vector containing the conservative variables. The simulation domain is discretized with hexahedral grid cells. For the actual computation, the grid cells are mapped to a reference element $E = [-1,1]^3$ with coordinates $\boldsymbol{\xi} = (\xi^1, \xi^2, \xi^3)^T$. To derive the discontinuous Galerkin (DG) formulation, the transformed conservative law is multiplied with a test function Φ. The solution in each reference element is approximated by tensor products of Lagrangian polynomials of degree N.

DG schemes are a hybrid of Finite Element schemes and Finite Volume schemes. The solution is approximated by an element-local polynomial basis and the solution is discontinues over element boundaries. Adjacent elements are coupled by fluxes through their interfaces. A disadvantage of height order methods, like the DG SEM, is the introduced instabilities by shock waves traveling through grid cells. If a jump in a grid cell has to be resolved, the high order polynomial in the coarse grid cell generates spurious oscillations. To circumvent this problem Sonntag and Munz [9] introduced an inherent refinement of the discontinuous Galerkin elements into several finite volume cells with a lower order approximation without changing the number of degrees of freedom or the general data structure [9]. To detect the instabilities in the solution they used the so-called Persson indicator [8]. With this approach they overcome the problem with the instabilities but locally generate additional computational effort because a finite volume subscell is more computation intensive then a DG-Element.

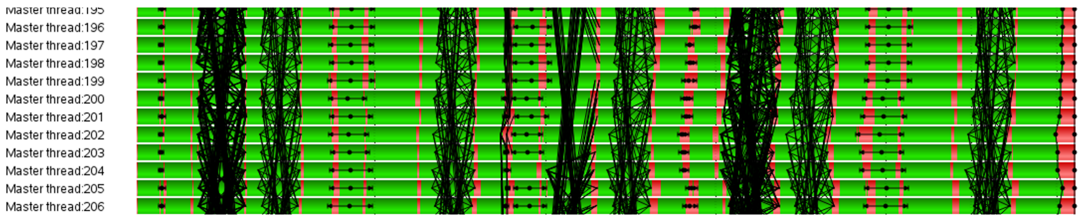
2 Load balancing methodology

In the initial domain decomposition it is not possible to take care of additional local computation induced by transient physical phenomena like traveling shockwaves. During run-time the computation load of the MPI-processes vary. In figure 1 details of two traces of a time-step of a CFD-application is shown. Figure 1a shows a well balanced time step. The ratio of MPI-functions is small and the communication is overlapped by the computation very well. In figure 1b a later time-step is shown. In some MPI-Processes the ratio of time in MPI-functions is much higher then in figure 1a. The MPI-processes with low computational effort have to wait for MPI-processes with huge computational effort. The whole performance of the CFD-application suffer in terms of the load imbalance between the MPI-processes.

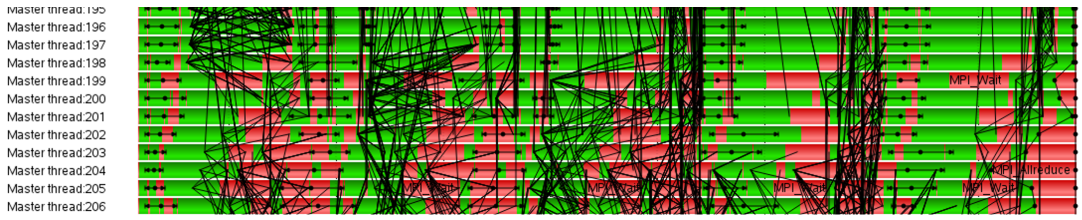
To overcome this problem it is necessary to do a redistribution of the computation effort during run-time to regain a well balanced application. The load balancing process can be divided in two main tasks.

First the optimization problem of calculating the new balanced distribution of the elements over all processes. Second the exchange of the elements between MPI-processes with high load and processes with low load. The challenges in the design of a dynamic load balancing are the requirement for a very low overhead due to the redistribution process and the complicated communication pattern that arises during run-time and depends on the current state of the simulation.

An approach for load balancing based on a space-filling Hilbert curve [6] is investigated. The space-filling curve maps the three dimensional domain decomposition problem to a much simpler one dimensional problem. An optimization algorithm for distributing the computation effort over all MPI-processes, based on the actual computational costs of the elements was developed. The



(a) A well balanced generic test-case



(b) A generic test-case with load imbalance

Figure 1. Detail of a tracing-result of one time-step of a CFD application for a generic test-case on 384 cpu-cores. In 1a a given time-step without load imbalances and in 1b a given time-step with huge load imbalance (green is computation and red are MPI-functions)

one-dimensional structure of the mapped problem reduces the communication effort for calculating the element distribution and also the communication effort for the data exchange between over- and under-loaded MPI-processes. To avoid an expensive all-to-all communication pattern the introduced load balancing methodology makes use of the MPI 3.0 shared memory window. The developed communication pattern reduces the communication only to communication between nodes and not communication between all MPI-processes.

3 Results

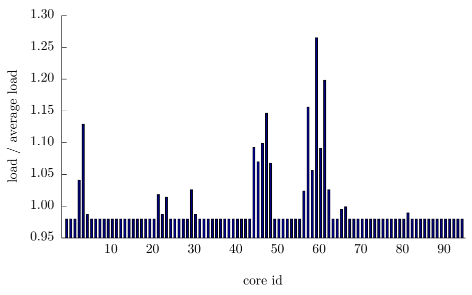
Figure 2 shows the normed load distribution before and after the load balancing. In figure 2a a huge number of the MPI-processes are under-loaded and a few of the MPI-processes are over-loaded. After running the optimization algorithm the load over all MPI-processes is nearly equal as you can see in figure 2b. The new element distribution will eliminate the performance penalty which occurs because of the additional local computation effort.

For a gas-injector with a huge load imbalance, simulations without and with dynamic load balancing were performed on the Cray CX40 (Hazel Hen) at the High Performance Computing Center Stuttgart on 1,200 cores. The dynamic load balancing was executed every 1,000th time step. Figures 3a to 3c show that for the simulation with dynamic load balancing, the domain decomposition chances over the run-time, depending on the additional local computation effort. The dynamic load balancing helps to reduce the run-time significantly.

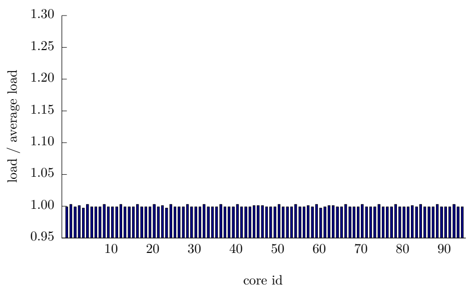
Conclusions

Today's HPC systems generate their performance by facilitating hundreds of thousands of cores. To make use of such highly parallel architectures special CFD-algorithms have been developed and implemented in a variety of CFD-applications. CFD-applications that simulate complex transient problems suffer from locally occurring additional computational effort caused by physical phenomena that are numerically costly. The locally occurring additional computational effort engenders load imbalances between MPI-Processes and decreases the parallel efficiency of CFD-applications. The problems which occur due to the load imbalance have been described and a methodology for dynamic load balancing of CFD-applications has been presented.

First results show how dynamic load balancing can increase the parallel performance. This work shows the necessity of dynamic load balancing for numeric methods which occurring local

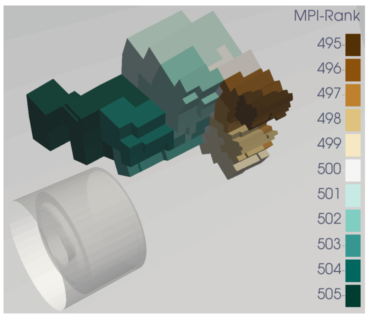


(a) Load distribution over 96 MPI-processes for a generic test-case before load balancing

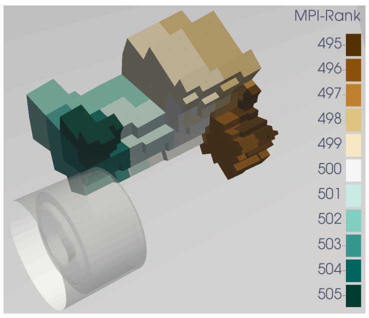


(b) Load distribution over 96 MPI-processes for a generic test-case avter load balancing

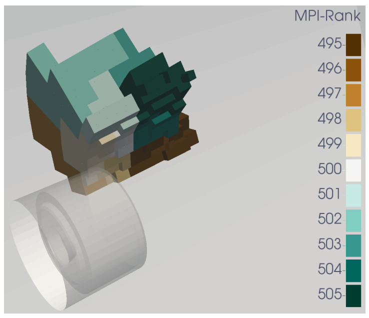
Figure 2. Load distribution over 96 MPI-processes for a generic test-case before (figure 2a) and after (figure 2b) the load balancing



(a) Domain decomposition for the MPI-processes 495 to 505 at timestep T_0



(b) Domain decomposition for the MPI-processes 495 to 505 at timestep T_1



(c) Domain decomposition for the MPI-processes 495 to 505 at timestep T_2

Figure 3. Visualization of the domain decomposition for the MPI-processes 495 to 505 of a gas injector for three different time steps. The simulation was performed on 1,200 cores

additional computational effort. In future, the problem of load balancing will increase with the increasing number of cores in new supercomputers. For this reason, techniques for dynamic load balancing must be further investigated and developed.

References

[1] Family of graph and hypergraph partitioning software. http://glaros.dtc.umn.edu/gkhome/views/metis. Accessed: 2017-09-08.

[2] C. Altmann, A. D. Beck, F. Hindenlang, M. Staudenmaier, G. J. Gassner, and C.-D. Munz. *An Efficient High Performance Parallelization of a Discontinuous Galerkin Spectral Element Method*, pages 37–47. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[3] M. Atak, A. Beck, T. Bolemann, D. Flad, H. Frank, and C.-D. Munz. *High Fidelity Scale-Resolving Computational Fluid Dynamics Using the High Order Discontinuous Galerkin Spectral Element Method*, pages 511–530. Springer International Publishing, 2016.

[4] M. Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing.* Springer Publishing Company, Incorporated, 2012.

[5] M. O. Cetin, A. Pogorelov, A. Lintermann, H.-J. Cheng, M. Meinke, and W. Schröder. *Large-Scale Simulations of a Non-generic Helicopter Engine Nozzle and a Ducted Axial Fan*, pages 389–405. Springer International Publishing, Cham, 2016.

[6] D. Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. *Mathematische Annalen*, (38):459–460, 1891.

[7] F. Hindenlang, G. J. Gassner, C. Altmann, A. Beck, M. Staudenmaier, and C.-D. Munz. Explicit discontinuous galerkin methods for unsteady problems. *Computers & Fluids*, 61:86–93, 2012.

[8] P.-O. Persson and J. Peraire. chapter Sub-Cell Shock Capturing for Discontinuous Galerkin Methods. Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan 2006. 0.

[9] M. Sonntag and C.-D. Munz. *Shock Capturing for Discontinuous Galerkin Methods using Finite Volume Subcells*, pages 945–953. Springer International Publishing, Cham, 2014.

[10] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. Top500 list - june 2017. https://www.top500.org/lists/2017/06/. Accessed: 2017-08-29.

[11] J. D. Teresco, K. D. Devine, and J. E. Flaherty. Partitioning and dynamic load balancing for the numerical solution of partial differential equations. In *Numerical solution of partial differential equations on parallel computers*, pages 55–88. Springer, 2006.