Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Optimising UCNS3D, a High-Order finite-Volume WENO Scheme Code for arbitrary unstructured Meshes

Thomas Ponweiser[a,*], Panagiotis Tsoutsanis[b,†]

[a]*Research Institute for Symbolic Computation (RISC), Johannes Kepler University, Altenberger Straße 69, 4040 Linz, Austria*
[b]*Centre for Computational Engineering Sciences, Cranfield University, College Rd, Cranfield MK43 0AL, United Kingdom*

Abstract

UCNS3D is a computational-fluid-dynamics (CFD) code for the simulation of viscous flows on arbitrary unstructured meshes. It employs very high-order numerical schemes which inherently are easier to scale than lower-order numerical schemes due to the higher ratio of computation versus communication. In this white paper, we report on optimisations of the UCNS3D code implemented in the course of the PRACE Preparatory Access Type C project "HOVE" in the time frame of February to August 2016. Through the optimisation of dense linear algebra operations, in particular matrix-vector products, by formula rewriting, pre-computation and the usage of BLAS, significant speedups of the code by factors of 2 to 6 have been achieved for representative benchmark cases. Moreover, very good scalability up to the order of 10,000 CPU cores has been demonstrated.

Keywords: CFD, WENO, Unstructured meshes, ILES, Turbulance, RANS, Hypersonic, Fortran, MPI, BLAS, MKL

1. Introduction

Unstructured grids have been widely used in science and engineering for their ability to accurately represent complicated geometries in an efficient manner. This particular arbitrariness in terms of the shape of the considered geometric elements such as hexahedral, tetrahedral, prismatic and pyramidal elements and in terms of their unstructured memory pattern poses a number of challenges for the development of numerical methods and computing algorithms especially when high-order of accuracy and excellent computing performance is required.

UCNS3D is a finite-volume CFD code for arbitrary unstructured meshes which employs high-order weighted-essentially-non-oscillatory (WENO) numerical schemes for e.g. LES simulations of canonical flows and RANS simulations of full aircraft geometries during take-off and landing. For a more comprehensive description of the employed computational method, we refer to [1] and [2].

UCNS3D is entirely written in Fortran 95. For parallelisation, UCNS3D can be run in MPI-only or hybrid mode (MPI + OpenMP). In the original code version, only METIS (for mesh domain decomposition) and TecIO (for output in Tecplot format) have been used as external libraries. The new code version additionally uses the Fortran 95 interface to BLAS provided by Intel MKL (usage of other BLAS implementations is possible) for dense linear algebra operations which we identified as computational hotspots.

Preliminary studies of UCNS3D have demonstrated that the code's parallel efficiency is proportional to the spatial order of accuracy of the chosen numerical scheme; in other words higher-order schemes scale better than lower-order ones since the ratio of computation to communication is increasing as the spatial order of accuracy increases. Due to the very satisfying scalability of UCNS3D already in its original version, the main focus of this project was intra-node performance optimisation. In particular, the core WENO reconstruction algorithm was already known as computational hotspot where 92% of the computational time has been spent with dense linear algebra operations, in particular with matrix-vector products and dot-products.

* Principal PRACE expert, E-mail address: thomas.ponweiser@risc-software.at
† Principal investigator, E-mail address: panagiotis.tsoutsanis@cranfield.ac.uk

This white paper is structured as follows: In Section 2 and Section 3, we shortly introduce the benchmark cases and HPC systems which have been used throughout for performance analysis of UCSN3D. Section 4 reports in detail on the applied optimisations and their impact on program performance. In Section 5, we compare the original and new code version with respect to performance and scalability. Finally, in Section 6 we summarise the project outcomes and give a short outlook on possible future improvements of UCNS3D.

2. Benchmark cases

2.1. Taylor Green Vortex

The Taylor-Green Vortex is the first case to be assessed in three dimensions. This case features transition to turbulence in the Taylor–Green vortex (TGV). The TGV has been used as fundamental prototype for vortex stretching and consequent production of small-scale eddies, to address the basic dynamics of transition to turbulence based on DNS, as well as on ILES and classical LES (based on subgrid scale models).
In the present study, the TGV has been selected to assess the dissipation rates of WENO schemes and their dependence upon the type of mesh. The mesh size is considered significantly larger than the Kolmogorov scale, thus the simulations are performed using the 3D compressible Euler equations. The TGV is an incompressible flow that evolves from a two-dimensional initial velocity profile of the form:

$$u(x,0) = \sin(kx)\cos(ky)\cos(kz)\,,$$
$$v(x,0) = -\cos(kx)\sin(ky)\cos(kz)\,,$$
$$w(x,0) = 0$$

With density and pressure being given by:

$$\rho(x,0) = 1,$$
$$p(x,0) = 100 + \frac{\rho}{16}[\cos(2z) + 2\cos(2x) + \cos(2y) - 2]$$

Meshes consisting of up to 2 million elements and 560 million reconstructed degrees of freedom where simulated. The following figure demonstrated the temporal evolution of the isosurfaces of the Q-criterion at the finest tetrahedral mesh using a WENO 6th order scheme.

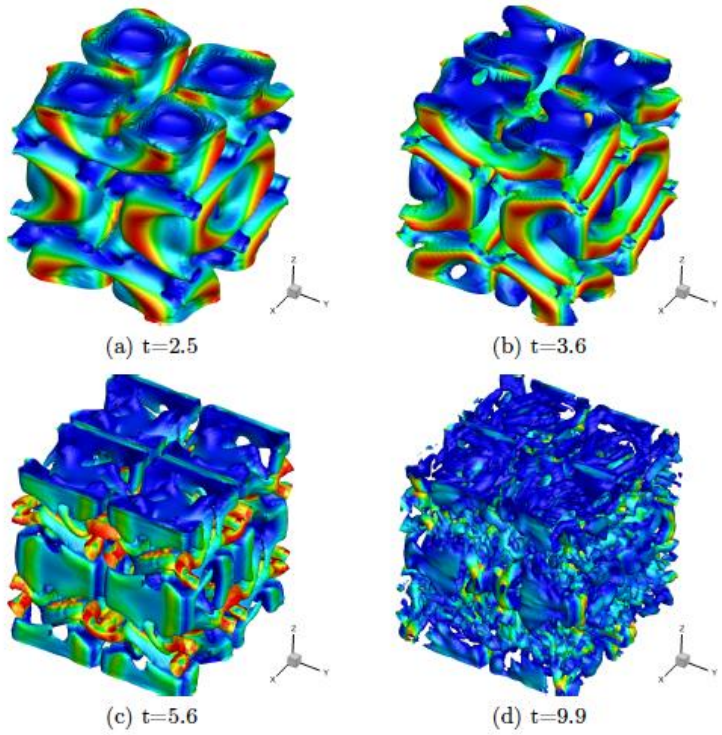

Figure 1: The Taylor Green Vortex case

2.2. Large Eddy Simulation (LES) of the airflow past an airfoil

The second benchmark was the transitional flow past the SD7003 airfoil, which was simulated in the implicit large eddy simulation context (ILES) using a WENO 4th order scheme, at a mesh of 5 million cells for a Mach number of 0.2 and a Reynolds number of 60,000. The instantaneous flow field is illustrated in the following figure where isosurfaces of the Q-criterion is visualised and is used coloured by the velocity magnitude.

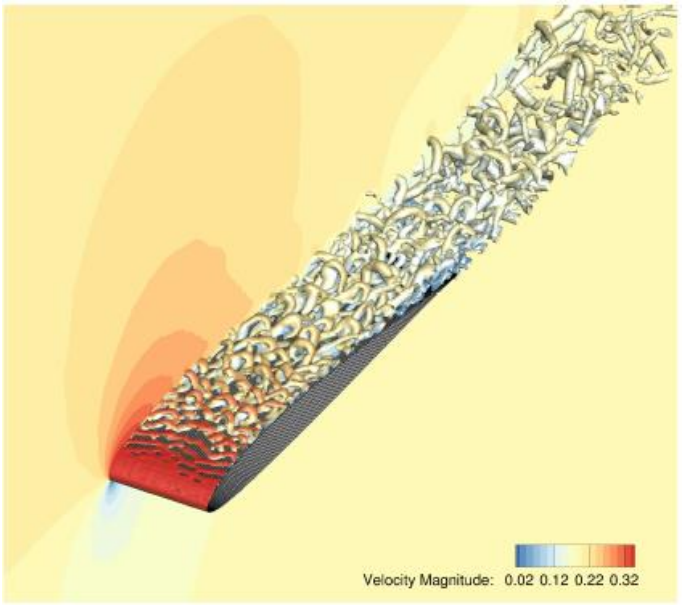

Figure 2: The LES case (transitional flow around an air foil)

2.3. Reynolds-averaged Navier–Stokes (RANS) of an aircraft at transonic cruise conditions

The third benchmark was the transonic flow of the Common Research Model of the 5th Drag Prediction workshop on a hybrid unstructured mesh of 11 million cells using a WENO 5th order scheme at Mach number 0.85 and a Reynolds number of 5 million for constant lift CL=0.5.
The obtained drag coefficient of simulations using 2nd to 5th-order of accuracy in two meshes, is compared with the experimental values. It is noticeable from the figure below that as the order of accuracy is increased the agreement with the experimental value is better, and we accelerate the convergence to a grid-independent solution by using higher-order numerical methods.
The grids and performance improvement in terms of total drag coefficient drag counts is shown in the figure that follows.

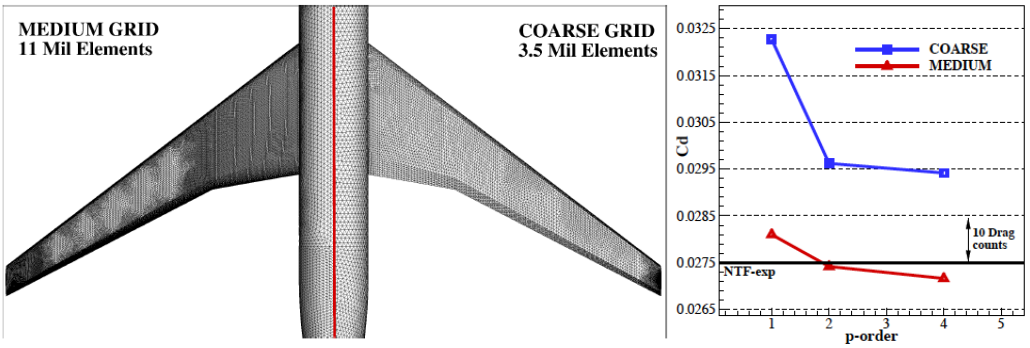

Figure 3: The RANS case: Mesh geometry and total drag coefficient

3. HPC resources

In the course of this project we used the PRACE Tier-1 HPC systems *HAZELHEN* and *SuperMUC*. In the latter case, we were using Sandy Bridge nodes (aka. Phase 1 thin nodes) as well as Haswell nodes (aka. Phase 2 nodes). Table 1 gives an overview of the main characteristics of these systems.

	HAZELHEN	**SuperMUC** **Sandy Bridge nodes**	**SuperMUC** **Haswell nodes**
Computing Site	High Performance Computing Center Stuttgart (HLRS)	Leibniz Supercomputing Centre (LRZ)	Leibniz Supercomputing Centre (LRZ)
Location	Stuttgart, Germany	Garching, Germany	Garching, Germany
Processor Type	Intel Haswell E5-2680 v3	Intel Sandy Bridge E5-2680	Intel Haswell E5-2697 v3
Processor Frequency	2.5 GHz	2.7 GHz	2.6 GHz
Cores per node	24	16	28
Memory per node	128 GB	32 GB	64 GB
Number of nodes	7712	9216	3072
Interconnect	Cray Aries	Infiniband FDR10	Infiniband FDR14

Table 1: Description of HPC resources

4. Performance analysis and optimisation

4.1. Build procedure and Profiling Setup

All performance results reported in this section are based on profiling runs of the pure-MPI variant of UCNS3D on SuperMUC thin nodes (Phase 1) with Intel Vtune Amplifier 2015. As underlying MPI implementation we used Intel MPI 5.1.3. We compiled the code with Intel Fortran compiler version 16.0.3. In order to collect trace data with performance metrics attributed to source-lines, we instructed the compiler to include debugging symbols. The exact compilation flags were `-i4 -r8 -g –DNDEBUG -debug inline-debug-info -O3 -ipo -xHOST`. It is worth noting that enabling inter-procedural optimisation (`-ipo`) has shown to have a quite significant effect on performance (≈ 30% speedup).

The full command-line for profiling UCNS3D with Intel Vtune was:

```
mpiexec -gtool "amplxe-cl -collect hotspots -no-auto-finalize -r [odir]:0-15=node-wide"
        -n [procs] ./ucns3d_p
```

Here, *`odir`* is the profiling output directory and *`procs`* the total number of MPI processes. Note that for reducing the amount of trace data, we were measuring the activity of the first 16 MPI ranks only (i.e. all processes running on the first computation node) and combined the results into one single profiling output directory (`0-15=node-wide`). For this reason, all timings given in this section have to be read as total CPU seconds spent by the first 16 MPI ranks for the given routine or source line.

4.2. Intra-node performance optimisation

Throughout this sub-section, all profiling results correspond to the small Taylor Green Vortex case with 6th order of spatial accuracy, running with a total of 512 MPI processes. In our analyses we focus on the main time-marching part of the code only, i.e. we intentionally disregard initialisation phase (mesh loading, decomposition) and output routines.

First profiling results with Intel Vtune confirmed what already has been seen from manual time measurement and log output: The main computational hotspot was the WENO reconstruction algorithm, implemented in the routine `wenoweights` and accounting for 92% of the runtime. Within this routine the most costly part (≈ 70%) was the computation of a matrix `gradchar` together with a vector `smoothind`, which are computed for each mesh element. Almost all of the remaining computation time (≈ 25%) has been spent within a sub-routine called `gradients_mean_lsq`. In effect, all hotspot operations were dense linear algebra operations, i.e. matrix-vector products and dot products, which have been computed using the Fortran intrinsic `matmul` function.

As a first straight-forward approach, we replaced all identified hotspot matrix-vector products with the equivalent call to `GEMV` provided by the Fortran 95 BLAS binding from Intel MKL, version 11.3. As can be seen in

Table 2, in the columns “Original Code” and “Preliminary Code (GEMV)”, in this way the computation of `gradchar` and `smoothind` could be speeded up by a factor of 5.

Routine	**Original Code**	**Preliminary Code (GEMV)**	**Final Code (GEMM)**
	Time	Time | Speedup	Time | Total Speedup
`wenoweights`	3073 s	1466 s | 2 ×	478 s | 6.5 ×
├ **`gradchar_smoothind`**	2135 s	429 s | 5 ×	197 s | 10 ×
└ **`gradients_mean_lsq`**	780 s	367 s | 2 ×	157 | 5 ×

Table 2: CPU times and speedups for hotspot routine `wenoweights` and two of its sub-routines. The sub-routines have been speeded up by a factor of 10 and 5 respectively, resulting in an overall speedup for `wenoweights` by a factor of 6.5.

For the sub-routine `gradients_mean_lsq`, a simple replacement of `matmul` by `GEMV` did not give any speedup. Here, the unfortunate data layout of the involved matrix objects turned out to be the performance limiting factor. As can be seen in Figure 4, the two hotspot matrix-vector products within `gradients_mean_lsq` involve for each mesh element with index `LL` two matrix objects `ILOCAL_RECON3(I)%STENCILS(LL,:,:)` and `ILOCAL_RECON3(I)%INVMAT(LL,:,:)`. Considering Fortran’s column major data layout convention, this means that the entries of any of these matrices are scattered in memory with a stride equal to the number of mesh elements (of course for contiguous memory access the index `LL` should come last, not first). In a preliminary version of the code, we therefore copied the two matrix objects into two temporary matrices `stencil` and `invmat` with contiguous data layout before passing them to `GEMV` (see Figure 5). In this way, the computational time for the two matrix-vector products dropped from originally 728 seconds to 280 seconds, corresponding to a total speedup of a factor 2 for `gradients_mean_lsq`.

Having a closer look at the code in Figure 5, one might realise that the two matrix-vector products together with the enclosing loop construct can be re-formulated as two matrix-matrix products, namely: `matrix_2` = `stencil`^{T} · `matrix_1` and `SOL_M` = `invmat` · `matrix_2`. By substituting for `matrix_2` and by exploiting associativity, it can be seen that `SOL_M` can be alternatively computed as `SOL_M` = (`invmat` · `stencil`^{T}) · `matrix_1`. In fact it turns out that the matrix product (`invmat` · `stencil`^{T}) does not depend on time and therefore can be precomputed. These observations lead to the final and new version of `gradients_mean_lsq`, which can be seen in Figure 6: The matrix product (`invmat` · `stencil`^{T}) is now precomputed in the code’s initialisation phase for each mesh element index `LL` as `ILOCAL_RECON3(I)%invmat_stencilt(:,:,LL)` (with cache-friendly contiguous data layout). In this way, within `gradients_mean_lsq` only one matrix-matrix product remains, and the routine is now 5 times faster than in the original code version. As additional positive side-effect, also a significant amount of memory could be saved by entirely removing the now unneeded matrix-arrays `ILOCAL_RECON3(I)%STENCILS(:,:,:)` and `ILOCAL_RECON3(I)%INVMAT(:,:,:)` (also the two temporary matrices `stencil` and `invmat` have been removed again in the final code version).

Note that through similar code transformations, i.e. the replacement of matrix-vector products and enclosing loop constructs with matrix-matrix products, we were also able to optimise the subroutine `gradchar_smoothind`, although no further opportunities for precomputation have been detected. As can be seen in Table 2, these additional transformations doubled the speedup for `gradchar_smoothind` from a factor of 5 in the preliminary code version to a factor of 10.

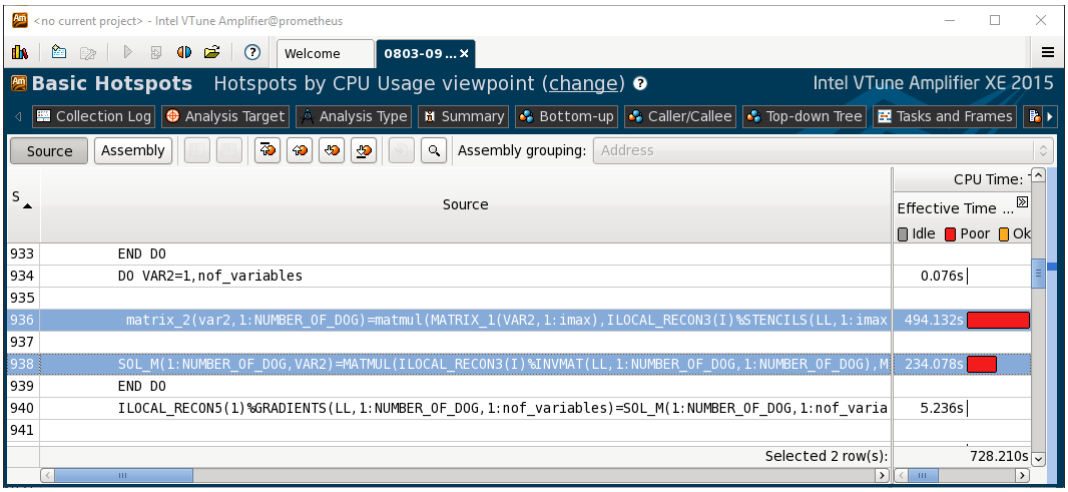

Figure 4: Hotspot operations in the sub-routine `gradients_mean_lsq`. Two matrix-vector products are computed using Fortran's intrinsic `matmul` function. The total computation time is 728 seconds for these two products.

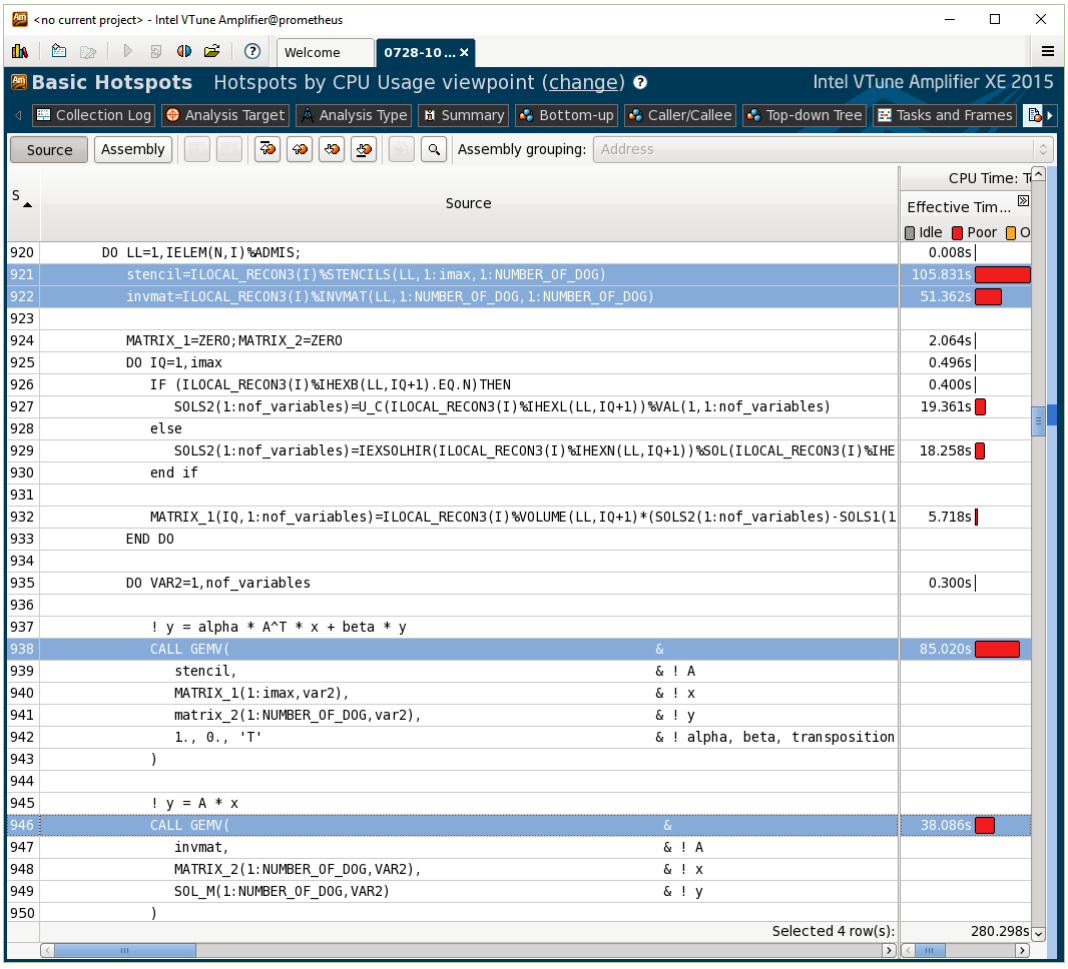

Figure 5: An improved, but still preliminary version of `gradients_mean_lsq`. For improved data access, the matrix objects are copied into temporaries before passing them to `GEMV`. The computation time dropped from originally 728 to 280 seconds, still the two copies are more costly than the actual computation.

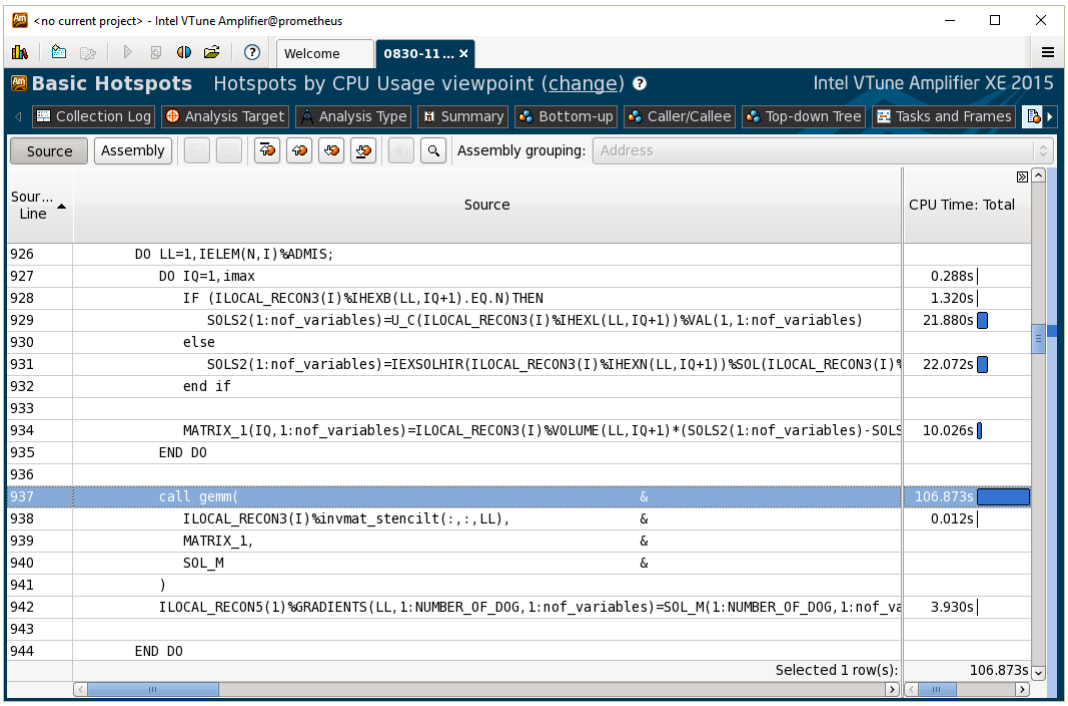

Figure 6: Final version of `gradients_mean_lsq`. Two matrix-vector products and their enclosing loop construct have been replaced by two matrix-matrix products, one of which is precomputed once in the code's initialisation phase. The computational time dropped from originally 728 to 106 seconds, corresponding to a speedup of almost factor 7.

4.3. Optimisation of MPI Communications

In addition to intra-node performance optimisation, we also had a look on the MPI Communication performance of UCNS3D. For this purpose, we used the more communication-intensive LES benchmark case, running with 1024 MPI processes.

The most time-consuming communication routine turned out to be `exhboundhigher`, which is responsible for exchanging the reconstructed, boundary extrapolated values of each Gaussian quadrature point of the direct-side halo neighboring elements (for details see also [1], Section 4.2). In the original code version, this routine accounted for 12% of the runtime of the time-marching part of the code – most of which was spent within `MPI_Barrier`. However, for correctness of the program, explicit synchronisation at that particular point was not needed, as synchronisation occurs implicitly within the `MPI_Sendrecv` calls for data transmission anyway. As can be seen in Table 3, just removing the call to `MPI_Barrier` made the routine approximately 3 times faster. An additional speedup of approximately 15% could be gained by replacing the blocking `MPI_Sendrecv` calls with non-blocking communications, i.e. `MPI_ISend`, `MPI_IRecv` and `MPI_Waitall`.

Note that also the usage of MPI-3 sparse collective operations (i.e. `MPI_Dist_graph_create_adjacent` and `MPI_Neighbour_alltoallw` with tailored MPI Types for avoiding the overhead of manual send and receive buffer packing and unpacking), was considered. However, from the profiling data it got apparent that in fact the time for copying data to and from send and receive buffers was negligible, which is why this approach has not been implemented.

Routine	**Original Code**	**No Barrier**	**Non-blocking comm.**
	Time	Time | Speedup	Time | Total Speedup
`exhboundhigher`	9194 s	3084 s | 3 ×	2642 s | 3.5 ×
├ **`MPI_Barrier`**	9137 s	-	-
├ **`MPI_Sendrecv`**	31.7 s	3059 s	-
└ **`MPI_Waitall`**	-	-	2616 s

Table 3: Runtime of the routine `exhboundhigher`.
By removing an unneeded MPI_Barrier, the routine could be speeded up by a factor of 3.
An additional speedup of 15% could be gained by switching from blocking to non-blocking communications.

5. Results

In this section, we compare the performance of the original version (dashed lines) and new version (solid lines) of UCNS3D running at different core-counts on the SuperMUC Sandy Bridge nodes (displayed in red) and SuperMUC Haswell nodes (in green) as well as HAZELHEN (in blue). On the vertical scale, the average computation time of one single simulation time step is displayed. Note that due to limited CPU time budget, unfortunately measurements for some of the data points are missing.

For the large Taylor Green Vortex case with 2nd order of spatial accuracy, the new code version is 40 to 50% faster than the original one (see Figure 7). The performance benefit gets higher, when the order of the numerical scheme is increased: Looking again at the large Taylor Green Vortex case, this time with 6th order of spatial accuracy, the speedup ranges between factors of 5 and 6.5 (Figure 8). For the Large Eddy Simulation case (Figure 9) as well as for the Reynolds-averaged Navier-Stokes case (Figure 10), which both use 4th order of spatial accuracy, the new code is approximately twice as fast.

Generally, it can be seen that the code scales very well up to several thousand cores.

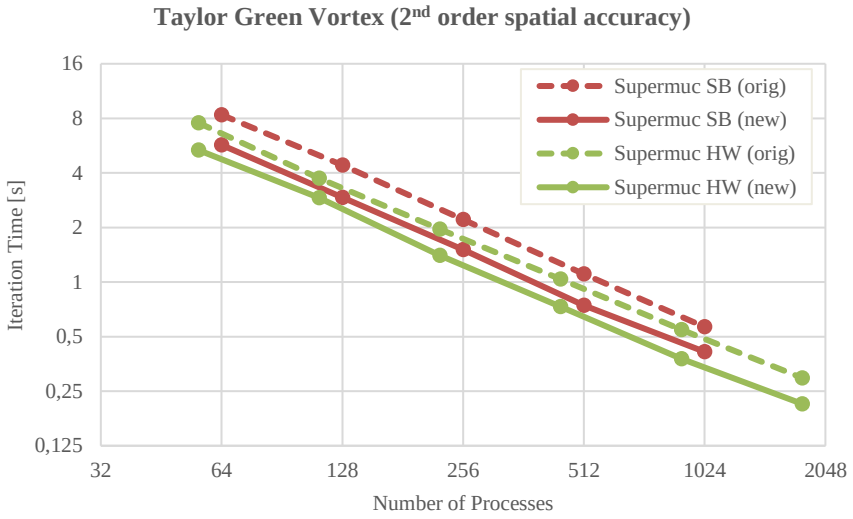

Figure 7: Strong scalability results for the Taylor Green Vortex case with 2nd order of spatial accuracy. A speedup between 40 and 50% has been achieved.

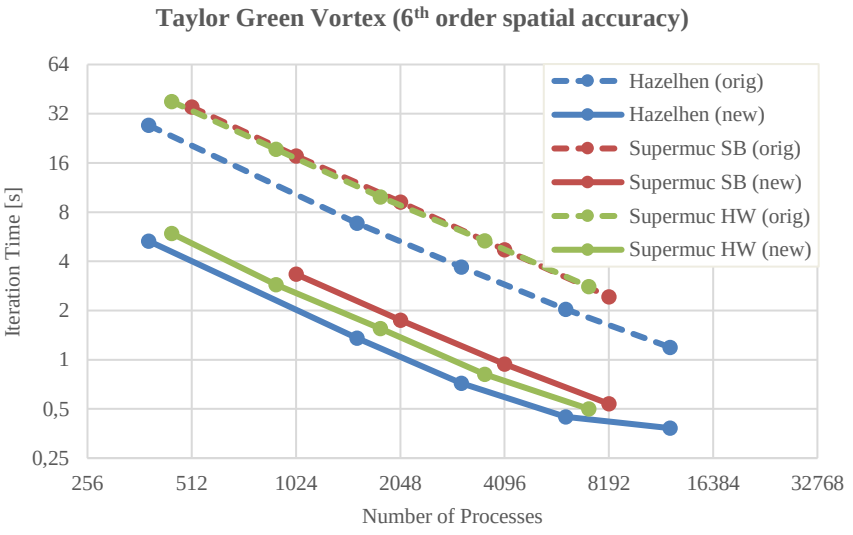

Figure 8: Strong scalability results for the Taylor Green Vortex case with 6th order of spatial accuracy. The achieved speedup factor for the new code version ranges between 5 and 6.5.

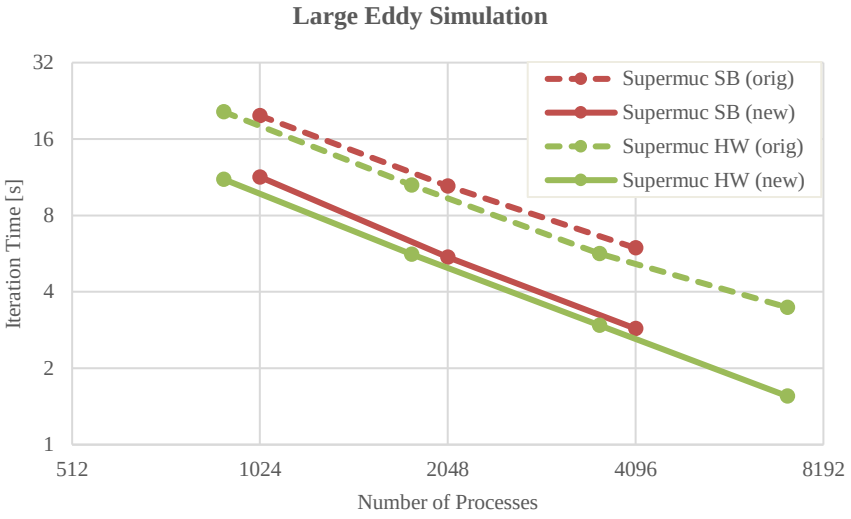

Figure 9: Strong scalability results for the LES case, using 4th order of spatial accuracy. A consistent speedup of factor 2 can be observed for the new code version.

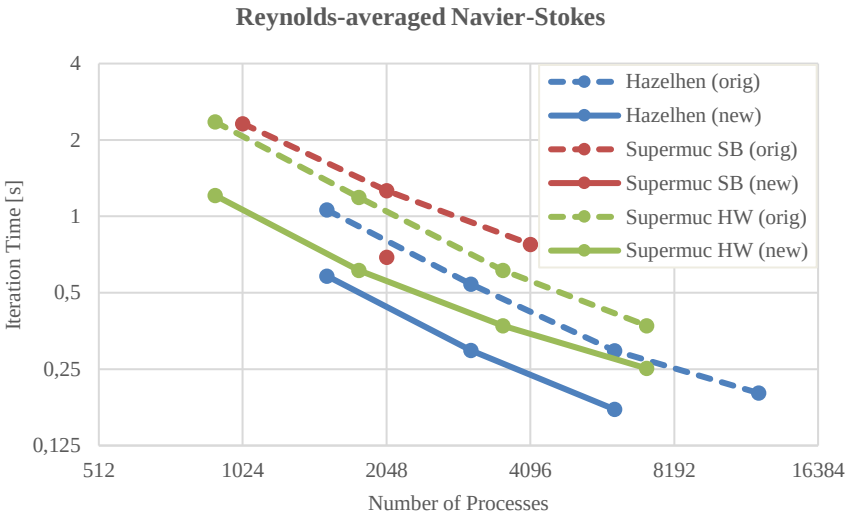

Figure 10: Strong scalability results for the RANS case, using 4th order of spatial accuracy. The speedup factor for the new code version is almost 2.

6. Conclusion

In this project, the performance of the CFD code UCNS3D has been improved significantly (1) by using BLAS for dense linear algebra operations instead of Fortran's intrinsic functions, (2) by reformulating loops of matrix-vector products to matrix-matrix products, (3) by identification and precomputation of time-independent matrix-matrix products and (4) by switching from blocking communications with explicit barrier synchronization to non-blocking communications with implicit synchronisation for halo data exchange. The performance benefit of the new code version grows as the accuracy order of the considered numerical scheme is increased. For 2nd order we observe a speedup of up to 50%, for 4th order a speedup of factor 2 and for 6th order a speedup of factor 5 and higher. The scalability of the code is very satisfactory up to core counts in the order of 10-thousand cores.

6.1. Outlook

Due to constrained time and effort, the performance of the hybrid (MPI+OpenMP) variant of UCNS3D has not been investigated and optimised in the scope of this project. This is still left open for possible future developments on UCNS3D or even a follow-up project. Moreover, in order to increase the scalability of UCNS3D to the order of 100-thousand cores, it will as well be necessary to work on the initialisation and IO routines of the code, part of which are still sequential. Additional optimisation potential in the initialization phase of the code may also arise from using LAPACK for performing e.g. QR factorisations.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU's Horizon 2020 research and innovation programme (2014-2020) under grant agreement 653838. We acknowledge that the results in this paper have been achieved using the PRACE Research Infrastructure resources *HAZELHEN* at the High-Performance Computing Center Stuttgart (HLRS), Germany and *SuperMUC* at the Leibniz Supercomputing Centre (LRZ) in Garching near Munich, Germany.

References

[1] Tsoutsanis P, Antoniadis AF & Drikakis D (2014) WENO schemes on arbitrary unstructured meshes for laminar, transitional and turbulent flows, Journal of Computational Physics, 256 254-276.

[2] Tsoutsanis P, Titarev VA & Drikakis D (2011) WENO schemes on arbitrary mixed-element unstructured meshes in three space dimensions, Journal of Computational Physics, 230 (4) 1585-1601.