2nd International Workshop on High-Order CFD Methods
May 27-28, 2013 at NH Hotel Köln City

A high-order accurate GPU accelerated flow solver

F. Spiering, T. Leicht

German Aerospace Center, Member of the Helmholtz Association,
Institute of Aerodynamics and Flow Technology,
Lilienthalplatz 7, 38108 Braunschweig
frank.spiering@dlr.de

Code description

With the increasing capabilities and performance of modern Graphics Processing Units (GPU) in different fields of High Performance Computing (HPC) over the last years, it became desirable to use them for Computational Fluid Dynamics (CFD) as well. In contrast to CPU architectures, a current GPU resembles a vector processor architecture with hundreds or thousands of small computing cores, allowing for massively parallel execution of threads. To exploit the performance of the GPU optimally, one has to choose numerical algorithms, which allow for a fine-grained parallel implementation with independent data accesses to avoid race conditions.

To investigate the potential of the GPU, a prototype CFD solver based on a nodal discontinuous Galerkin discretization of the compressible Navier-Stokes equations has been developed as CPU code which was then used as a basis for a GPU implementation. The scheme employs a collocation approach using the Gauss-Lobatto-Legendre points on hexahedral elements. Convective terms are treated by Roe's upwind scheme, while diffusive terms are discretized using the second scheme of Bassi and Rebay (BR2). To save on memory requirements, the time integration is performed by means of an explicit five-stage fourth-order low-storage Runge-Kutta scheme. All computations use tri-cubic polynomial approximations and are thus formally fourth order accurate which is suited well to the GPU architecture properties. As the code is of prototypical character so far, it features no boundary treatments like walls or farfield. The periodic boundary treatment needed for this test case is realized by marking the elements on the opposite boundaries of the grid as adjacent to each other. The geometrical representation of the grid elements is iso-parametric, though this would not be required for this specific test case, which employs a Cartesian mesh.

To port the flow solver to the GPU architecture, the computation loops had to be rearranged, which results in a heavily increased computational effort, but allows for independent data accesses of the parallel threads. To minimize the data transfer between GPU device and host system, the conservative variables of the flow field are only transferred to the host in a user-defined time step interval. Derived data like kinetic energy and vorticity is then calculated on the CPU of the host system.

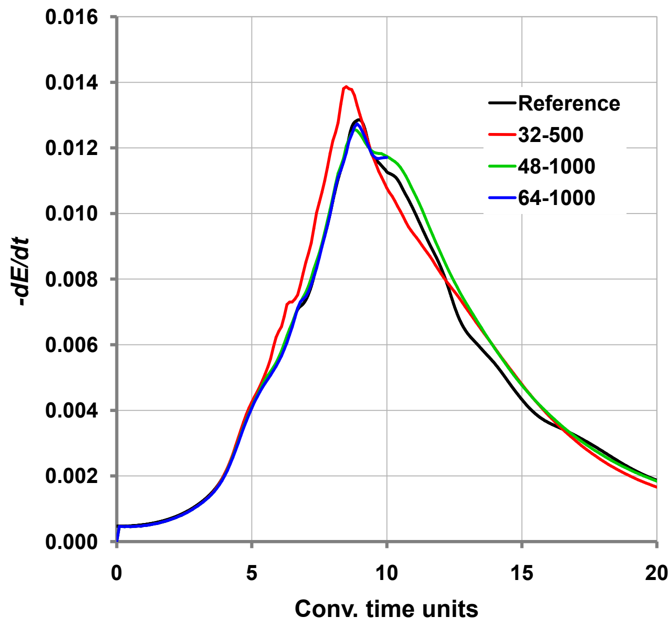
Case summery

The flow simulations of this test have been performed on a NVIDIA GeForce 660 Ti™ graphics card based on the GK104 architecture. This graphics card is an upper middle-class device in the current line-up of desktop gaming hardware. All floating point computations on the GPU are performed in single precision, for the double-precision floating point performance of this GPU is only 1/24 of the single-precision performance. The CPU code is run for comparison on an Intel Core i5-2400S™ quad-core CPU with Sandy Bridge architecture.

Up to now the test case was simulated on grids with different resolutions of 32 and 48 grid elements per dimension. With a fourth-order accurate approach the equivalent resolution is 96 and 144 mesh spacings per dimension. A simulation on a grid with 64^3 elements (192 spacings) has been performed up to a simulated time of ten convective time units. This results in overall numbers of points from approx. 2.1 millions to 16.8 millions. The time step size used in the simulations was $dt = 1/500\ tc$ for the coarse grid and $dt = 1/1000\ tc$ for the finer grids.

Results

The results of the simulations using the GPU code are illustrated in the figures below. Figure 1 shows the kinetic energy dissipation rate throughout the simulated time of 20 convective time units whereas the temporal evolution of the enstrophy is illustrated in figure 2. Both plots clearly underline the necessity of a sufficient grid resolution. On the finest grid the temporal local location of the peak values of both kinetic energy dissipation rate and enstrophy is in good accordance to the reference data, although the peak values are slightly underpredicted.



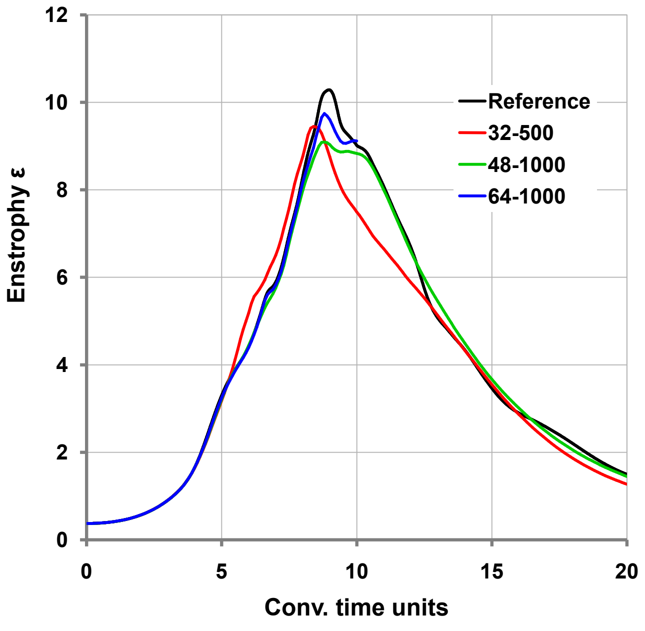


Figure 1: History plot of kinetic energy dissipation rate

Figure 2: History plot of enstrophy

In figure 3 the vortical structures in a section of the periodic plane $x/L = -\pi$ are shown as a contour plot of the vorticity norm. The block-like data-distribution may result from an insufficient resolution of the grid and will be topic of further investigations.

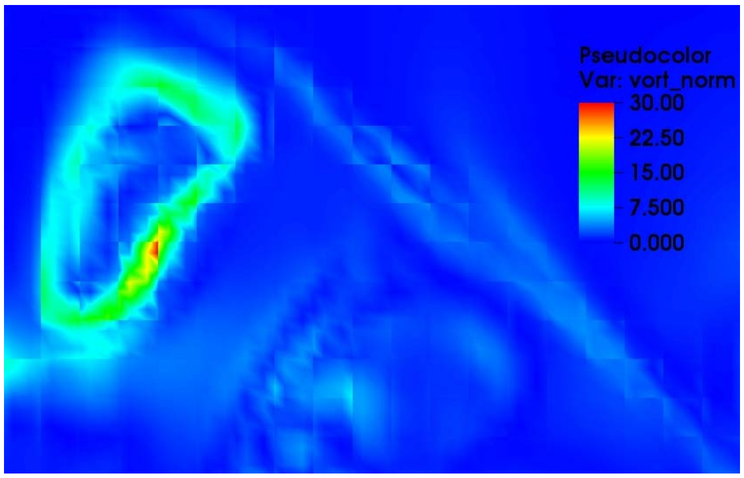


Figure 3: Contour plot of vorticity norm in a section of the plane *x/L* = *-π*

TauBench comparison

To provide a meaningful comparison of the performance between CPU and GPU code the power efficiency should be taken into account, for the power consumption of the hardware is one of the crucial aspects of operating HPC systems today. Therefore this comparison is split to a comparison of the original CPU code against TauBench (see table 1) and a comparison between the CPU code and the GPU implementation. All performance measurements have been performed on the grid with 32 elements per dimension. The runtime of the whole simulation of 20 convective time units is extrapolated from a limited number of 100 time steps without data output. Additionally the GPU code is run without intermediate data transfer to the host system.

Test	Sequential run time	Work units
TauBench	7.84 sec.	1.0
CPU code (32^3 elem., scaled to 10000 time steps)	255200 sec.	32551.0
CPU code (64^3 elem., scaled to 20000 time steps)	4083200 sec.	520816.3

Table 1: Performance of original CPU code normalized with TauBench

With respect to the power consumption of the GPU device and the CPU, it is assumed that one GPU device roughly equals 2.5 of the CPU used for this work. Therefore the performance of the GPU code is compared to the linearly extrapolated performance of ten CPU cores. As shown in table 2, the GPU implementation delivers a noticeable speed up compared to the CPU code which therefore results in a higher power efficiency.

Test	Run time	Speed up
CPU code (32^3 elem., 100 time steps, scaled to 10 cores)	~ 255 sec.	
GPU code (32^3 elem., 100 time steps, single GPU)	103 sec.	2.47

Table 2: Performance comparison between original CPU code and GPU implementation

Outlook

For the time until the Workshop it is envisaged to perform computations on finer grids up to the limit of memory requirements (72^3 or 80^3 elements) and evaluate the simulations to generate the mandatory result data. Furthermore the possibility of reordering the loops and data accesses back to the form of the CPU implementation should be investigated to decrease the computational effort. Additionally an implementation of a fifth- or sixth-order accurate scheme is planned.

For the future after the workshop it is planned to implement different boundary treatments to be able to simulate various flow scenarios and to realize a domain decomposition-like parallelization of the code to utilize multiple GPUs.

2nd International Workshop on High-Order CFD Methods
May 27-28, 2013 at NH Hotel Köln City

F. Spiering, T. Leicht

German Aerospace Center, Member of the Helmholtz Association,
Institute of Aerodynamics and Flow Technology,
Lilienthalplatz 7, 38108 Braunschweig
frank.spiering@dlr.de

Addendum
TC 3.5 Computational effort of various simulations

The purpose of this document is to guide the reader through the calculation of meaningful data of the computational effort of the GPU code as the recommended calculation of work units is not suitable to GPU codes.

If one wishes to skip the calculation steps, the work unit equivalent numbers with respect to the four simulations performed are given in bold digits in table 4 at the end of this document.

Remarks on the GPU code implementation

The GPU code is based on a CPU implementation of the Discontinous-Galerkin method for under-resolved DNS computations and incorporates the same numerical algorithms. To port the original CPU implementation to the GPU, some adaptations had to be done to utilize the massive parallelism of the GPU. These adaptations to the GPU architecture did not only include optimizations to benefit from the massive parallelism, but also a rearrangement of the calculation operations which results in heavily more computational effort, but it was necessary to avoid race conditions.

The original CPU implementation was implemented in a straight-forward manner and did not include performance tuning like explicit usage of SIMD units. Nevertheless it was compiled using the optimization flags -O3 -mss2. All computations on the GPU have been performed with only single-precision numbers, for the available GPU device offers only very limited double-precision performance, as it is not a professional HPC GPU device.

Calculation of GPU speed-up

To provide a meaningful performance comparison of the GPU code versus the CPU implementation and reasonable figures of the computational effort for this test case, the power consumptions of both CPU and GPU device are taken into account, for power consumption is one of the main costs in today's High Performance Computing. Therefore the runtime of the GPU code in the CFD simulations is compared to extrapolated runtimes of the CPU code. These CPU runtimes are extrapolated from a small benchmark test, as there were no resources available to run the full simulations using the CPU code.

The CPU benchmark test incorporated a grid of only 32^3 elements and only 100 time steps. Afterwards the runtime for the full simulation is calculated by scaling to the full number of time steps (20 convective time units with 1000 steps per convective time unit) and the grid size used in the simulation.

The benchmark run (32^3 elements, p=3, 100 steps) of the CPU code was measured at 2509 seconds.

The benchmark run (32^3 elements, p=5, 100 steps) of the CPU code was measured at 7573 seconds.

These numbers have been used to extrapolate the runtimes of the full simulations.

Simulation	**Number of elements/ polynomial degree**	**Extrapolated runtime of CPU code**
N48-P3	48*48*48 / 3	1693778 sec
N64-P3	64*64*64 / 3	4014880 sec
N72-P3	72*72*72 / 3	5716499 sec
N32-P5	32*32*32 / 5	1514720 sec

Table 1: Extrapolated runtime of the CPU code (not actually performed)

The runtime of the GPU code was measured during the simulations performed for the upcoming workshop.

Simulation	**Number of elements/ polynomial degree**	**Measured runtime of GPU code**
N48-P3	48*48*48 / 3	61542 sec
N64-P3	64*64*64 / 3	147208 sec
N72-P3	72*72*72 / 3	209107 sec
N32-P5	32*32*32 / 5	85266 sec

Table 2: Measured runtime of the GPU code from the performed simulations

To compare the performance of CPU and GPU the power consumption of the computing devices is taken into account. The CPU used in this calculation was an intel Core i5-2400S (Sandy Bridge architecture) with 4 cores, 2.5 GHz (fixed, SpeedStep & Turbo deactivated) and a TDP of 65 watts. The GPU used was a nVidia Geforce 660 Ti (GK104 architecture) with a TDP of 150 watts (TDP taken from data sheets from the respective manufacturers).

Therefore it is assumed that a 10-core of the Sandy Bridge architecture and the clock frequency given above would consume the same power as the GPU used here. Hence, the speed-up of the GPU is calculated by comparing against the CPU performance of ten cores assuming a parallel efficiency of 100%.

Simulation	**GPU runtime**	**CPU runtime**	**GPU speed-up versus 10 CPU cores**
N48-P3	61542 sec	1693778 sec	2.75
N64-P3	147208 sec	4014880 sec	2.73
N72-P3	209107 sec	5716499 sec	2.73
N32-P5	85266 sec	1514720 sec	1.78

Table 3: Power consumption based speed-up of the GPU code versus CPU code

The first three simulations show a consistent power consumption based speed-up of about 2.73. The fourth simulation with a polynomial degree of p = 5 shows only a speed-up of 1.78 due to two different reasons:

1. The rearranged computation loops on the GPU implementation show a worse scaling behavior for higher polynomial degrees due to an extra inner loop over the number of support points per direction in the gradient evaluation of the viscous terms.

2. The number of support points per element ($6^3 = 216$), which is also the number of threads per threadblock used in the GPU code is not optimal for the GPU thread scheduling (The number of $4^3=64$ threads for polynomial degree p = 3 was chosen because it is suited so well for the GPU).

Calculation of work unit equivalents

To calculate the GPU work unit equivalent numbers the work units for the CPU code are calculated as described in the test case description and then scaled by the power consumption based speed-up.

The runtime of the TauBench code was measured at t = 8.211 seconds.

Simulation	Extr. CPU runtime / CPU work units	GPU speed-up	GPU work unit equivalent
N48-P3	1693778 sec / 206282	2.75	75011
N64-P3	4014880 sec / 488964	2.73	179108
N72-P3	5716499 sec / 696200	2.73	255018
N32-P5	1514720 sec / 184474	1.78	103637

Table 4: GPU work unit equivalents for the GPU code based on extrapolated runtime of the CPU code and the power consumption based speed-up

The bold number give the final work unit equivalent numbers for the simulations performed with the GPU code.

A high-order accurate GPU accelerated flow solver

2nd International Workshop on High-Order CFD Methods
May 27th-28th, 2013 – Köln

F. Spiering, T. Leicht
DLR – Institute of Aerodynamics and Flow Technology

Outline

- Overview of flow solver
- Details of GPU implementation
- Exemplary test case results
- Performance evaluation
- Conclusion, Outlook

Overview of flow solver

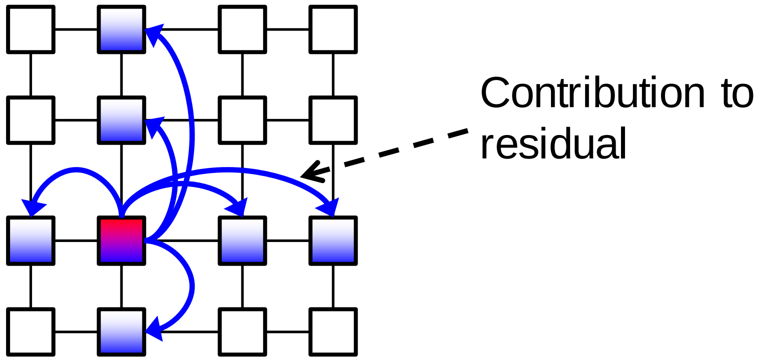
- Discontinous Galerkin discretization of compressible Navier-Stokes equations
- Collocation approach on Gauss-Lobatto-Legendre points
- Roe upwind scheme for convective terms
- BR2 scheme for diffusive terms
- Isoparametric representation of geometries
- Five-stage fourth-order 2N-storage RK scheme for global time stepping

- GPU implementation based on CPU code
  (Same algorithms, CPU code as performance reference)
- Code supports only hexahedral elements
- Only periodic boundaries

Element flux evaluation (CPU)

🟥 Compute residual contrib.

🟦 Add residual contrib.



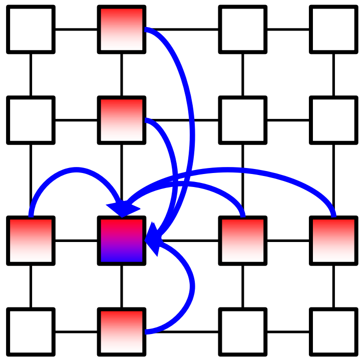
Finite element

- CPU: For each support point, compute residual contribution and add value to affected other support points

Element flux evaluation (GPU)

🟥 Compute residual contrib.

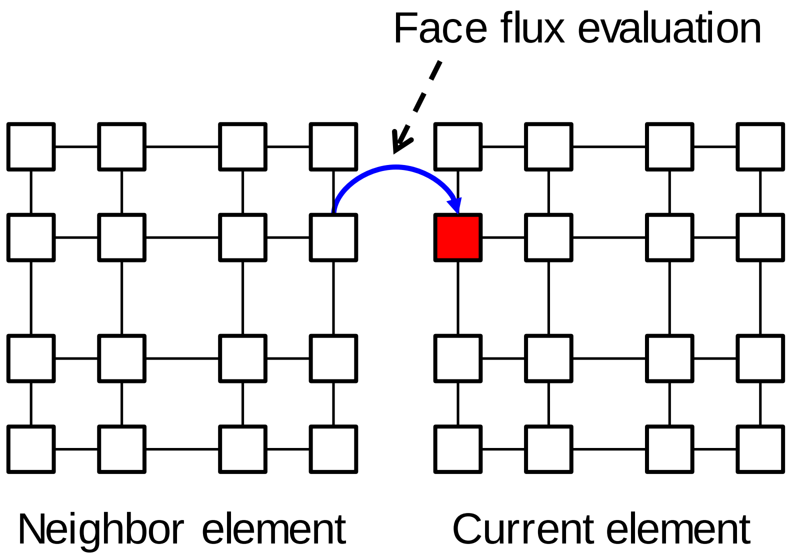
🟦 Add residual contrib.



Finite element

- CPU: For each support point, compute residual contribution and add value to affected other support points
- GPU: For each support point, compute and sum up residual contributions from all contributing support points
  - Significantly more computational effort than CPU implementation
  - Avoids race conditions in parallel execution

Face flux evaluation (CPU)



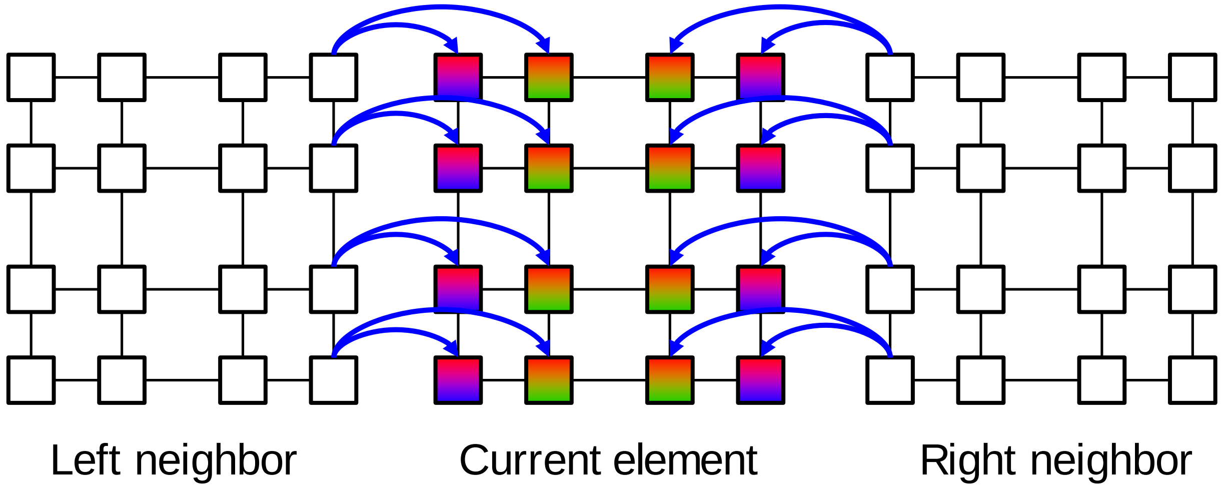
- CPU: For each face point, evaluate face flux with neighbor point

Face flux evaluation (GPU)

🟥 Face flux evaluation

🟦 Masking coeff. = 1.0

🟩 Masking coeff. = 0.0



- CPU: For each face point, evaluate face flux with neighbor point
- GPU: For each point, evaluate face flux, then multiply with masking coefficient (1.0 at face points, 0.0 at inner points)
  - Unneccessary operations at inner points
  - Avoids diverging code paths in parallel execution

Grids

- Computations on GPU (nVidia Geforce 660 Ti)
  - 48^3 elements, $p = 3$, $\Delta t/tc = 0.001$
  - 64^3 elements, $p = 3$, $\Delta t/tc = 0.001$
  - 72^3 elements, $p = 3$, $\Delta t/tc = 0.001$
  - 32^3 elements, $p = 5$, $\Delta t/tc = 0.001$

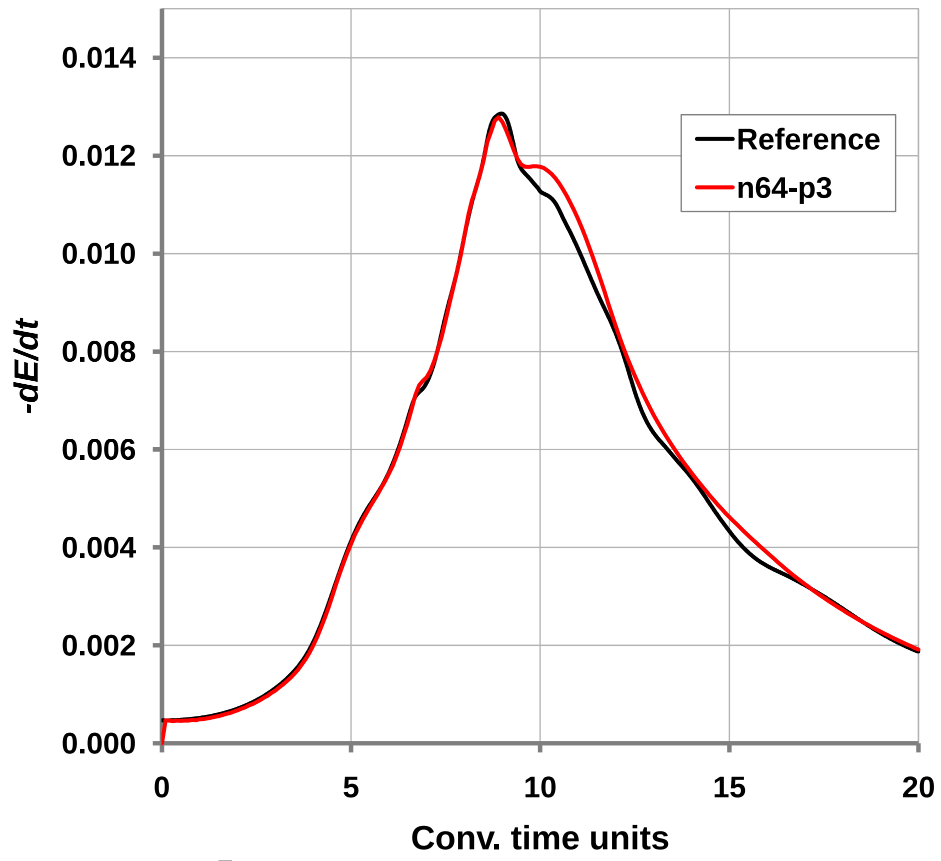
- Computations on CPU (Intel Core i5-2400S)
  - 32^3 elements, $p = 3$, $\Delta t/tc = 0.001$, 100 time steps
  - 32^3 elements, $p = 5$, $\Delta t/tc = 0.001$, 100 time steps

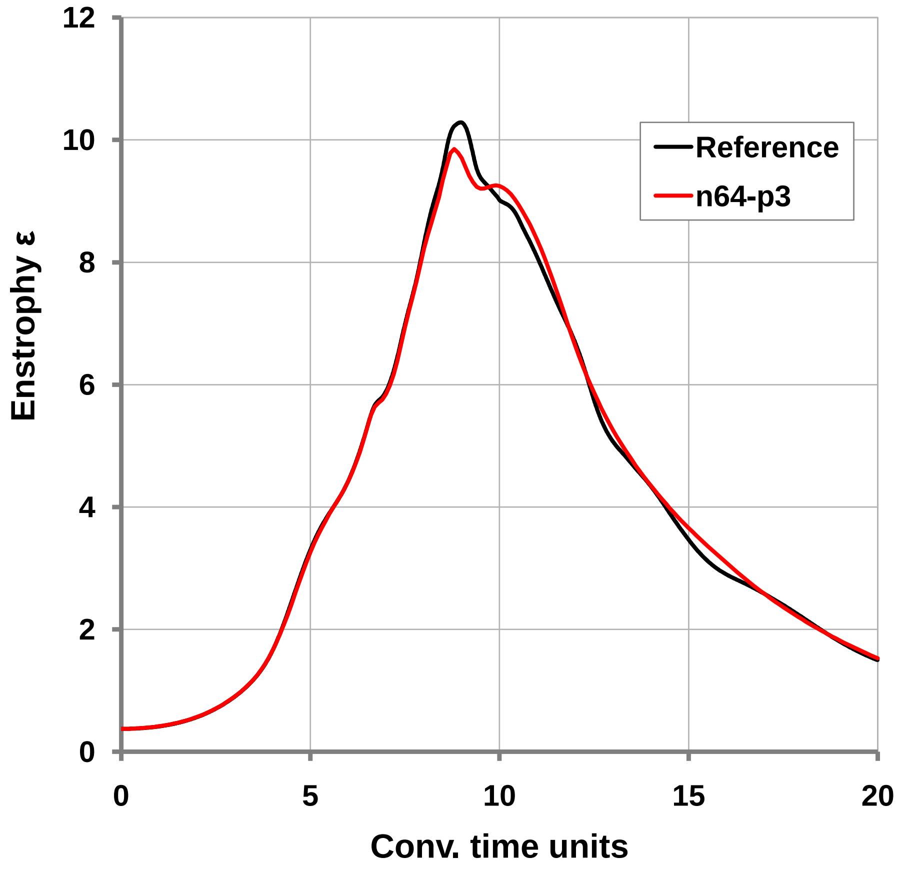
  (Used as performance reference – runtime for larger grids can be scaled from these tests)

Remark: Single-precision calculation on GPU
Very low DP performance on GK104 (Gaming-optimized GPU)

Exemplary test case results

Test case: 64^3 elements, p=3, 1000 time steps / tc

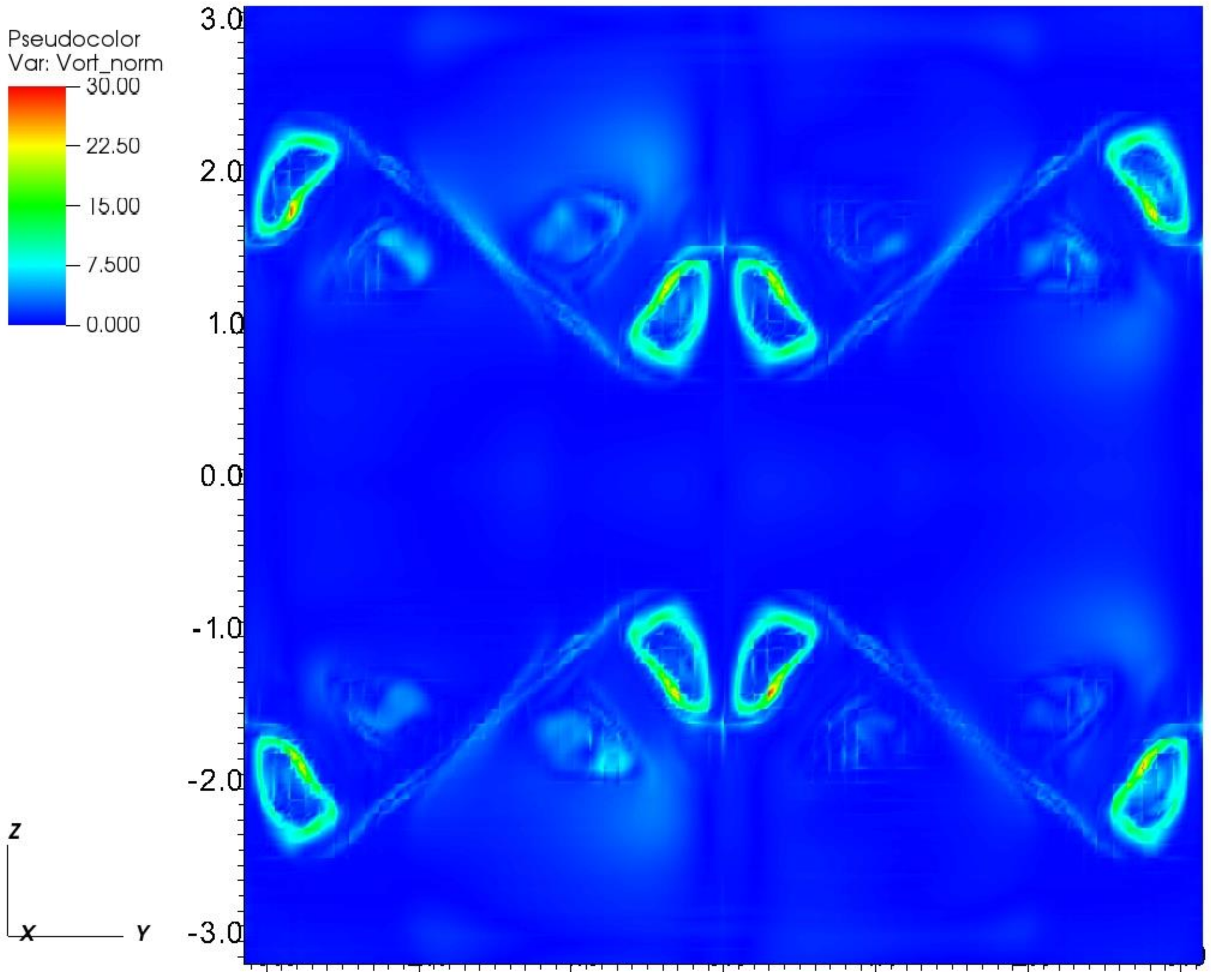




Exemplary test case results

Test case:
- 64^3 elements, p=3
- 1000 t.s. / tc

Plot of normalized vorticity
at t/tc = 8
in periodic plane x = 0



Performance evaluation

- Comparison between different architectures
  - CPU: Intel Core i5-2400S (Sandy Bridge, 4 cores, TDP 65 watts)
  - GPU: nVidia Geforce 660 Ti (GK 104, TDP 150 watts)

- Power consumption is a major part of the costs of HPC
  - Performance evaluation based on TDP of compute devices

→ ~10 CPU cores equivalent to GPU

Performance evaluation - GPU vs. CPU

Test case: 64^3 elements, p = 3, 20 tc, $\Delta t/tc = 0.001$

- **CPU** runtime (seq.) (Runtime is scaled from test with 32^3 elements and less time steps)	4,014,880 sec (~46.5 d)
- **CPU** walltime of 10 cores (Assumption 100% parallel efficiency)	401,488 sec (~4.65 d)
- **GPU** runtime: (Measured time of test case)	147,208 sec (~40.9 h)
- **GPU speed-up**: (vs. 10 cores - Equal power consumption)	~ 2.73

Performance evaluation - GPU equivalent work units

Test case: 64^3 elements, p = 3, 20 tc, $\Delta t/tc = 0.001$

- **TauBench** runtime:	8.211 sec
- **CPU** runtime (seq.) (cf. previous slide)	4,014,880 sec (~46.5 d)
- **CPU** work units	488,960
- **GPU** equivalent work units (CPU w.u./ GPU speed-up)	179,100

Conclusion

- DG method well suited for GPU architectures
  - Implementation not difficult (concept of thread parallelization needed)
  - Significant speed-up possible

Outlook

- Support of multiple GPUs
- Implementation of boundary conditions (Viscous wall, Farfield, Symmetry …)

Thank you for your attention!