Technische Universität Braunschweig

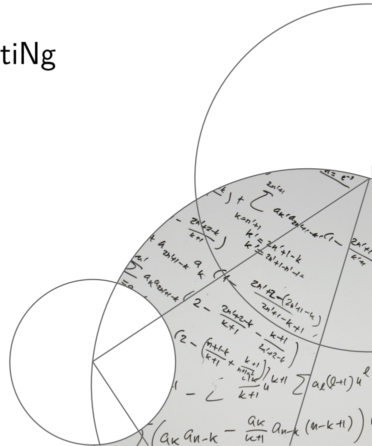
Carl-Friedrich Gauß Faculty

GPGPU acceleration in OpenFOAM

Northern germany OpenFoam User meetiNg
Braunschweig Institute of Technology

Thorsten Grahs
Institute of Scientific Computing/move-csc

2nd October 2013



Overview

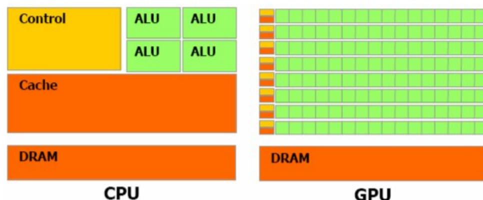
1. HPC on GPGPUs
2. GPU-plugins f. OpenFOAM
3. Comparison CPU-GPU

Why GPGPUs are perfect for HPC?

CPU vs. GPU

Chip-Design

- CPU is optimized for serial tasks (single thread)
- GPU is optimized for massive parallel data handling (multiple threads)
- GPU does not care if pixel data has to be handled (tessellation, transformation, rendering)
- . . . ... or scientific calculation has to be performed.



Single instruction multiple threads

Massive parallel data throughput

High demand on computational power (real time rendering)

- Programing model inspired by vector computers (SIMD)
- Goal: Work off as many threads in parallel as possible
  $\Rightarrow$ Through-put orientated approach
- Accomplished by:
  – Many Arithmetic Logical Units
  – High clock rate of the data bus

$\Rightarrow$ **Highly** suitable for massive parallel computing

NVIDIA Quadro 6000

NVIDIA Quadro 6000

Affordable computing cluster in your workstation

- CUDA cores: 448
- GFlops (SP): 1030.4
- GFlops (DP): 515.2
- Frame buffer: 6 GB GDDR5
- Memory bandwidth: 144 GB/s

Cost: ~ 3.500 €(4,800 $)

GPGPU accelerated CFD

GPGPU-plugins for OpenFOAM

Make use of one of this libraries

- Open Source
  - ofgpu v1.0 Linear solver library (Symscape)
  - CUFFlink (CUda For Foam Link) Dan Combest
  - speed-IT classic (Vratis free version)
- Commercial (not tested)
  - Culises (FluidDyna)
  - Speed-IT 2.3 (Vratis)

Proceeding

- Compile the plugin
- Check in the library (controlDict)

```
functions {
  cudaGpu
  {
     type cudaGpu;
     functionObjectLibs ( " gpu " );
     cudaDevice 0;
   }
}
```

- Declare solver (fvSolutions)

```
p {
    solver   PCGgpu;
    preconditioner  smoothed_aggregation;
    tolerance 1e-06;
    relTol  0.01;
}
```

Cuda side

Open Libraries mostly make use of

- CUSP
  library for sparse linear algebra and graph computations on CUDA
  - Linear Solver on GPU – CG, BiCG, BiCGstab, GMRES
  - Sparse Matrix Formats – CSR, DIA, ELL, HYP
- Thrust
  parallel algorithms library which resembles the C++ Standard Template Library (STL).

Commerial libraries based on

- own solver implementation in CUDA

... so far

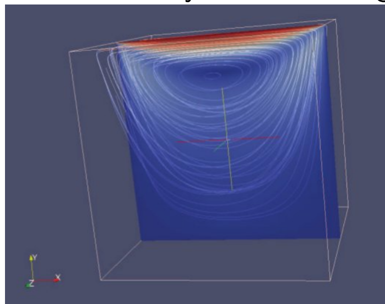
Very nice (easy use) but

where's the beef???

- What is the acceleration one can get?
- With how many CPUS/GPGPUS (Hardware)
- Does this pay off?

Validation

Coming to numbers one mostly see something like this:



- Test case lid-driven cavity
- with very small residuals i.e (1e-15) in order to keep the matrix solver busy.

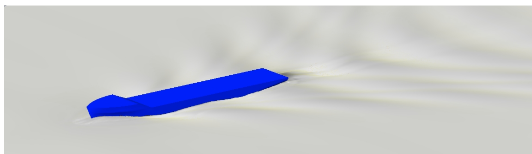
but who needs a lid-driven cavity...

Speed-up for lid-driven cavity (icoFoam solver)

Hardware	512000	1 Mill cells	remark
CPU only	1.00	1.00	
ofgpu1.0	3.03	3.35	Single Precision
cufflink	fpe	fpe	OF 1.6ext.
SpeedITclassic	2.74	3.24	
SP (tubs)	1.32	1.54	Student project TUBS
8 CPUs	2.56	2.05	

How does these plugins perform on real test cases?

KRISO container ship example



Test Cases (1.8 M cells)

a) Solver: simpleFoam (steady state)
relative Tolerance `relTol = 0.1`
absolute Tolerance `aTol=e-7/e-8`

b) Solver: simpleFoam (steady state)
relative Tolerance `relTol = 0.0`
absolute Tolerance `aTol=e-7/e-8`

c) Solver: interFoam (transient)
relative Tolerance `relTol = 0.1`
absolute Tolerance `aTol=1e-8`

Speedup KCS example

Speed-up for KCS-test case

Hardware	a)	b)	c)	remark
CPU only	1.00	1.00	1.00	
ofgpu1.0	3.35	7.95	1.85	SinglePrecision
cufflink	1.21	3.84	1.07	OF 1.6ext.
speedITclassic	fpe	fpe	0.66	CG for pressure
SP (tubs)	1.26	3.52	1.03	
8 CPUs only	3.75	4.00	2.88	
8 CPUs + cufflink	1.98	2.05	1.21	

Commercial library Culises

Speedup by adding multiple GPUs:

(a) single-socket board

Mesh - # CPUs	**9M - 1 CPU**	18M - 1 CPU	27M - 1 CPU	36M - 1 CPU
# GPUs added	**+1 GPU**	**+2 GPUs**	**+3 GPUs**	**+4 GPUs**
Speedup linear solver a	**3.5**	5.7	7.8	10.6
Speedup total simulation	**1.45**	1.59	1.67	1.74
Theoretical max speedup s_{max}	**1.78**	1.82	1.85	1.89

(b) dual-socket board

Mesh - # CPUs	9M - 2 CPU	**18M - 2 CPU**	27M - 2 CPU	36M - 2 CPU
# GPUs added	**+1 GPU**	**+2 GPUs**	**+3 GPUs**	**+4 GPUs**
Speedup linear solver a	2.5	**4.2**	6.2	6.9
Speedup total simulation	1.36	**1.52**	1.63	1.67
Theoretical max speedup s_{max}	1.78	**1.82**	1.85	1.89

B. Landmann, Accelerating the Numerical Simulation of Heavy-Vehicle Aerodynamics Using GPUs with Culises, ISC 2013, Leipzig , June 2013

Ahmdals Law...?

July 2013

Our recent findings indicate that the SpeedIT alone cannot accelerate OpenFOAM (and probably other CFD codes) to the satisfactory extent. If you follow our recent reports you will see SpeedIT is attractive for desktop computers but performs worse when compared to server class CPUs, such as Intel Xeon. The reason for such mild acceleration is the **Amdahl's Law** which states that the acceleration is bounded by the percentage of the code that cannot be parallalized.

???

Ahmdal's law stems from 1967...

What happened

- GPUs are made for number crunching
  - Sometimes there is too few to crunch
    Inner solver iterations for KCS:
    – Velocity ≈ 3
    – pressure ≈ 3
    – $(k, \omega) \approx 1$
- To few inner solver iterations in this example
- Time to copy matrix from CPU to GPU is dominant
- Most validation test cases are tuned to very small tolerances
- This gains the impressive speed-up

Realistic test cases needs usually less iterations & more relaxed tolerance settings

Overhead influence

Case	t^{PCG}_{step}, ms	$t^{GPU}_{startup}$, ms	GPU Speedup	$N^{=}_{iter}$	N^{2x}_{iter}
AIRFOIL	14	150	2.8	17	75
CAVITY	30	130	3.9	6	18
MOTORBIKE	50	250	3.0	8	30

t^{PCG}_{step}: time for one PCG iteration

$t^{GPU}_{overhead}$: GPU solver overhead (CPU-GPU memory copying)

$N^{=}_{iter}$: number of iterations until GPU solver is not slower

N^{2x}_{iter}: iterations until GPU solver is at least 2x faster

(A. Monakov, V. Platono: Accelerating OpenFOAM with a Parallel GPU, 8th OpenFOAM Workshop 2013)

Consequences

For **real** speedup I see two possible ways:

- Bring the whole algorithm (i.e. Simple/PISO) to the GPGPU
- Not only the matrix
- Student research Project at the Institute Scientific Computing
  Matthias Huy, TU Braunschweig
  Problems:
  - Object-oriented manner of OpenFOAM
  - A whole bunch of basic classes has to be brought to the GPU
- Program your own solver to the GPU...

Comparison CPU-FV vs. LBM-GPU

CPU-Finite Volume code

- Commercial solver running on 80 cores, Two-phase flow
- Hardware: 80 Core cluster
- Simulation time: $\approx$ 2 weeks

GPU-LBM code

- Lattice-Bolzmann code on a workstation Two-phase flow
- Hardware: 1 NVIDIA Quadro 6000 GPGPUs
- Simulation time : 8 hours

Comparison not really fair, but...

$\Rightarrow$ Speed up 40