Technische Universität Braunschweig

Institute of Scientific Computing

OpenFOAM on GPUs

3rd Northern germany OpenFoam User meetiNg

September 24th 2015

Haus der Wissenschaften, Braunschweig

Overview

- HPC on GPGPUs
- OpenFOAM on GPUs – 2013
- OpenFOAM on GPUs – 2015
- BiCGstab/IDR(s)

GPGPUs are perfect for HPC

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

Massive parallel data throughput

- Programming model inspired by vector computers (SIMD)
- Goal: Work of 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

So could we use GPUs to accelerate OpenFOAM?

So far...

several GPGPU-plug-ins for OpenFOAM available

- Usage:
  - Compile the plug-in
  - Check in the library (controlDict)

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

  - Declare the solver (fvSolutions)

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

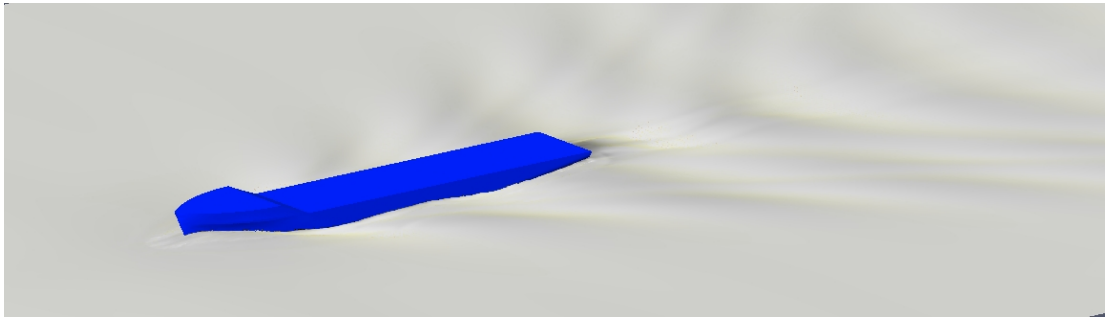
Situation 2013

My talk from the 1st NOFUN

- Comparison of different GPU plug-ins
- Based on realistic scenarios/test cases.
- Here:

  Korean research institute Container Ship

  KCStest case – 1.8 M cells



KCS example

Test Cases

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`

Speed-up KCS example

Comparison for different plug-ins

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	

Overhead influence

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

t_{step}^{PCG}: time for one PCG iteration

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

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

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

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

Consequences

What to do do gain the profit from GPU use?

- For **real** GPU speed-up:
  - Bring the whole algorithm
    (i.e. Simple/PISO)
    to the GPGPU
  - Not only the matrix
- Better port the whole stuff to the GPUs
  - i.e do it in
    - CUDA
    - OpenCL

What happened in the mean time?

- We were awarded as NVIDIA education centre
  (My courses Parallel Computing I & II)

- Fresh hardware
  - 1 Tesla K40
    2880 ALUs, 12 GB,
    288 GB/s, 1.43 Tflops (DP)
  - 5 Tesla Geforce GTX 760
    1152 ALUs, 2 GB,
    192,3 GB/s, 94,1 Gflops (DP)

- GPU porting project for OpenFOAM (mid 2015)

Paradigm change

- OpenFOAM running on GPU
  - All the computation is done entirely on GPU.
  - No need to copy data during calculations between CPU and GPU.
  - i.e. no overhead for GPU-CPU memory copy
- Open source
- Multi-GPU support
- Download from github:

  `https://github.com/Atizar/RapidCFD-dev`

Solvers

Status

- Most of the compressible & incompressible solvers (some with dynamic mesh support) are ported:

```
adjointShapeOptimizationFoam
buoyantBoussinesqPimpleFoam
buoyantBoussinesqSimpleFoam
buoyantPimpleFoam
buoyantSimpleFoam
driftFluxFoam
icoFoam
interDyMFoam
interFoam
laplacianFoam
nonNewtonianIcoFoam
pimpleDyMFoam
pimpleFoam
pisoFoam
porousInterFoam
porousSimpleFoam
potentialFoam
rhoCentralDyMFoam
rhoCentralFoam
rhoPimplecFoam
rhoPimpleDyMFoam
rhoPimpleFoam
rhoPorousSimpleFoam
rhoSimplecFoam
rhoSimpleFoam
scalarTransportFoam
shallowWaterFoam
simpleFoam
sonicDyMFoam
sonicFoam
sonicLiquidFoam
SRFPimpleFoam
SRFSimpleFoam
thermoFoam
```

Equation solvers

- System-of-equation solvers
  - BICCG
  - GAMG
  - ICCG
  - PBiCG
  - smoothSolver (Jacobi)
- Lack of good and robust preconditioners
  (up til now: `diagonal, AINV`)
- Mesher & tools not available
  (use from CPU version)
- Some schemes/B.C.s are missing

Single GPU/Multi-GPU support

Single GPU

- Single GPU support is directly usable, i.e. uses standard OpenMPI
  - from your operation system (systemMPI)
  - from OpenFOAM `ThirdParty` directory

Multi GPU

- Multi-GPU support needs a **CUDA-aware MPI**:
  - OpenMPI 1.8.4 or later
  - compiled with –with-cuda option
  - Should be located in `ThirdParty` directory.

GPU device selection

Device selection is done using command line arguments

- For single GPU use `-device` argument:

```
simpleFoam -device 2
```

  - Tells OFgpu to use GPU with ID 2.
- For multi GPU execution you use `-devices` argument followed by a list of GPU IDs:

```
mpirun -n 2 simpleFoam -parallel -devices "(2 3)"
```

  - Tells OGgpu to use GPU 2 for process 0 and GPU 3 for process 1.

Comparison CPU–GPU

Case:

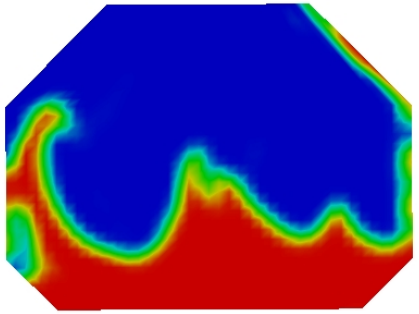
- buoyantCavity
- solver: `buoyantSimpleFoam`
- RANS model: `kOmegaSST`
- # cells: 3,375,000

Solver	**Clock time** (sec)	**Speed-up**
1 CPU	20320	1.0
8 CPU	7551	2.7
GPU (k40)	2147	9.5

Comparison CPU–GPU

Case:

- sloshingTank3d
- solver: `interDyMFoam`
- RANS model: `kEpsilon`
- # cells: 2,584,000

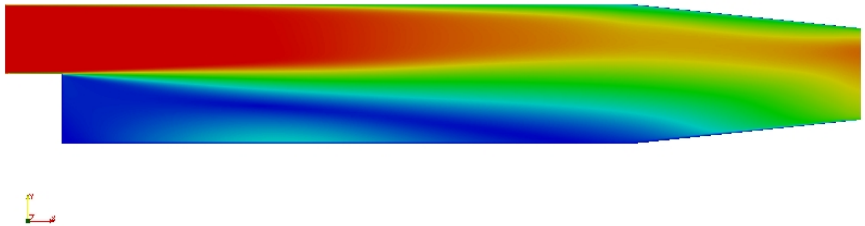


Solver	**Clock time** (sec)	**Speed-up**
1 CPU	1959	1.0
8 CPU	631	3.1
1 GPU (K40)	708	2.8

Comparison CPU–GPU

Case:

- pitzDaily
- solver: `simpleFoam`
- RANS model: `kEpsilon`
- # cells: 1,222,500

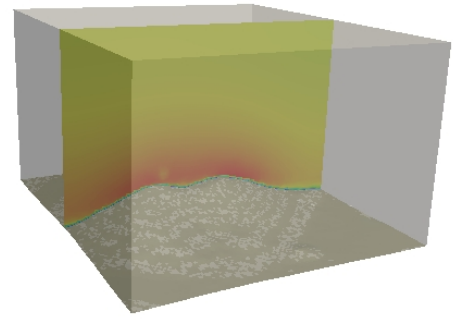


Solver	**Iterations**	**Clock time** (sec)	**Speed-up**
1 CPU	511	2910	1.0
8 CPU	518	950	3.1
1 GPU (K40)	488	405	7.2

Comparison CPU–GPU

Case:

- turbineSiting
- solver: `simpleFoam`
- RANS model: `kEpsilon`
- # cells: 120246



Solver	**Iterations**	**Clock time** (sec)	**Speed-up**
1 CPU	251	108	1.0
8 CPU	266	26	4.1
1 GPU (K40)	5000	14831	(not converged)

Development: BiCGstab to GPU

Porting Biorthogonal Conjugated Gradient stabilized [1]

Motivation

- Disadvantages BiCG
  - There are multiplications with A^T needed.
  - For a regular matrix, the method could terminate without solution
  - The method has no minimization properties for the iteration vector. This could cause an oscillatory behaviour in the convergence
- Advantages BiCGstab
  - In general, BiCGstab has smoother convergence properties
  - Multiplication with A^T is not necessary.

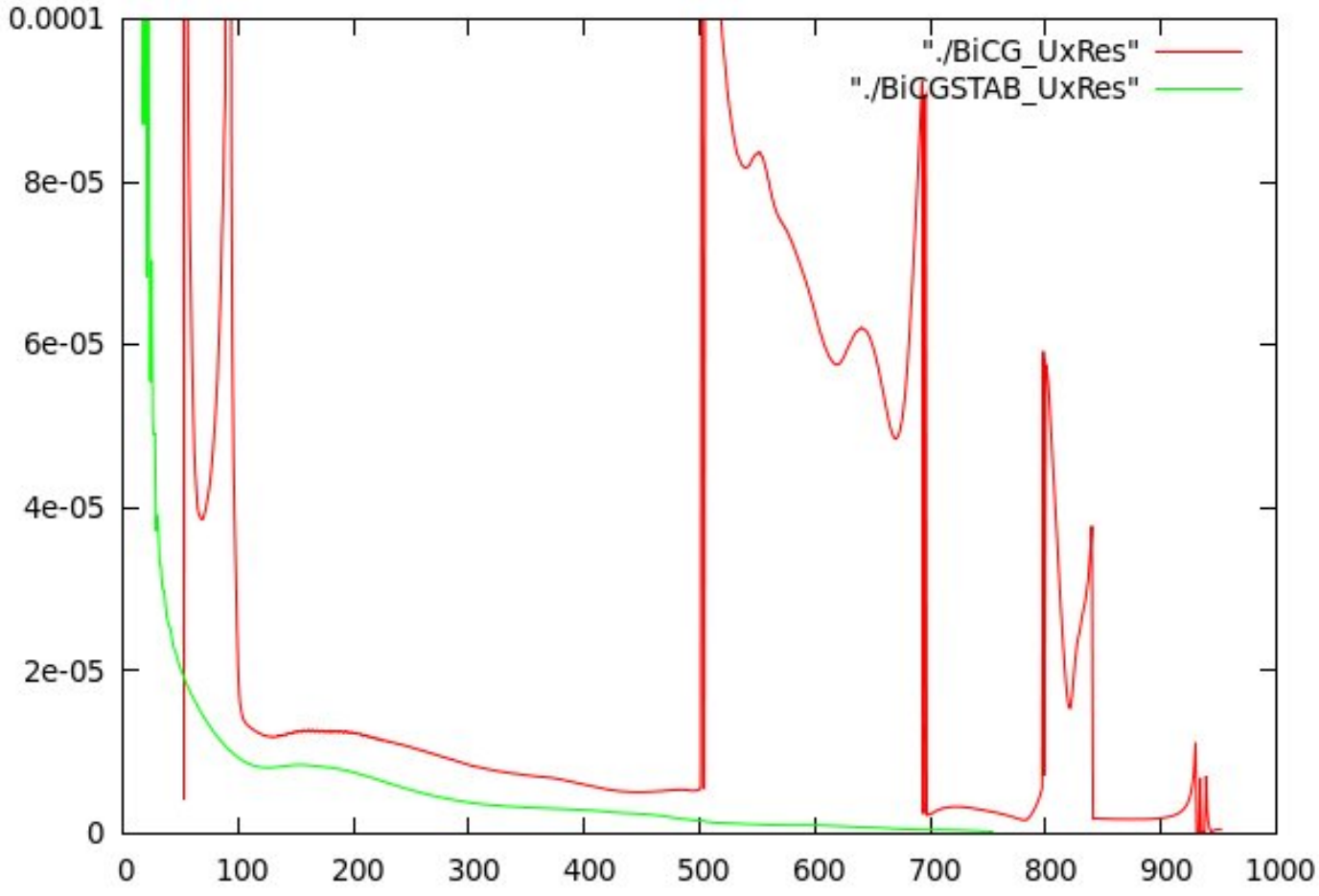
[1] van der Vorst, H. A. *BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13: 631-644, 1992.

Development: BiCGstab to GPU

Code snippet

```
scalargpuField vA(PBCache::vA(matrix_.level(),nCells),nCells);

// --- Calculate v=A*p and precondition v
  matrix_.Amul(vA, pA, interfaceBouCoeffs_, interfaces_, cmpt);
  preconPtr->precondition(vP, vA, cmpt);

  // --- Calculate alpha=rPr0/vPr0
  scalar alpha2 = gSumProd(vP, r0, matrix().mesh().comm());
  scalar alpha = rPr0/alpha2;

    thrust::transform
    (
        rA.begin(), rA.end(), vA.begin(), sA.begin(), rAMinusAlphaWAFunctor(alpha)
    );

    // --- Precondition s
    preconPtr->precondition(sP, sA, cmpt);

    // --- Calculate t=A*sP and precondition t
    matrix_.Amul(tA, sP, interfaceBouCoeffs_, interfaces_, cmpt);
    preconPtr->precondition(tP, tA, cmpt);
```

Comparison BiCG/BiCGSTAB – Residuals



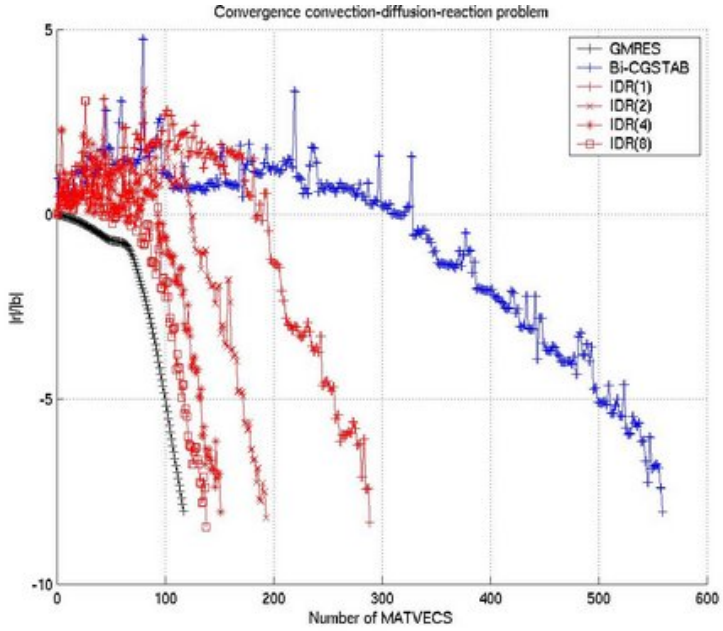
Comparison of the residuals BiCG/BiCGSTAB

Plan: IDR(s)

Induced Dimension Reduction method [2]

- Efficient methods for large nonsymmetric systems

- Based on the Induced Dimension Reduction (IDR) method proposed by Sonneveld (1980)
- Competitive with or superior to most Bi-CG-based methods
- Outperforms BiCGstab for $s > 1$.



[2] Peter Sonneveld and Martin B. van Gijzen, IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems. SIAM J. Sci. Comput. Vol. 31, No. 2, pp. 1035-1062, 2008

Conclusions

- The portation of OpenFOAM to GPU can be a promising approach
- But:

Some work has to be done

  - Solvers (convergence?)
  - Preconditioners
  - Environment/Utilities

- Examining the possibility to use external GPU equation solver libraries with OpenFOAM.