Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations

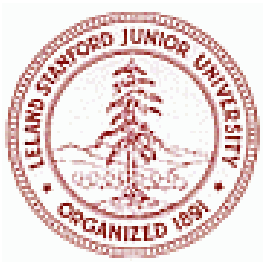
Dominik Göddeke

Angewandte Mathematik und Numerik,
Universität Dortmund

Acknowledgments

Joint work with

- Robert Strzodka (Stanford)
- Stefan Turek and the FEAST Group (Dortmund)
- Patrick McCormick (LANL)

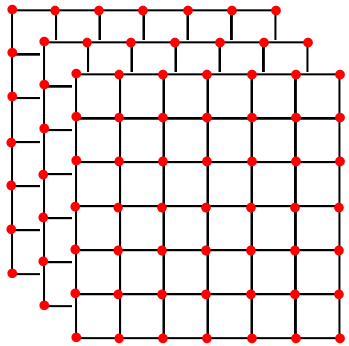


Overview

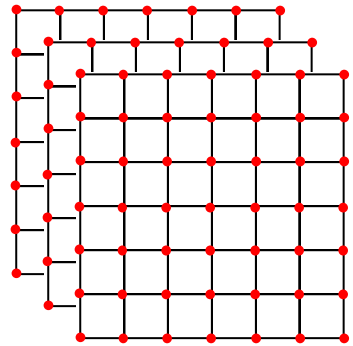
- High Level Introduction to GPUs and the Cell BE
- Precision and Accuracy
- High Precision Emulation
  vs.
  Mixed Precision Iterative Refinement
- High Performance Computing and FEAST

The GPU: A Fast, Parallel Array Processor

Input Arrays:
1D, 3D,
2D (typical)



Output Arrays:
1D, 3D (slice),
2D (typical)



Vertex Processor (VP)

Kernel changes index **regions** of input arrays

Fragment Processor (FP)

Kernel changes each element independently, reads more input arrays

Rasterizer

Creates data streams from index **regions**

Stream of array elements, order unknown

GPU Characteristics

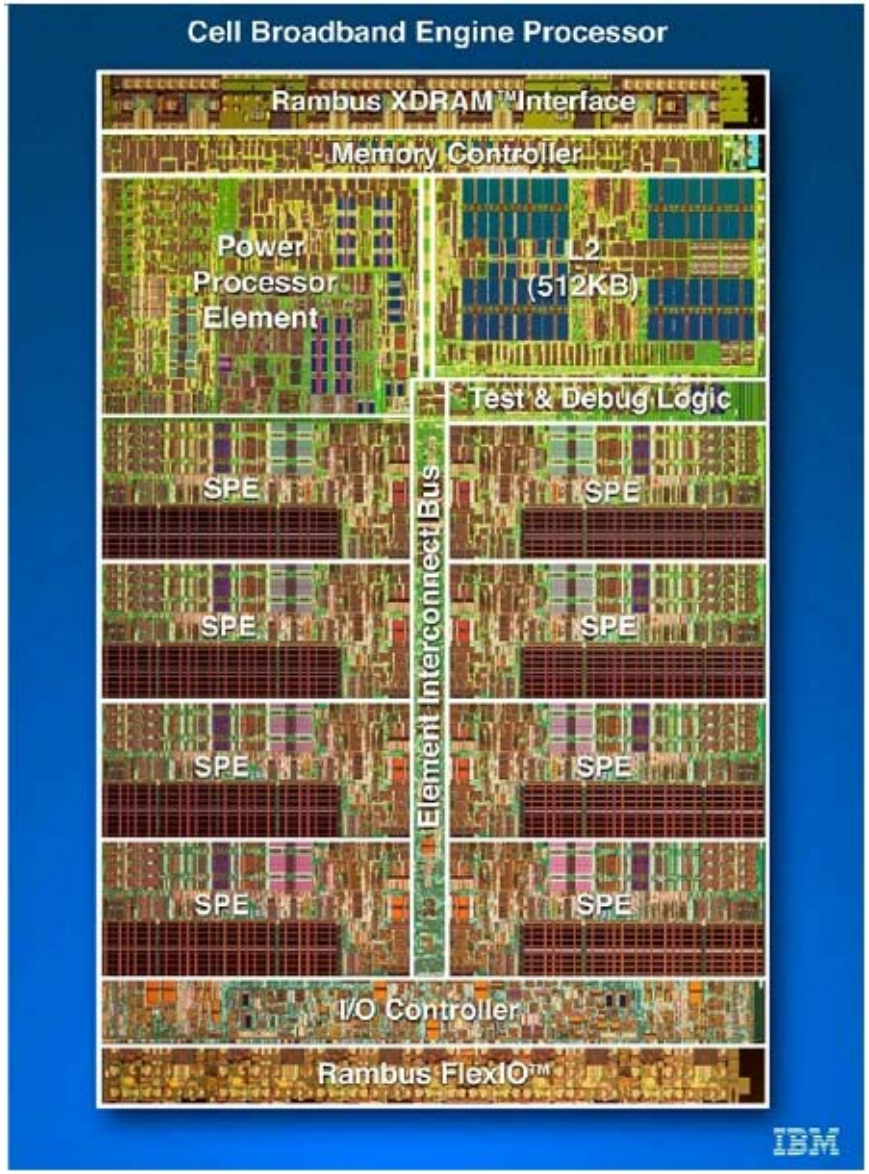
- **Only quasi-IEEE single precision**
  - no denorms, round-to-zero
- **Peak performance > 300 GFLOP/s for latest ATI**
  - only of theoretical value (no app has that arithmetic intensity)
  - Folding@Home (Stanford): sustained 100 GFLOP/s
  - Dense Matrix-Matrix-Multiplication: sustained 110 GFLOP/s (Stanford)
- **Peak memory performance**
  - Sustained memory bandwidth of 20-40 Gbyte/s for sequential, uniform access patterns

GPU Characteristics

- **Lots of limitations**
  - Almost no scatter, no read-modify-write
  - Transfers GPU-CPU: 1-2 GB/s only
  - Only good for LARGE problem sizes
  - Indirect programming (but: ATI's "close to metal" low-level abstraction currently under NDA'ed beta)
- **Ubiquitous availability**
  - Ideal for commodity clusters and small workstations
- **Limited programming model allows for many optimisations in hardware!**

STI Cell Broadband Engine

- 21 GFLOP/s double, 230 GFLOP/s single precision peak
- Programmable memory interface (manual shifting of data to the SPEs)
- Only 256K data/program memory per SPE
- 100s of threads in parallel



Overview

- High Level Introduction to GPUs and the Cell BE
- Precision and Accuracy
- High Precision Emulation
  vs.
  Mixed Precision Iterative Refinement
- High Performance Computing and FEAST

Confusing Example: S.M. Rump (1988)

Evaluating (powers as multiplications)

$$f(x,y) = (333.75 - x^2)\, y^6 + x^2\,(11x^2y^2 - 121y^4 - 2) + 5.5y^8 + x/(2y)$$

$$x = 77617,\ y = 33096$$

as float s23e8	1.1726
as double s52e11	1.17260394005318
as long double s63e15	1.172603940053178631

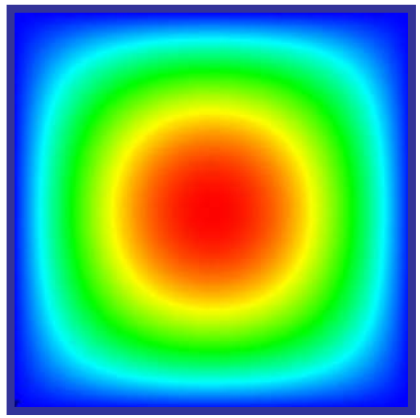
Looks good? Not even the sign is correct!

-0.827396059946821368141165095479816291…

Computational Precision ≠ Accuracy of Result

PDE Example: Poisson Problem

- Unit square
- Bilinear conforming FEs (Q1)
- Zero Dirichlet BCs
- Regular refinement



- FEM theory: pure discretization error,
- Expect error reduction of a factor of 4 (i.e. h^2) in each level

- Solved using MG until norms of residuals *indicate* three digit relative reduction
- Comparison of integral L2 error against analytically known reference solution in double precision

PDE Example: Poisson Problem

- **Error reduction: single and double precision**

NEQs	SINGLE	REDUCTION	DOUBLE	REDUCTION
3^2	5.208e-3		5.208e-3	
5^2	1.440e-3	3.62	1.440e-3	3.62
9^2	3.869e-4	3.72	3.869e-4	3.72
17^2	1.015e-4	3.81	1.015e-4	3.81
33^2	2.611e-5	3.89	2.607e-5	3.89
65^2	6.464e-6	4.04	6.612e-6	3.94
129^2	1.656e-6	3.90	1.666e-6	3.97
257^2	**5.927e-7**	**2.79**	4.181e-7	3.98
513^2	**2.803e-5**	**0.02**	1.047e-7	3.99
1025^2	**7.708e-5**	**0.36**	2.620e-8	4.00

Precision and Accuracy: Poisson

- **Residuals indicate that the problem has been solved sufficiently accurate in single precision**
  - But the result is absolutely off
  - Further refinement (with implied significant increase in time to solution) results in error increase, not reduction

- **We all use double precision exclusively to avoid this catastrophic behaviour**
  - Same problem in double precision as soon as the systems become sufficiently large or ill-conditioned
  - But we might want to exploit the speed of native low precision computations without sacrificing accuracy

Precision and Accuracy: High level view

- There is no monotonic relation between the computational precision and the accuracy of the final result.
- Increasing precision can decrease accuracy !
- Even when one can prove positive effects of increased precision, it is very difficult to quantify them.
- We often simply rely on the experience that increased precision helps in common cases.
- But for common cases we need high precision only in very few places to obtain the desired accuracy.

Overview

- High Level Introduction to GPUs and the Cell BE
- Precision and Accuracy
- High Precision Emulation
  vs.
  Mixed Precision Iterative Refinement
- High Performance Computing and FEAST

High Precision Emulation

- **Combine two native floating point values**
  - Effectively doubles mantissa
  - Common exponent used to align mantissas
  - Normalisation required after each operation
  - Combined dynamic range slightly less than native range
  - Increases op count by 11x (ADD) and 18-32x (MUL)
  - Doubles bandwidth requirements

- **Example**
  - Two s23e8 yield a quasi s46e8 float with same storage requirements as a full s52e11 double
  - For large scale problems, we will need quad precision.

Example: Addition c=a+b

- **Compute high-order sum and error**

  t1 = a.hi + b.hi

  e = t1 – a.hi

- **Compute low order term including error and overflows**

  t2 = ((b.hi - e) + (a.hi - (t1 - e))) + a.lo + b.lo

- **Normalise to get final result**

  c.hi = t1 + t2

  c.lo = t2 – (c.hi – t1)

Mixed Precision Iterative Refinement

- **Exploit the speed of low precision to achieve the accuracy of high precision**

$d_k = b - Ax_k$	**Compute** in high precision (cheap)
$Ac_k = d_k$	**Solve** in low precision (fast)
$x_{k+1} = x_k + c_k$	**Correct** in high precision (cheap)
$k = k+1$	Iterate until convergence in high precision

- **Low precision solution is used as preconditioner in a high precision iterative method**
  - A is small and dense: Solve $Ac_k = d_k$ directly
  - A is large and sparse: Solve (approximately) $Ac_k = d_k$ with an iterative method itself

Direct Scheme for Small, Dense A

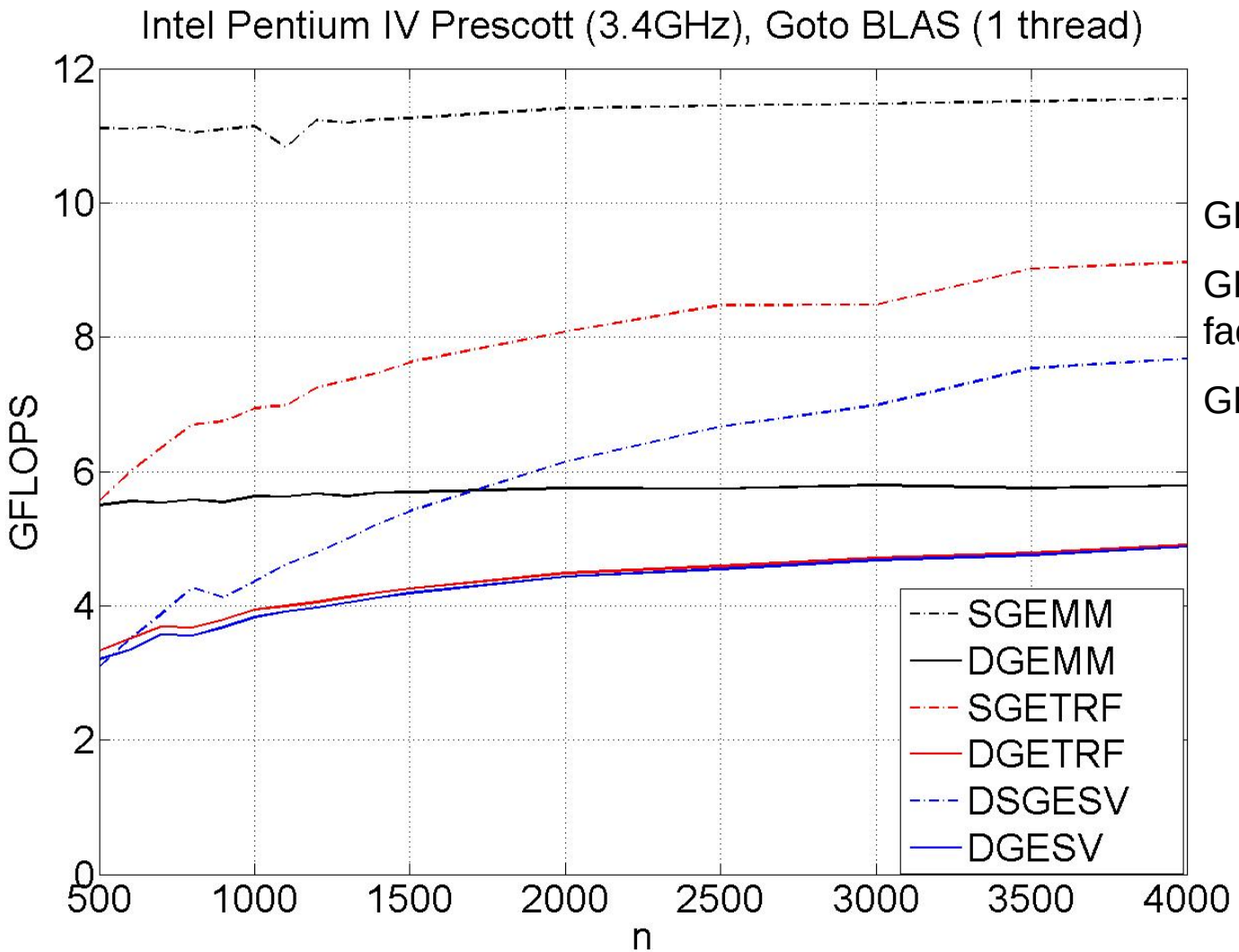
- **Algorithm**
  - Compute PA=LU once in single precision
  - Use LU decomposition to solve $Ly=Pd_k$,$Uc_k=y$ in each step
  - Implemented into next release of LAPACK

- **Main reasons for speedup**
  - Computation of LU decomposition is O(n^3)
  - Computation of LU is much faster in single than in double
  - Solution using LU for several RHS is only O(n^2)

- **Upper bound for iteration count**
  - ceil(t_d/(t_s-K), where K,t_d,t_s are log10 of matrix condition and double and single precision (e.g. t_d approx 16)

CPU Timings: LU Solver

→ Larger is better →



GEMM: MatMat

GETRF: LU factorisation only

GESV: full solution

chart courtesy of Jack Dongarra

[Langou et al. *Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems)*, SC 2006, to appear]

CPU Timings: LU Solver in Quad Precision

	QGESV	QDGESV	
n	time (s)	time (s)	speedup
100	0.29	0.03	9.5
200	2.27	0.10	20.9
300	7.61	0.24	30.5
400	17.81	0.44	40.4
500	34.71	0.69	49.7
600	60.11	1.01	59.0
700	94.95	1.38	68.7
800	141.75	1.83	77.3
900	201.81	2.33	86.3
1000	276.94	2.92	94.8

[Langou et al. *Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems),* SC 2006, to appear]

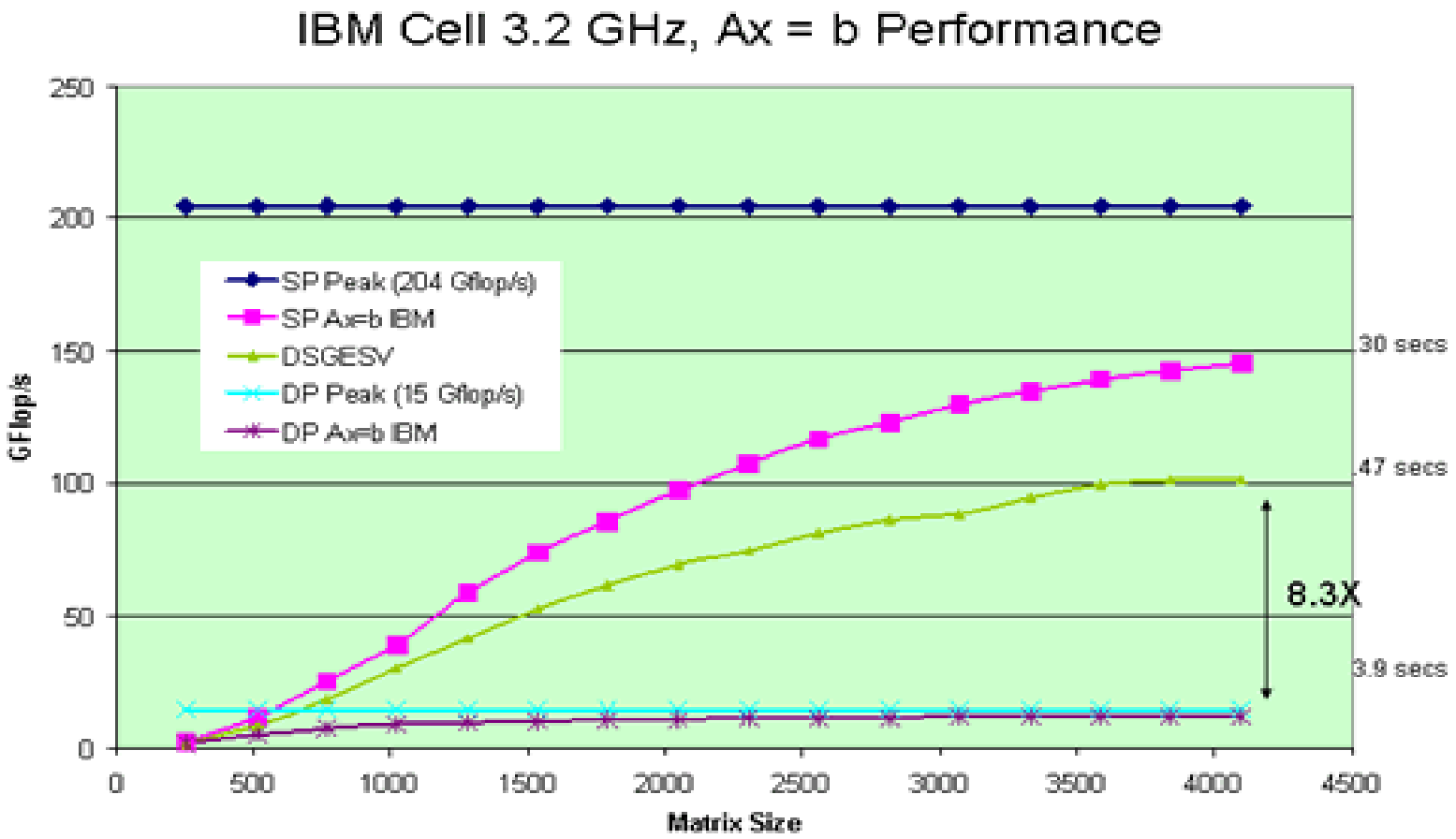
Table courtesy of Jack Dongarra

This is O(n^3) work.

First test by Michael:

QD-FEAT2DTEST (O(n) work) runs 18x slower than standard double FEAT2DTEST

Cell Timings: LU Solver



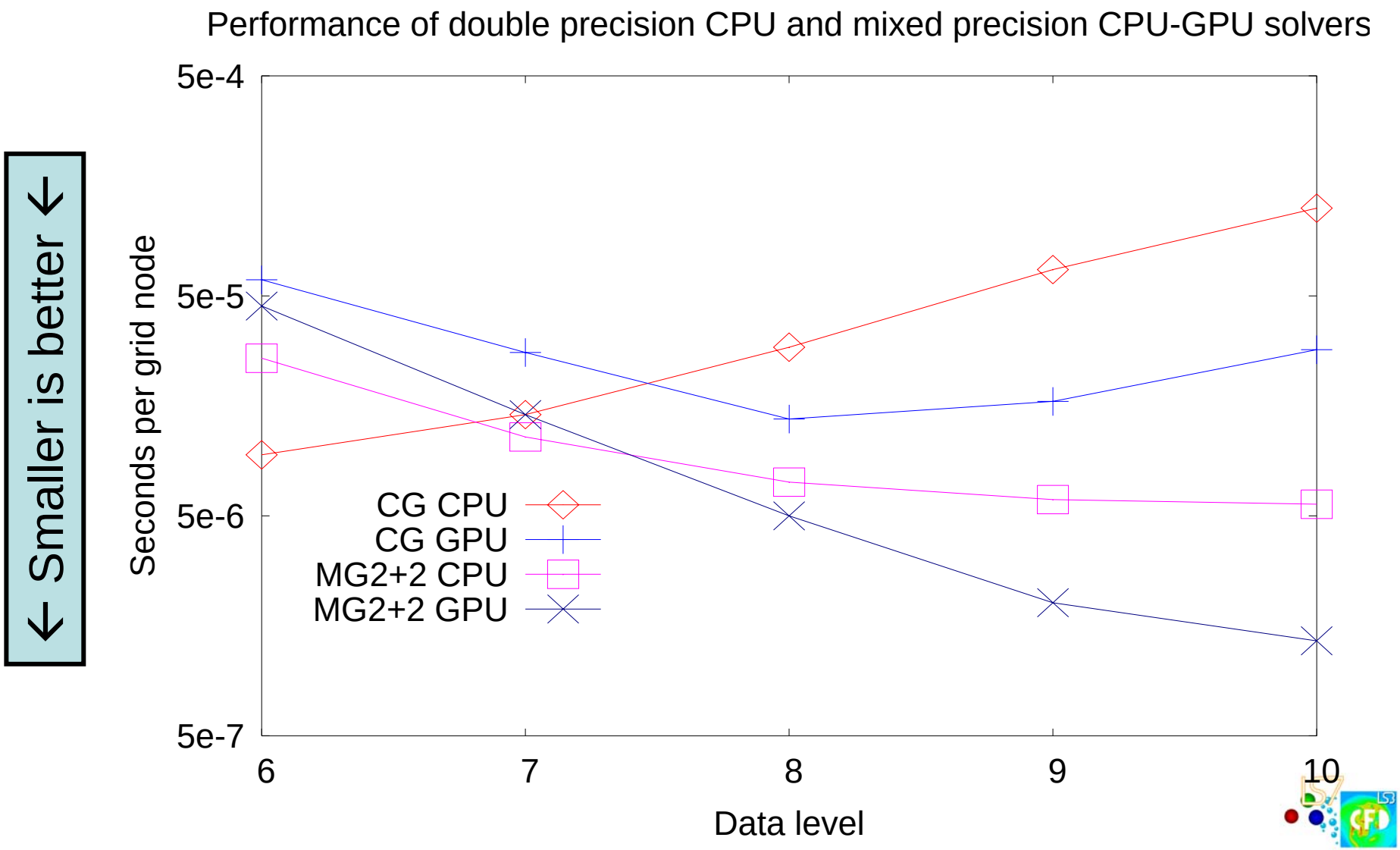
[Langou et al. *Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems),* SC 2006, to appear]

chart courtesy of Jack Dongarra

Iterative Scheme for Large, Sparse A

- **Algorithm**
  - Inner solver: CG, Multigrid
  - Terminate inner solver after fixed number of iterations, fixed error reduction or convergence
- **Main reason for speedup**
  - Inner solver can run on the GPU close to peak bandwidth
  - FEAST matrix structure has too low arithmetic intensity to allow real speedups on CPUs unless you explicitely code this in SSE for a specific CPU
- **Compared to double precision emulation**
  - Faster by a factor 2-3
  - Memory requirements cut in half

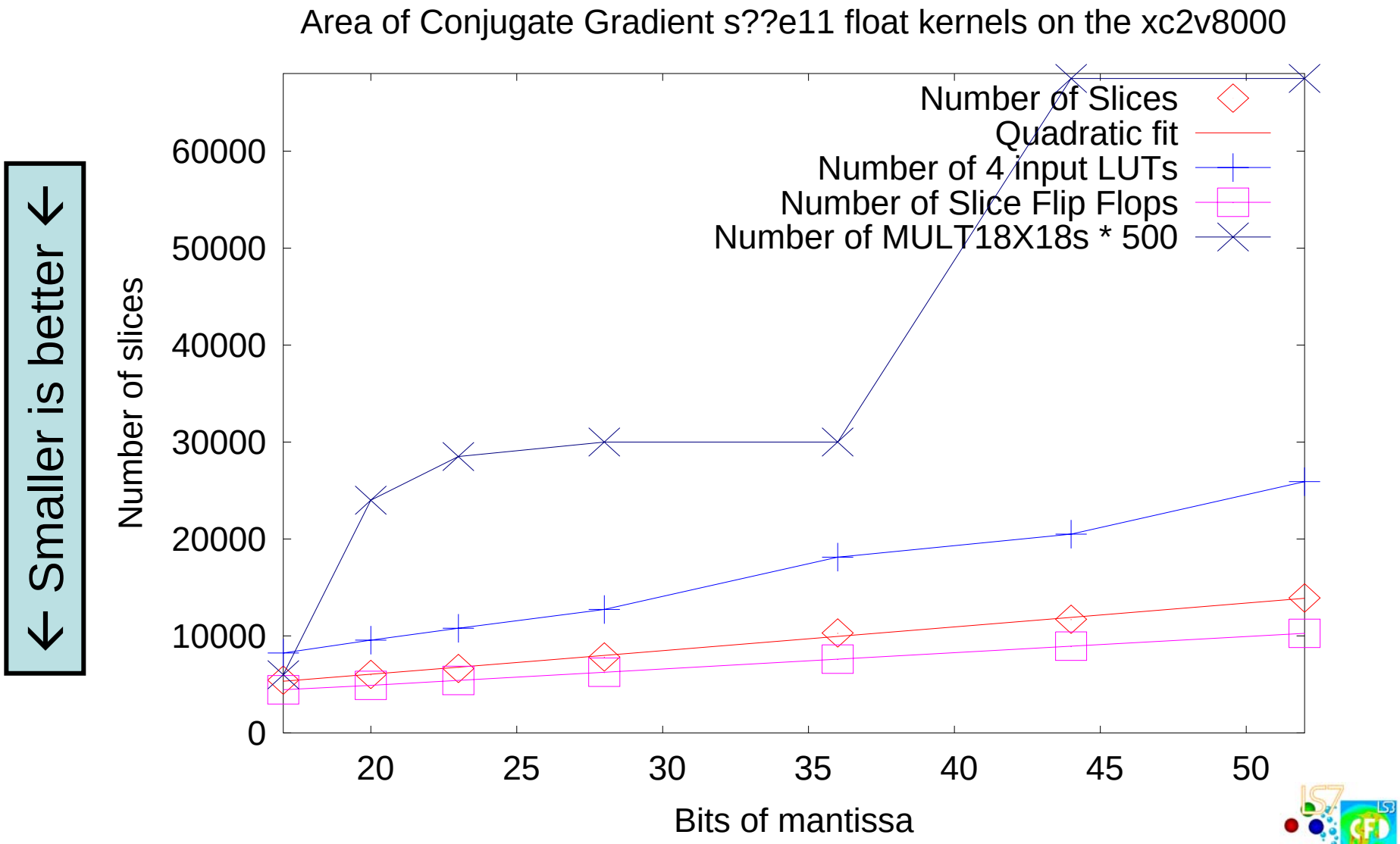
GPU Timings: CG and Multigrid



Excursion: Savings in space

- **Mixed precision methods allow not only savings in time, but also in space**
  - Invest transistors more efficiently
  - Fabricate one double or four single precision multipliers
- **Prototypical implementation of FPGA - CG core**
  - New CG formulation that matches FPGA data flow
  - Fine-grained control of mantissa bit length
  - Employ hardwired 18x18 integer multipliers
  - True quadratic savings

FPGA Results: CG with MUL18x18



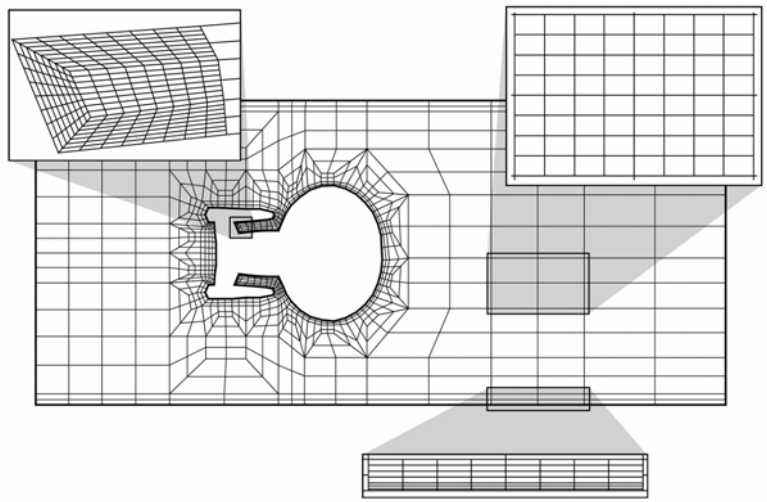
← Smaller is better ←

Overview

- High Level Introduction to GPUs and the Cell BE
- Precision and Accuracy
- High Precision Emulation
  vs.
  Mixed Precision Iterative Refinement
- High Performance Computing and FEAST

FEAST: Generalised Tensor-Product Grids

- **Sufficient flexibility in domain discretization**
  - Global unstructured macro mesh, domain decomposition
  - (an)isotropic refinement into local tensor-product grids



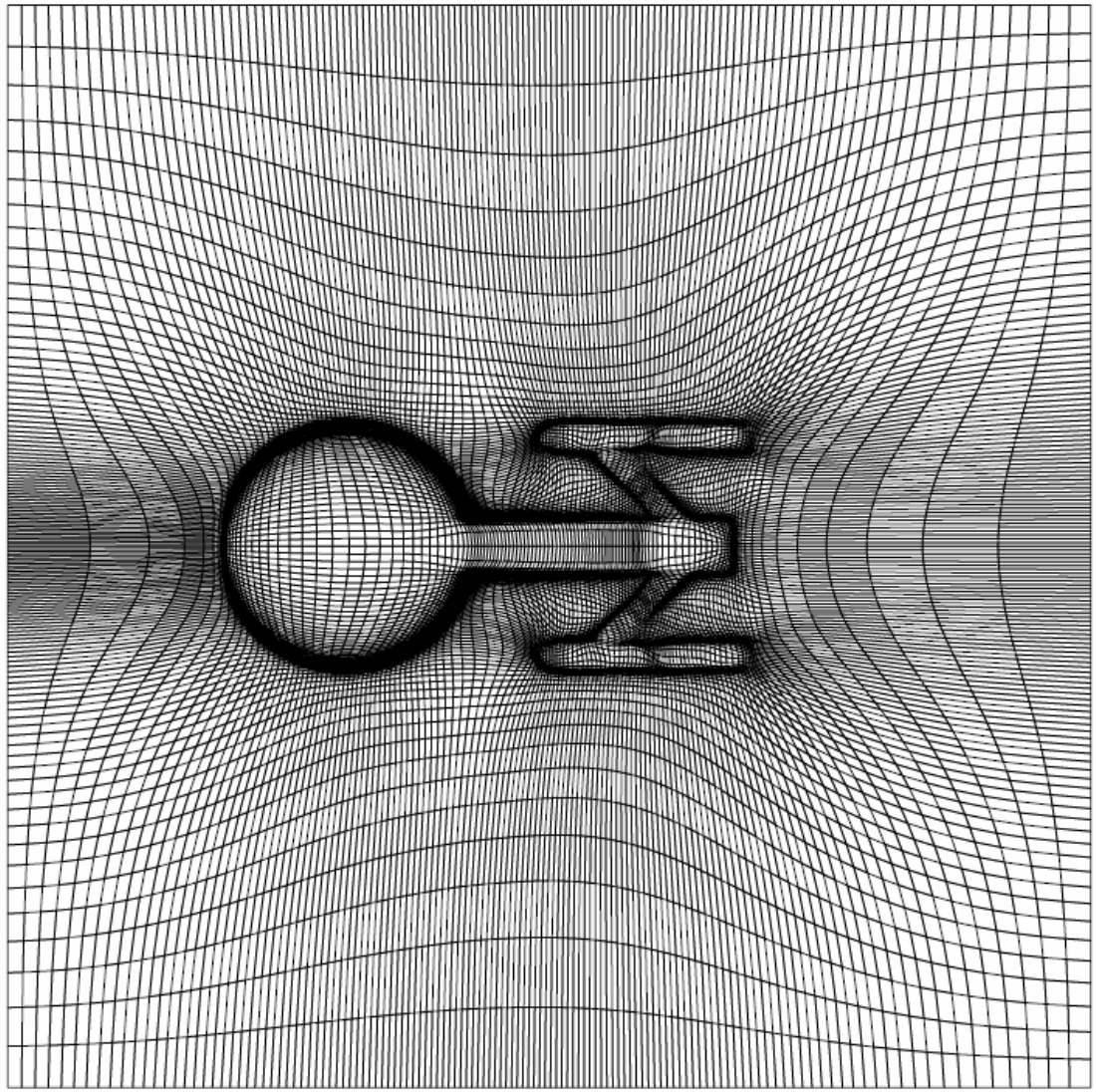
- **Efficient computation**
  - High data locality, large problems map well to clusters
  - Problem specific solvers depending on anisotropy level
  - Hardware accelerated solvers on regular sub-problems

FEAST and ScaRC

- **Find and exploit locally structured parts**
  - Regular data structures are key to good performance
  - High GFLOP/s rates
  - Offloading to hardware co-processors possible
- **Find and hide locally anisotropic parts**
  - Robust solver with good convergence rates
  - Local anisotropies do not impact global multigrid
- **Combine advantages of parallel MG and DD**
  - More details in future talks by the FEAST folks

FEAST: Deformation Adaptivity

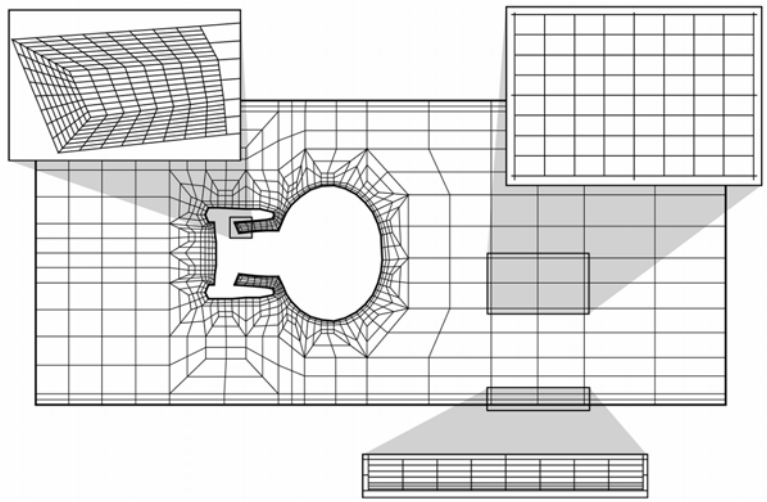
- This grid is a tensor-product !
- Easier to accelerate in hardware than resolution adaptive grids
- Anisotropy level determines optimal solver
- More in Grabo's talk



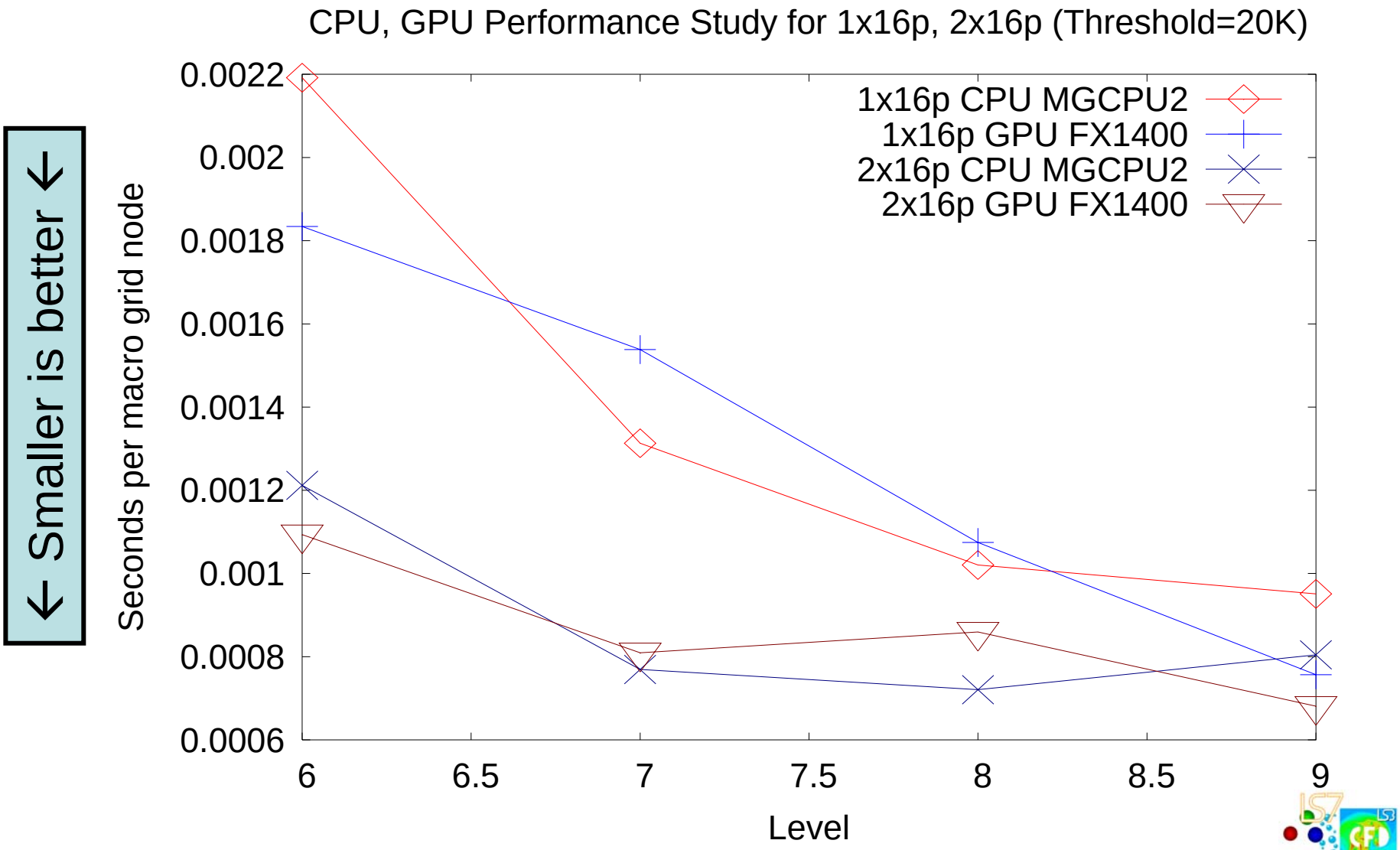
FEASTGPU

- **Minimally invasive integration**
  - Do not reinvent the wheel, do not rewrite FEAST fully
  - Offload time-critical parts to the GPU

- **CPU and GPU both execute what they are best at**
  - CPU: global high precision MG
  - CPU: communication
  - GPU: local smoothing
  - (CPU: powerful local smoothers)
  - Mixed precision MG--MG



FEAST: GPU Cluster Performance



RESULTS OUTDATED

Conclusions

- Computational precision ≠ result accuracy
- Mixed precision schemes: Exploit low precision speed to achieve high precision accuracy faster
- GPUs (and soon, Cells) as numerical coprocessors