
Massively Parallel Computing with Cuda

- System Integration -

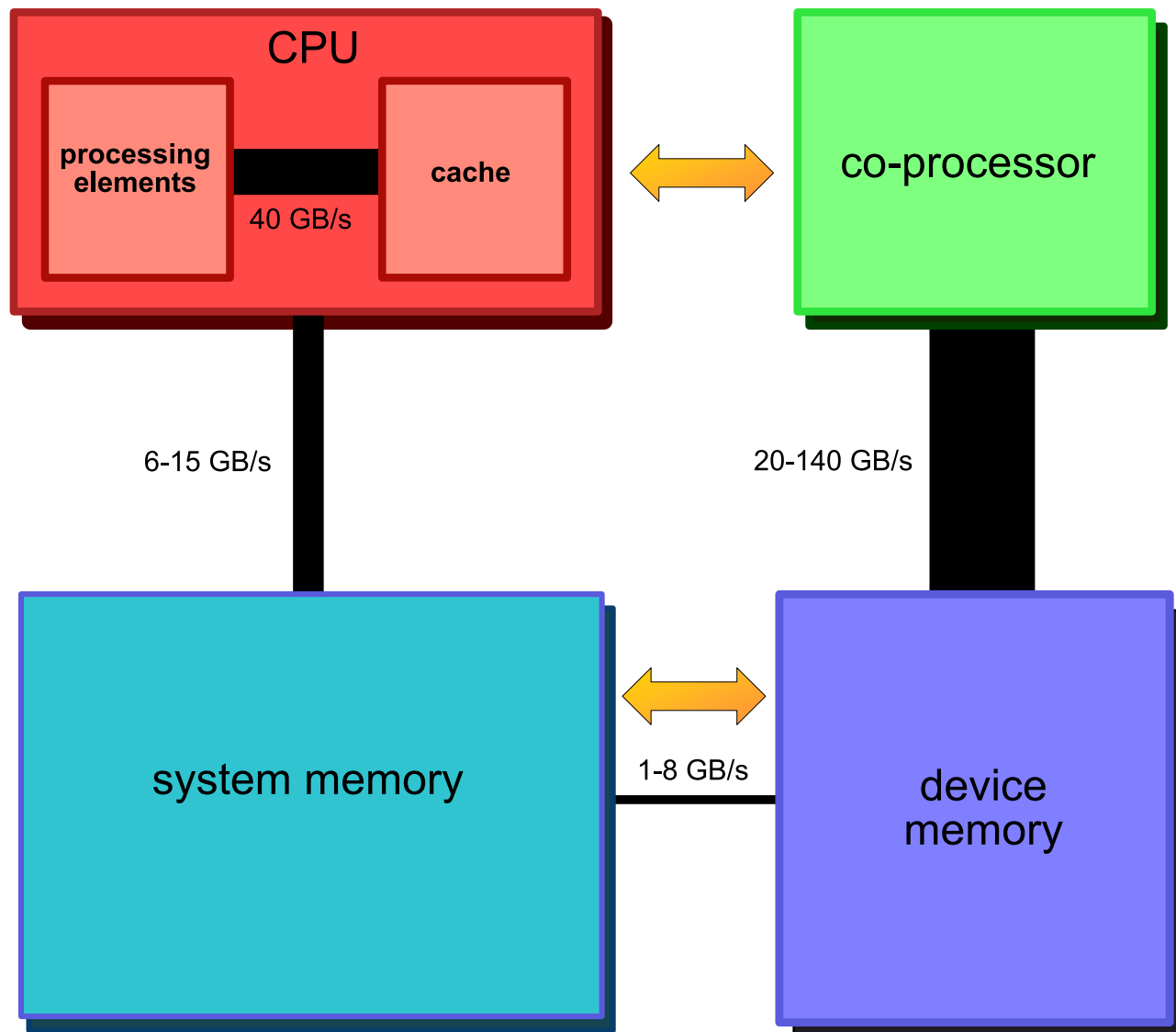
Hendrik Lensch
Robert Strzodka

Today

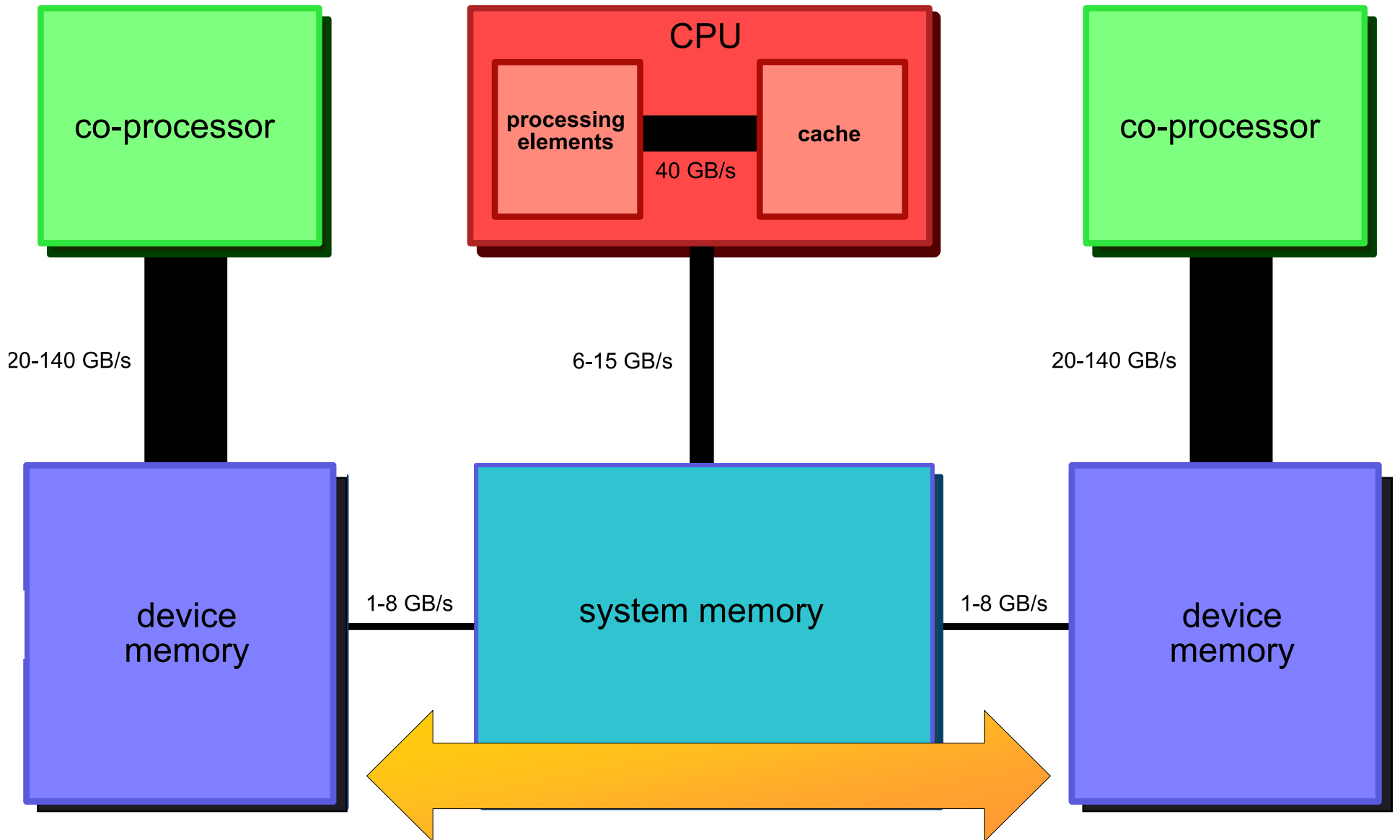
- **Multi-GPU Systems**
- **Large Scale SW-HW Integration**

Multi-GPU Systems

Accelerator Node with PCI (Express) Connection

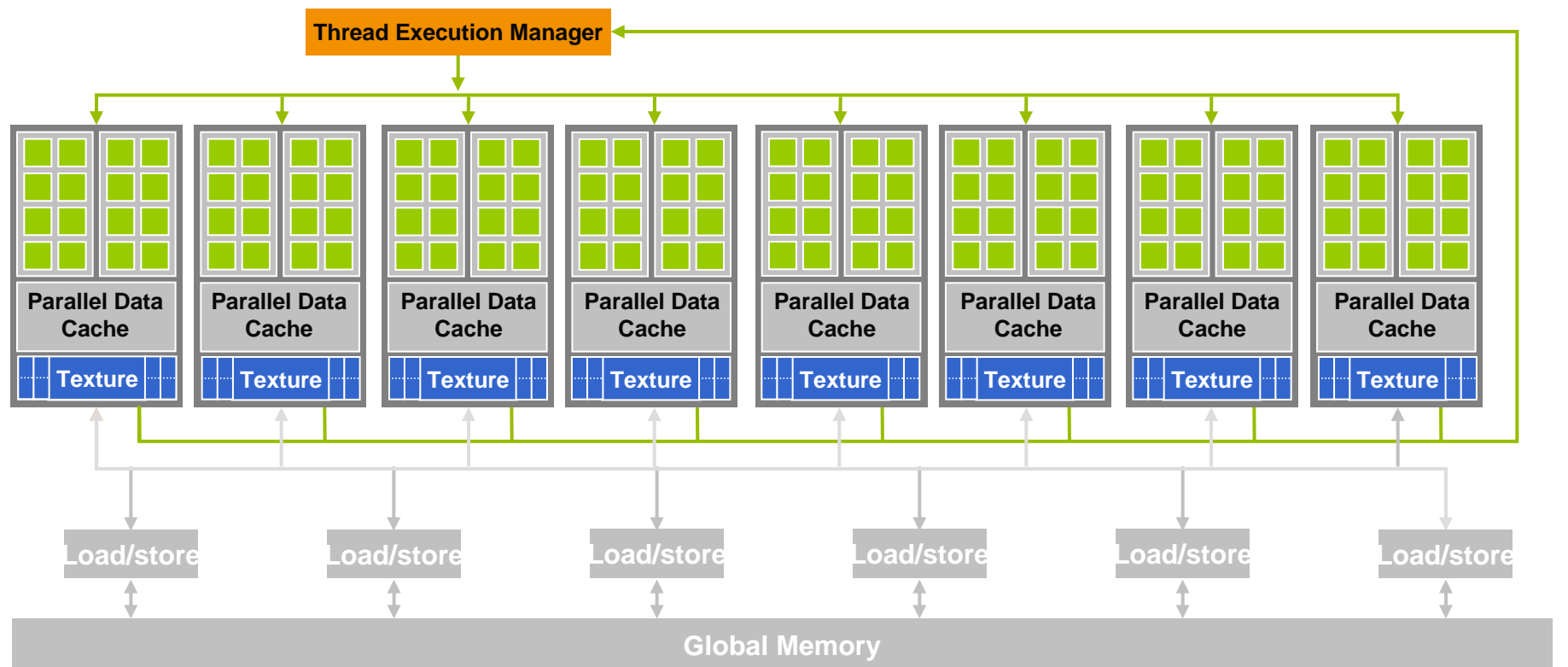


Multi-GPU Node



GPU to Multi-GPU vs. SM to GPU Relation

- **Similarity:** Independent processes exchange data through common memory
- **Difference:** Unlike an SM, a GPU cannot trigger direct communication to the common memory



GPU to Multi-GPU vs. SM to GPU Relation

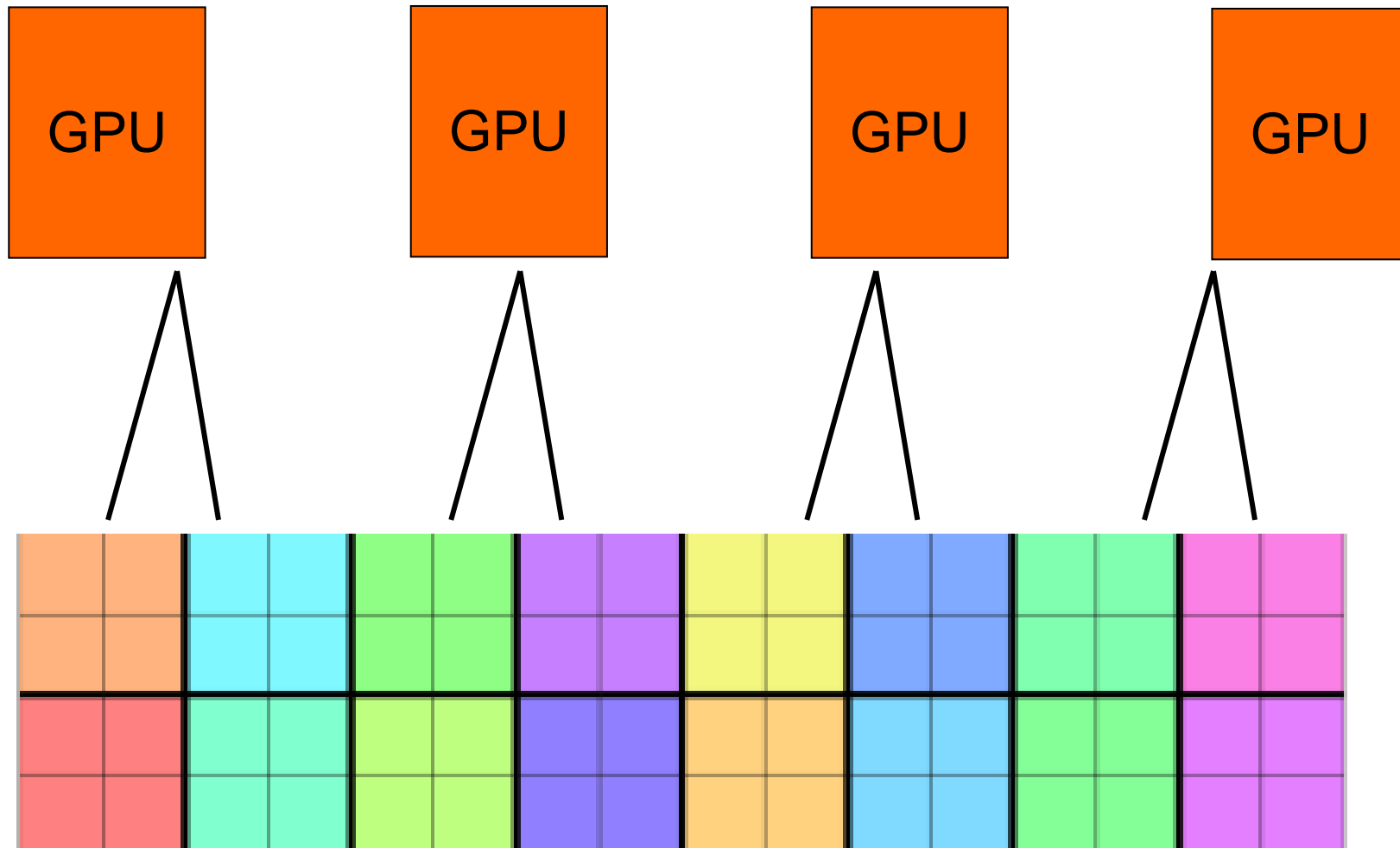
SM to GPU

- Data exchange $\geq 64\text{B}$
- **Immediate** synchronization with atomics
- No or **small** local data duplication

GPU to Multi-GPU

- Data exchange $\geq 1\text{KB}$
- **Delayed** synchronization between kernel calls
- **Full** local data duplication may be necessary

Data Distribution



Data Distribution

- **Boundary data duplication**
 - Data to GPU assignment is **known a-priori**
 - Boundary values require neighbors
 - The **data domain is cut into pieces** (e.g. tiles) and assigned to GPUs
 - E.g. A grid subdivided into tiles
- **Pros**
 - Data transport to GPUs only slightly bigger than overall data
 - Each GPU examines only **its own data**
 - **Predictable** performance
 - For a single execution only one synchronization at the end
- **Cons**
 - Either entire boundary tiles are used, or **special cases** required
 - In a loop all boundary layers must be **redistributed to neighbors**
 - Suitable only for **local operators**

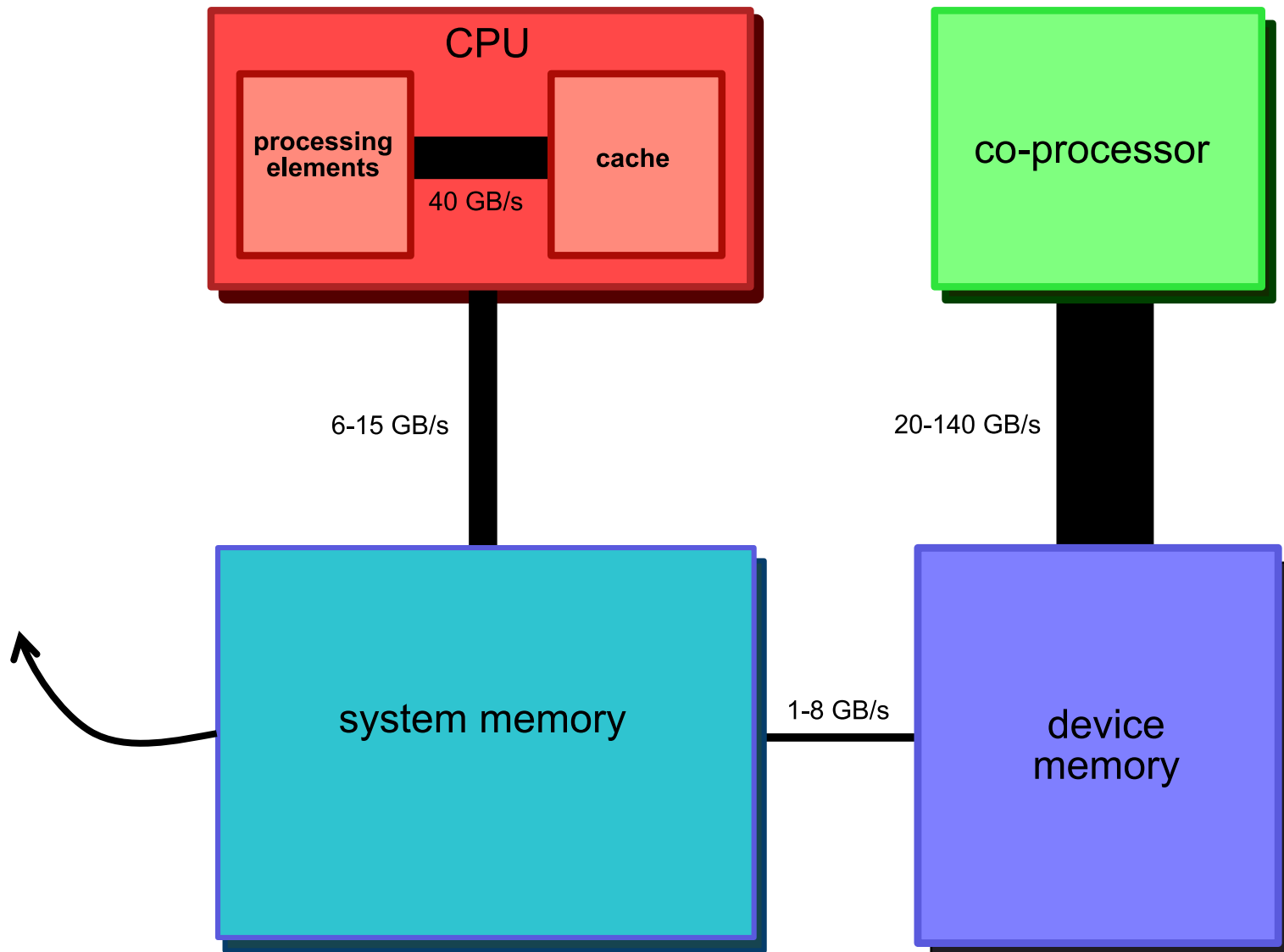
Data Distribution

- **Full data duplication**
 - Data to GPU assignment **not known a-priori**
 - Classification would be expensive or impossible
 - All data is **duplicated across all GPUs**
 - E.g. Stream of graphical primitives
- **Pros**
 - **Simple** solution covers **difficult dependencies**
 - **Predictable** worst case performance
 - For a single execution only one synchronization at the end
- **Cons**
 - Data transport to GPUs **multiplies**
 - Each GPU must check **entire data set** for classification
 - In a loop all results must be redistributed in **N one-to-all** communications

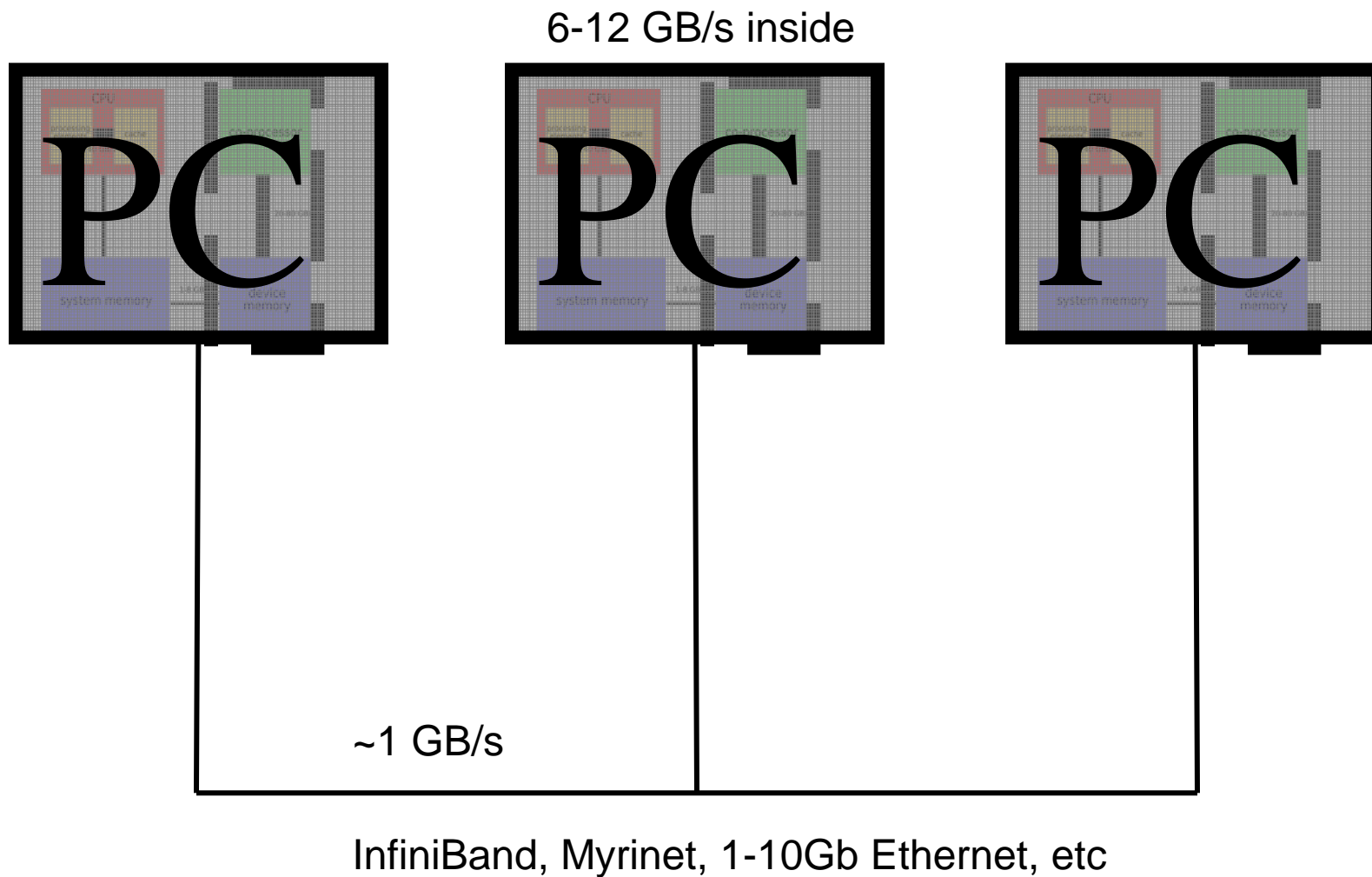
Data Distribution

- **On demand data transfers**
 - Data to GPU assignment **not known a-priori**
 - Classification would be expensive or impossible
 - Data tiles are **fetches from the host on-demand** (multiple requests)
 - E.g. Grid with data dependent gathers
- **Pros**
 - Handles **difficult dependencies**
 - Data transport to GPUs only slightly bigger than overall data
 - Each GPU examines only **its own data**
 - More fine-grained synchronization
- **Cons**
 - **Unpredictable** performance
 - **Many synchronizations** throughout the execution
 - **High device memory load** for multiple requests (latency hiding)

Accelerator Node with PCI (Express) Connection



GPU Cluster



GPU Cluster

- **Data Distribution**

- There is **no common memory**
- Host has the same problem as GPU, no global address space
 - Explicit solution: **Message Passing Interface (MPI)**
 - Implicit solution: **Virtual global address space**

- **Delays**

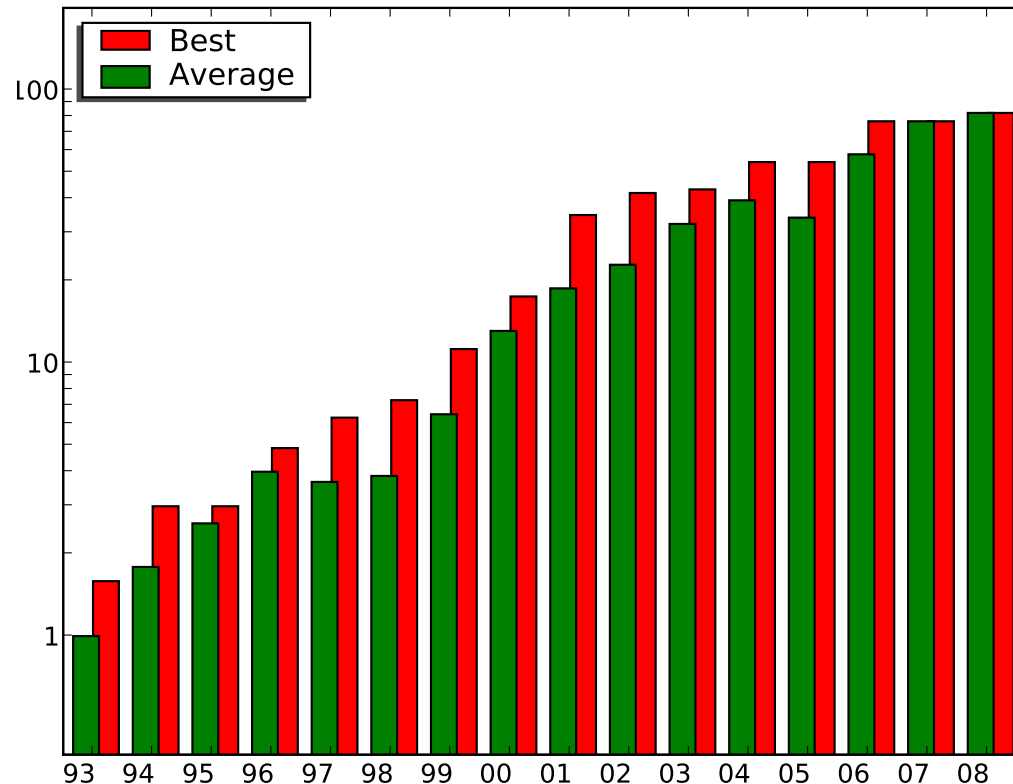
- GPU communication suffers **another indirection** through network
- Network calls for even **bigger data sizes** and **latency tolerance**
- Experiments with direct GPU-GPU by use of DVI connector

- **Challenges**

- Utilization of **heterogeneous** HW resources
- **Mixture** of coarse-grained (CPU) and fine-grained (GPU) parallelism
- Algorithms with **very high data localization** (spatial and temporal)
- Integration of HW acceleration into existing software packages

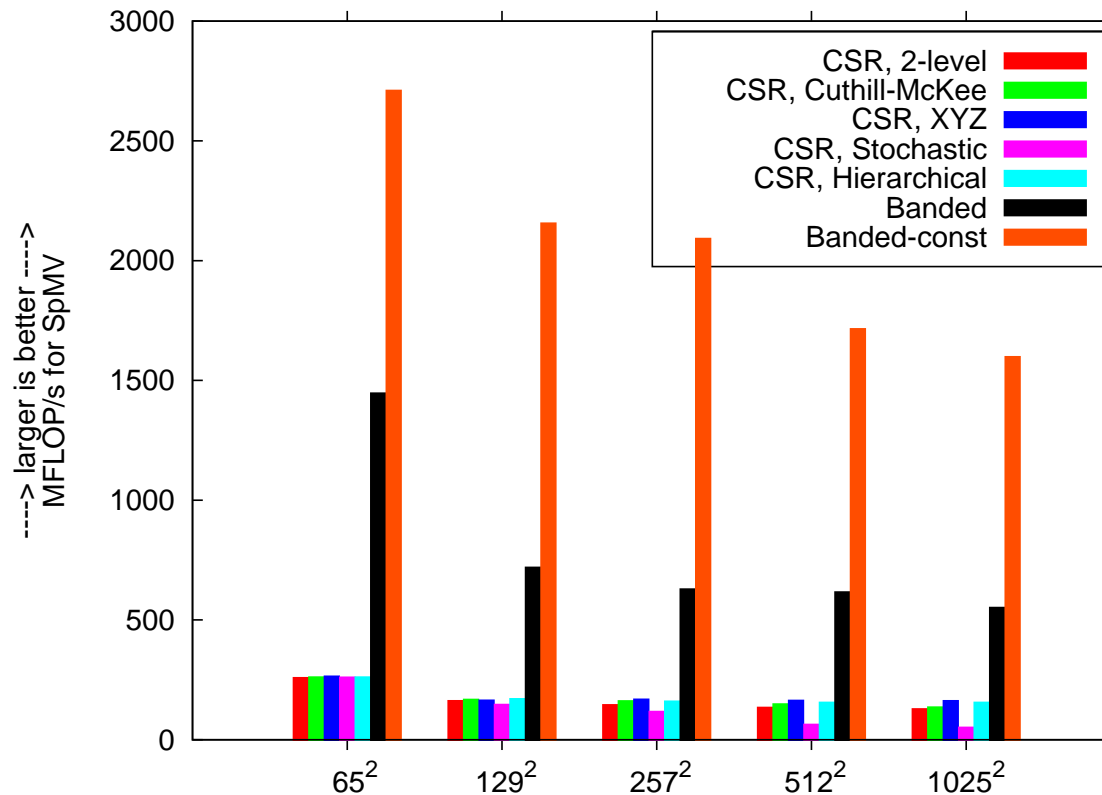
Large Scale SW-HW Integration

SW Performance: FeatFlow 1993–2008



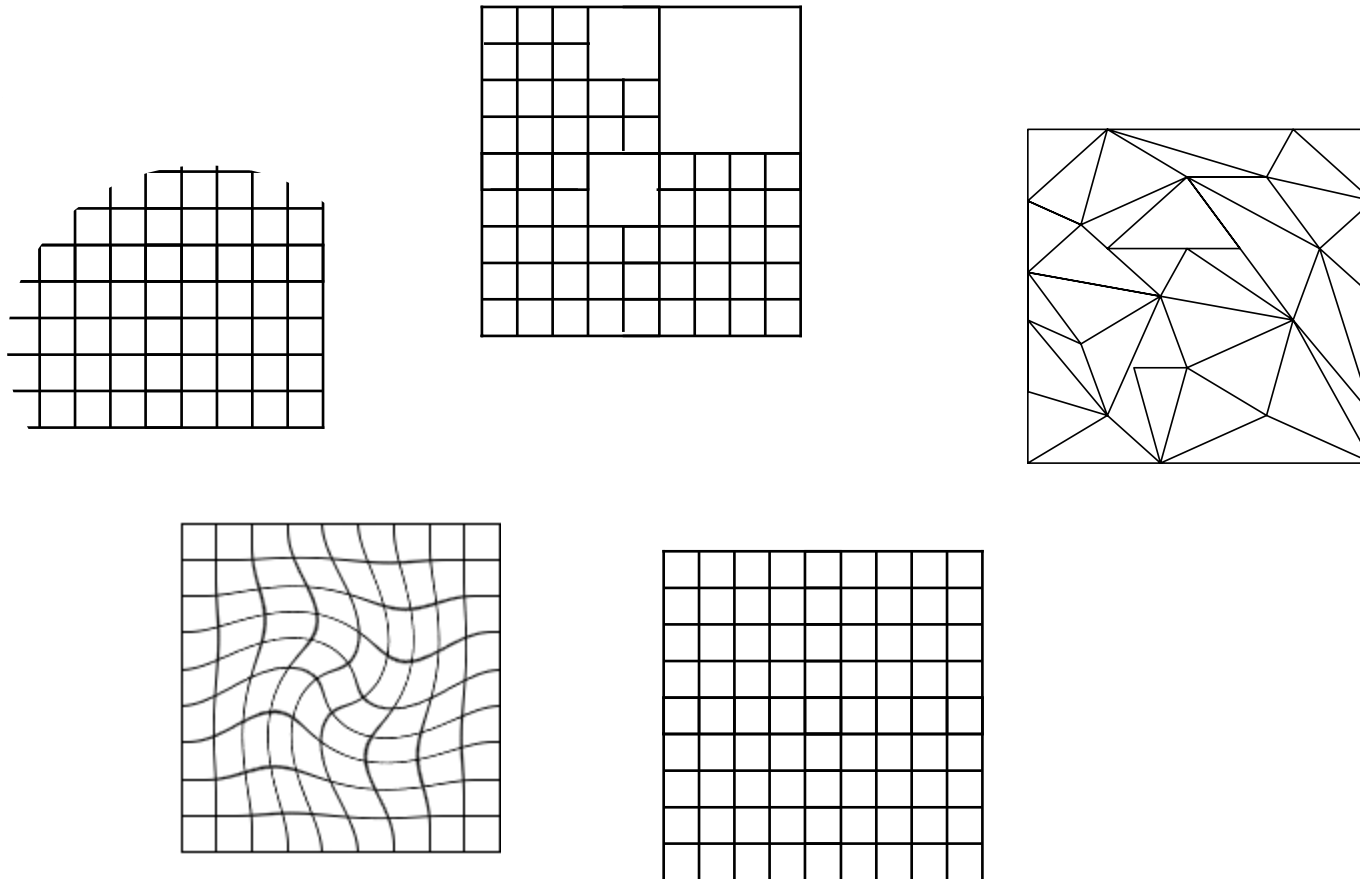
- **80x speedup** in 16 years for free
- **But: More than 1000x improvement** in peak processor performance
- **And: Performance gain stagnates**
- **Serial (legacy) codes no longer run faster automatically**

Sparse MatVec on Tensor Product Grid



- Opteron 2214, 2.2GHz, 2x1MB L2 cache, one thread
- 50 vs. 550MFLOP/s for interesting large problem size
- Cache-aware implementation \Rightarrow 90% of memory throughput
- const: Stencil-based computation

Discretization Grids



Sparse Computation Problems

- **Bandwidth** most crucial for sparse computations
- **Arithmetic intensity** of MatVec very low
- **Interconnects** between devices very slow
- **Explicit topology** in grids saves nodes through adaptivity but diminishes data locality
- **Large libraries** utilize no application specific, but rather **general data formats and solvers**

Parallelization of Large SW Packages

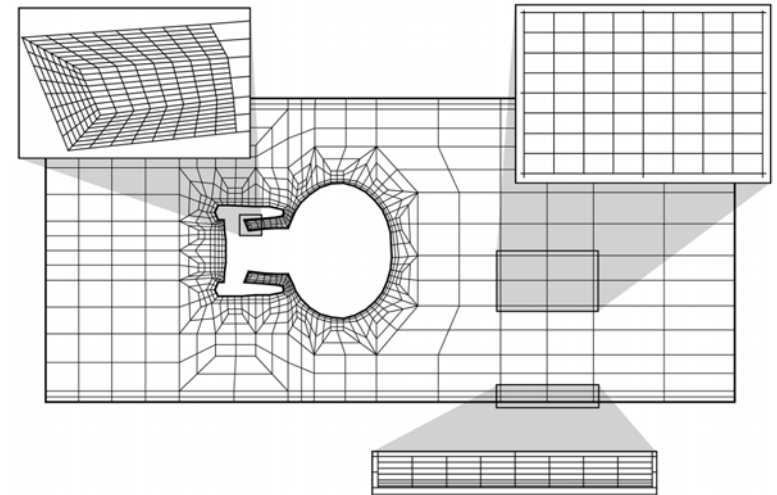
- FeatFlow superceded by FEAST
- FEAST (Finite Element Analysis & Solver Tools)
 - **Data** oriented library for parallel linear equation system solvers
 - **Central** scheme configured from solver blocks and parameters
 - Projects focusing on **large-scale** CFD and CSM simulations
 - >100k lines of code in **Fortran 77/90**
 - **Procedural** , **modular** style

[Stefan Turek et al. *Hardware-oriented numerics and concepts for PDE software*, 2006]

FEAST: Generalized Tensor-Product Grids

- **Sufficient flexibility in domain discretization**

- Global unstructured macro mesh, domain decomposition
- (an)isotropic refinement into local tensor-product grids



- **Efficient computation**

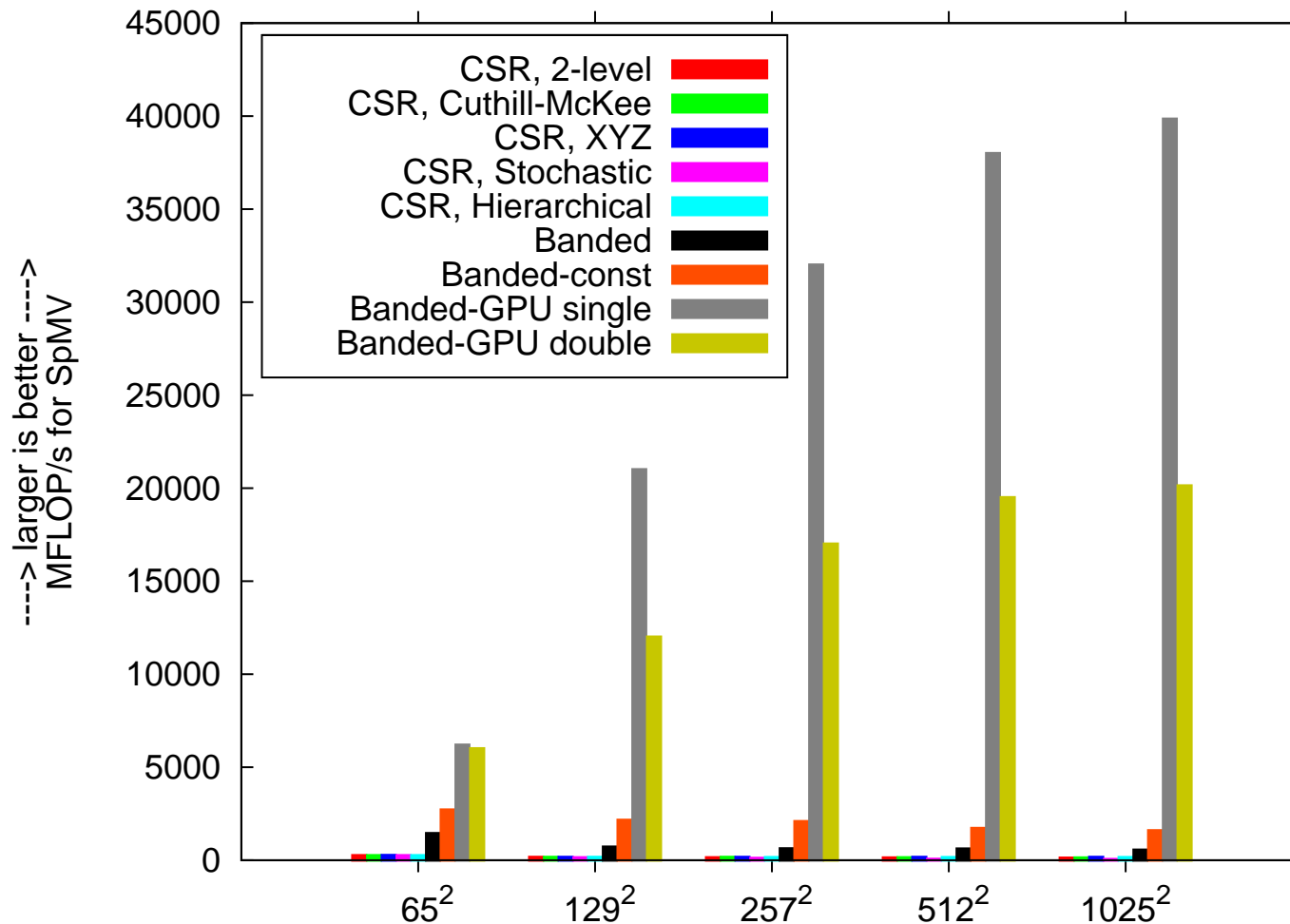
- High data locality, large problems map well to clusters
- Problem specific solvers depending on anisotropy level
- Hardware accelerated solvers on regular sub-problems

Scalable Recursive Clustering (ScaRC)

- **A tradeoff between global coupling and locality**
 - Run a global MG (coupling of data on all scales)
 - With a block-Jacobi smoother (local computation)

- **Typical 2-level ScaRC in FEAST**
 - global BiCGStab, preconditioned by
 - global multigrid (V-cycle, 1+1 steps), smoothed by
 - block-Jacobi, block (macro) inverses computed with a **local multigrid (V/F-cycle, 2+2 steps), smoothed by local Jacobi**

Sparse MatVec on Tensor Product Grid



- **13GFLOP/s with GPGPU on GeForce 8800 GTX**
- **40GFLOP/s, 140GB/s with CUDA on GeForce GTX 280**

Scalable Recursive Clustering (ScaRC)

- **A tradeoff between global coupling and locality**
 - Run a global MG (coupling of data on all scales)
 - With a block-Jacobi smoother (local computation)

- **Typical 2-level ScaRC in FEAST**

global BiCGStab, preconditioned by

global multigrid (V-cycle, 1+1 steps), smoothed by

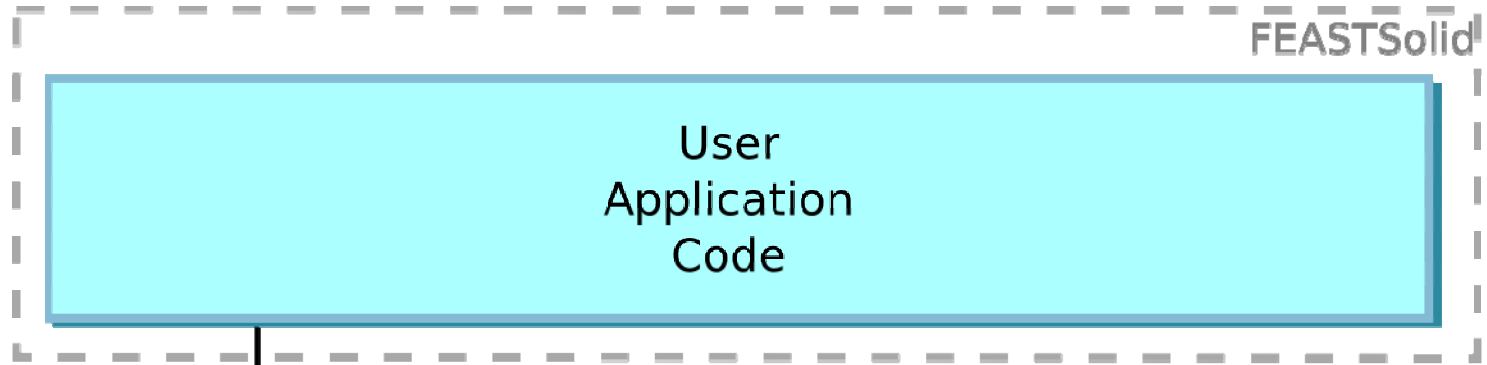
block-Jacobi, block (macro) inverses computed with a

local multigrid (V/F-cycle, 2+2 steps), smoothed by
local Jacobi

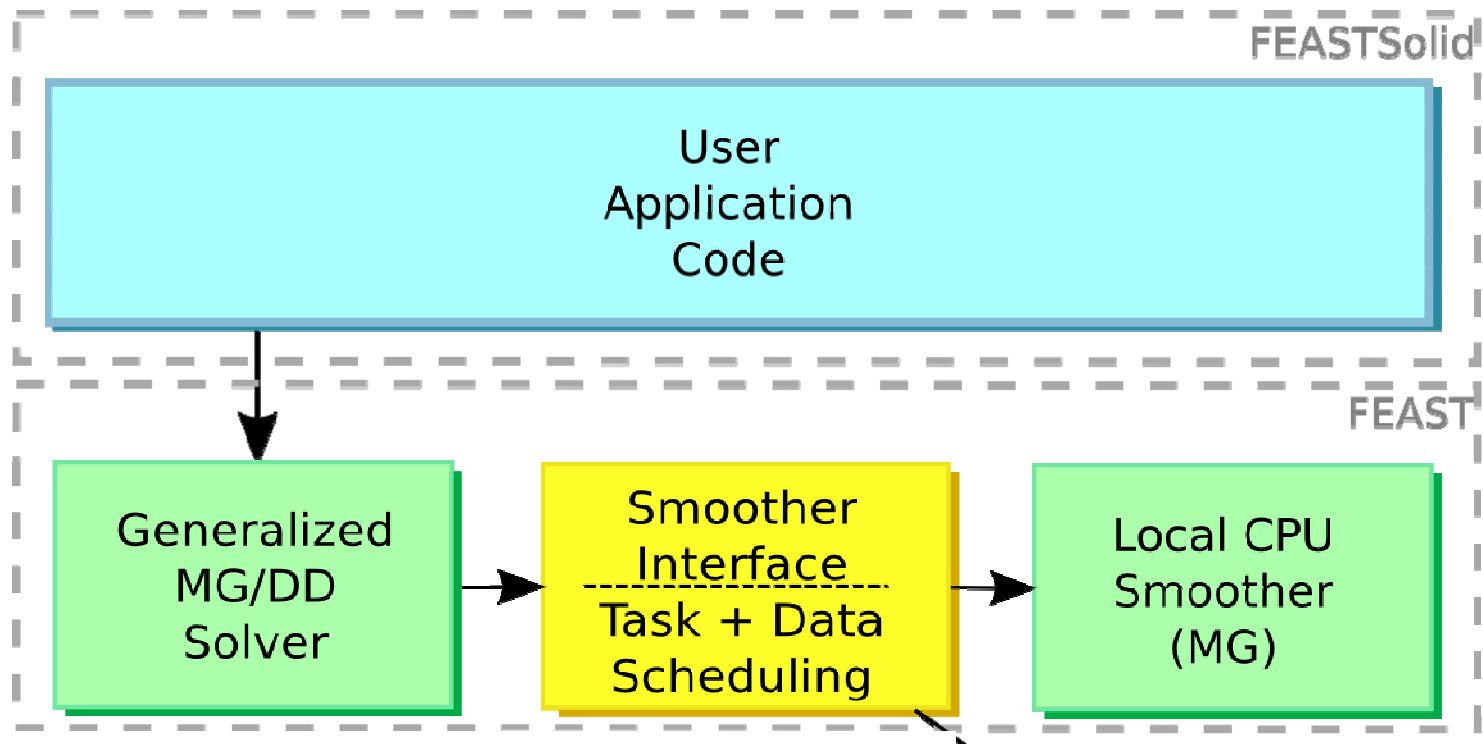


GPU

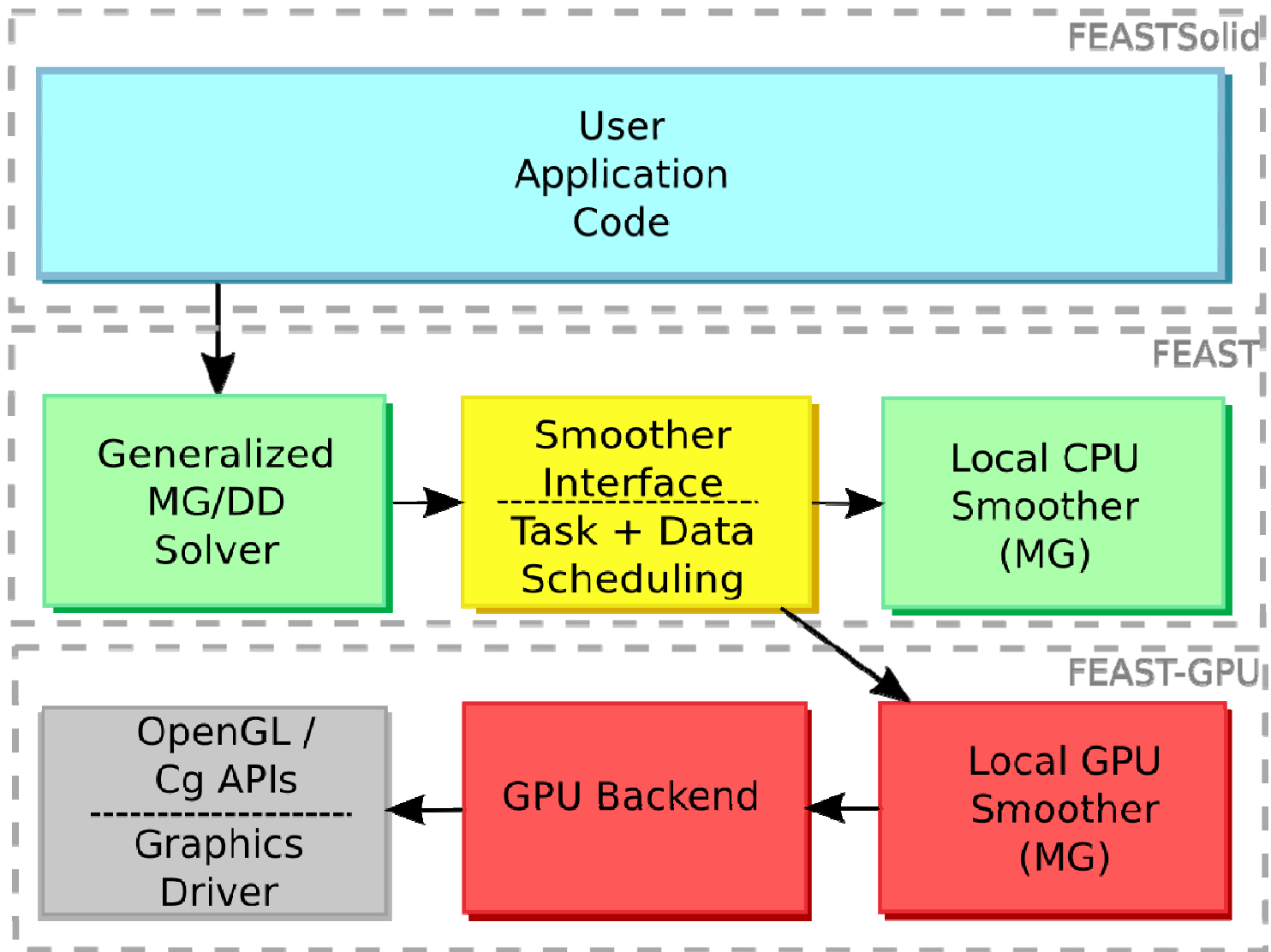
SW – HW Integration



SW – HW Integration



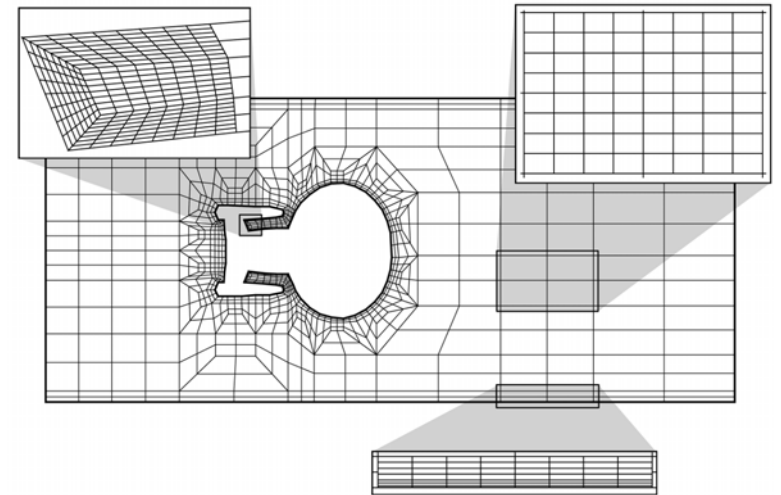
SW – HW Integration



FEASTGPU and Task Scheduler

- **CPU and GPU both execute what they are best at**

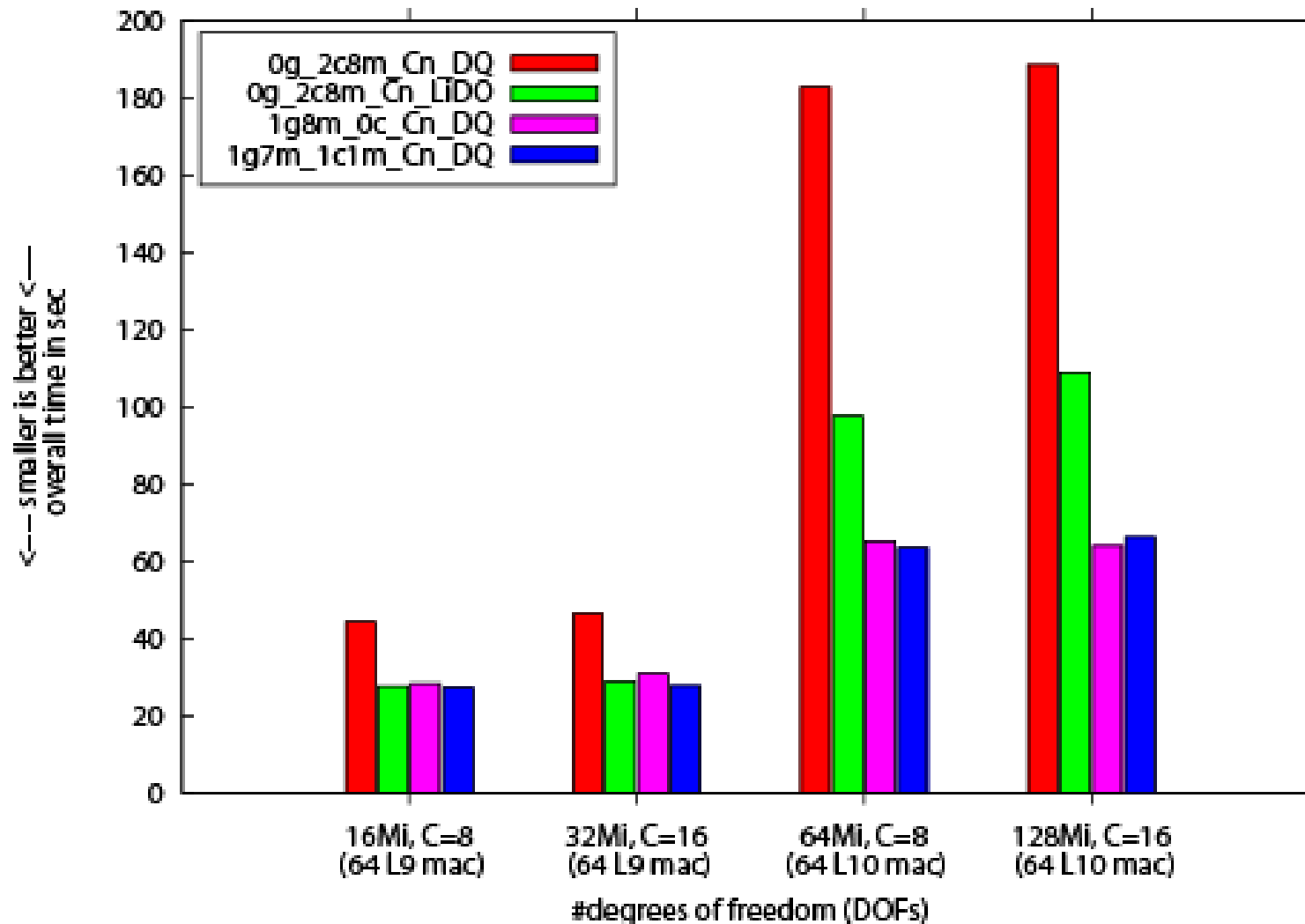
- CPU: **global** high precision MG
- CPU: **communication**
- GPU: **local** smoothing
- (CPU: **complex** local smoothers)
- Mixed precision MG-MG



- **Task scheduling to CPU/GPU depending on**

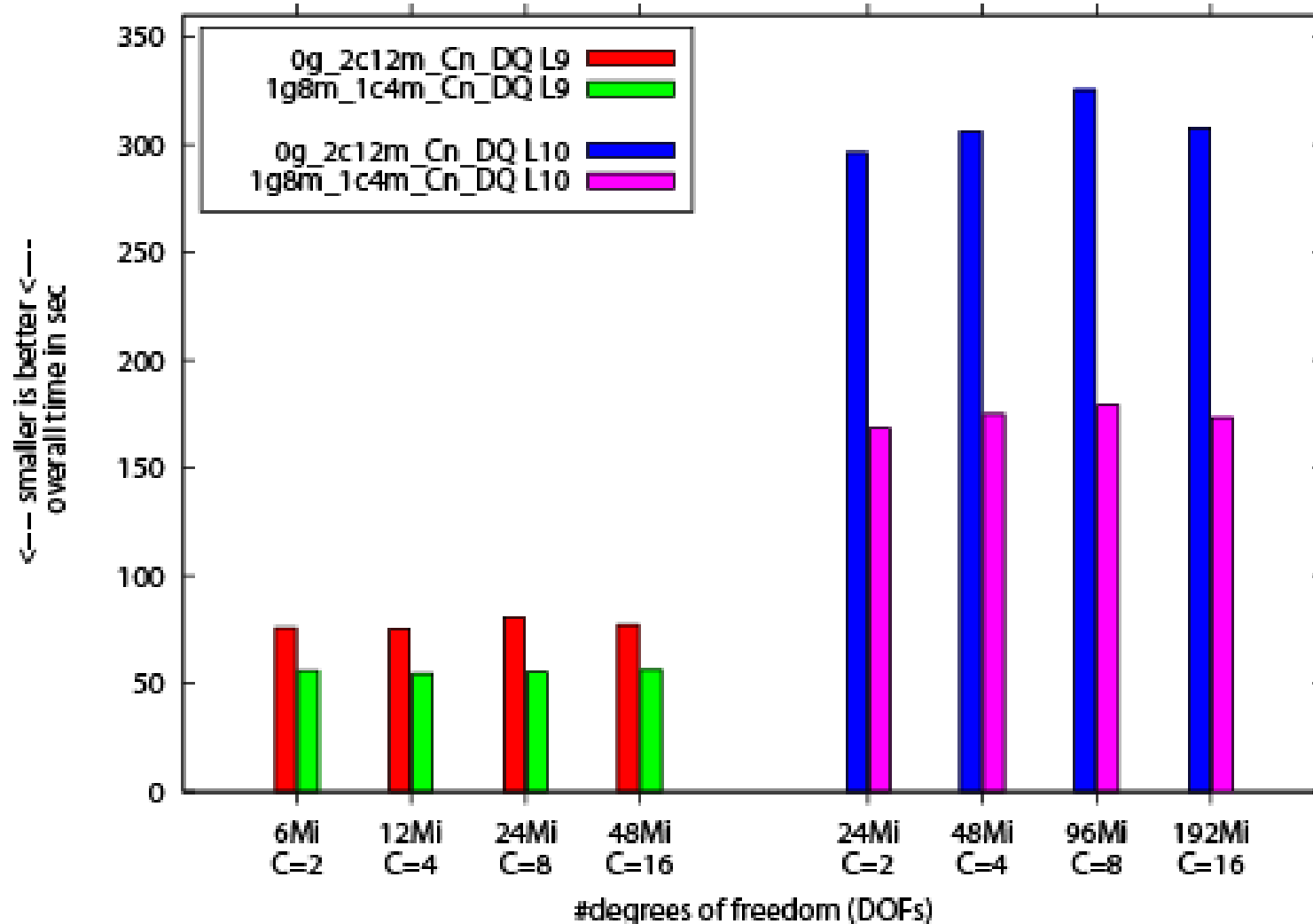
- **Type** of required solver (GPU supports only few solvers)
- **Size** of macro (local memory size&speed, communication overhead)
- Memory bus **congestion** (more important than hybrid Gflops)
- **Dynamic load** balance (not implemented yet)

CPU, GPU and Hybrid Configurations (FX4500)



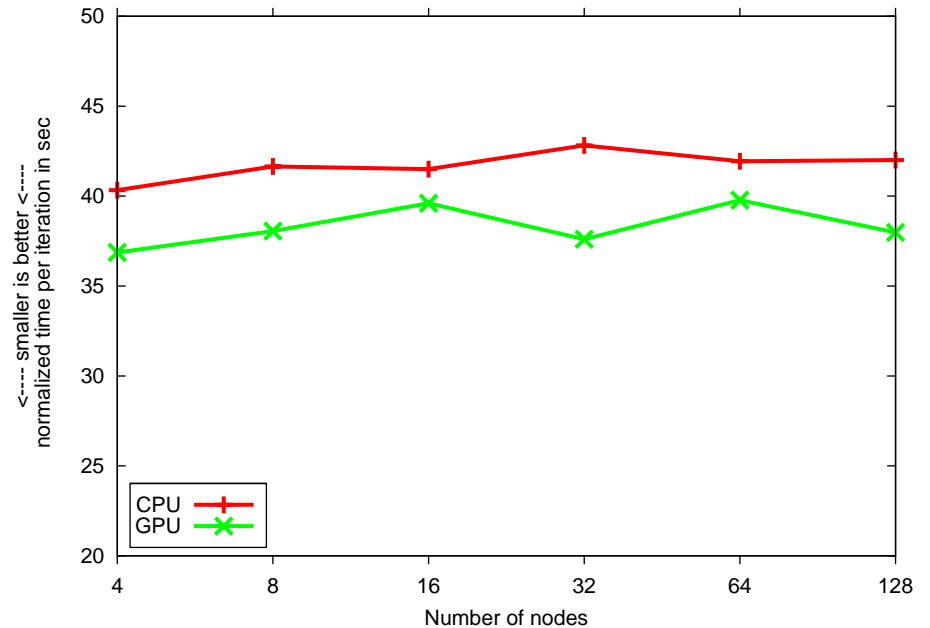
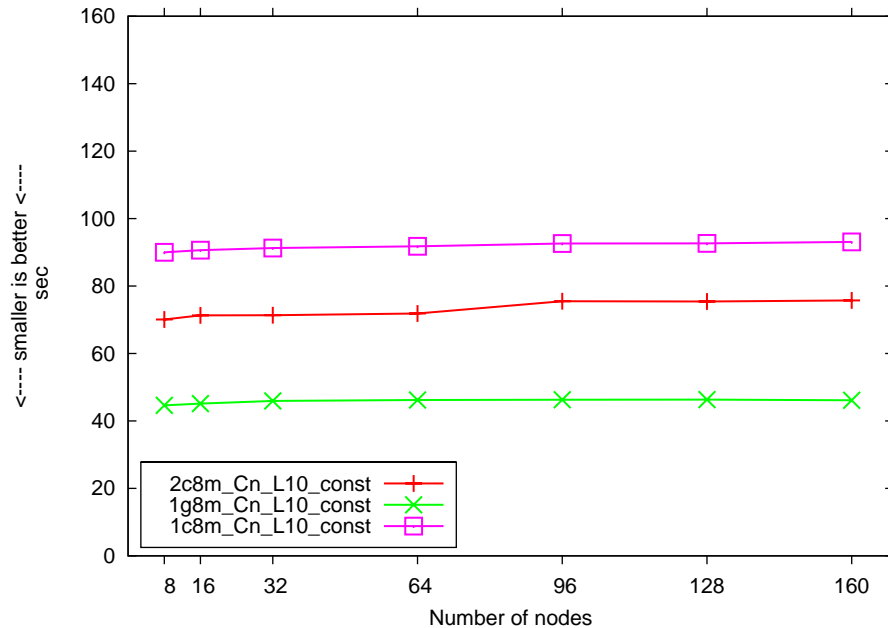
[Göddeke et al. *Using GPUs to Improve Multigrid Solver Performance on a Cluster*, IJCSE 2008]

Heterogeneous Domains and Solvers (FX4500)



[Göddeke et al. *Using GPUs to Improve Multigrid Solver Performance on a Cluster*, IJCSE 2008]

Weak Scalability on the CPU and GPU

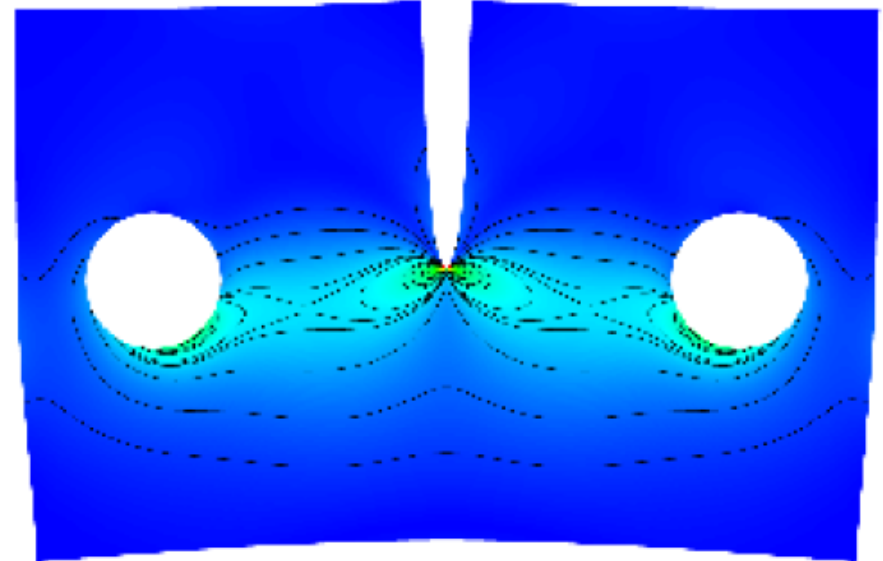
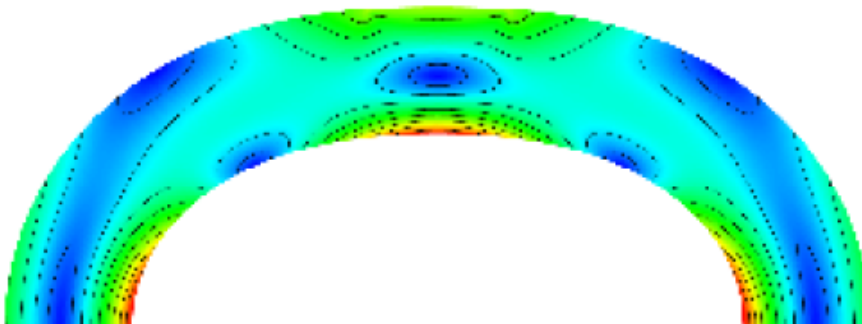
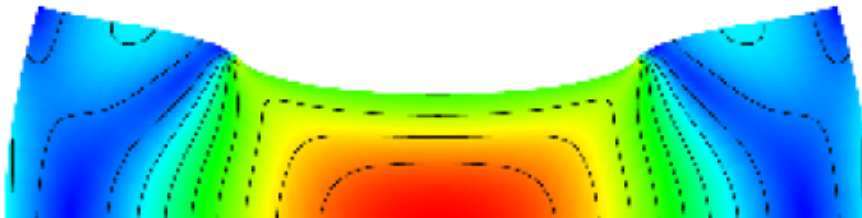
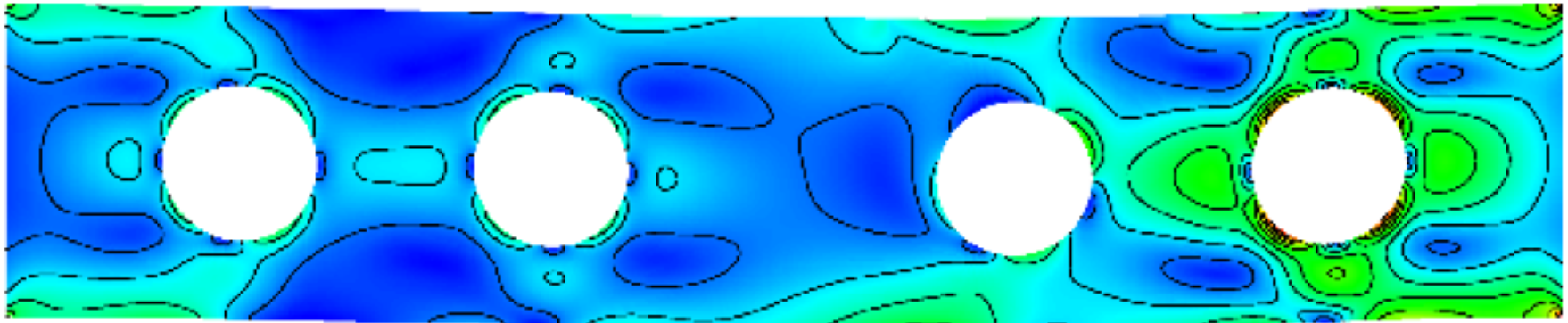


- **Old cluster, dual Xeon EM64T, one Quadro 1400 per node**
- **Poisson problem (left): up to 1.3B DOF, 160 nodes**
- **Elasticity (right): up to 1 B DOF, 128 nodes**

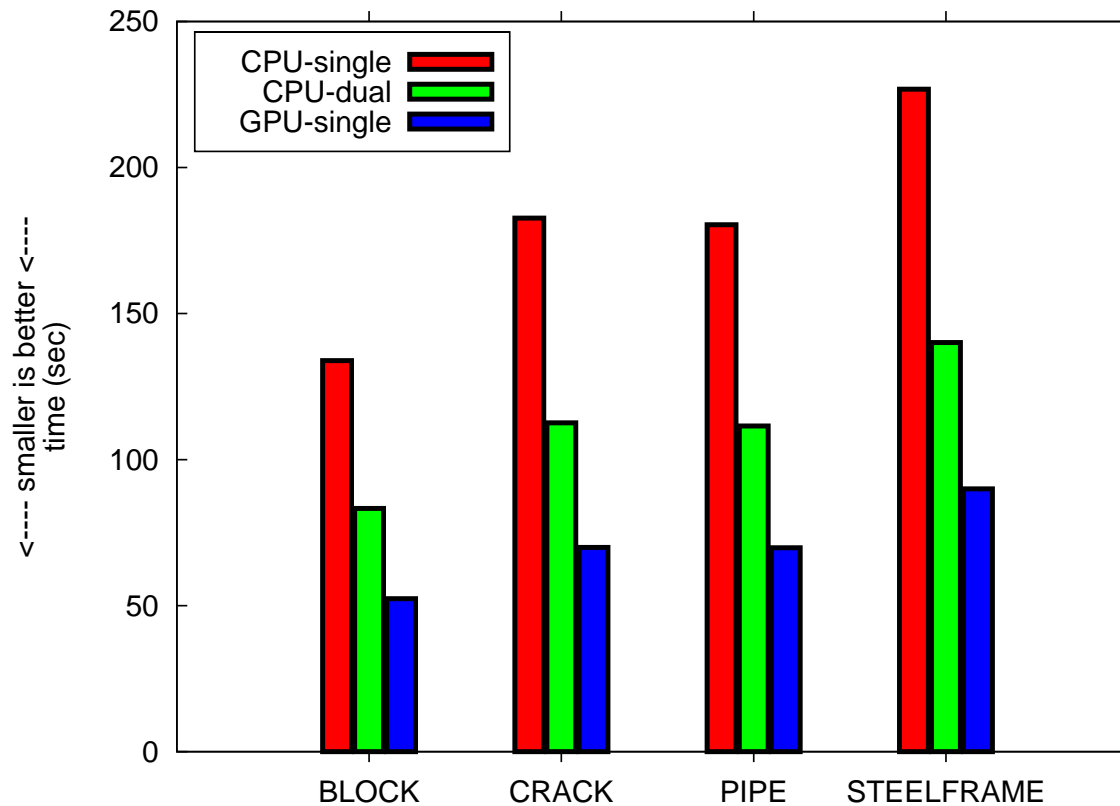
[Göddeke et al. *Exploring weak scalability for FEM calculations on a GPU-enhanced cluster*, Par.Computing 2007]

[Göddeke et al. *Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU*, IJSE 2009]

FEASTSolid: Von Mises Stress



FEASTSolid: Absolute Performance

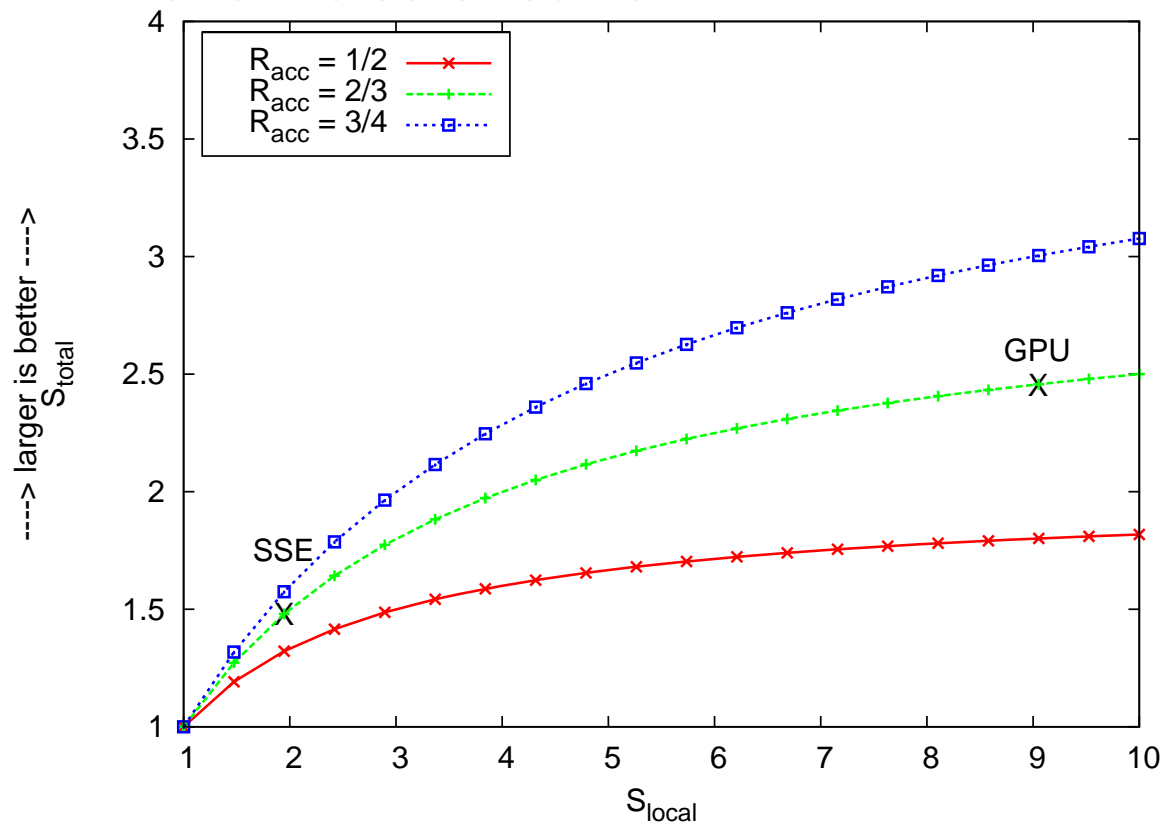


- **16 nodes, Opteron 2214 X2, Quadro 5600, OpenGL**
- **Problem size 128M DOF**
- **Dualcore 1.6x faster than singlecore**
- **GPU 2.6x faster than singlecore, 1.6x than dual**

FEASTSolid: Performance Analysis

- Addition of GPUs increases resources
- \Rightarrow Correct model: **strong scalability** inside each node
- Accelerable fraction of the elasticity solver: $2/3$
- Remaining time spent in MPI and the outer solver

- **Local speedup: 9x**
- **Global speedup: 2.6x**
- **Theoretical limit: 3x**



[Göddecke et al. *Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU*, IJCSE 2009]

Summary

- **Multi-GPU machines**

- Currently very **high discrepancy** in local to global bandwidth
- Type of **data distribution/synchronization** very important
- **API support** in its infancy
- In future HW **resource virtualization** is likely

- **GPU clusters**

- **Delays even worse** than for multi-GPU machines
- **Hierarchy of synchronization** levels (multi-CPU, multi-GPU)
- **Heterogeneity** can appear both on the node and cluster level

- **Large Scale HW-SW Integration**

- **HW oriented SW design** is crucial for future scalability
- Benefits apply both to classical CPU clusters and HW acceleration
- **“No changes to application code”** policy achievable, but new local scalability problems arise