Finite Element Multigrid Solvers for PDE Problems on GPUs and GPU Clusters

Robert Strzodka
Integrative Scientific Computing
Max Planck Institut Informatik
www.mpi-inf.mpg.de/~strzodka

Dominik Göddeke
Institute for Applied Mathematics
Technical University of Dortmund
www.mathematik.tu-dortmund.de/~goeddeke

Structure of Double Lecture 2 x 90 min

- **PART 1**

- **Parallelism**
- **Grid Discretization**
- **Multigrid & Smoothers**
- **Mixed-Precision**
- **Data Layout**

- **PART 2**

- **FEM on GPU Clusters**
- **New MPI Application (Rewrite)**
- **Legacy MPI Code (Accelerate)**

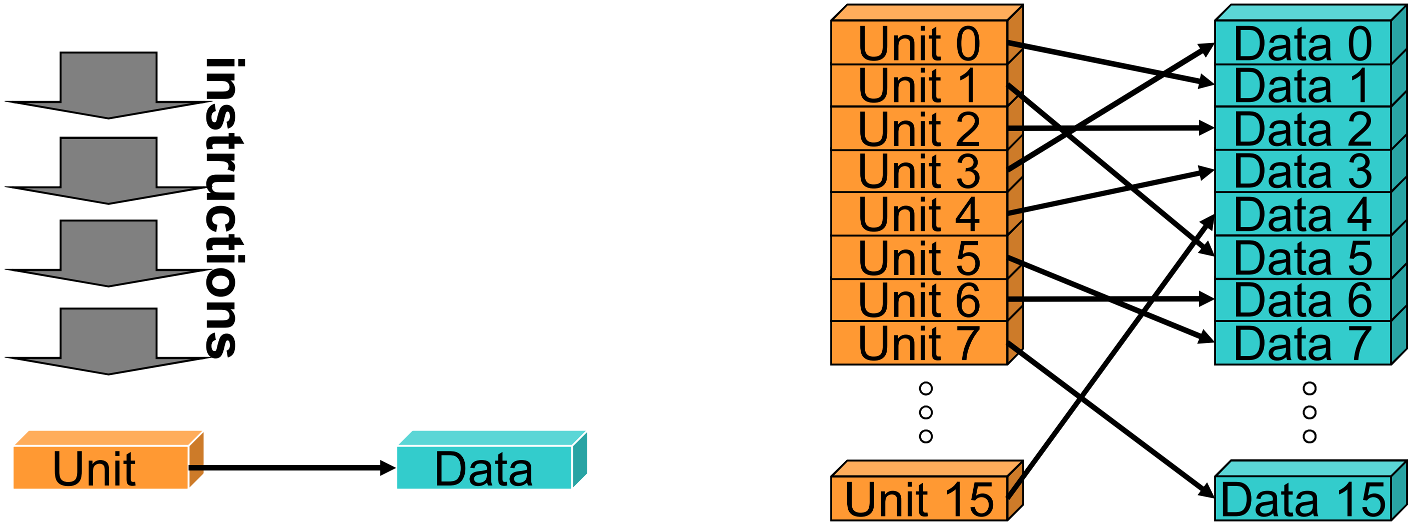
Overview

- Levels of Parallelism
- Grid Discretizations of PDEs
- Multigrid and Strong Smoothers
- Mixed Precision Iterative Refinement
- Layout of Multi-valued Data

Parallelism

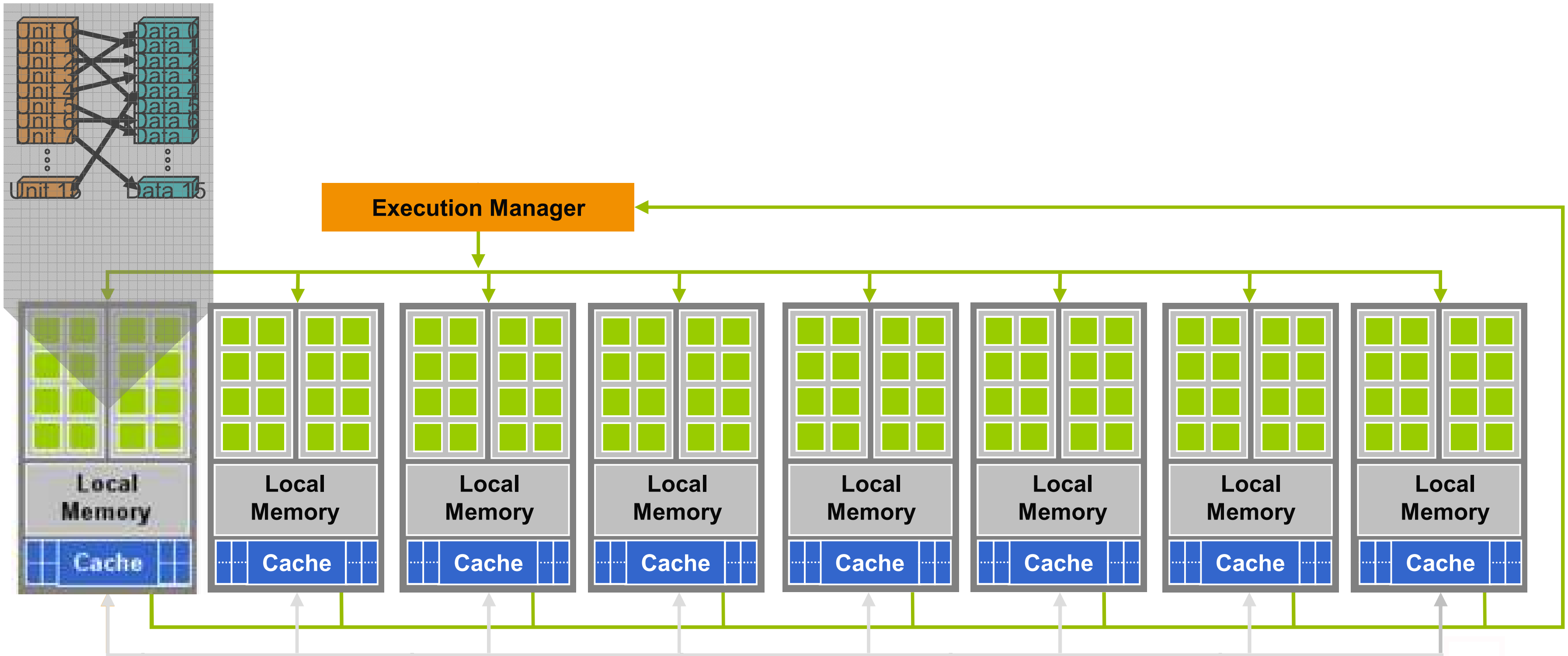
- **Sequential execution is an illusionary software concept**
- **All transistors always do something in parallel !**
- **Billions of transistors in modern CPUs(>0.5) & GPUs(>2)**
- **Old: Implicit parallelism with caches, ILP, speculation**
  → diminishing returns, power constraints
- **New: Explicit parallelism on multiple levels**
  → much more efficient & natural

SIMD Parallelism



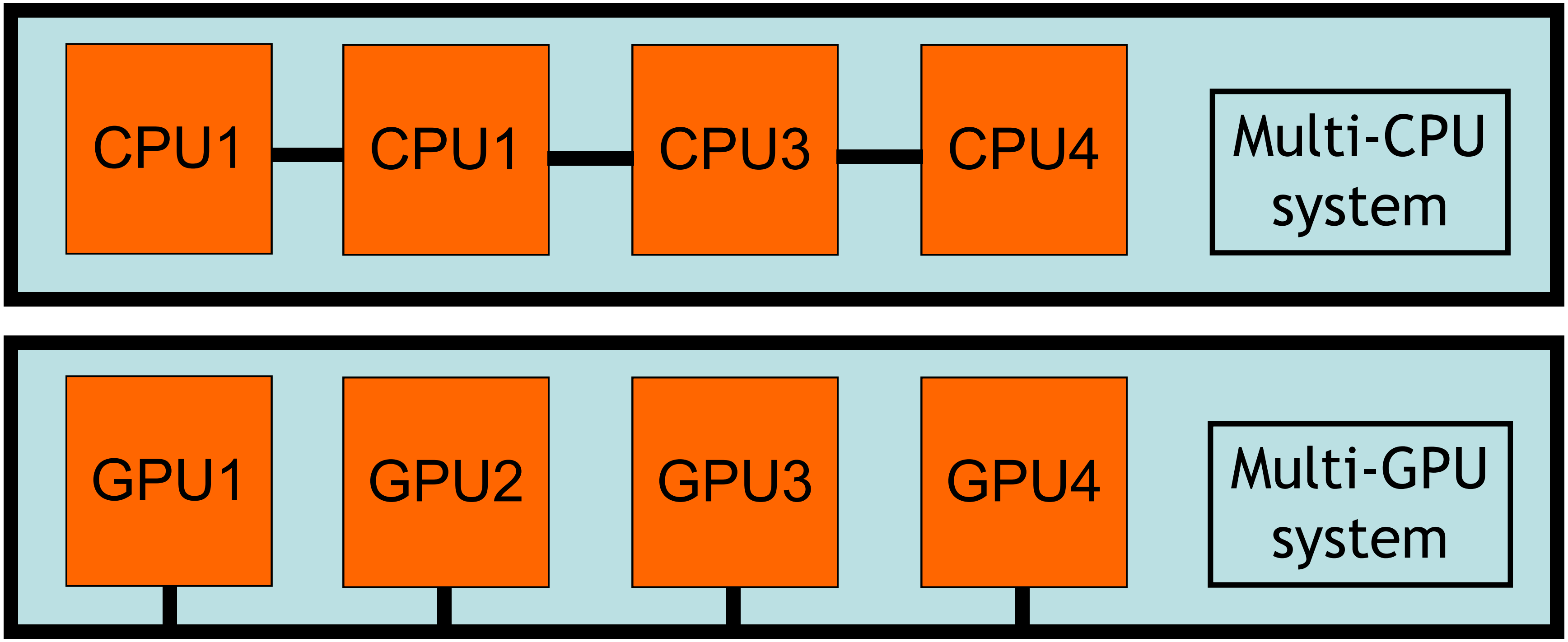
- **It is impossible to execute just one instruction**
  - a= b + c;
  - Actually means execute **(add, nop, nop, nop, …)**
- **Penalty for ignoring SIMD**
  - 4x on current CPUs (SSE)
  - 8-16x on future CPUs (AVX, LRB)
  - 16-80x on GPUs

Many-Core Parallelism



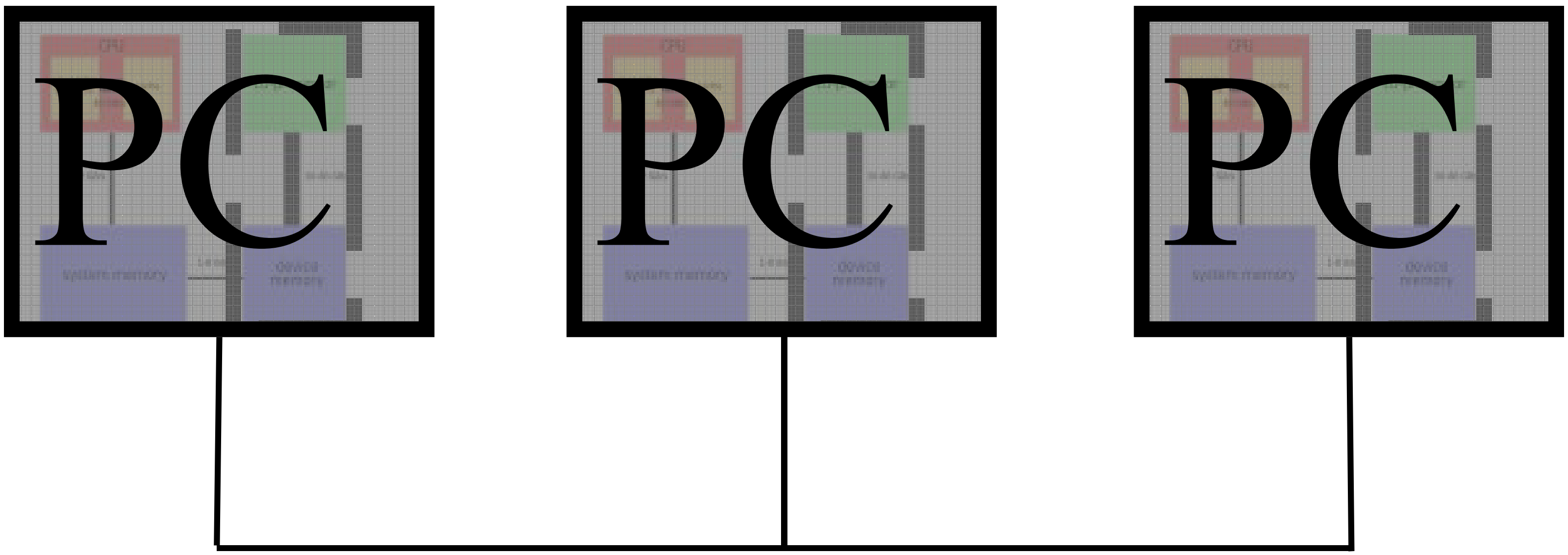
- **Penalty for ignoring many-cores**
  - 4-8x on current CPUs
  - 32-48x on future CPUs (LRB)
  - 10-30x on GPUs

Intra-Node Parallelism (multiple CPUs/GPUs per PC)



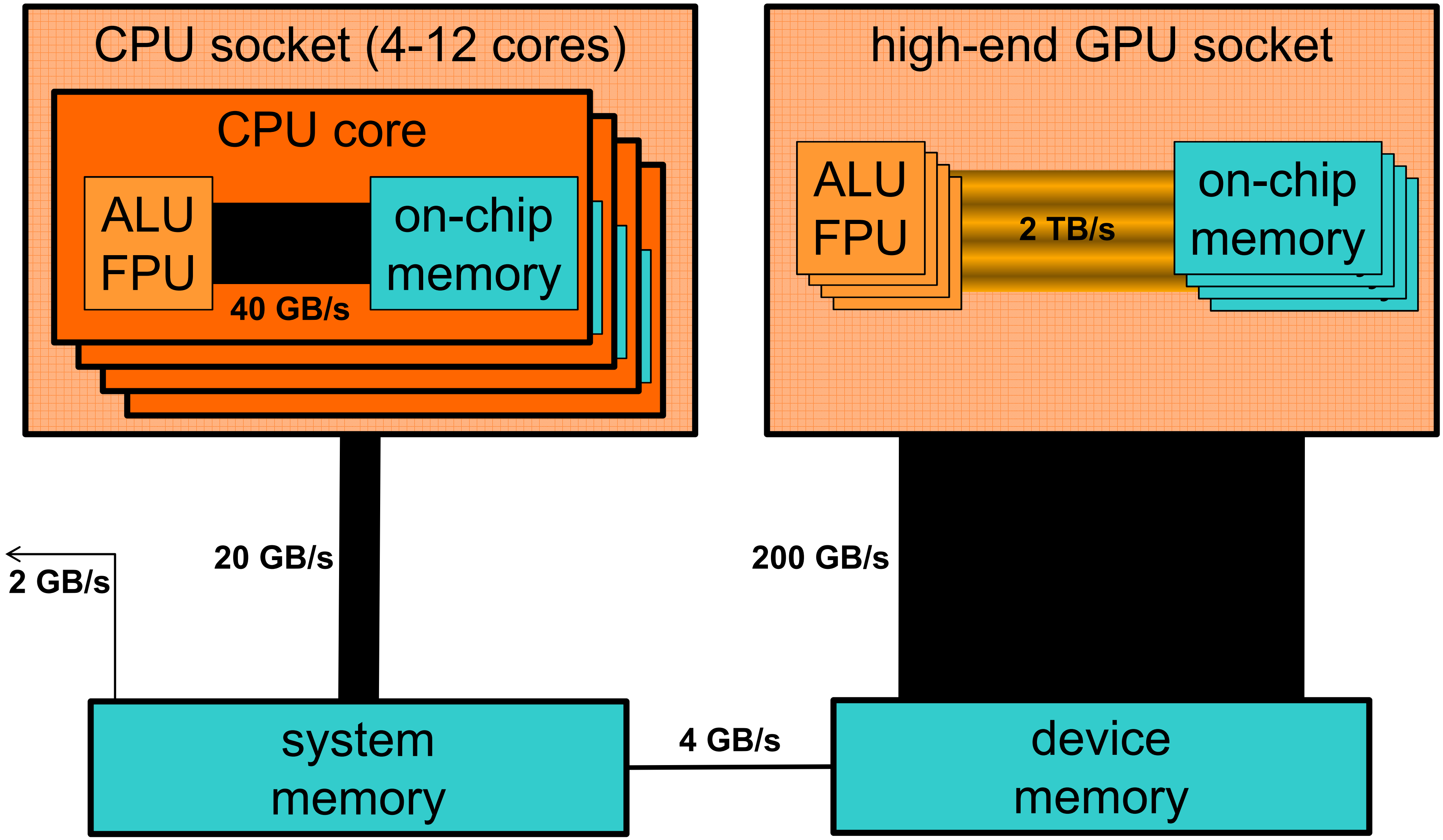
- **Penalty for ignoring intra-node parallelism**
  – 4-8x for multi-CPU systems
  – 4-8x for multi-GPU systems

Inter-Node Parallelism in a Cluster



- **Penalty for ignoring inter-node parallelism**
  - Depends on the number of nodes in the cluster
  - Capability computing

Bandwidth in a CPU-GPU System



Overview

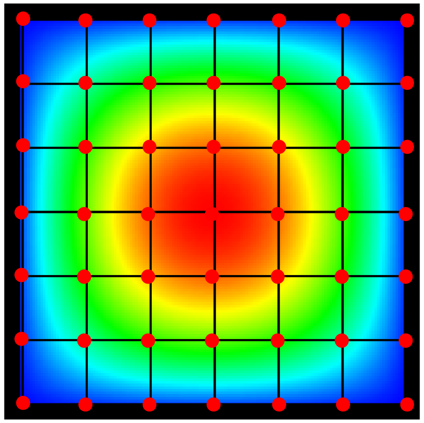
- Levels of Parallelism
- Grid Discretizations of PDEs
- Multigrid and Strong Smoothers
- Mixed Precision Iterative Refinement
- Layout of Multi-valued Data

Generalized Poisson Problem

We seek a function $u(x):\Omega \rightarrow \Re^m, \Omega \subseteq \Re^d$ which satisfies

interior $\quad -\operatorname{div}(\mathbf{G}\nabla\mathbf{u}) = \mathbf{f} \quad$ in Ω

boundary $\quad \partial_\nu \mathbf{u} = \mathbf{b}_N$ or $\mathbf{u} = \mathbf{b}_D \quad$ on $\partial\Omega$



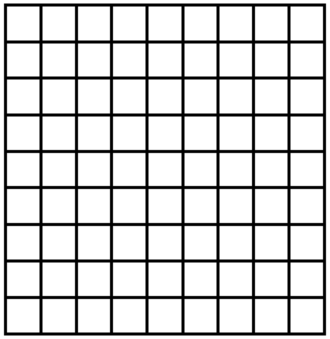
We consider the scalar (m=1) 2D case (d=2) with operator anisotropies.
Given a vector field ($v_1(x,y)$, $v_2(x,y)$) we define:

$$\mathbf{G} := \mathbf{R} \cdot \mathbf{S} \cdot \mathbf{R}^T$$

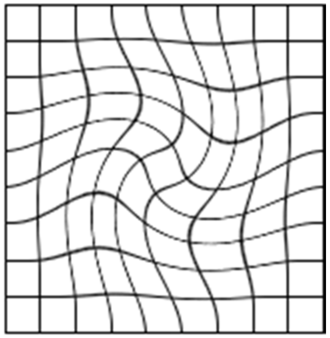
$$\mathbf{R}(x,y) := \frac{1}{\|\mathbf{v}(x,y)\|_2} \begin{pmatrix} v_1(x,y) & v_2(x,y) \\ -v_2(x,y) & v_1(x,y) \end{pmatrix}, \ \mathbf{S}(x,y) := \begin{pmatrix} \|\mathbf{v}(x,y)\|_2 & 0 \\ 0 & 1 \end{pmatrix}$$

Discretization Grids

- Equidistant grid
  - topology: implicit
  - geometry: implicit
  - access: **direct**

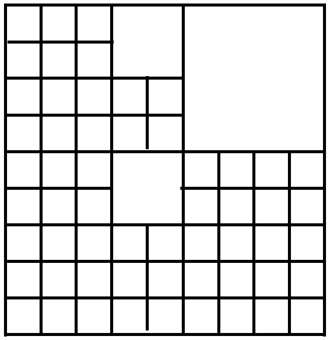


- Generalized tensor-product grid
  - topology: implicit
  - geometry: explicit
  - access: **direct**

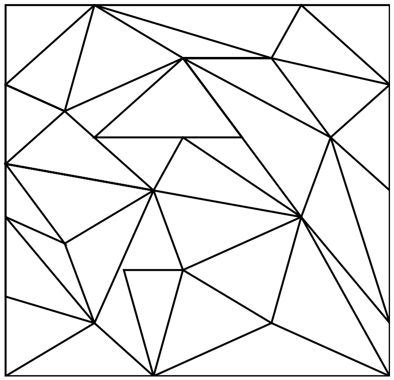


Discretization Grids

- Adaptive grid
  - topology: **explicit**
  - geometry: implicit/explicit
  - access: hash, tree or page table

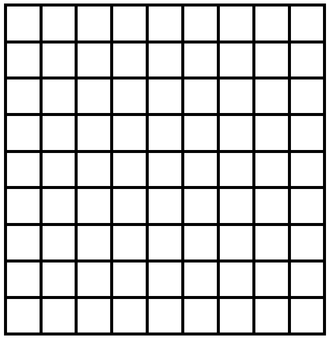


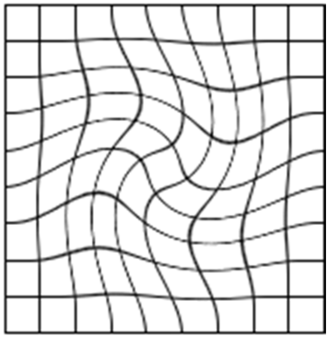
- Unstructured grid
  - topology: **explicit**
  - geometry: explicit
  - access: index array



nD Arrays

- Generalized tensor-product grid
  - topology: implicit
  - geometry: implicit/explicit
  - access: **direct**





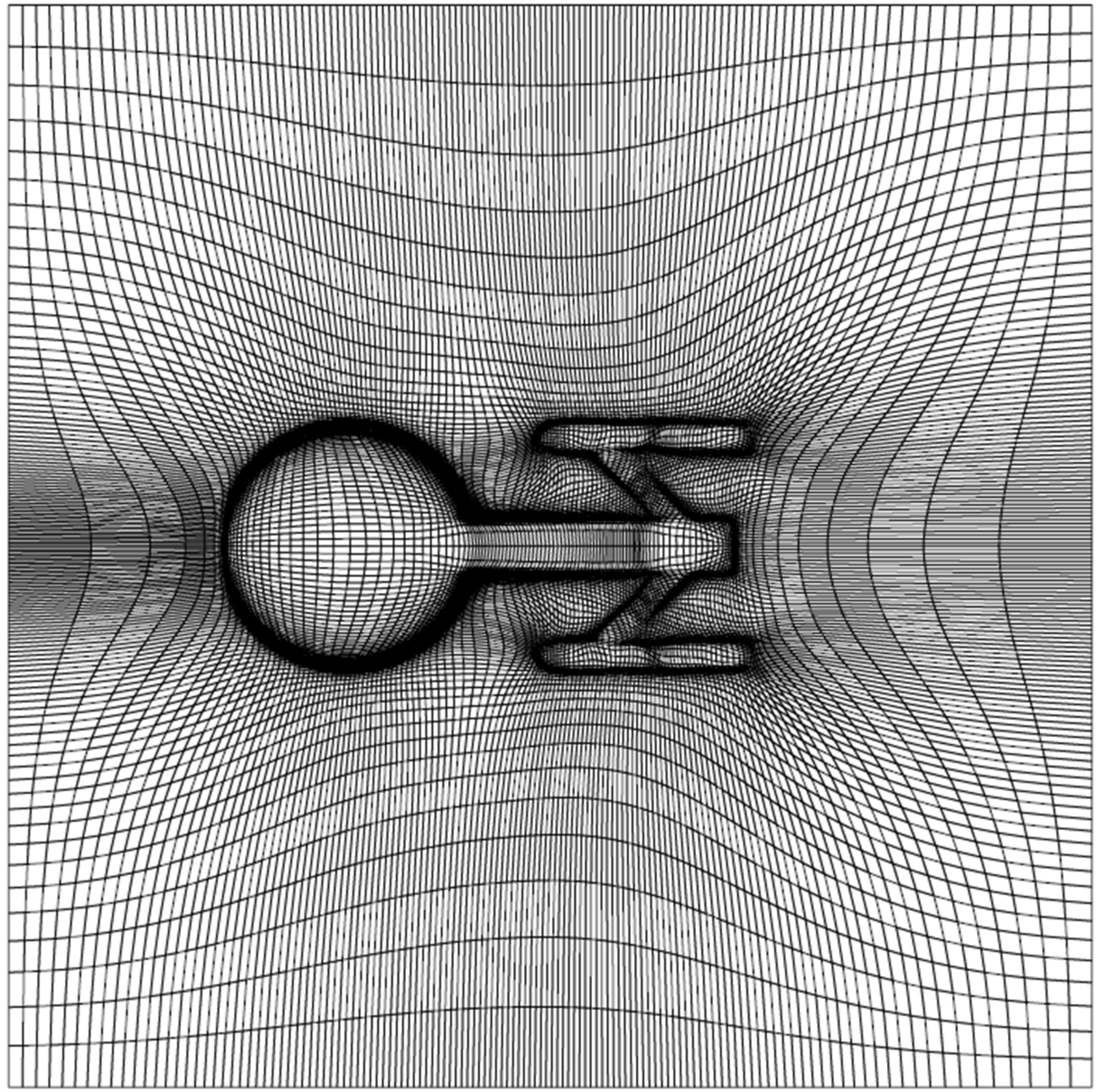
- Pros
  - **simple** implementation
  - implicit topology saves **precious** global bandwidth
  - allows efficient on-chip stencil operations
  - good SIMD utilization
- Cons
  - topology constraints make object modeling more difficult
  - element form may require a more powerful solver

Deformation Adaptivity

- This grid is a **tensor-product !**

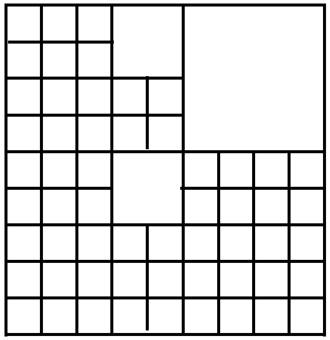
- Easier to **accelerate in hardware** than resolution adaptive grids

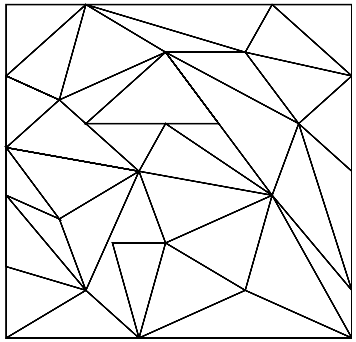
- **Anisotropy level** determines optimal solver



nD Arrays

- Adaptive/Unstructured grid
  - topology: **explicit**
  - geometry: implicit/explicit
  - access: indirect

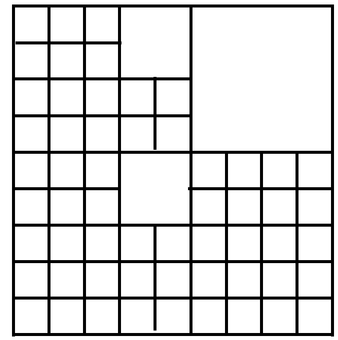


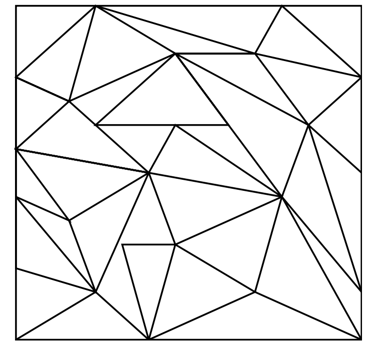


- Pros
  - general scheme for arbitrary node arrangements
  - only one indirection for **local** data access
  - clever node numberings preserve **some data locality**
- Cons
  - expensive encoding of topology/connectivity
  - no global view of the structure
  - difficult to handle dynamic changes in parallel

Hash

- Adaptive/Unstructured grid
  - topology: **explicit**
  - geometry: implicit/explicit
  - access: indirect

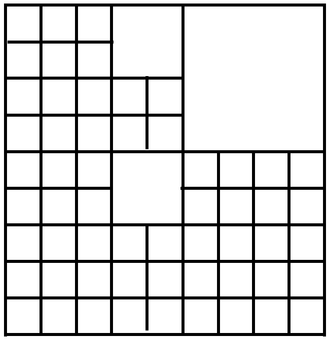




- Pros
  - general scheme for arbitrary node arrangements
  - only one indirection for **arbitrary global** data access
  - **perfect hashes** have no collisions, thus good SIMD use
- Cons
  - expensive encoding of topology/connectivity
  - hashes tend to produce fine-grained random data access
  - perfect hash generation too expensive for dynamic changes

Tree

- Adaptive grid
  - topology: **explicit**
  - geometry: implicit/explicit
  - access: indirect



- Pros
  - allows refinement of arbitrary depths
  - compact encoding of **global** topology/connectivity
  - allows **dynamic changes** in parallel
- Cons
  - several memory indirection in data access
  - difficult to extract SIMD parallelism

Structured and Unstructured Sparse MatVec

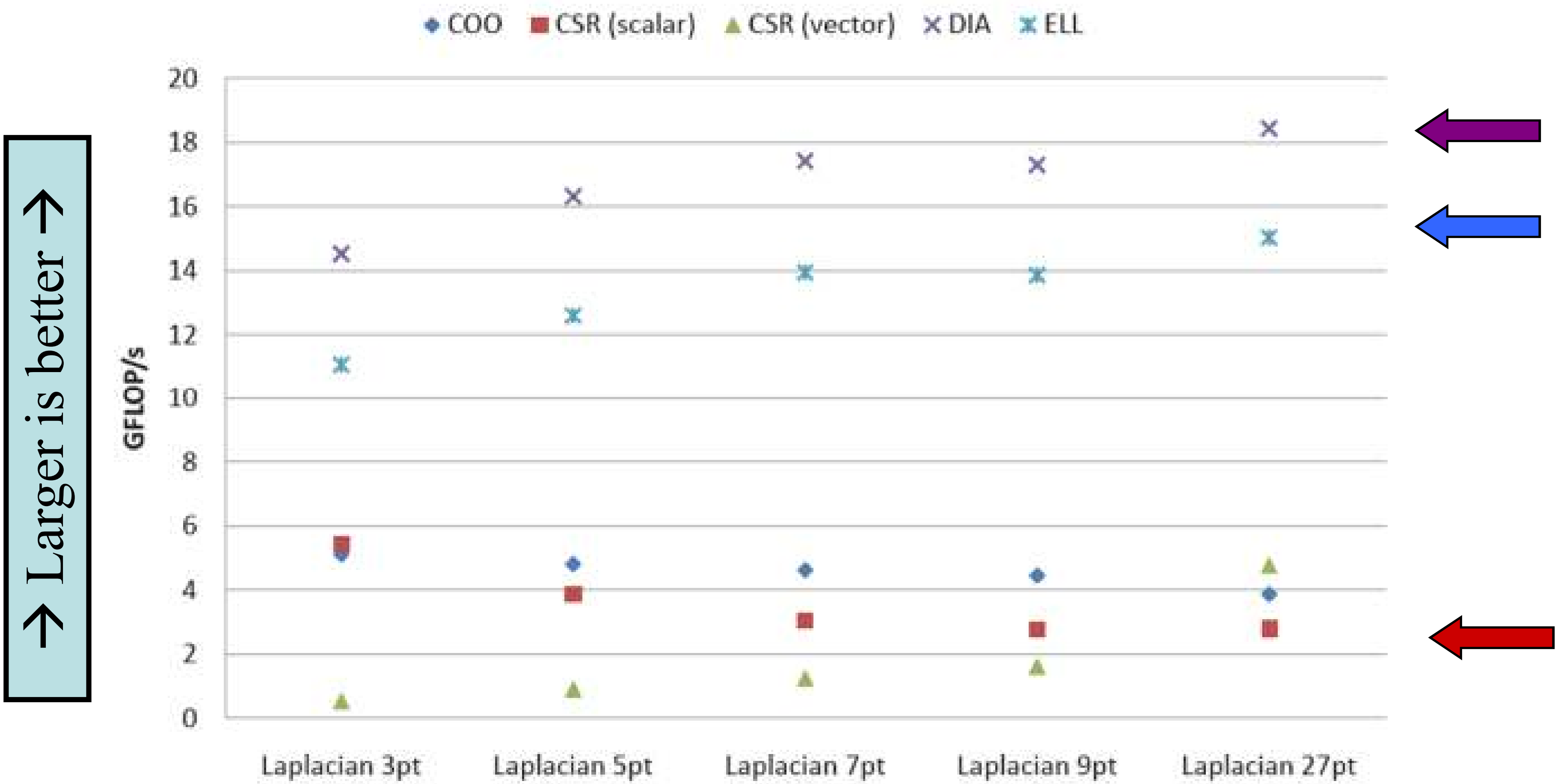


chart from [Bell and Garland SC 2009]

Overview

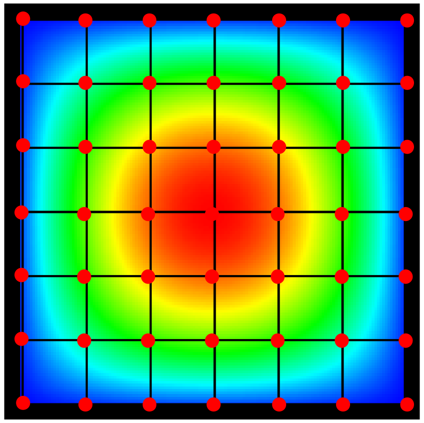
- Levels of Parallelism
- Grid Discretizations of PDEs
- Multigrid and Strong Smoothers
- Mixed Precision Iterative Refinement
- Layout of Multi-valued Data

Generalized Poisson Problem

We seek a function $u(x):\Omega \rightarrow \Re^m, \Omega \subseteq \Re^d$ which satisfies

interior $\quad -\mathrm{div}(\mathbf{G}\nabla \mathbf{u}) = \mathbf{f} \quad$ in Ω

boundary $\quad \partial_\nu \mathbf{u} = \mathbf{b}_N$ or $\mathbf{u} = \mathbf{b}_D \quad$ on $\partial\Omega$

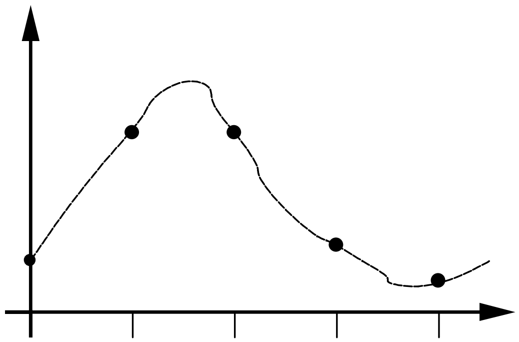


We consider the scalar (m=1) 2D case (d=2) with operator anisotropies.
Given a vector field ($v_1(x,y)$, $v_2(x,y)$) we define:

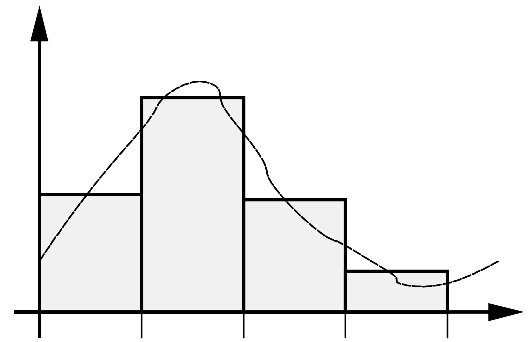
$$\mathbf{G} := \mathbf{R} \cdot \mathbf{S} \cdot \mathbf{R}^T$$

$$\mathbf{R}(x,y) := \frac{1}{\|\mathbf{v}(x,y)\|_2} \begin{pmatrix} v_1(x,y) & v_2(x,y) \\ -v_2(x,y) & v_1(x,y) \end{pmatrix}, \quad \mathbf{S}(x,y) := \begin{pmatrix} \|\mathbf{v}(x,y)\|_2 & 0 \\ 0 & 1 \end{pmatrix}$$

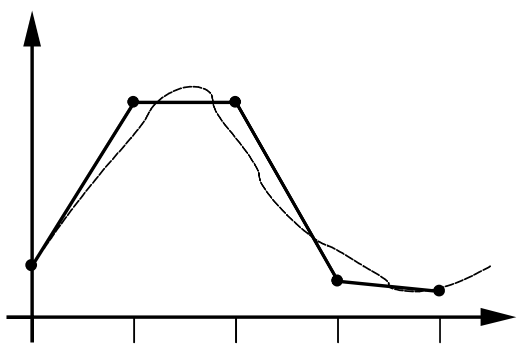
Discretization Approach



- Finite Differences



- Finite Volumes



- Finite Elements

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

For 2D linear FEM **A** is a 9-band matrix.

Geometric Multigrid Method

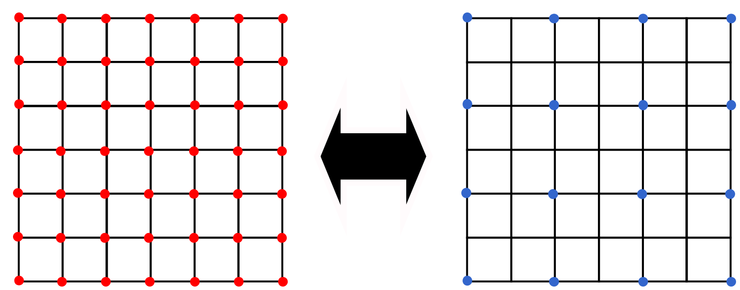
Linear equation system after discretization

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

Observation: Basic solvers quickly reduce the high frequency error components, but struggle with low frequencies

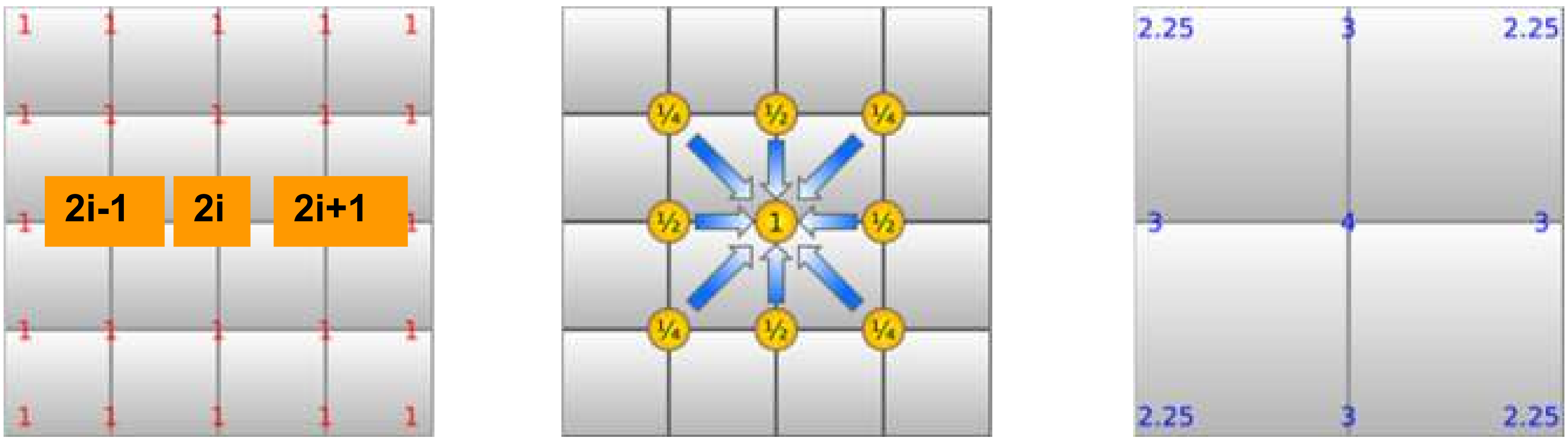
Idea: Solve the system on a pyramid of grids, thus dealing with different frequencies one after another

$d^k = b - Ax^k$	**Fine** grid
$Ac^k = d^k$	**Coarse** grid
$x^{k+1} = x^k + c^k$	Back on **fine** grid



Multigrid Transfers

- **Restriction**
  - Interpolate values from fine into coarse array
  - Local weighted gather operation



fine

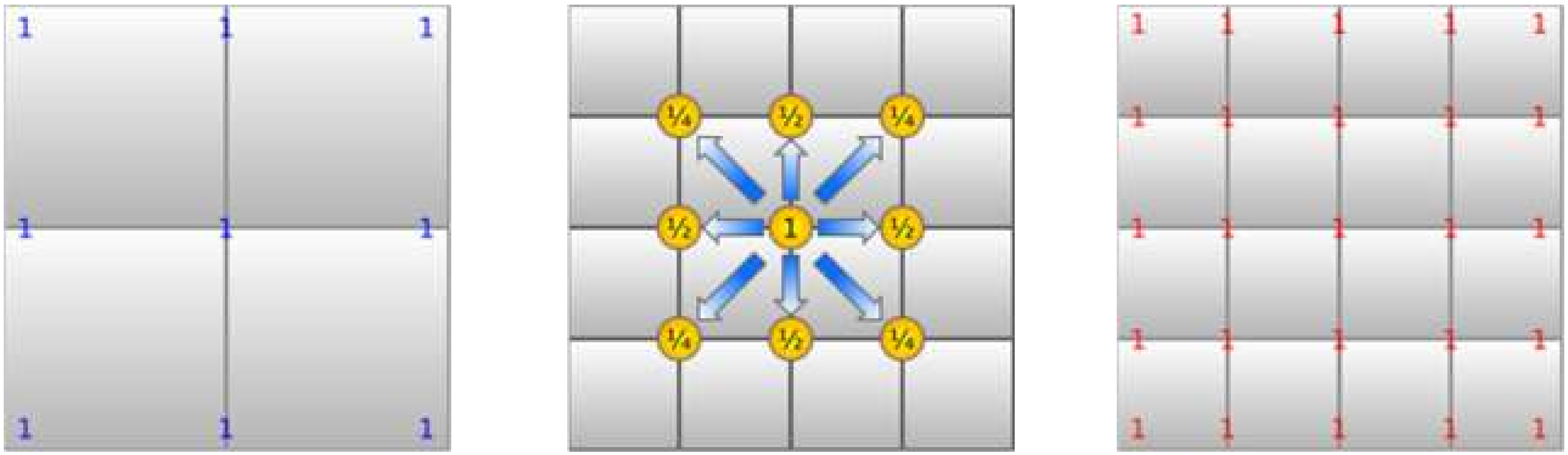
adjust index to read neighbors

output region coarse

result

Multigrid Transfers

- **Prolongation**
  - Scatter values from fine to coarse with weighting stencil
  - Local weighted scatter operation

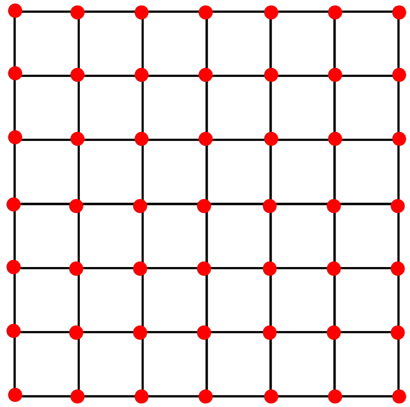


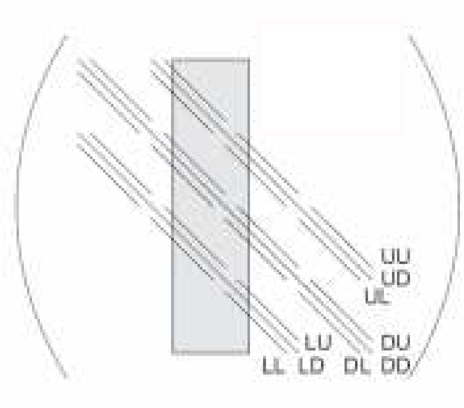
Preconditioners

$$\mathbf{Ax} = \mathbf{b}$$

Damped, preconditioned defect correction:

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + \omega \mathbf{C}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{k})$$





$$\mathbf{A} = (\mathbf{LL} + \mathbf{LD} + \mathbf{LU}) + (\mathbf{DL} + \mathbf{DD} + \mathbf{DU}) + (\mathbf{UL} + \mathbf{UD} + \mathbf{UU})$$

JACOBI $\quad \mathbf{C} = \mathbf{DD}$

GSROW $\quad \mathbf{C} = (\mathbf{LL} + \mathbf{LD} + \mathbf{LU}) + (\mathbf{DL} + \mathbf{DD})$

TRIDI $\quad \mathbf{C} = (\mathbf{DL} + \mathbf{DD} + \mathbf{DU})$

TRIGSROW $\quad \mathbf{C} = (\mathbf{LL} + \mathbf{LD} + \mathbf{LU}) + (\mathbf{DL} + \mathbf{DD} + \mathbf{DU})$

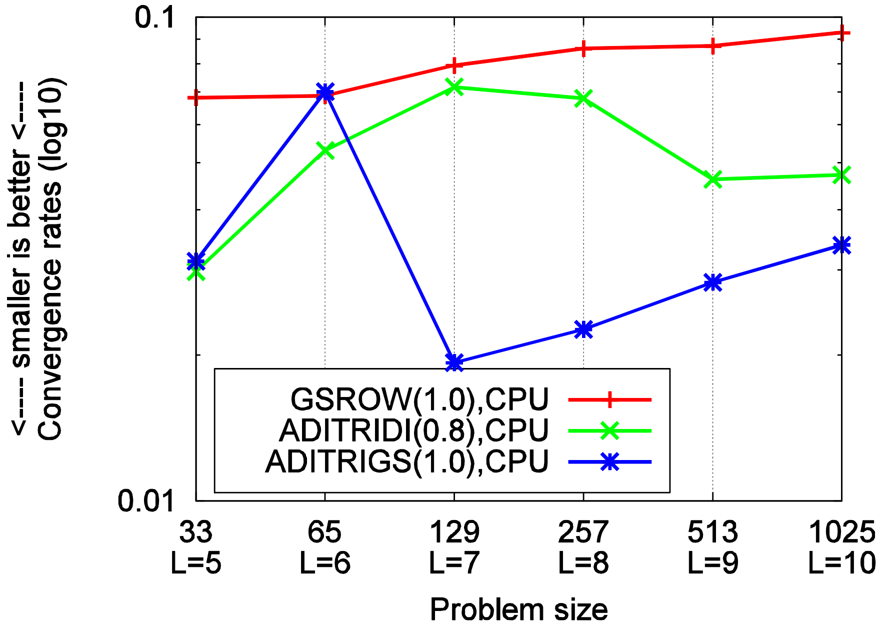
CPU Numerical and Runtime Efficiency

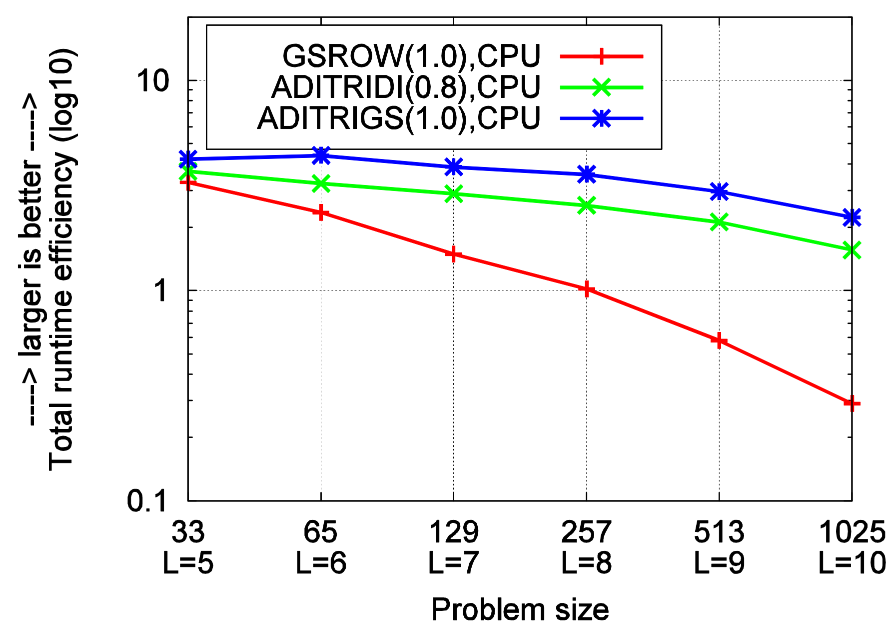
Iter.Ref.(double) MG(float) V(2,2) CG

x = 0,34??????

$$\rho := \left(\frac{\|\mathbf{A}\mathbf{x}^k - \mathbf{b}\|_2}{\|\mathbf{A}\mathbf{x}^0 - \mathbf{b}\|_2} \right)^{1/k}$$

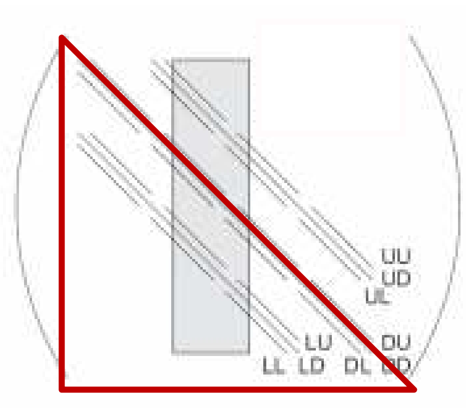
$$t_{\text{rel}} := \frac{t_{\text{total}} \cdot 10^6}{N \cdot k \cdot \log_{10} \rho}$$





Gauss-Seidel Preconditioner

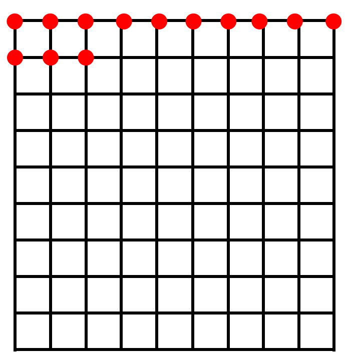
$$\mathbf{Ax} = \mathbf{b}$$

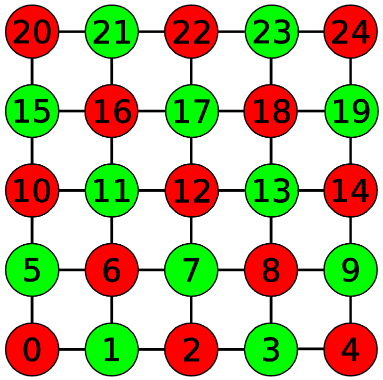


Damped, preconditioned defect correction:

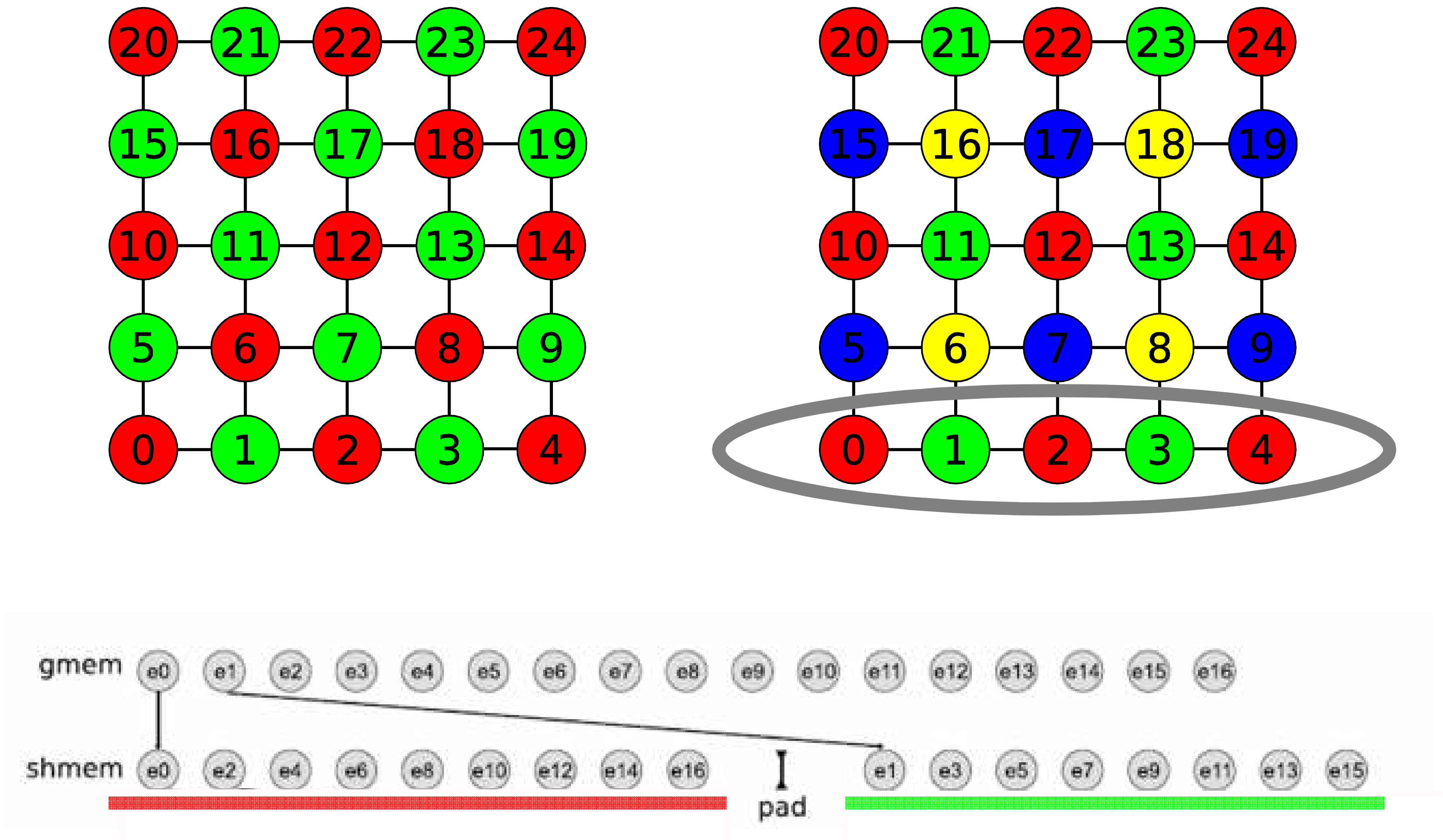
$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + \omega \mathbf{C}^{-1}(\mathbf{b} - \mathbf{Ax}^{k})$$

GSROW $\mathbf{C} = (\mathbf{LL} + \mathbf{LD} + \mathbf{LU}) + (\mathbf{DL} + \mathbf{DD})$



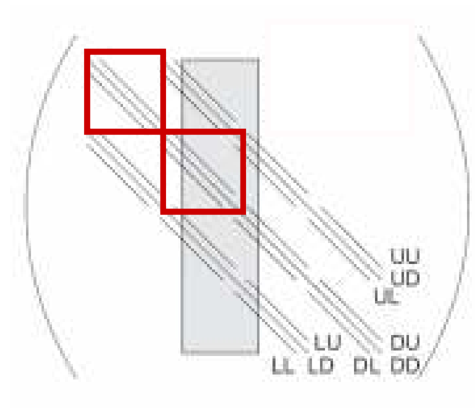


Multi-Colored Gauss-Seidel



ADI-TRIDI Preconditioner

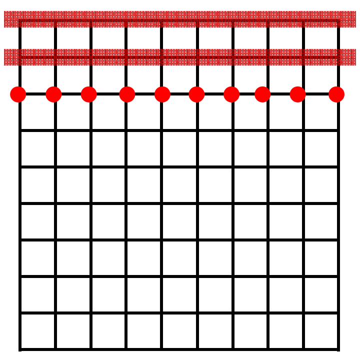
$$\mathbf{Ax} = \mathbf{b}$$



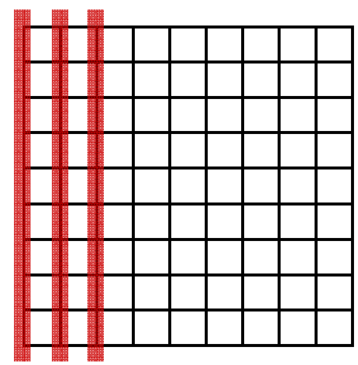
Damped, preconditioned
defect correction:

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + \omega \mathbf{C}^{-1}(\mathbf{b} - \mathbf{Ax}^{k})$$

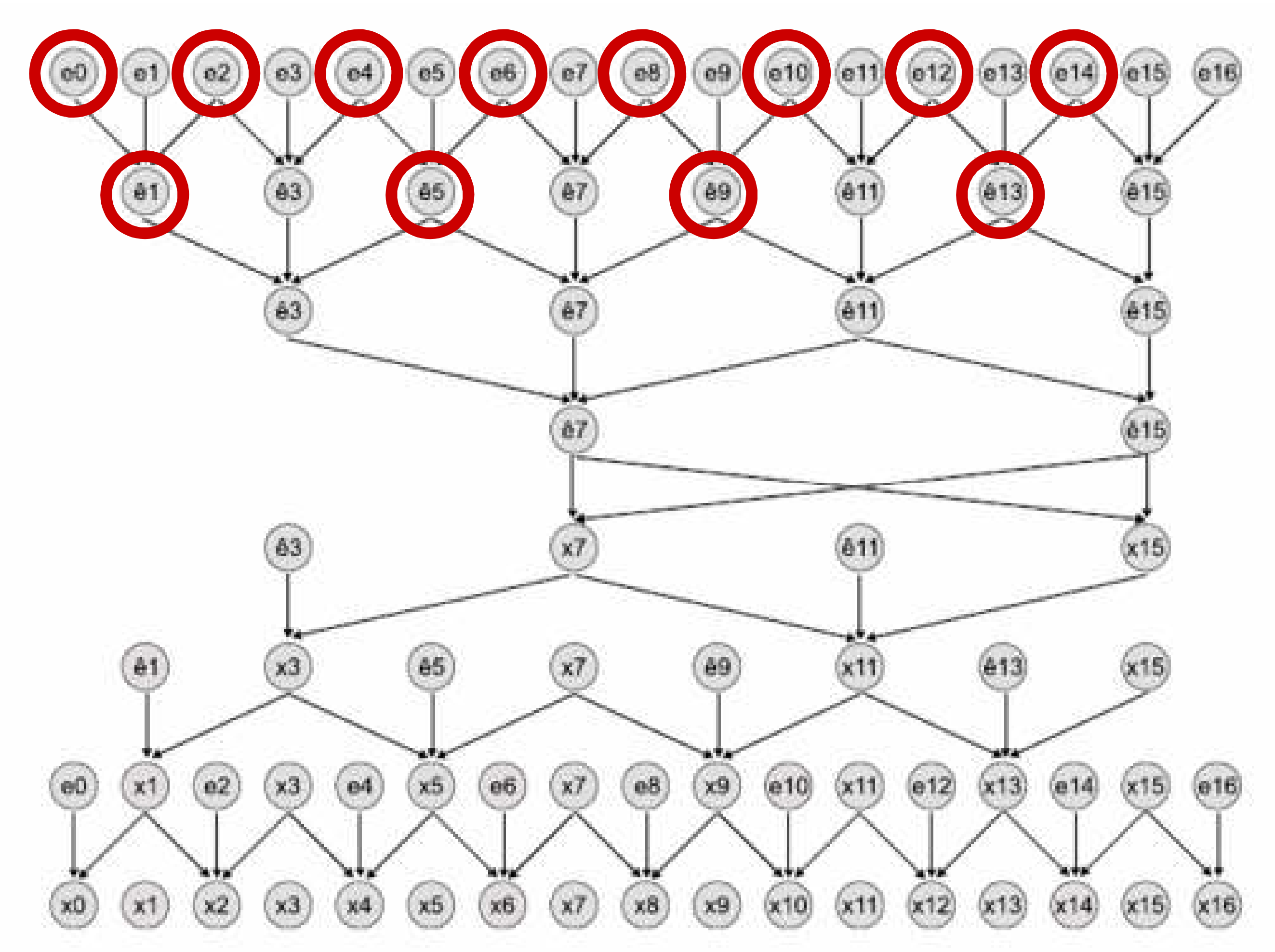
TRIDI-ROW $\mathbf{C} = (\mathbf{DL} + \mathbf{DD} + \mathbf{DU})$



TRIDI-COLUMN

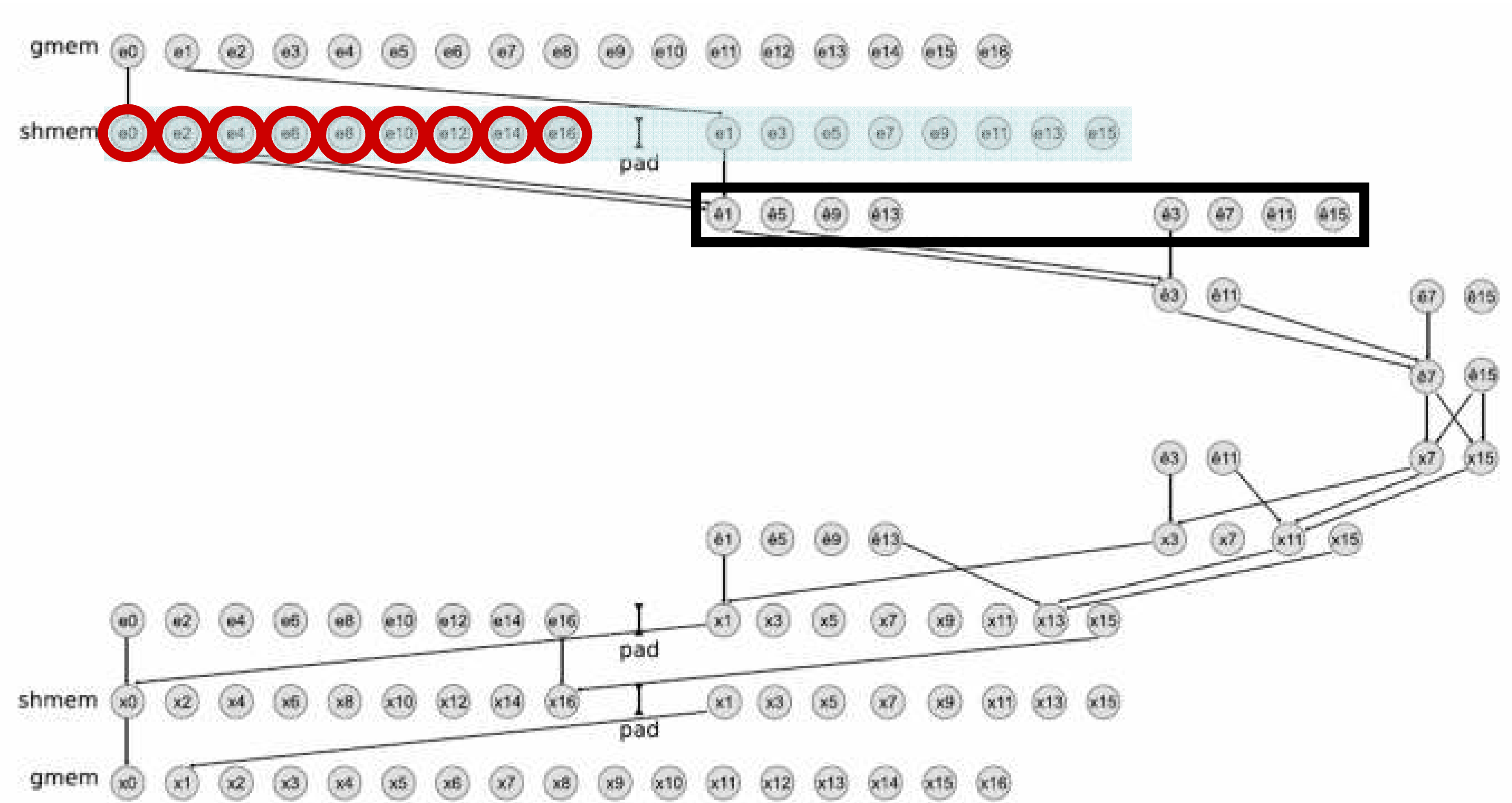


SIMD Parallelism: Cyclic Reduction



3*8	***2**
3*4-1	***4**
3*2-1	***8**
3*2-1	***8**
3*4-1	***4**
3*9-2	***2**
O(N)	**O(N*logN)**

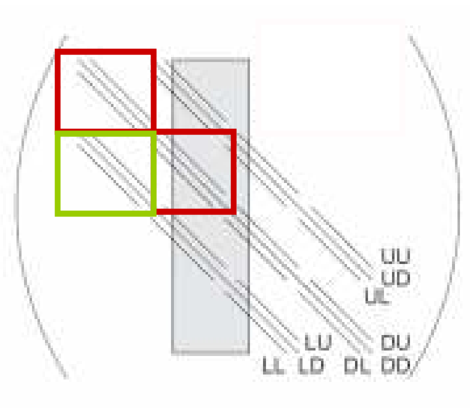
Memory Friendly Cyclic Reduction



[Göddeke et al. *Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed Precision Multigrid* , TPDS 2011]

ADI-TRIGS Preconditioner

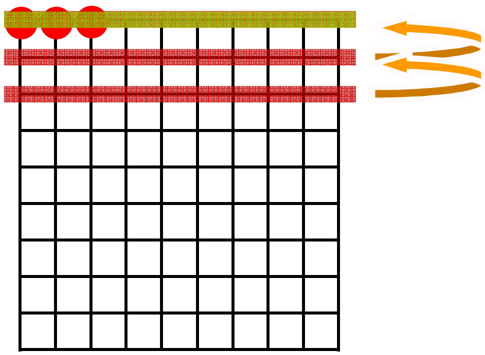
$$\mathbf{Ax} = \mathbf{b}$$



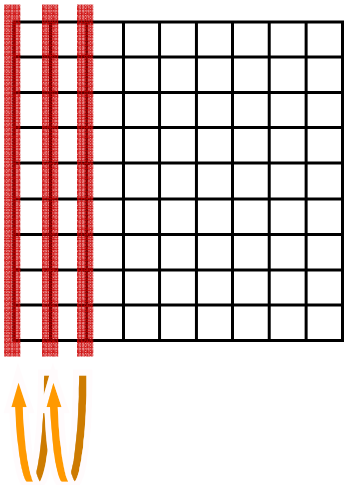
Damped, preconditioned
defect correction:

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + \omega \mathbf{C}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{k})$$

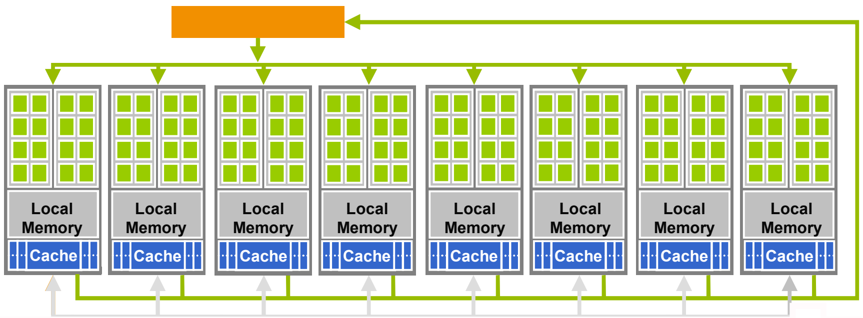
TRIGS-ROW $\mathbf{C} = (\mathbf{LL} + \mathbf{LD} + \mathbf{LU}) + (\mathbf{DL} + \mathbf{DD} + \mathbf{DU})$



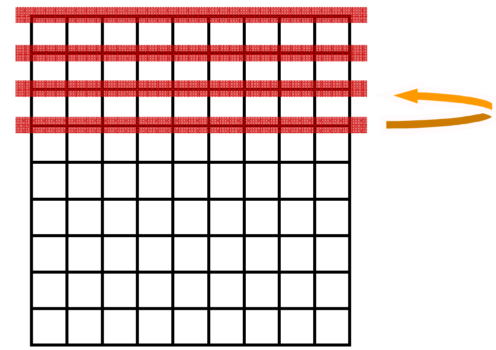
TRIGS-COLUMN



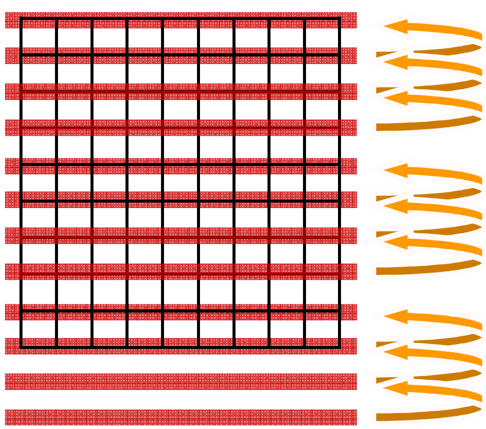
Many-Core Parallelism



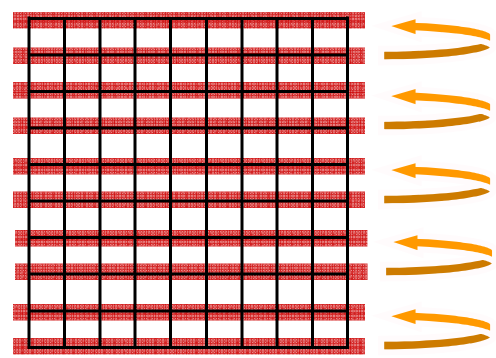
full/serial coupling

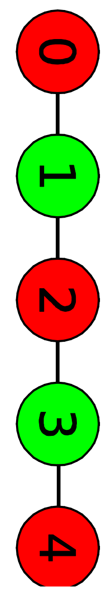


4-way coupling



2-way coupling

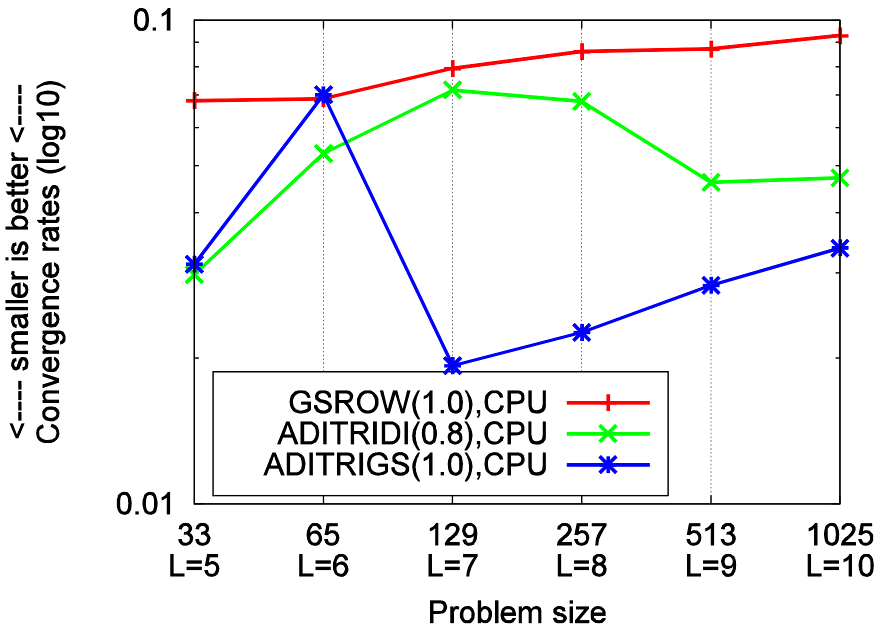


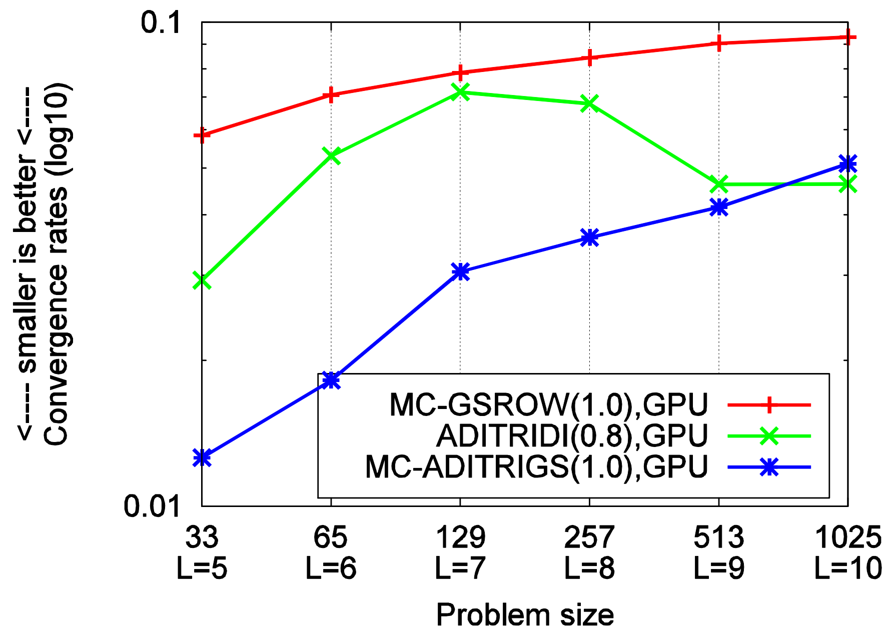


CPU vs. GPU Numerical Efficiency

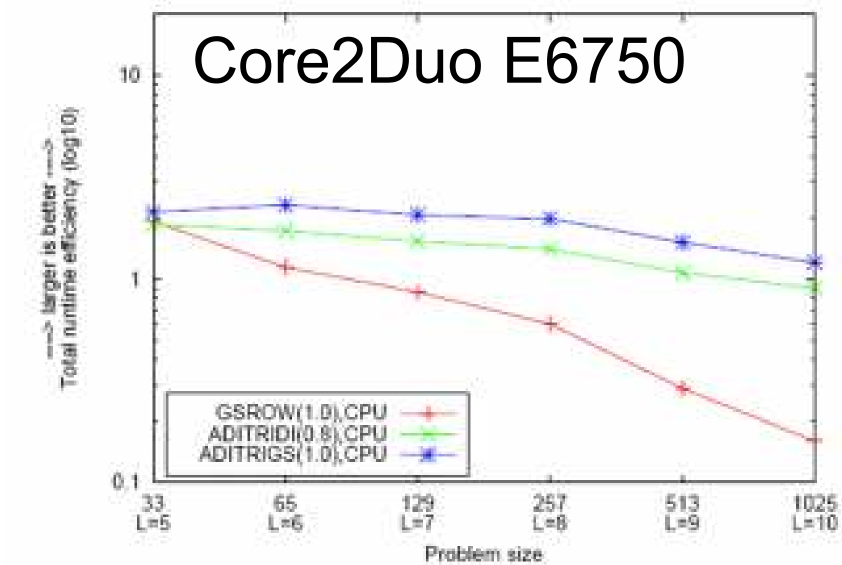
Iter.Ref.(double) MG(float) V(2,2) CG

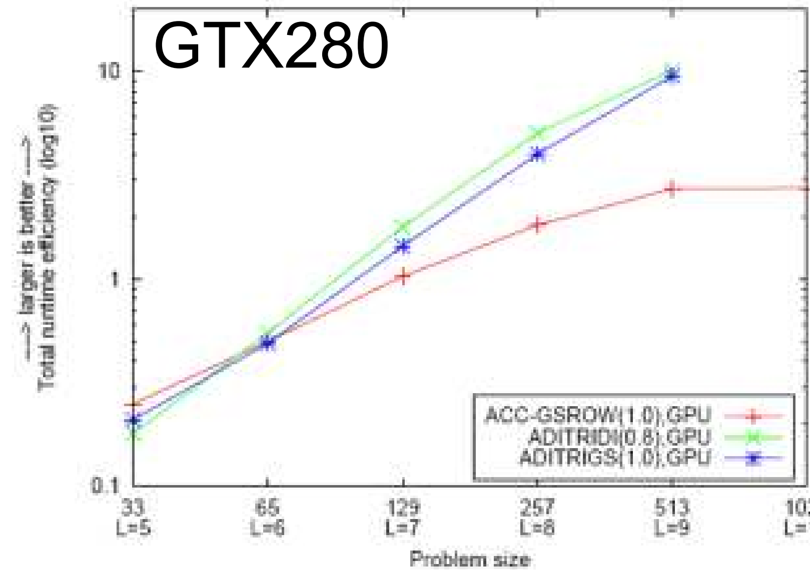
$$\rho := \left(\frac{\|\mathbf{A}\mathbf{x}^k - \mathbf{b}\|_2}{\|\mathbf{A}\mathbf{x}^0 - \mathbf{b}\|_2} \right)^{1/k}$$

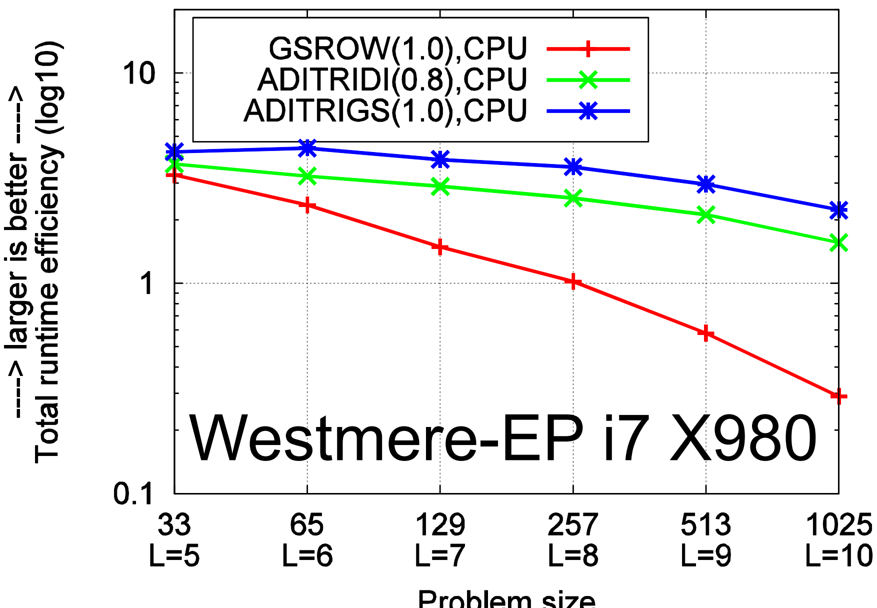


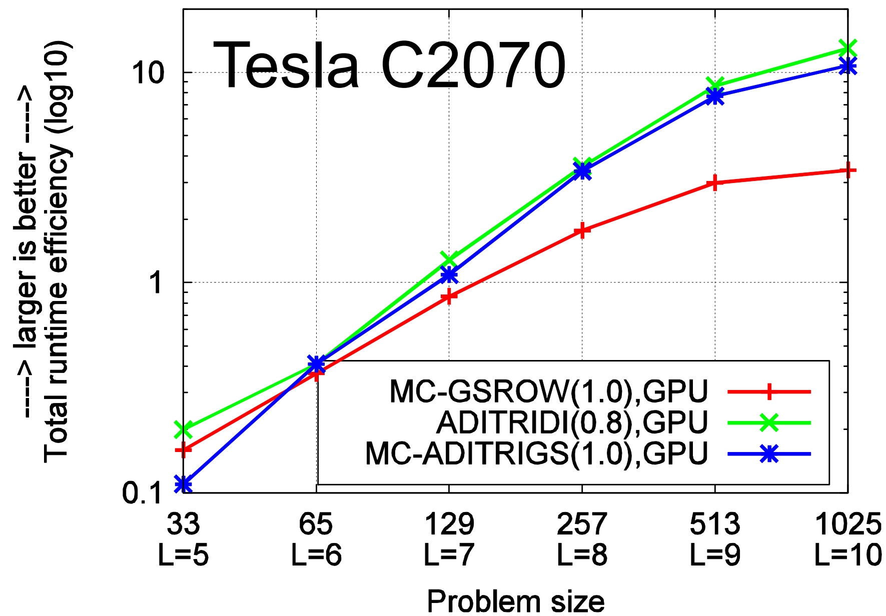


CPU vs. GPU Runtime Efficiency



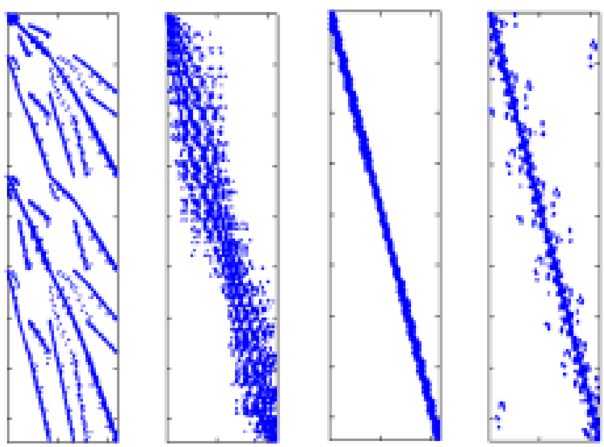






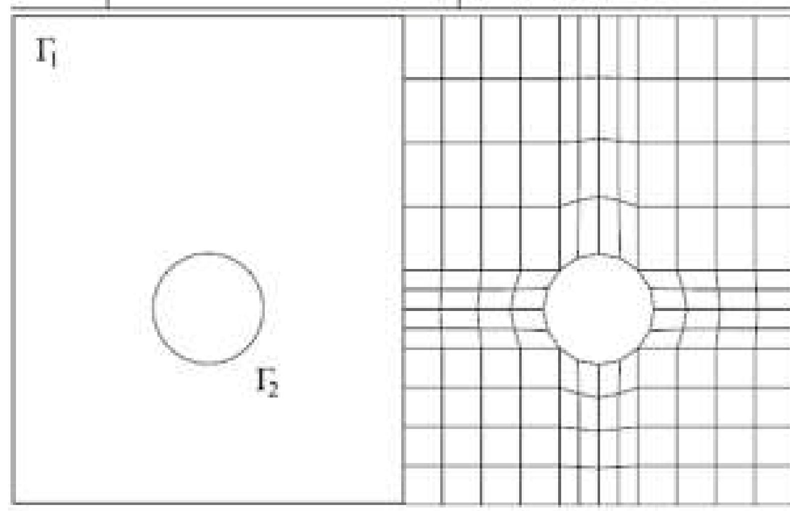
Multigrid on Refined Unstructured Grid

- **FE-gMG**
  - Unstructured grid
  - Regular refinement
  - Restriction & Prolongation as MatVec
  - SPAI preconditioner
- **Order & Storage (ELL)**

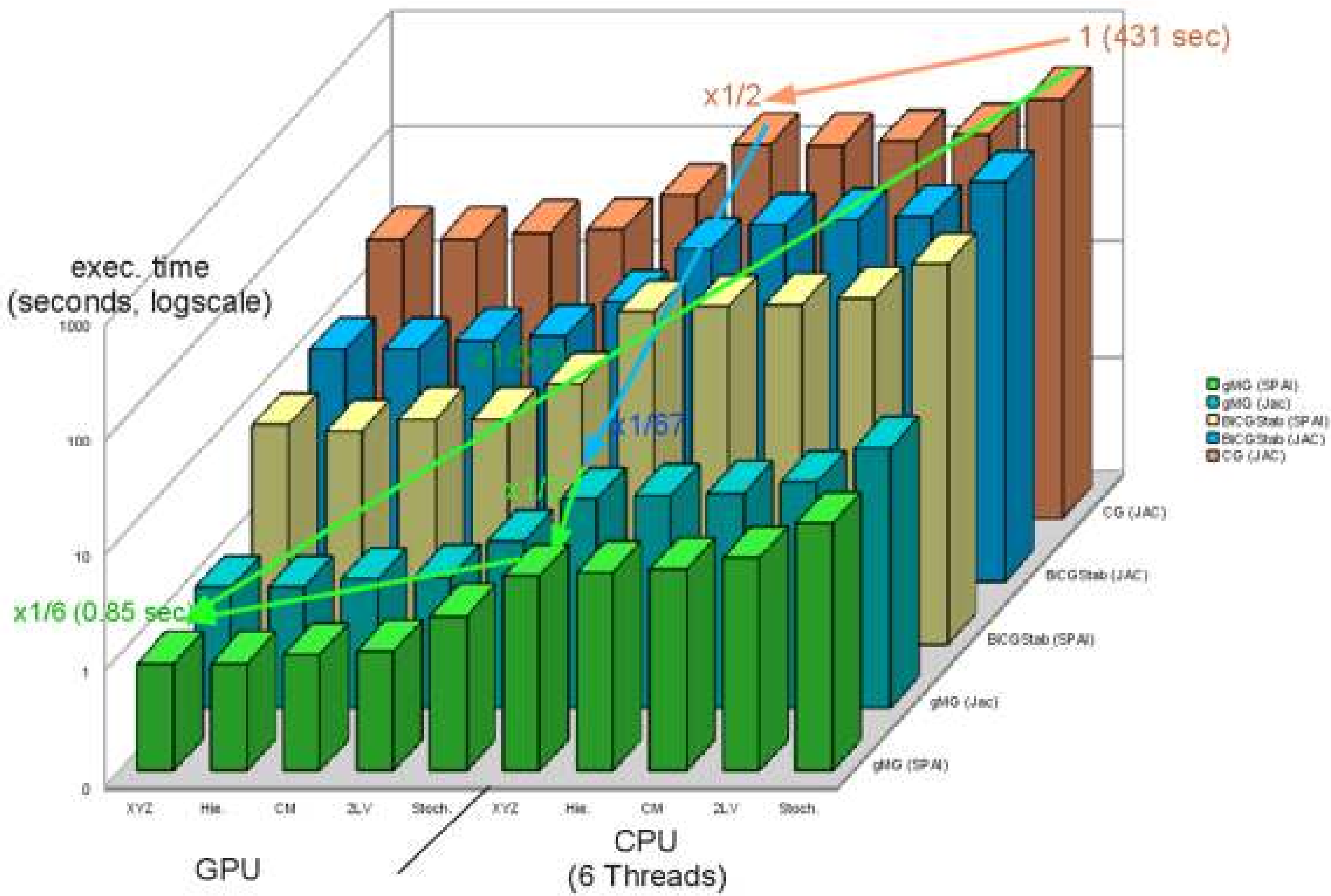


$$\begin{cases} -\Delta u = 1, & \mathbf{x} \in \Omega \\ u = 0, & \mathbf{x} \in \Gamma_1 \\ u = 1, & \mathbf{x} \in \Gamma_2 \end{cases}$$

L	Q_1 N	Q_1 non-zeros	Q_2 N	Q_2 non-zeros
4	576	4552	2176	32192
5	2176	18208	8448	128768
6	8448	72832	33280	515072
7	33280	291328	132096	2078720
8	132096	1172480	526336	8351744
9	526336	4704256	2101248	33480704
10	2101248	18845696	-	-



FE-gMG Results with SPAI Preconditioner



Overview

- Levels of Parallelism
- Grid Discretizations of PDEs
- Multigrid and Strong Smoothers
- Mixed Precision Iterative Refinement
- Layout of Multi-valued Data

Precision Comparison

	Numerical Algorithms	GPU Hardware
Precision	long 64bit double s52e11	int 32bit float s23e8

Comparison

Bandwidth	1 word	2 words
Storage	1 word	2 words
Operator+	1 adder	2 adders
Operator*	1 multiplier	4 multipliers

Hardware Precision

float s23e8	double s52e11
23 bit	52 bit

Data Error

$1/2 =_{fl} 0.5$

$1/3 =_{fl} 0.33333333$

$1/2 =_{db} 0.5$

$1/3 =_{db} 0.333333333333333$

Roundoff Error

$1.0002 * 0.9998 =_{fl} 1$

$1 + 4e\text{-}8 =_{fl} 1$

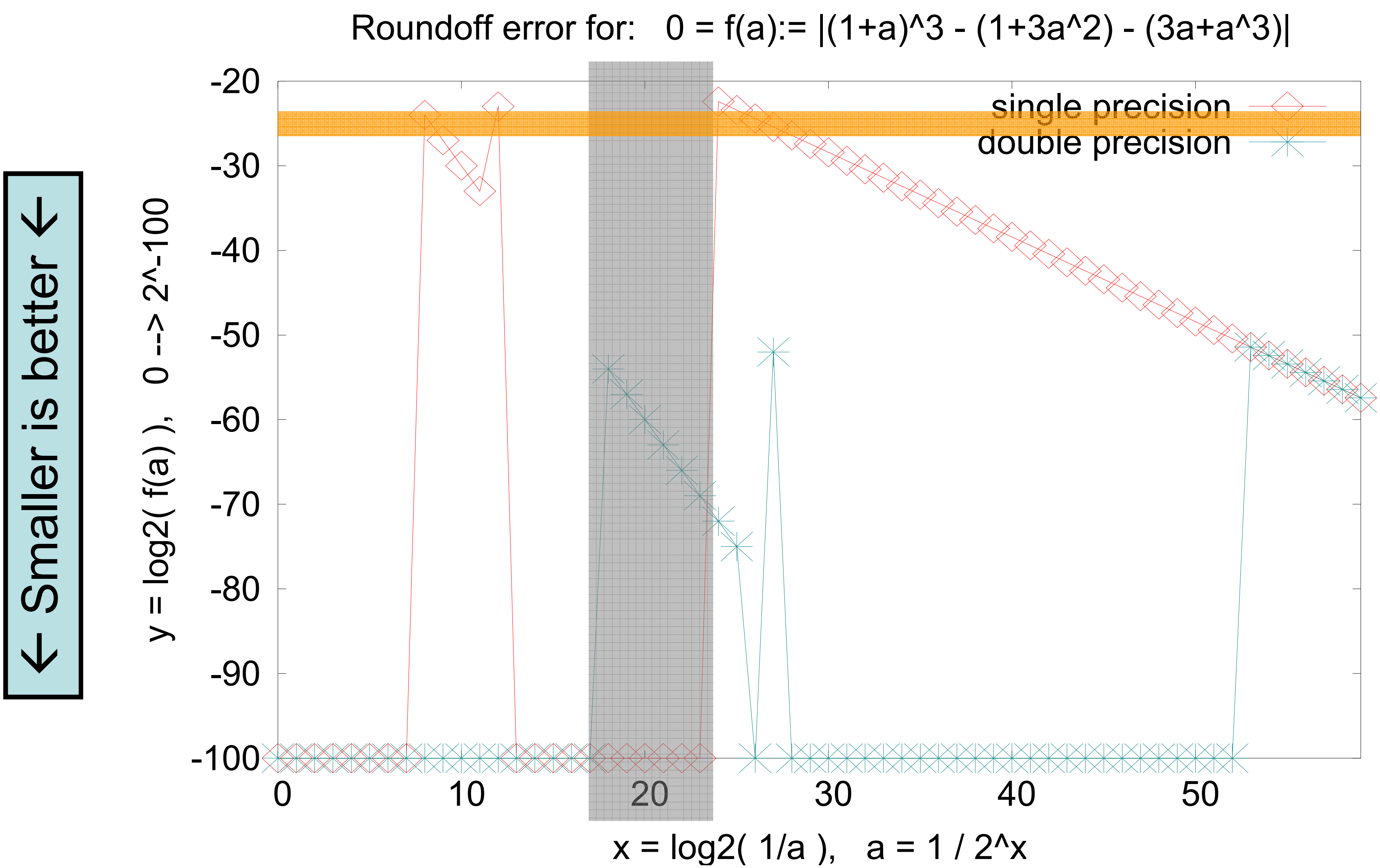
$f(a, b) =_{fl} f_{fl}(a, b)$

$1.0002 * 0.9998 =_{db} 0.99999996$

$1 + 4e\text{-}15 =_{db} 1$

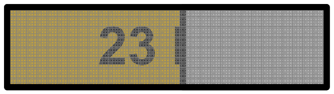
$f(a, b) =_{db} f_{db}(a, b)$

The Erratic Roundoff Error

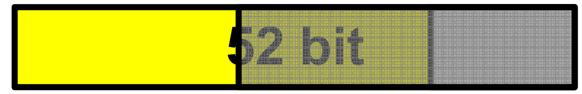


Numerical Accuracy

float s23e8



double s52e11



Condition of Ax = b

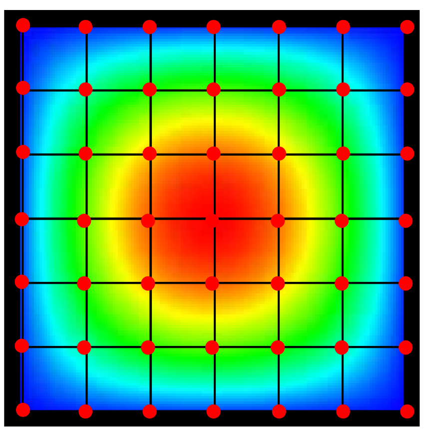
$$(\mathbf{A}-\mathbf{A}_{\varepsilon})\mathbf{x}^{\mathrm{fl}} = \mathbf{b}-\mathbf{b}_{\varepsilon}$$

$$\mathbf{x}-\mathbf{x}^{\mathrm{fl}} = c(\mathbf{A})\cdot\mathbf{x}_{\varepsilon}$$

$$(\mathbf{A}-\mathbf{A}_{\delta})\mathbf{x}^{\mathrm{db}} = \mathbf{b}-\mathbf{b}_{\delta}$$

$$\mathbf{x}-\mathbf{x}^{\mathrm{db}} = c(\mathbf{A})\cdot\mathbf{x}_{\delta}$$

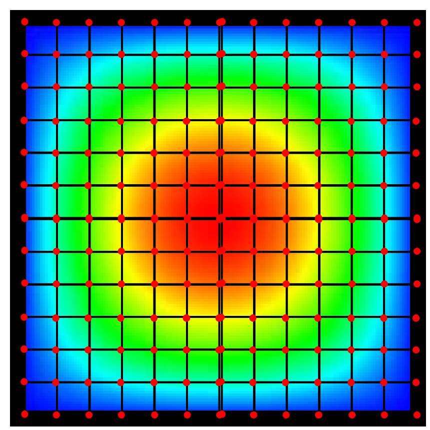
Discretization Error



$$-\mathrm{div}(\mathbf{G}\nabla\mathbf{u}) = \mathbf{f}$$

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

$$\mathrm{err} = \|\mathbf{u}-\mathbf{x}\|$$

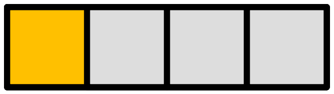


$$\mathrm{err} = \|\mathbf{u}-\mathbf{x}\| \quad \downarrow$$

$$c(\mathbf{A}) \quad \uparrow$$

Mixed Precision Iterative Refinement

float s23e8



double s52e11

Condition of Ax = b

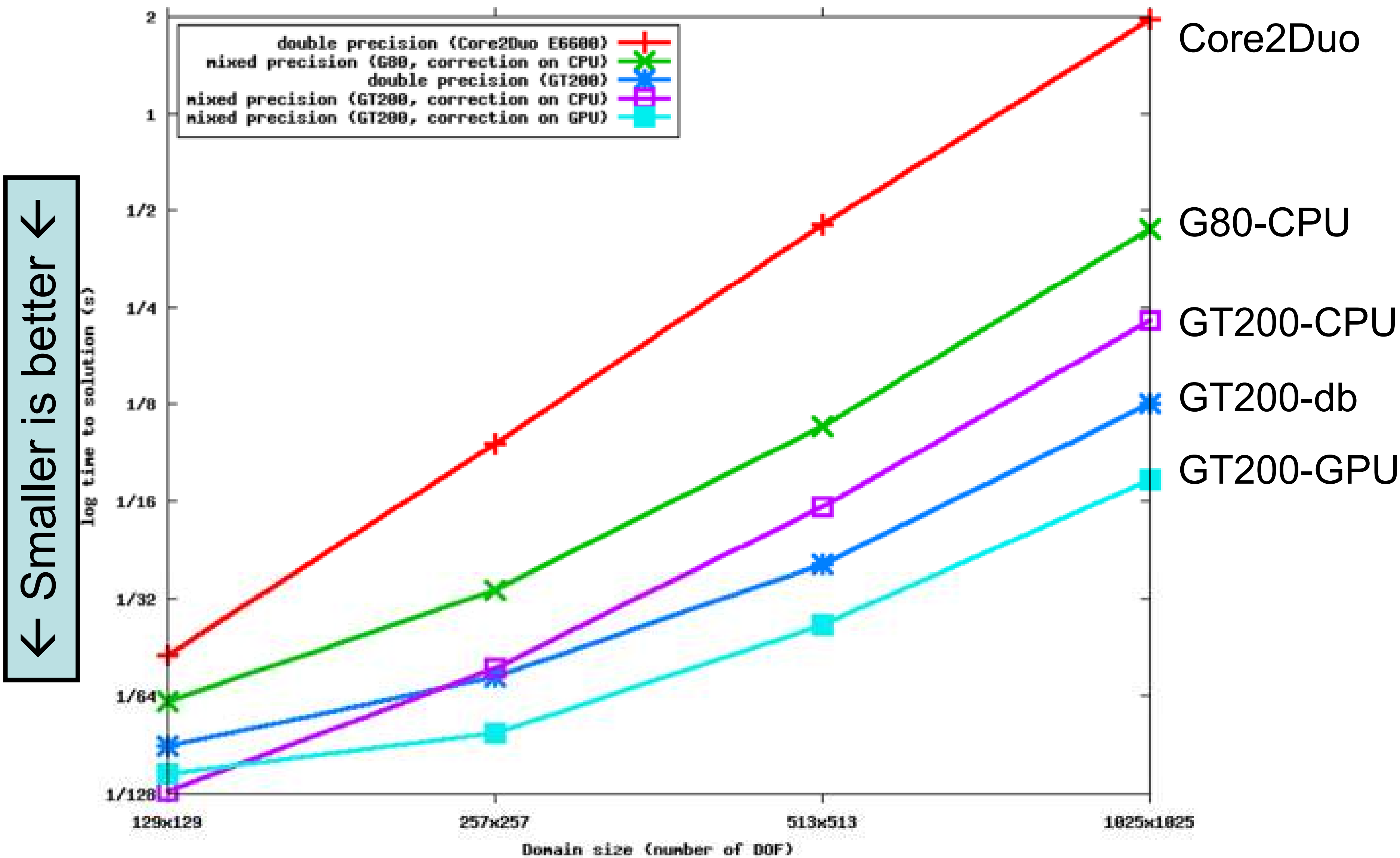
$$(\mathbf{A} - \mathbf{A}_{\varepsilon})\mathbf{x}^{\mathrm{fl}} = \mathbf{b} - \mathbf{b}_{\varepsilon} \qquad \mathbf{x}_{l+1}^{\mathrm{fl}} = F(\mathbf{A}, \mathbf{b}, \mathbf{x}_{l}^{\mathrm{fl}})$$

$$\mathbf{x} - \mathbf{x}^{\mathrm{fl}} = c(\mathbf{A}) \cdot \mathbf{x}_{\varepsilon} \qquad \mathbf{x}_{l+1} - \mathbf{x}_{l+1}^{\mathrm{fl}} = c(F) \cdot \mathbf{x}_{\varepsilon}$$

- **Iterative Refinement for Ax = b**

$d_k = b - Ax_k$	**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

Mixed Precision Multigrid on GPU



Overview

- Levels of Parallelism
- Grid Discretizations of PDEs
- Multigrid and Strong Smoothers
- Mixed Precision Iterative Refinement
- Layout of Multi-valued Data

Multi-Valued Data

- **Multi-valued data is ubiquitous**

  – Class 1: mathematical properties, e.g. multiple derivatives or moments

  – Class 2: discrete features, e.g. colors of a pixel, multiple scores

  – Class 3: per-dimension properties, e.g. coordinates, velocities on a 3D grid

AoS and SoA

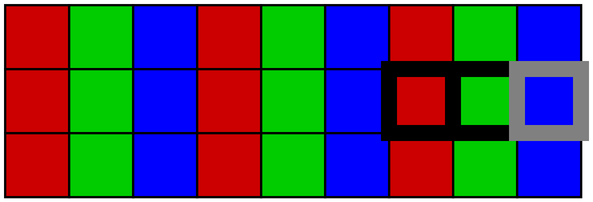
Array of Structs (AoS)

```
struct NormalStruct {
   Type1 comp1;
   Type2 comp2;
   Type3 comp3;
};

typedef NormalStruct
   AoSContainer[SIZE];

AoSContainer container;

container[5].comp3++;
```

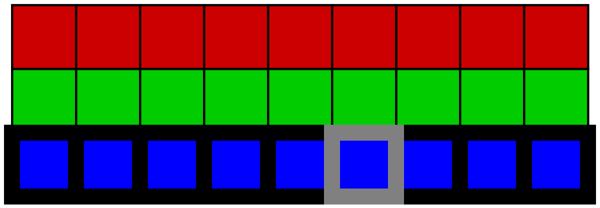


Struct of Arrays (SoA)

```
struct SoAContainer {
   Type1 comp1[SIZE];
   Type2 comp2[SIZE];
   Type3 comp3[SIZE];
};

SoAContainer container;

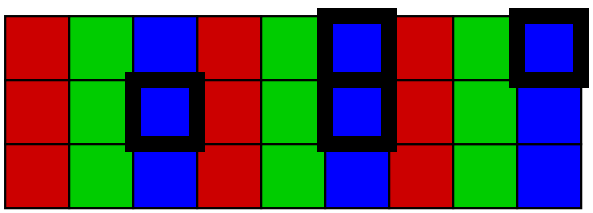
container.comp3[5]++;
```



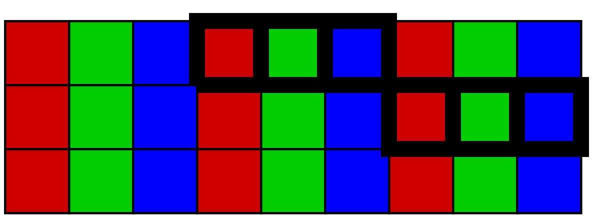
Parallel Access in AoS and SoA

Array of Structs (AoS)

```
container[1].comp3
container[2].comp3
container[3].comp3
container[4].comp3
```

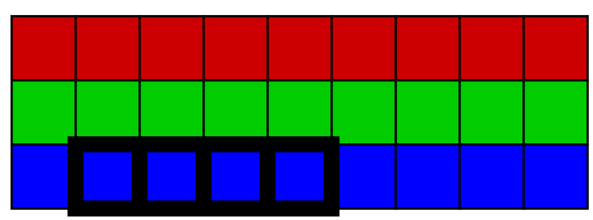


```
container[1].comp{1,2,3}
container[5].comp{1,2,3}
```

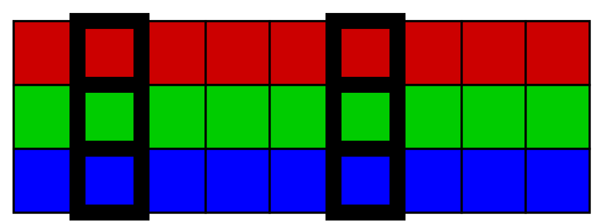


Struct of Arrays (SoA)

```
container.comp3[1]
container.comp3[2]
container.comp3[3]
container.comp3[4]
```



```
container.comp{1,2,3}[1]
container.comp{1,2,3}[5]
```



Operating on Container Elements

- ```
  struct NormalStruct {
    Type1 comp1;
    Type2 comp2;
    Type3 comp3;
  };
  typedef NormalStruct  AoSContainer[SIZE];
  ```

- ```
  NormalStruct single;
  AoSContainer container;
  ```

- **In-place update of single and indexed structs**
  ```
  void update( NormalStruct& s ) { s.comp3 += s.comp1; }
  ```

- ```
  update( single );                  // OK
  update( container[5] );            // OK
  ```

- **This is not possible with standard SoA/C++ syntax:**
  ```
  container.comp3(5);
  ```
  ```

Abstraction: AoS + SoA = ASA

Array of Structs (AoS)

```
struct NormalStruct {
   Type1 comp1;
   Type2 comp2;
   Type3 comp3;
};

typedef NormalStruct
   Container[SIZE];

NormalStruct single
   Container container;

void
   update(NormalStruct& s);

container[index].comp3++;
update( container[5] );
```

Array of Structs of Arrays (ASA)

```
template <ID t_id=ID_value>
struct FlexibleStruct {
   typedef ASAGroup<Type1,t_id> ASX_ASA;
   union{ Type1 comp1; ASX_ASA d1; };
   union{ Type2 comp2; ASX_ASA d2; };
   union{ Type3 comp3; ASX_ASA d3; };
};
typedef ASX::Array<FlexibleStruct,
   SIZE, ??? > Container;

FlexibleStruct<> single
   Container container;

template <ID t_id> void
   update(FlexibleStruct<t_id>& s);

container[index].comp3++;
update( container[5] );

??? = ASX::AOS or ASX::SOA
```

[Strzodka. *Abstraction for AoS and SoA layout in C++* . GCG 2011]

Overview

- **Levels of Parallelism**
  – Explicit exploitation of all levels: SIMD, core, socket, cluster
- **Grid Discretizations of PDEs**
  – Regularity of memory access
- **Multigrid and Strong Smoothers**
  – Balancing numerical and hardware requirements
- **Mixed Precision Iterative Refinement**
  – Same accuracy with faster computation
- **Layout of Multi-valued Data**
  – Choice of layout determines memory access patterns

Questions?

Robert Strzodka
Integrative Scientific Computing
Max Planck Institut Informatik
www.mpi-inf.mpg.de/~strzodka

Dominik Göddeke
Institute for Applied Mathematics
Technical University of Dortmund
www.mathematik.tu-dortmund.de/~goeddeke