

# Accurate High Performance Multigrid Solvers on GPUs

**Robert Strzodka**

**Integrative Scientific Computing**

**Max Planck Institut Informatik**

**[www.mpi-inf.mpg.de/](http://www.mpi-inf.mpg.de/)**

**~strzodka**

**Dominik Göddeke**

**Institute for Applied Mathematics**

**Technical University of Dortmund**

**[www.mathematik.tu-dortmund.de/](http://www.mathematik.tu-dortmund.de/)**

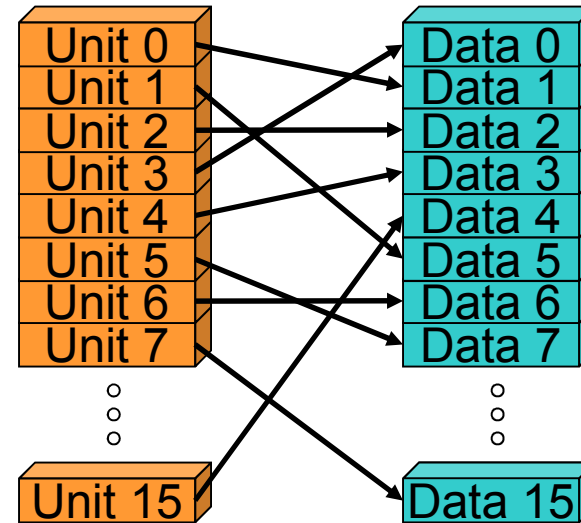
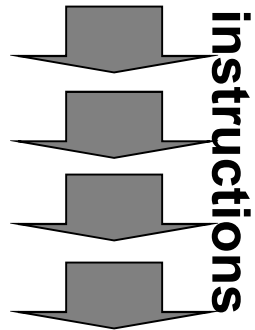
**~goeddeke**

# Parallelism

---

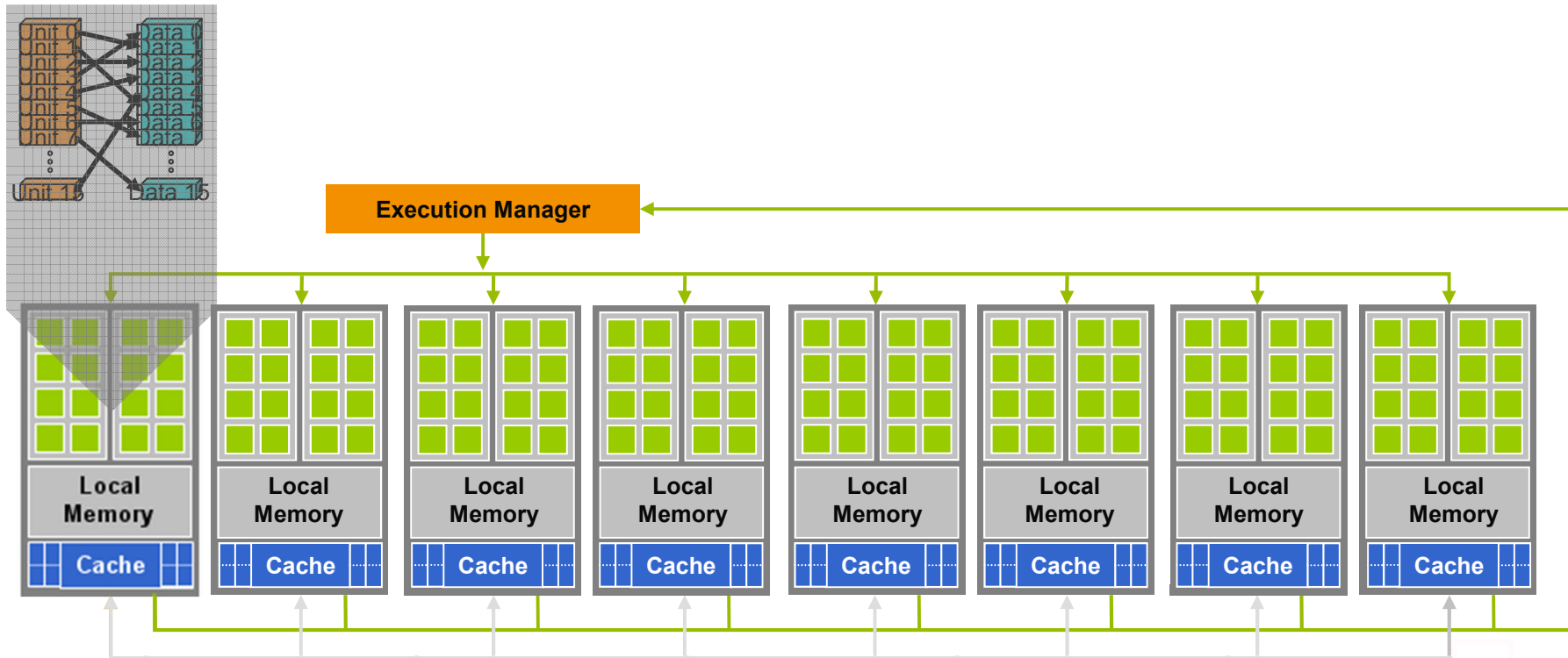
- Sequential execution is an **illusionary software concept**
- **All transistors always do something in parallel !**
- **Billions of transistors in modern CPUs(>0.5) & GPUs(>2)**
- **Old: Implicit parallelism** with caches, ILP, speculation  
→ diminishing returns, power constraints
- **New: Explicit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

# SIMD Parallelism



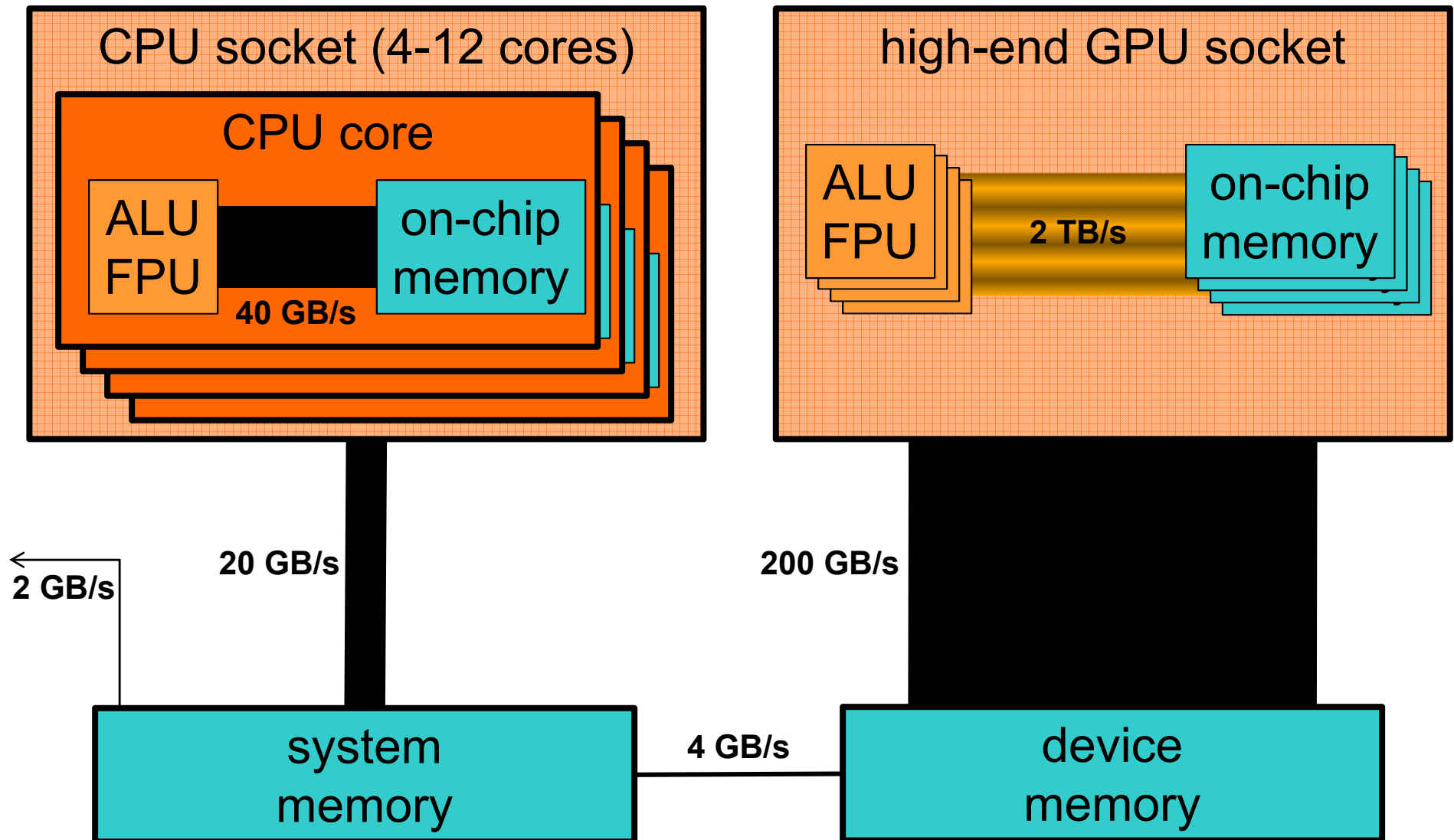
- **It is impossible to execute just one instruction**
  - $a = b + c$ ;
  - Actually means execute (**add, nop, nop, nop, ...**)
- **Penalty for ignoring SIMD**
  - 4x on current CPUs (SSE)
  - 8-16x on future CPUs (AVX, MIC)
  - 16-80x on GPUs

# Many-Core Parallelism



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

# Bandwidth in a CPU-GPU System



# Overview

---

- **Multigrid and Strong Smoothers**
- **Mixed Precision Iterative Refinement**
- **Layout of Multi-valued Data**

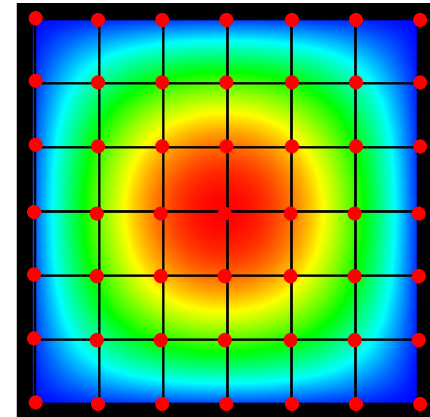
# Generalized Poisson Problem

---

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

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

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



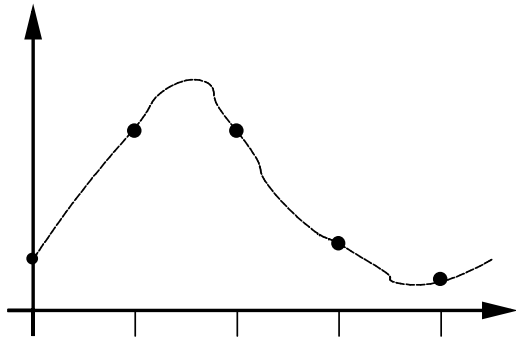
We consider the scalar ( $m=1$ ) 2D case ( $d=2$ ) with operator anisotropies. Given a vector field  $(v_1(x,y), v_2(x,y))$  we define:

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

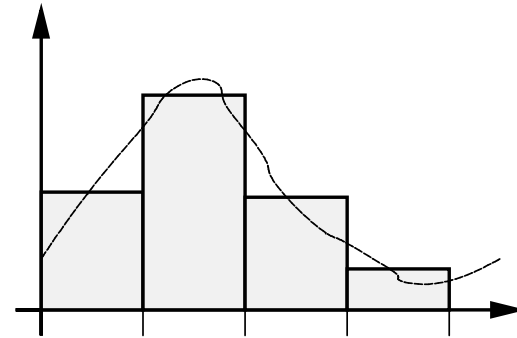
$$\mathbf{R}(x, y) := \frac{1}{\|\mathbf{v}(x, y)\|_2} \begin{pmatrix} v_1(x, y) & v_2(x, y) \\ -v_2(x, y) & v_1(x, y) \end{pmatrix}, \quad \mathbf{S}(x, y) := \begin{pmatrix} \|\mathbf{v}(x, y)\|_2 & 0 \\ 0 & 1 \end{pmatrix}$$

# Discretization Approach

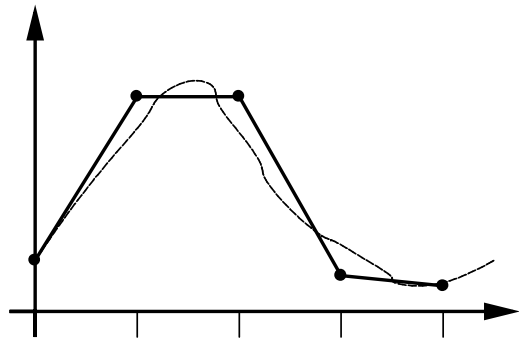
---



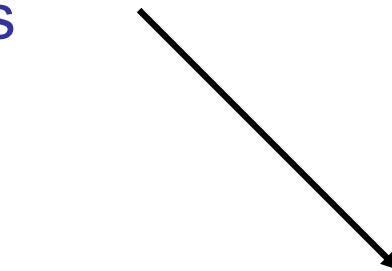
● Finite Differences



● Finite Volumes



● Finite Elements



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

For 2D linear FEM  $\mathbf{A}$  is a 9-band matrix.

# Geometric Multigrid Method

---

Linear equation system after discretization

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

**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

$$\mathbf{d}^k = \mathbf{b} - \mathbf{Ax}^k$$

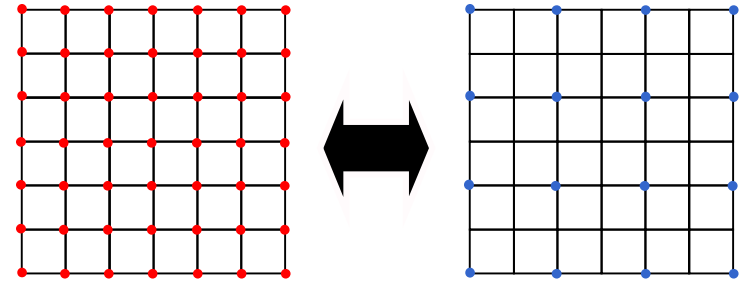
$$\mathbf{Ac}^k = \mathbf{d}^k$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{c}^k$$

**Fine grid**

**Coarse grid**

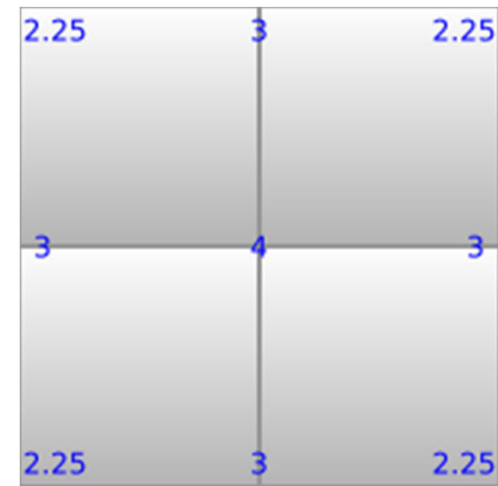
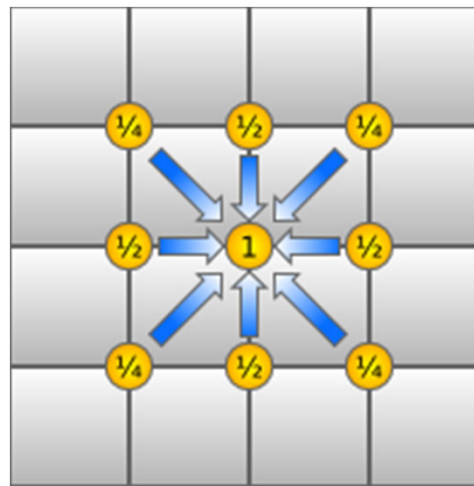
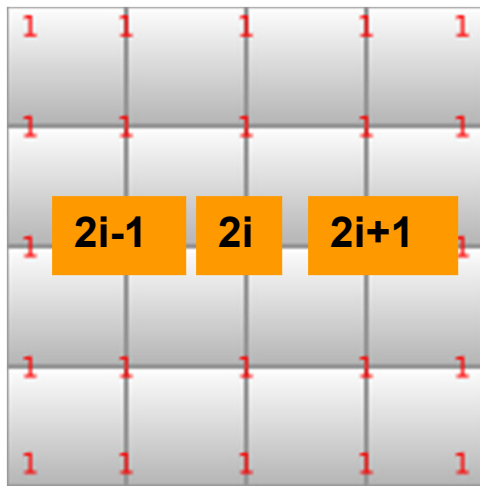
Back on **fine** grid



# Multigrid Transfers

- **Restriction**

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



**fine**

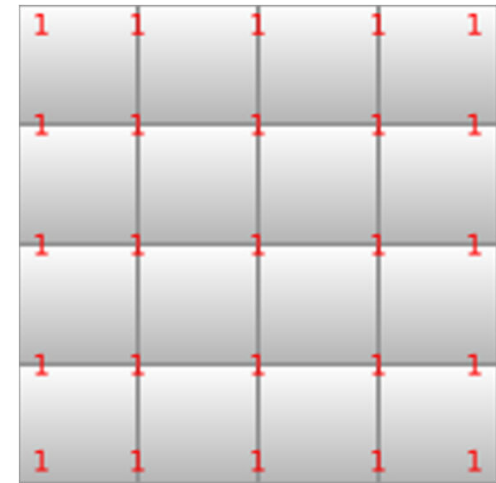
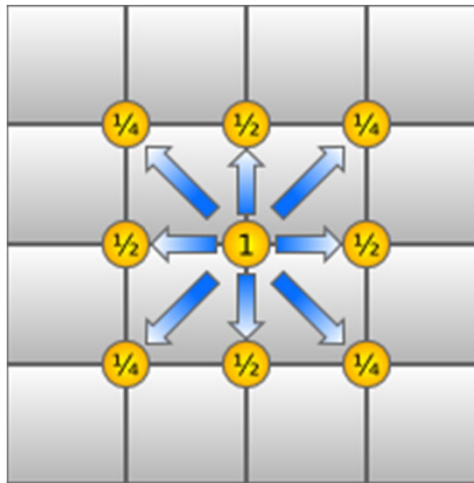
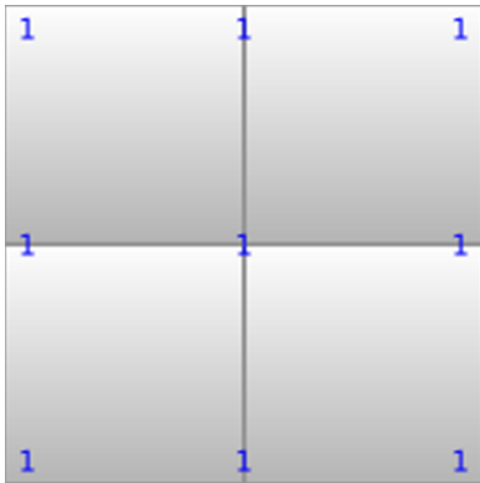
**adjust index  
to read  
neighbors**

**output region  
coarse result**

# Multigrid Transfers

- **Prolongation**

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

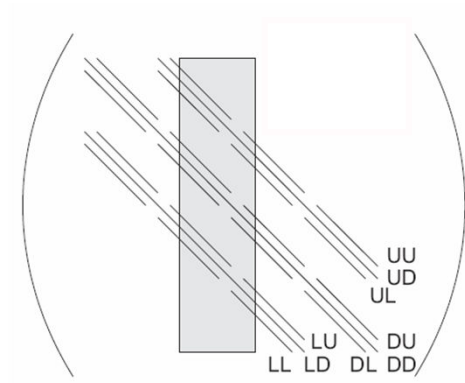
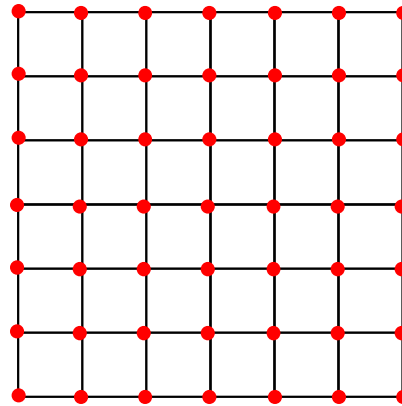


# Preconditioners

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

Damped, preconditioned defect correction:

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



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

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

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

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

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

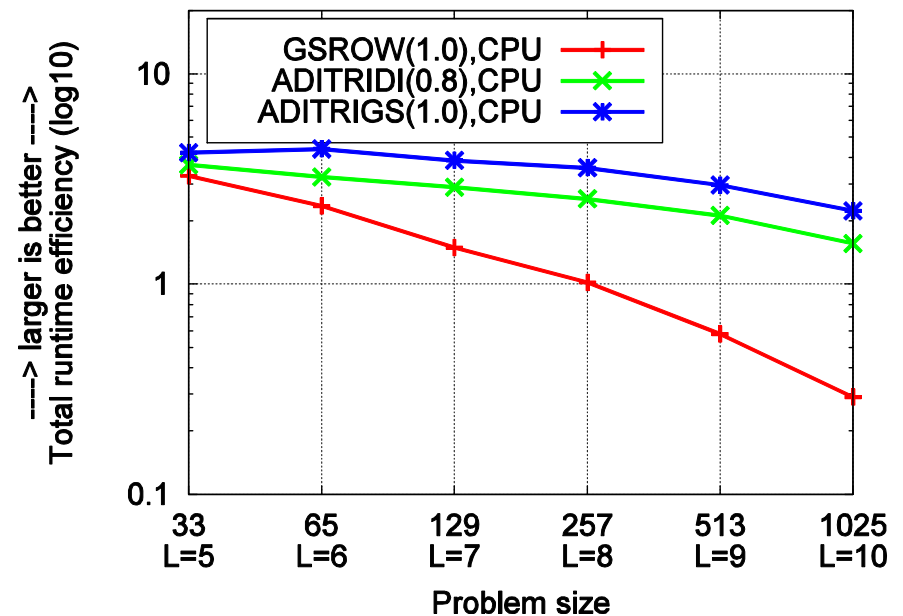
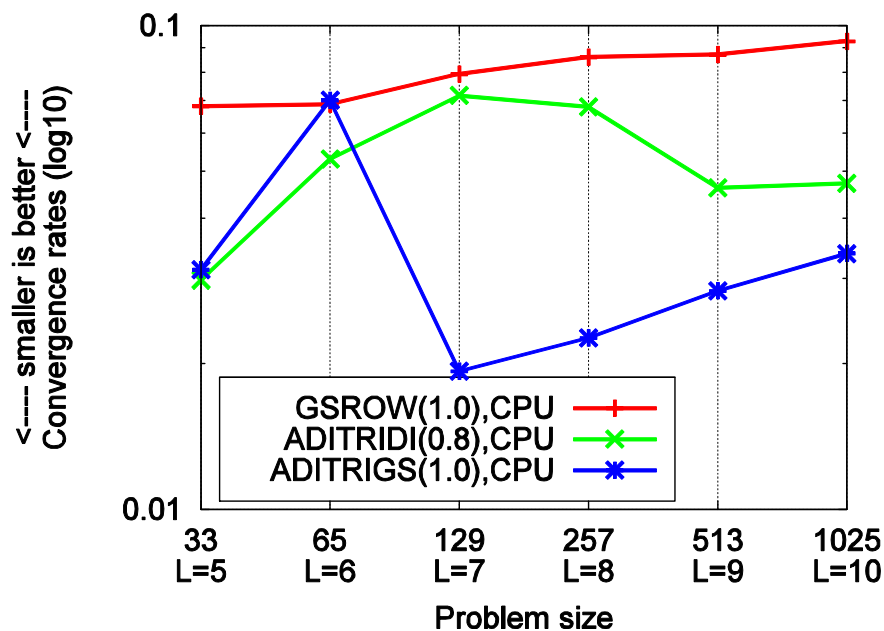
# CPU Numerical and Runtime Efficiency

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

$\alpha = 0,34??????$

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

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

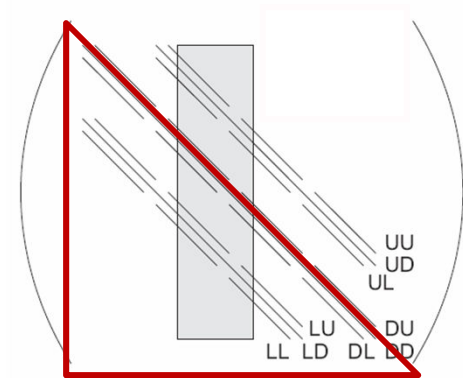


# Gauss-Seidel Preconditioner

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

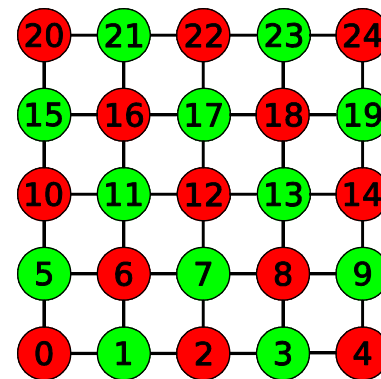
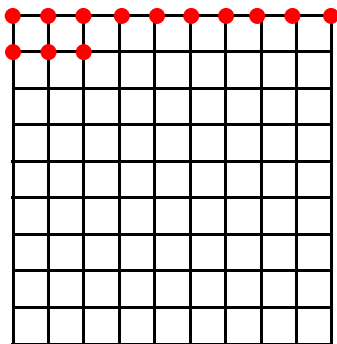
Damped, preconditioned  
defect correction:

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

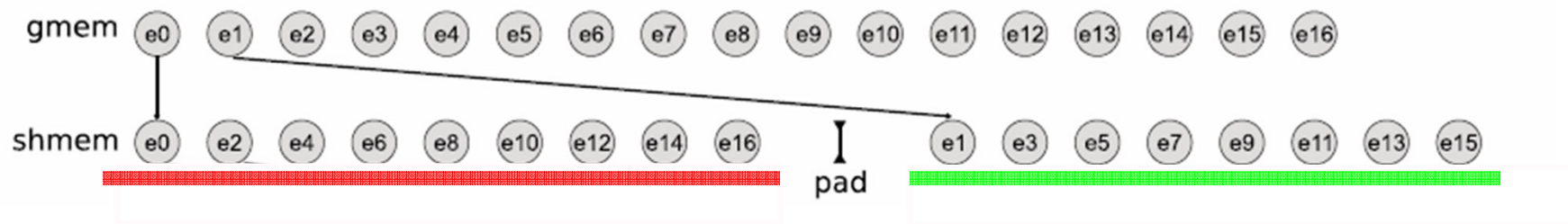
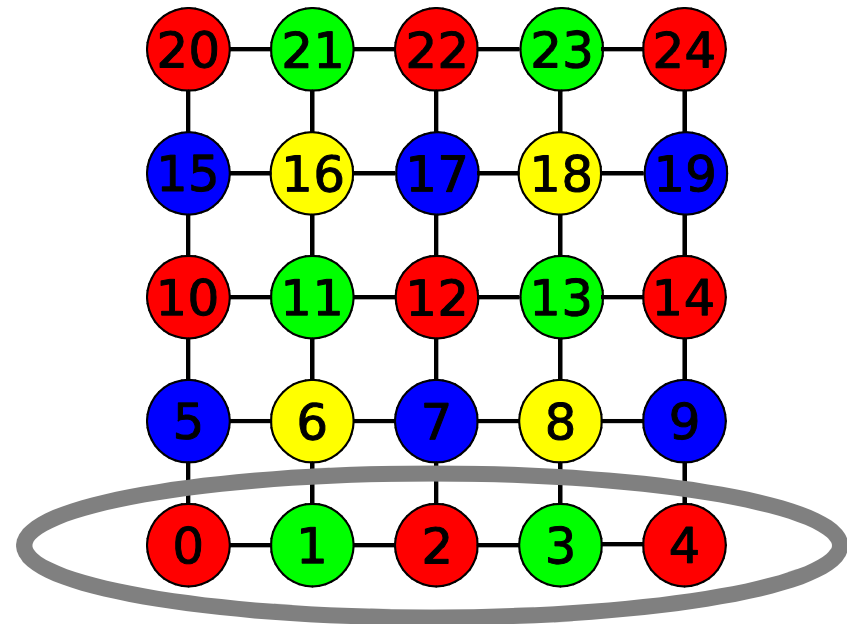
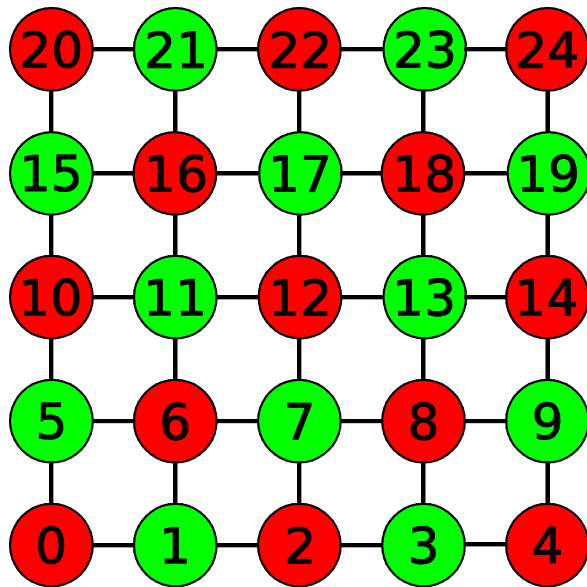


GSROW

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



# Multi-Colored Gauss-Seidel

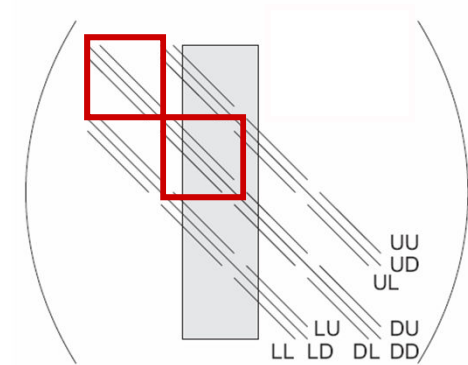


# ADI-TRIDI Preconditioner

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

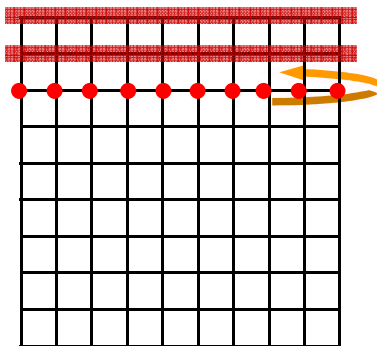
Damped, preconditioned  
defect correction:

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

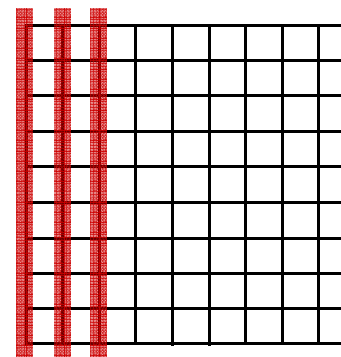


TRIDI-ROW

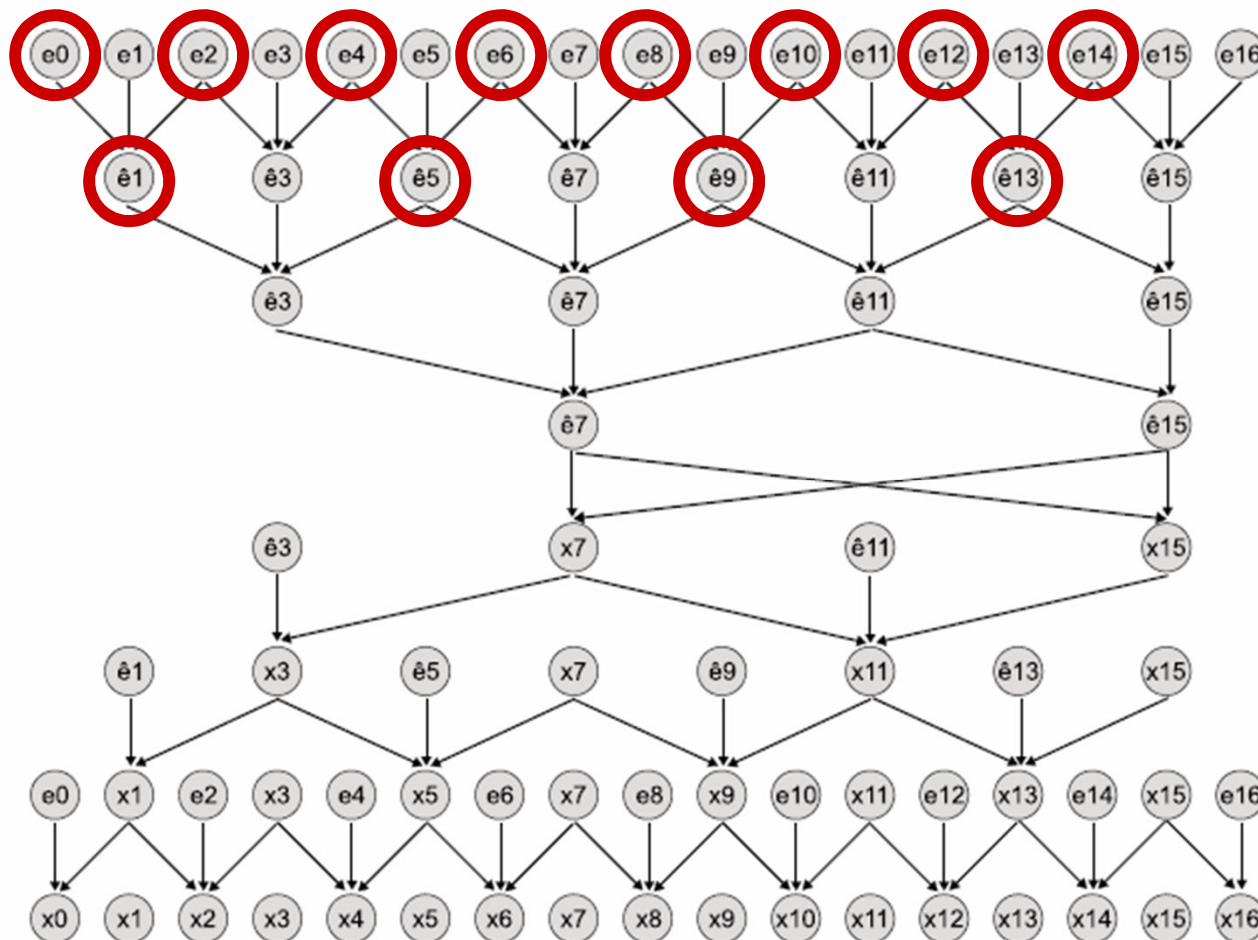
$$\mathbf{C} = (\mathbf{DL} + \mathbf{DD} + \mathbf{DU})$$



TRIDI-COLUMN

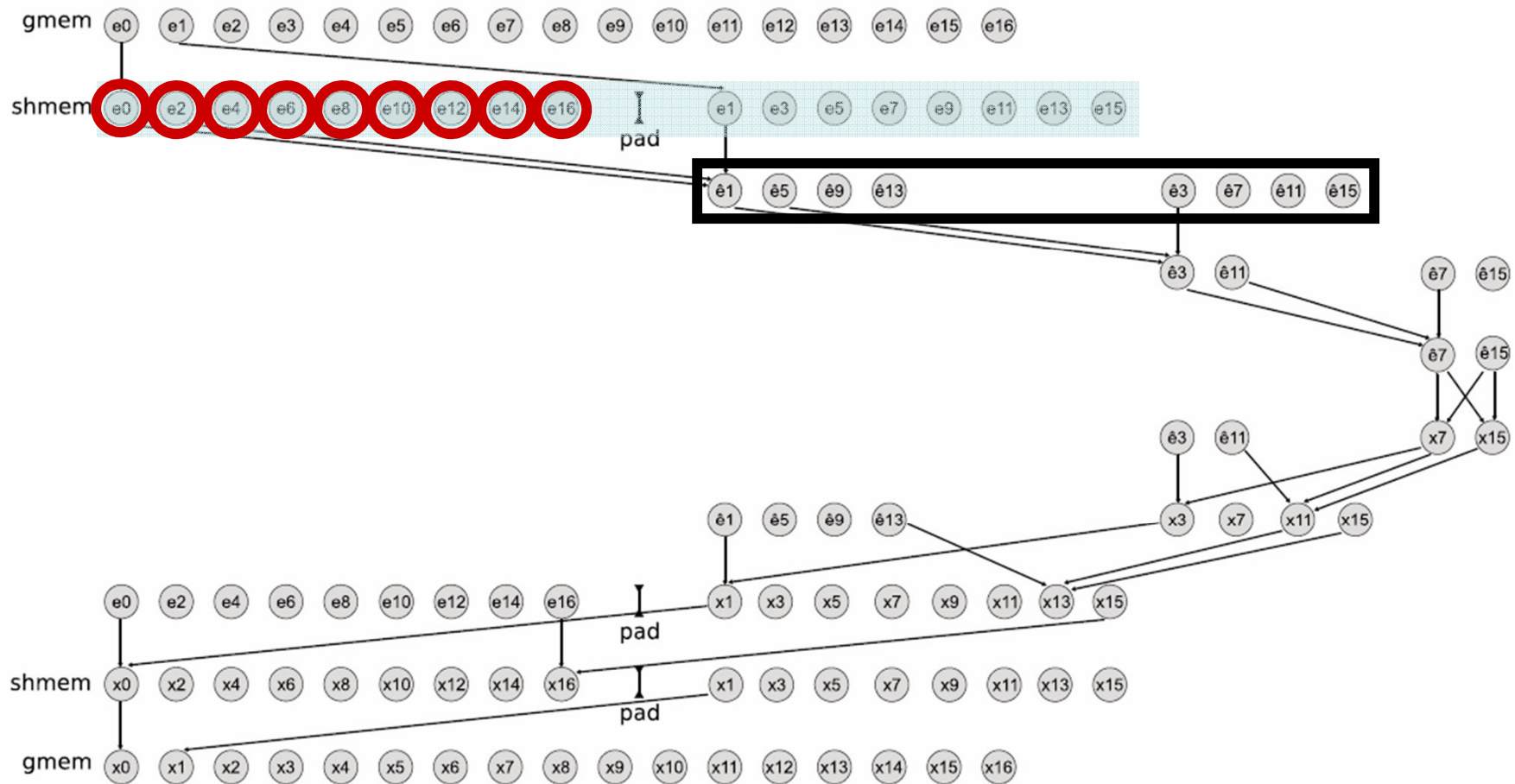


# SIMD Parallelism: Cyclic Reduction



$3 \cdot 8$        $\cdot 2$   
 $3 \cdot 4 - 1$      $\cdot 4$   
 $3 \cdot 2 - 1$        $\cdot 8$   
  
 $3 \cdot 2 - 1$        $\cdot 8$   
 $3 \cdot 4 - 1$        $\cdot 4$   
 $3 \cdot 9 - 2$         $\cdot 2$   
  
 $O(N)$      $O(N \cdot \log N)$

# Memory Friendly Cyclic Reduction



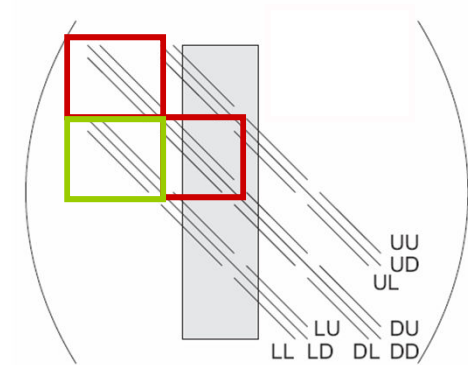
[Göddeke et al. *Cyclic Reduction Tridiagonal Solvers on GPUs Applied to Mixed Precision Multigrid*, TPDS 2011]

# ADI-TRIGS Preconditioner

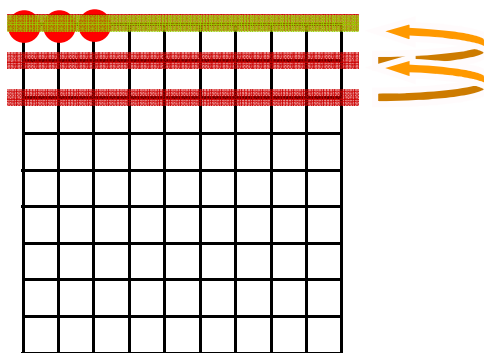
$$\mathbf{Ax} = \mathbf{b}$$

Damped, preconditioned defect correction:

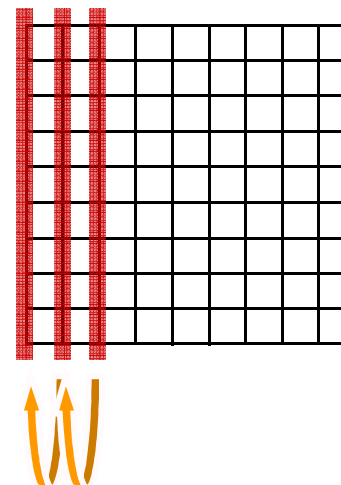
$$\mathbf{x}^{k+1} = \mathbf{x}^k + \omega \mathbf{C}^{-1} (\mathbf{b} - \mathbf{Ax}^k)$$



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

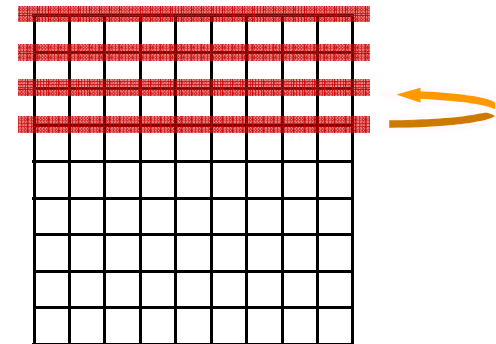


TRIGS-COLUMN

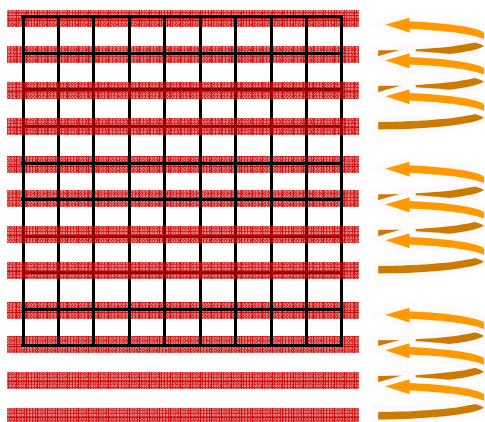


# Many-Core Parallelism

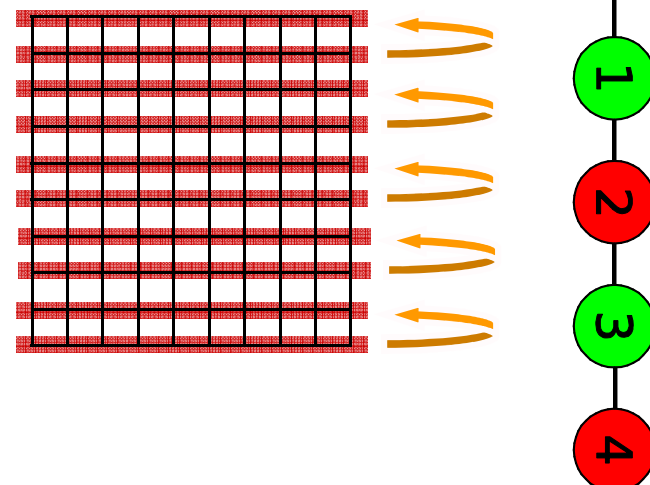
full/serial coupling



4-way coupling



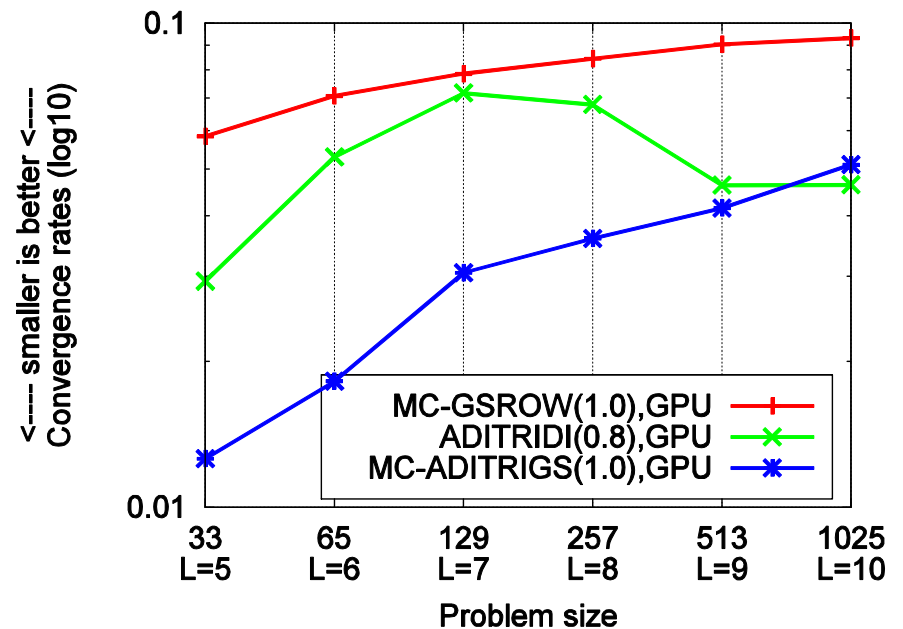
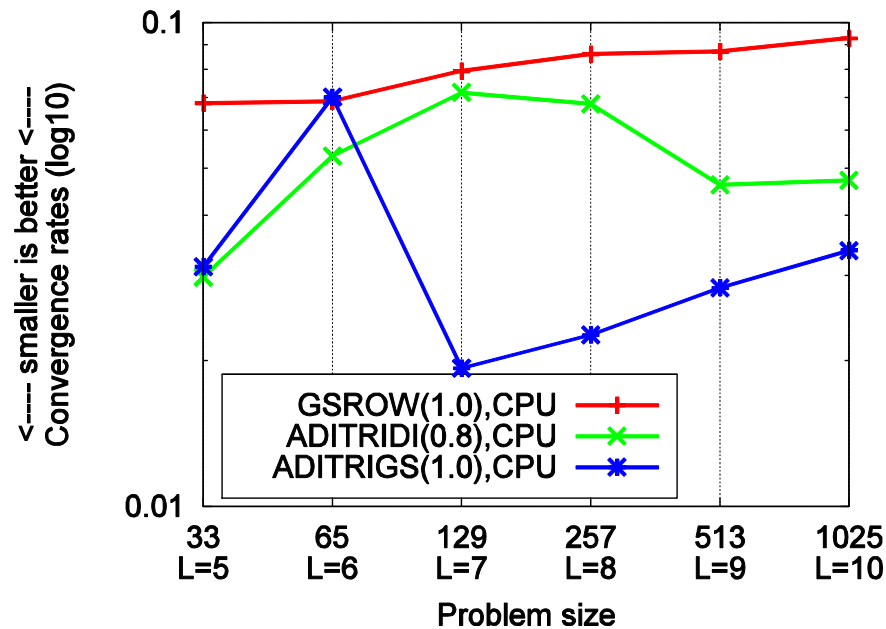
2-way coupling



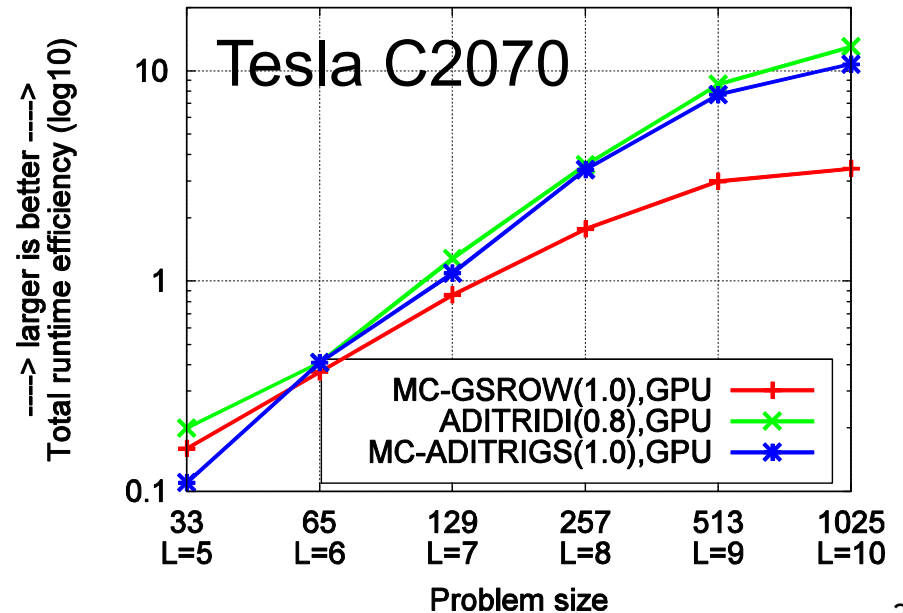
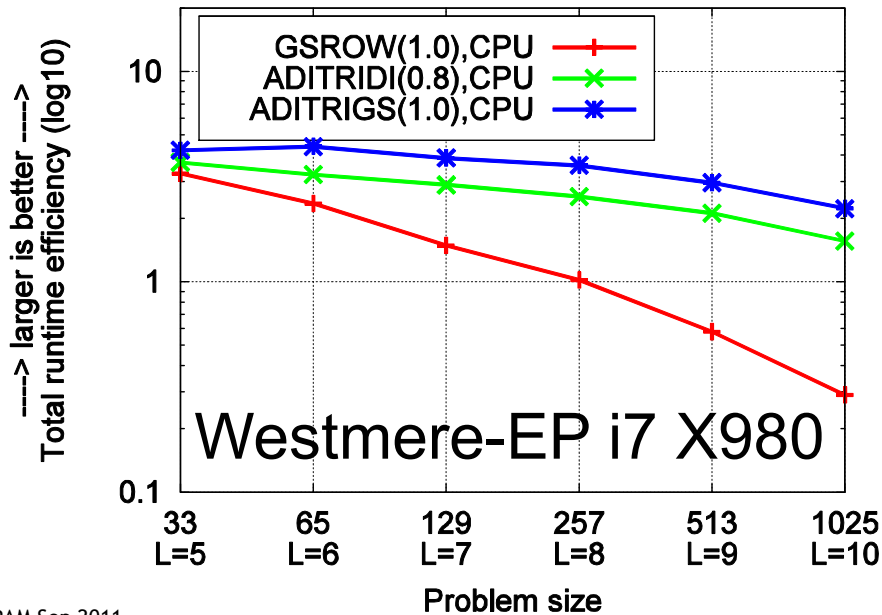
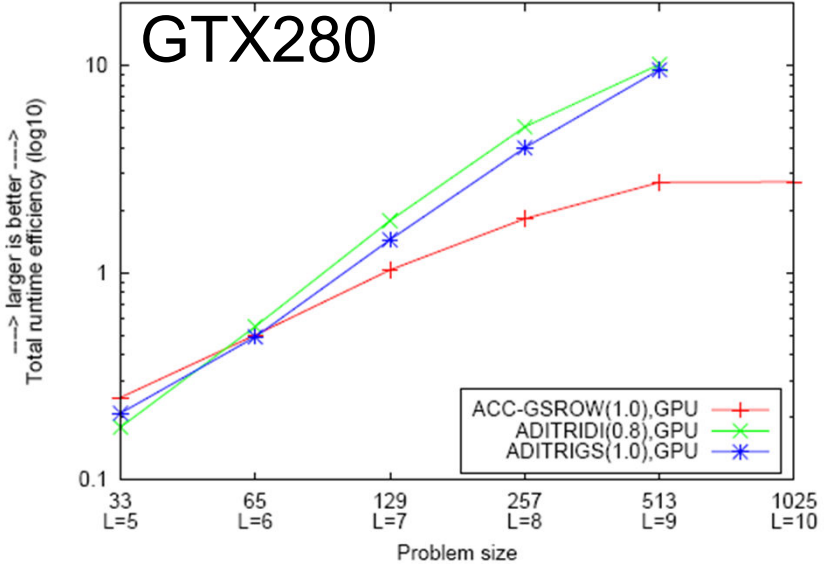
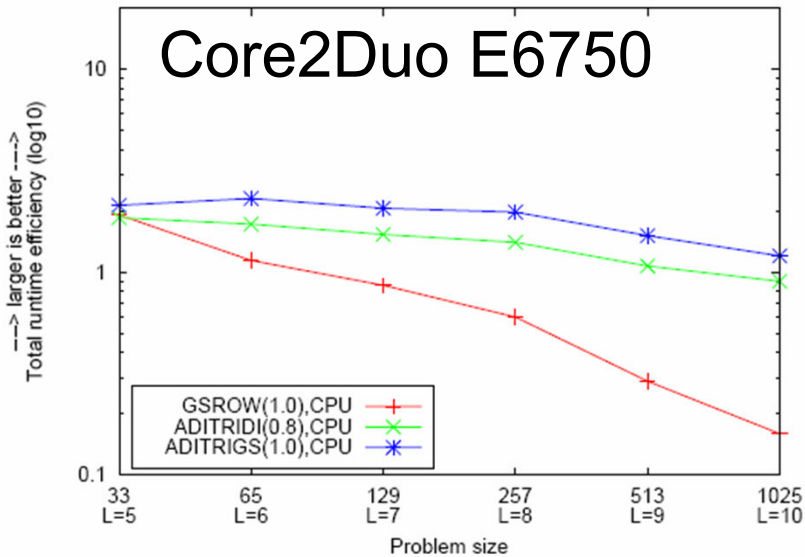
# CPU vs. GPU Numerical Efficiency

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

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



# CPU vs. GPU Runtime Efficiency



# Overview

---

- **Multigrid and Strong Smoothers**
- **Mixed Precision Iterative Refinement**
- **Layout of Multi-valued Data**

# Hardware Precision

---

float s23e8

23 bit

double s52e11

52 bit

## Data Error

$$1/2 =_{\text{fl}} 0.5$$

$$1/3 =_{\text{fl}} 0.33333333$$

$$1/2 =_{\text{db}} 0.5$$

$$1/3 =_{\text{db}} 0.3333333333333333$$

## Roundoff Error

$$1.0002 * 0.9998 =_{\text{fl}} 1$$

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

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

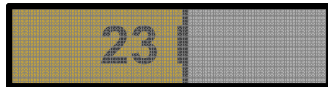
$$1.0002 * 0.9998 =_{\text{db}} 0.999999996$$

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

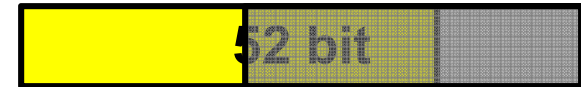
$$f(a, b) =_{\text{db}} f_{\text{db}}(a, b)$$

# Numerical Accuracy

float s23e8



double s52e11



## Condition of $Ax = b$

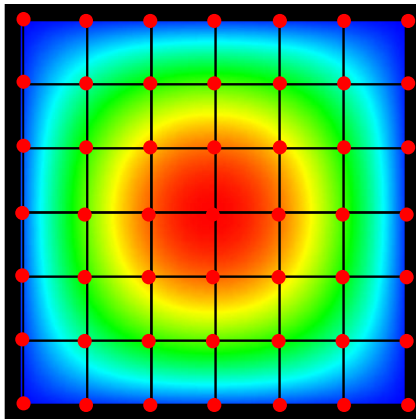
$$(A - A_\varepsilon)x^{\text{fl}} = b - b_\varepsilon$$

$$x - x^{\text{fl}} = c(A) \cdot x_\varepsilon$$

$$(A - A_\delta)x^{\text{db}} = b - b_\delta$$

$$x - x^{\text{db}} = c(A) \cdot x_\delta$$

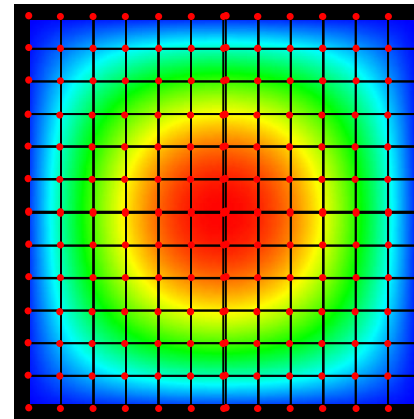
## Discretization Error



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

$$Ax = b$$

$$\text{err} = \|u - x\|$$



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

$$c(A) \quad \uparrow$$

# Mixed Precision Iterative Refinement

---

float s23e8



double s52e11



## Condition of $Ax = b$

$$(A - A_\varepsilon)x^{\text{fl}} = b - b_\varepsilon$$

$$x_{l+1}^{\text{fl}} = F(A, b, x_l^{\text{fl}})$$

$$x - x^{\text{fl}} = c(A) \cdot x_\varepsilon$$

$$x_{l+1} - x_{l+1}^{\text{fl}} = c(F) \cdot x_\varepsilon$$

- **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

# Overview

---

- **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

---

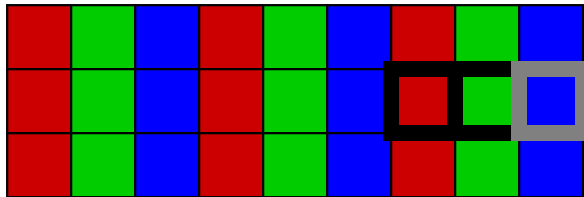
## Array of Structs (AoS)

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

```
typedef NormalStruct  
    AoSContainer[SIZE];
```

```
AoSContainer container;
```

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

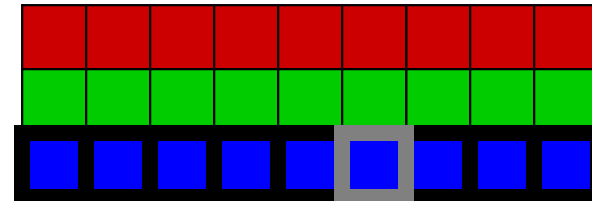


## Struct of Arrays (SoA)

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

```
SoAContainer container;
```

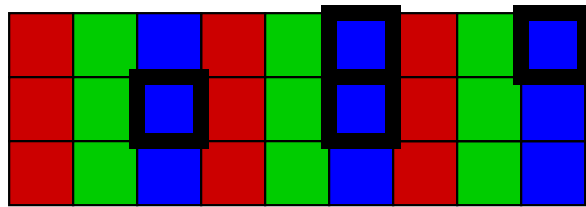
```
container.comp3[5]++;
```



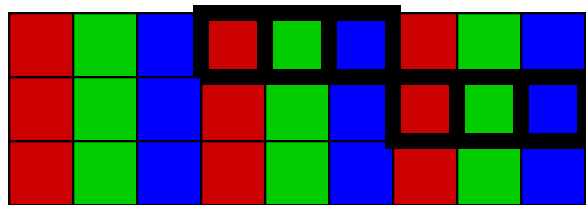
# Parallel Access in AoS and SoA

## Array of Structs (AoS)

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

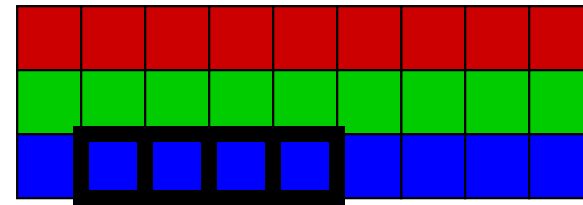


```
container[1].comp{1,2,3}  
container[5].comp{1,2,3}
```

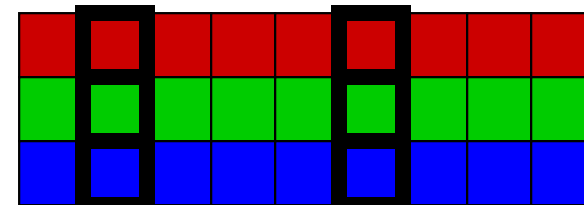


## Struct of Arrays (SoA)

```
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] );
```

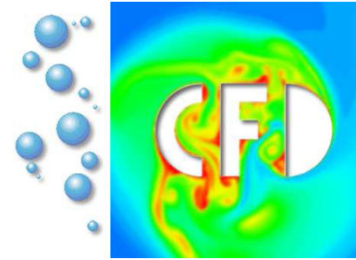
??? = ASX::AOS or ASX::SOA

[Strzodka. *Abstraction for AoS and SoA layout in C++* . GPU Computing Gems 2011]

# Overview

---

- **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/](http://www.mpi-inf.mpg.de/)

[~strzodka](#)

**Dominik Goddeke**

**Institute for Applied Mathematics**

**Technical University of Dortmund**

[www.mathematik.tu-dortmund.de/](http://www.mathematik.tu-dortmund.de/)

[~goeddeke](#)