Multi-resolution flow simulations on multi/many-core architectures

Dr. Diego Rossinelli

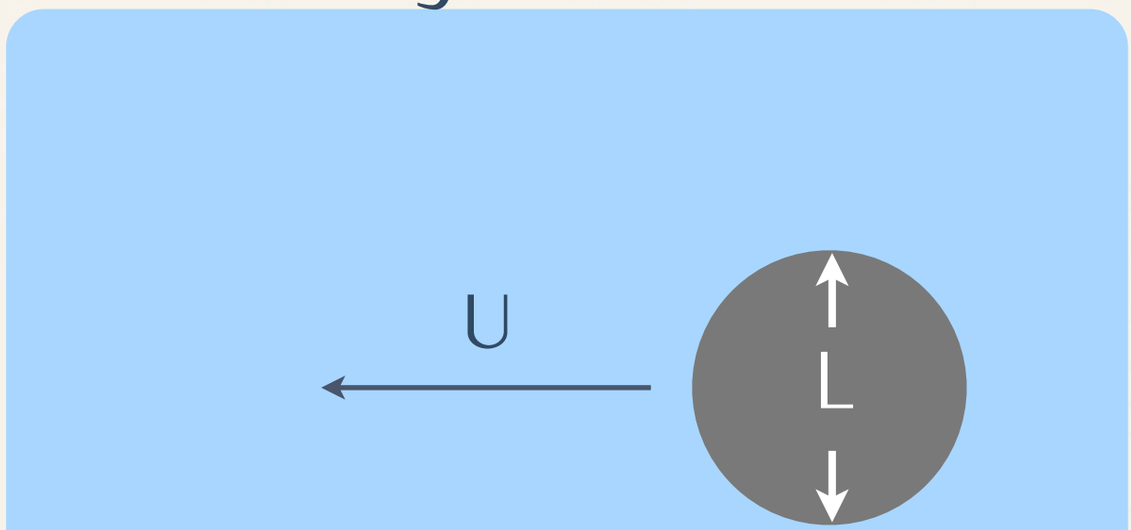
With: Babak Hejazialhosseini, Christian Conti, Petros Koumoutsakos

HPC-CH, Thursday, October 27, 2011

Flows and Scales

Flows and Scales

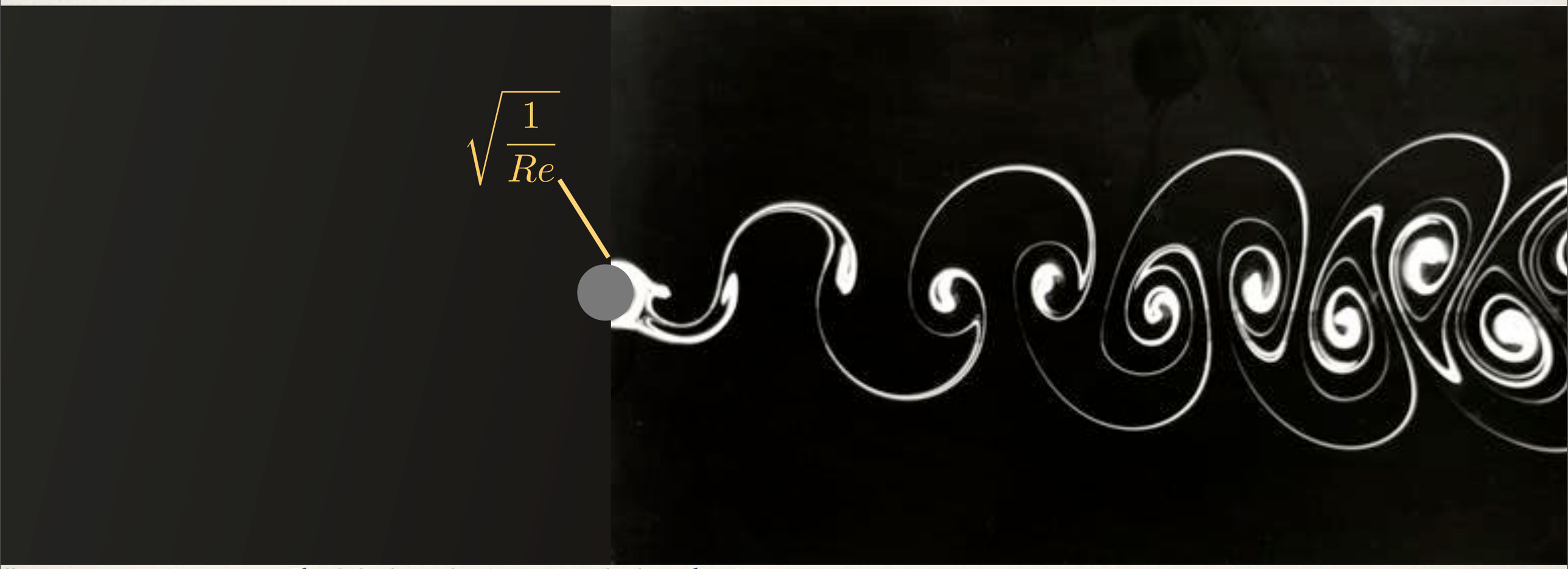
Bluff body flows



ν

$$Re = \frac{UL}{\nu}$$

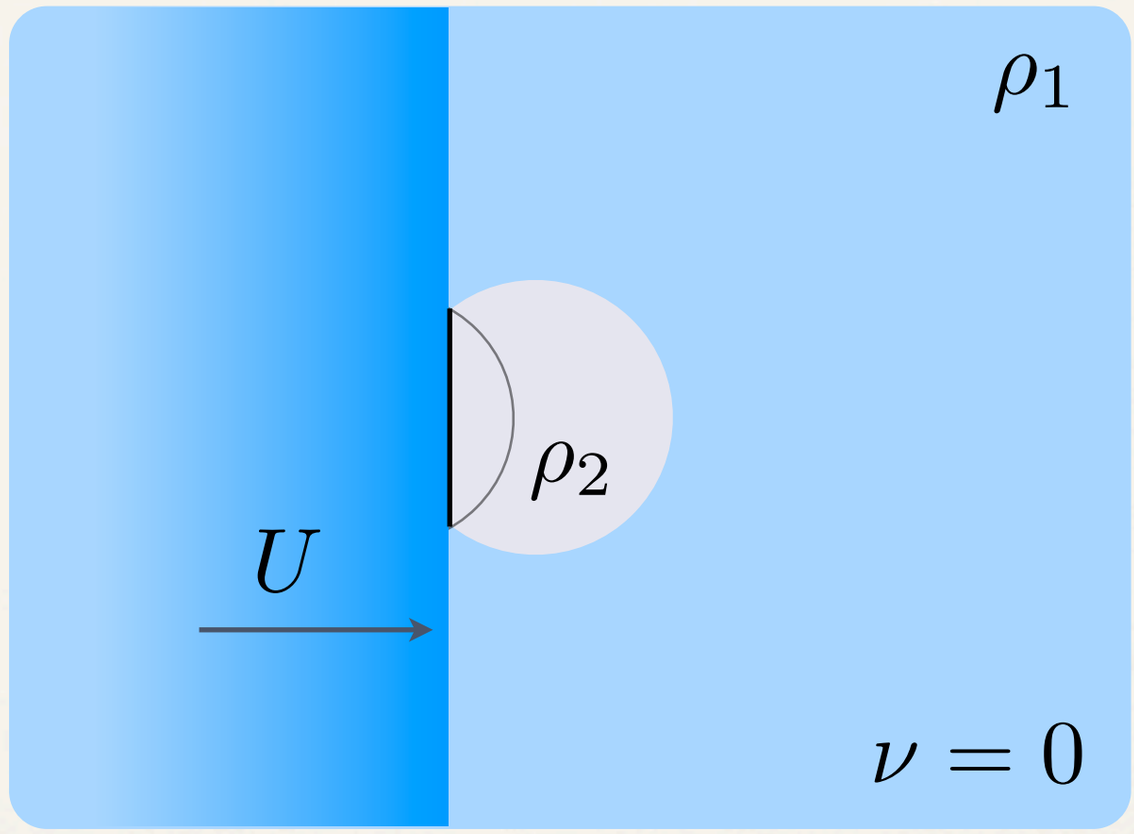
Flows and Scales



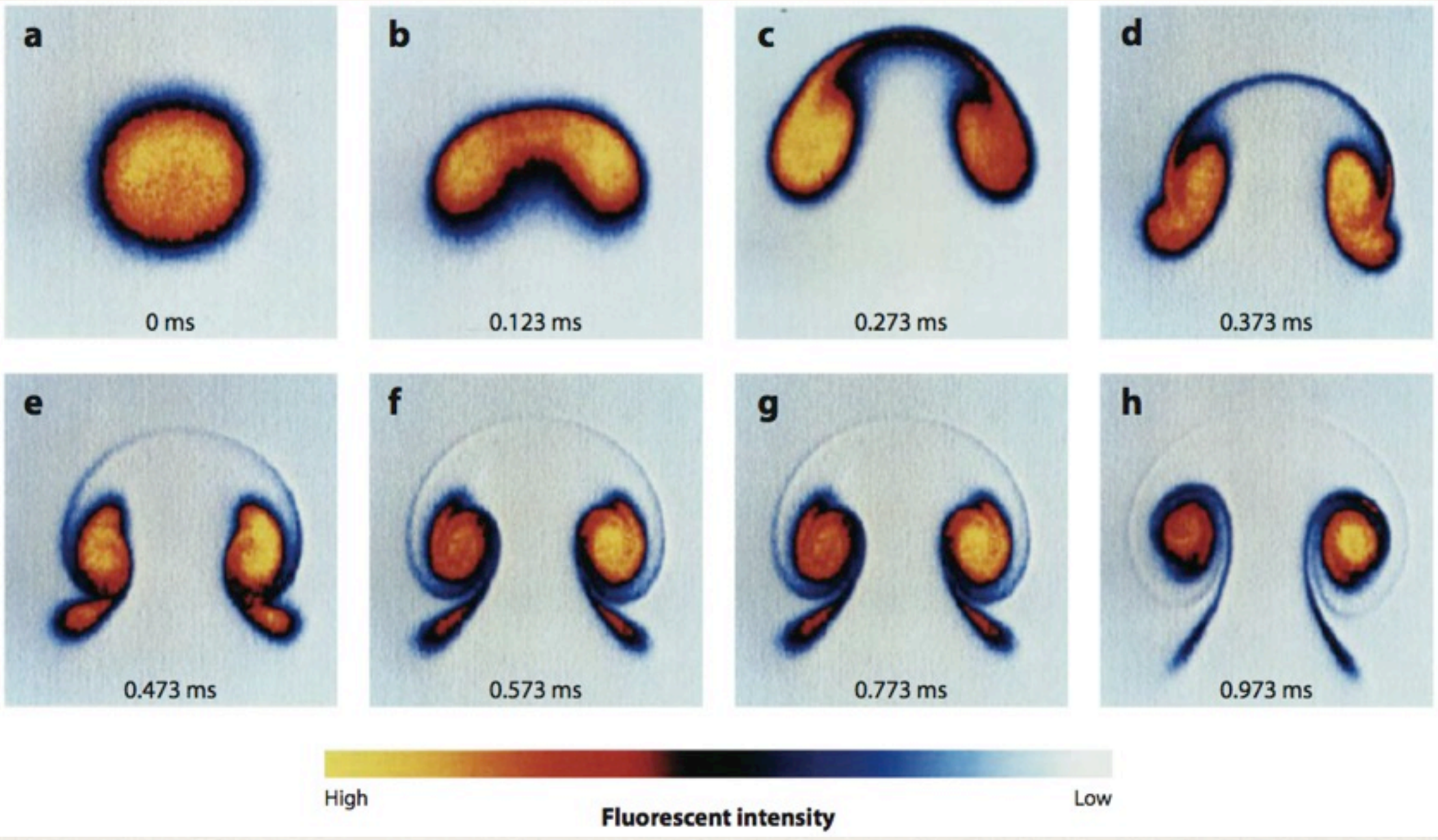
illiamson, Vortex Dynamics In The Cylinder Wake, Annu. Rev. Fluid. Mech., 1996

Flows and Scales

Shock-bubble interaction



Flows and Scales



Jacobs,Shock-induced mixing of a light-gas cylinder, J. Fluid Mech., 1992

Flows and Scales

Flows and Scales

- Bluff body flows (incompressible)

Flows and Scales

- Bluff body flows (incompressible)
- Shock bubble interaction (compressible)

Flows and Scales

- Bluff body flows (incompressible)
- Shock bubble interaction (compressible)
- Emergence of **multiple scales:**

Flows and Scales

- Bluff body flows (incompressible)
- Shock bubble interaction (compressible)
- Emergence of **multiple scales:**
  - From **wall boundaries**

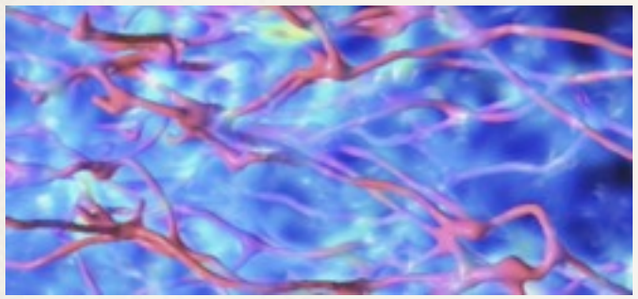
Flows and Scales

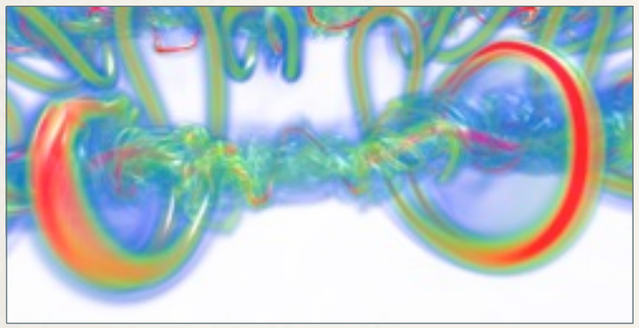
- Bluff body flows (incompressible)
- Shock bubble interaction (compressible)
- Emergence of **multiple scales:**
  - From **wall boundaries**
  - From **interface** between fluids

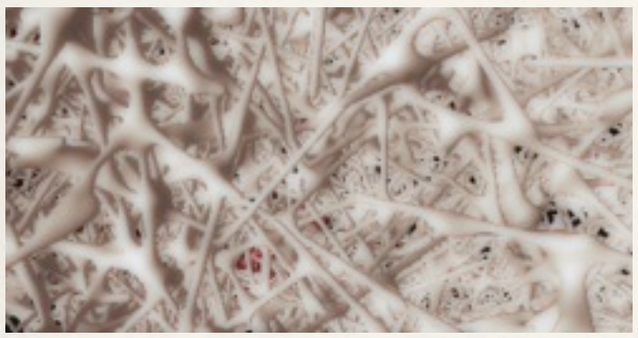
The need for adaptivity

- **Small (space/time) scales could be localizable**
  - Advection dominated process
  - Complex, unsteady geometries
  - Multi-physics phenomena (including growth)
- **Fast, high-res simulations**
  - Concentrate computation on small scales
  - **Efficient execution on multi/many-core**

The need for adaptivity







- **Small (space/time) scales could be localizable**
  - Advection dominated process
  - Complex, unsteady geometries
  - Multi-physics phenomena (including growth)
- **Fast, high-res simulations**
  - Concentrate computation on small scales
  - **Efficient execution on multi/many-core**

Wavelets

- Explicit control of the overall error
- High compression rates
- Multiresolution analysis

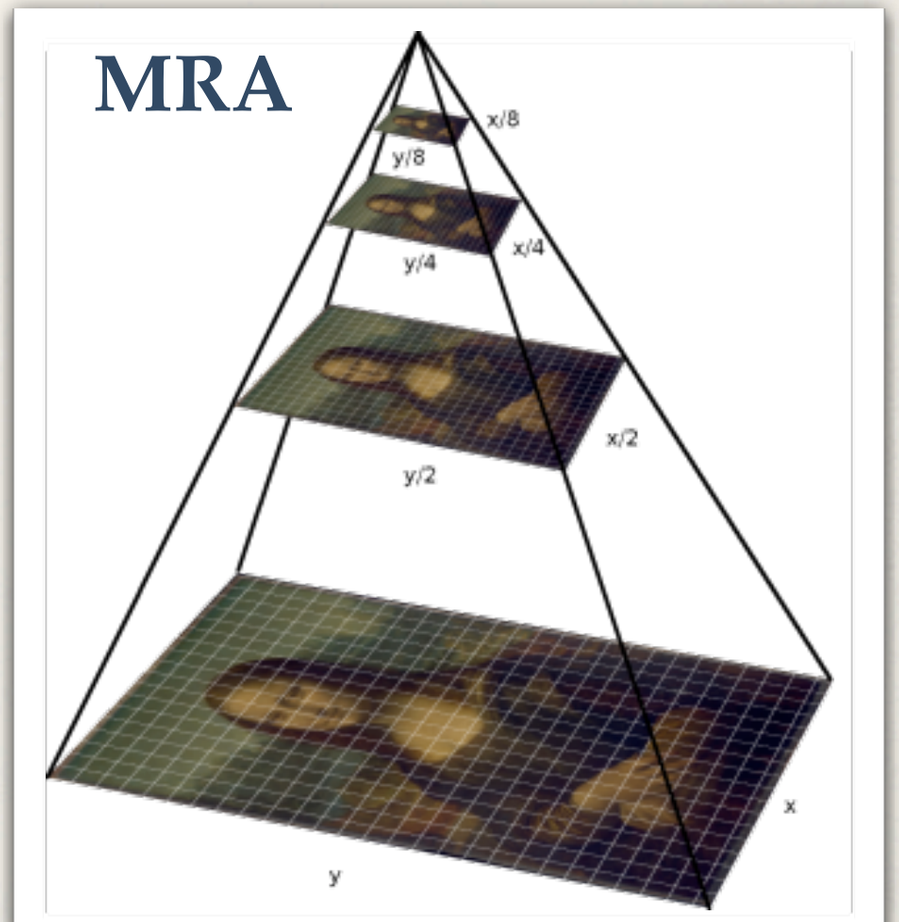
Wavelets

- Explicit control of the overall error
- High compression rates
- Multiresolution analysis

FWT and Inverse

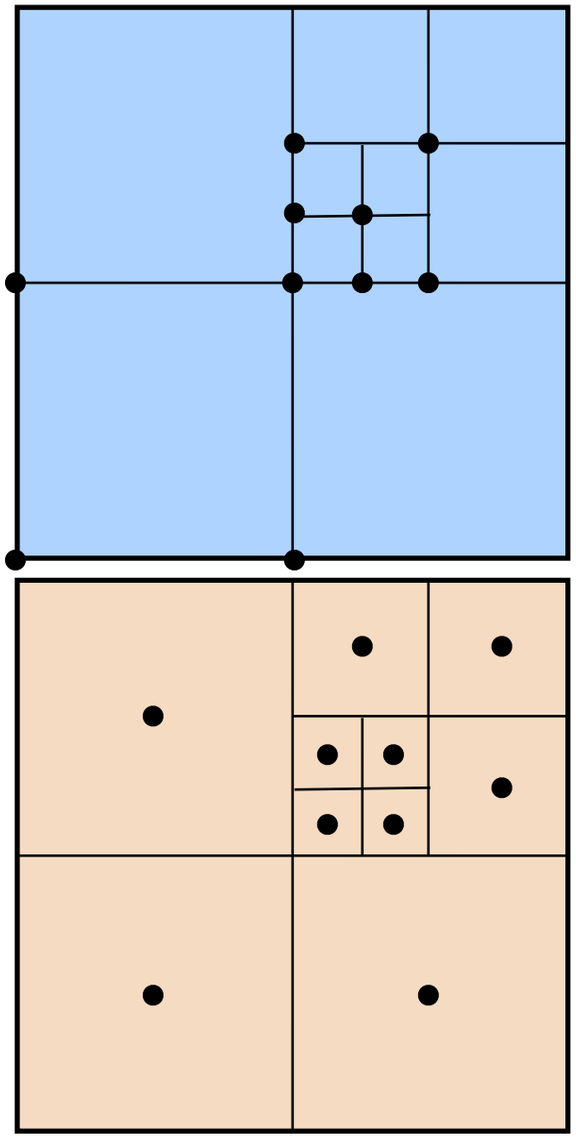
$$c_k^l = \sum_m h_{2k-m}^A c_m^{l+1},$$
$$d_k^l = \sum_m g_{2k-m}^A c_m^{l+1},$$

$$c_k^{l+1} = \sum_m h_{2m-k}^S c_m^l + \sum_m g_{2m-k}^S d_m^l.$$



1. Cohen and Daubechies, and Feauveau, Biorthogonal Bases of Compactly Supported Wavelets, Communication on Pure and Applied Mathematics, 1992

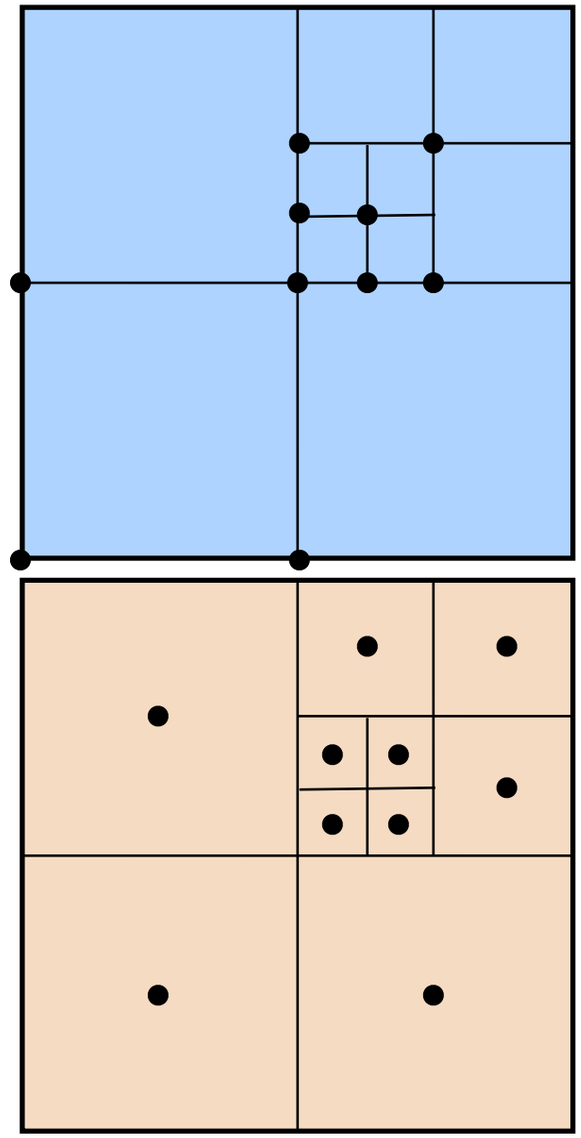
Wavelet-adapted grids



Wavelet-adapted grids

- Discard negligible grid points:

$$f_{\geq\varepsilon} = \sum_k c_k^0 \phi_k^0 + \sum_l \sum_{k:|d_k|>\varepsilon} d_k^l \psi_k^l$$



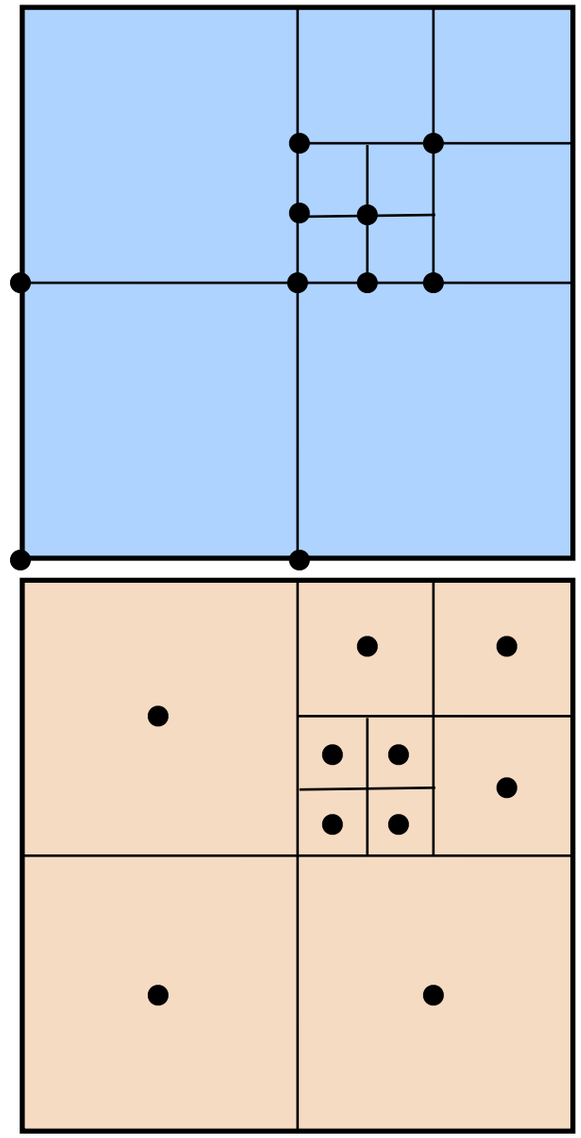
Wavelet-adapted grids

- Discard negligible grid points:

$$f_{\geq\varepsilon} = \sum_k c_k^0 \phi_k^0 + \sum_l \sum_{k:|d_k|>\varepsilon} d_k^l \psi_k^l$$

- Solve a PDE on the grid:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi = 0$$



Wavelet-adapted grids

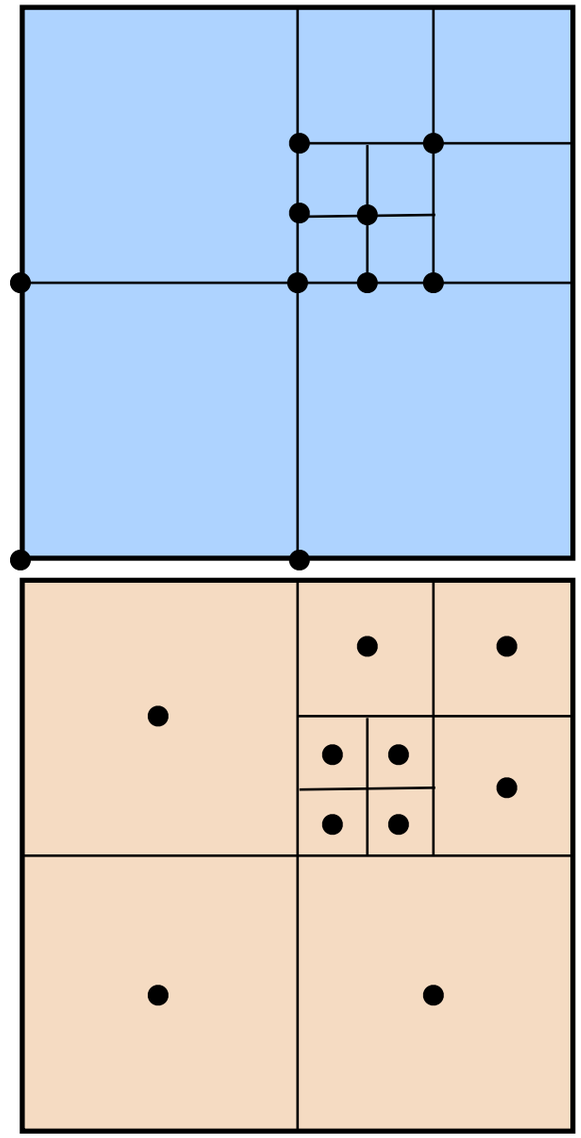
- Discard negligible grid points:

$$f_{\geq\varepsilon} = \sum_k c_k^0 \phi_k^0 + \sum_l \sum_{k:|d_k|>\varepsilon} d_k^l \psi_k^l$$

- Solve a PDE on the grid:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi = 0$$

- Spatial derivatives = uniform filtering



Wavelet-adapted grids

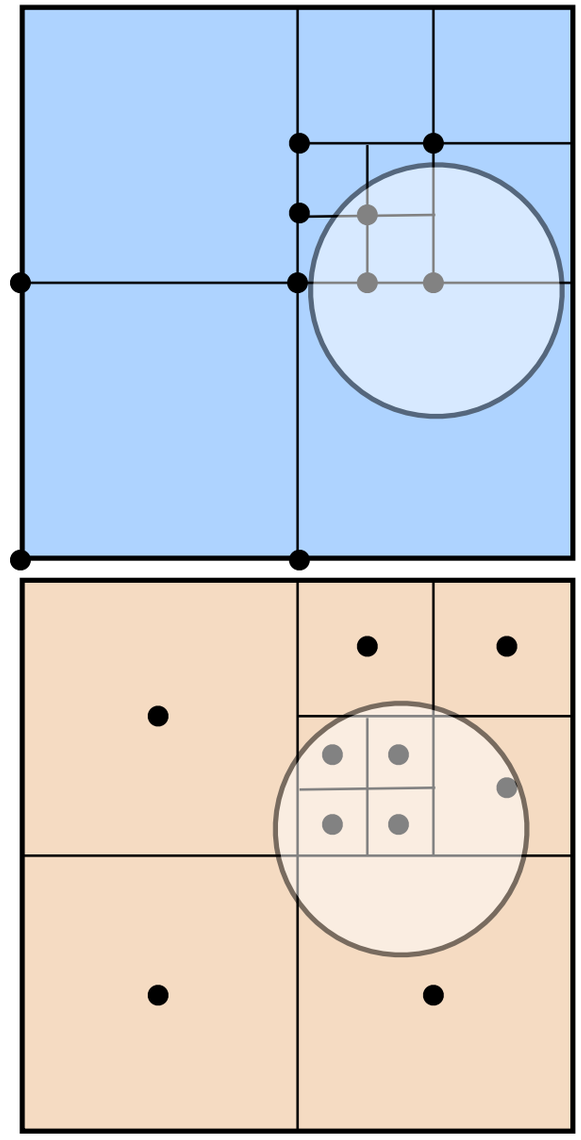
- Discard negligible grid points:

$$f_{\geq\varepsilon} = \sum_k c_k^0 \phi_k^0 + \sum_l \sum_{k:|d_k|>\varepsilon} d_k^l \psi_k^l$$

- Solve a PDE on the grid:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi = 0$$

- Spatial derivatives = uniform filtering



Wavelet-adapted grids

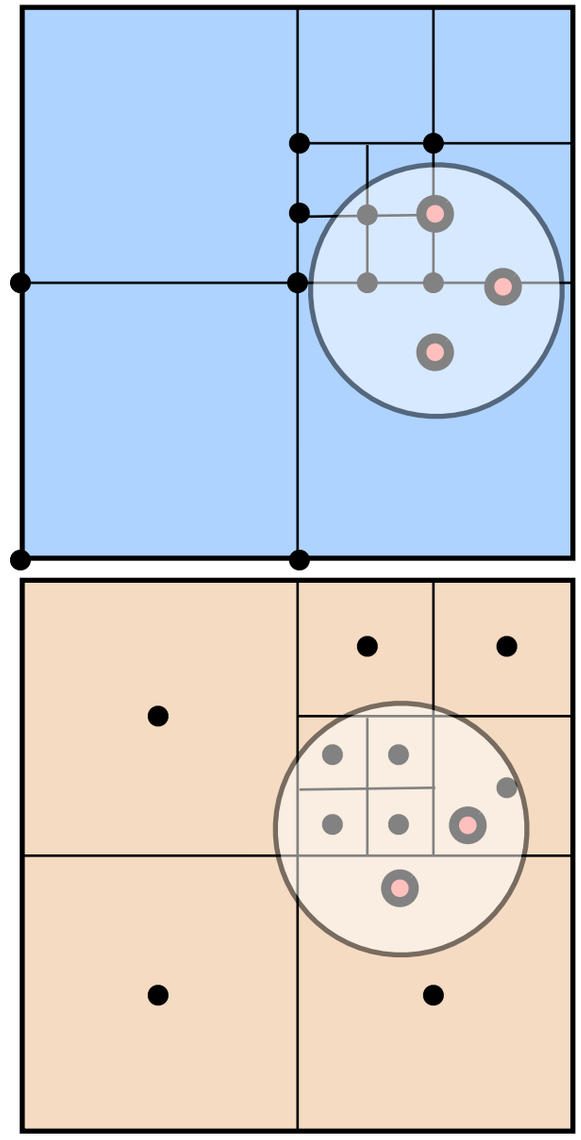
- Discard negligible grid points:

$$f_{\geq\varepsilon} = \sum_k c_k^0 \phi_k^0 + \sum_l \sum_{k:|d_k|>\varepsilon} d_k^l \psi_k^l$$

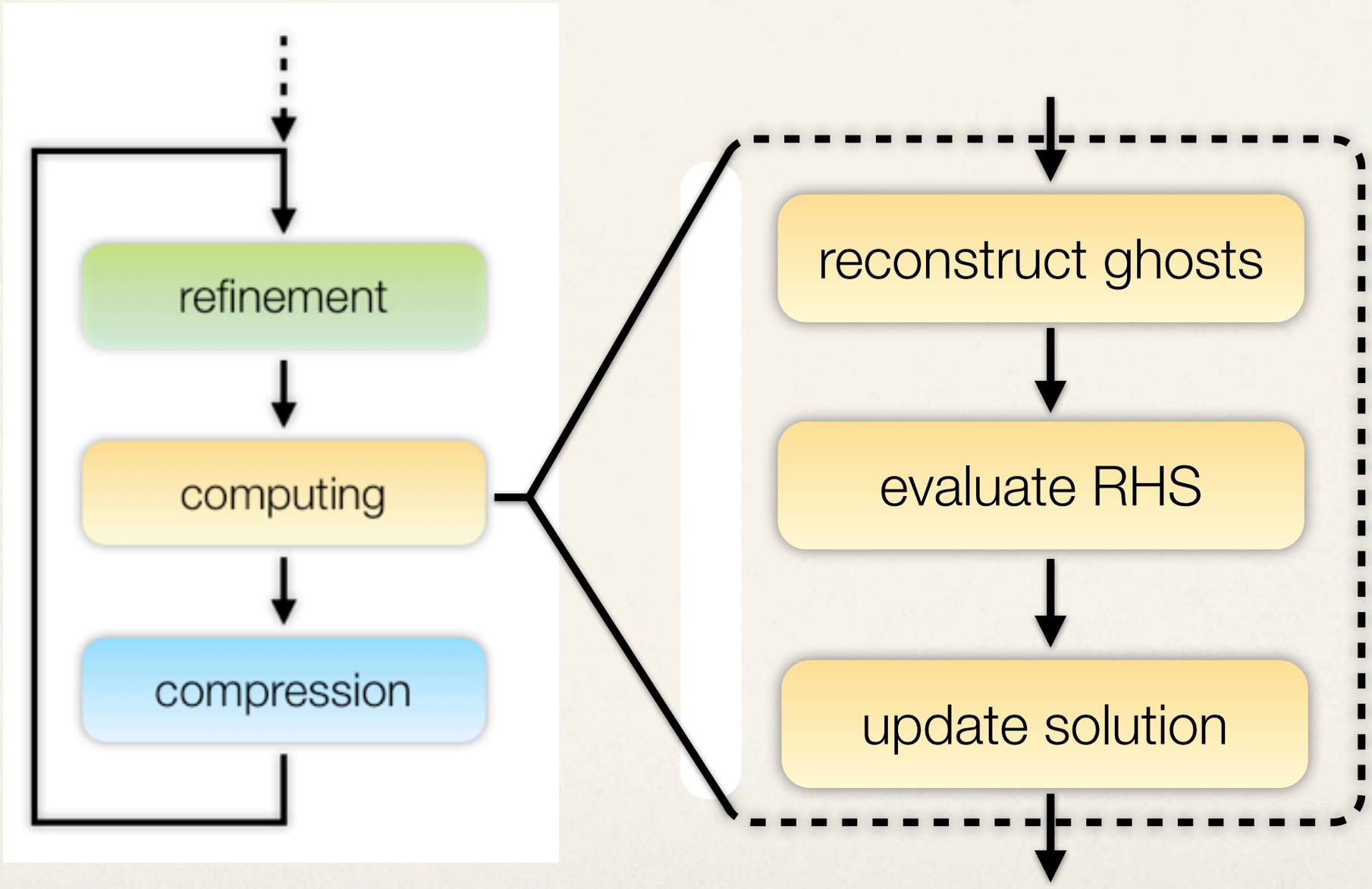
- Solve a PDE on the grid:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u} \cdot \nabla \phi = 0$$

- Spatial derivatives = uniform filtering

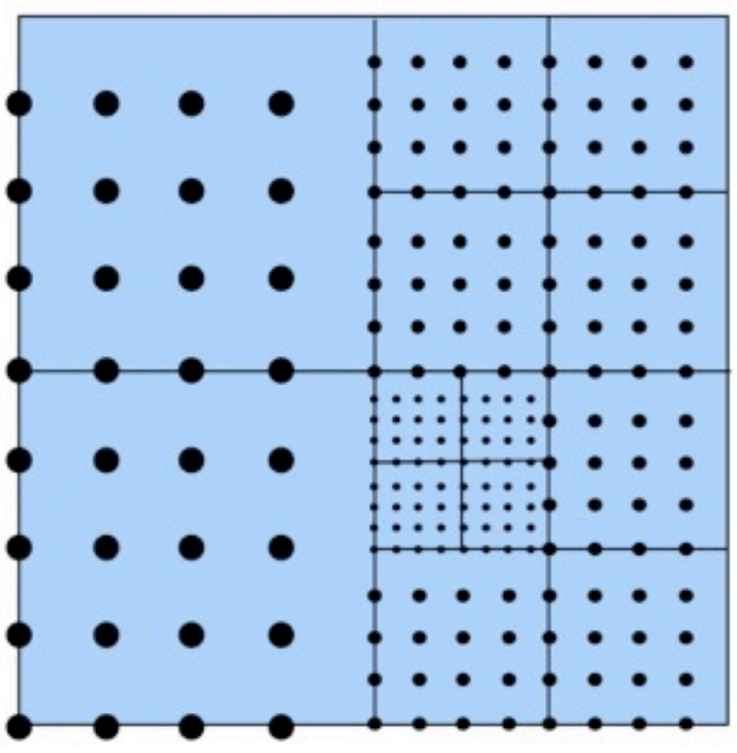


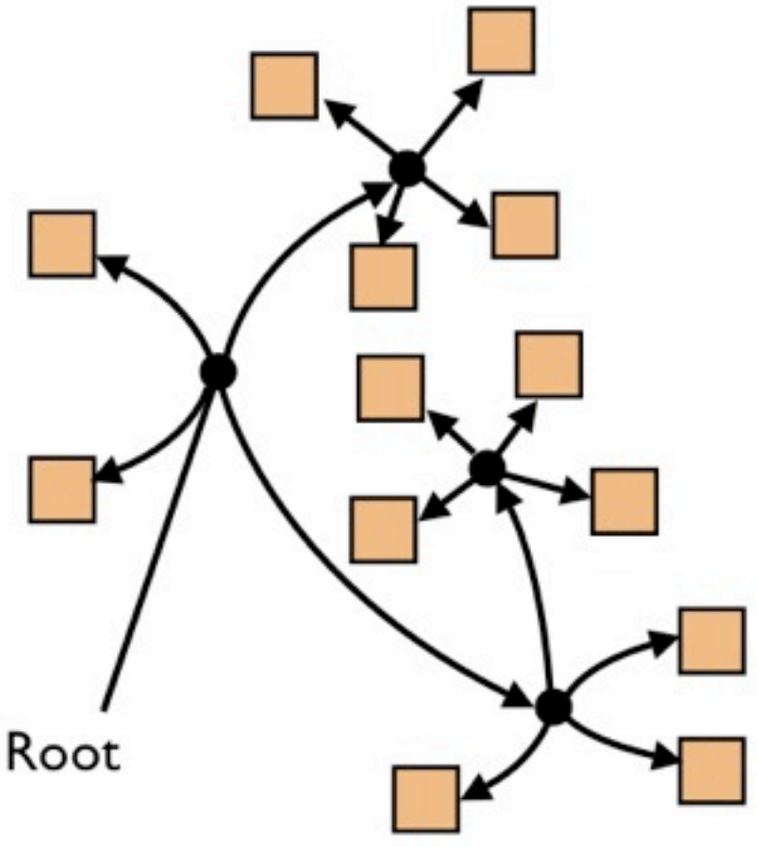
How to capture new scales?



Wavelet blocks

- Neighbors look-up: less memory indirections
- Less #ghosts
- Efficient ghost reconstructions
- Within a block: random access





GPU acceleration

- Split the computing stage:

  1. Task parallel, load-unbalanced ghost computing (multi-core)

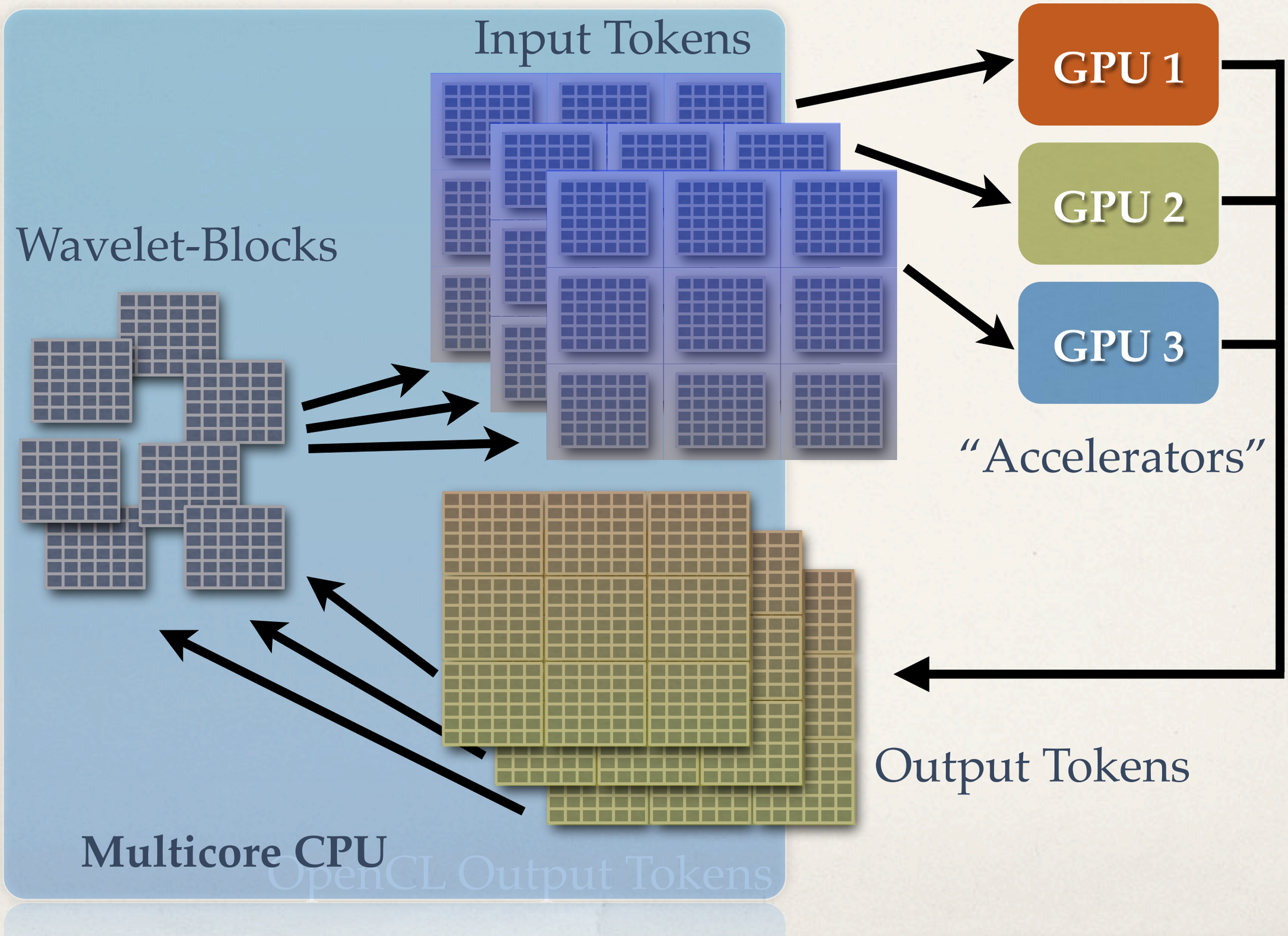
  2. Fine-grained data parallelism for RHS (GPUs)

  3. Integration step (multi-core)

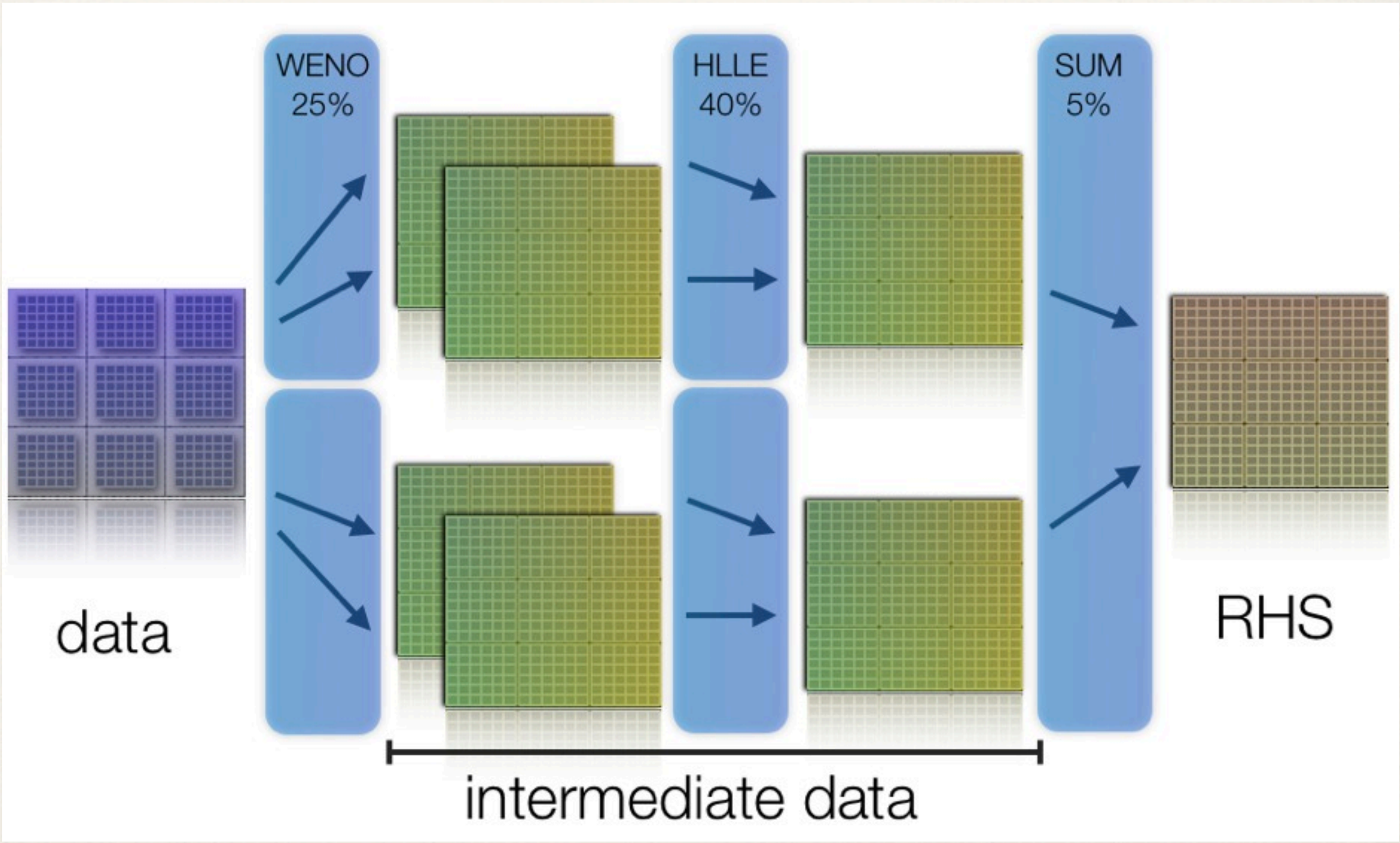
$$\mathbf{q}^{\mathbf{new}} = \mathbf{q}^{\mathbf{old}} + \delta t \underbrace{\mathbf{F}\left(\mathbf{q}^{\mathbf{old}}, \nabla \mathbf{q}^{\mathbf{old}}\right)}_{\text{GPUs}}$$

- **How much *faster*** than CPU-only execution?
- **How much *different*** are CPU/GPU and CPU-only solutions?

Execution model



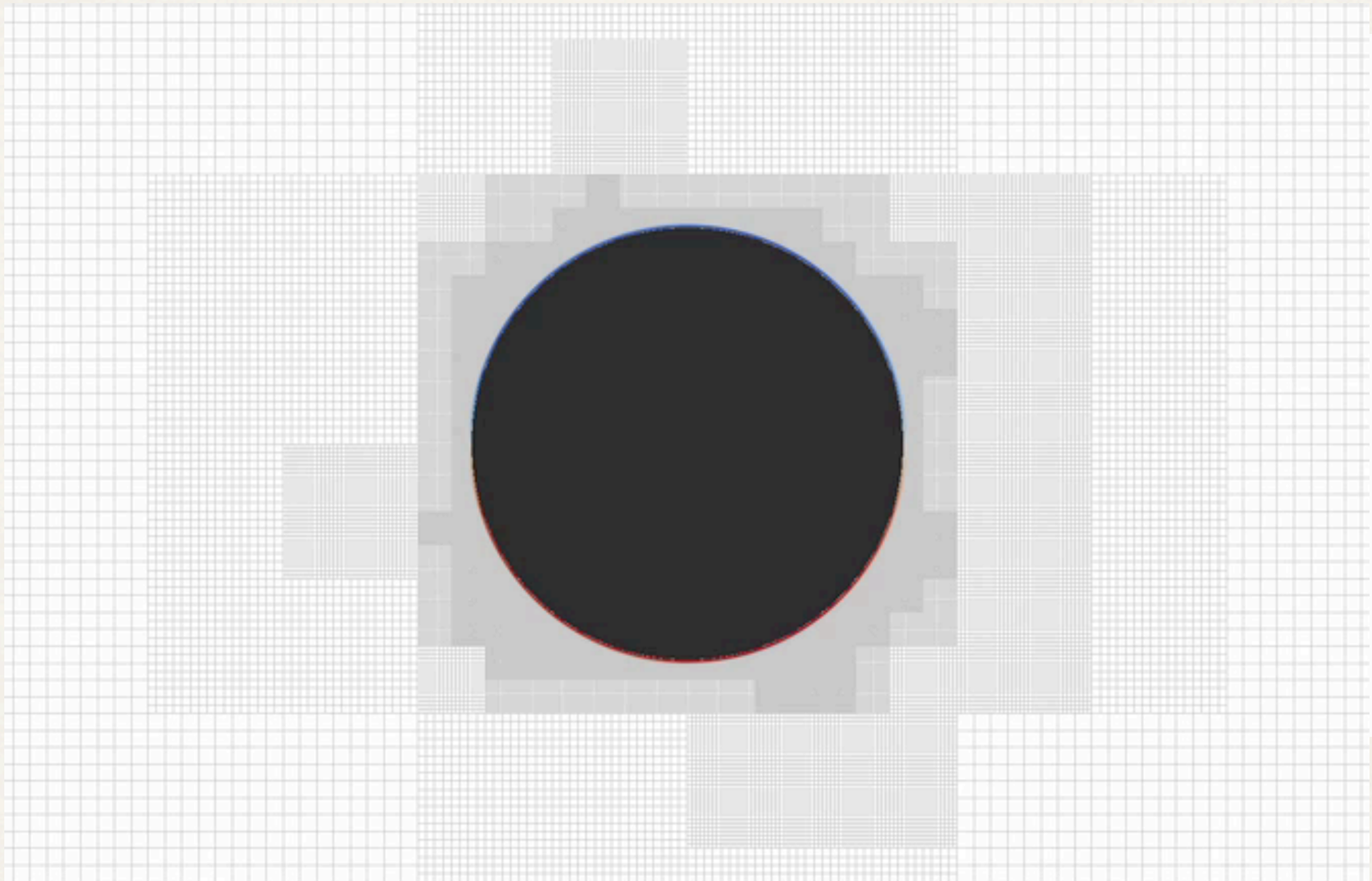
GPU kernels



Re 9500

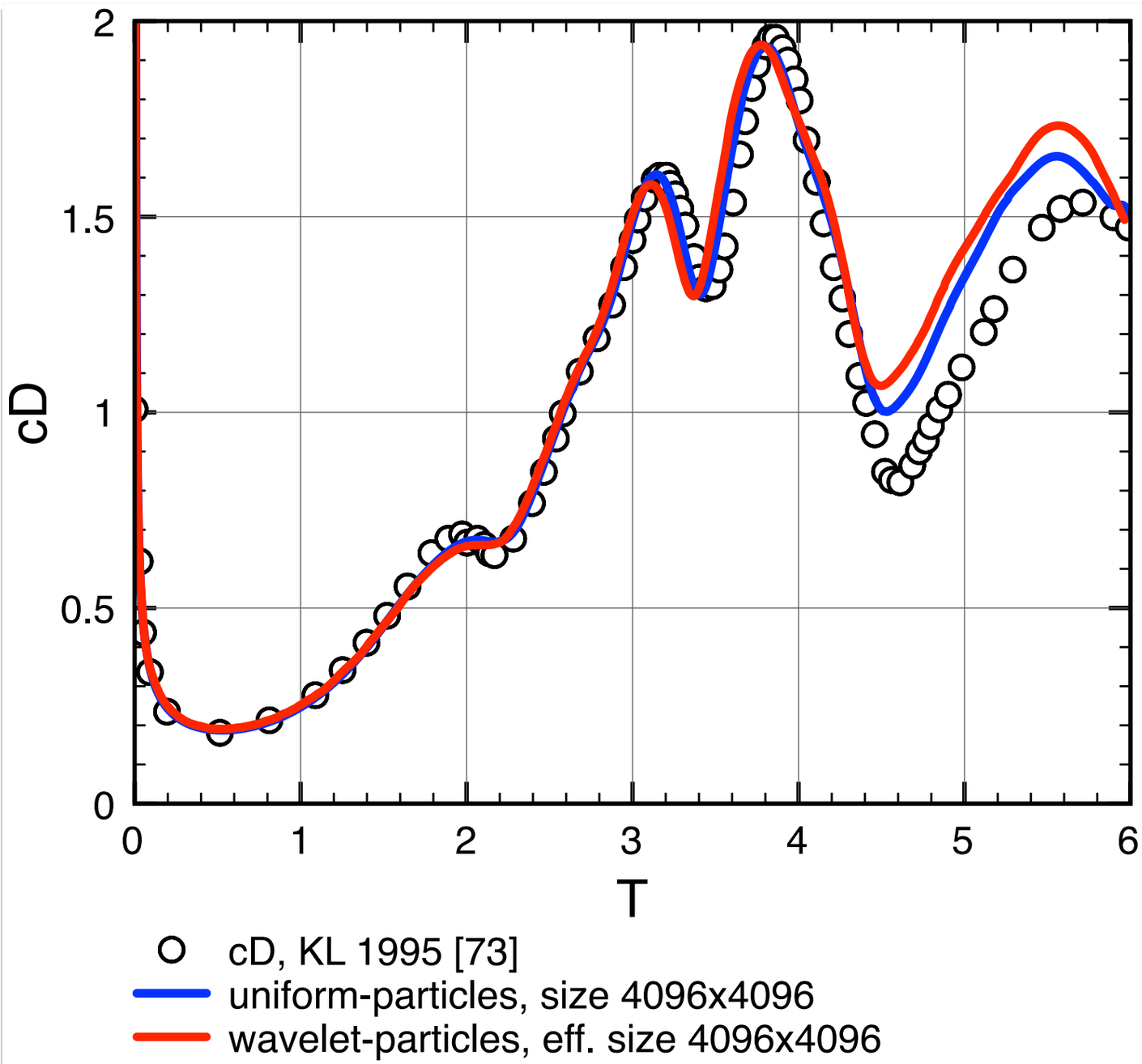
Accuracy

Re 9500



Accuracy

Re 9500

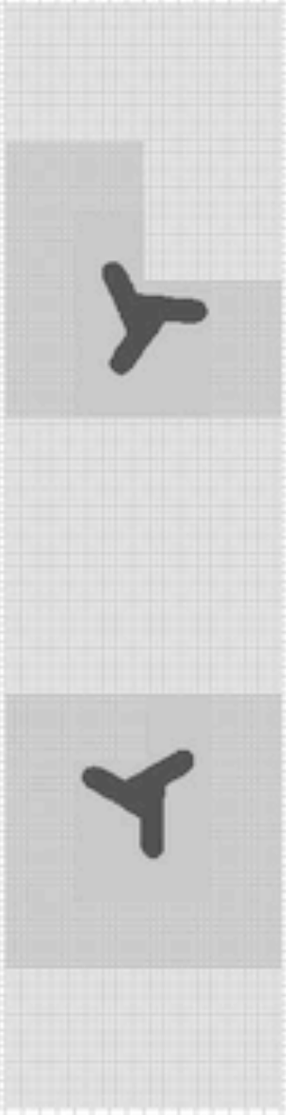


Accuracy

Results: complex geometries

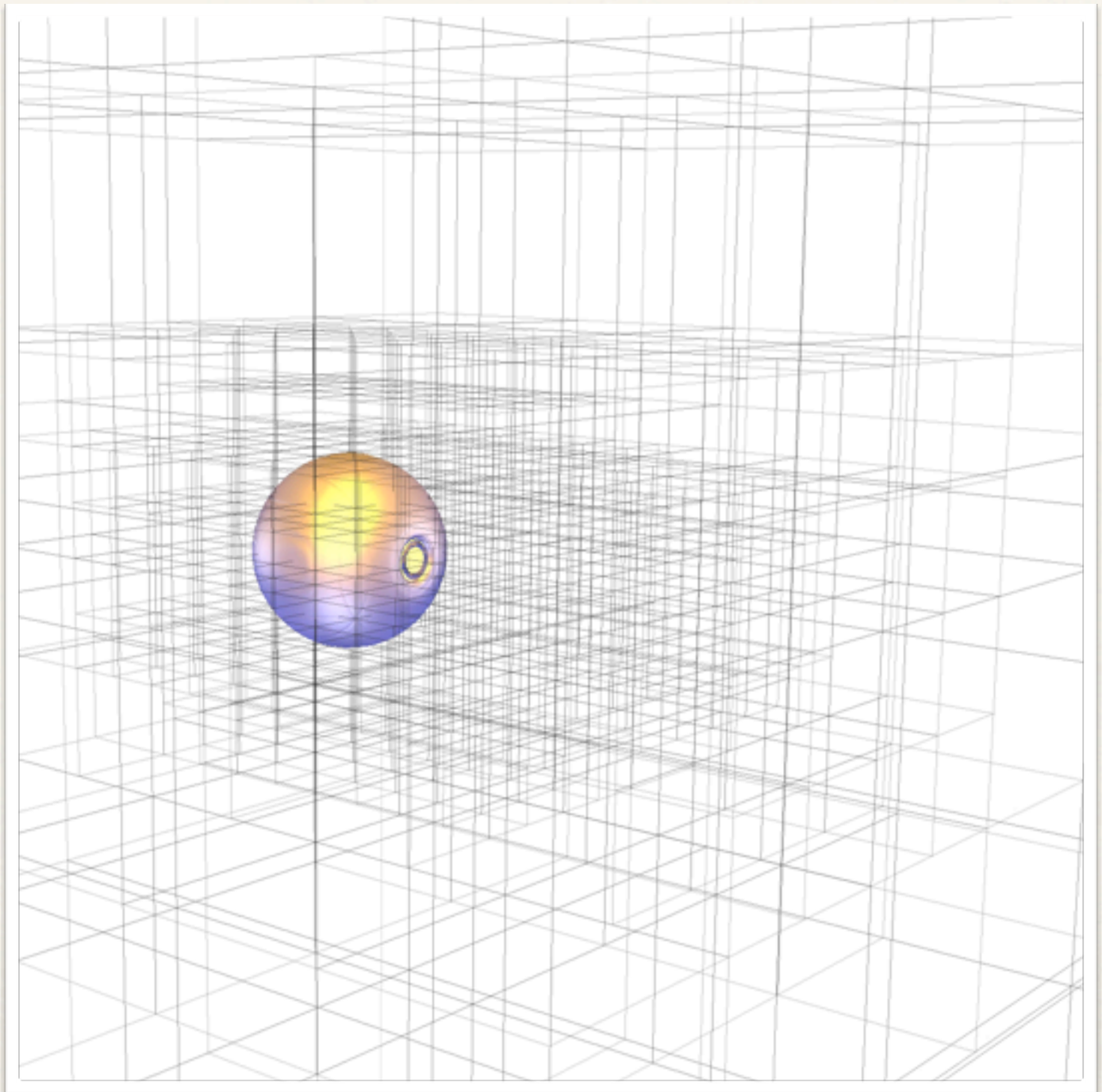
Results: complex geometries

Results: complex geometries



Results: 3d simulations, Re 1000

Results: 3d simulations, Re 1000



SIMDization of tree codes

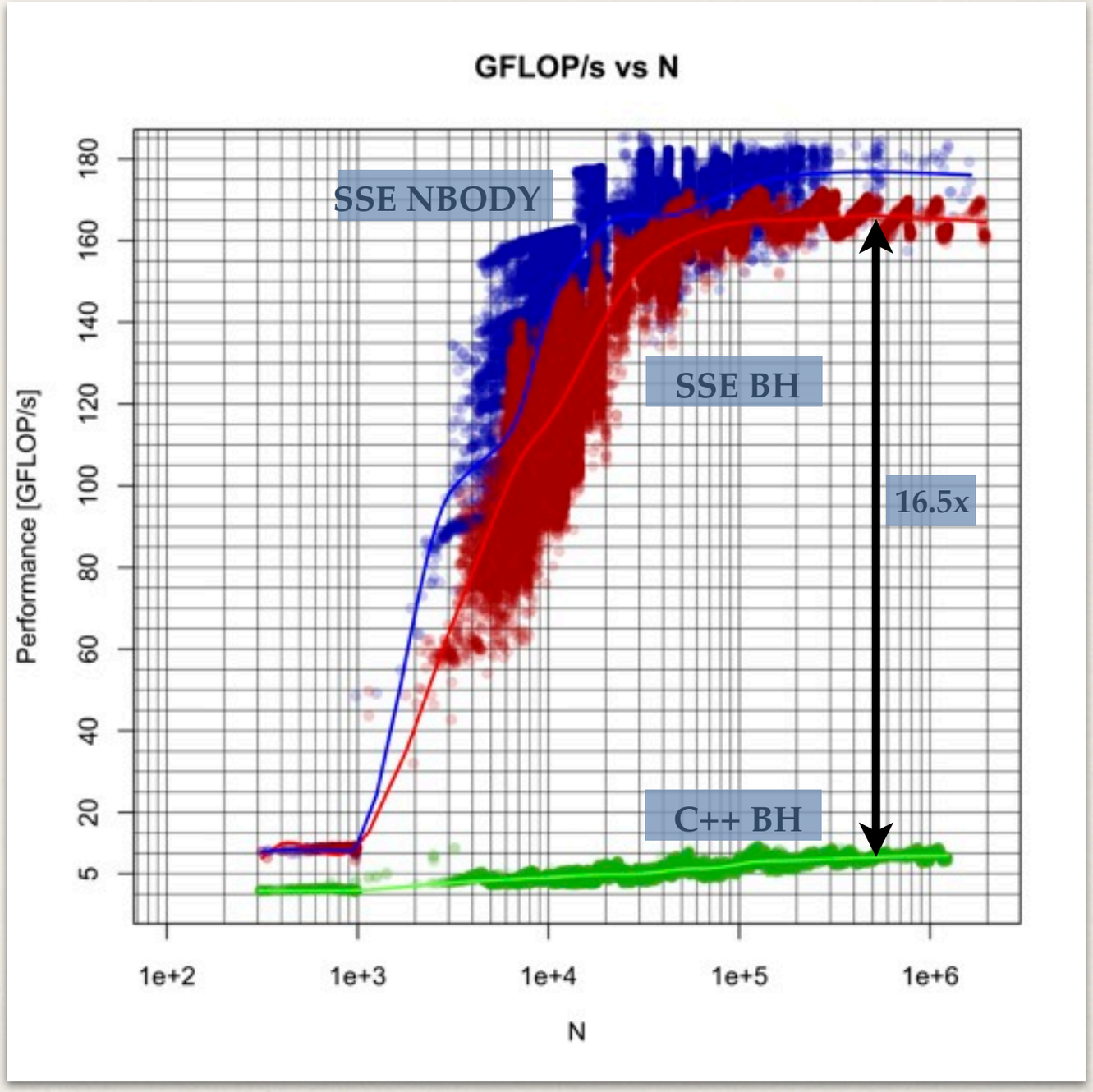
Computation on brutus cluster:

- One AMD 4P Quad-core node
- Opteron 8380 core @ 2.5 GHz

➡ **Peak performance: 320 GFLOP/s**

- Tree code: **180 GFLOP/s (clean FLOPs)**

➡ **Hardware utilization: 55%**



SIMD

Computation on l

- One AMD 4P Q
- Opteron 8380 c

➡ **Peak performa**

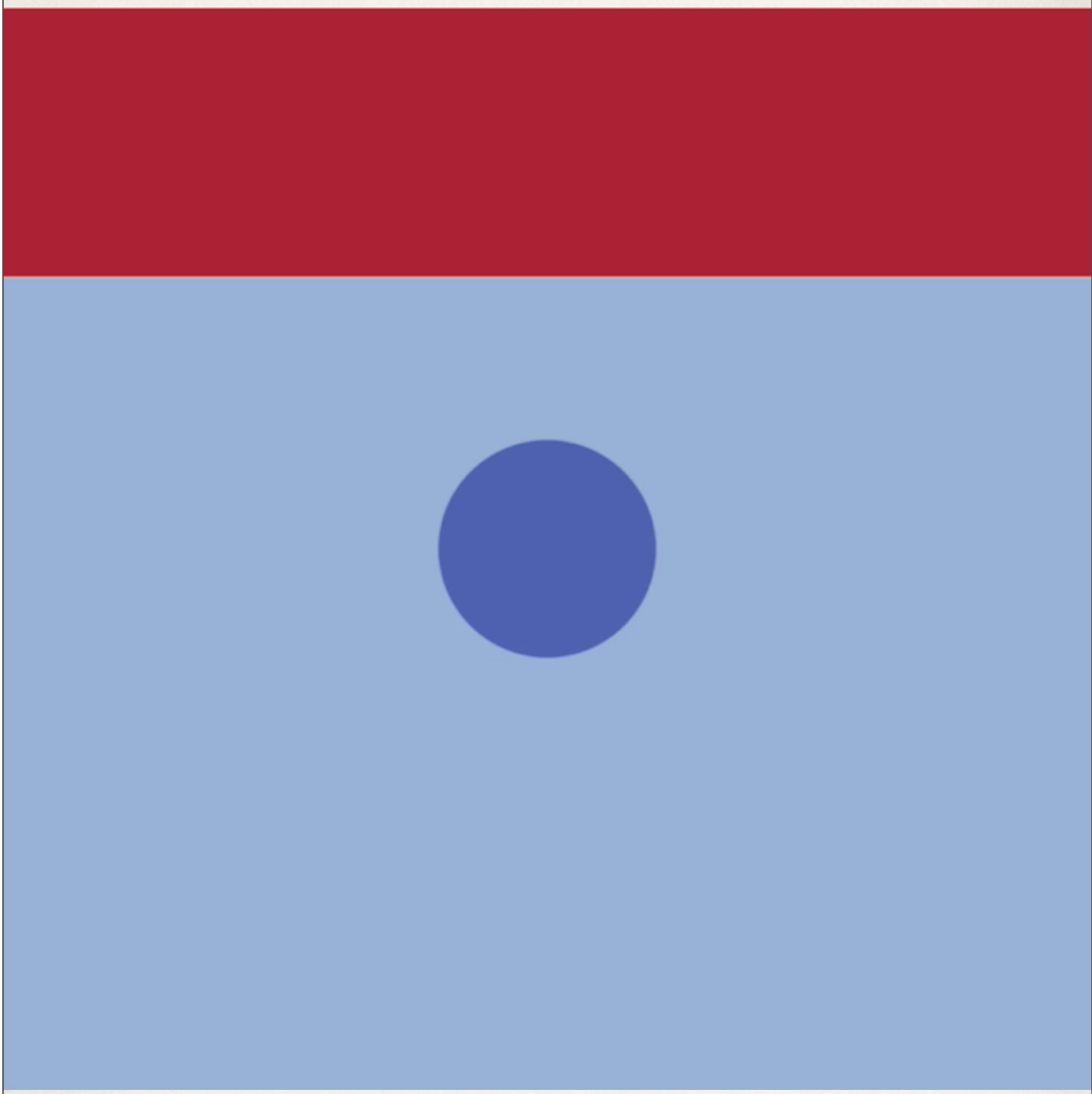
- Tree code: **180 G**

➡ **Hardware util**

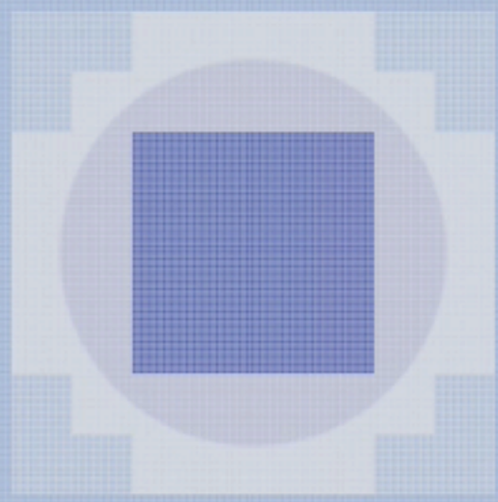
```
template<>
void DirectInteractions::kernel<false>(const __m128 xd, const __m128 yd,
                                       const __m128 xs, const __m128 ys, const __m128 ws,
                                       float * const u_dest, float * const v_dest)
{
        const __m128 rx1 = xd - _select<0>(xs);
        const __m128 ry1 = yd - _select<0>(ys);
        const __m128 r2A = rx1*rx1 + ry1*ry1;

        const __m128 rx2 = xd - _select<1>(xs);
        const __m128 ry2 = yd - _select<1>(ys);
        const __m128 r2B = rx2*rx2 + ry2*ry2;

        const __m128 rx3 = xd - _select<2>(xs);
        const __m128 ry3 = yd - _select<2>(ys);
        const __m128 r2C = rx3*rx3 + ry3*ry3;

        const __m128 rx4 = xd - _select<3>(xs);
        const __m128 ry4 = yd - _select<3>(ys);
        const __m128 r2D = rx4*rx4 + ry4*ry4;

#ifdef _FMM_NOPRECDIV_KERNELS_
        const __m128 factor1 =  worse_division(_select<0>(ws), r2A);
        const __m128 factor2 =  worse_division(_select<1>(ws), r2B);
        const __m128 factor3 =  worse_division(_select<2>(ws), r2C);
        const __m128 factor4 =  worse_division(_select<3>(ws), r2D);
#else
        const __m128 factor1 =  _mm_div_ps(_select<0>(ws), r2A);
        const __m128 factor2 =  _mm_div_ps(_select<1>(ws), r2B);
        const __m128 factor3 =  _mm_div_ps(_select<2>(ws), r2C);
        const __m128 factor4 =  _mm_div_ps(_select<3>(ws), r2D);
#endif

        _mm_store_ps(u_dest, _mm_sub_ps(_mm_load_ps(u_dest), factor1*ry1));
        _mm_store_ps(u_dest, _mm_sub_ps(_mm_load_ps(u_dest), factor2*ry2));
        _mm_store_ps(u_dest, _mm_sub_ps(_mm_load_ps(u_dest), factor3*ry3));
        _mm_store_ps(u_dest, _mm_sub_ps(_mm_load_ps(u_dest), factor4*ry4));

        _mm_store_ps(v_dest, _mm_add_ps(_mm_load_ps(v_dest), factor1*rx1));
        _mm_store_ps(v_dest, _mm_add_ps(_mm_load_ps(v_dest), factor2*rx2));
        _mm_store_ps(v_dest, _mm_add_ps(_mm_load_ps(v_dest), factor3*rx3));
        _mm_store_ps(v_dest, _mm_add_ps(_mm_load_ps(v_dest), factor4*rx4));
    }
```

Mach 3, 12 cores, 3 GPUs T10

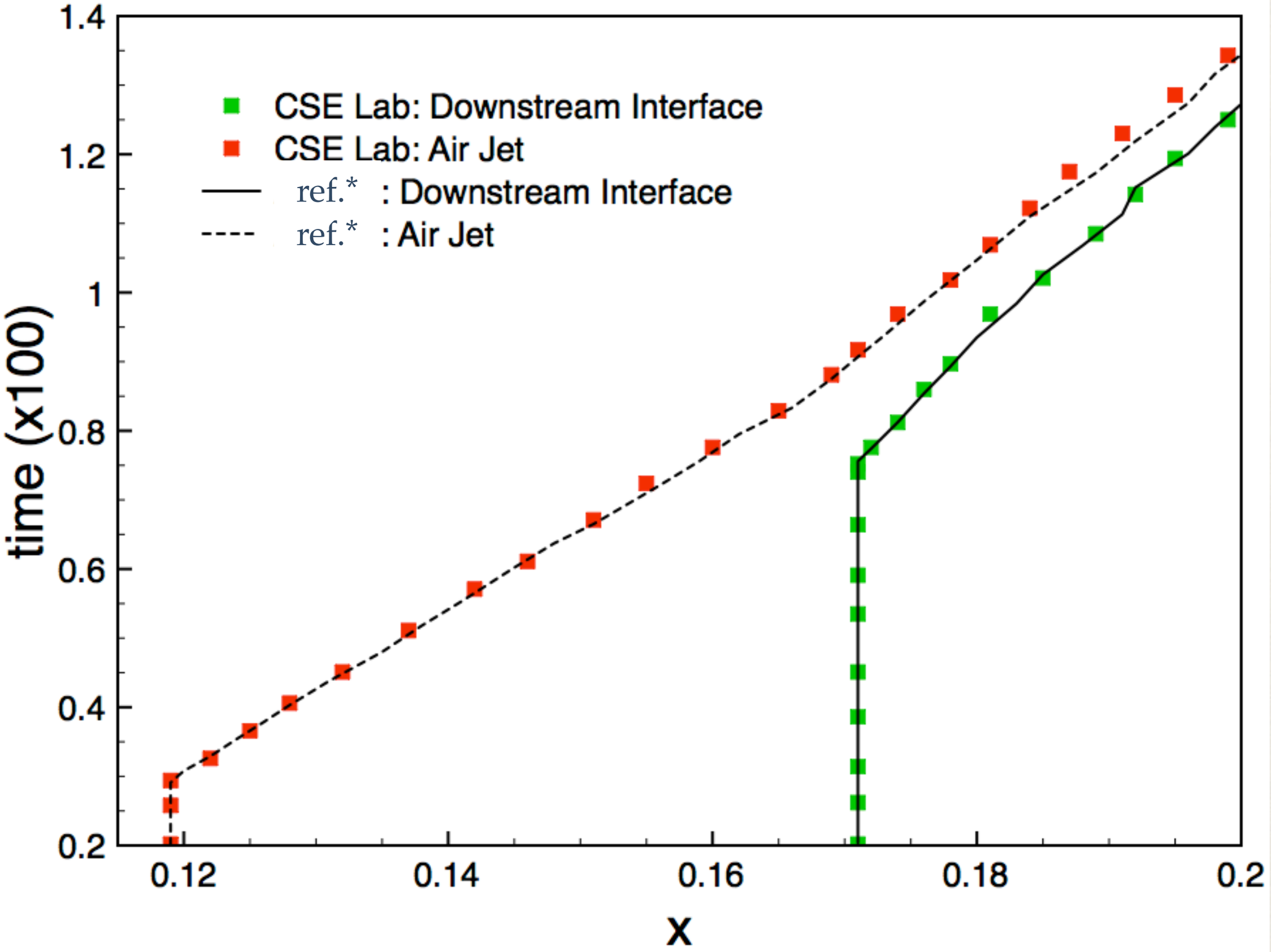
Mach 3, 12 cores, 3 GPUs T10



Mach 3, 12 cores, 3 GPUs T10

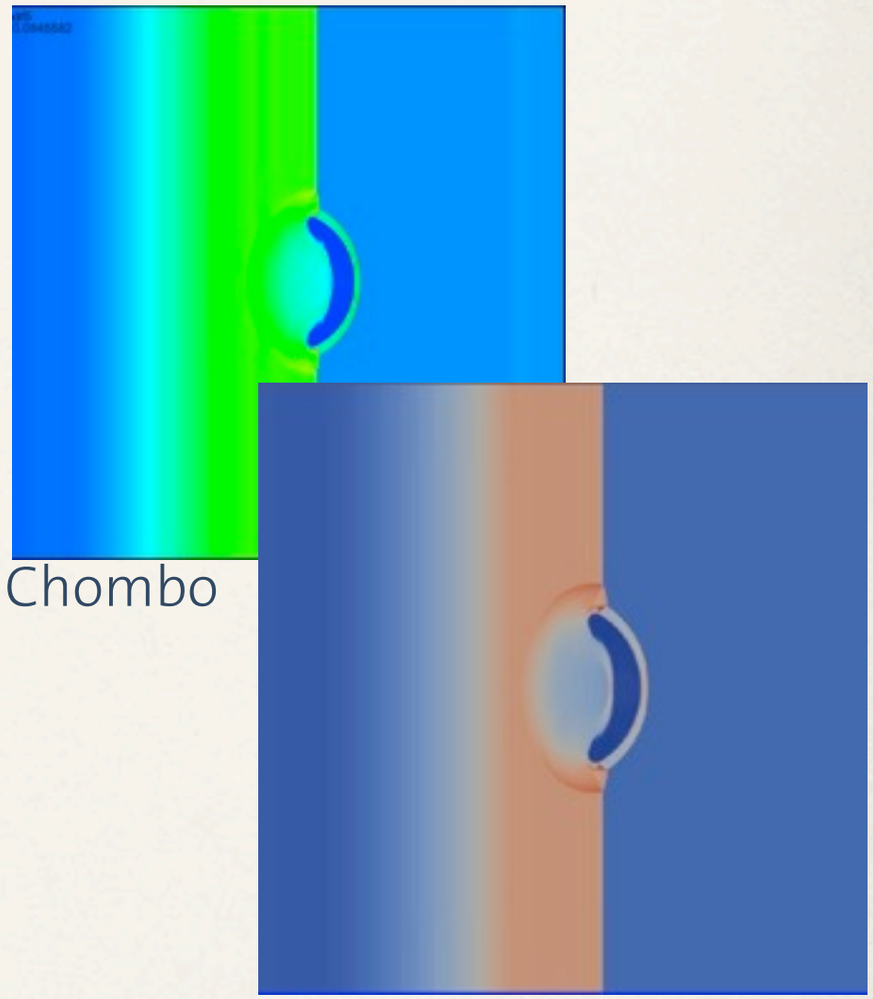


Mach 3, 12 cores, 3 GPUs T10



*Hu, Khoo, Adams and Huang, 2006

Is Chombo faster?



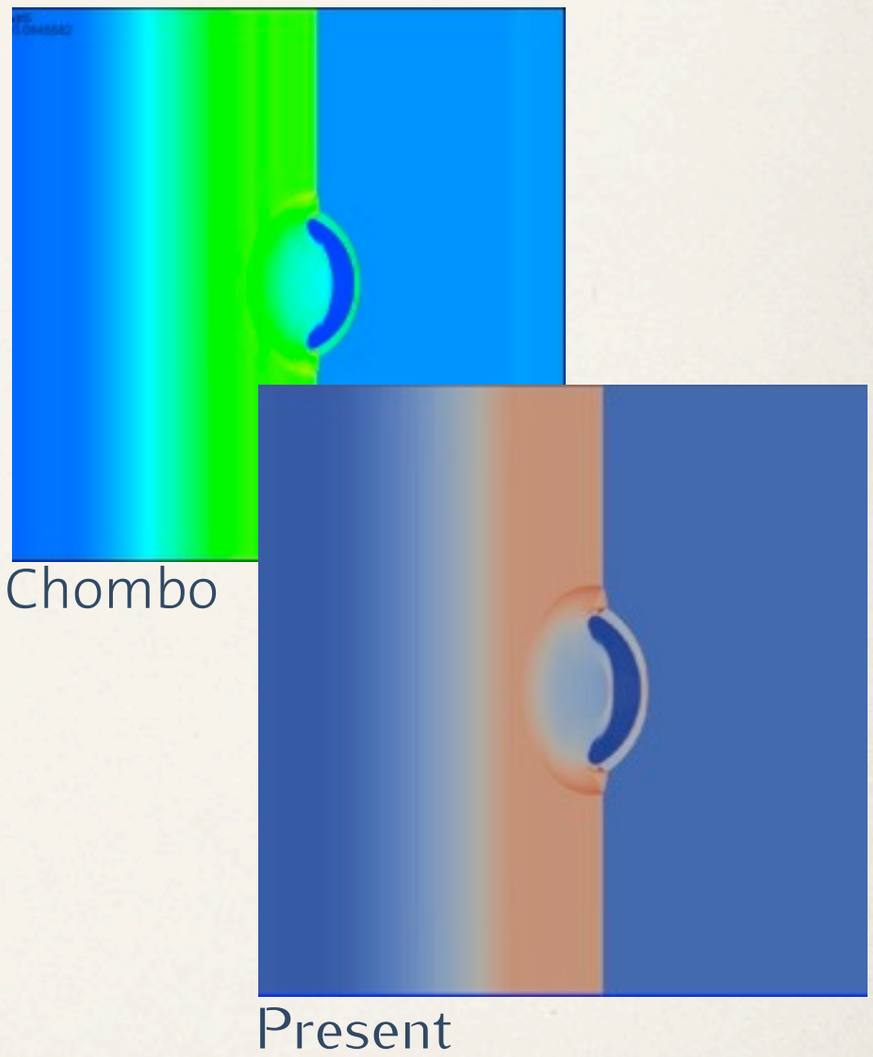
Chombo

Present

Is Chombo faster?

Chombo

- Alternatives: CLAWPACK, AMRITA
- Single-phase
- 2nd-order spatial discretization



1. CHOMBO: Colella et al., software package for AMR applications, Technical Report(LBNL), 2000
2. CLAWPACK: LeVeque, software, http://www.amath.washington.edu/~claw/, 1997
3. AMRITA: Quirk, An Introduction to Amrita. http://www.amrita-cfd.com, 1997

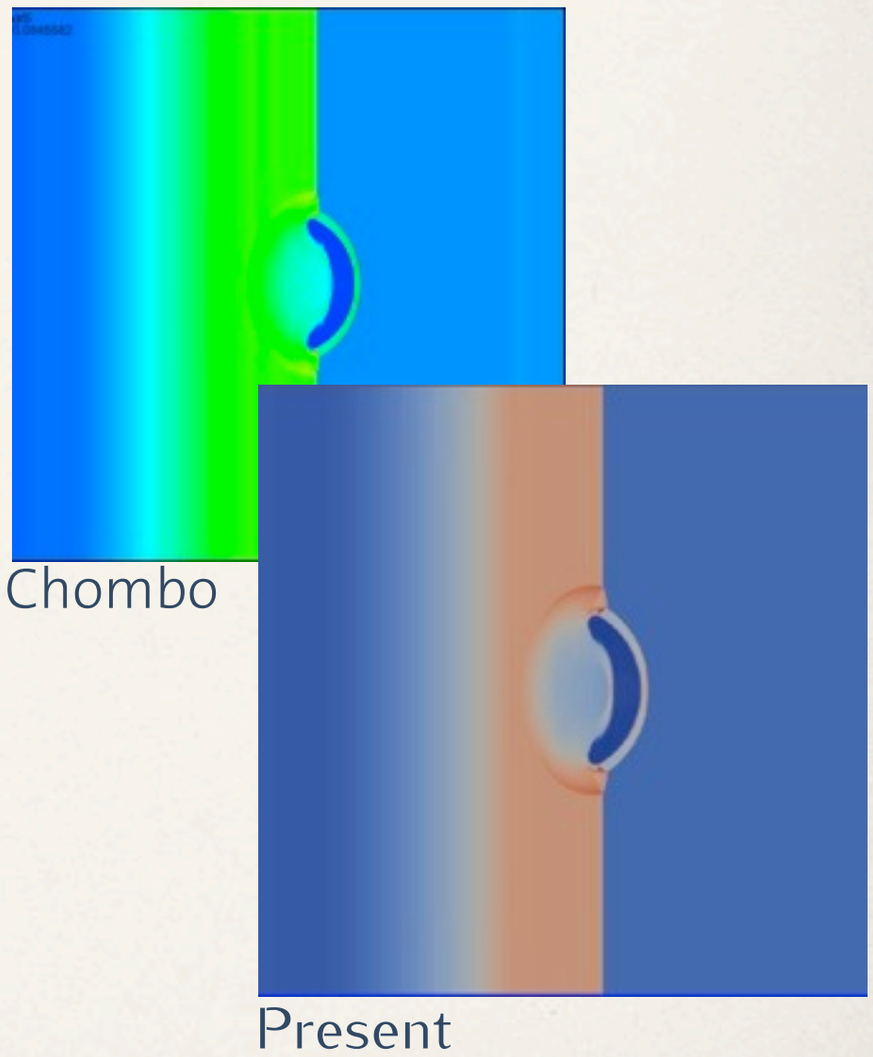
Is Chombo faster?

Chombo

- Alternatives: CLAWPACK, AMRITA
- Single-phase
- 2nd-order spatial discretization

Present solver

- Multi-phase
- 5th-order spatial discretization



1. CHOMBO: Colella et al., software package for AMR applications, Technical Report(LBNL), 2000
2. CLAWPACK: LeVeque, software, http://www.amath.washington.edu/~claw/, 1997
3. AMRITA: Quirk, An Introduction to Amrita. http://www.amrita-cfd.com, 1997

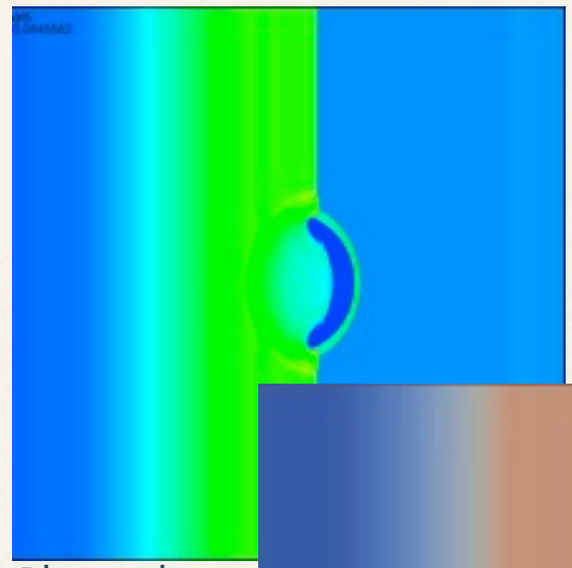
Is Chombo faster?

Chombo

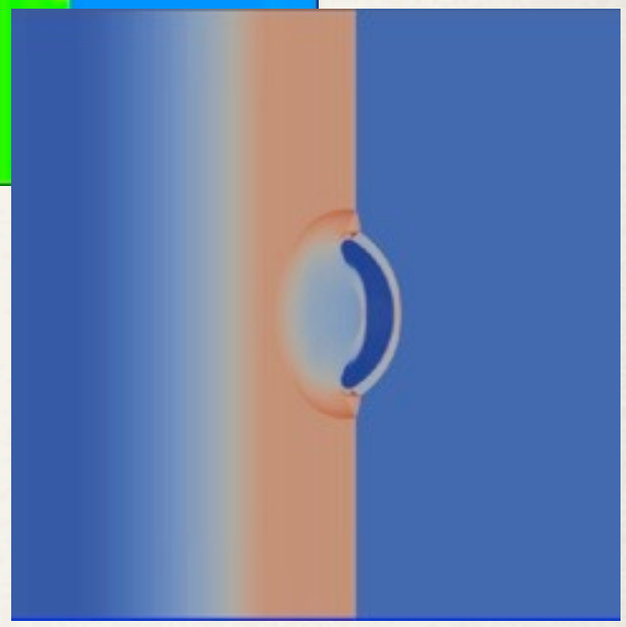
- Alternatives: CLAWPACK, AMRITA
- Single-phase
- 2nd-order spatial discretization

Present solver

- Multi-phase
- 5th-order spatial discretization



Chombo



Present

Present: 56 min, 244 MB

Chombo: 91 min, 230 MB

1. CHOMBO: Colella et al., software package for AMR applications, Technical Report(LBNL), 2000
2. CLAWPACK: LeVeque, software, http://www.amath.washington.edu/~claw/, 1997
3. AMRITA: Quirk, An Introduction to Amrita. http://www.amrita-cfd.com, 1997

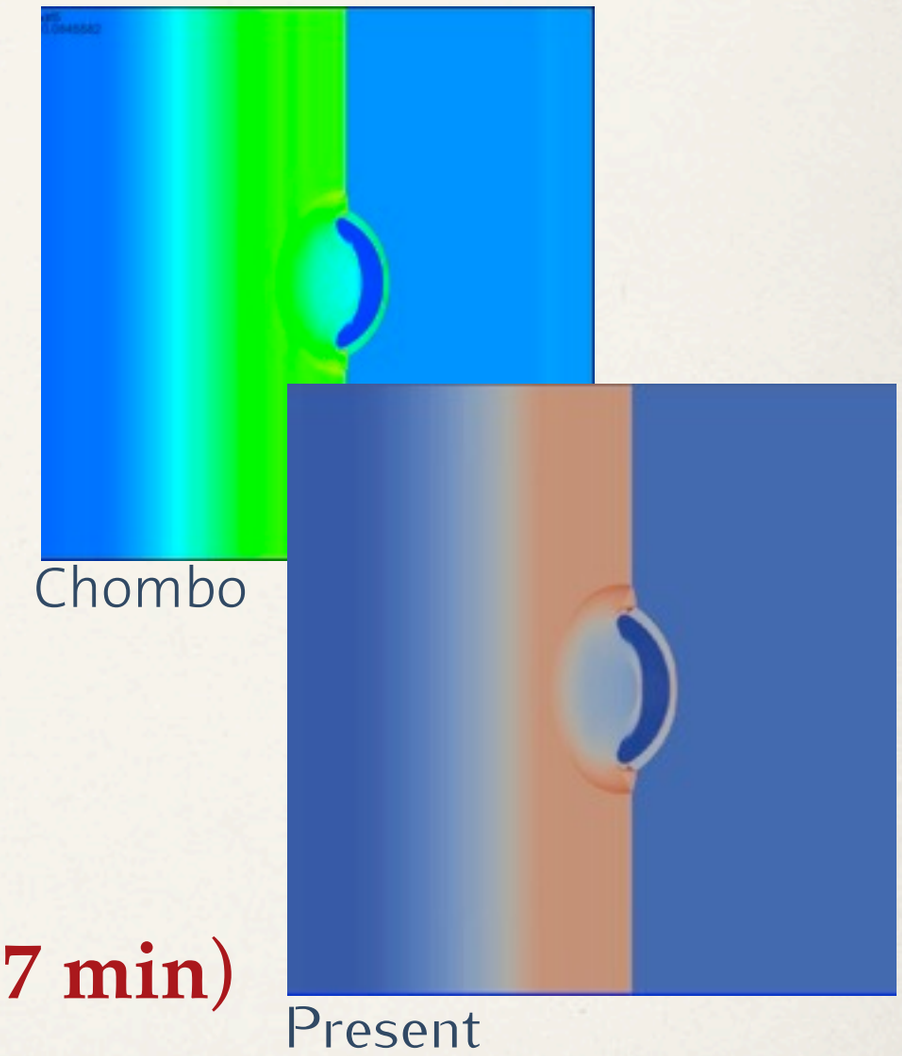
Is Chombo faster?

Chombo

- Alternatives: CLAWPACK, AMRITA
- Single-phase
- 2nd-order spatial discretization

Present solver

- Multi-phase
- 5th-order spatial discretization



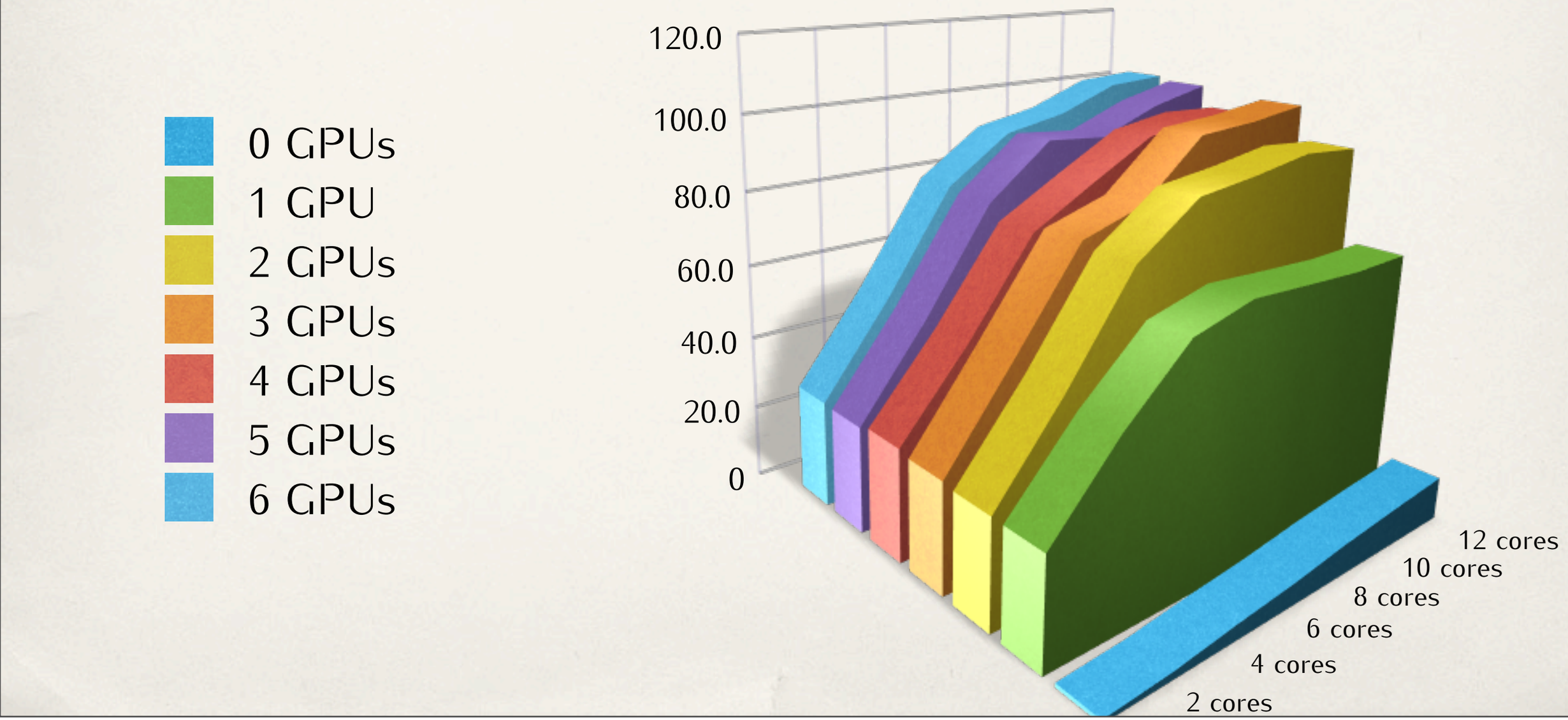
Present: 56 min, 244 MB (+ 1 GPU: 7 min)

Chombo: 91 min, 230 MB

1. CHOMBO: Colella et al., software package for AMR applications, Technical Report(LBNL), 2000
2. CLAWPACK: LeVeque, software, http://www.amath.washington.edu/~claw/, 1997
3. AMRITA: Quirk, An Introduction to Amrita. http://www.amrita-cfd.com, 1997

Performance

Strong scaling versus #GPUs, #CPU cores



Performance

Compared to a space adaptive, single-threaded solver:

Performance

Compared to a space adaptive, single-threaded solver:

- Algorithmic improvements: **24X** faster
- CPU optimization (vectorization): **1.8X** faster
- Task-based parallelism (threading): **12X** (over 16)
- GPUs as accelerators: **3X**

Performance

Compared to a space adaptive, single-threaded solver:

- Algorithmic improvements: **24X** faster
- CPU optimization (vectorization): **1.8X** faster
- Task-based parallelism (threading): **12X** (over 16)
- GPUs as accelerators: **3X**

- Overall speed-up: **3 orders of magnitude (x1500)**

Performance

Conclusions

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile
- **Slow CPU-GPU transfers:**

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile
- **Slow CPU-GPU transfers:**
  - ✓ Can be hidden with computation

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile
- **Slow CPU-GPU transfers:**
  - ✓ Can be hidden with computation
  - X Cannot always be addressed by moving everything on the GPU

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile
- **Slow CPU-GPU transfers:**
  - ✓ Can be hidden with computation
  - X Cannot always be addressed by moving everything on the GPU
- **GPU execution models impose rigid requirements**

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile
- **Slow CPU-GPU transfers:**
  - ✓ Can be hidden with computation
  - X Cannot always be addressed by moving everything on the GPU
- **GPU execution models impose rigid requirements**
  - X Irregular codes do not directly map well on GPU

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile
- **Slow CPU-GPU transfers:**
  - ✓ Can be hidden with computation
  - X Cannot always be addressed by moving everything on the GPU
- **GPU execution models impose rigid requirements**
  - X Irregular codes do not directly map well on GPU
  - ✓ Relaxation/regularization dramatically improves performance

Conclusions

- **GPUs are offering a performance gain of (no more than) 10X:**
  - ✓ 3-7 X (maximum) in terms of GFLOP/s
  - ✓ 2-7 X (maximum) in terms of GByte/s
  - X Even if high, performance gains are fragile
- **Slow CPU-GPU transfers:**
  - ✓ Can be hidden with computation
  - X Cannot always be addressed by moving everything on the GPU
- **GPU execution models impose rigid requirements**
  - X Irregular codes do not directly map well on GPU
  - ✓ Relaxation/regularization dramatically improves performance
  - ✓ Use heterogenous computing instead