Scaling a Reverse Time Migration Algorithm on the TSUBAME 2.0 Supercomputer

Andreas Schäfer

Friedrich-Alexander-Universität Erlangen-Nürnberg

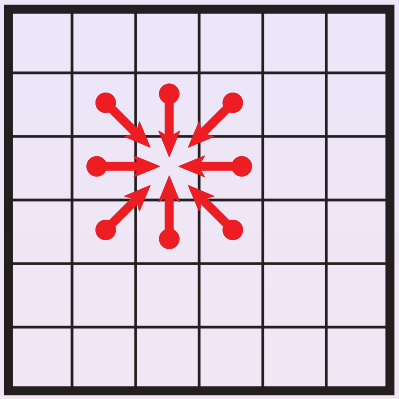
Aoki Laboratory Meeting, 2012.08.01

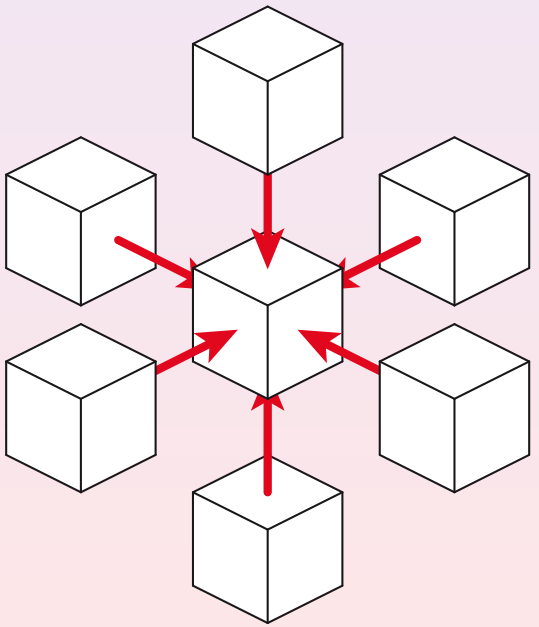
Outline

1. Introduction

What are Stencil Codes?

- time- and space-discrete simulations
- simulation space: regular 2D/3D grid, consists of *cells*
- challenges:
  - low arithmetic intensity
  - heterogeneous supercomputers
  - computations are tightly coupled





LibGeoDecomp: An Overview

- similar to Maruyama-san's *Physis*
- goal: reusable, efficient solution
- simple to use
  1. reroute access to neighbor cells
  2. replace space/time loop with library call
- implemented as C++ template library
- user supplies model
- library contains parallelization
  - plugins for different architectures

LibGeoDecomp: An Overview

- similar to Maruyama-san's *Physis*
- goal: reusable, efficient solution
- simple to use
  1. reroute access to neighbor cells
  2. replace space/time loop with library call
- implemented as C++ template library
- user supplies model
- library contains parallelization
  - plugins for different architectures

Applications Built with LibGeoDecomp

- simulation of dendritic growth
  - regular grid and
  - meshless
- simulation of granular gases

LibGeoDecomp API: Jacobi Iteration

```
#include <libgeodecomp.h>
using namespace LibGeoDecomp;

class Cell {
public:
    static inline unsigned nanoSteps() { return 1; }

    Cell(const double& _value = 0.0) : value(_value) {}

    void update(CoordMap<Cell>& neighbors, unsigned&) {
        value = (neighbors[Coord(0, -1)] +
                 neighbors[Coord(-1, 0)] +
                 neighbors[Coord( 1, 0)] +
                 neighbors[Coord( 0, 1)]) * 0.2;
    }

    double value;
};
```

LibGeoDecomp API: Jacobi Iteration (cont.)

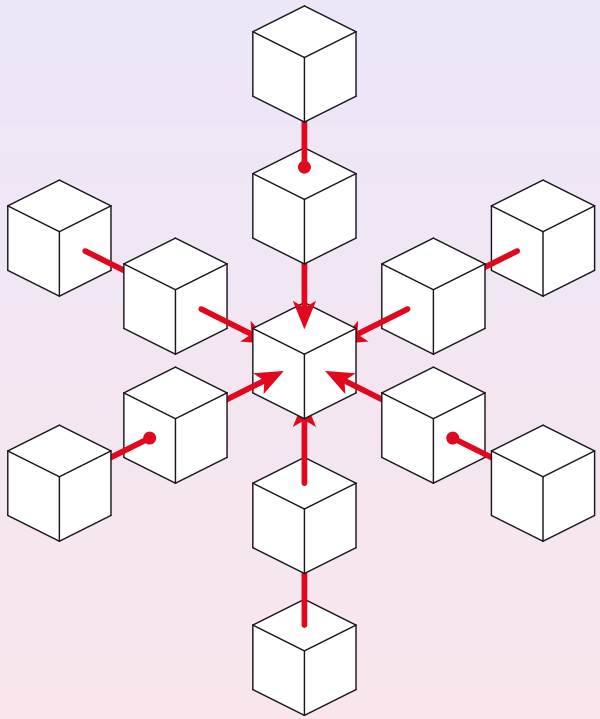
```
class Cell { ... };
class CellInit : public SimpleInitializer<Cell> { ... };

int main(int argc, char *argv[])
{
    MPI::Init(argc, argv);
    Simulator<Cell> *sim =
        LibGeoDecomp::getSimulator<Cell>(new CellInit());
    sim->run();
    MPI::Finalize();
    return 0;
}
```

2. Implementation of Reverse Time Migration

Reverse Time Migration (RTM)

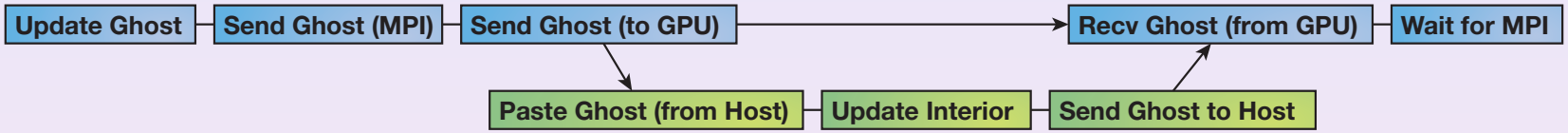
- kernel used in seismic imaging
- convolution of cell with 12 neighbors
- double precision
- relatively simple to implement
- good for comparison
- memory bound
  - hard to achieve peak performance



Implementation on TSUBAME 2.0

- heavily modified LibGeoDecomp
  - hack to support wide stencil radius
  - hybrid multi-GPU code not ready for release
- full 3D domain decomposition
  - recursive bisection
  - block width affects GPU performance
- hybrid code
  - CPU handles ghost zones
  - GPU handles interior
  - shortens latency critical data path

Hybrid CPU/GPU Code



- good:
  - communication & calculation overlap
  - short data paths
  - memory locality
- bad: CPU and GPU calculation don't overlap
- possible solution: additional intro node ghost zones (not implemented yet)

GPU Code

- GPU code updates interior grid fraction
- written in CUDA
- wavefront, travels in Z direction
- register blocking
- L1 cache to cover redundant reads
- 60 registers per thread

CPU Code

- CPU handles ghost zones
- lots of L1 cache misses on X boundaries
  CPU handles this better than GPU
- moves in X direction
- address calculation
  - fixed block sizes DIM_X, DIM_Y, DIM_Z
  - access to fixed relative coordinates
    REL_X, REL_Y, REL_Z $\in \{-2, -1, 0, 1, 2\}$
  - line pointer **double** *cursor
  - **double** *addr =
    cursor + REL_Z * DIM_X * DIM_Y + REL_Y * DIM_X + REL_X
  - on X86: address generation units yield 1 address per clock

CPU Code (cont.)

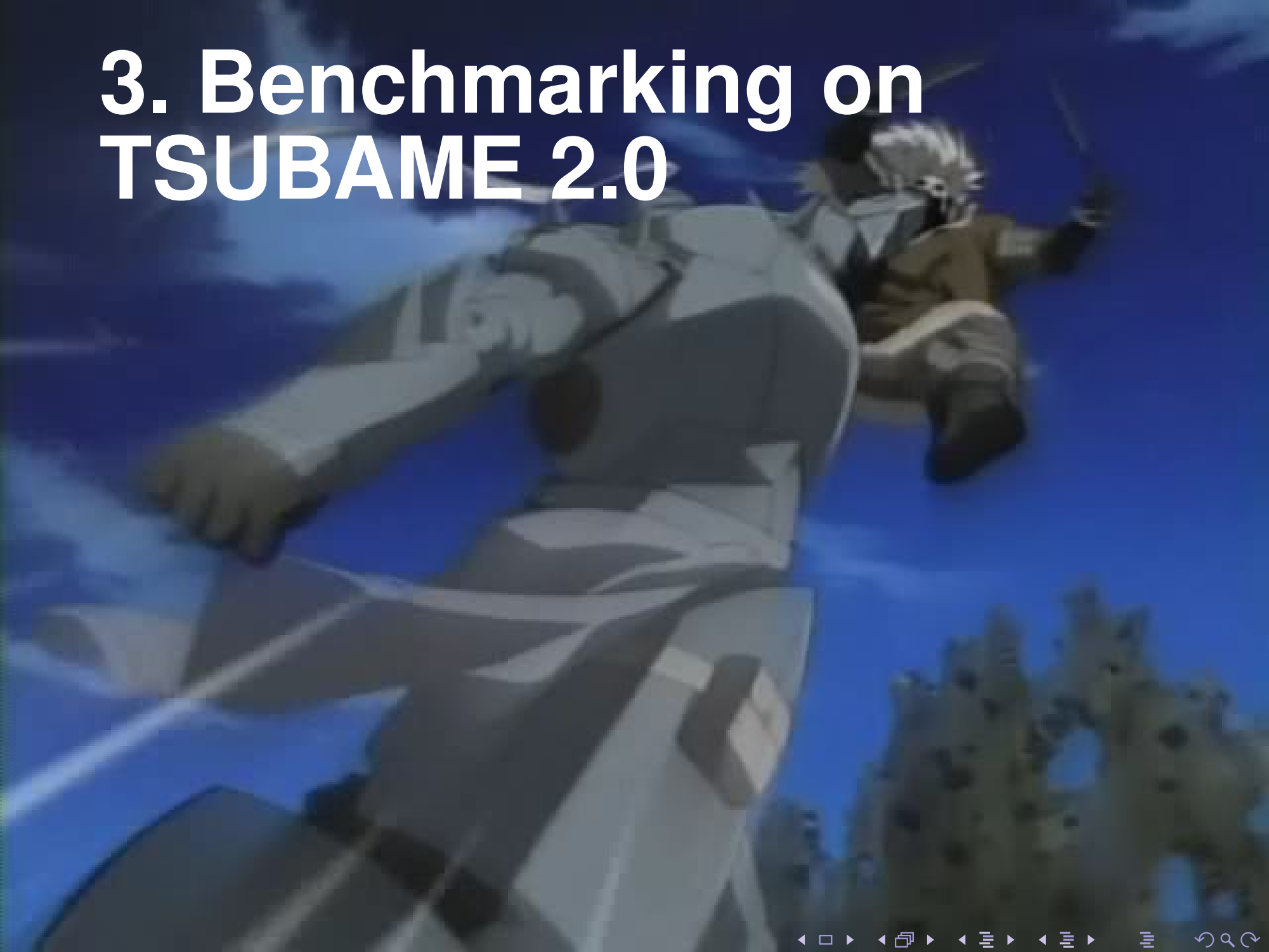
- SSE, 8x loop unrolled
- operates on whole cache lines (64 B)
- use `_mm_shuffle_pd()()` to avoid unaligned loads
- excerpt:

```
__m128d tmp2 = _mm_load_pd(source + i +  4);
__m128d tmp3 = _mm_load_pd(source + i +  6);
__m128d tmp4 = _mm_load_pd(source + i +  8);
__m128d tmp5 = _mm_load_pd(source + i + 10);

__m128d buf1 = _mm_shuffle_pd(tmp1, tmp2, (1 << 0) | (0 << 2));
__m128d buf2 = _mm_shuffle_pd(tmp2, tmp3, (1 << 0) | (0 << 2));
__m128d buf3 = _mm_shuffle_pd(tmp3, tmp4, (1 << 0) | (0 << 2));
__m128d buf4 = _mm_shuffle_pd(tmp4, tmp5, (1 << 0) | (0 << 2));

tmp1 = _mm_add_pd(buf0, tmp1);
tmp2 = _mm_add_pd(buf1, tmp2);
tmp3 = _mm_add_pd(buf2, tmp3);
tmp4 = _mm_add_pd(buf3, tmp4);
...
```

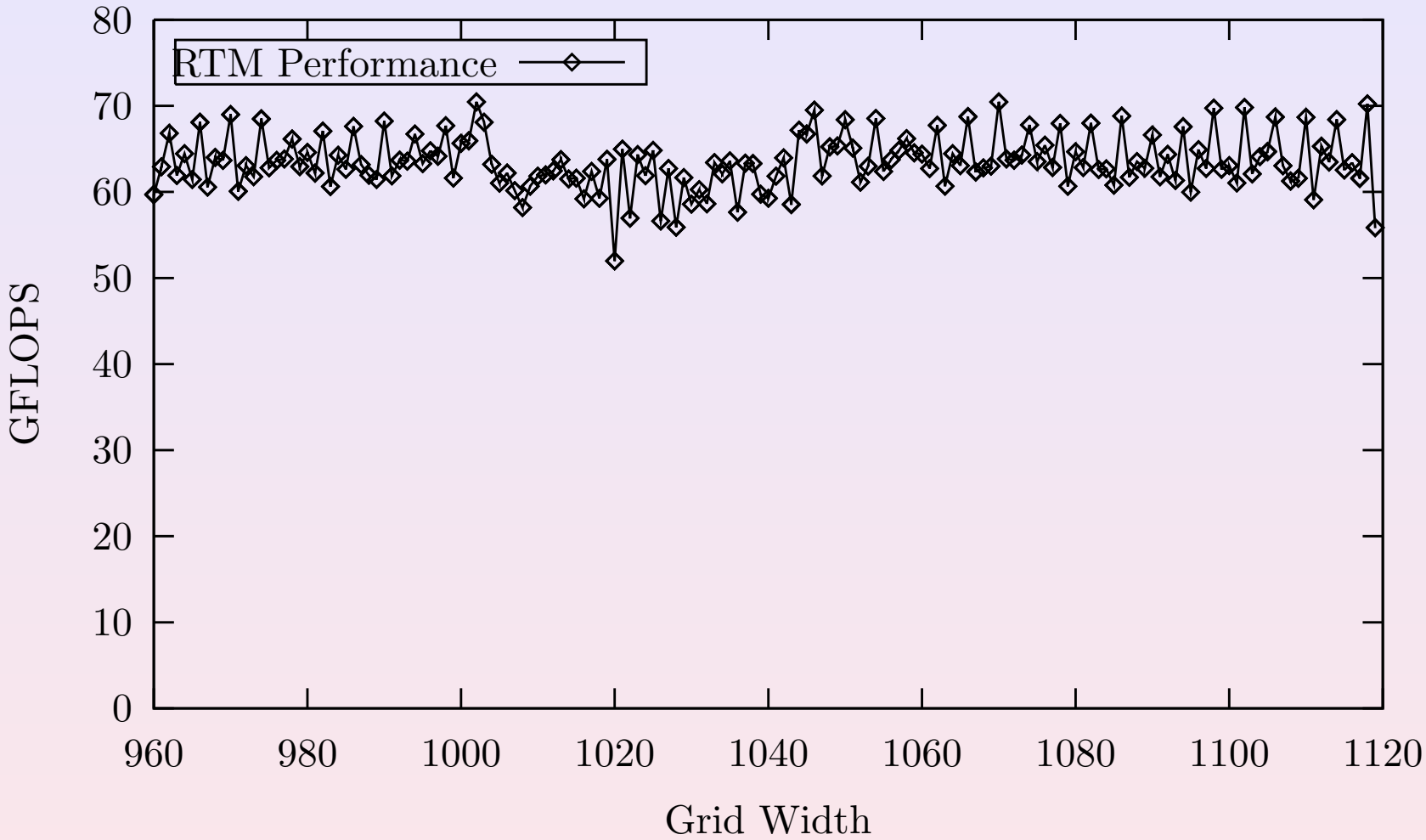
3. Benchmarking on TSUBAME 2.0



Tools

- hybrid compilation
  - Intel C++ compiler `icpc` 11.1 for host code
  - NVIDIA CUDA Toolkit 4.1 `nvcc` for GPU code
- Boost 1.50
- Open MPI 1.6
- (CMake 2.8.8)

GPU Grid Size Tuning

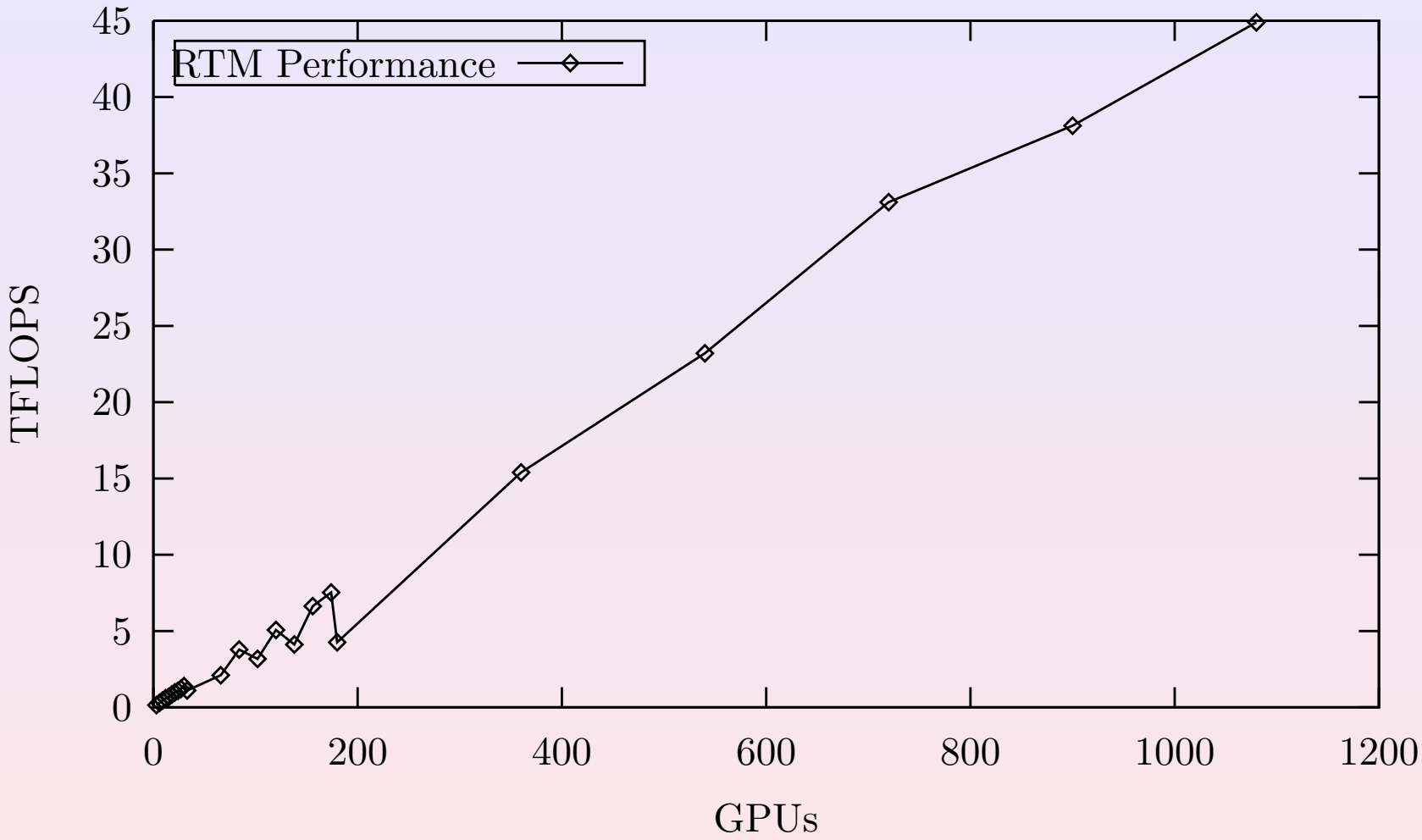


- measurement taken on `faui36i` @ Erlangen
- Tesla C2050, ECC off

GPU Grid Size Tuning

- wobbling caused by odd grid widths
  (even width better)
- best widths avoid powers of two
  (cache thrashing)
- significant boost compared to GPU in Tsubame
  but reduced reliability (not suited for production runs)
- single GPU performance still disappointing

Weak Scaling of Reverse Time Migration



- $\text{grid\_size} = (1070, 256, 512) \cdot \text{GPUs}^{1/3}$

Weak Scaling of Reverse Time Migration (cont.)

- peaks at approx. 45 TFLOPS
- packed process placement reduces inter-node communication
  - rank 0, 1, 2 on node A,
  - rank 3, 4, 5 on node B,
  - etc.
- single GPU performance lower due to ECC
- good scalability on up to 1080 NVIDIA Tesla M2050 GPUs

Summary and Outlook

Summary and Outlook

- version 0.1.0 available at
  `http://www.libgeodecomp.org/download.html`
- multi GPU code not ready yet for release
- very promising scalability
- goals for future releases
  - incorporate fixed block size code to fast addressing
  - improve overlap of GPU/CPU calculation
  - 1 PFLOPS?