Ingredients for Ultra-Scalable Simulation Codes

Andreas Schäfer

Friedrich-Alexander-Universität Erlangen-Nürnberg

Facing the Multicore Challenge, 19.09.2012

Outline

Challenges...

for the Software Engineer

Challenges for the Software Engineer

1. increased model complexity
2. increased system complexity
   - heterogeneity
   - rising core counts
3. memory gap
4. network latency vs. system scale
5. vector units vs. object-oriented programming

Achieving peak performance gets harder and harder.

Ingredients...

Which Every Code Needs

Ingredients Which (Almost) Every Code Needs

- 1. Object-Orientation
  - the #1 countermeasure to software complexity
  - slow?
- 2. Hierarchical Parallelization
  - allows you to cope with system complexity
  - emphasis on efficient domain decomposition
- 3. Cache Blocking
  - reduces memory gap
  - reschedule calculations to move data closer to ALUs

Ingredients Which (Almost) Every Code Needs (cont.)

- 4. Overlapping Calculation/Communication
  - hides network latency
  - respect network topology during process placement
- 5. Struct of Arrays
  - enables vectorization
  - tricky: object-oriented facade

Who expects the domain scientist to do all the work?

Ready Meals

Why not try an existing tool?

Typical HPC Paper 1

- solved scientific problem A
- on machine X
- fabulous performance

Typical HPC Paper 2

- solved scientific problem A'
- on machine X
- fabulous performance

Typical HPC Paper 3

- solved scientific problem A
- on machines X and Y
- fabulous performance

Problematic HPC Paper

- solved class of scientific problems
- on class of machines
- almost fabulous performance

Software Engineering for HPC – Is it Science?

[I]s more a nice work of engineering than a scientific contribution.

(Anonymous reviewer, Vecpar 20012)

[T]he work is presented entirely as a software development project, not as research.

(Anonymous reviewer, SC'12)

Libraries for Computer Simulations

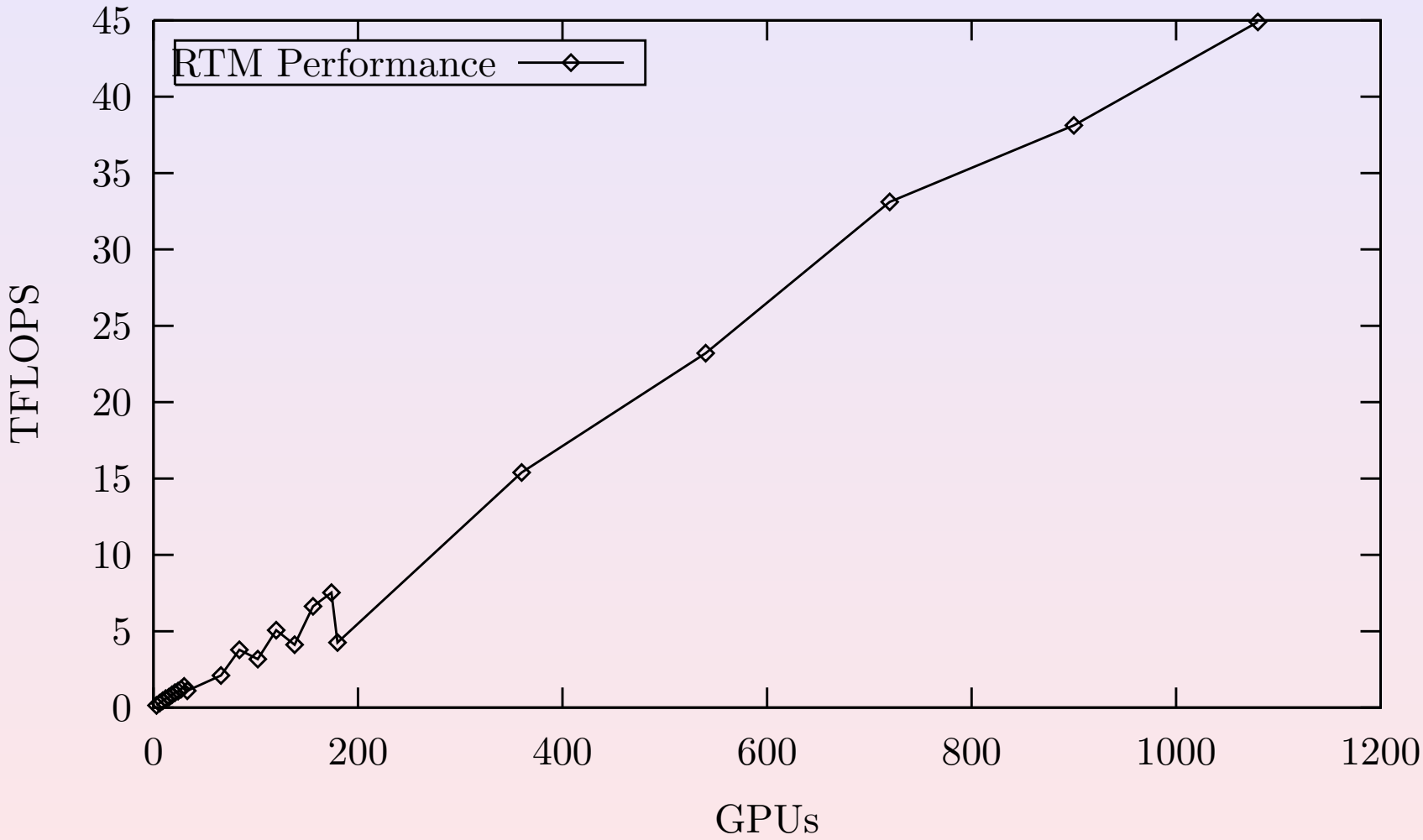
Why cook your own code if you can have a gourmet dinner?

- problem solving environments
  - Cactus `http://cactuscode.org/`
  - waLBerla `http://www10.informatik.uni-erlangen.de/en/Research/Projects/walberla/`
- generic libraries:
  - patus `http://code.google.com/p/patus/`
    - domain specific language (DSL)
    - CPU + GPU
    - single node
    - auto-tuning
  - Physis `https://github.com/naoyam/physis`
    - DSL (C subset)
    - CPU + GPU
    - multi node (MPI)

LibGeoDecomp

- disclaimer: my own project
- Library for Geometric Decomposition Codes (i.e. simulations)
- `http://www.libgeodecomp.org/`
- C++ API (fully object-oriented)
- (CPU + GPU)
- implements most features seen above
  - plus some more: parallel IO, application level checkpoint/restart, n-body simulations...
- cache blocking and structure of arrays on the way (Dec. 2012)

LibGeoDecomp Benchmark: Reverse Time Migration



LibGeoDecomp Benchmark: Reverse Time Migration

- hybrid parallelization
  - halos on CPU
  - interior on GPU
  - minimizes communication latency
- Tsubame 2.0
- 1080 GPUs
- 45 TFLOPS
- 90 % efficiency (weak scaling)

Summary: Use a tool!