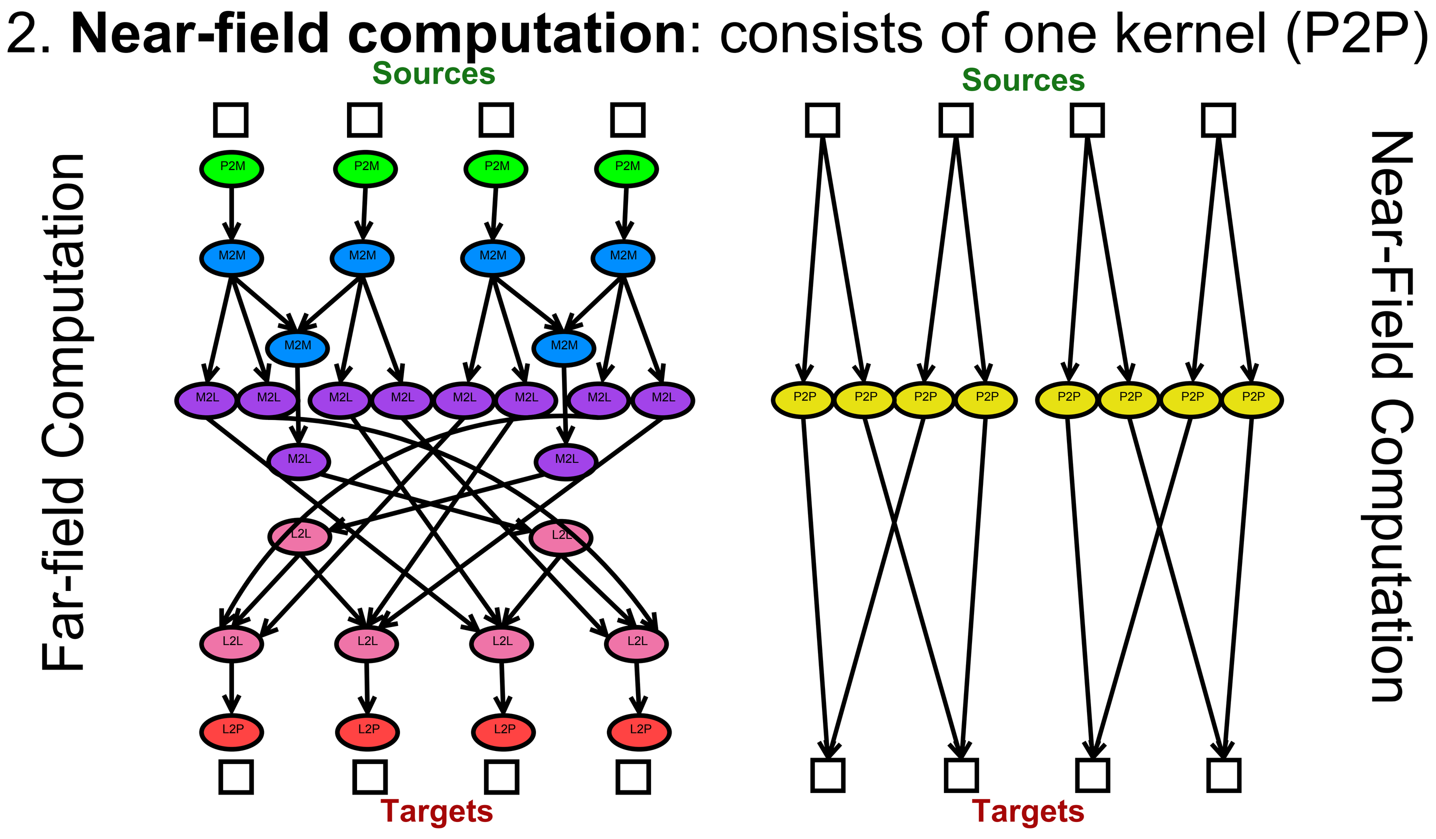
System Software

Programming Models and Runtimes

Fast Multipole Method

The Fast Multipole Method is a hierarchical method with many applications: N-Body simulation, turbulence simulations, PDE solvers, etc.. It features two parallel flows:

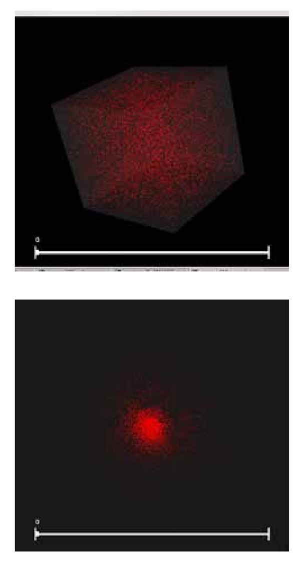
1. **Far-field computation**: consists of 5 kernels (P2M, M2M, M2L, L2L and L2P)
2. **Near-field computation**: consists of one kernel (P2P)

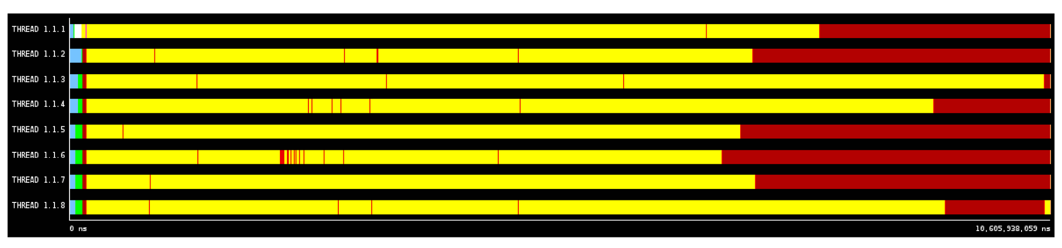


Complex scheduling problem. High variability in task runtimes

Many parameters affect the runtime of the tasks:

- Input distribution (Cube, Plummer, Sphere, etc)
- Device (CPU, GPU, etc)
- Memory locality
- Task granularity (interaction-level, cell-level, ...)
- Number of particles per cell (q)



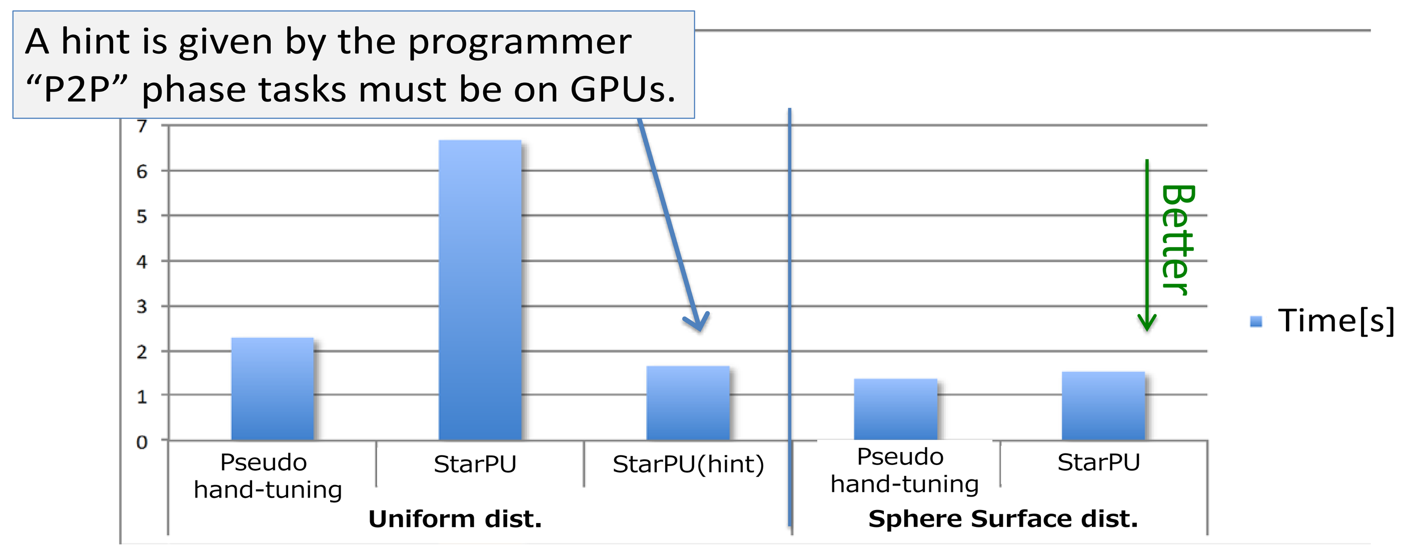


Load imbalance due to variability in task sizes while simulating a Plummer-like globular star cluster (yellow=running, red=waiting)

Scheduling on Heterogeneous Architectures

Employing *StarPU* to schedule CPU/GPU tasks

→ Towards robust dynamic load balancing



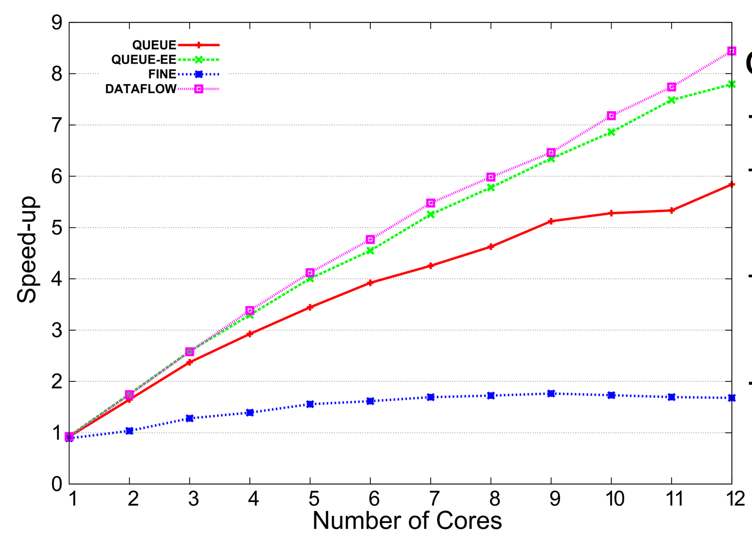
- Remaining Challenges:

Smarter scheduling / better programming models

Scheduling on Multicore Architectures

OmpSs scalability results on a dual Xeon X5670 (12 cores)

Based on ExaFMM code `(http://www.bu.edu/exafmm/)`



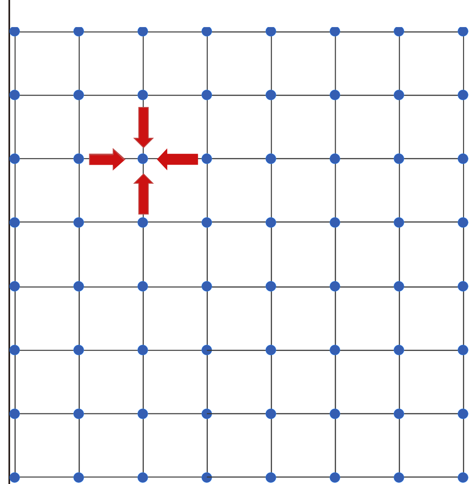
q = 100, Plummer Distribution, 80000 bodies

- `FINE` operates at interaction level
- `QUEUE`, `QUEUE-EE` & `DATAFLOW` operate at cell level
- `QUEUE-EE` & `DATAFLOW` start execution during construction of the interaction lists
- `DATAFLOW` pipelines `M2L`→`L2L`→`L2P`

Multicore scalability depends heavily on:

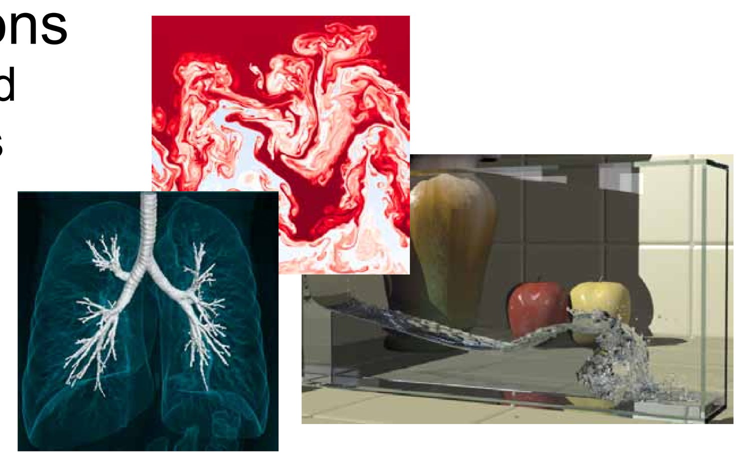
- task granularity
- executing tasks early and removing global synchronization points

Physis: An Implicitly Parallel Framework for Stencil Computations



Stencil Computations

- Iteratively updates grid points using neighbors
- A fundamental computation pattern in scientific simulations



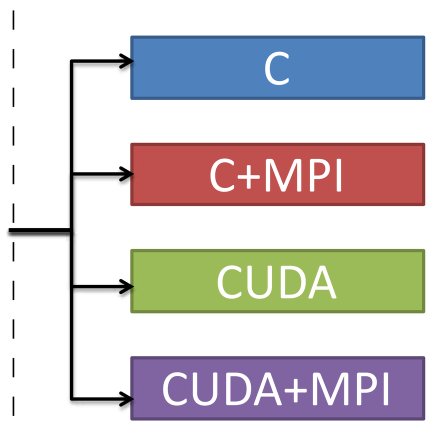
Physis DSL

- Dense grid types
- Intrinsics for manipulating grids
- Functions expressing stencils

DSL Translator

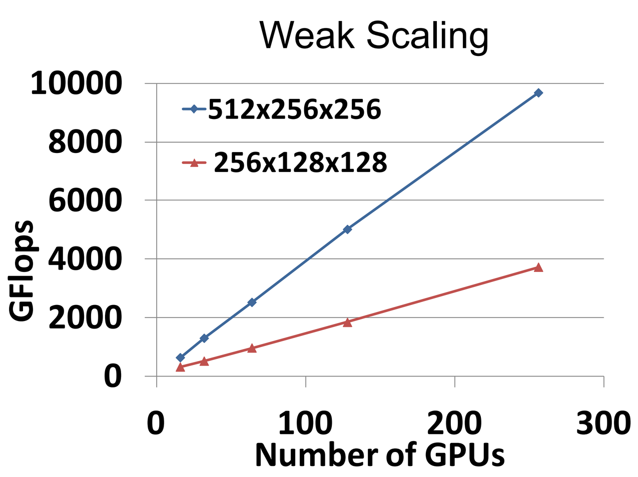
- Using the ROSE framework
- Domain-specific optimizations
- Automatic parallelization

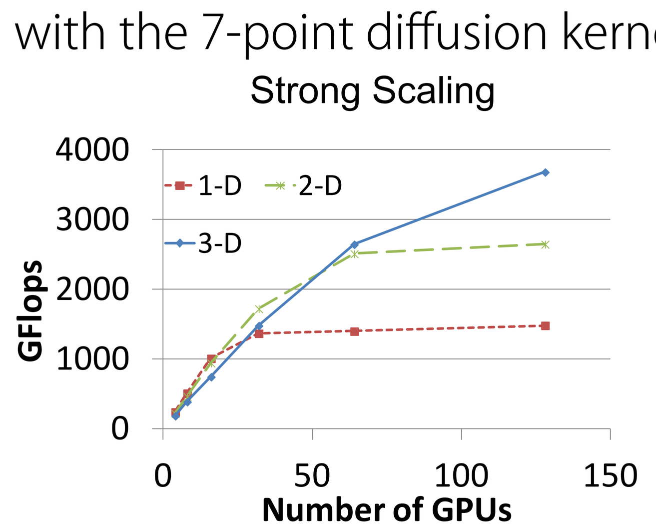
```
void diffusion(const int x, const int y, const int z,
               PSGrid3DFloat g1, PSGrid3DFloat g2,
               float t) {
  float v = PSGridGet(g1,x,y,z)
    +PSGridGet(g1,x-1,y,z)+PSGridGet(g1,x+1,y,z)
    +PSGridGet(g1,x,y-1,z)+PSGridGet(g1,x,y+1,z)
    +PSGridGet(g1,x,y,z-1)+PSGridGet(g1,x,y,z+1);
  PSGridEmit(g2,v/7.0*t);
}
```



Performance Results on Tsubame

- Performance evaluation with the 7-point diffusion kernel





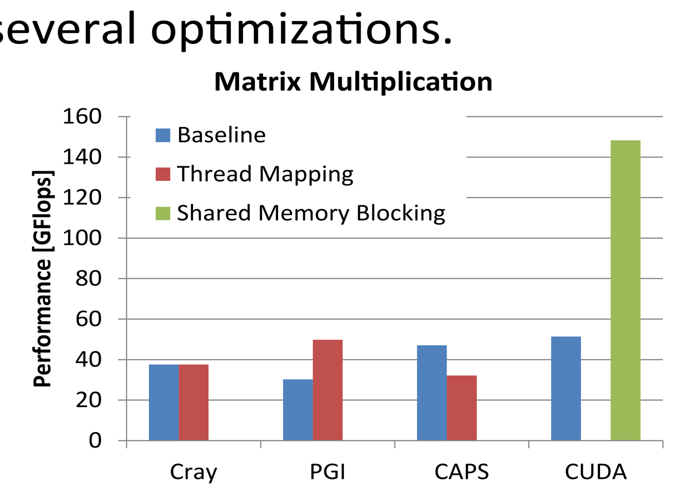
Download at http://github.com/naoyam/physis

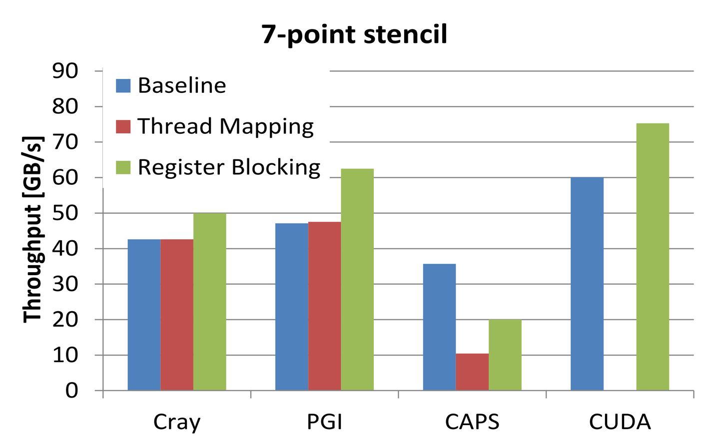
Preliminary Evaluation of OpenACC Performance

OpenACC is a new directive-based programming language for accelerators. We focus on OpenACC performance compared with CUDA.

Micro-Benchmarks

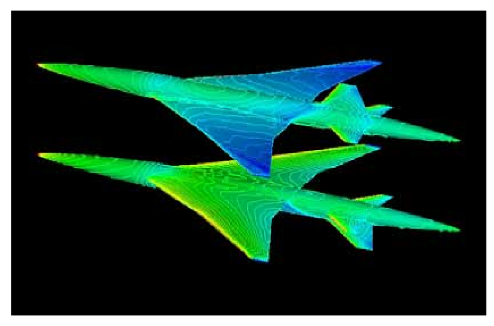
We compared the OpenACC implementations provided by Cray, PGI and CAPS with CUDA by using Matrix Multiplication and a 7-point stencil while applying several optimizations.



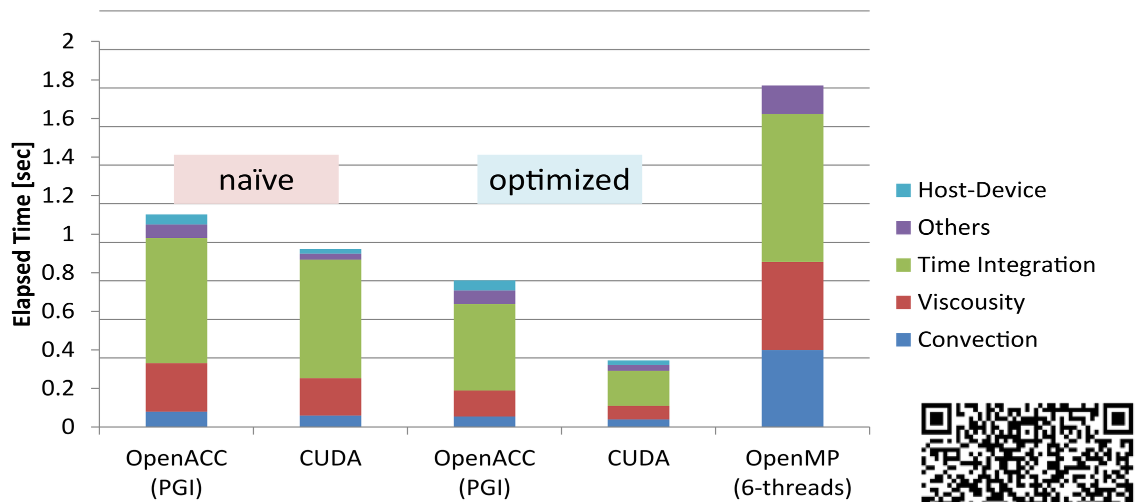


A Real-world CFD Application : UPACS

UPACS (Unified Platform for Aerospace Computational Simulations) is a large scale CFD application developed by the Japan Aerospace Exploration Agency.



We ported this application to OpenACC and CUDA, and applied several optimizations to each implementation. In the naïve implementation, OpenACC achieves 80% of the performance of CUDA. But in the optimized implementation, OpenACC gives only 40% of the performance of CUDA. We observe that some limitations of OpenACC prevent achieving high performance.



http://www.gsic.titech.ac.jp/sc12