Multigrid on Heterogeneous Systems

A Look at Structured Stencil Computations

MPI + OpenMP + OpenCL

Lukas Spies[1], Luke Olson[1], Scott MacLachlan[2], Michael Campbell[1], Daniel Bodony[1], William Gropp[1]

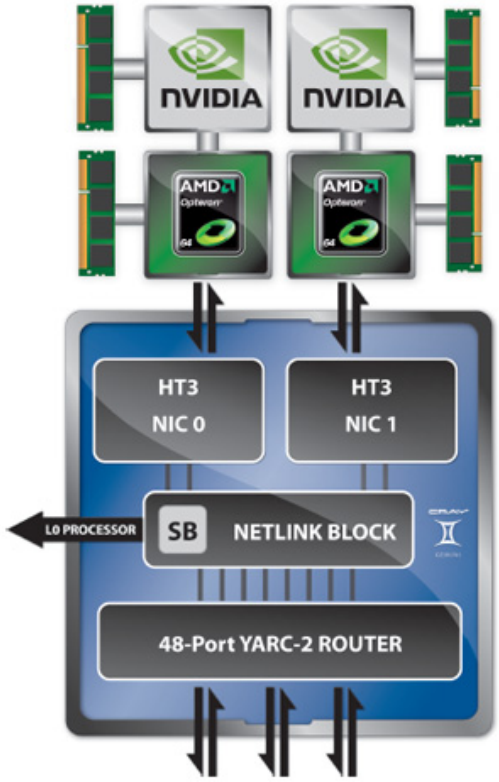
[1] The Center for Exascale Simulation of Plasma-Coupled Combustion (XPACC), University of Illinois at Urbana-Champaign, USA, `http://xpacc.illinois.edu`

[2] Department of Mathematics and Statistics, Memorial University of Newfoundland, Canada

Introduction

Why target heterogeneous systems?

- Combine compute power of both CPUs and GPUs
- Avoid having one sit idle while the other is busy
- Chance for performance increase



Challenges when working with BLAS-1 (basic linear algebra subprograms, level 1 (e.g., "axpy"-type operations))

- BLAS-1 has low computational intensity
- BLAS-1 requires a lot of communication

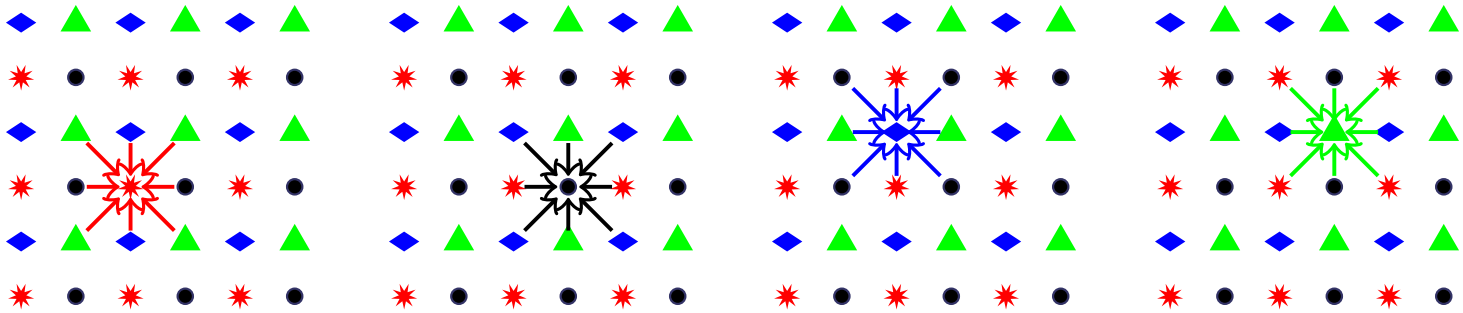
Image source: https://bluewaters.ncsa.illinois.edu/image/image_gallery?img_id=11748

Model problem

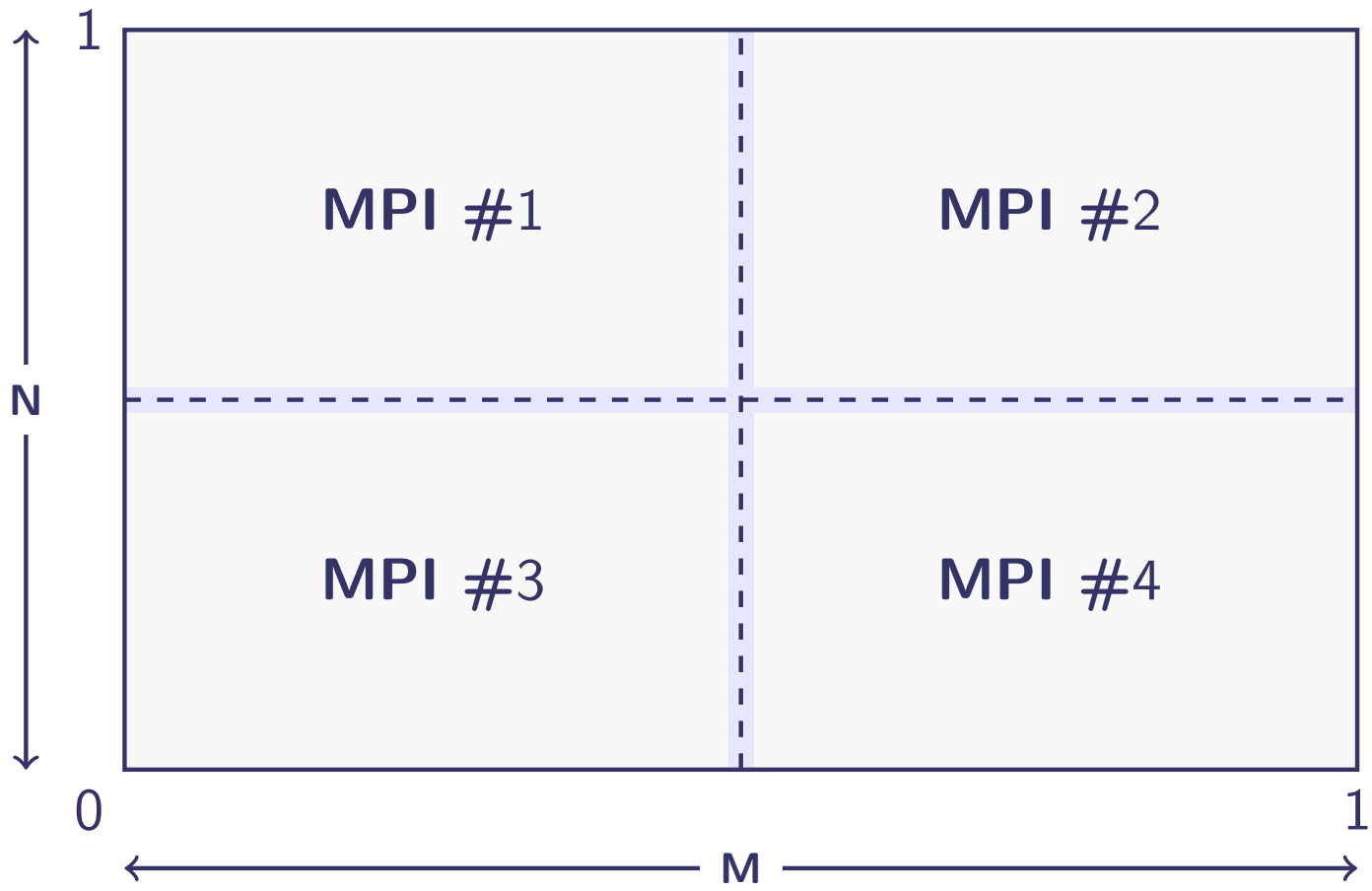
- Model problem: Laplace's equation in two dimensions,

$$\nabla^2 u = 0.$$

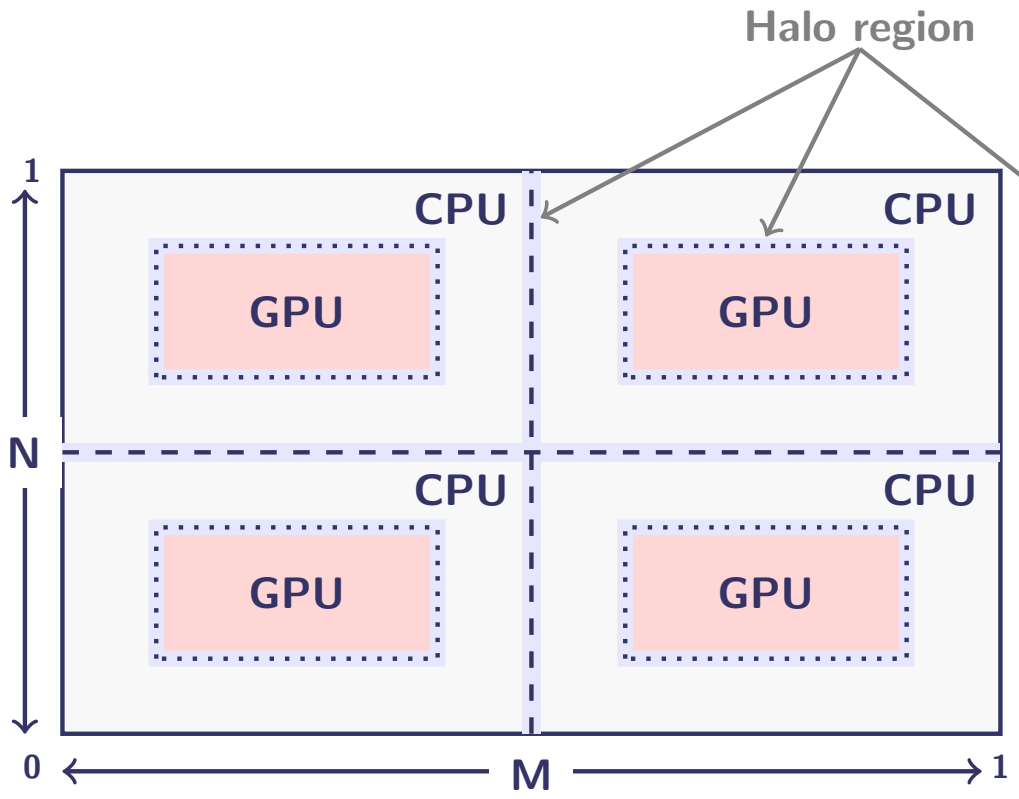
- Dirichlet boundary conditions (set to 0)
- Domain: $[0, 1] \times [0, 1]$
- Four-color Gauss-Seidel: Take weighted average of each point and its 8 surrounding points (Mehrstellen FD stencil)



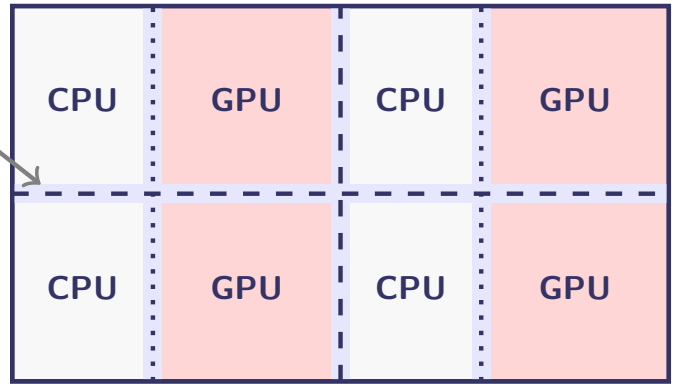
Coarse partitioning

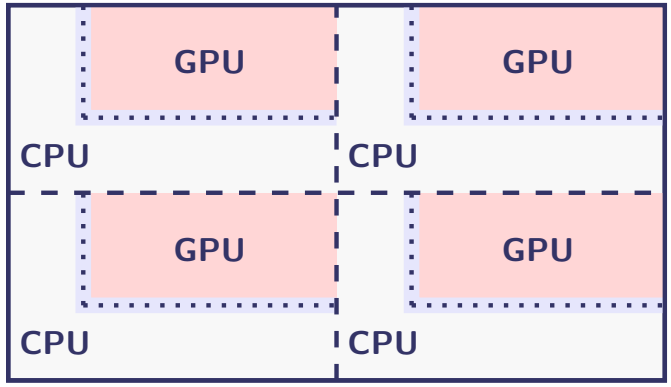


CPU/GPU partitioning designs



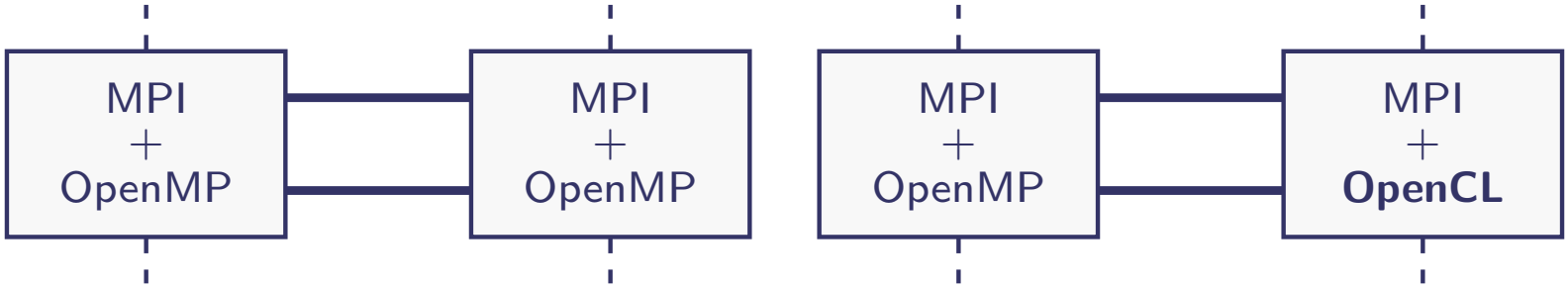
Halo design





L-shape design

Blue Waters

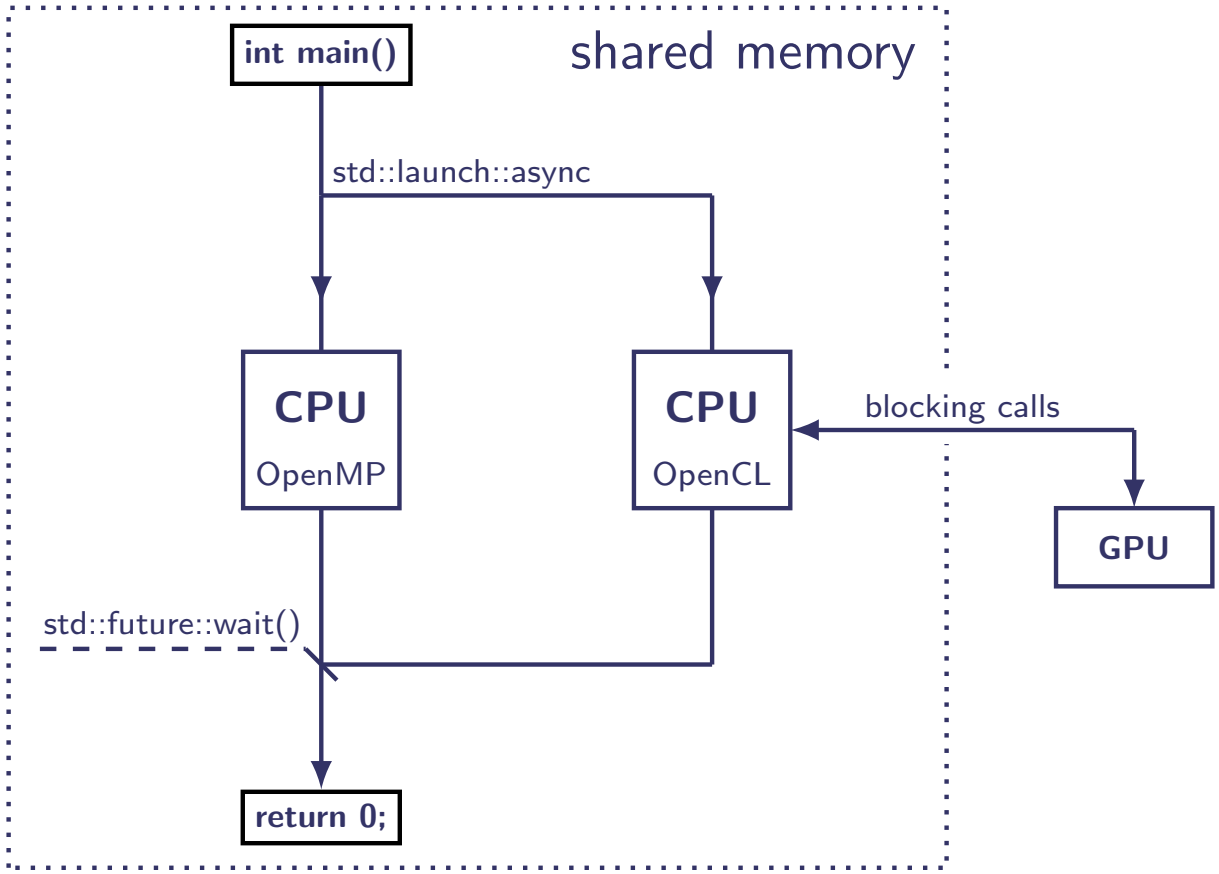


XE6 node:
2 AMD 6276 Interlagos

XK7 node:
1 AMD 6276 Interlagos
1 NVIDIA K20X

- each Interlagos: 2.3 GHz peak performance, 4 memory channels, 32 GB physical memory
- each K20X: 1.31 TFLOPS peak performance, 6 GB physical memory

Threading for hybrid approach



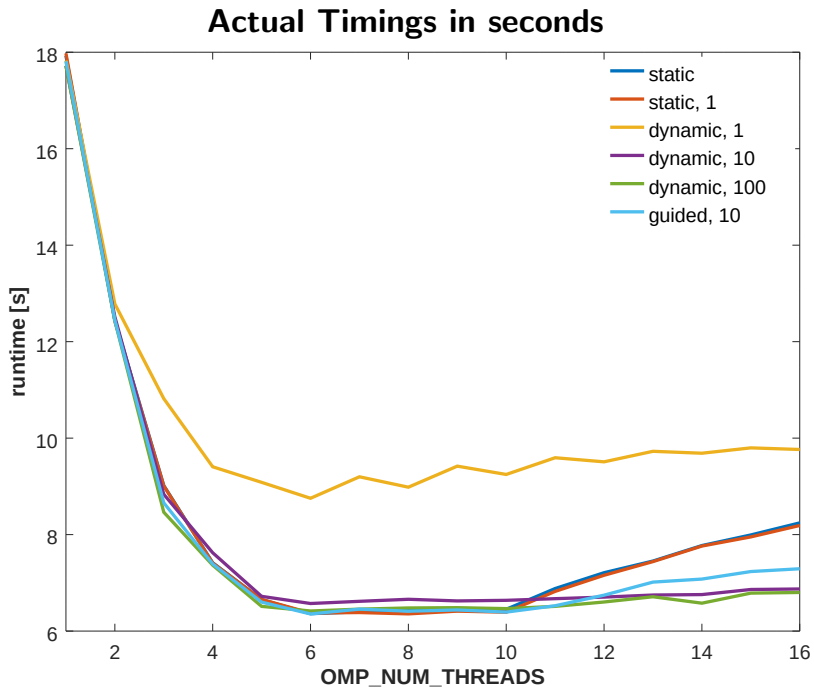
- Using C++11 *future* class for asynchronous threading

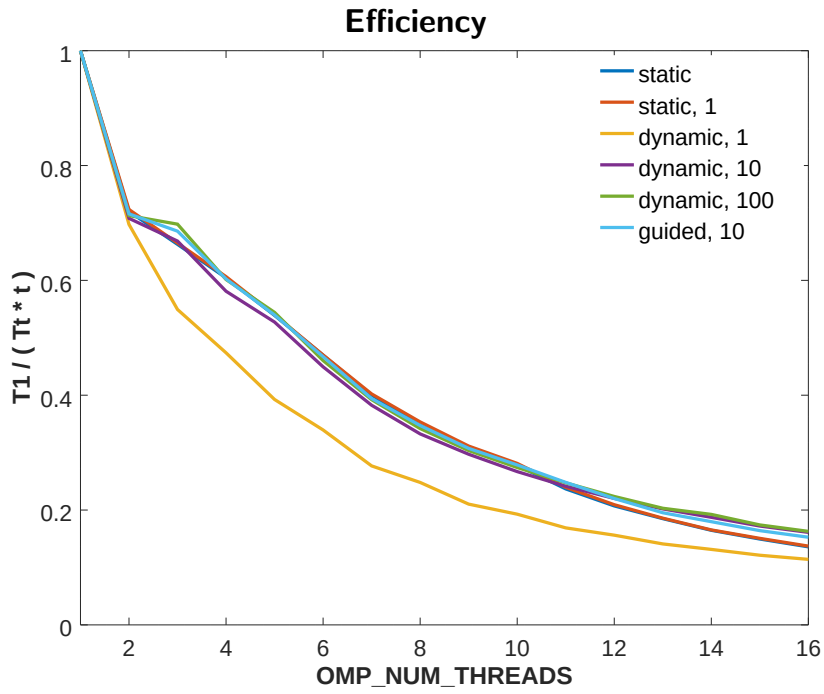
Numerical experiments

1. Test various OpenMP thread numbers and OpenMP schedulers:
   - static: equal chunks for each thread
   - dynamic: assign small chunks to threads, let each thread check back for more work
   - guided: like dynamic, but with decreasing chunk size
2. Test various device-to-host ratios
3. Scaling behavior of hybrid code
4. Analyse roofline model
5. Performance model

Notation: host-to-device ratio $:= \omega$

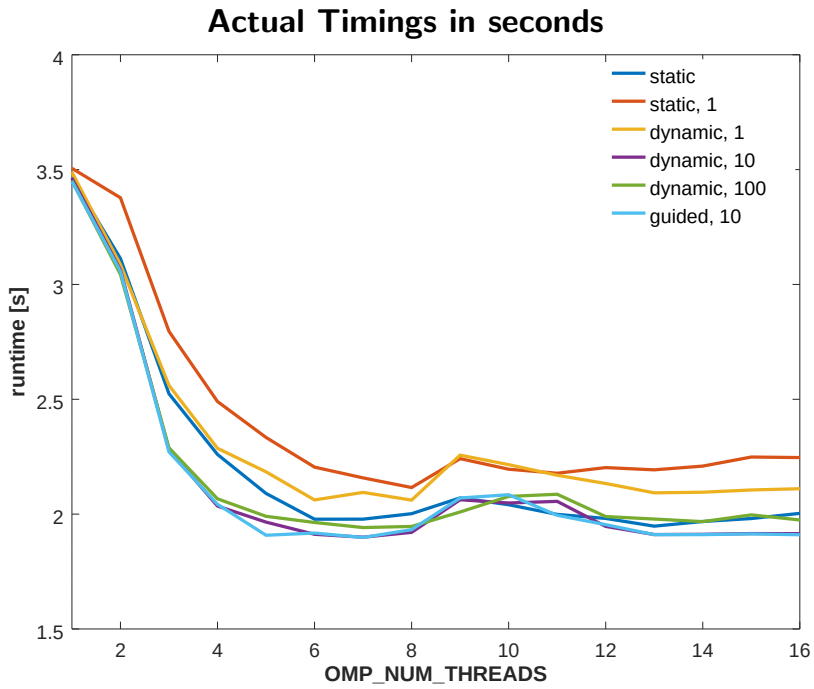
CPU-only benchmark

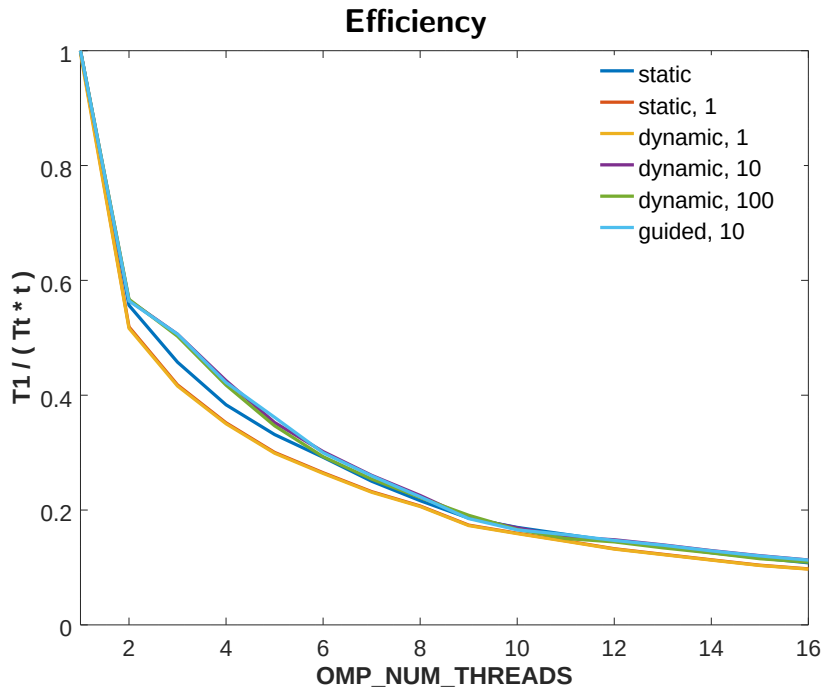




- Varying number of OpenMP threads for various OpenMP schedulers
- $40,000 \times 40,000$ points, 4 MPI ranks, 5 iterations
- Fastest time: 6356.43 ms

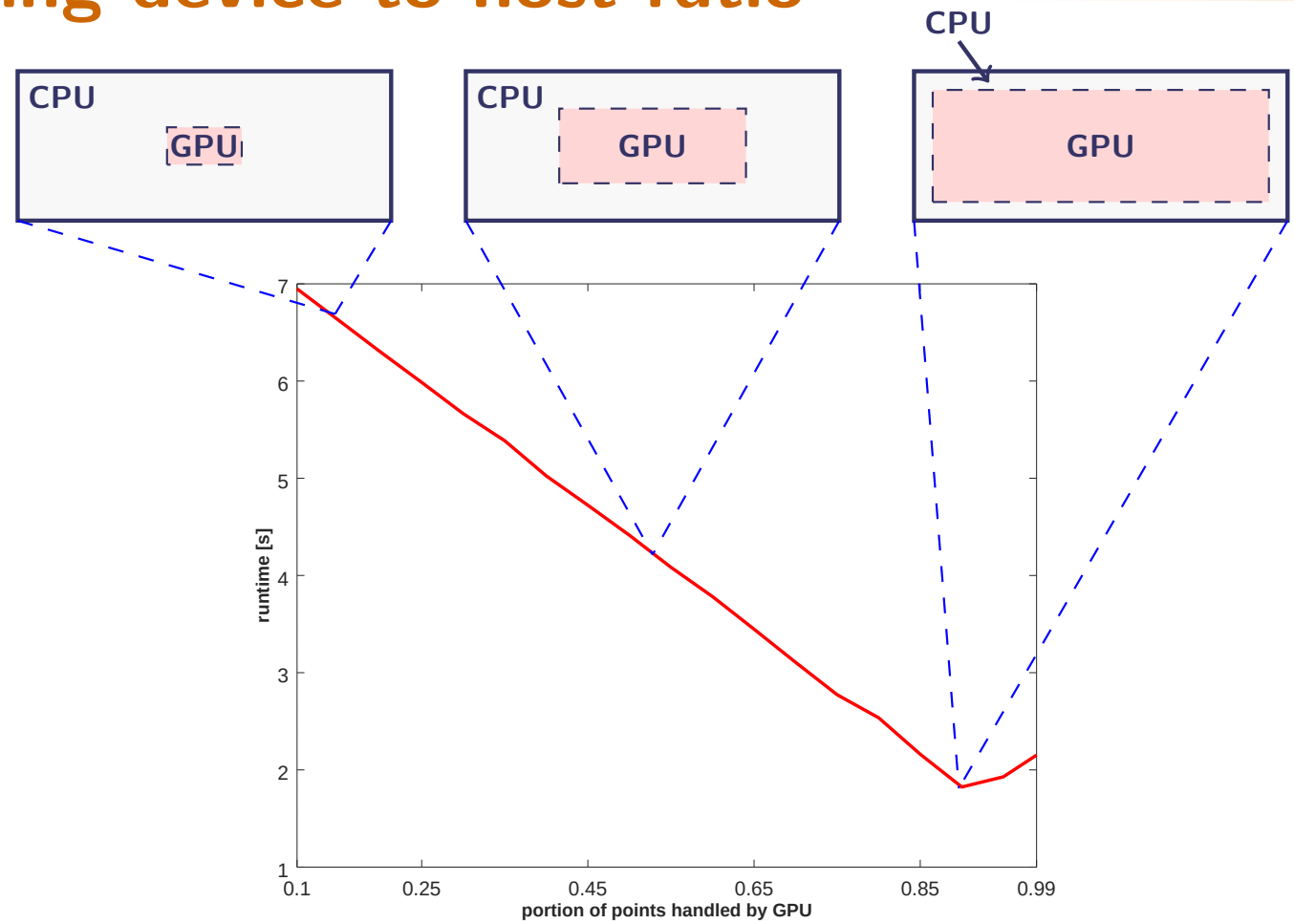
Hybrid OpenMP benchmarks





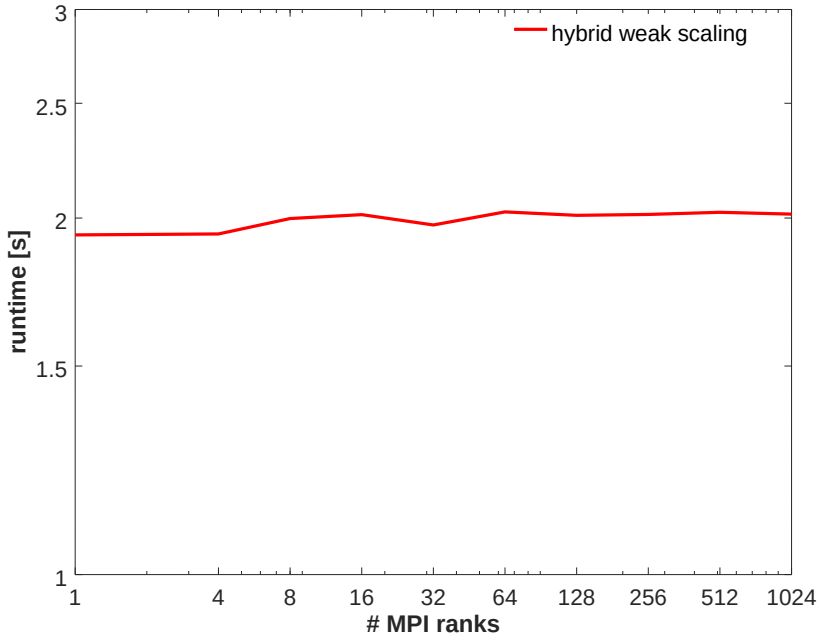
- Varying number of OpenMP threads for various OpenMP schedulers
- $40,000 \times 40,000$ points, 4 MPI ranks, 5 iterations, $\omega = 0.9$
- Fastest time: 1898.2 ms

Varying device-to-host ratio

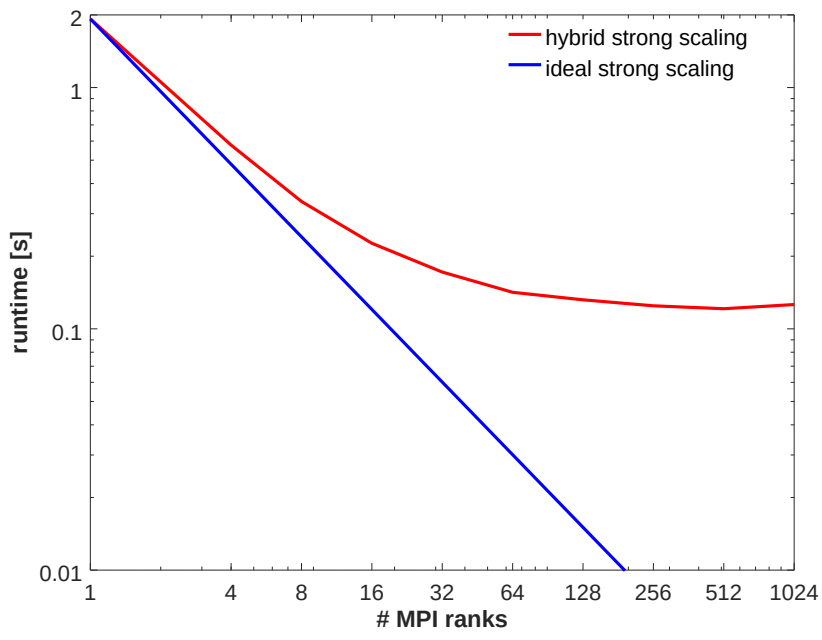


- $40,000 \times 40,000$ points, 4 MPI ranks, 5 iterations, 8 OpenMP threads per rank

Weak and strong scaling



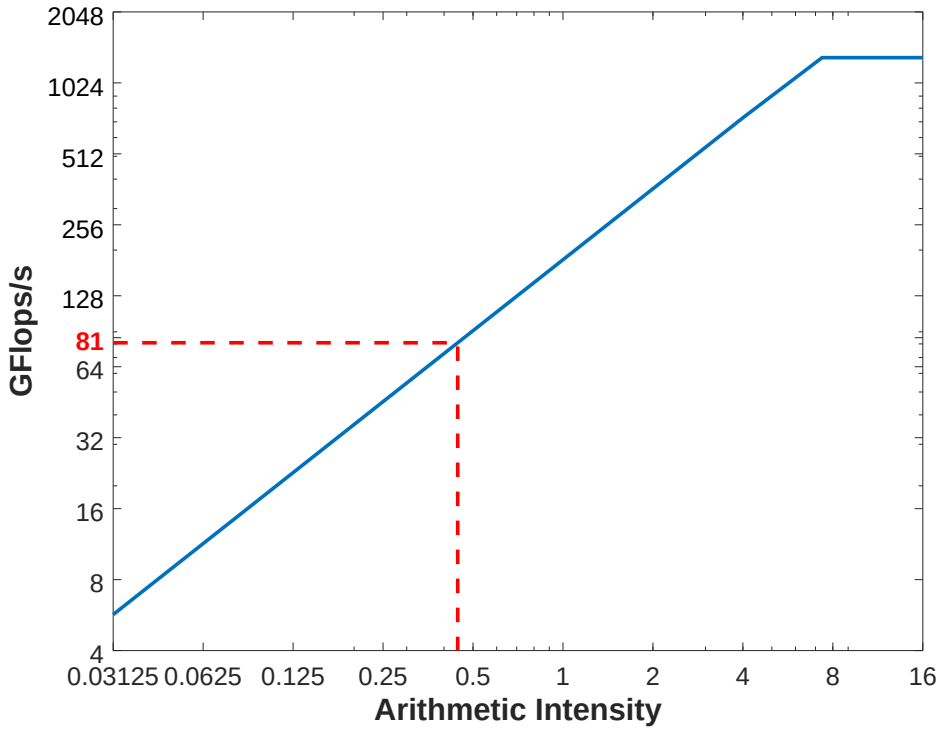
(a) weak scaling



(b) strong scaling

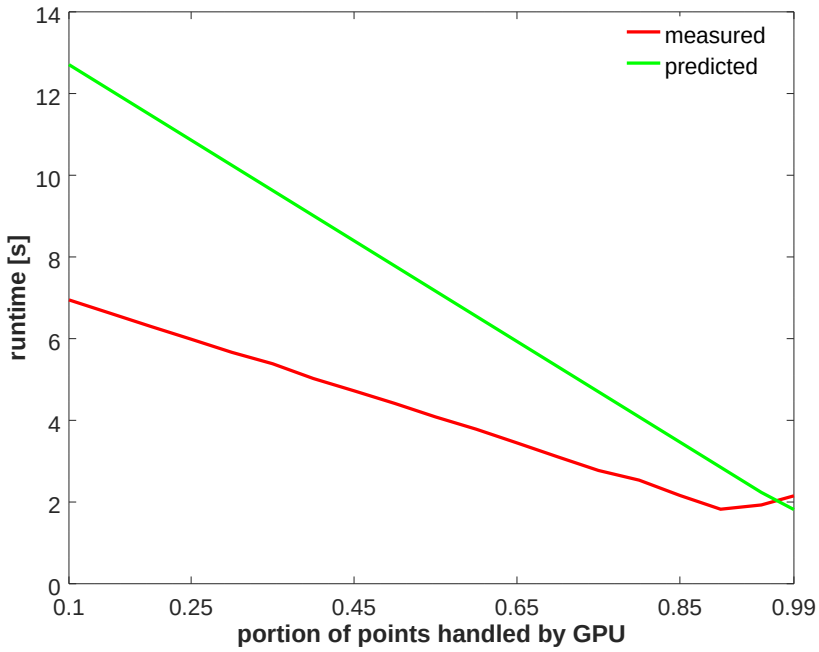
- weak scaling: $20,000 \times 20,000$ points per MPI rank
- strong scaling: $20,000 \times 20,000$ points overall problem size
- both: 5 iterations, $\omega = 0.9$

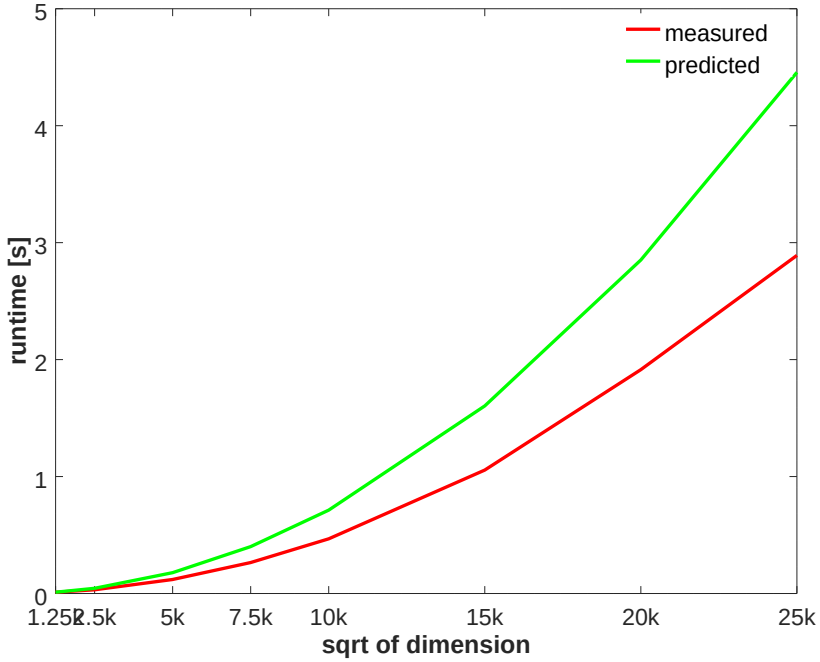
Roofline model



- Arithmetic intensity: Measure of FLOPS vs amount of memory accesses; we measured it with Oclgrind
- Theoretical peak performance of model problem: 81 GFLOPS, about 6.2% of peak performance of GPU (1.31 TFLOPS)

Performance model





$$\mathbf{T} = \max(\mathbf{T_c}, \mathbf{T_g})\mathbf{k} + (\mathbf{T_{c\to c}} + \mathbf{T_{c\to g}} + \mathbf{T_{g\to c}})(\mathbf{k}-\mathbf{1}) + \mathbf{T_{moveSoln}}$$

kernel execution

halo exchanges

download solution from GPU

k := iteration count (here: $k = 5$)

Why not use package abc?

- Frameworks/Libraries:
  - Maestro: provides automatic data transfer, task decomposition across multiple devices, autotuning of certain dynamic parameters
  - SnuCL: provides API for using all compute units in a cluster as if they were located in the host node (no need for MPI)
  - ViennaCL: linear algebra library for computations on many-core architectures (GPUs, MIC) and multi-core CPUs
- Key challenges for frameworks/libraries:
  - Find parameters for autotuning
  - Fine-grained data-movement is tricky
- We needed ability to control fine-grained data movement between compute units
- Results and performance modelling can give important information for improving existing libraries/frameworks

Takeaway

- Heterogeneous systems provides possibility of using CPUs and GPUs simultaneously
- C++11 *future* class provides easy access to threading
- OpenMP alone can only provide limited speedup (low efficiency)
- Hybrid code achieved speedup of 3.3 over CPU-only code
- Optimal for GPU to handle a large portion of the points
- Very low arithmetic intensity of 0.4375 leading to a peak performance of the code of 81 GFLOPS, i.e., more than 90% of GPU remain idle
- Improving data bandwidth between CPU and GPU can improve performance of code, e.g., by using one GPU for multiple partitions

Future work

- apply concept to multigrid V-cycle and eventually GMRES with multigrid preconditioner
- improve performance model
- extend to three dimensions
- use higher order stencils (e.g., Q2 FEM) and coupled systems
- increase halo width

Thank you!

This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0002374.