Swept time-space domain decomposition on GPUs and heterogeneous computing systems

Daniel Magee

Mechanical Engineering MS Thesis Defense
Oregon State University - School of MIME

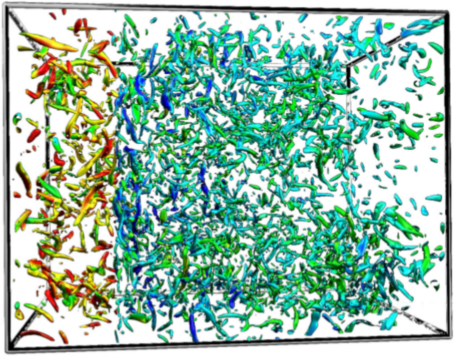
8 June 2018

Topics

The future of CFD

Challenges

- Unsteady Turbulent Flow Simulations Including Transition and Separation
- Multidisciplinary, Multiphysics Simulations and Frameworks [1]



Turbulent eddies, flowing from left to right in a shock wave (uses 1.7 million cores) [2].

How do we get there?

High performance computing - HPC

advances in HPC hardware systems and related computer software are critically important to the advancement of the state of the art in CFD simulation

---

The effectiveness and impact of CFD on the design and analysis of aerospace products and systems is largely driven by the power and availability of modern HPC systems.

- NASA CFD Vision 2030 [1]

Exascale

Exascale is the current goal of HPC development

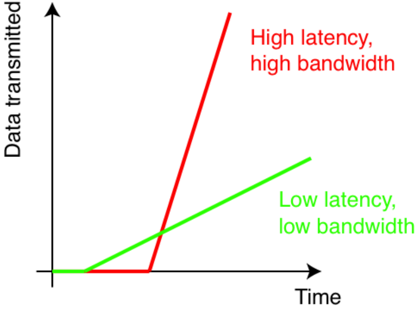
10^{18} FLOPS (Floating point operations per second)

- 4600 nodes, 25,000 Nvidia V100 GPUs.
- 200 petaFLOPS double-precision.
- 3 exaFLOPS mixed (single and half-precision) [3].

[0] Summit Supercomputer - Oak Ridge TN (Soon to be world #1)

Latency, an Exascale Challenge

As systems get larger, latency increases
TimeCost = f(flops, bandwidth, latency)

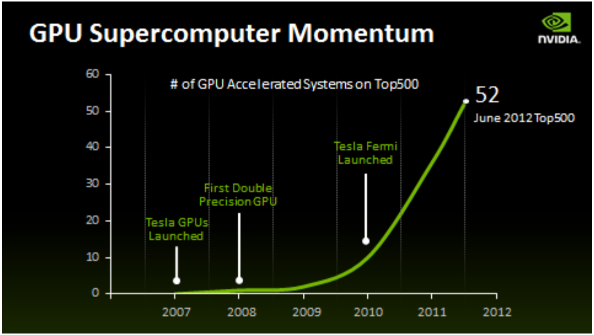


Latency and Bandwidth

"Bandwidth is money, Latency is physics."
Latency, fixed cost of memory access, is related to distance.

HPC becoming More heterogeneous

Heterogeneous: A system containing more than one processor architecture.



This is from 2012, by 2015 it was 100.

[0] https://blogs.nvidia.com/blog/2012/07/02/new-top500-list-4x-more-gpu-supercomputers

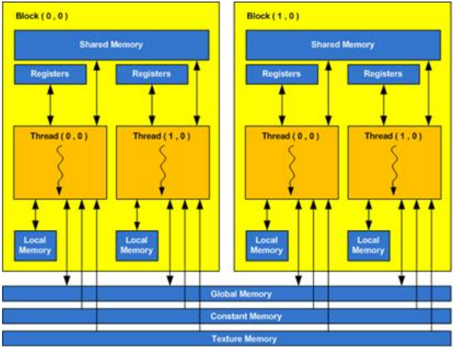
Why are GPUs good for computing?

- We realized that what is good for graphics is good for many applications.
- Many weak cores that process simple tasks quickly.
- The Memory hierarchy is exposed so we can assign values to cache and registers.

Thread Hierarchy

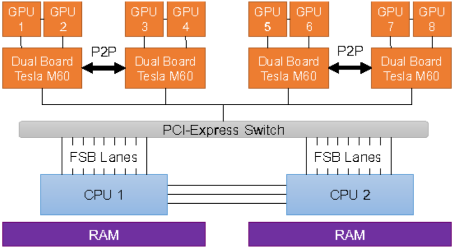
The simplicity restricts and liberates the hardware

- Threads are weak because cores are weak. Branching is penalized.
- Threads are not tied to cores but to groupings called blocks.



What Is To Be Done?

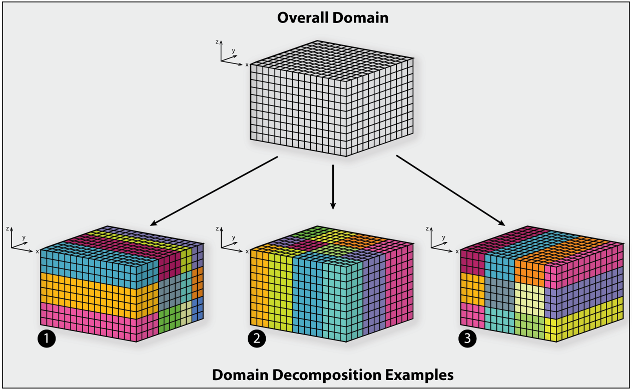
We need software to exploit the diverse architecture, but...



It gets complicated quickly.

There are many variables just on the hardware side.

Domain Decomposition



Definition

Domain decomposition is the act of splitting up a large grid among several parallel work units, essential to parallelizing grid domain problems.

[0] https://stomp.pnnl.gov/estomp_guide/44304376.stm

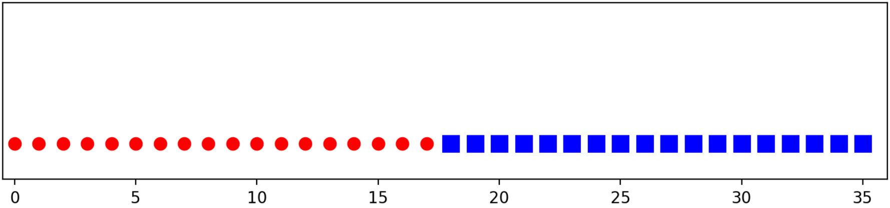
Scope of Work

Implement and analyze the performance of a swept solver on a GPU and a GPU/CPU HPC system.

Topics

Problem: Parallelizing Dependency

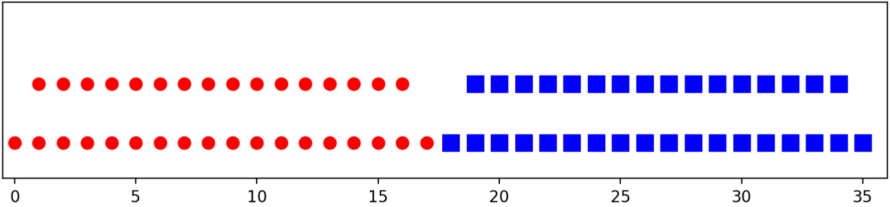
A simple (classic) decomposition:



- Initial conditions - Processes know the values at the locations they are responsible for and extra values at the edges

Problem: Parallelizing Dependency

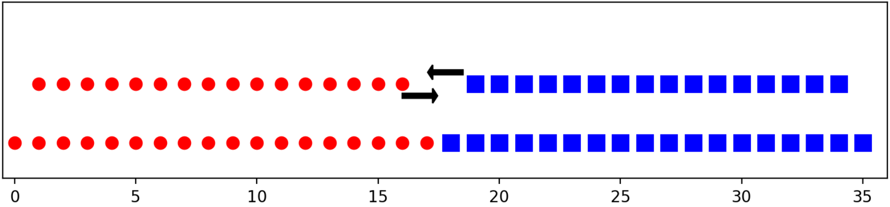
A simple (classic) decomposition:



- Initial conditions - Processes know the values at the locations they are responsible for and extra values at the edges
- Step forward - Process calculates next values at all spatial points available

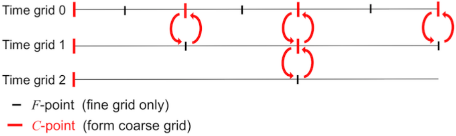
Problem: Parallelizing Dependency

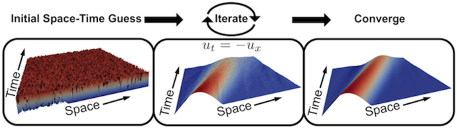
A simple (classic) decomposition:



- Initial conditions - Processes know the values at the locations they are responsible for and extra values at the edges
- Step forward - Process calculates next values at all spatial points available
- Pass Edge to neighbor process each sub-timestep.

Solution: Parallel-in-Time





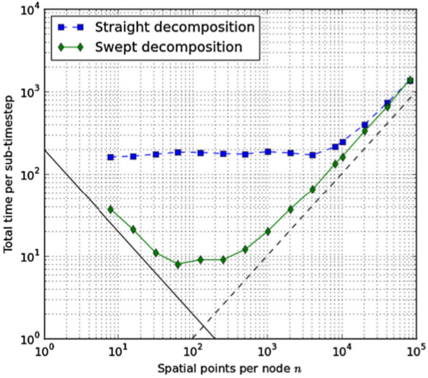
MGRID

Parallel-in-time treats the entire space-time domain as independent, begins with an initial guess, solves at various grid granularities, converges on solution.

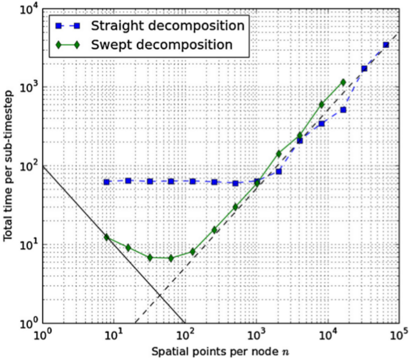
[0] computation.llnl.gov/projects/parallel-time-integration-multigrid

Similar solution: The Swept rule

The Swept Rule CPU Results from Alhubail et al. [4]



Kuramoto-Sivashinsky Equations



Euler Equations

Topics

The Swept Rule as a rule

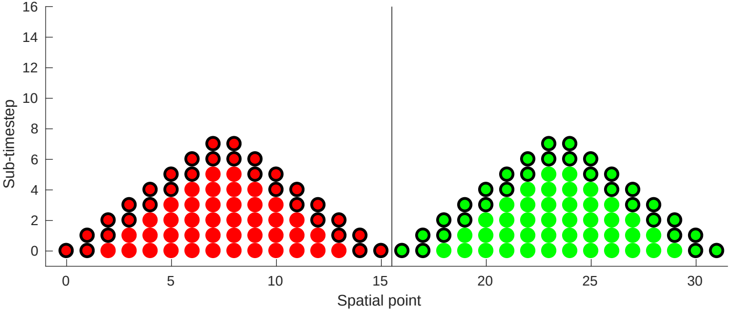
Simple Principle

Do as much work with the data closest to the processor as possible.

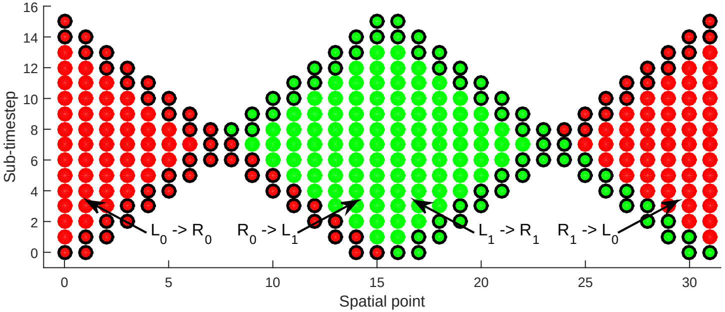
Could also say: it fully exploits the domain of dependence at all grid points.
Domain of Dependence: The region of the space-time grid that can be calculated from the initial condition.

- 1D - Triangle
- 2D - Pyramid
- 3D - Hypercube

The Swept Rule as a method



The Swept Rule as a method



Topics

Heat

Finite Difference | Time: Forward, Space: Centered

$$\frac{\partial T}{\partial t} = \alpha \nabla^2 T \, .$$

$$T_i^{m+1} = \mathrm{Fo}(T_{i+1}^m + T_{i-1}^m) + (1 - 2\mathrm{Fo})\, T_i^m \, .$$

Kuramoto-Sivashinsky (KS)

The Kuramoto–Sivashinsky equation is a nonlinear, fourth-order, one-dimensional unsteady PDE.

Finite Difference | Time: Midpoint, Space: Centered

$$u_t = -(uu_x + u_{xx} + u_{xxxx}) = -\left(\frac{1}{2}u_x^2 + u_{xx} + u_{xxxx}\right) ,$$

$$\frac{u_i^{m+1} - u_i^m}{\Delta t} = -\left(\frac{(u_{i+1}^m)^2 - (u_{i-1}^m)^2}{4\Delta x} + \frac{u_{i+1}^m + u_{i-1}^m - 2u_i^m}{\Delta x^2} + \frac{u_{i+2}^m - 4u_{i+1}^m + 6u_i^m - 4u_{i-1}^m + u_{i-2}^m}{\Delta x^4}\right) .$$

Euler Equations

Finite Volume | Time: Midpoint, Space: Minmod Limited

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} = 0$$

$$Q = \begin{Bmatrix} \rho \\ \rho u \\ \rho e \end{Bmatrix}, F = \begin{Bmatrix} \rho u \\ \rho u^2 + P \\ u(\rho e + P) \end{Bmatrix},$$

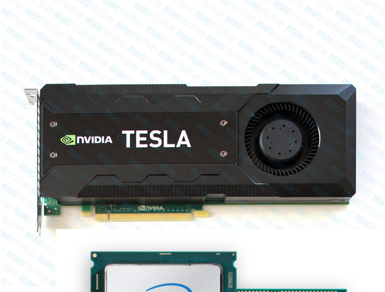
$$Q_i^{n+1} = Q_i^n + \frac{\Delta t}{\Delta x}(\text{Flux}_{i+1/2}^{n+1/2} - \text{Flux}_{i-1/2}^{n+1/2})$$

Hardware

Same Hardware CPU and GPU both studies

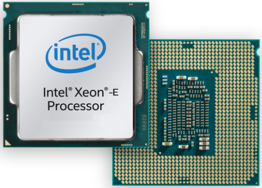
Tesla K40:

Global Memory (GB)	12
Shared Memory (kB/Block)	48
Max Threads Per Block	1024
Compute Capability	3.5
SM Count	15
ClockRate (MHz)	745
CudaCores	2880



Intel Xeon 2630-E5:

- 8 Cores
- 2.5 GHz



Topics

What we want to know

- Which GPU memory strategy is best for the swept rule?
- Is swept decomposition effective compared to a simple **(Classic)** scheme on the GPU?

1^{st} Study Test Procedure

Performance Metric

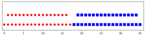
Average time per **timestep**.

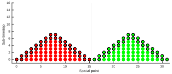
Test Run Details

- CUDA 8, Double Precision
- 32 to 1024 threads per block by powers of 2.
- 1024 to 1048576 spatial points by powers of 2.
- 50,000 timesteps

Implementation choices

- **Classic:** One sub-timestep at a time.

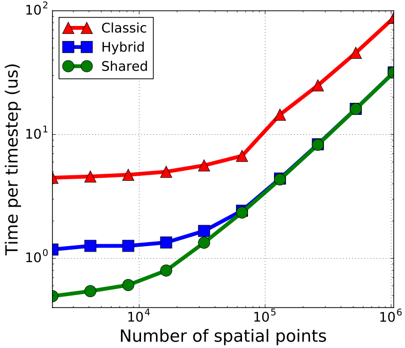


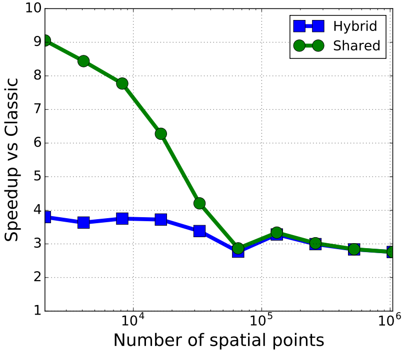


- **Swept:**
  - **Shared:** Shared Memory - performs all computation on GPU
  - **Hybrid:** Passes edge domains to CPU to avoid boundary conditions.
  - **Register:** Register memory - shuffled between warp threads.

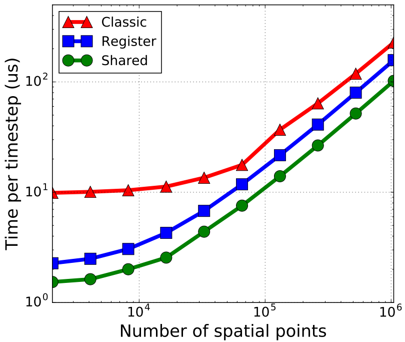
Heat Equation

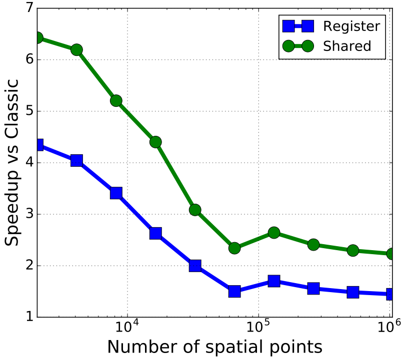
Best Run at each problem size



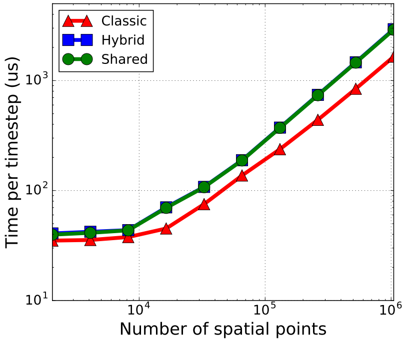


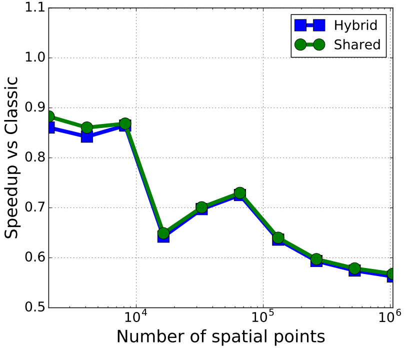
KS Equation


Classic
Register
Shared
Time per timestep (us)
Number of spatial points
10^0
10^1
10^2
10^4
10^5
10^6

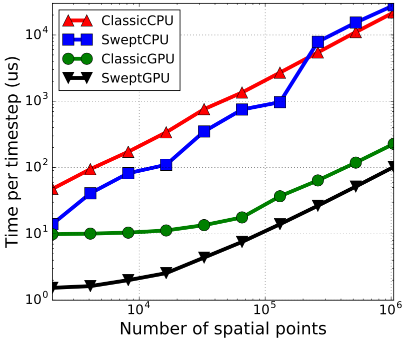

Register
Shared
Speedup vs Classic
Number of spatial points
1
2
3
4
5
6
7
10^4
10^5
10^6

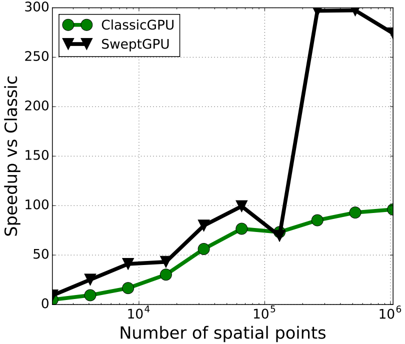
Euler Equation





KS Equation CPU vs GPU



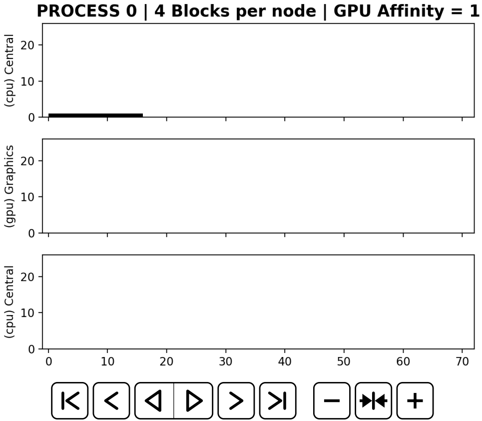


Takeaways

- Shared memory is generally the most effective storage strategy.
- GPUs are faster than CPUs for these types of problems.
- The swept rule becomes less effective as problem complexity grows.

Topics

Heterogeneous swept rule domain splitting

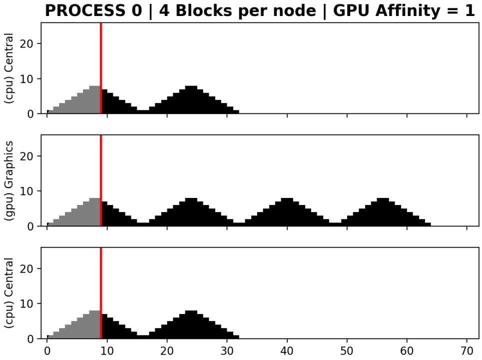


Allocate
$tpb * (nDomains + .5)$
slots per process

Constraints

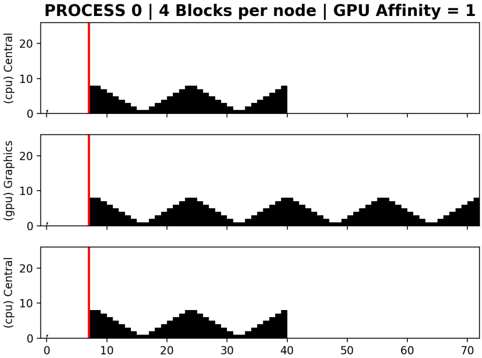
- Each process receives an even number of blocks.
- GPUs communicate with a single process, and computes blocks are embedded in that process.

Heterogeneous swept rule domain splitting



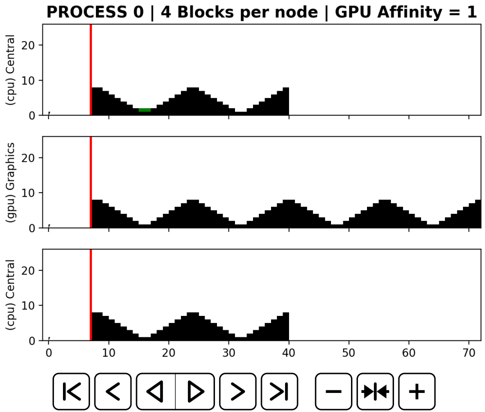
Gather items to pass

Heterogeneous swept rule domain splitting



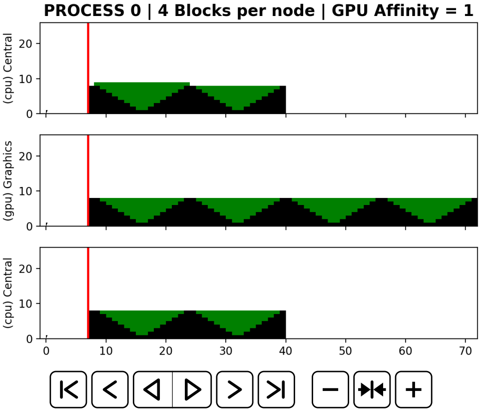
Set new starting point for domains

Heterogeneous swept rule domain splitting



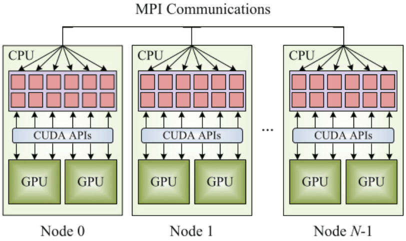
Fill in the voids

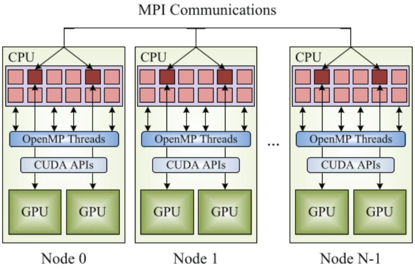
Heterogeneous swept rule domain splitting



March forward to next triangle

Design Point: Software Pattern

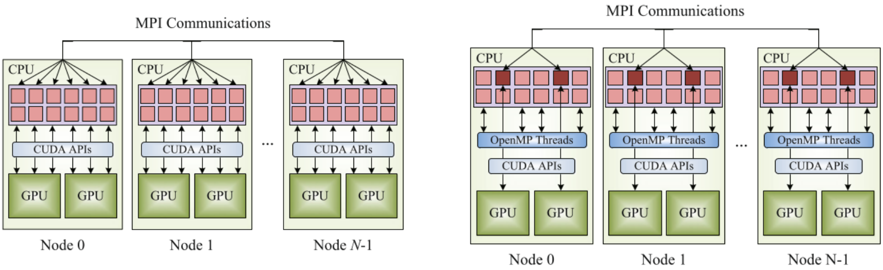




- **MPI:** Message Passing Interface - Industry standard for distributed memory paralleization.
- **OpenMP:** Open Multiprocessing - Launches threads in shared memory space
- **CUDA:** API for GPU execution

Should we parallelize within sockets with OpenMP? [5]

Design Point: Software Pattern

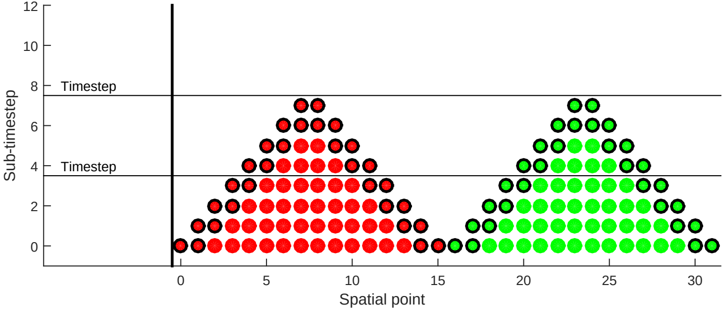


- **MPI:** Message Passing Interface - Industry standard for distributed memory paralleization.
- **OpenMP:** Open Multiprocessing - Launches threads in shared memory space
- **CUDA:** API for GPU execution

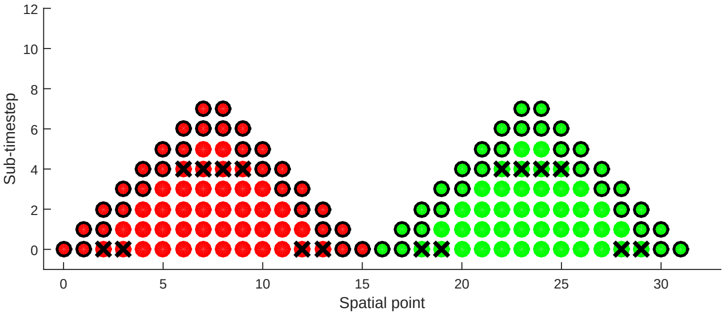
Should we parallelize within sockets with OpenMP? [5]
No, literature shows little evidence of utility [6, 7].

There's a catch

Anything other than the simplest method (FTCS, Leapfrog) will overwrite values needed to continue the computation.

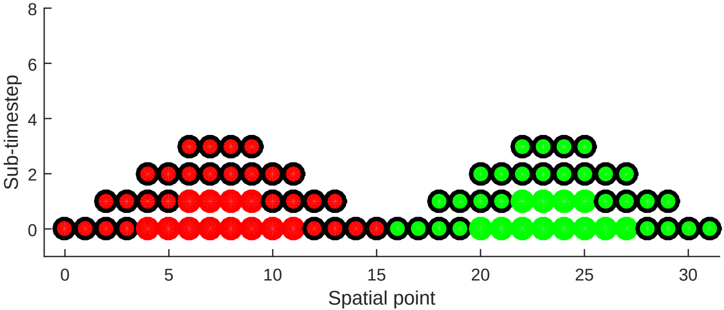


There's a catch



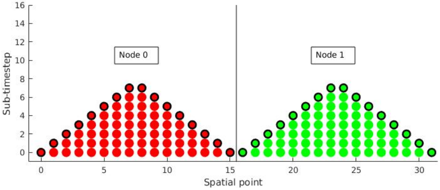
Solution 1: Flattening

Multi-step methods can often combine steps in with wider stencil.



Solution 2: Lengthening (Atomic Decomposition) [8]

All explicit schemes can be decomposed into three-point stencil steps




```
// Q = {rho, rho*u, rho*E} Euler
struct states {
  double3 Q[2]; // State Vars
  double Pr; // Pressure ratio
};

// KS
struct states {
  double u[2]; // Velocity
  double uxx; // Jerk
};

// Heat
struct states {double T[2];};
```

Long Flat KS Discretiztions

Finite Difference | Time: Midpoint, Space: Centered

Using a 5 point stencil

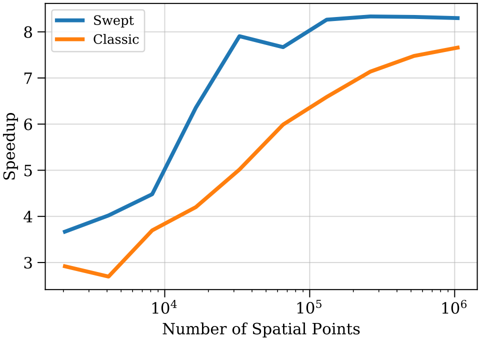
$$\frac{u_i^{m+1} - u_i^m}{\Delta t} = -\left(\frac{(u_{i+1}^m)^2 - (u_{i-1}^m)^2}{4\Delta x} + \frac{u_{i+1}^m + u_{i-1}^m - 2u_i^m}{\Delta x^2} + \frac{u_{i+2}^m - 4u_{i+1}^m + 6u_i^m - 4u_{i-1}^m + u_{i-2}^m}{\Delta x^4}\right).$$

But we can treat u_{xxxx} as $\frac{\partial^2 u_{xx}}{\partial x^2}$

$$\frac{u_i^{m+1} - u_i^m}{\Delta t} = -\left(\frac{(u_{i+1}^m)^2 - (u_{i-1}^m)^2}{4\Delta x} + \frac{(u + u_{xx})_{i+1}^m + (u + u_{xx})_{i-1}^m - 2(u + u_{xx})_i^m}{\Delta x^2}\right).$$

Flattening vs Lengthening

Tested under same conditions as GPU-only with Kuramoto-Sivashinsky equation.



The flexibility of Lengthening on the GPU comes at a substantial cost. We still use the lengthening strategy for its universal qualities in the heterogenous case.

Topics

New Questions

- How much work should we give to the GPU in a heterogeneous system?
- Which strategy for higher order methods is faster?
- Is swept decomposition more effective for more complex equations on heterogeneous architecture?

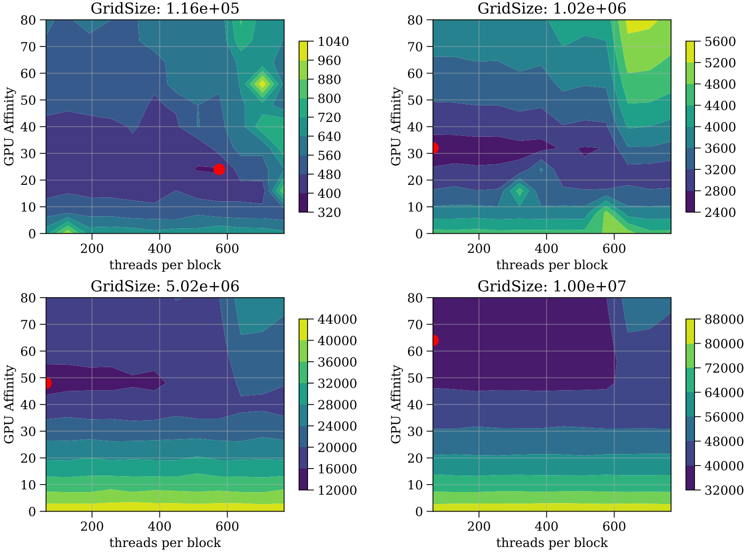
Heterogenous Changes

Use what we learned last time.

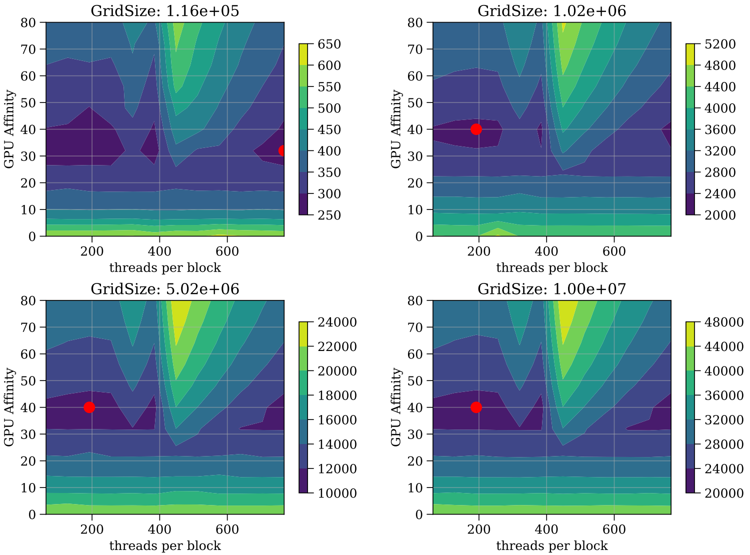
New Conditions

- Shared is the best GPU-only algorithm, so we'll use it.
- OSU COE cluster across 2 nodes with 20 cores each and one GPU.
- Increase test grid size.
- Use a screening study to narrow the test grid

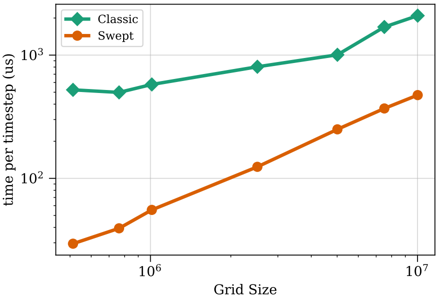
Launch Configuration Study Euler Classic

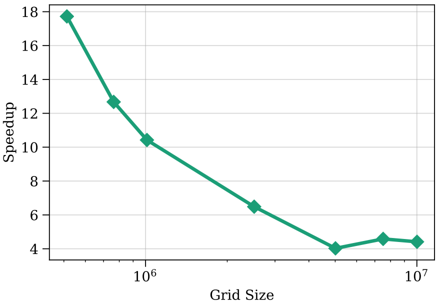


Launch Configuration Study Euler Swept

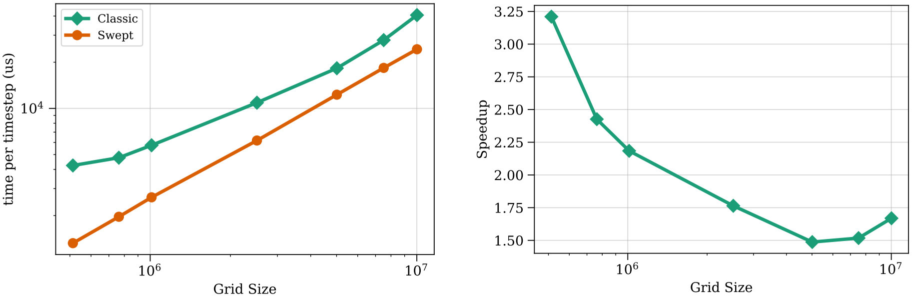


Heat Results





Euler Results



Conclusions

- Shared memory is an effective storage strategy.
- The swept rule is comparatively more effective for simpler problems
- The swept rule is more effective when communication costs are greater, i.e. cluster.
- GPUs must be given many times more work than CPUs to stay busy.

Future Work

- Use Euler for lengthening vs flattening comparison.
- 2D Implementation
- Refine hSweep library workflow
- Unstructured grids

Acknowledgments

NASA award
No. NNX15AU66A

Nvidia (donated GPU)

Niemeyer Research
Group

Questions

Questions?

Works Cited I

J. P. Slotnick, A. Khodadoust, J. J. Alonso, D. L. Darmofal, W. D. Gropp, E. A. Lurie, and D. J. Mavriplis, "CFD vision 2030 study: A path to revolutionary computational aerosciences," NASA Technical Report, NASA/CR-2014-218178, NF1676L-18332, Mar. 2014.

I. Bermejo-Moreno, J. Bodart, J. Larsson, B. M. Barney, J. W. Nichols, and S. Jones, "Solving the compressible Navier–Stokes equations on up to 1.97 million cores and 4.1 trillion grid points," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 62:1–62:10.

Works Cited II

M. Feldman, "Oak ridge readies summit supercomputer for 2018 debut," https://top500.org/news/oak-ridge-readies-summit-supercomputer-for-2018-debut/, Nov. 2017, accessed: 27 May 2018.

M. Alhubail and Q. Wang, "The swept rule for breaking the latency barrier in time advancing PDEs," *Journal of Computational Physics*, vol. 307, pp. 110–121, 2016.

F. Lu, J. Song, F. Yin, and X. Zhu, "Performance evaluation of hybrid programming patterns for large cpu/gpu heterogeneous clusters," *Computer Physics Communications*, vol. 183, no. 6, pp. 1172–1181, 2012.

Works Cited III

R. T. Mills, K. Rupp, M. Adams, J. Brown, T. Isaac, M. Knepley, B. Smith, and H. Zhang, "Software strategy and experiences with manycore processor support in petsc," SIAM Pacific Northwest Regional Conference, Oct. 2017.

D. A. Jacobsen and I. Senocak, "Multi-level parallelism for incompressible flow computations on gpu clusters," *Parallel Computing*, vol. 39, no. 1, pp. 1–20, 2013.

Q. Wang, "Decomposition of stencil update formula into atomic stages," 2017.

QuestionsPlus

Questions?