Parallel 3D Geometric Multigrid Solver on GPU Clusters

BOISE STATE UNIVERSITY

College of Engineering

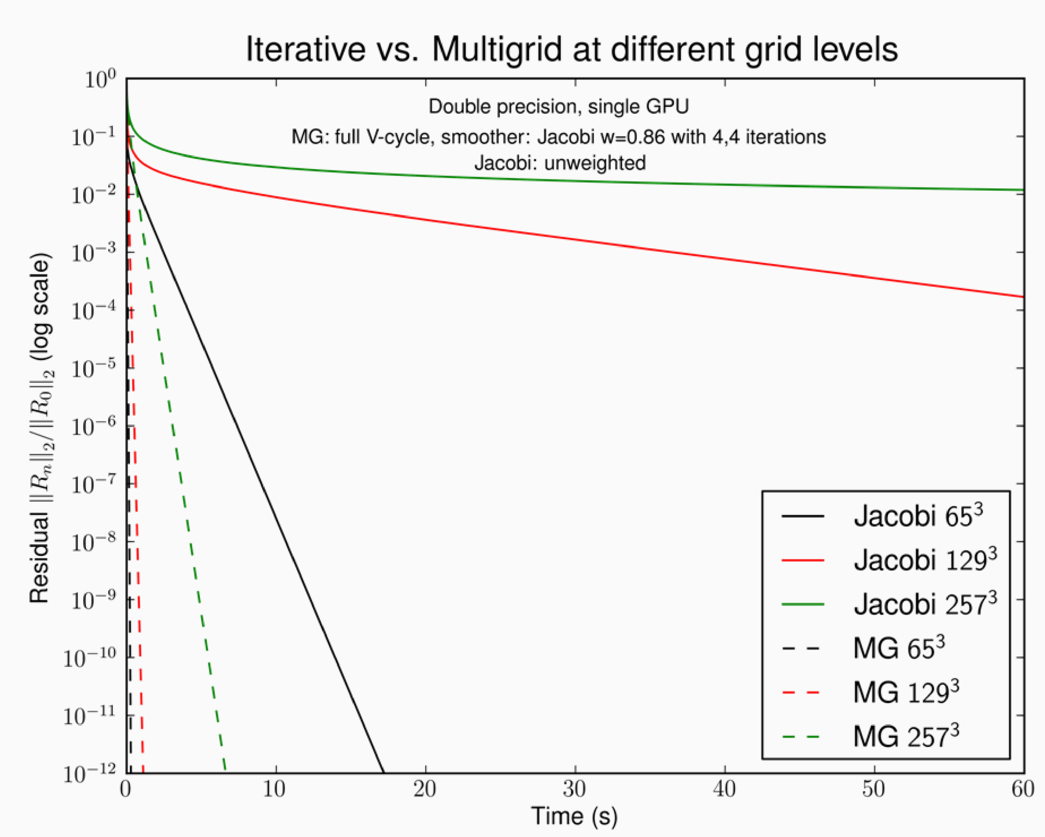
Dana Jacobsen and İnanç Şenocak

Objective: Investigate the performance and scalability of a multigrid pressure Poisson equation solver running on a GPU cluster.

Method: A parallel 3D multigrid pressure solver was written for GIN3D, a 3D incompressible Navier-Stokes flow solver which runs on GPU clusters. Tests were performed using the well-known lid-driven cavity and natural convection in a cavity problems.

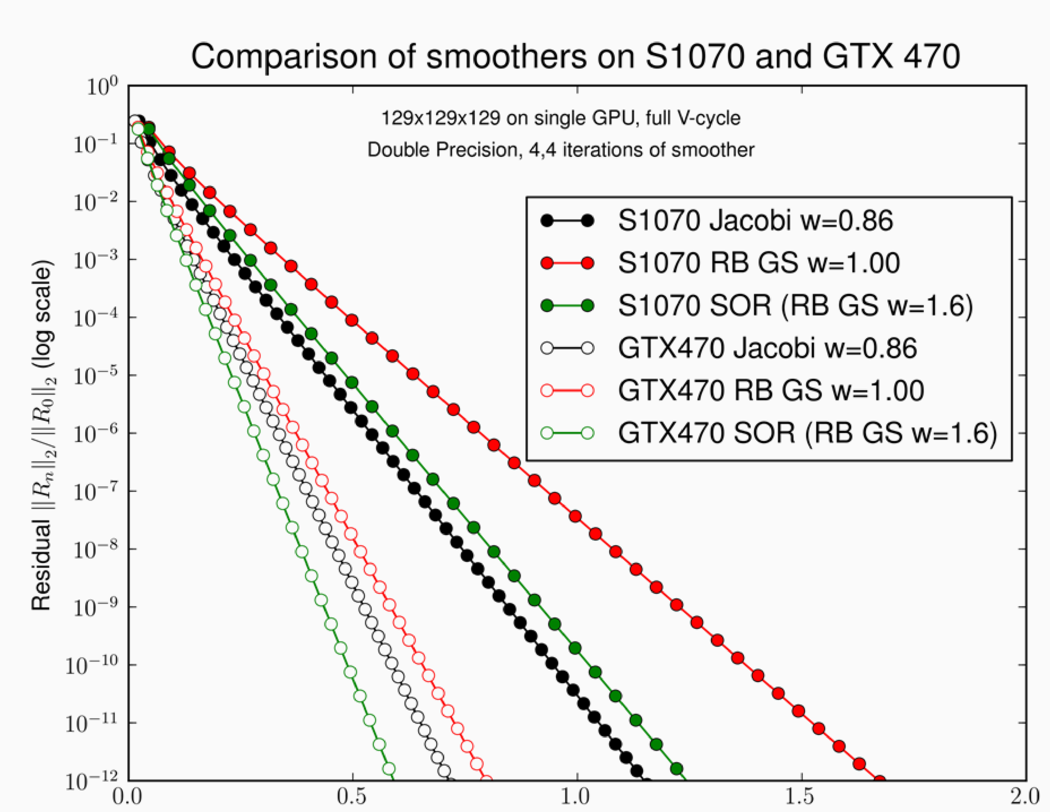
Solver:

- For pressure Poisson equation in incompressible Navier-Stokes flow solver.
- Typically consumes a majority of the time.
- Simple iterative solvers such as Jacobi and Gauss-Seidel can run very efficiently on the GPU, but solutions converge very slowly.
- Multigrid methods can converge rapidly and allow convergence to be relatively independent from the grid size, which is important for large computational meshes.



Plotting residual reduction for a single timestep. Multigrid V-cycles were repeated as needed to lower the residual to machine precision. Similarly, the Jacobi solver was iterated repeatedly.

As the grid size increases, $O(N^2)$ iterative solvers take excessively long to reduce the error. In contrast, the multigrid solver shows much better performance.



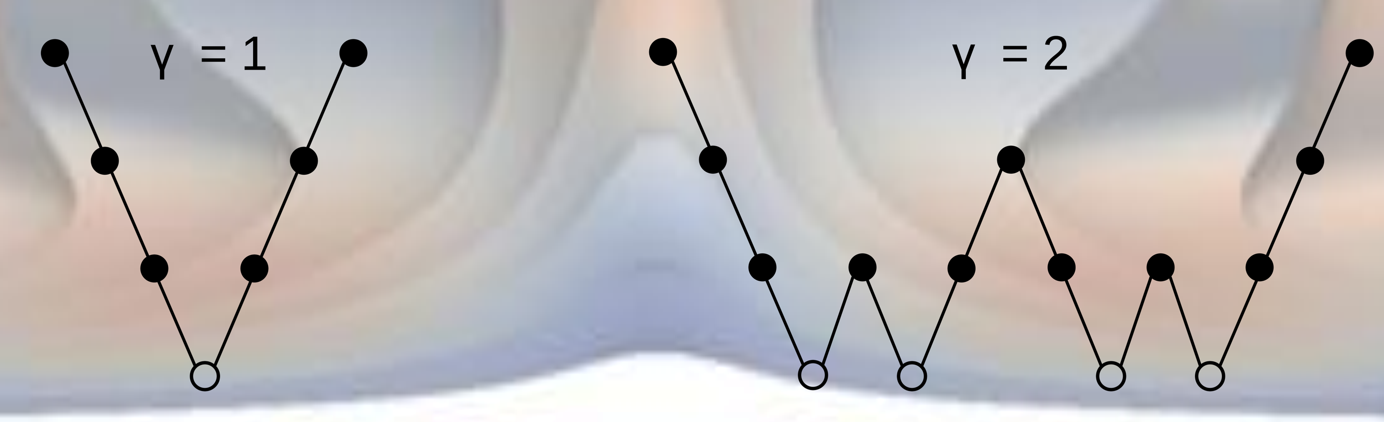
This chart compares different smoothers inside the multigrid implementation. Again, residual reduction is plotted against time, and shown for both a single S1070 GPU and a GTX 470 (Fermi).

While Red-Black Gauss-Seidel takes fewer iterations, and SOR even fewer, the particular Jacobi implementation is more efficient in time on the S1070.

In contrast, the same kernels running on a GTX 470 (Fermi) show SOR as slightly more efficient. All solvers are faster on the new architecture.

Multigrid Algorithm

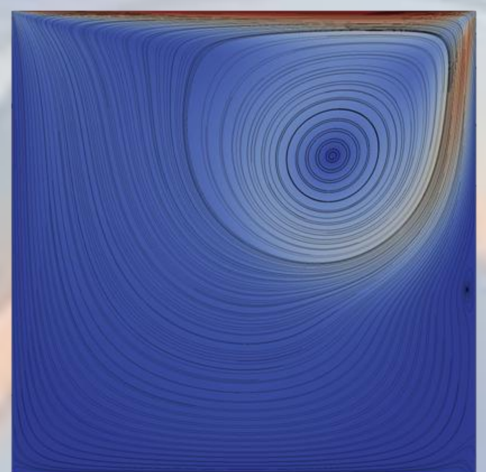
- *cycle*(γ, u_k, f, v_1, v_2)
  - Smooth v_1 times
  - Compute residual $r_k = f - \mathcal{L}\, u_k$
  - Restrict residual $r_{k-1} = \mathcal{R}\, r_k$
  - Compute approximate solution v_{k-1}:
    - "Direct Solve" if lowest level, otherwise
    - Repeat γ times: *cycle*(γ, 0, r_{k-1}, v1, v2)
  - Prolongate correction: $u_k = u_k + \mathcal{P}\, v_{k-1}$
  - Smooth v_2 times



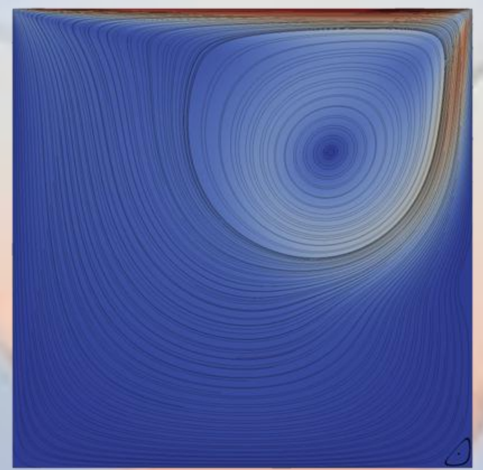
V-cycles W-cycles

Multigrid Implementation

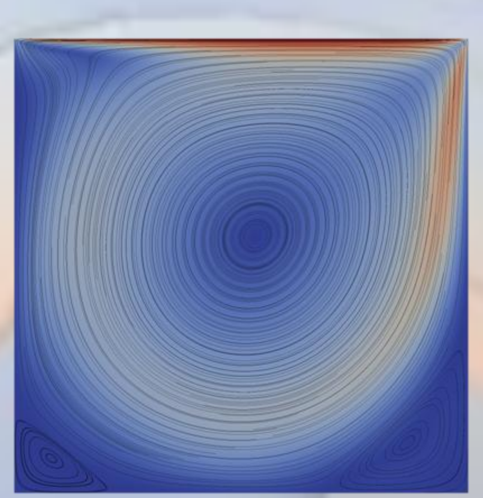
Four kernels: Laplacian, Restriction (full-weighting), Smooth (Jacobi or Red-Black Gauss-Seidel), and Prolongation.



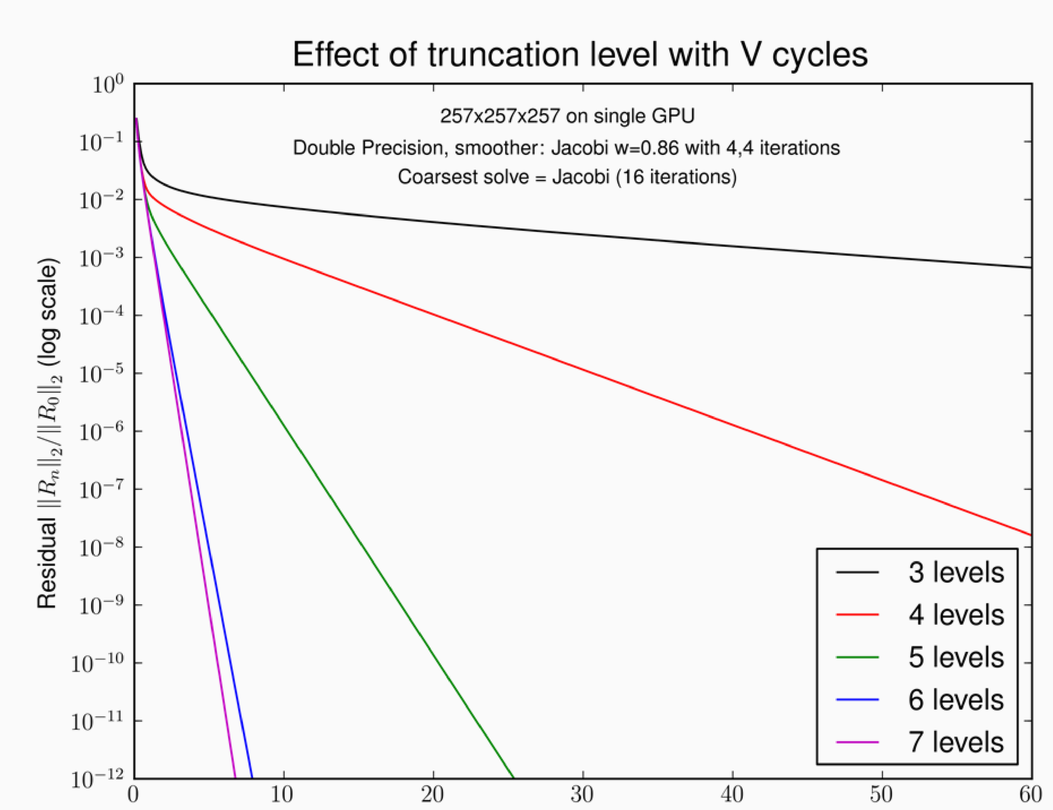
Jacobi (20 iterations) after 4.7s



Multigrid after 4.7s



After 59 seconds

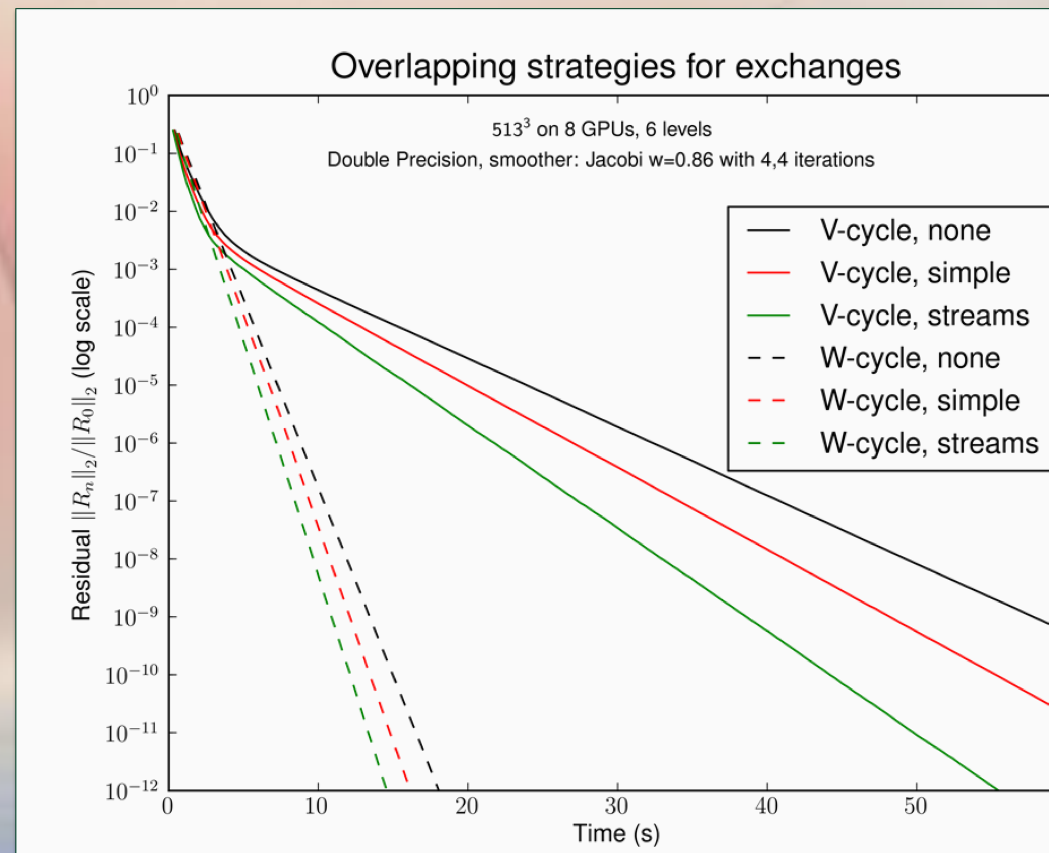


The simple parallel multigrid implementation which always has some data on each GPU (and does not repartition) has its truncation level restricted by the number of GPUs. With a simple fixed iteration coarse grid solver, the time taken grows rapidly.

As the number of GPUs grows, a strategy such as amalgamation (grouping work from multiple GPUs to one for coarse grids), semi-coarsening (reduction in fewer than all three dimensions), repartitioning, or some combination is needed.

Question:

Which of these parameters are most important to tune? *Truncation level*, *cycle type*, *smoother type*, *method for coarsest grid solve*.



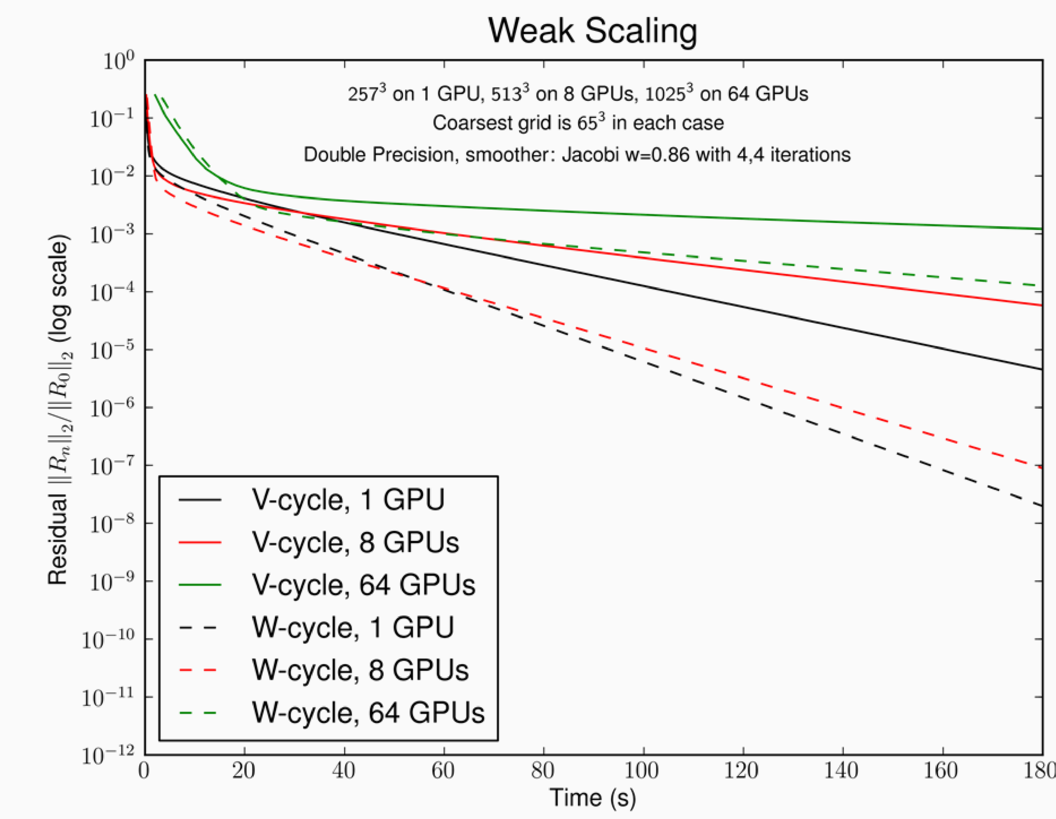
Different overlapping strategies as well as cycle types are compared.

Overlapping device computation, device/host memory transfers, and MPI network communication is at least as important with multigrid methods as with simple iterative ones. Maximizing overlap with CUDA streams is particularly effective.

While V-cycles are faster at the start, the coarse grid solver used isn't effective enough to keep them efficient, so W-cycles are asymptotically superior for this example.

Details of the overlapping strategies are shown in the 2009 NVIDIA GTC poster and 2010 AIAA paper by the authors.

- No overlap: computation followed by synchronous exchange.
- Simple overlap: Compute edges, do async MPI exchange while computing middle.
- Streams overlap: Use CUDA Streams to asynchronously overlap computation, host / device memory transfers, and network communication.



Below: Percent of time spent in computation vs. copy + network communication.

	V-cycles	W-cycles
1 GPU	100%	100%
8 GPUs	18%	14%
64 GPUs	41%	21%

Changing the problem size for weak scaling results in significantly different convergence behavior with truncated multigrid. The weak scaling analysis of multigrid is much more complex than for iterative solvers.

Conclusions:

- Multigrid on GPUs can be effective and efficient.
- Multigrid performance significantly benefits from deeper truncation levels.
- The Fermi architecture makes a difference in relative smoother performance.
- Coarsest level operations have a large impact on cluster scalability.

Acknowledgments: ***■ NASA Idaho Space Grant Consortium ■ NVIDIA Professor Partnership Program ■ NCSA grant #ATM100032***