Accelerating Wave-Propagation Algorithms on Adaptive Mesh with the Graphics Processing Unit (GPU)

Xinsheng (Shawn) Qin[1], Randall LeVeque[1], Michael Motley[1] ,
[1]University of Washington, Seattle, WA

Introduction

Clawpack is a library for solving nonlinear hyperbolic partial differential equations using high-resolution finite volume methods based on Riemann solvers and limiters. It supports Adaptive Mesh Refinement (AMR), which is essential in solving multi-scale problems. Recently, we added capabilities to accelerate the code by using the Graphics Process Unit (GPU). Routines that manage CPU and GPU AMR data and facilitate the execution of GPU kernels are added. Customized and CPU thread-safe memory managers are designed to manage GPU and CPU memory pools, which is essential in eliminating the overhead of memory allocation and de-allocation. A global reduction is conducted every time step for dynamically adjusting the time step based on Courant number restrictions. Some small GPU kernels are merged into bigger kernels, which greatly reduces kernel launching overhead. A speed-up between 2 and 3 for the total running time is observed in an acoustics benchmark problem.

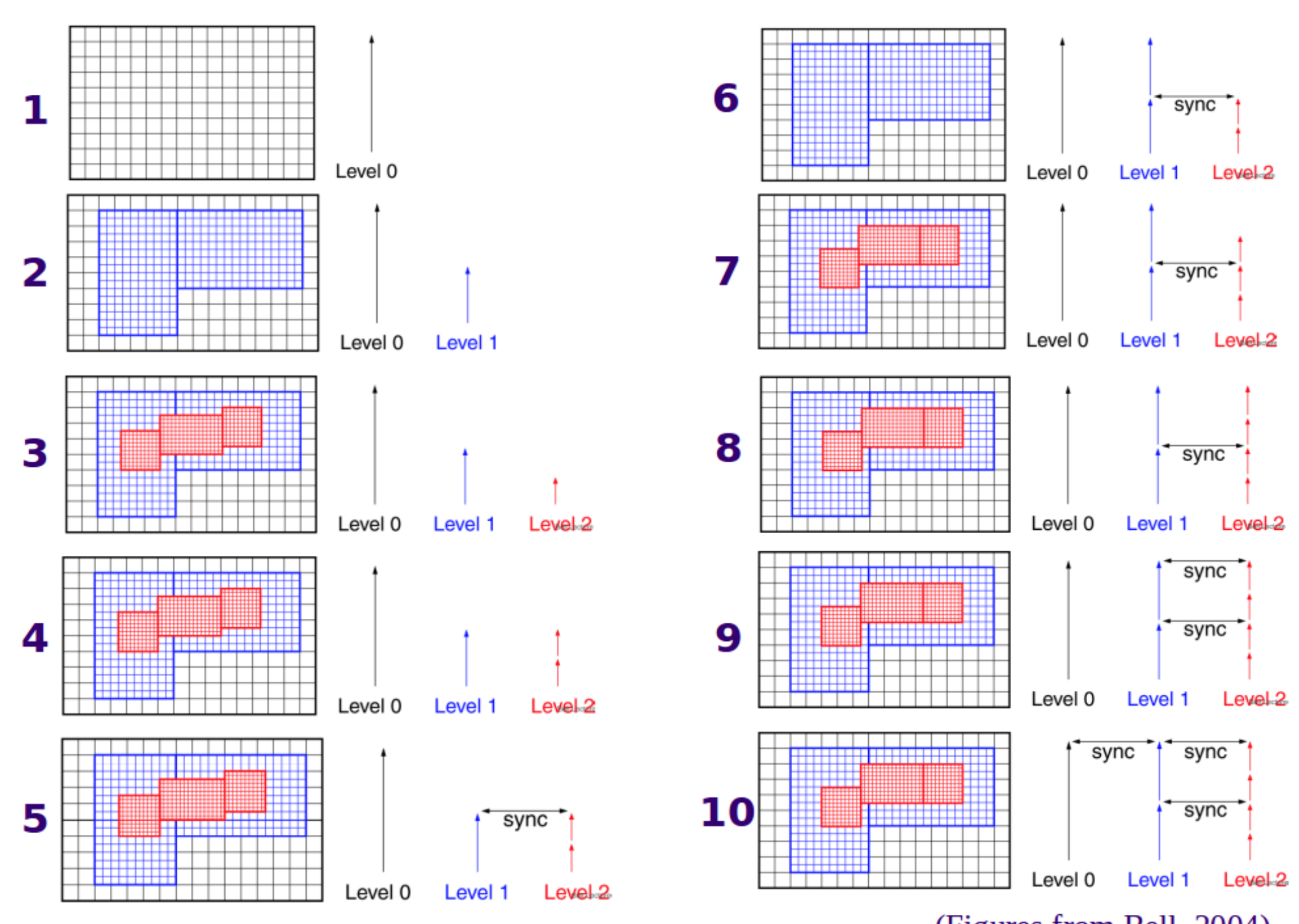
Wave Propagation Algorithm

Hyperbolic PDEs in form:
$$q_t + f(q)_x + g(q)_y = 0$$

Cell approximation:
$$Q_{ij} = \frac{1}{\Delta x \Delta y}\int_{y_{j-1/2}}^{y^{j+1/2}}\int_{x_{i-1/2}}^{x^{i+1/2}} q(x,y,t^n)dxdy$$

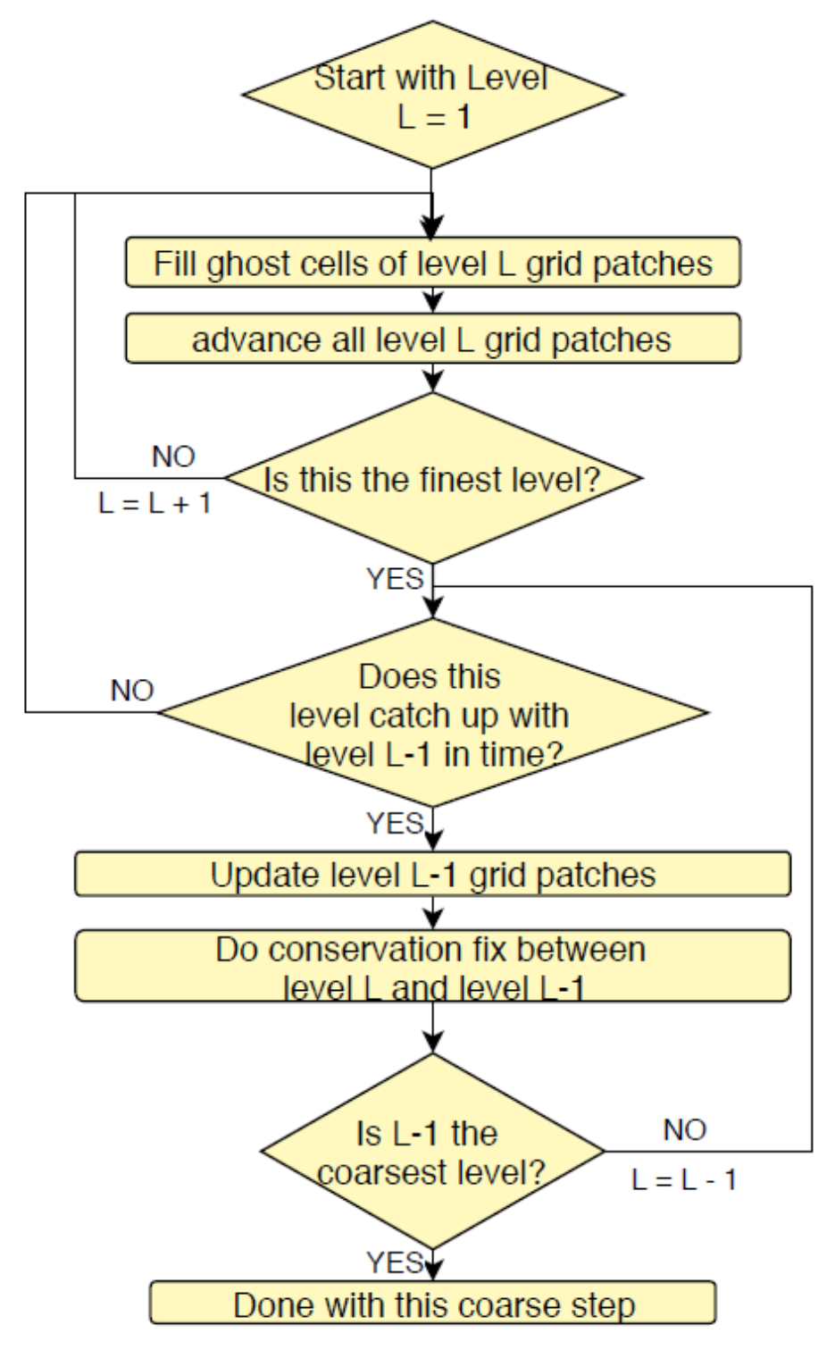
Wave propagation algorithm:
$$\begin{aligned} Q_{ij}^{n+1} =& Q_{ij} - \frac{\Delta t}{\Delta x}\left(\mathcal{A}^+\Delta Q_{i-1/2,j} + \mathcal{A}^-\Delta Q_{i+1/2,j}\right) - \frac{\Delta t}{\Delta y}\left(\mathcal{B}^+\Delta Q_{i,j-1/2} + \mathcal{B}^-\Delta Q_{i,j+1/2}\right) \\ & - \frac{\Delta t}{\Delta x}\left(\tilde{F}^n_{i+1/2,j} - \tilde{F}^n_{i-1/2,j}\right) - \frac{\Delta t}{\Delta y}\left(\tilde{G}^n_{i,j+1/2} - \tilde{G}^n_{i,j-1/2}\right)\end{aligned}$$

Adaptive Mesh Refinement (AMR)



(Figures from Bell, 2004)

Time stepping with AMR grid patches.


Start with Level L = 1
Fill ghost cells of level L grid patches
advance all level L grid patches
Is this the finest level?
NO
L = L + 1
YES
Does this level catch up with level L-1 in time?
NO
YES
Update level L-1 grid patches
Do conservation fix between level L and level L-1
Is L-1 the coarsest level?
NO
L = L - 1
YES
Done with this coarse step

A flow chart of AMR algorithm including 1) advancing solution, 2) updating process 3) and conservation fix. Assume at least two levels of grid patches exist.

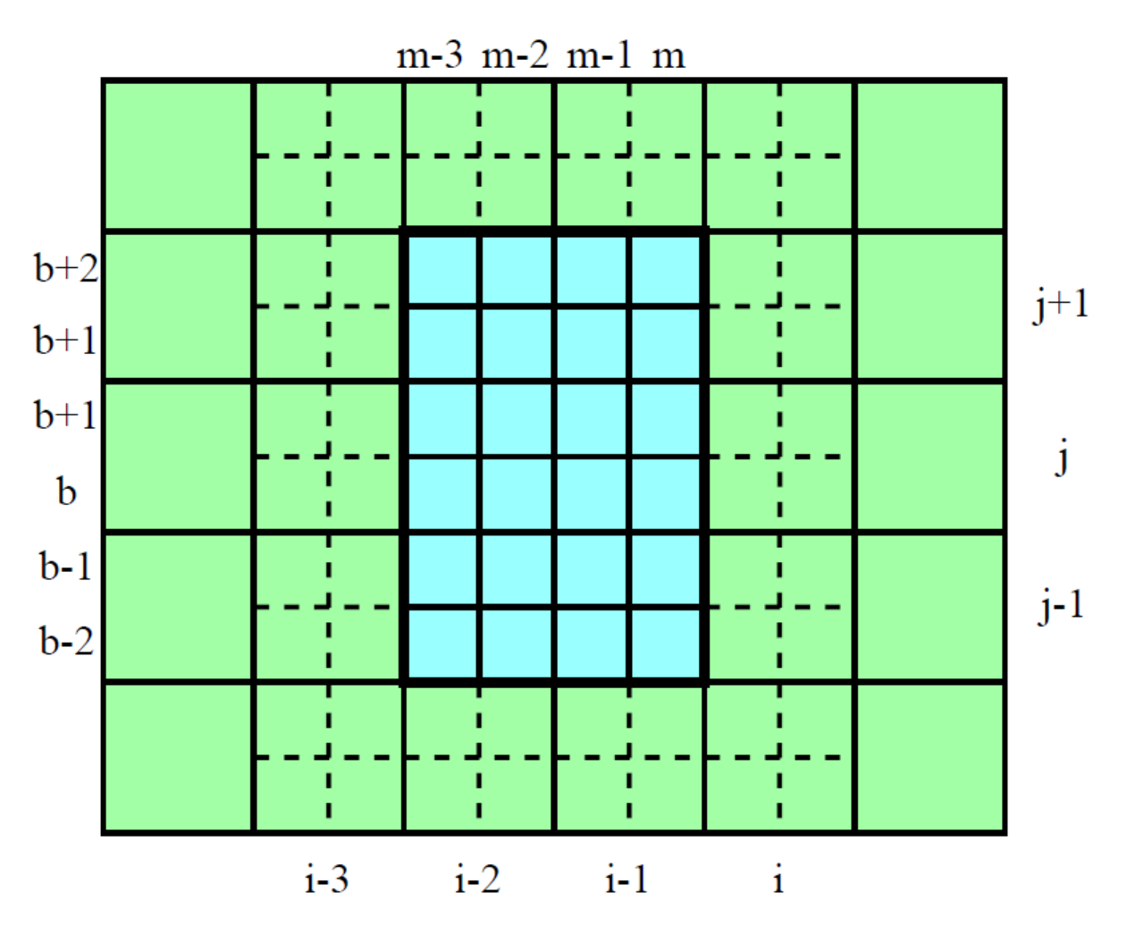
- **Updating process:**
  When the fine grid patches catch up with the coarse grid patches in time then the solution in coarse grid cells covered by any fine grid cell will be over-written by the average solution in fine grid cells.

$$Q_{i-1,j} := \frac{1}{4}\left(\hat{Q}_{m-1,b} + \hat{Q}_{m,b} + \hat{Q}_{m-1,b+1}\hat{Q}_{m,b+1}\right)$$

- **Conservation fix:**
  After the updating process, solution in coarse cells that are next to the coarse-fine grid patch interface must be modified to preserve global conservation of state variables.

$$Q_{ij}^{n+1} := Q_{ij}^{n+1} + C1 + C2 + C3$$

where C1, C2 and C3 are modification terms and are functions of waves and fluxes at the coarse-fine grid patch interface.

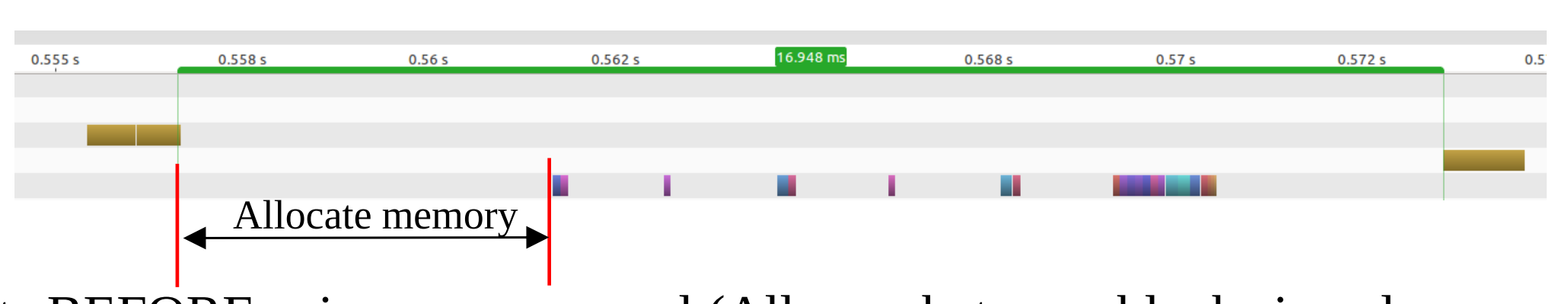


A 2D coarse grid patch and a nested fine grid patch. i and j are index for the coarse gird in x- and y-directions. m and b are index for the fine gird in x- and y-directions

Code Design and Implementations

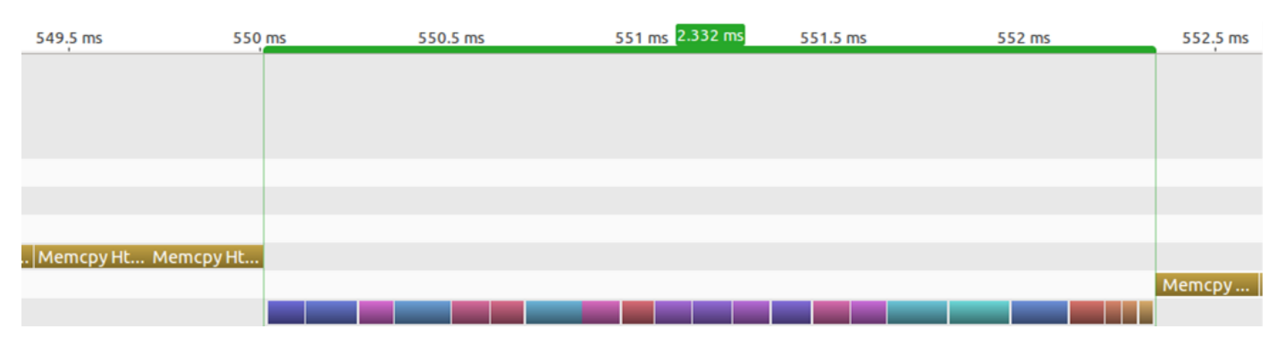
- **Customized GPU and CPU Memory Allocators**

- Allocating system CPU/GPU memories by calling CUDA runtime API directly has a very large overhead (can take up to 80% of the computational time).
- We wrote memory allocator that manages pre-allocated CPU and GPU memory pools.
- This type of overhead is almost completely removed.



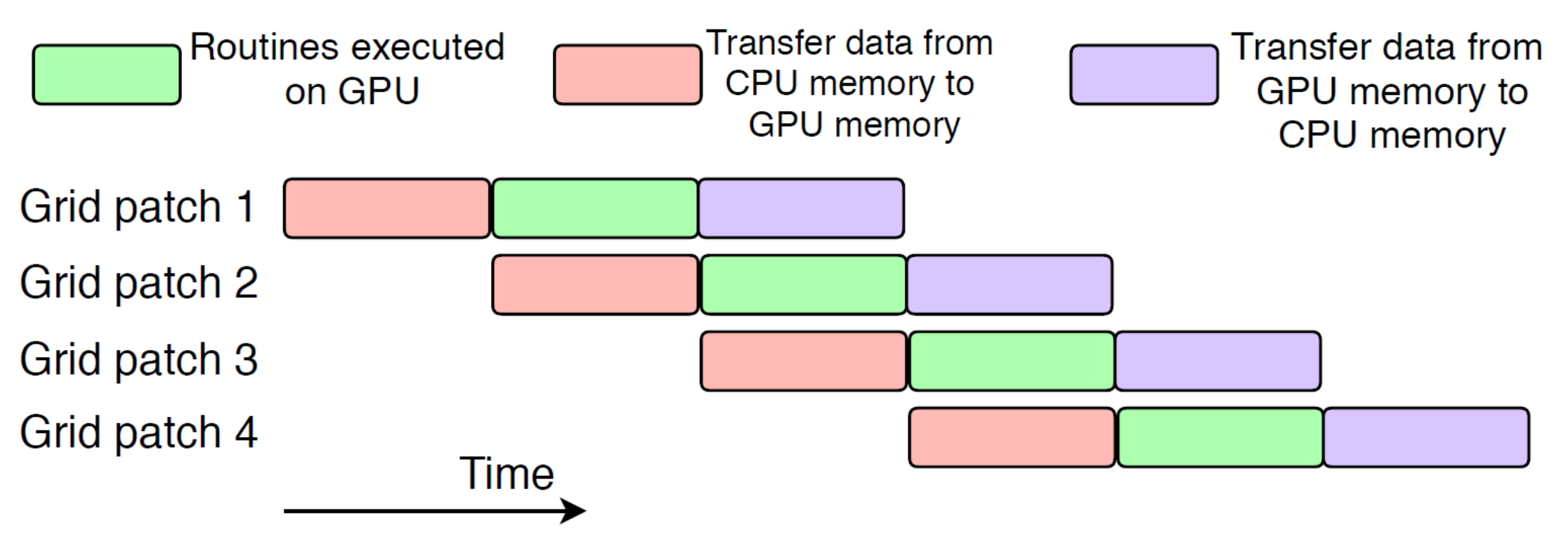
Profiling results BEFORE using memory pool (All gaps between blocks in colors represent time spent on allocating GPU memory. The blocks represent time spent on actual computation on GPUs)

use memory pool



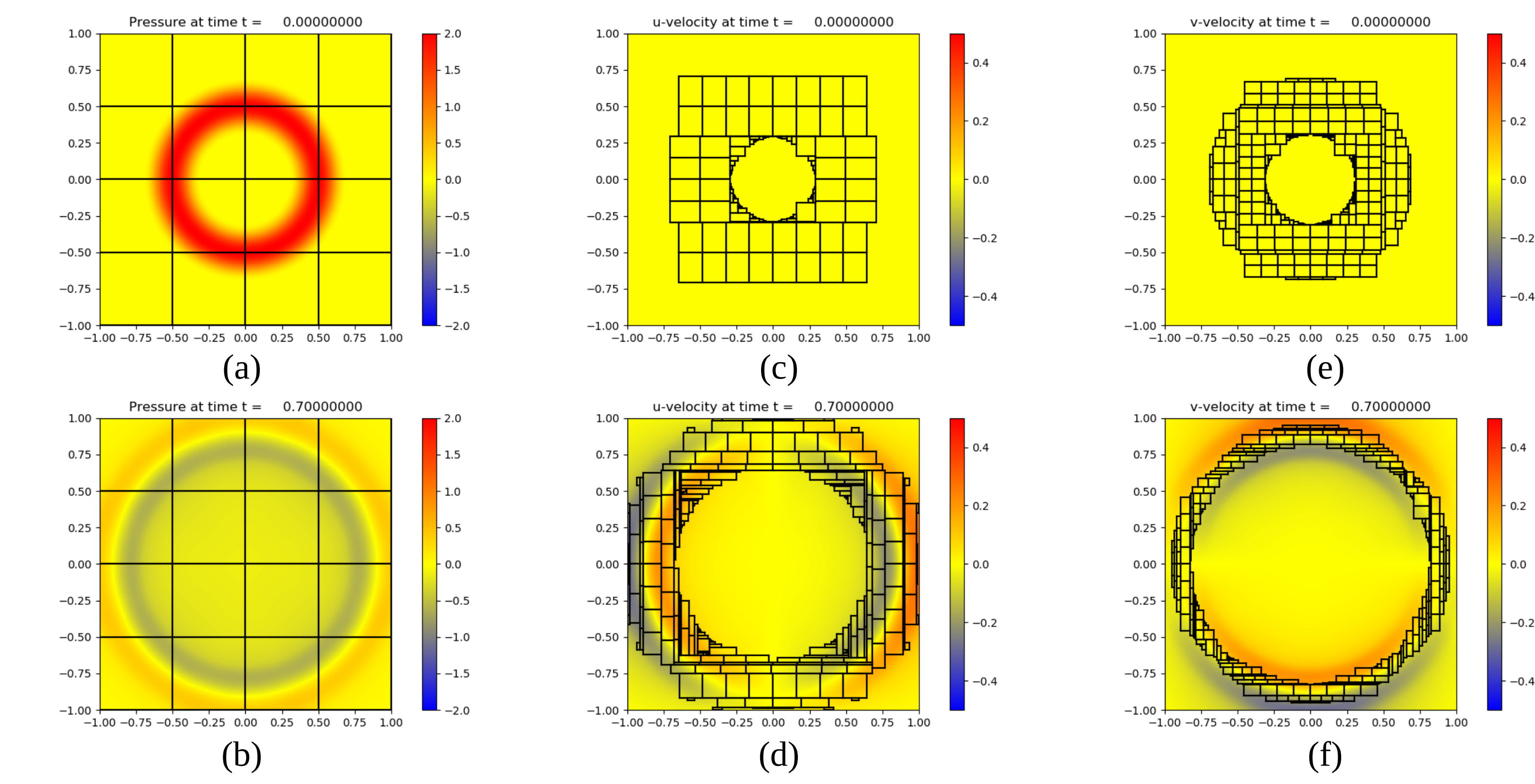
Profiling results AFTER using memory pool (There is almost no gap between blocks)

- **Pipeline for Hiding Data Transfer**

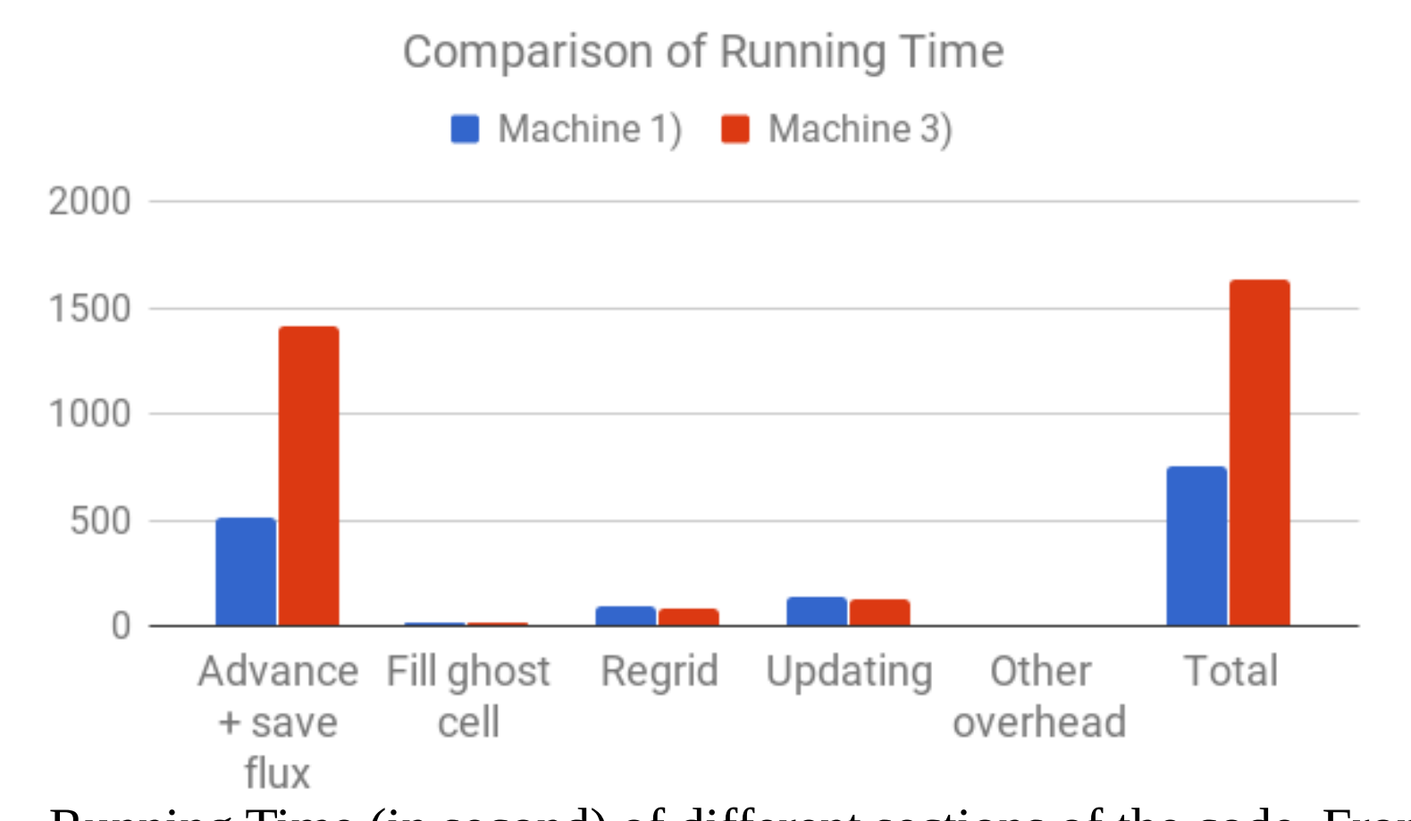

Routines executed on GPU
Transfer data from CPU memory to GPU memory
Transfer data from GPU memory to CPU memory
Grid patch 1
Grid patch 2
Grid patch 3
Grid patch 4
Time

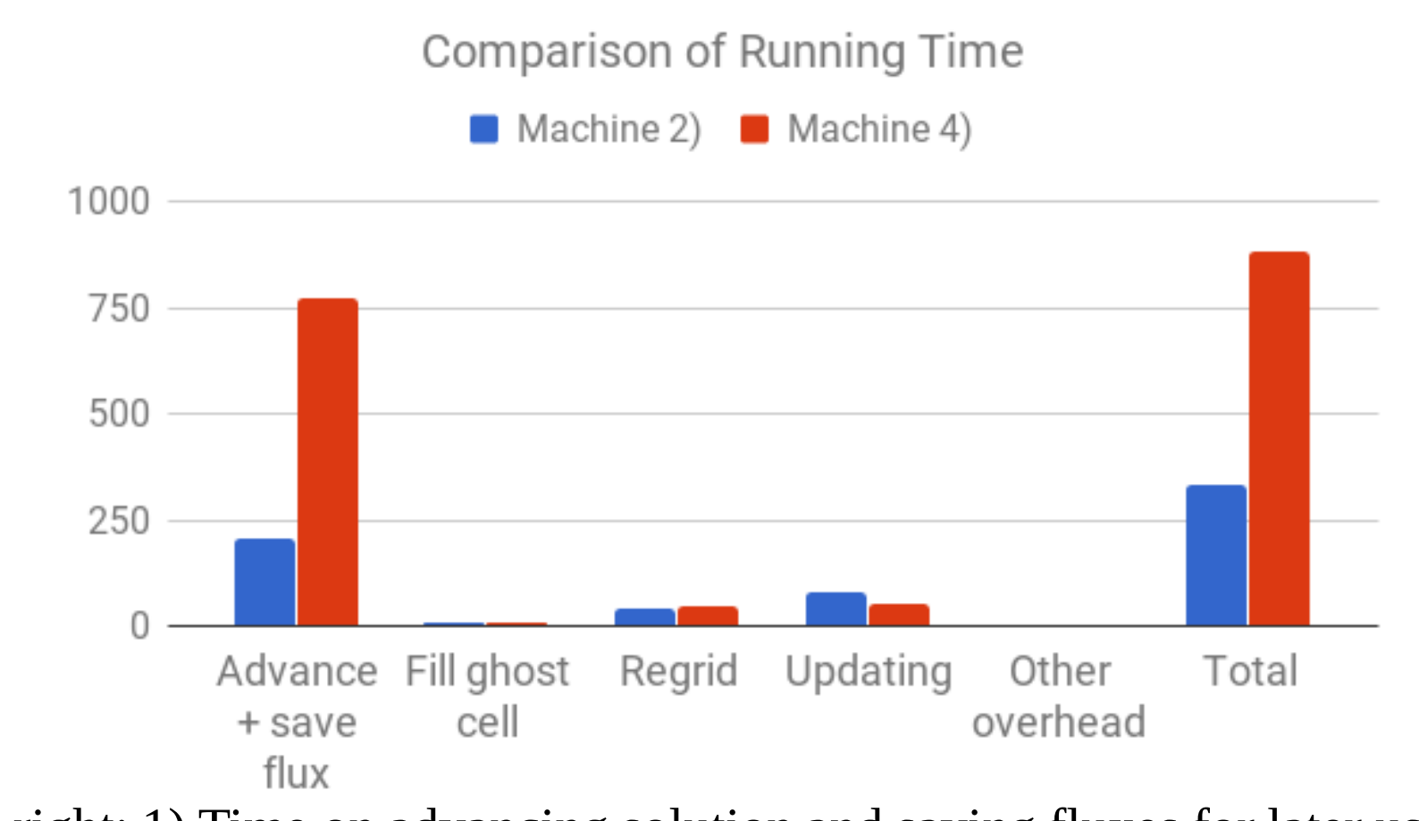
Benchmark and Results

- **PDEs solved:**
  A linearized 2D acoustic equation:
  $$q_t + Aq_x + Bq_y = 0 \quad \text{where} \quad q = \begin{bmatrix} p \\ u \\ v \end{bmatrix}, \quad A = \begin{bmatrix} 0 & K_0 & 0 \\ \frac{1}{\rho_0} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & K_0 \\ 0 & 0 & 0 \\ \frac{1}{\rho_0} & 0 & 0 \end{bmatrix}.$$
- **Machines used:**
  1. A single NVIDIA Kepler K20x GPU with a 16-core AMD Opteron 6274 CPU running at 2.2 GHz as the host;
  2. A single NVIDIA Pascal 100 GPU with a 20-core Intel E5-2698 CPU running at 2.2 GHz as the host (but only 16 CPU threads are used for fair comparison with others);
  3. A single 16-core AMD Opteron 6274 CPU running at 2.2 GHz;
  4. A single 16-core Intel Xeon E-2650 CPU running at 2.0 GHz;
- **Benchmark setup:**
  A total of 3 levels of AMR grids are used with refinement ratios of 2 between each two levels.
  The computational domain is square with 1000 by 1000 cells on the base level, leading to an effective 4000-by-4000-cell resolution on the finest level
- **Simulation results and performance:**



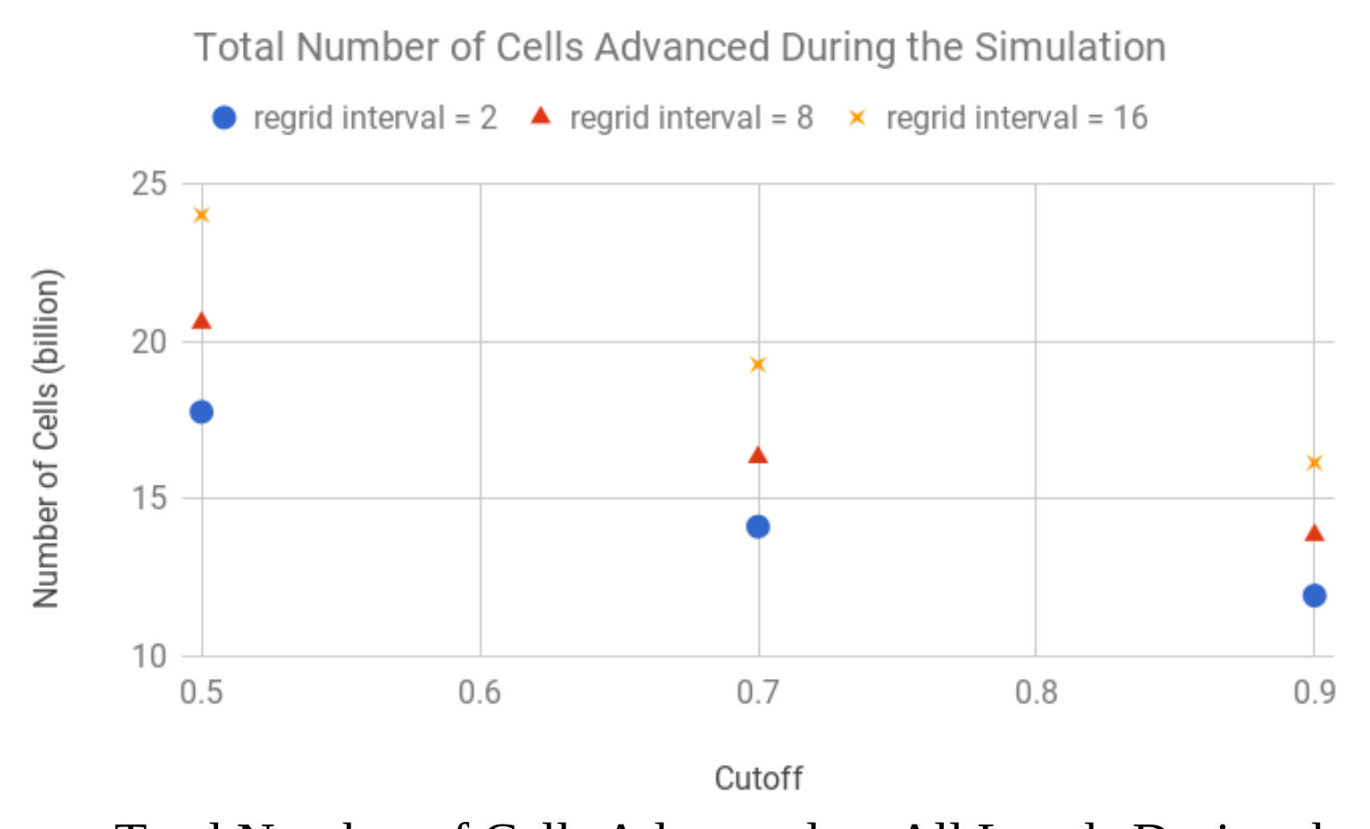
Snapshots of the simulation at t = 0 (top) and t = 0.7 (bottom). (a) and (b) show pressure field with only level 1 AMR grid patches; (c) and (d) show u-velocity field with only level 2 AMR grid patches; (e) and (f) show v-velocity field with only level 3 AMR grid patches. Note that only grid patch edges, not grid cells, are showed.



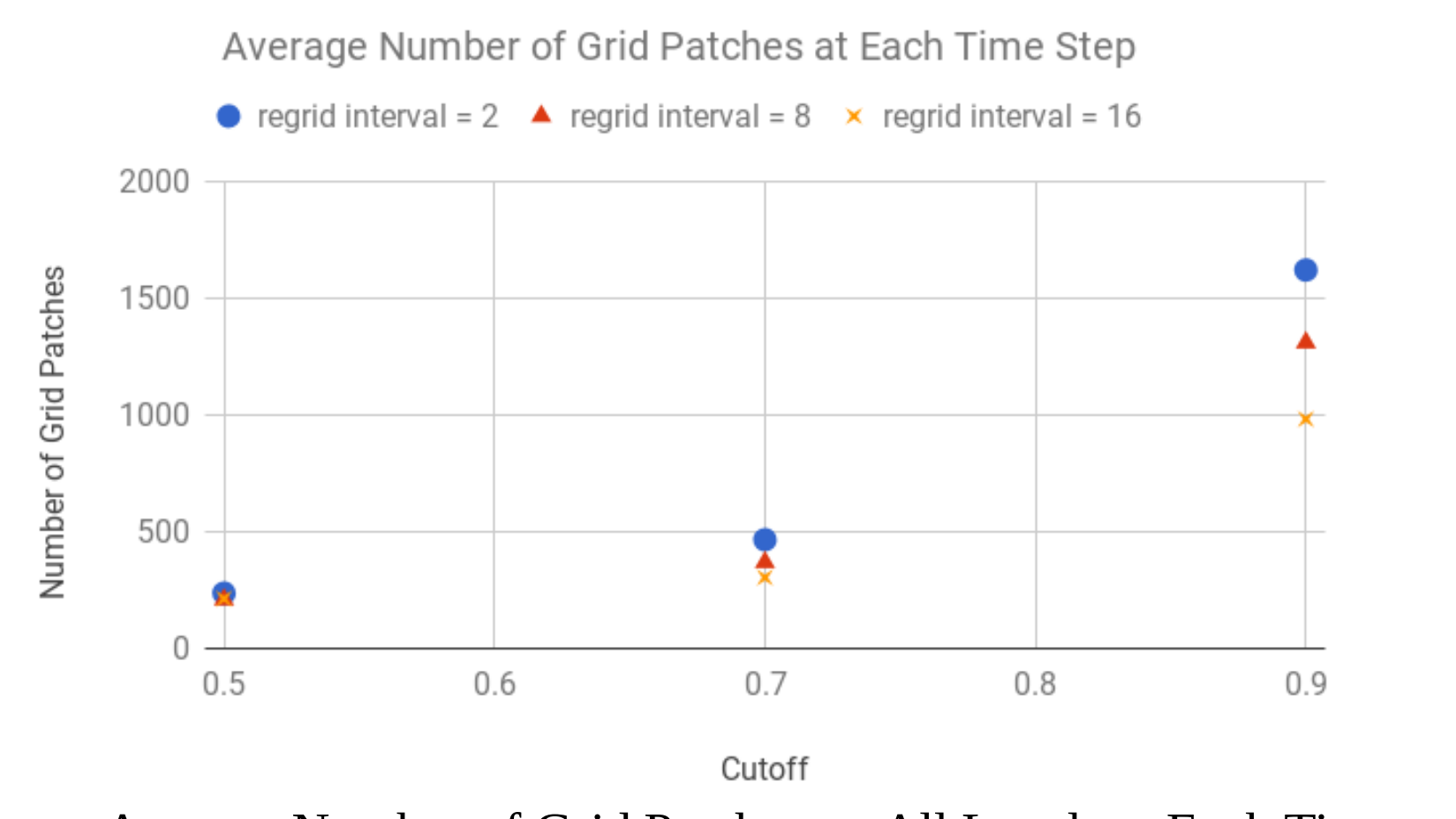


Running Time (in second) of different sections of the code. From left to right: 1) Time on advancing solution and saving fluxes for later use in conservation fix; 2) Time on filling ghost cells; 3) Time on regridding; 4) Time on the updating process; 5) Time on all other overhead; 6) Total running time

- **The Influence of AMR Parameters:**
  - Cutoff: specifies the minimum ratio of the number of flagged cells to all cells on a grid patch. A larger cutoff results in a larger collection of smaller grid patches on each level, chosen such that each grid patch does not contain too many unflagged cells. As a result, there are fewer cells in total.
  - Regrid interval: specifies how frequent the regridding process is conducted. If regrid interval is *a*, usually an extra layer of *a* cells surrounding the original flagged cells will be flagged during the regridding process, such that the waves in the solution do not propagate beyond the refined region before the next regridding process. As a result, each grid patch is approximately *a* cells wider at each side and there are more cells in the entire domain.

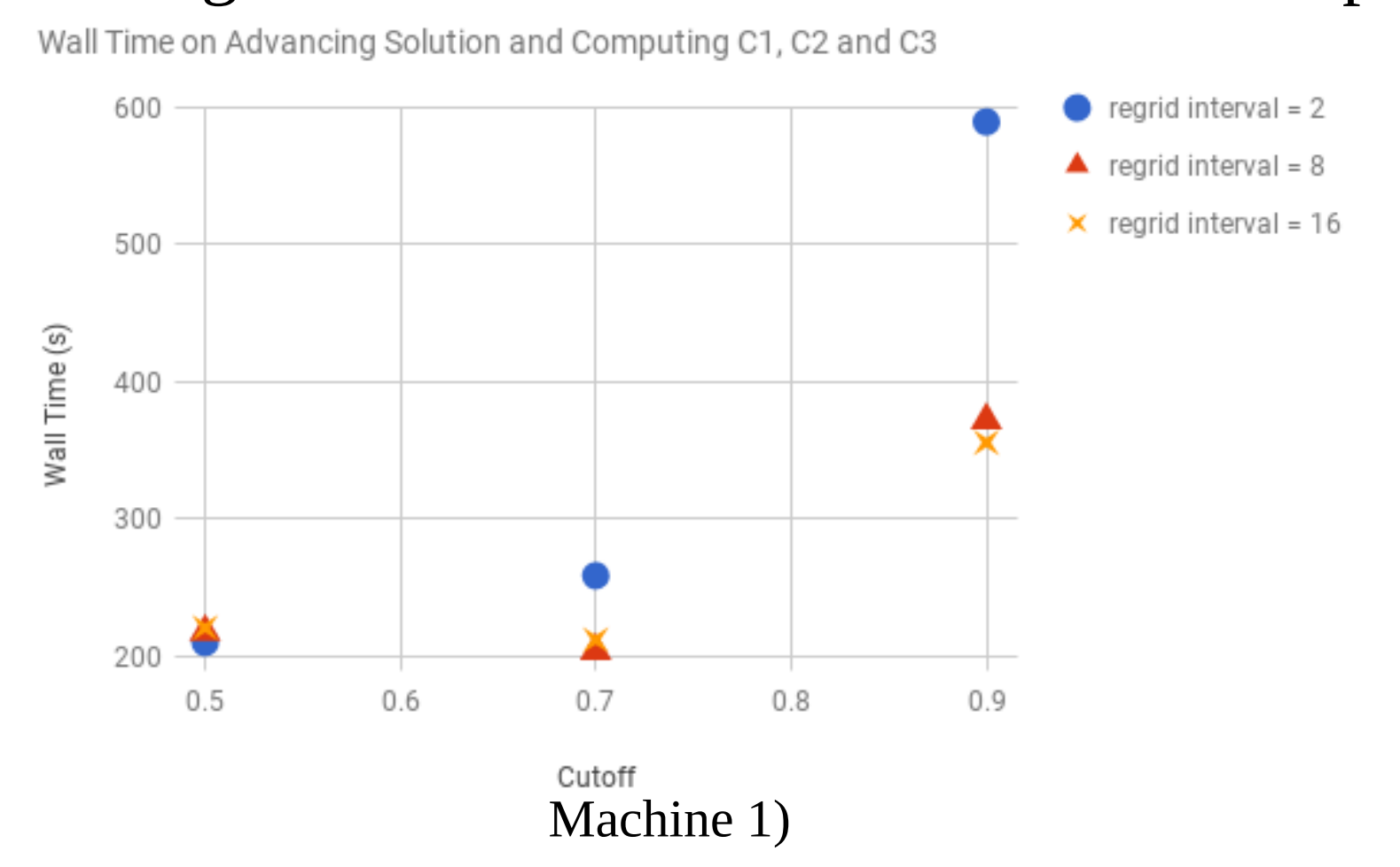


Total Number of Cells Advanced on All Levels During the Entire Simulation.

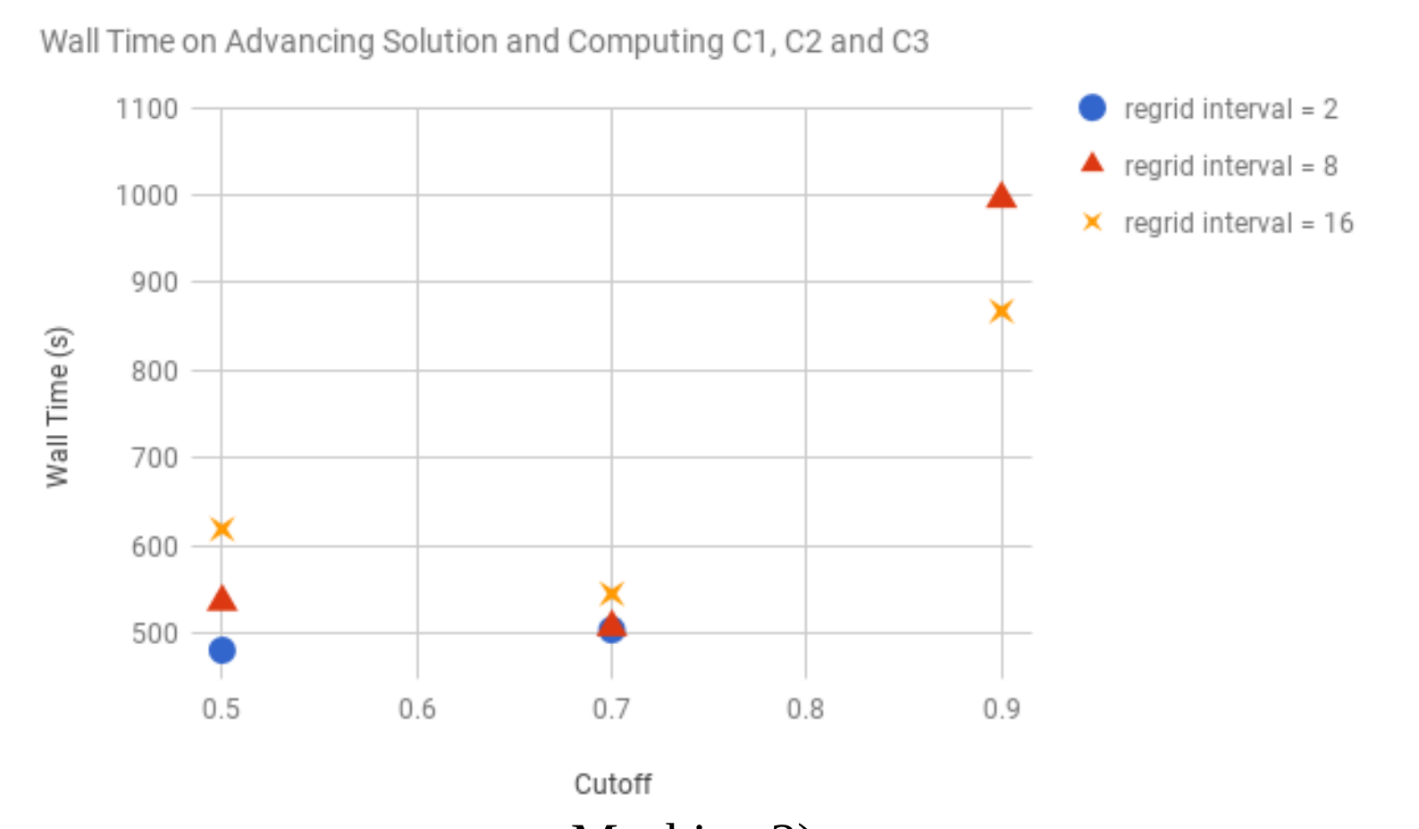


Average Number of Grid Patches on All Levels at Each Time Step.

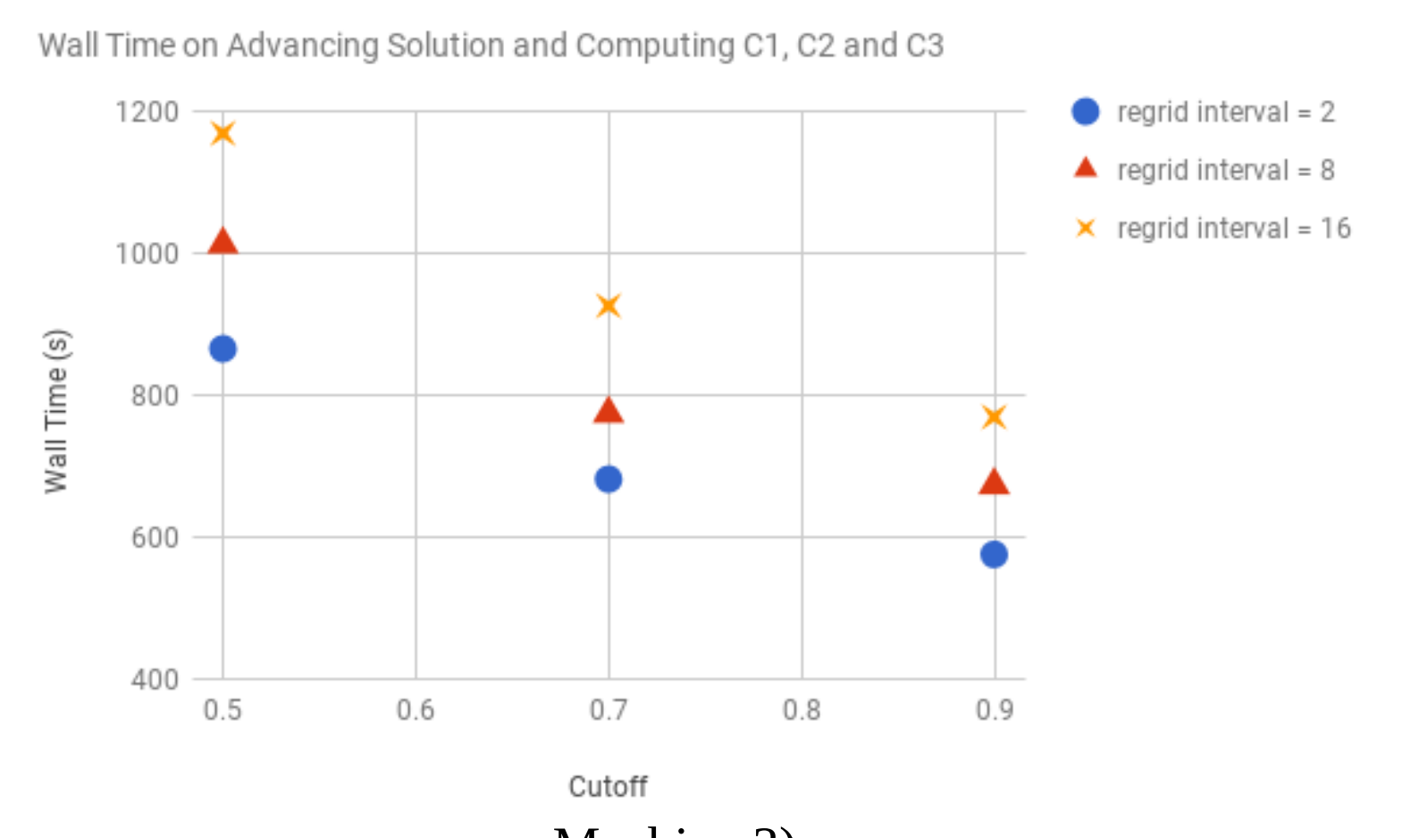
Running time for different choices of the two parameters:



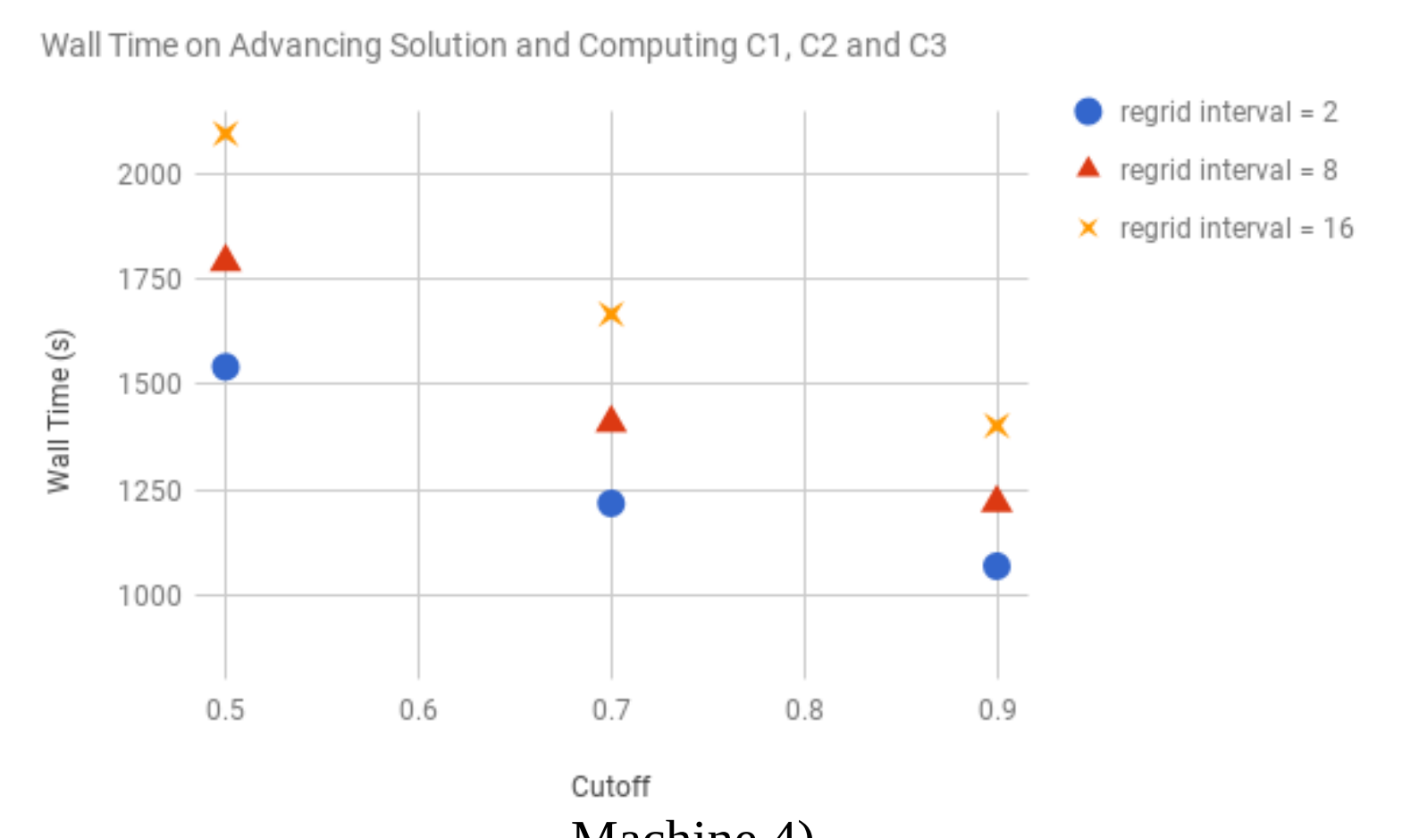
Machine 1)



Machine 2)



Machine 3)



Machine 4)