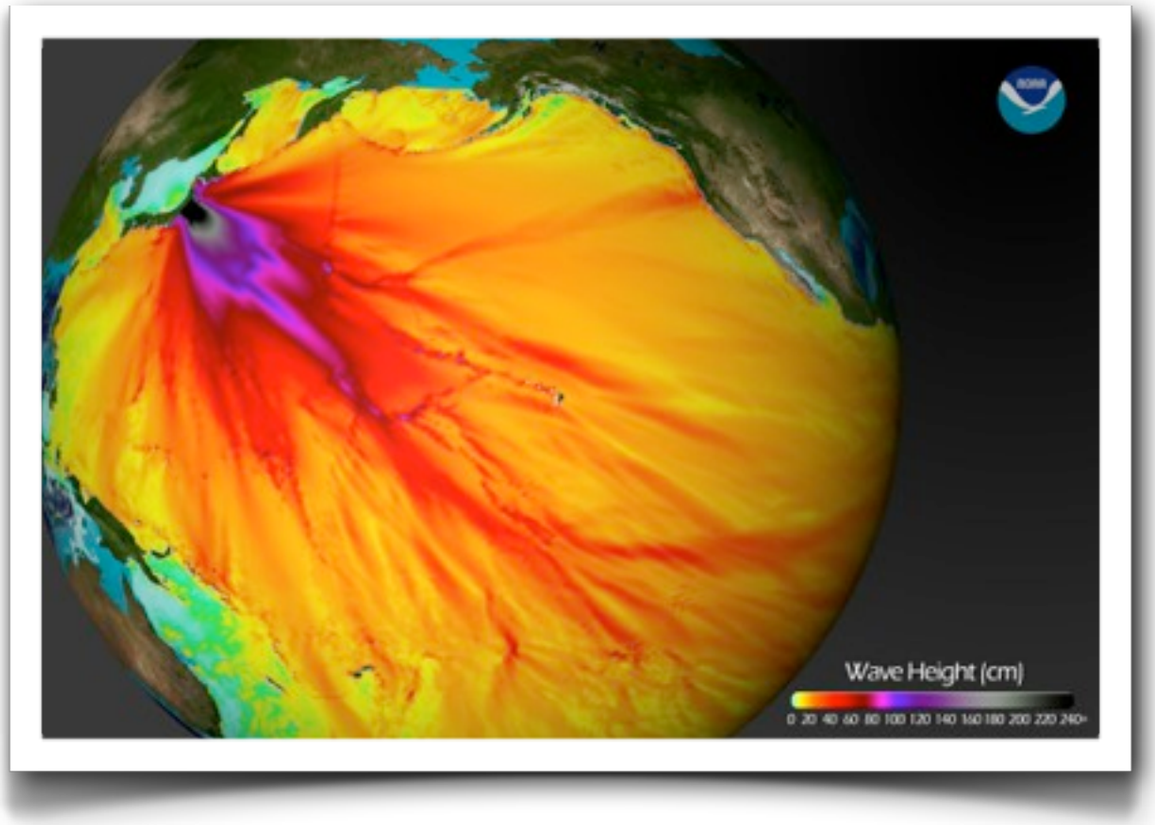
GPU-acceleration of a Finite Difference Method for Tsunami Simulation

Junle Jiang

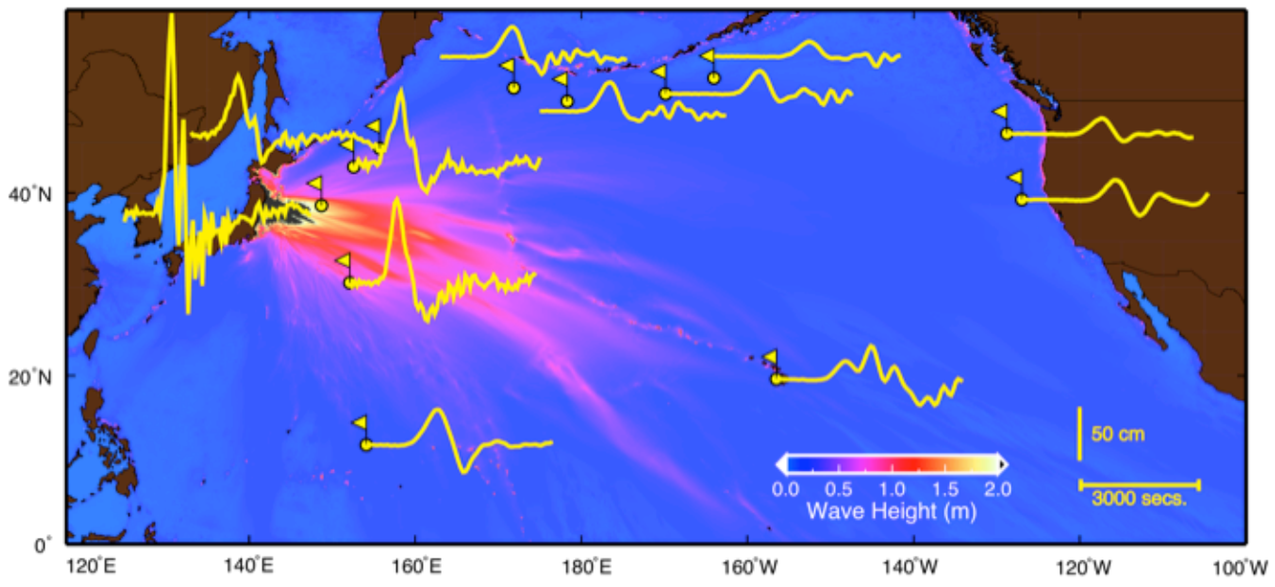
Geophysics
Geological and Planetary Sciences
Caltech

ACM/CS 114
Parallel Algorithms
for Scientific Applications



Motivation

- Numerical simulations of natural hazards, such as earthquake-induced tsunami
  - High demand on computational resources
  - Rapid analysis and response
  - e.g., tsunami produced by 2011 Mw 9.0 Tohoku-oki megathrust earthquake
    - Hit the nearest coastal regions in 30 min; arrived at California later in about 10 hours
    - In sequential implementation with relatively simple physics, simulation takes tens of hours, comparable to or even longer than the physical time



Common observations
1. Time series at open ocean buoys
2. Maximum surface uplift (radiation pattern)

Tsunami propagation across Pacific Ocean (Simons, et al., 2011)

Motivation

- Parallelization of the large-scale simulation of tsunami propagation is desirable
  - Speedup of the computational simulation
  - Time-appropriate for tsunami rapid analysis and early warning
  - Facilitate the intensive study of parameters for tsunamigenic source
- CUDA (Compute Unified Device Architecture, NVidia) programming on GPU (Graphical Processing Unit) is used as the parallelization tool in this project, due to the superb data-processing power of GPU
- This project provides an initiative for the practice on the GPU architecture and evaluates feasibility and prospect for GPU-accelerated simulations of this problem

Physical Problems

- Tsunami generation (instantaneous seafloor deformation)
- Tsunami propagation
  - Shallow water wave equation (equations for conservation of mass/ momentum integrated over depth in long-wavelength approximation)

Mass equation

$$\frac{\partial \eta}{\partial t} + \left\{ \frac{\partial P}{\partial x} + \frac{\partial Q}{\partial y} \right\} = -\frac{\partial h}{\partial t}$$

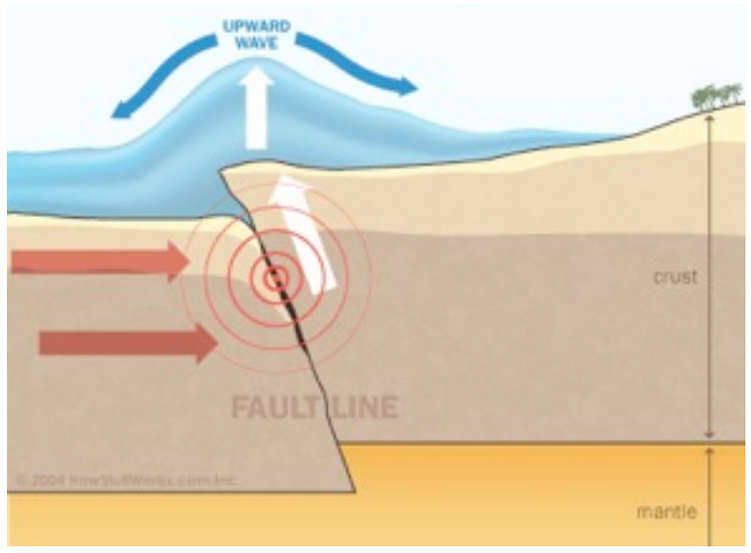
Moment equation

$$\frac{\partial P}{\partial t} + gh\frac{\partial \eta}{\partial x} - fQ = 0$$

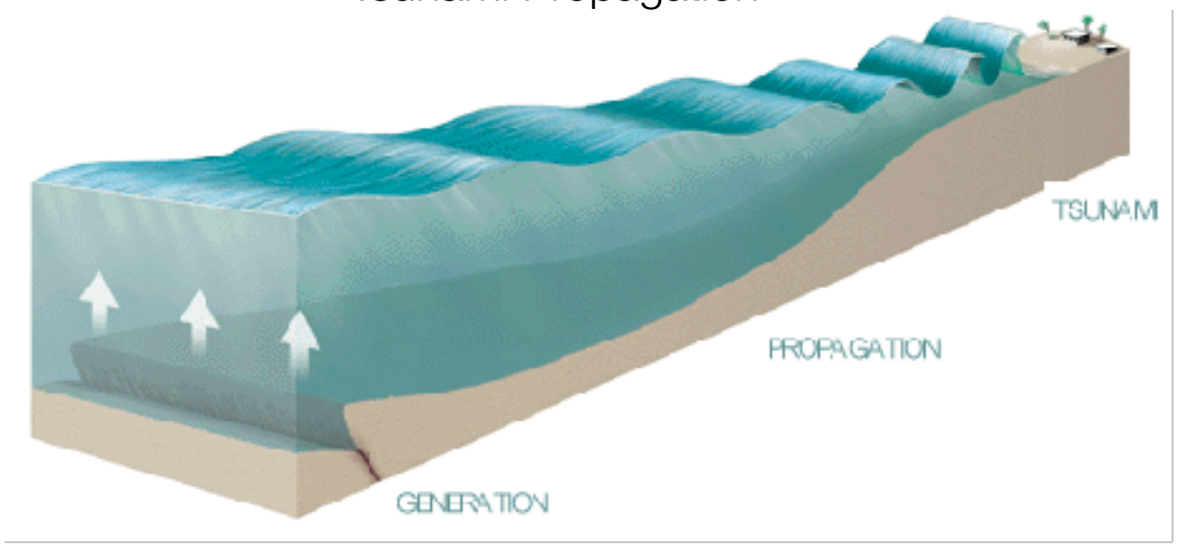
$$\frac{\partial Q}{\partial t} + gh\frac{\partial \eta}{\partial y} + fP = 0$$

h: bathymetry depth
η: water displacement
P: volume flux in X direction
Q: volume flux in Y direction

Tsunami Generation



Tsunami Propagation

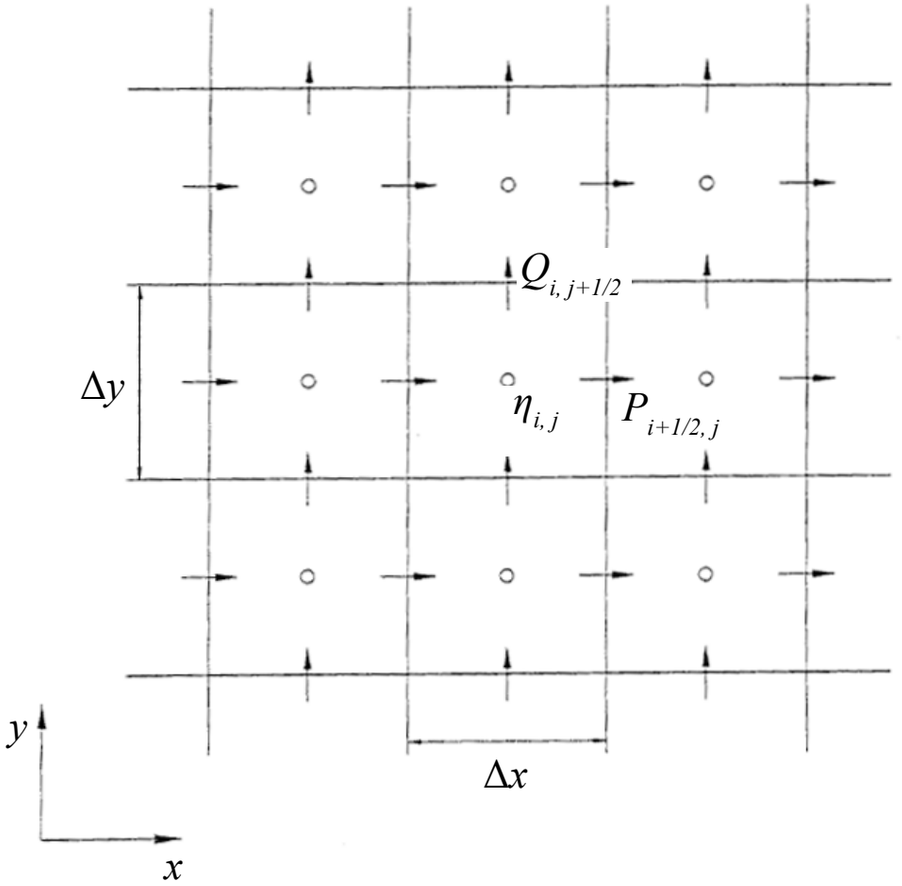


Numerical Methods

- COMCOT (COrnell Multi-grid COupled Tsunami model, Wang et al., 2009)
  - Staggered-Grid leap-frog finite difference method
  - Linear/Nonlinear shallow water equation in Cartesian/spherical coordinates
  - Other ingredients: different boundary conditions, multi-grid for spatially different resolution
- In this project
  - Linear shallow water with open boundary conditions in Cartesian/spherical coordinates are implemented for simplicity
  - As data output, time series at a particular station and full-ocean water surface displacement are recorded every certain time interval
  - Improving/modify coding for CUDA-compatibility

Numerical Methods

- Leap-frog Scheme & Staggered Grid
  - The free surface displacement, η, is configured at the center of a grid cell (i, j)
  - The leap-frog scheme calculates the free surface elevation at grid point (i, j) on the time step (n + 1/2) in fully explicit way using P, Q and η from the previous time step (n - 1/2).
  - Volume flux components, P and Q, are configured at the centers of four edges of the grid cell, i.e., $P_{i-1/2,j}$, $P_{i+1/2,j}$, $Q_{i,j+1/2}$ and $Q_{i,j-1/2}$.
  - Calculations for the free surface displacement and the volume flux components are staggered both in space and time
  - Higher accuracy than 2nd-order centered FD



Numerical Methods

- Leap-frog time step on a staggered grid
  - (a) 2D field η at time step *n*+1/2 is updated using its previous value and the spatial differences of *P* and Q, respectively, at time step *n* by Equation (4);

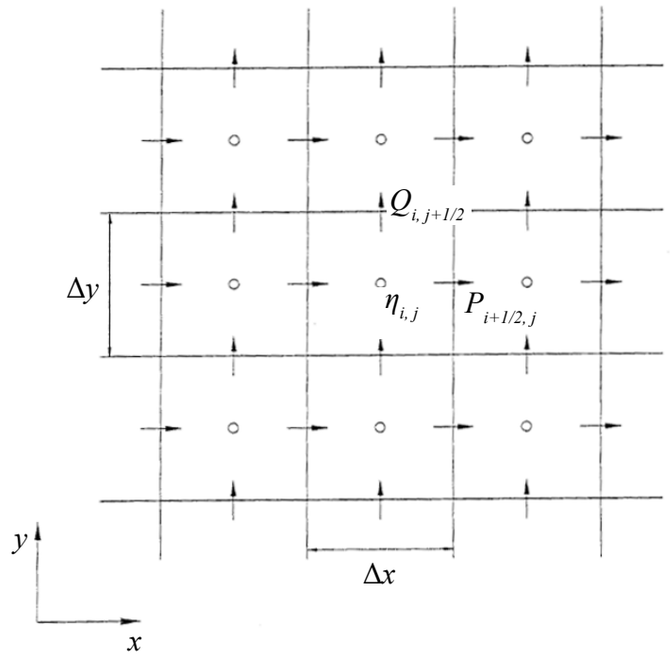
$$\eta_{i,j}^{n+1/2} = \eta_{i,j}^{n-1/2} - r_x(P_{i+1/2,j}^{n} - P_{i-1/2,j}^{n}) - r_y(Q_{i,j+1/2}^{n} - Q_{i,j-1/2}^{n})$$

  - (b) open boundary conditions (i.e., without wave reflection) are applied for variables;
  - (c) *P* and *Q*, at the time step *n*+1 are obtained by updating with the spatial difference in the new η at time step *n*+1/2 and are also dependent on the local water depth (bathymetry);

$$Q_{i,j+1/2}^{n+1} = Q_{i,j+1/2}^{n} - r_y g H_{i,j+1/2}^{n+1/2}(\eta_{i,j+1}^{n+1/2} - \eta_{i,j}^{n+1/2})$$

$$P_{i+1/2,j}^{n+1} = P_{i+1/2,j}^{n} - r_x g H_{i+1/2,j}^{n+1/2}(\eta_{i+1,j}^{n+1/2} - \eta_{i,j}^{n+1/2})$$

  - (d) the next time step starts from (a).

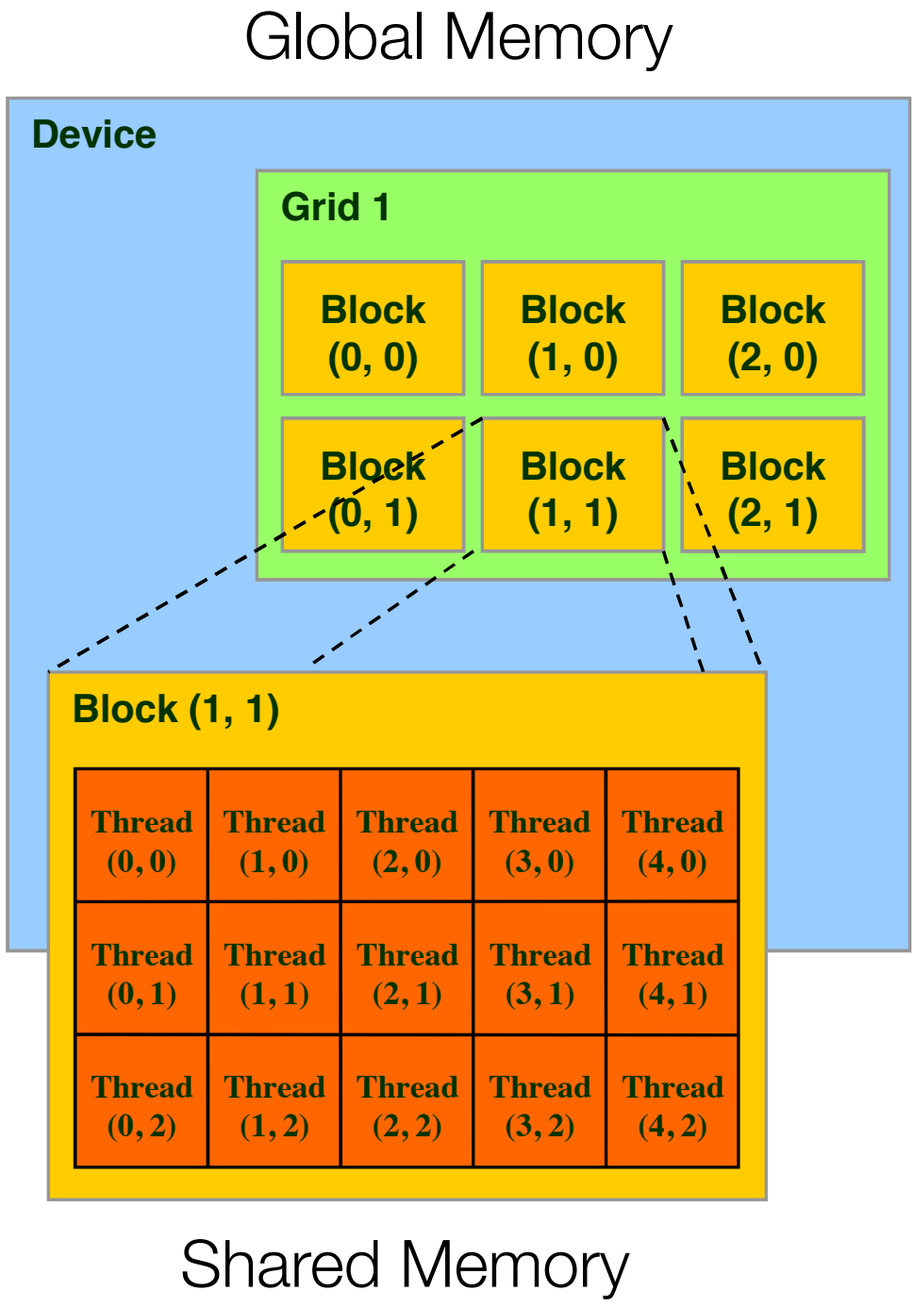


Parallelization with CUDA on GPU

- GPU (Graphical Processing Unit) has recently become an important choice for parallel computing problems. By design, its architecture allows more transistors to be devoted to to data processing rather than data caching and flow control (NVIDIA, 2011)
- The GPU memory hierarchy
  - Local, texture and Global memory
  - **Local shared memory** has a extraordinary access speed, almost equivalent to the velocity of accessing registers
  - Texture memory (read-only) is slower than local but twice faster than global memory
  - **Global memory** is readable and writable, allowing communications with all the processors and threads

GPU Memory Structure

- CUDA is a popular choice for the programming languages on GPU
  - Concurrent multi-threads within a set of multi-processor cores (multi-blocks)
  - Shared memory for data-processing on threads within the same block
- Global memory for all blocks and threads
- Fine and coarse levels of parallelism are inherently achieved in this configuration

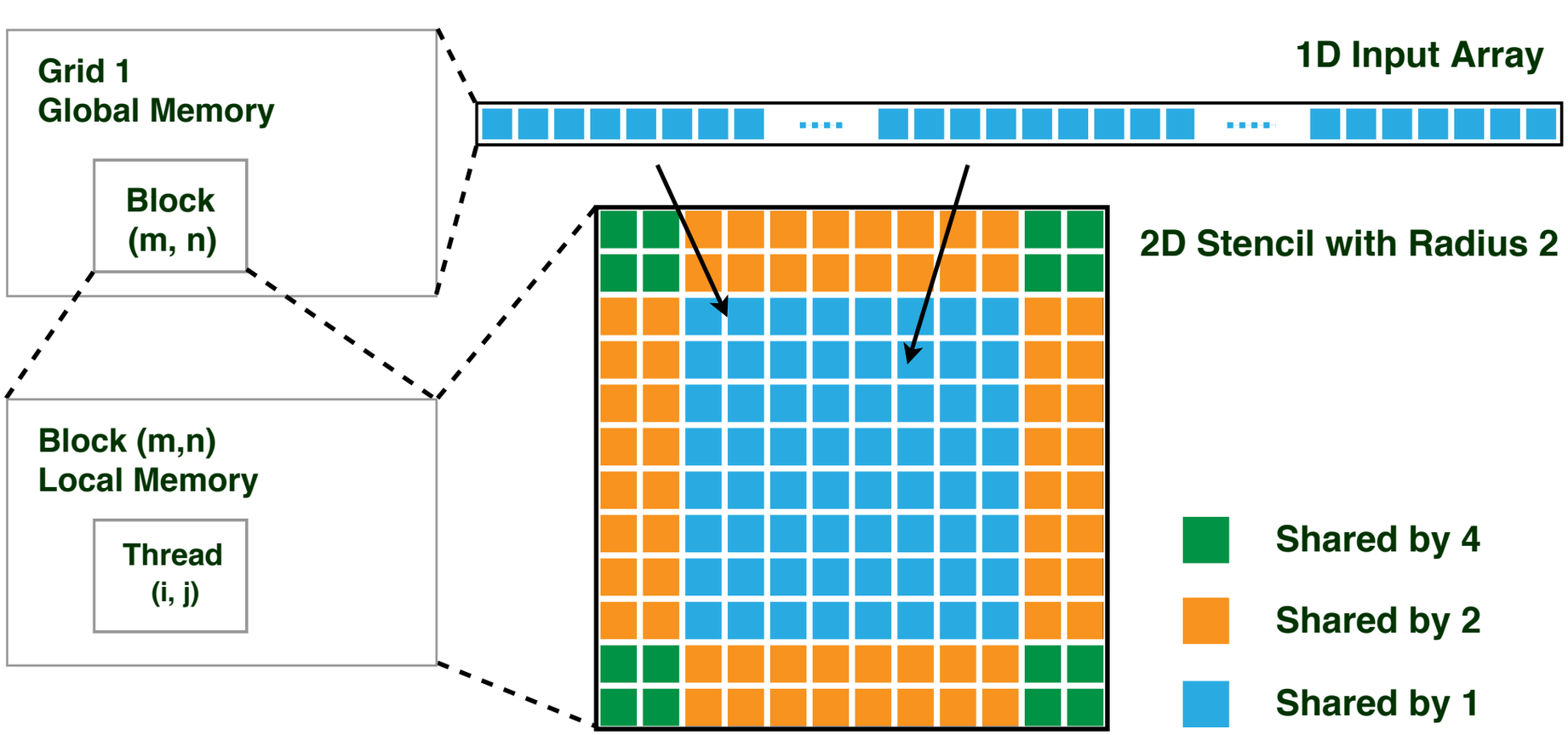


Parallelization Strategy for COMCOT

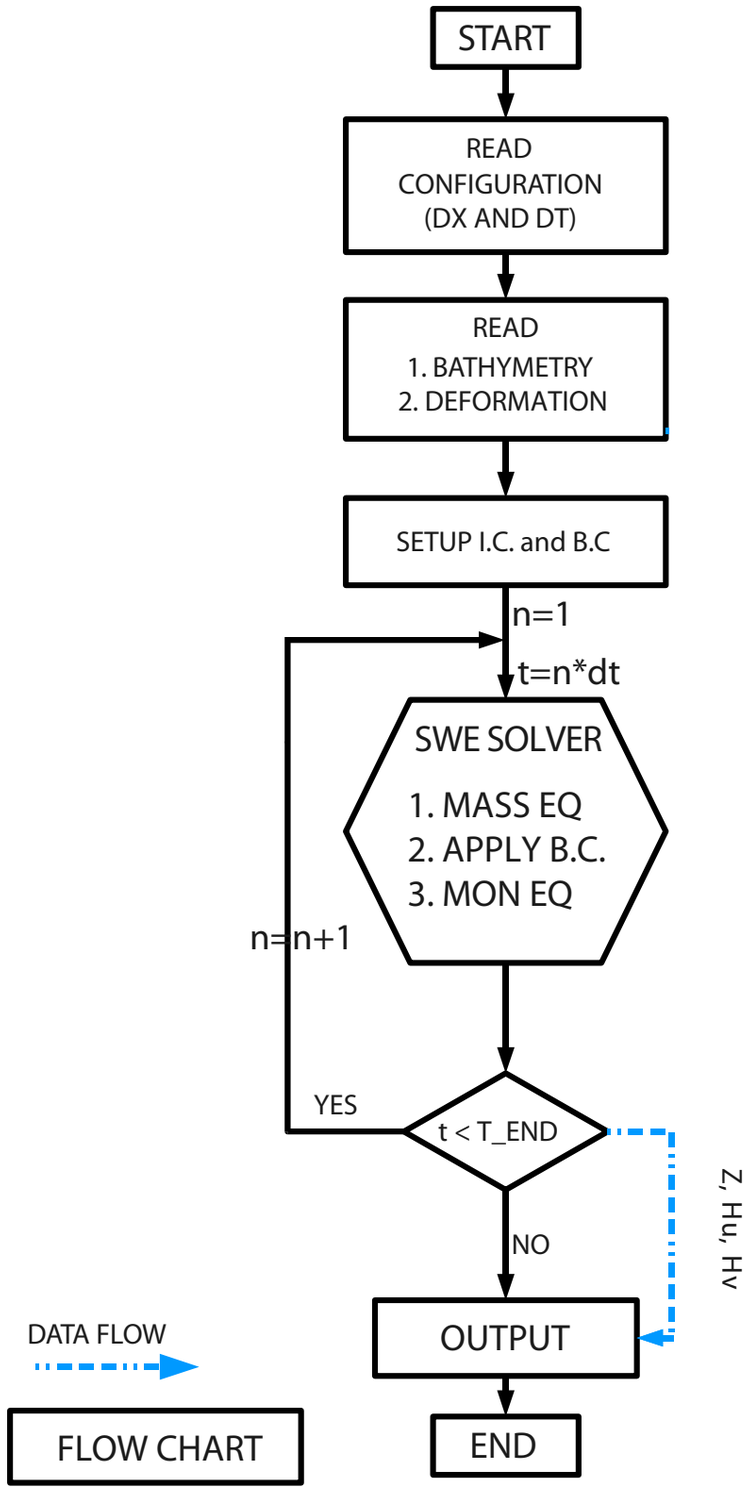
- **Parts for parallelization** : The update of 2D field of variables with time
- **Finest-grained job**
  - Update the field variables (η, *P* and *Q*) with time on each grid point
    => assigned to each thread in a block
- **Coarser-grained job**
  - Update a subdomain of the 2D fields over time
    => assigned to each block on the device
- **Communication**
  - Threads within a block have shared memory that's fast to access but different blocks cannot communicate with each other.
  - Therefore all values of variables will first be copied from the global GPU memory to the locally shared memory in each block and then processed within the block

Computational Procedures

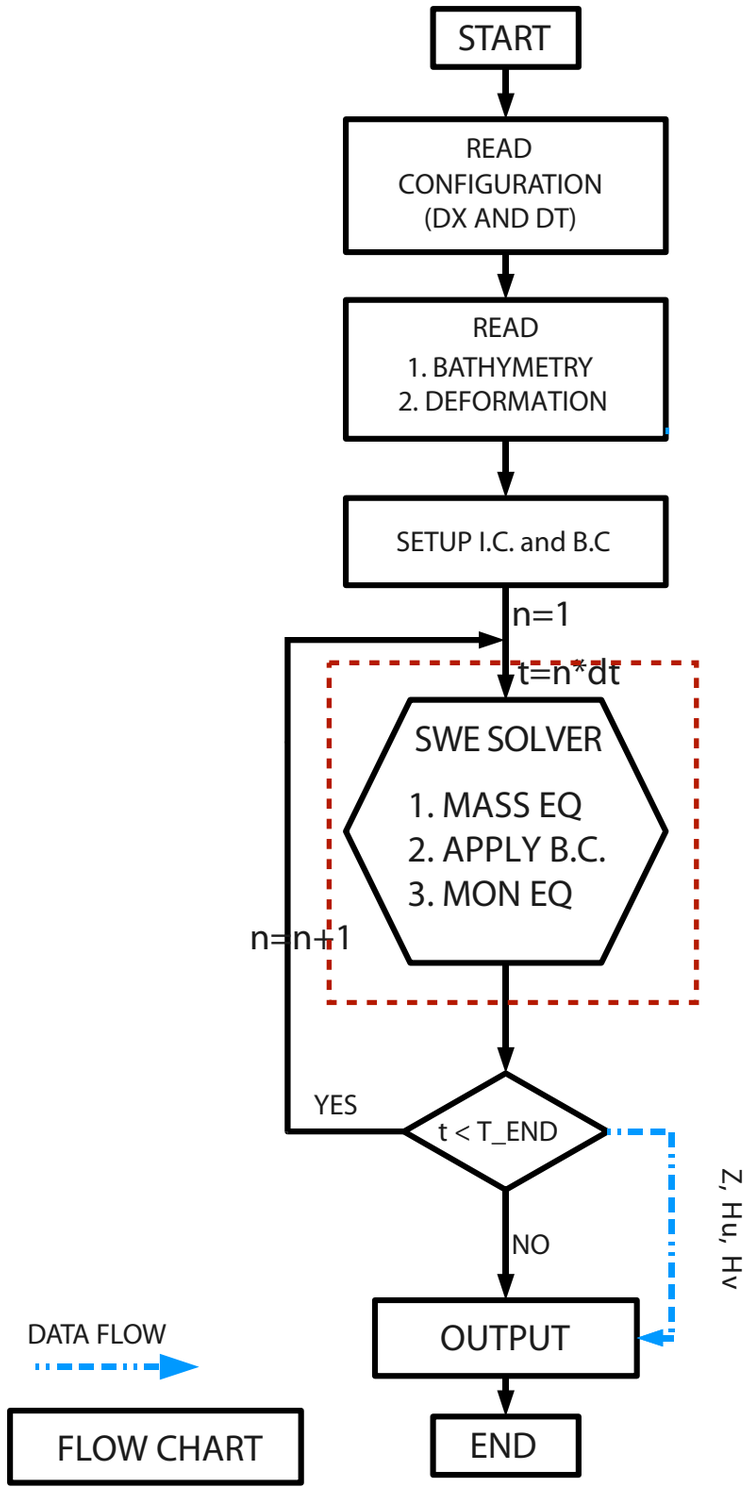
- Partition data in global memory into local shared memory
  - 1D array with 2D index mapping are stored in global memory
  - Local shared memory stores subdomain; each thread processes one element; elements for real output are read once; marginal elements are read several times from the global memory;



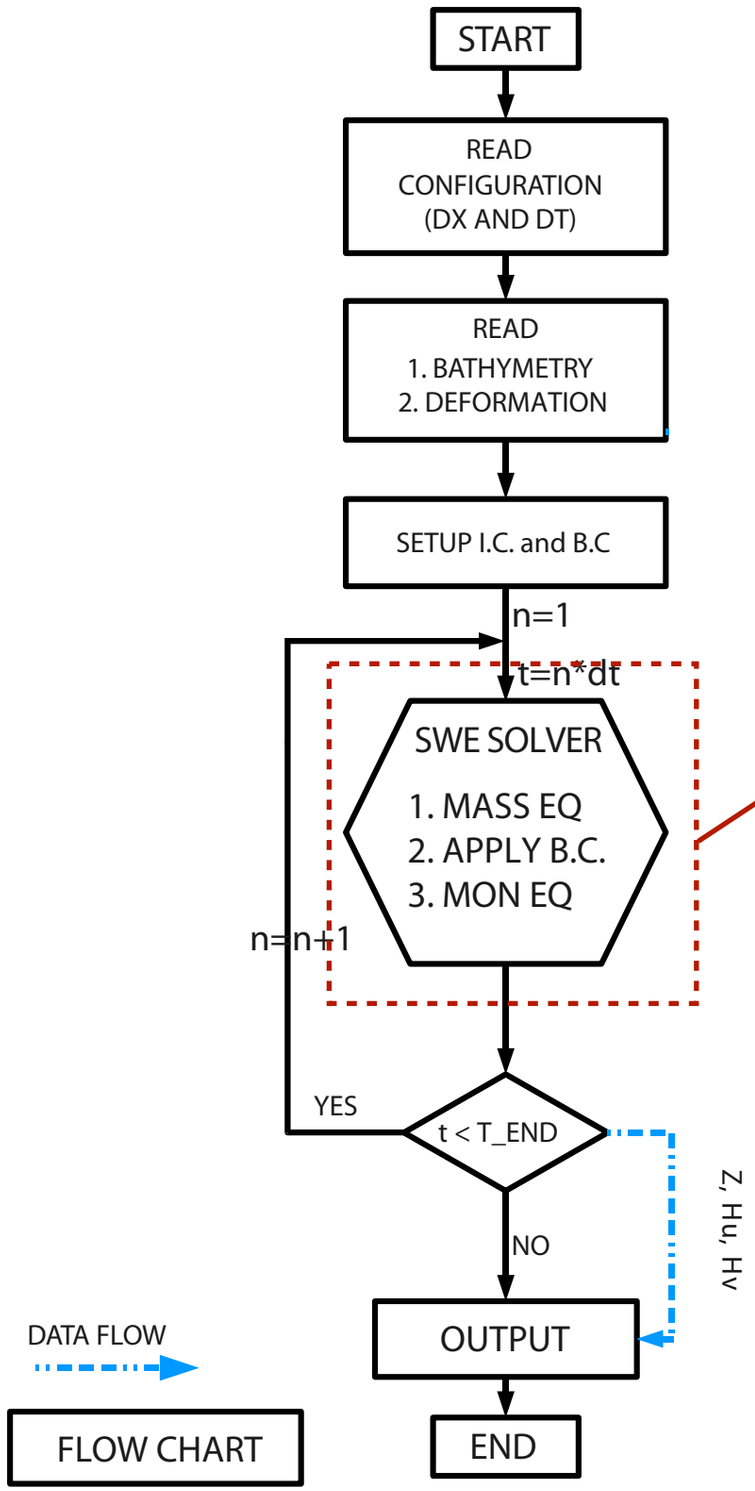
Computational Procedures



Computational Procedures



Computational Procedures



GPU implementation

(a) Copy variables η, P and Q at the current the time step from the CPU to GPU (bathymetry depth H_P and H_Q), using

```
cudaMemcpy(h, lo.h, num_bytes, ...
cudaMemcpyHostToDevice)
```

(b) Solving the equation of mass conservation by invoking the kernel `cuda_mass<<<grid,block>>>(h,P,Q,...)`.

Within the GPU, P and Q are loaded consecutively from global memory to local memory within each block to perform the calculations of spatial derivatives, one operation per thread. h is finally updated in global memory and ready for copying back to CPU;

(c) With updated values on h, open boundary conditions are currently applied in the CPU;

(d) Solving the equations for momentum conservations by invoking the kernel

`cuda_moment<<<grid, block>>>(VARIABLES)`.
Similar to (b);

(e) The next time step starts from (a).

Comparisons of Simulation Time (second)

Grid Dimension	1000×1000	2000×2000	5000×5000
CPU	88	396	2955
CPU (threads)	39	120	764
CPU-GPU	14	38	216

- Simulation time is for 1 hour in physical world;
- 100 threads on CPU and N×N threads on GPU (GeForce GTX580)
- The speed-up could be limited to the output of variables at every time interval, and could be potentially larger with improved implementation

Conclusion

- Parallelization of the finite difference method for simulation of tsunami propagation produces a speed up of 15 times as compared to the sequential version, and about 4 times faster than its CPU-thread counterpart
- Such speedup gives good prospect for its potential application in rapid analysis and early warning of tsunami, and exploration of physical parameters for tsunamigenic sources
- Future work includes improved implementation and incorporation of more functionalities (multi-grids, nonlinear terms and etc.)