

GAMER: a GPU-accelerated Adaptive-MEsh-Refinement Code for Astrophysics

H. Y. Schive (薛熙于)

*Graduate Institute of Physics, National Taiwan University
Leung Center for Cosmology and Particle Astrophysics (LeCosPA)*

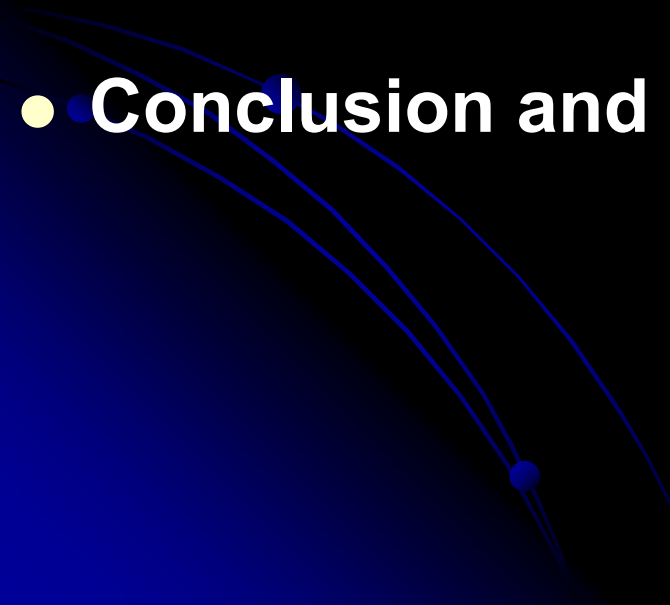
T. Chiueh (闕志鴻), Y. C. Tsai (蔡御之)

*Graduate Institute of Physics, National Taiwan University
Leung Center for Cosmology and Particle Astrophysics (LeCosPA)*

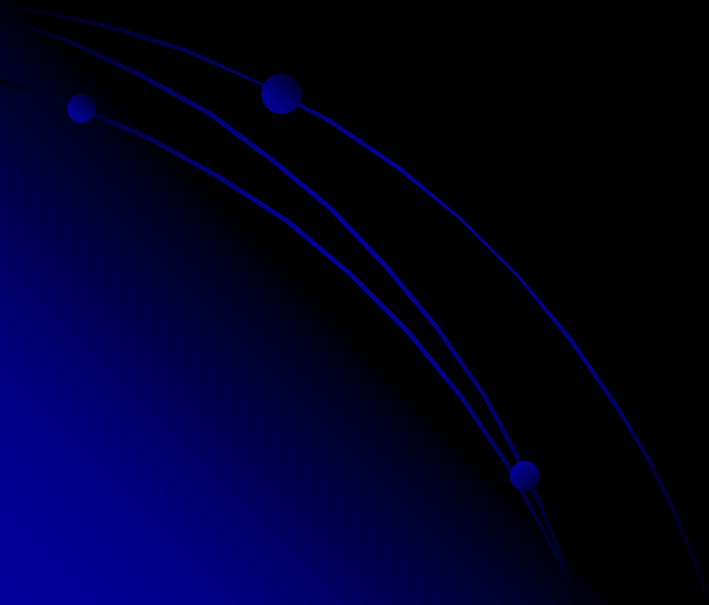


Accelerator-based Computing and Manycore (12/2/2009)

Outline

- Introduction to AMR
 - AMR + GPUs
 - GPU Solvers (Hydrodynamic + Poisson)
 - Optimizations
 - Conclusion and Future Work
- 

Introduction to AMR

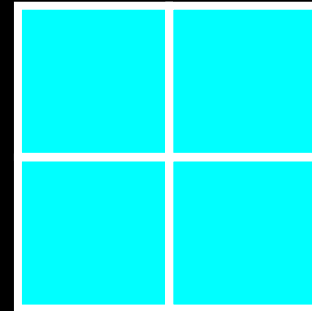


Adaptive-Mesh-Refinement

- **Boring Region :**

- ◆ smooth, empty, low error ...

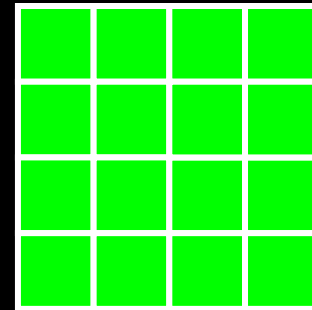
➔ **coarse mesh**



- **Interesting Region :**

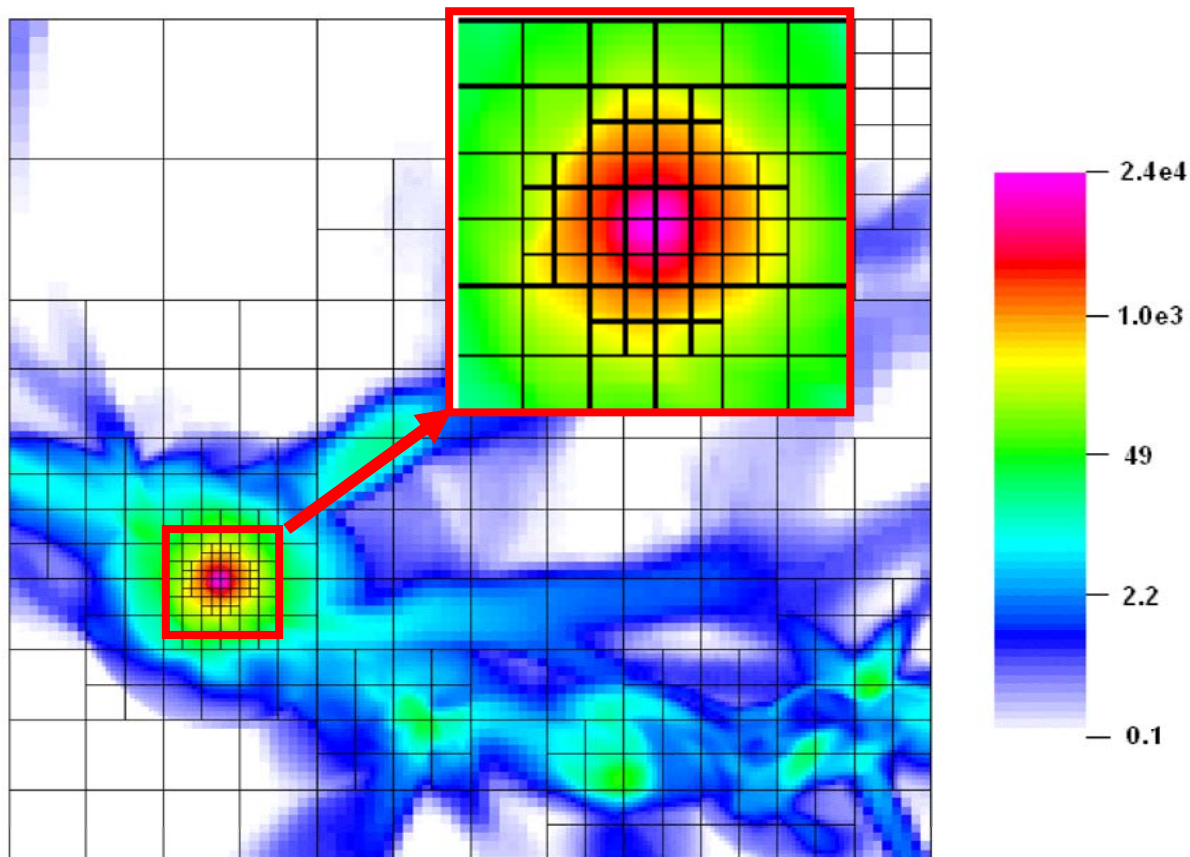
- ◆ high density, high contrast, high error ...

➔ **fine mesh**



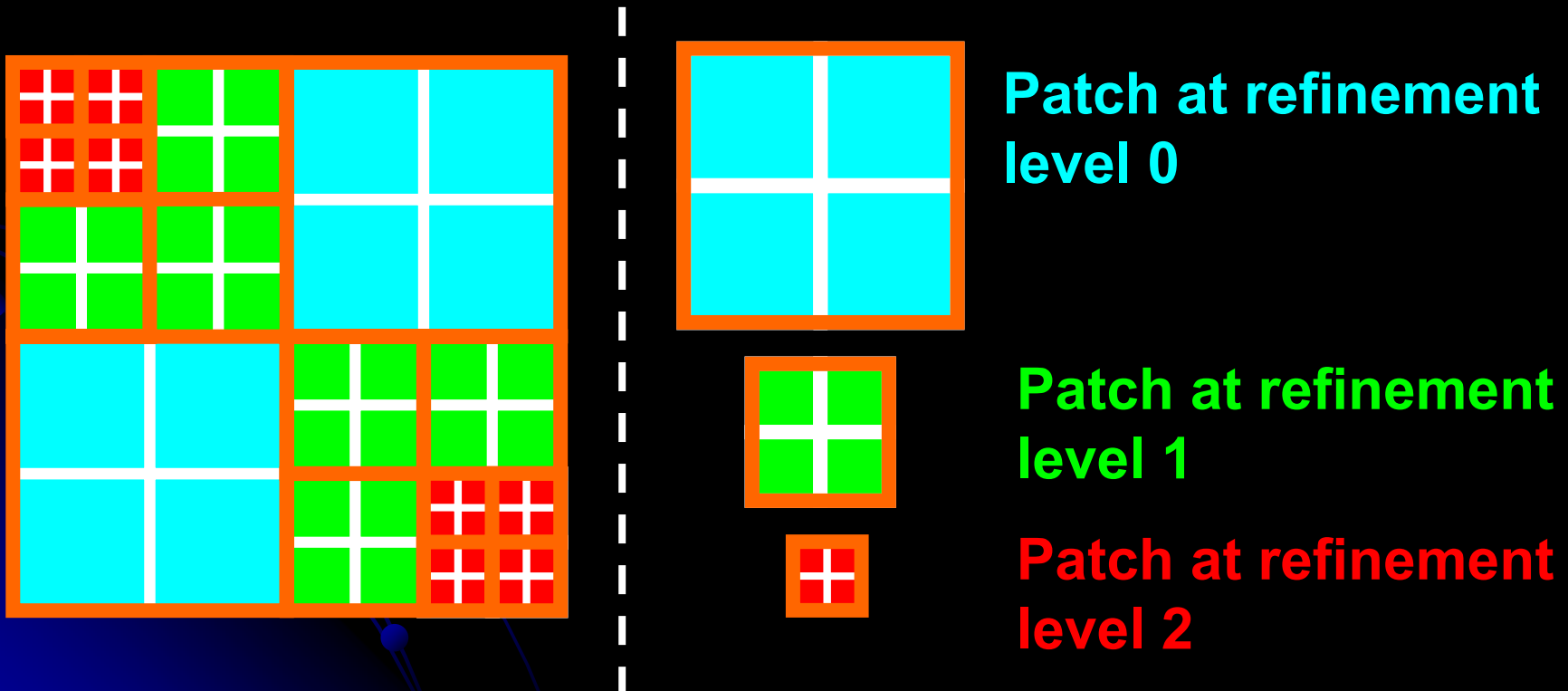
AMR Example

- Refinement criterion : density

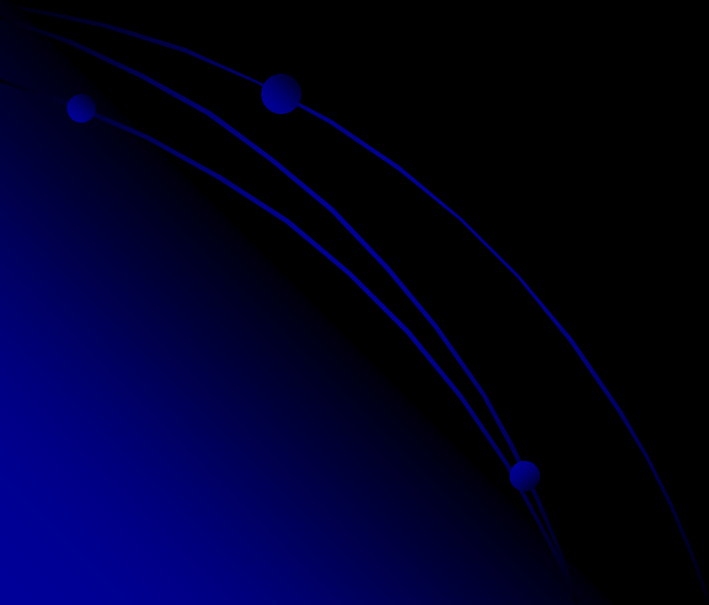


AMR Scheme in GAMER

- ◆ Refinement unit : **patch** (containing a fixed number of cells)
- ◆ Hierarchical oct-tree data-structure



AMR + GPUs



CPU-GPU Collaboration

- Two main tasks in AMR:

1. **Patch construction** : decision making, interpolation, complex data-structure, data assignment ...

~ complicated, but consume less time

➡ CPUs

2. **3-D hydrodynamic + Poisson solvers** :

~ straightforward, but time-consuming

➡ GPUs

➡ feed with hundreds of patches simultaneously

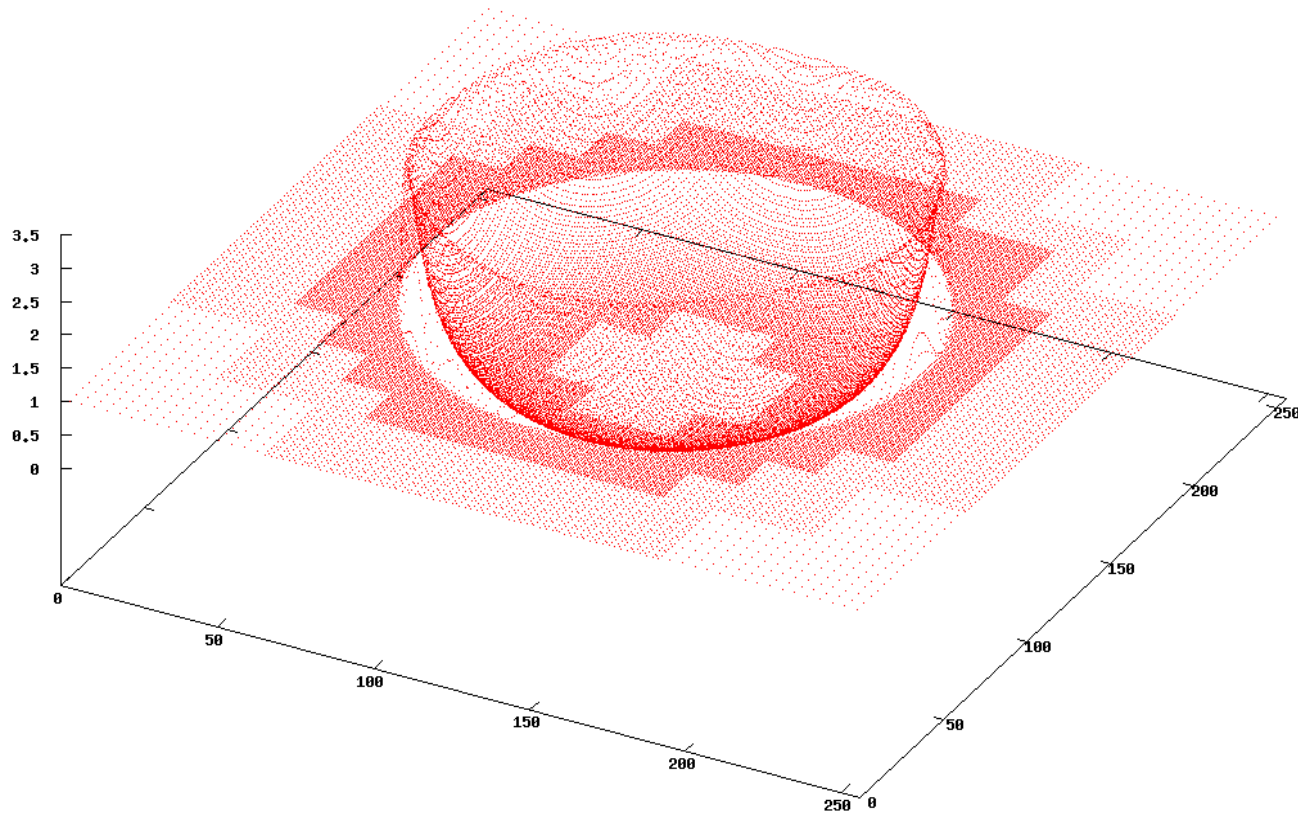
Multi-GPU Acceleration

- Multi-GPU acceleration in GAMER relies on **three parallelization levels**
 1. **Cells within the same patch** \leftrightarrow **Threads within the same thread block**
 - ◆ Store common and frequently reused data (e.g., fluid flux) in the **shared memory**.
 2. **Different patches** \leftrightarrow **Different thread blocks**
 - ◆ The **boundary condition** of each patch is prepared by **CPU** in advance.
 3. **Different sub-domains** \leftrightarrow **Different GPUs**
 - ◆ Rectangular domain decomposition
 - ◆ Data exchange : **MPI**

Example : Blast Wave Test

Density

'Data_000161' u 6:7:4

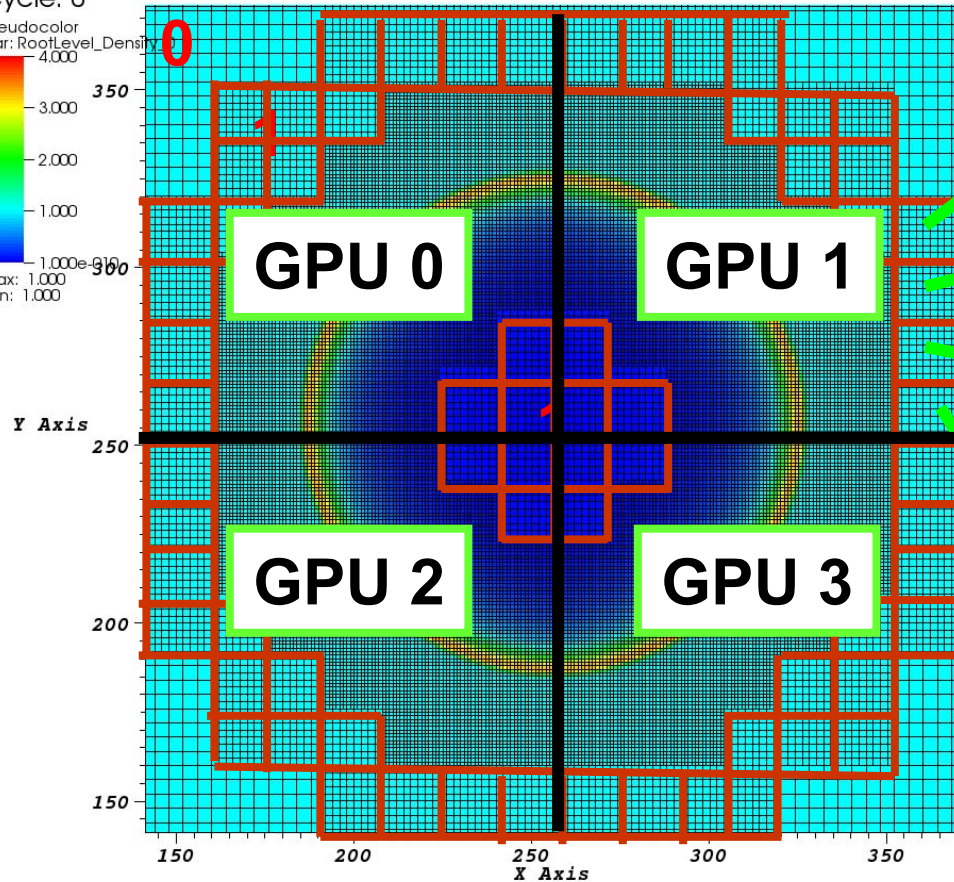
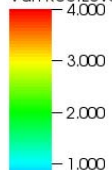


Example : Blast Wave Test

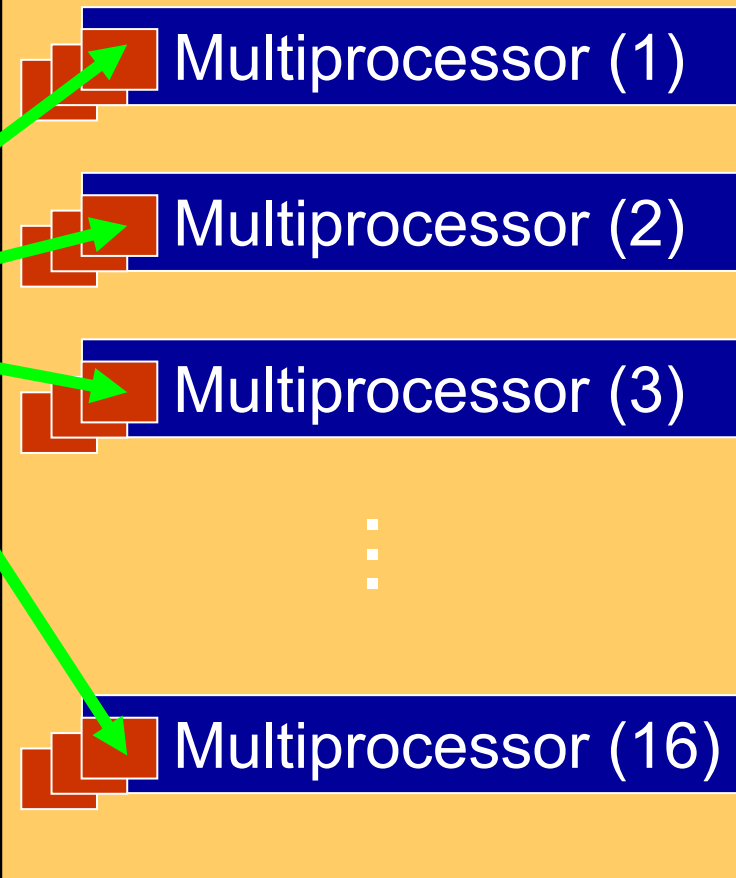
DB: RootLevel.silo

Cycle: 0

Pseudocolor
Var: RootLevel_Density

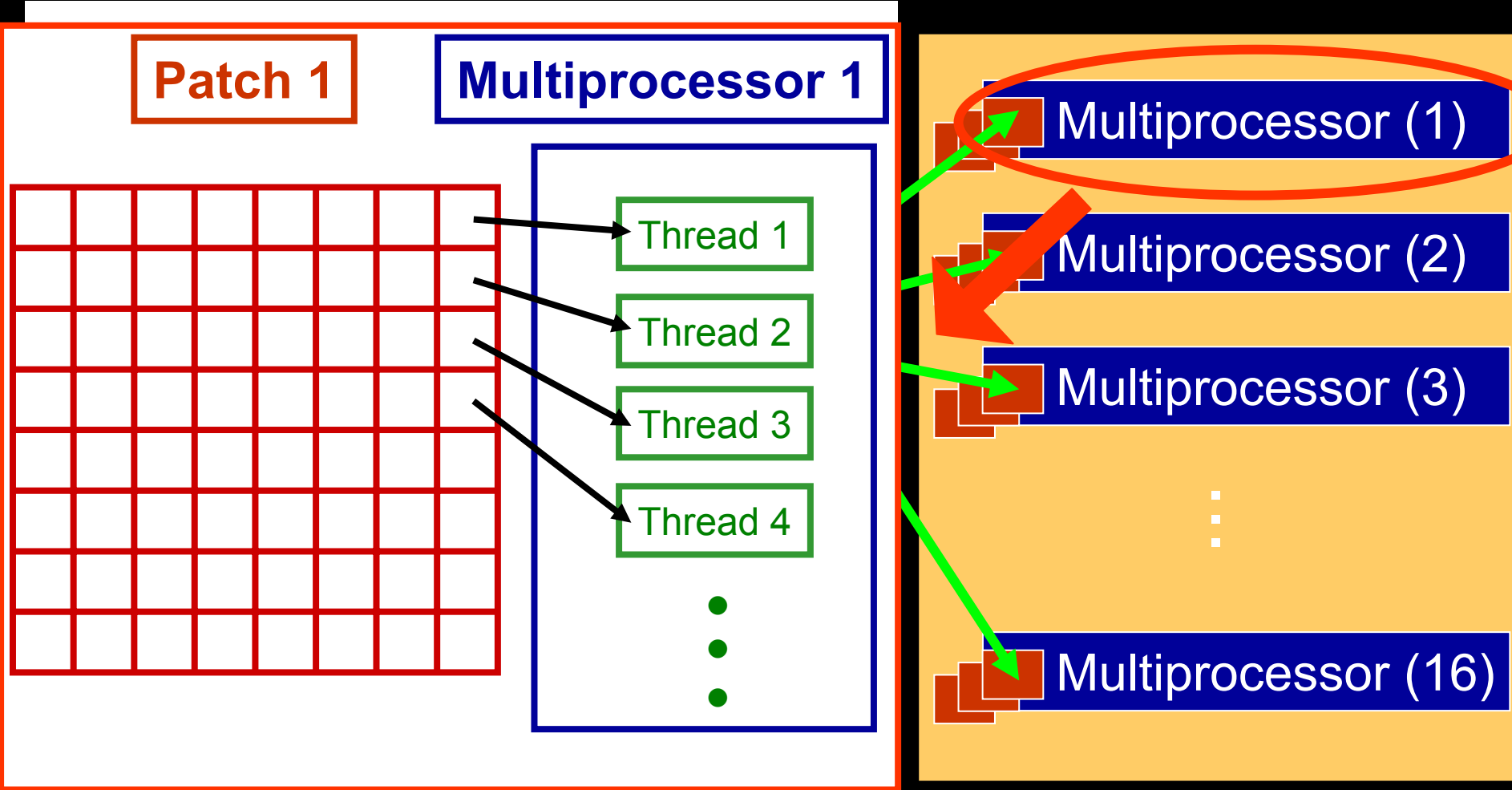


user: USER
Fri Oct 03 23:00:40 2008

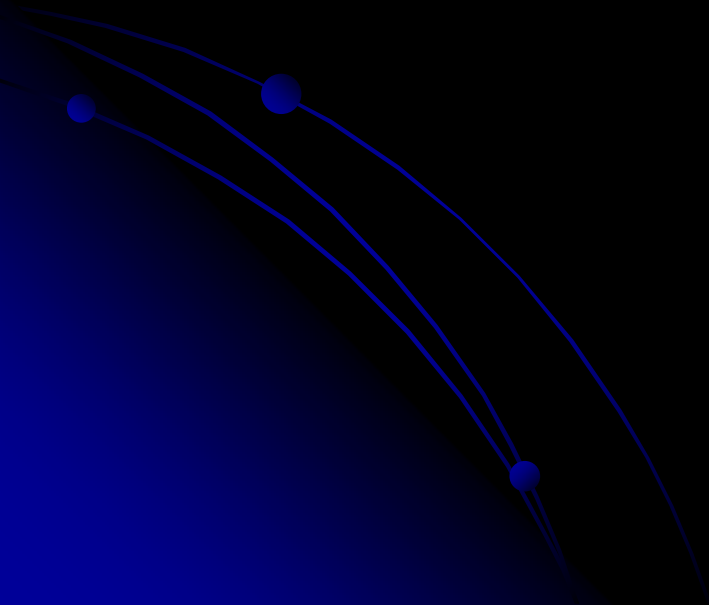


GPU 1

Example : Blast Wave Test



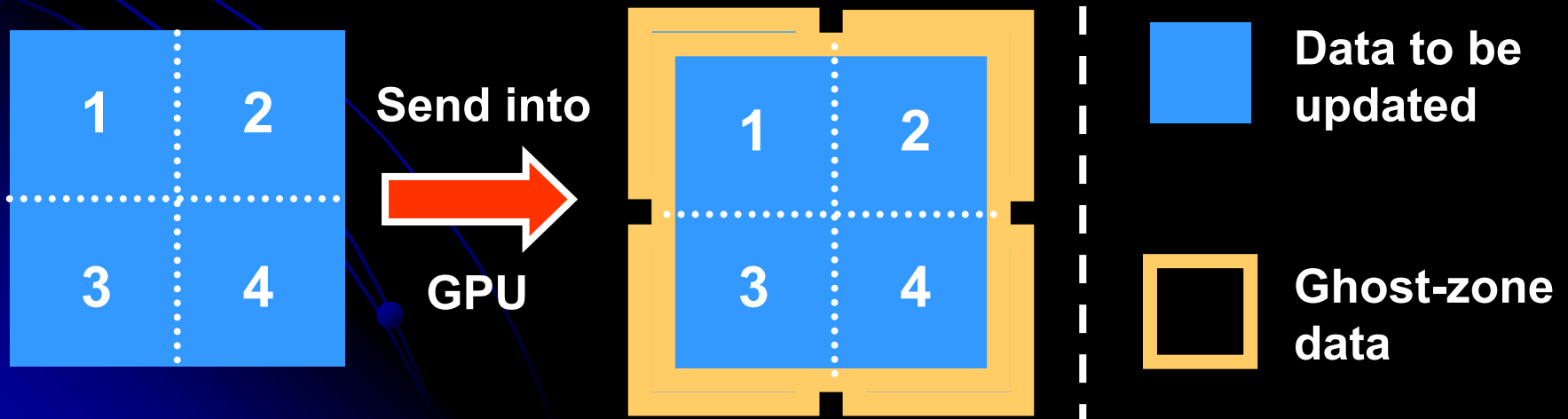
GPU Solvers



GPU Hydrodynamic Solver

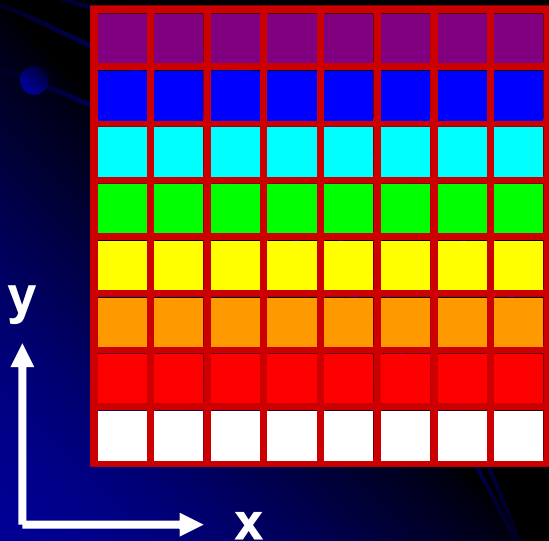
- Second-order relaxing **TVD** scheme (Trac & Pen 2003)
- Ghost-zone data of each patch are obtained by either direct memory copy or interpolation
- Eight nearby patches are always combined into a **patch group** before sending into GPU

→ Reduce the workload of ghost-zone preparation



GPU Hydrodynamic Solver

- **Dimensional splitting method** ($x \rightarrow y \rightarrow z$)
 - ◆ The 3-D data of a single patch group can be regarded as a set of 1-D **data columns**
 - ◆ Different data columns can be updated **independently** and hence **in parallel**
 - ◆ Data within the same column are shared through the **shared memory**

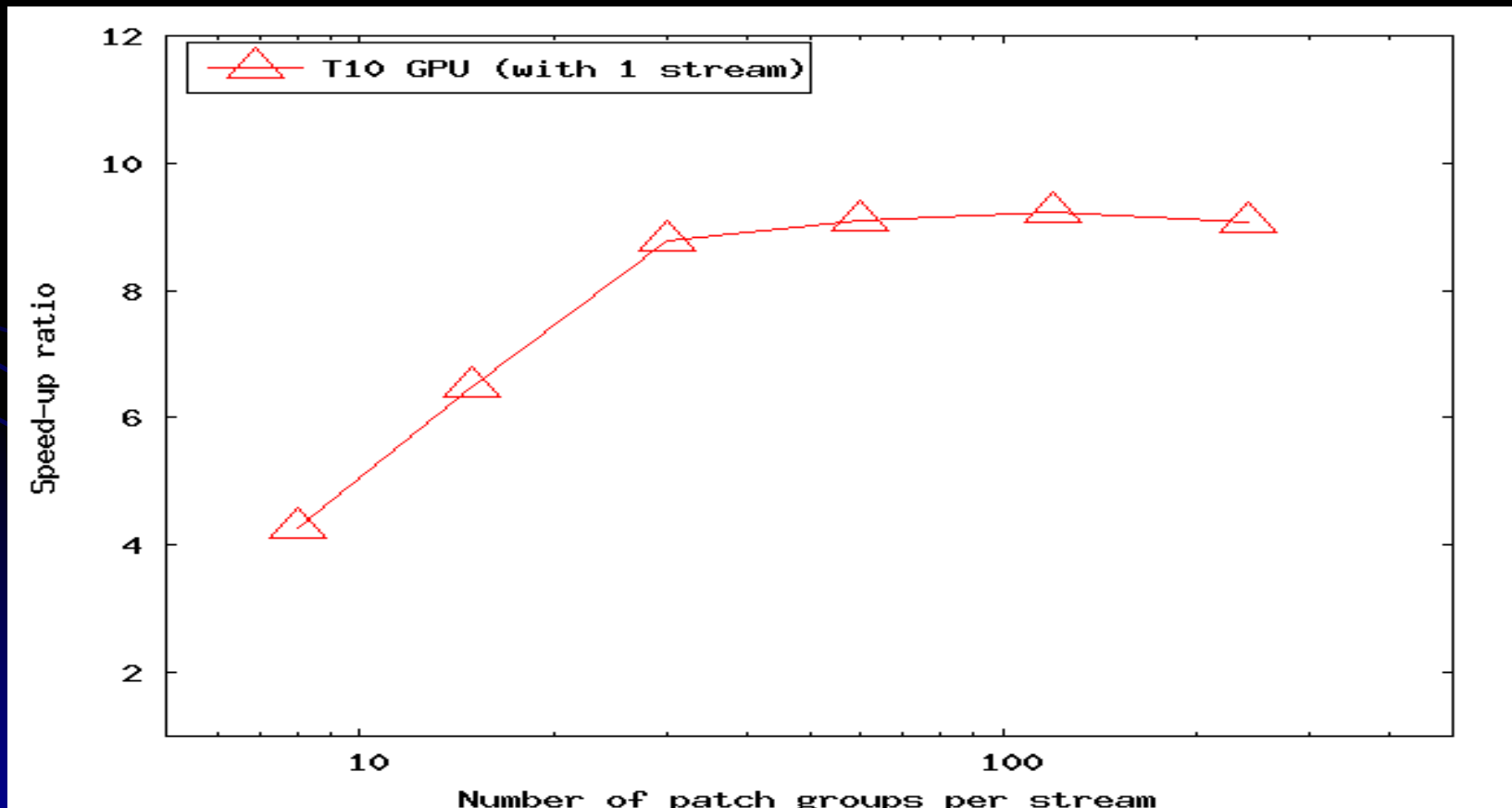


- **Shared memory** : only need to store the data columns being updated
- **The shortage of shared memory is not an issue !!**

GPU Hydrodynamic Solver : *Performance*

- One T10 GPU vs. one Intel Xeon X5472 CPU core (including data transfer between CPU and GPU)

→ **Speed-up ratio : 9.07**



GPU Poisson Solver

- **Root level : fast Fourier transform (FFT)**
 - use **CPUs** only
 - Refinement levels : successive overrelaxation method (SOR)**
 - use **GPUs**
- **Ghost-zone data of each patch group are always obtained by interpolation**
 - **The interpolation can be performed in GPU**
 - **Only need to send the coarse-grid data into GPU**

GPU Poisson Solver

- SOR method: three-dimensional relaxation
 - ◆ Data in each cell must be **re-accessed many times** during the relaxation iterations
 - ◆ **Arithmetic intensity** in each iteration is relatively **low**

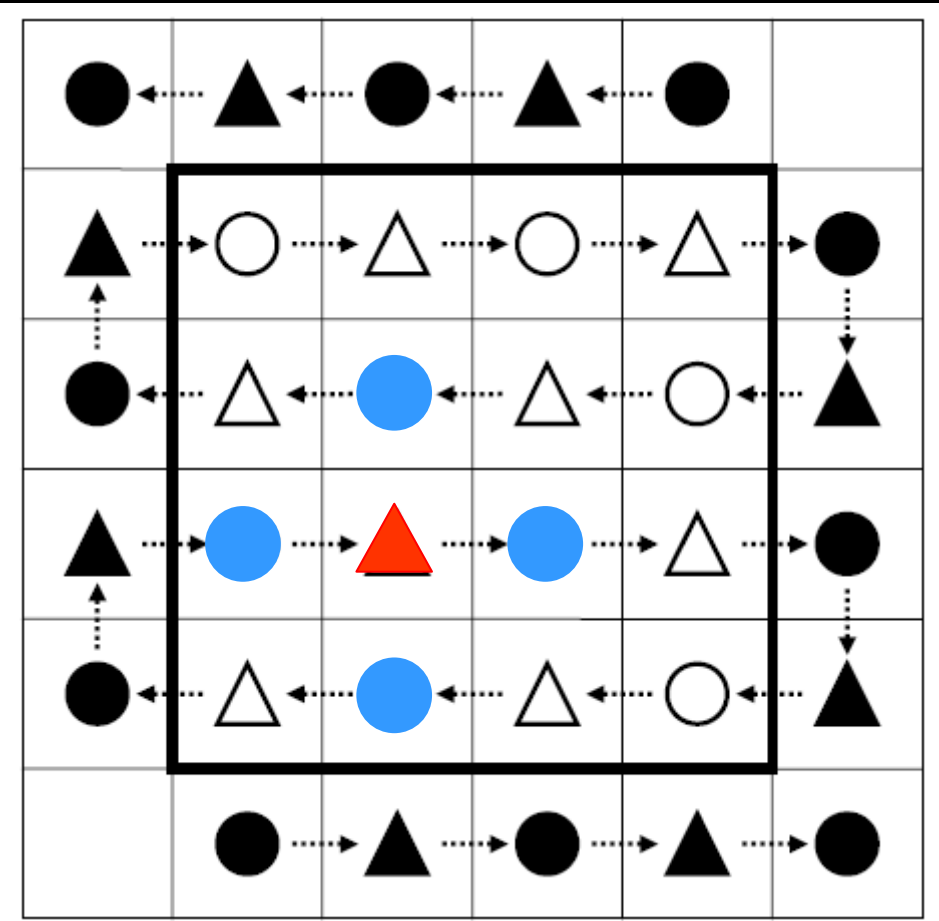


It's better to store all data in the shared memory

- **Issue : shortage of shared memory**
 - ◆ Potential data in one patch group ~ **22.8 KB**
 - ◆ Shared memory per multiprocessor = **16.0 KB**
- **Solution : odd-even ordering + per-thread registers**
(another **64 KB** storage)

Solution to the Shortage of Shared Memory

▲ Even cells ● Odd cells



No ▲ data need to be shared when updating ●

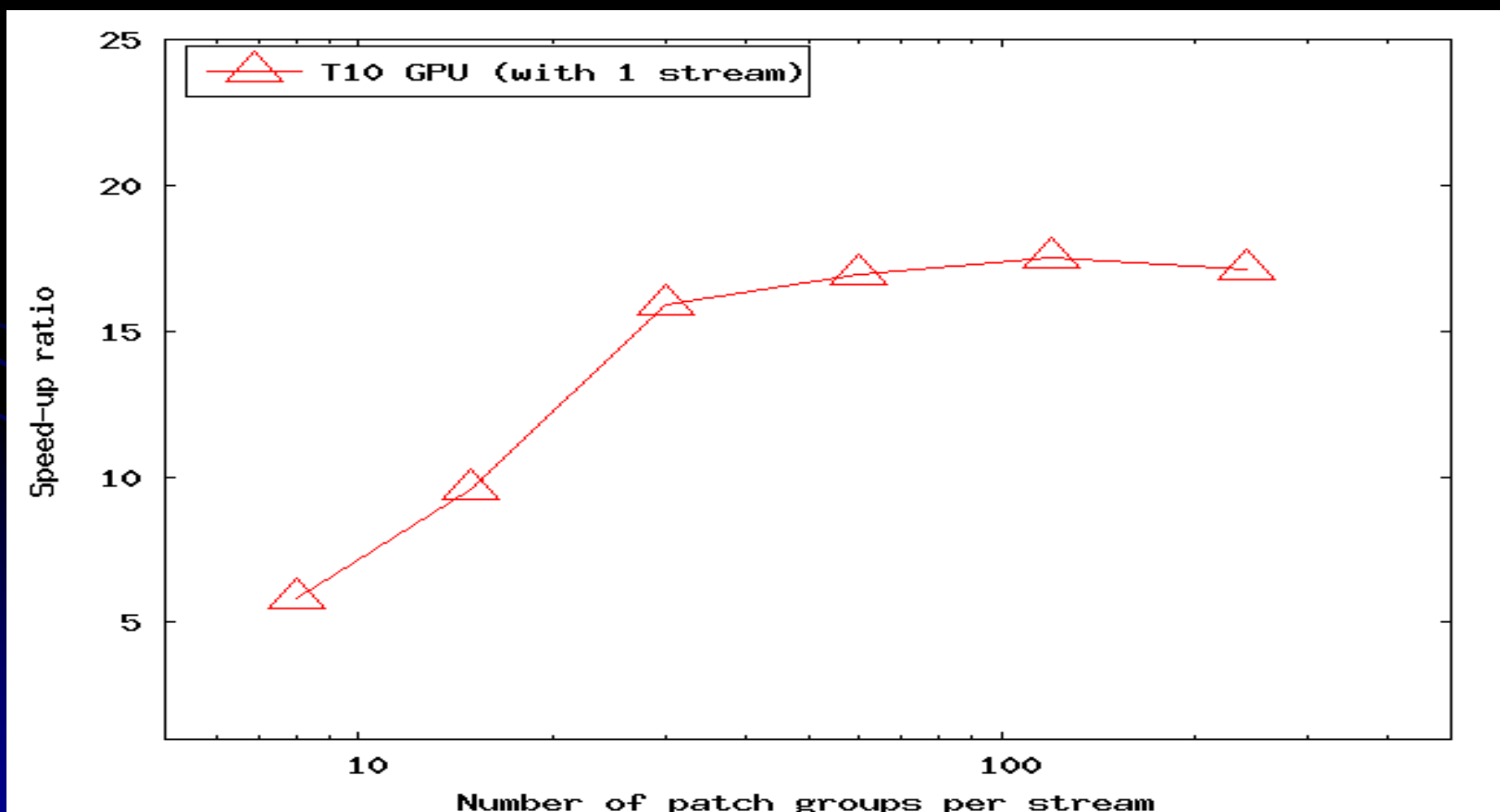
▲ data → per-thread register
● data → shared memory

Exchange data stored in the per-thread registers and shared memory before updating ●

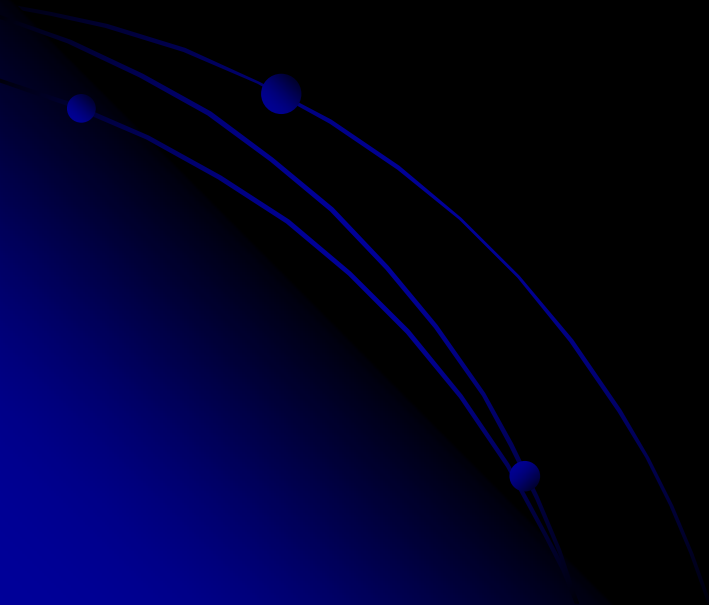
GPU Poisson Solver : *Performance*

- One T10 GPU vs. one Intel Xeon X5472 CPU core (including data transfer between CPU and GPU)

→ **Speed-up ratio : 17.12**



Optimizations

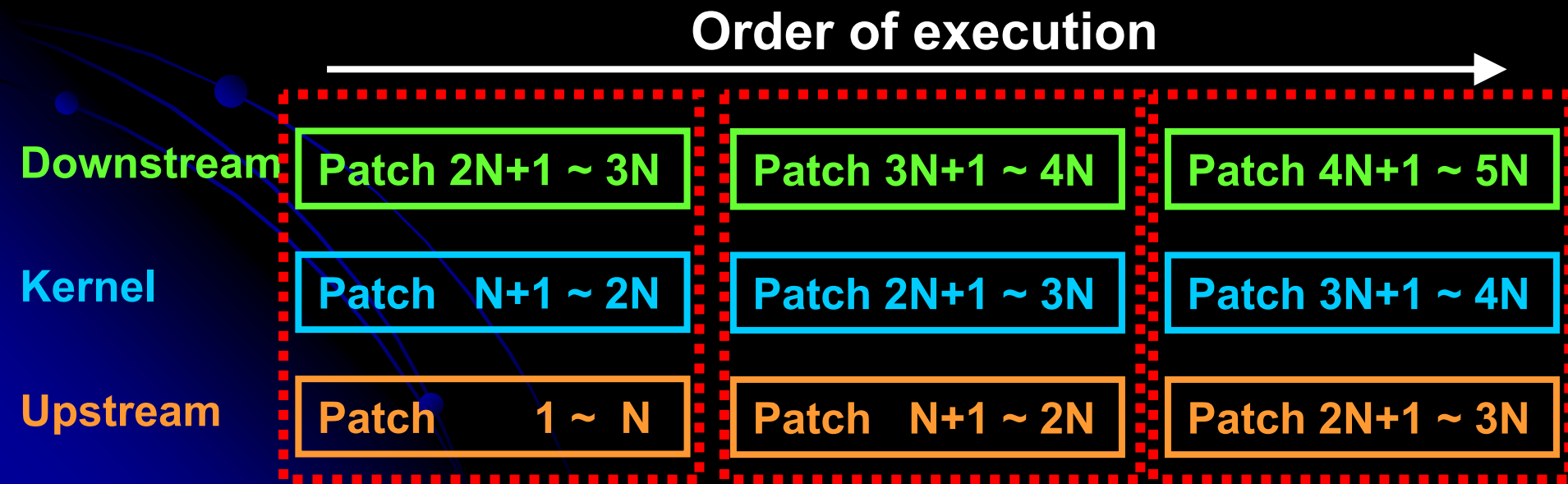


Optimization I :

Asynchronous Memory Copy

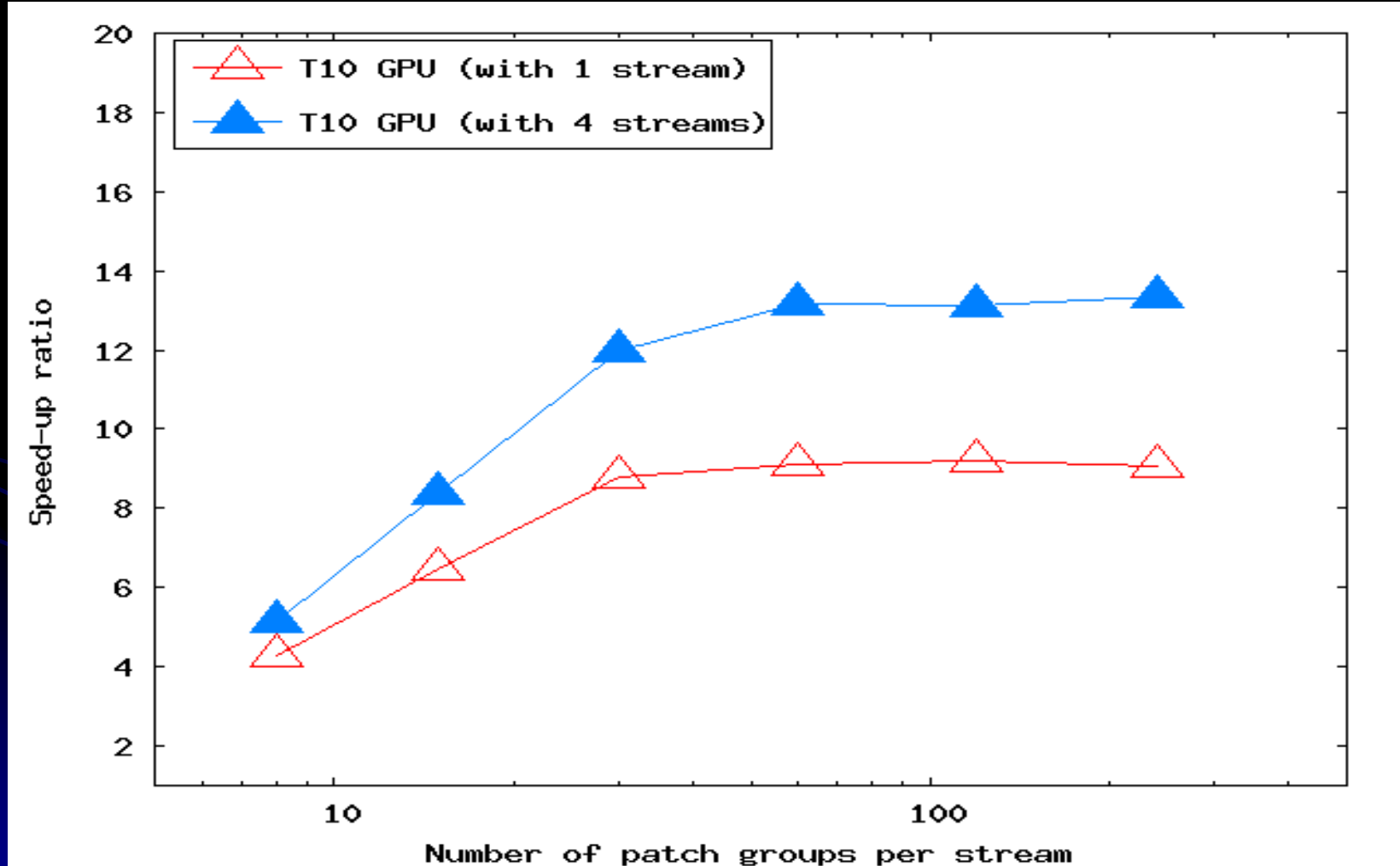
- Data transfer between CPU and GPU :
25% ~ 44% of the total GPU execution time !!

➔ Use **CUDA streams** to perform memory copy concurrently with kernel execution.



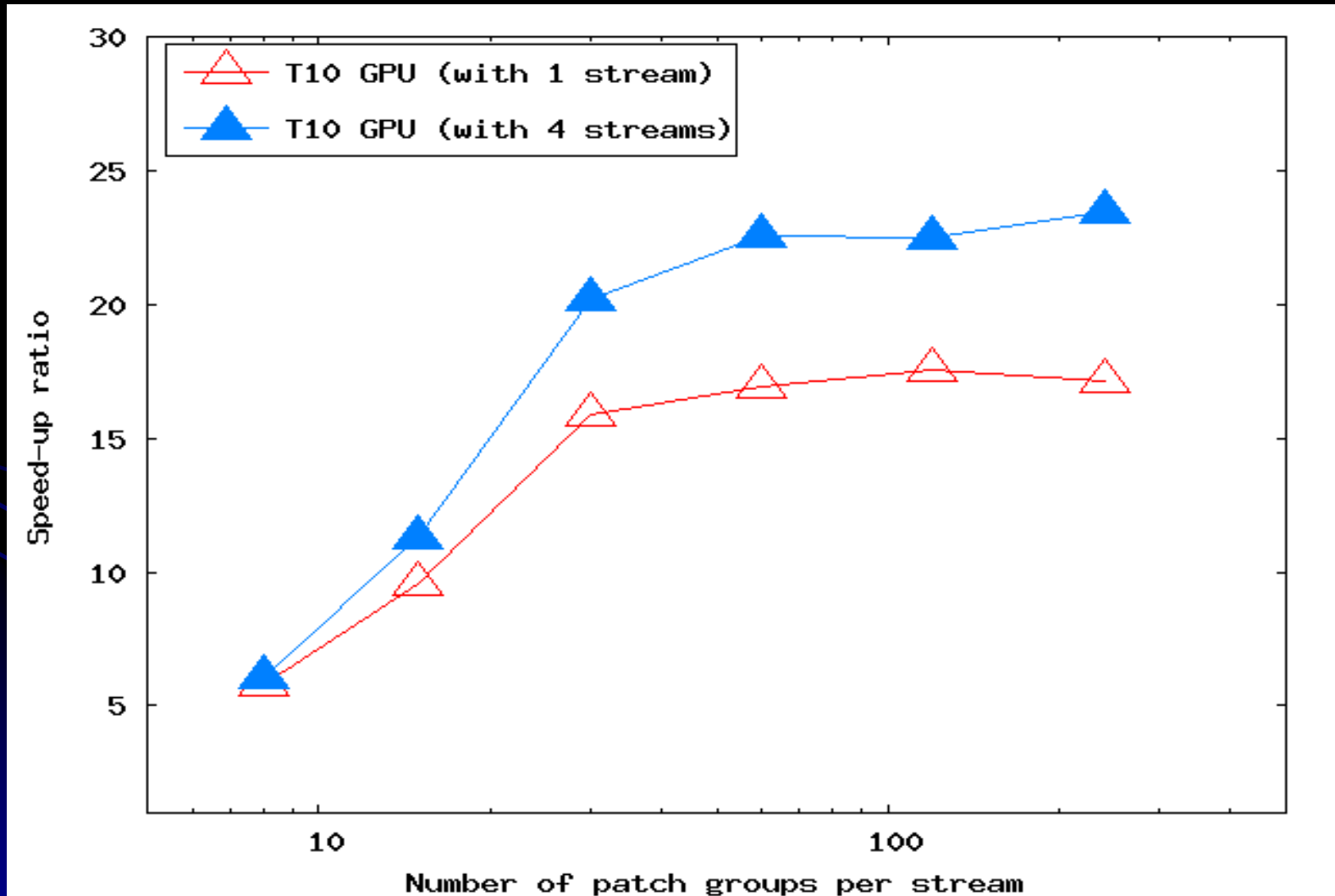
Optimization I : *Fluid Solver*

- GPU vs. CPU speed-up ratio : **9.07** \rightarrow **13.37** (1.47x)



Optimization I : *Poisson Solver*

- GPU vs. CPU speed-up ratio : **17.12** \rightarrow **23.44** (1.37x)



Optimization II :

Concurrent Execution between CPU and GPU

• Invoking a GPU solver involves 3 steps :

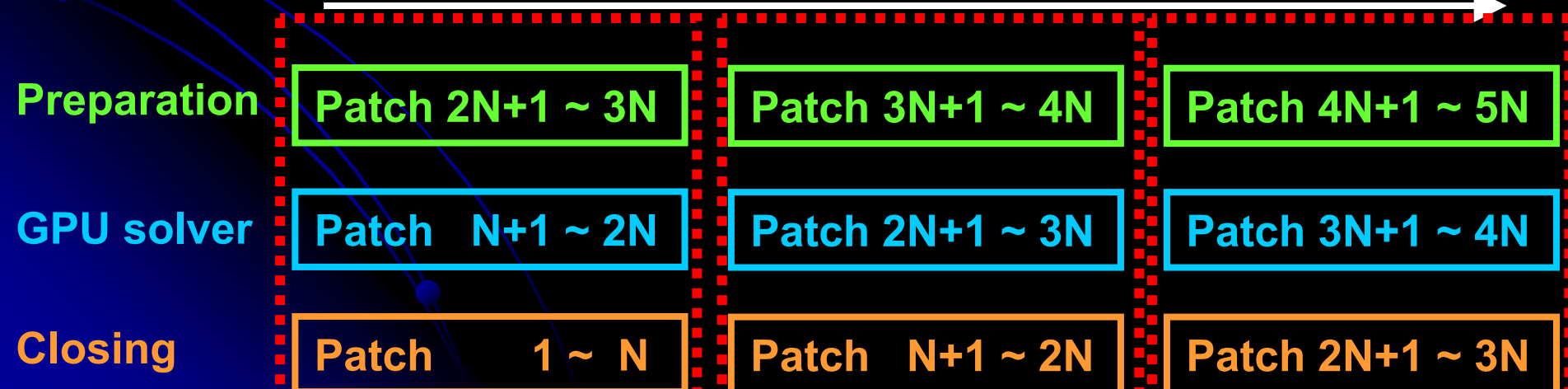
1. **Preparation step** : prepare input array (by CPU)

2. **Execute GPU solver** : memory copy + CUDA kernel

3. **Closing step** : store output array + fix-up operation (by CPU)

→ Hide the time of preparation and closing step by executing the GPU solver concurrently

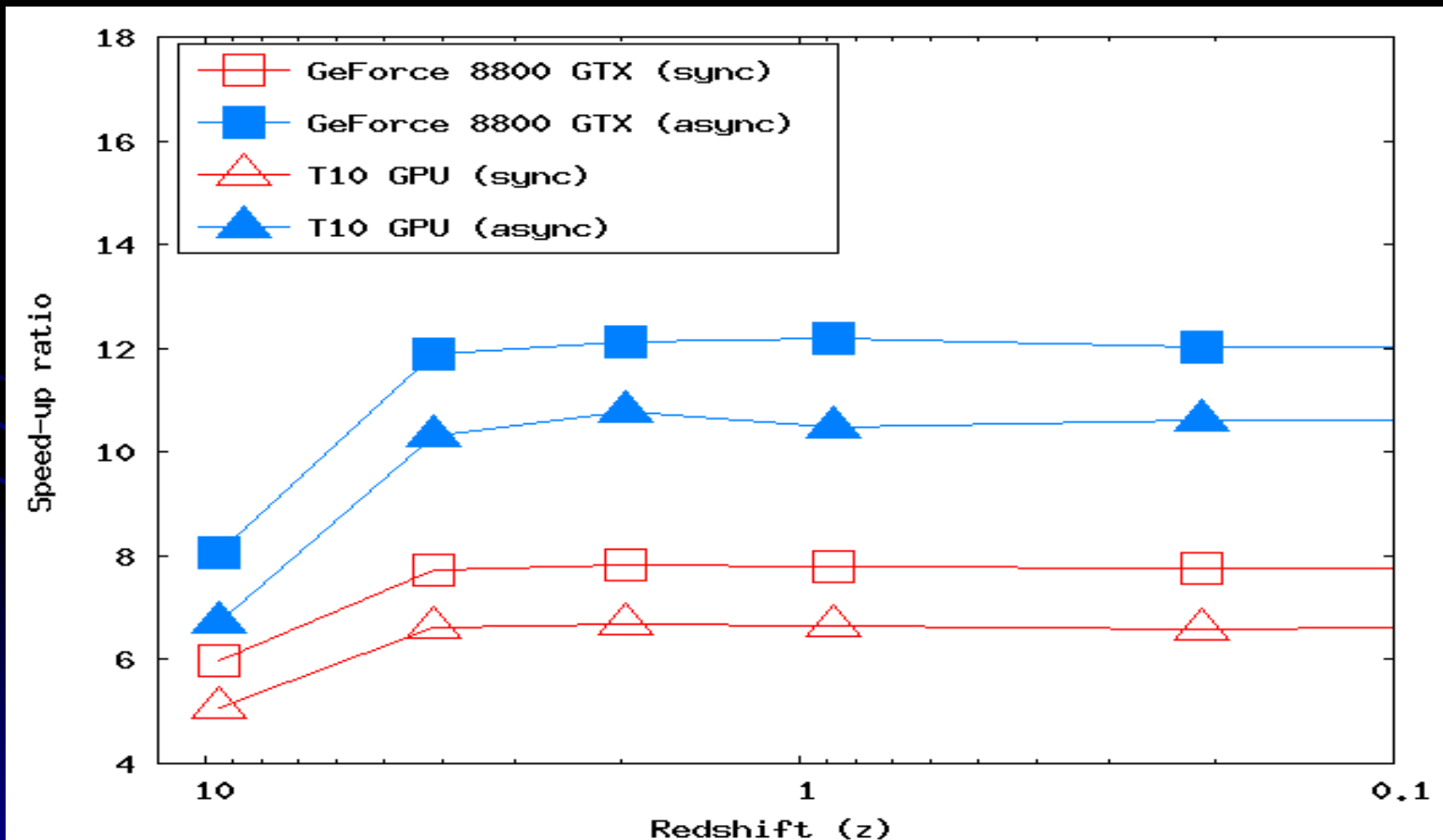
Order of execution



Optimization II : Cosmological Simulation

- One GPU vs. one CPU core speed-up ratio :

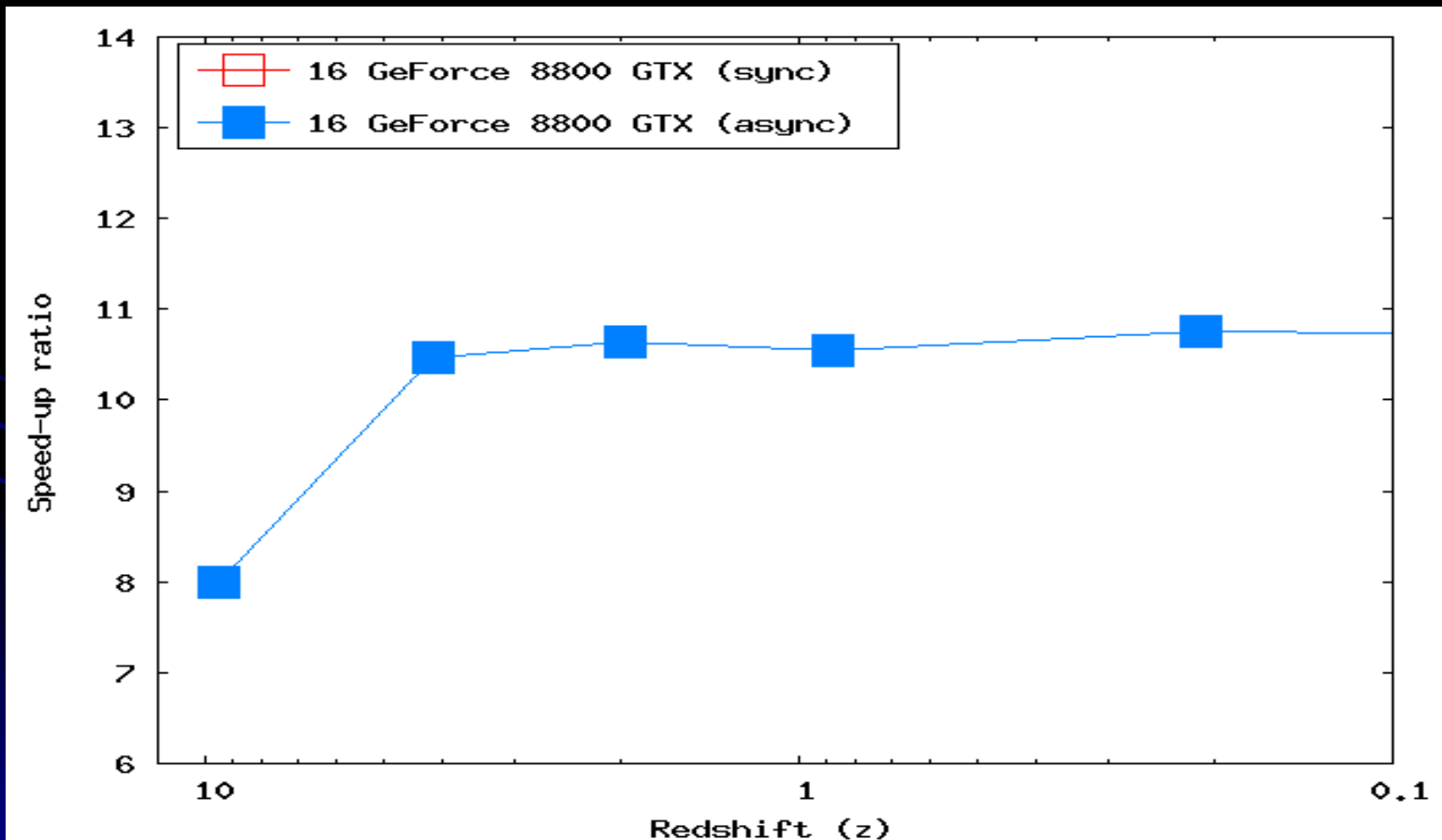
6.68 ~ 7.77 → 10.65 ~ 12.11 (~1.57x)



Optimization II :

Cosmological Simulation

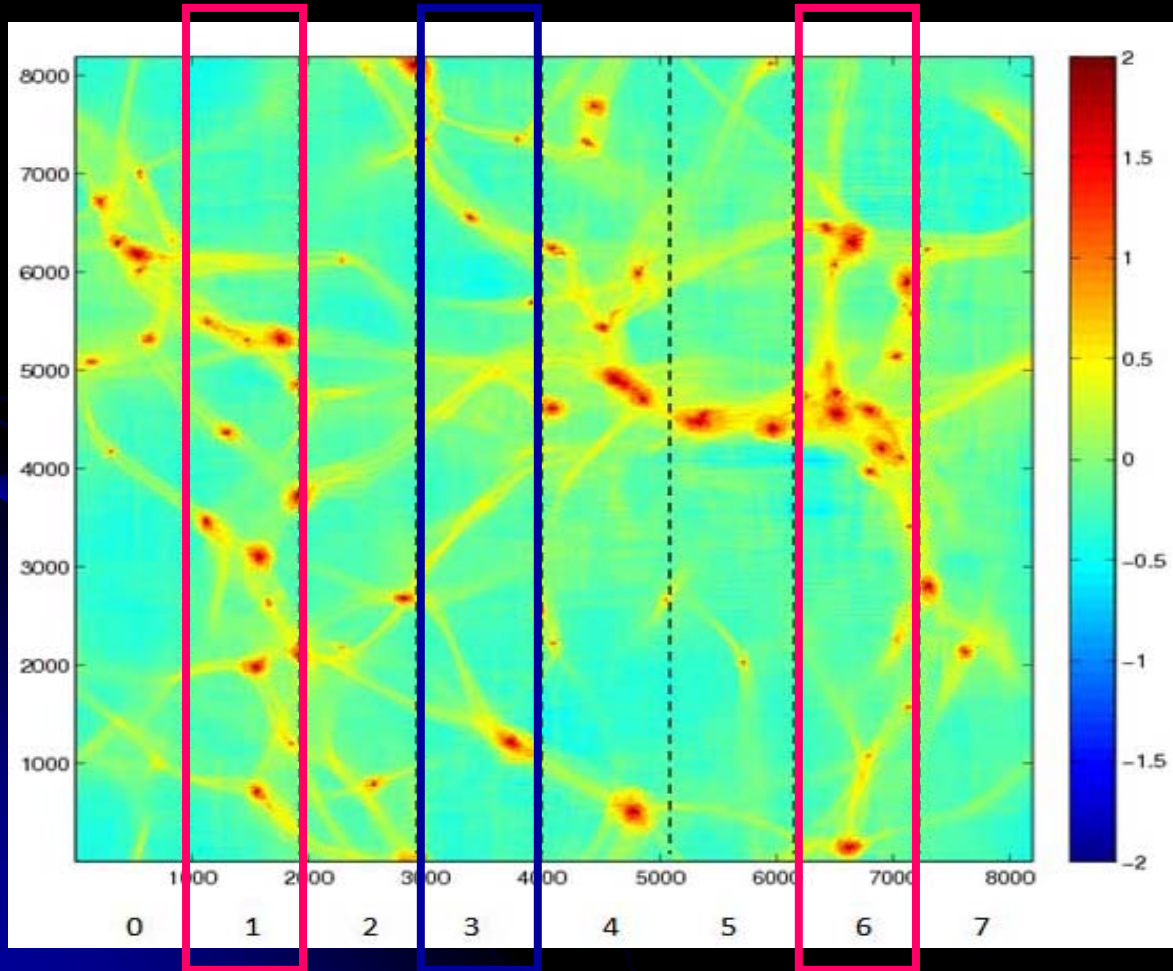
- 16 GPUs vs. 16 CPU cores speed-up ratio :
> 10.50 after $z \leq 4$



Optimization III :

Space-filling Curve for Domain Decomposition

- The **rectangular domain decomposition** can lead to an issue of **load imbalance**.



More load

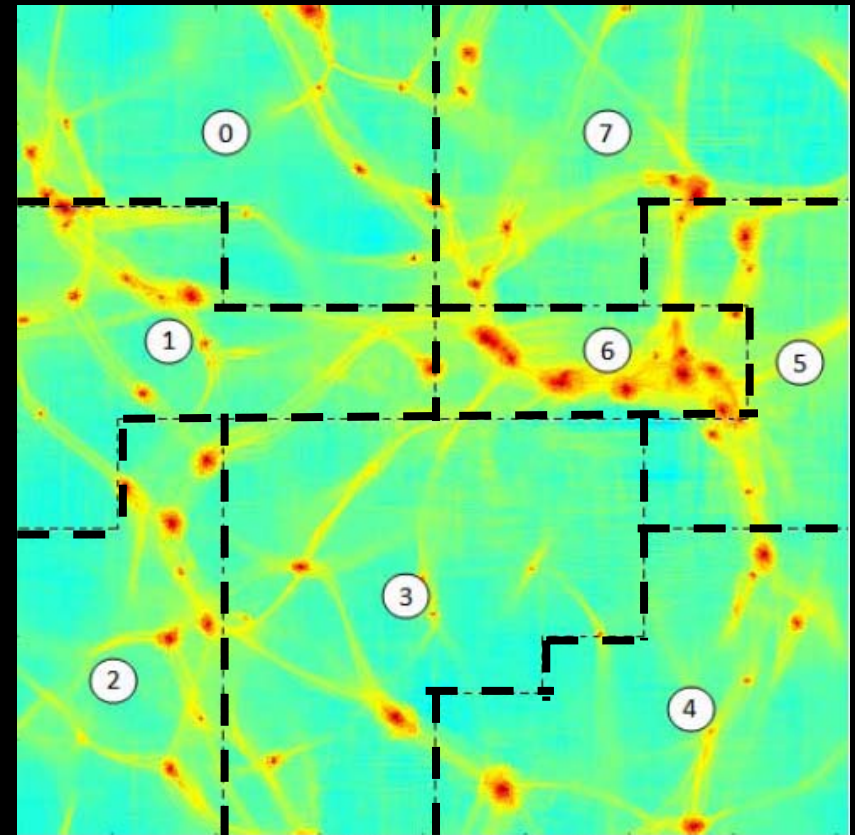
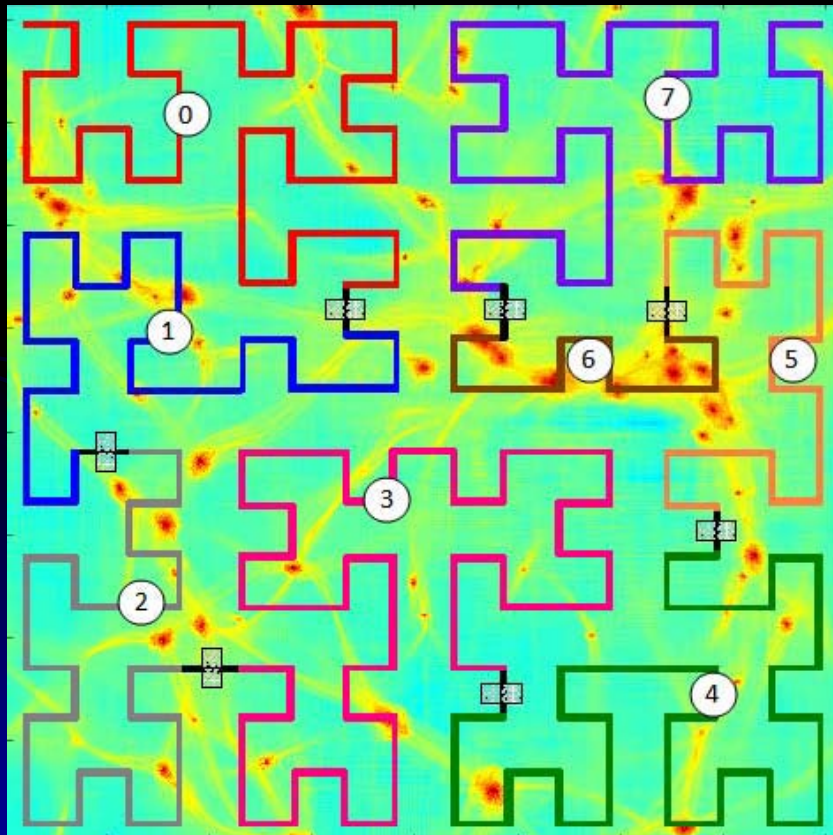


Less load

Optimization III :

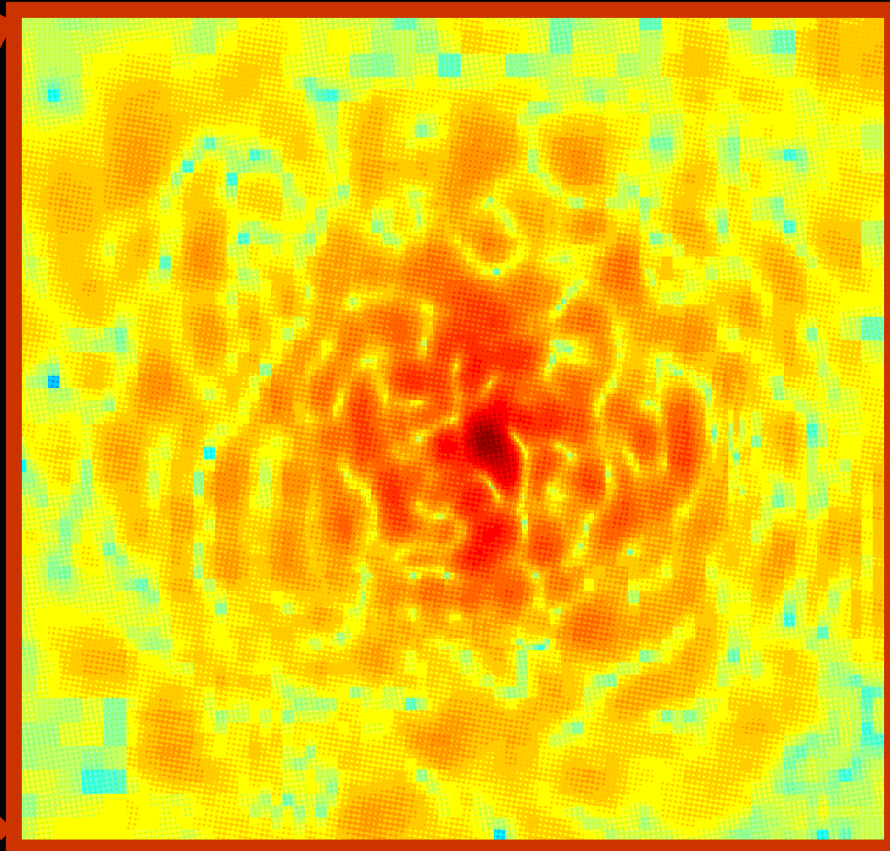
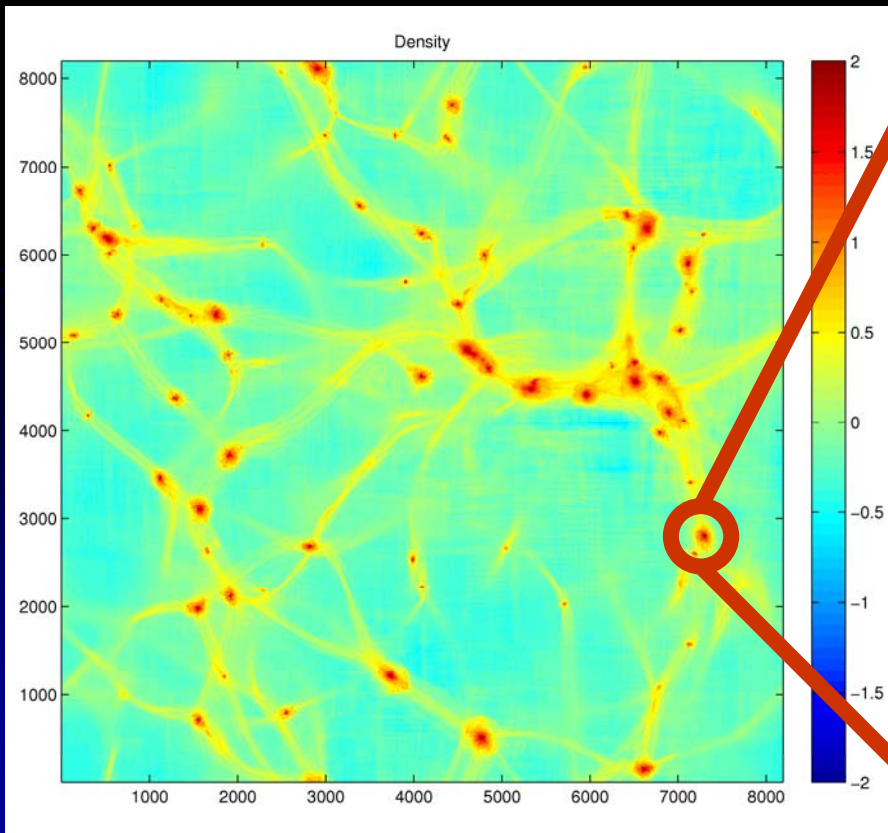
Space-filling Curve for Domain Decomposition

- The standard **space-filling curve** method can be applied to GAMER (not complete yet)



Other Applications

- Large-scale structure simulation with an extremely light dark matter model
→ Schrödinger equation with self gravity



Conclusion and Future Work

- **GAMER** : GPU-accelerated Adaptive-Mesh-Refinement Code
 - ◆ GPU hydrodynamic and Poisson solvers
 - ◆ **Parallelized** (multi CPUs + multi GPUs)
 - ◆ A framework of AMR + GPUs → **general-purpose**
 - ◆ **10x** faster than CPUs (N GPUs vs. N CPU cores)
- **Optimizations**
 - ◆ **Concurrency** of memory copy and kernel execution
 - ◆ **Concurrency** of CPU work and GPU work
- **Future work**
 - ◆ **Space-filling curve** for domain decomposition
 - ◆ **More physics** (e.g., dark matter particles, MHD, higher-order fluid solver ...)