GPU/CPU work sharing on high level parallel programming & Collaboration with applications

Taisuke Boku

Leader, HPC Division

Center for Computational Sciences

University of Tsukuba

My major topics in HPC Division, CCS

- TCA/PEACH2 R&D
- GPU/CPU work sharing on XMP-dev
- Collaborative code development on domain science
  - LES (with Global Environmental Science Division)
    GPU porting from original Fortran
  - TD-DFT (Astrophysics and Nuclear Physics Division)
    Performance tuning and scalable coding
  - GT5D (under G8RCI program, JAEA)
    GPU porting from original Fortran
  - GTC-P (under G8RCI program, Princeton)
    XMP porting from original C

GPU/CPU work sharing based on XMP-dev/StarPU

(Collaborative work with
INRIA Bordeaux)

Background

- GPGPU is widely used for HPC
  - Impact of NVIDIA CUDA, OpenCL
  - Programming became easy on single node
  - > Many GPU cluster appear on TOP500 list
- Problem of programming on GPU cluster
  - Inter-node programming model (such as MPI)
  - Data management among CPU and GPU
  - > Programmability and productivity are very low
- GPU is very powerful, but...
  - CPU's performance have been improved.
  - We cannot neglect its performance.

・Complex
・Include program source

Purpose

- High productivity programming on GPU clusters
- Utilization of both GPU and CPU resources
  - Work sharing of the loop execution

- Implementation and Evaluation of XMP-dev/StarPU
  - Statically
    - It is difficult to decide the balance among resources
  - Dynamically
    - Decide in execute time of application
    - High usability

XcalableMP (XMP)

- A PGAS language designed for distributed memory systems
  - Directive-base; easy to understand
    - array distribution, inter-node communication, loop work sharing on CPU, etc…
  - Low programming cost
    - little change from the original sequential program
- XMP is developed by a working group by many Japanese universities and HPC companies

XcalableMP acceleration device extension (XMP-dev)

- Developed HPCS laboratory, University of Tsukuba, and RIKEN AICS
- XMP-dev is an extension of XMP for accelerator-equipped cluster
  - Additional directives (manage accelerators)
    - Mapping data onto accelerator's device memory
    - Data transfer between CPU and accelerator
    - Work sharing on loop with accelerator device (ex. GPU cores)

```
1  int x[N], y[N];
2
3  #pragma xmp nodes p(4)
4  #pragma xmp template t(0:N-1)
5  #pragma xmp distribute t(BLOCK) onto p
6  #pragma xmp align [i] with t(i) :: x, y
7
8  int main() {
9    int i;
10
11 #pragma xmp loop on t(i)
12   for (i = 0; i < N; i++) {
13     x[i] = func(i);
14     y[i] = func(i);
15   }
16
17 #pragma xmp device replicate (x, y)
18   {
19
20
21 #pragma xmp device replicate_sync in (x, y)
22
23 #pragma xmp device loop on t(i)
24     for (i = 0; i < N; i++)
25       y[i] += x[i];
26
27
28 #pragma xmp device replicate_sync out (y)
29   }
30 }
```

Execution on Host

Data Allocation

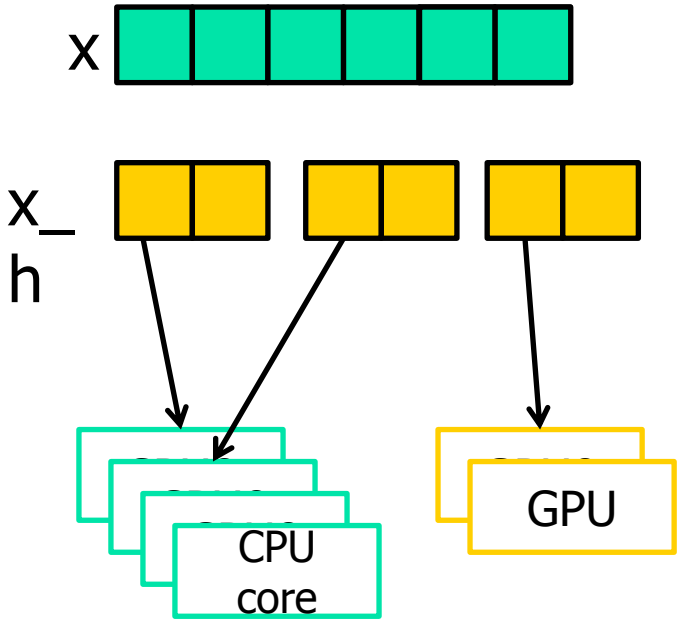
Data Copy Host to Device

Execution on Device

Data Copy Device to Host

StarPU

- Developed by INRIA Bordeaux, France
- StarPU is a runtime system
  - allocates and dispatches resource
  - schedules the task execution dynamically
- All the target data is recorded and managed in a data pool shared by all computation resources.
  - guarantees the coherence of data among multiple task executions

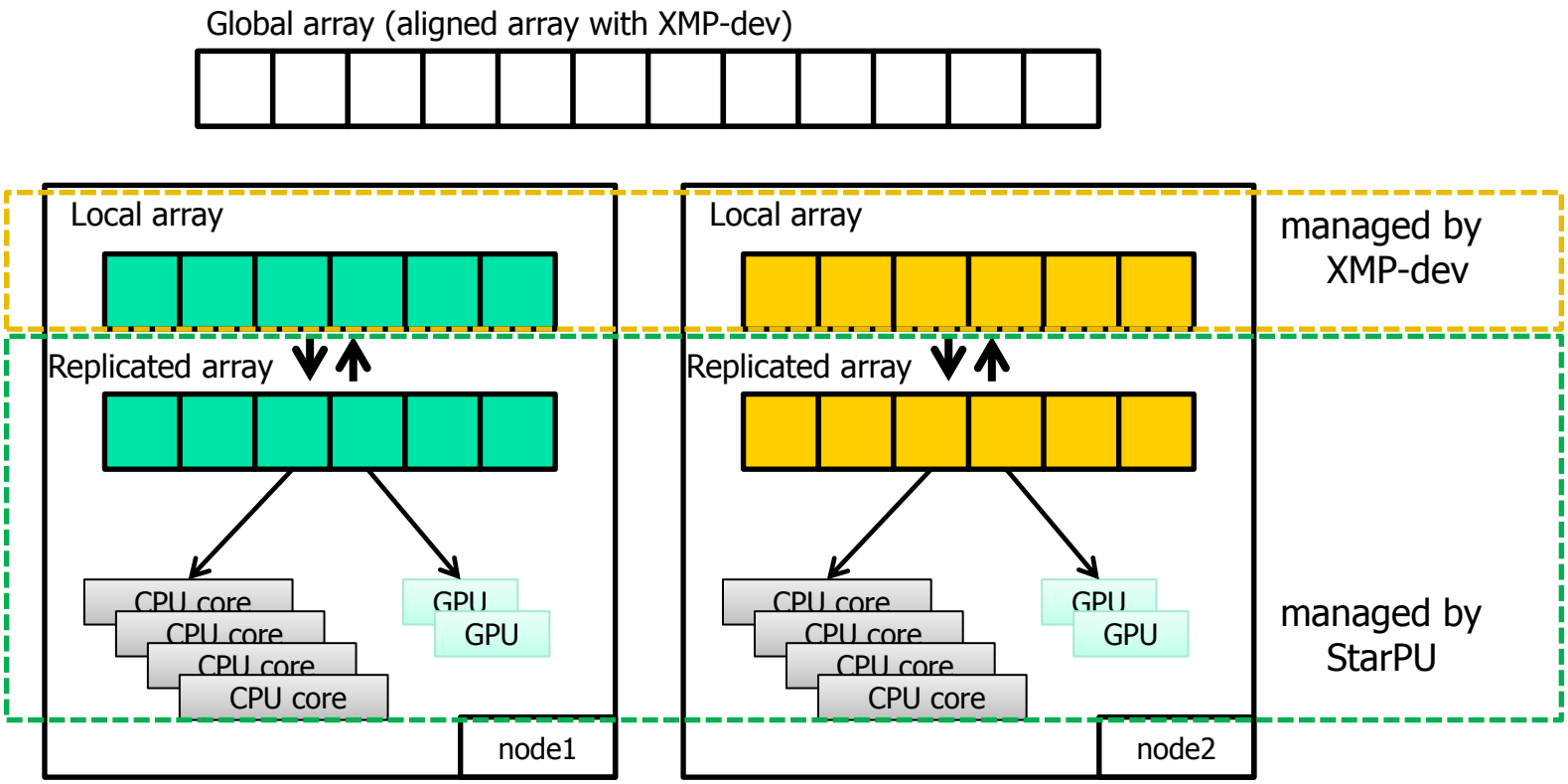


Implementation of Prototype XMP-dev/StarPU

- We combine XMP-dev and StarPU to enhance the function and performance
  - GPU/CPU work sharing on multi-node GPU cluster

- Advantage of XMP-dev
  - Using not only GPU, but also CPU power
- Advantage of StarPU
  - It is not necessary to write complex StarPU code

Implementation of Prototype XMP-dev/StarPU



XMP-dev
- inter-node communication
- data distribution

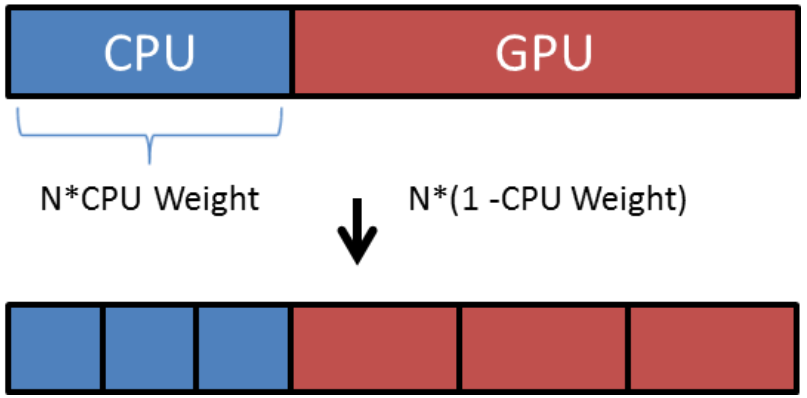
StarPU
- data transfer between GPU and CPU
- GPU/CPU work sharing on single-node

Problem of Prototype Implementation

- Prototype implementation is low performance
  - Only 45% of XMP-dev/CUDA (only GPU)
- Performance gap between GPU and CPU core
  - Divide Replicated array equally
  - Large execution time gap in same task size
  - >Performance decrement by that gap
- Equalizing load balancing
  - Allocate proper task size to resources

Load Balancing on XMP-dev/StarPU

- For proper allocating…
  - Define a parameter named “CPU Weight”
    - 0 ≦ CPU Weight ≦ 1.0
  - Set the region of Replicated array for CPU
  - >Load balancing
- CPU Weight is affected
  - Problem
  - Problem size
  - Application feature
  - etc.



“Dynamic” Load Balancing on XMP-dev/StarPU

- Introduce “reset_weight directive”

```
double cpu_weight;
#pragma xmp device reset_weight  (cpu_weight) :: list
```

- Example

```
double new_cpu_weight;
for (int t = 0; t < TIMESTEP; t++) {
  …
  double cpu_ratio = cpu_time / (cpu_time + gpu_time) * 100;

  if (cpu_ratio > 50) new_cpu_weight -= 0.01;
  else                new_cpu_weight += 0.01;

#pragma xmp device reset_weight (new_cpu_weight)
}
```

Performance Evaluation

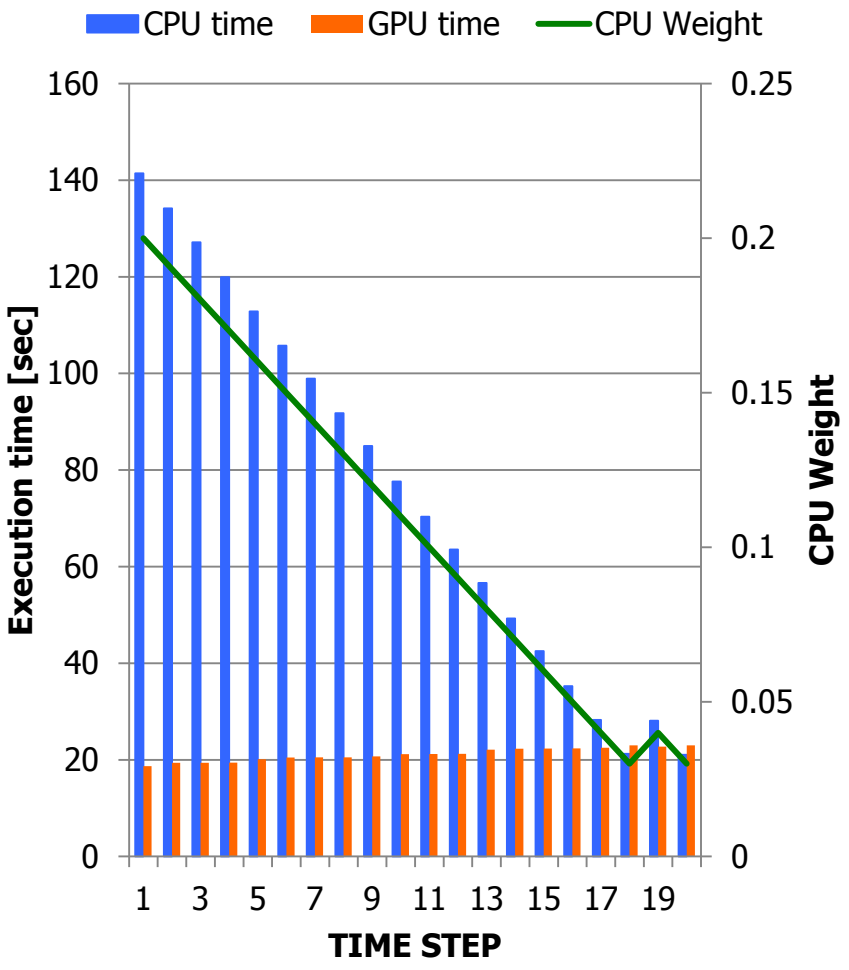
- Benchmarks
  - N-body
  - Matrix-Matrix Multiplication (MM)
  - XMP-dev/StarPU vs. XMP-dev/CUDA (only GPU)
- Node specification

CPU	Intel Xeon E5-2670 * 2 (16 cores)
Memory	DDR3 128GB
GPU	NVIDIA Tesla M2090 * 4
CUDA toolkit	4.2
MPI	MVAPICH2 1.8.1
Interconnection	Infiniband QDR 4x 2 rails
# of node	2~16

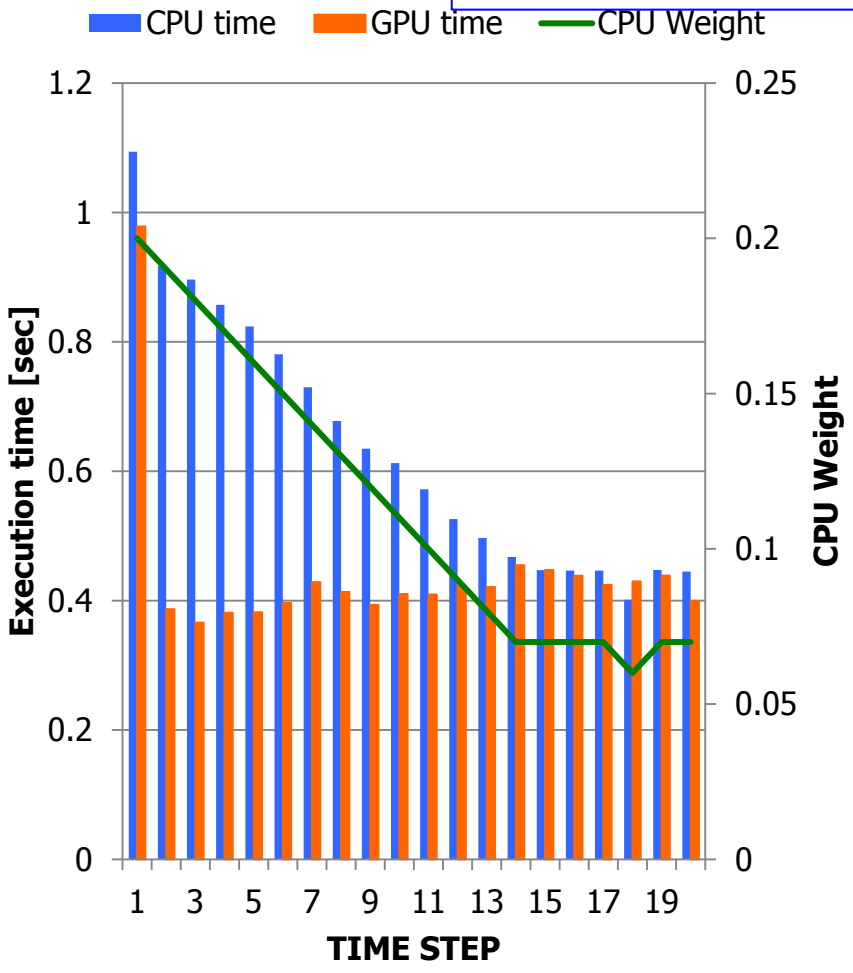
Evaluation : task size annealing

Environment
HA-PACS GPU Cluster
CCS, U. of Tsukuba

N-Body

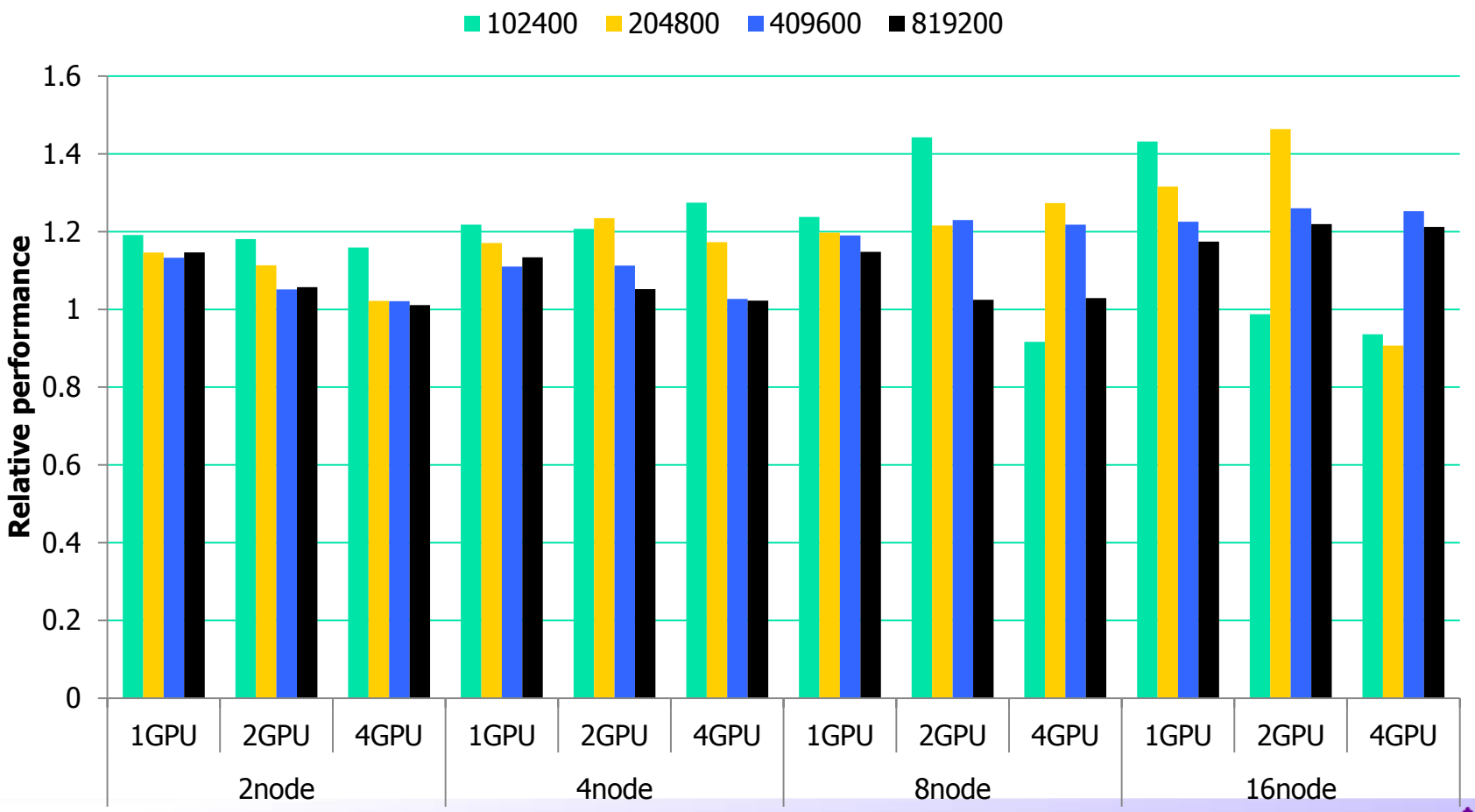


MM



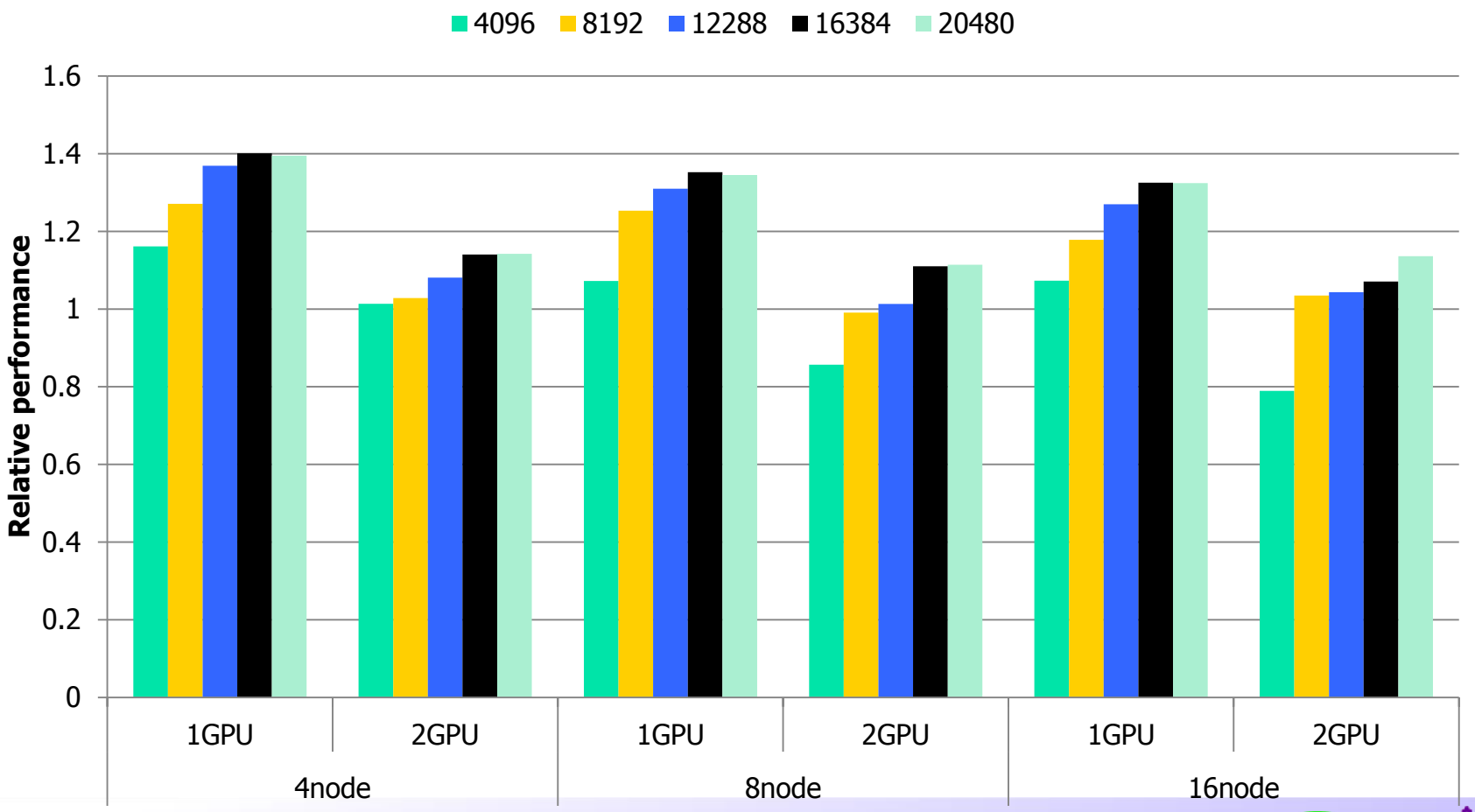
Performance gain to GPU-only case (N-body)

Environment
HA-PACS GPU Cluster
CCS, U. of Tsukuba



Performance gain to GPU-only case (MM)

Environment
HA-PACS GPU Cluster
CCS, U. of Tsukuba



Nuclear Fusion Code Development

(1) GPU version of GT5D

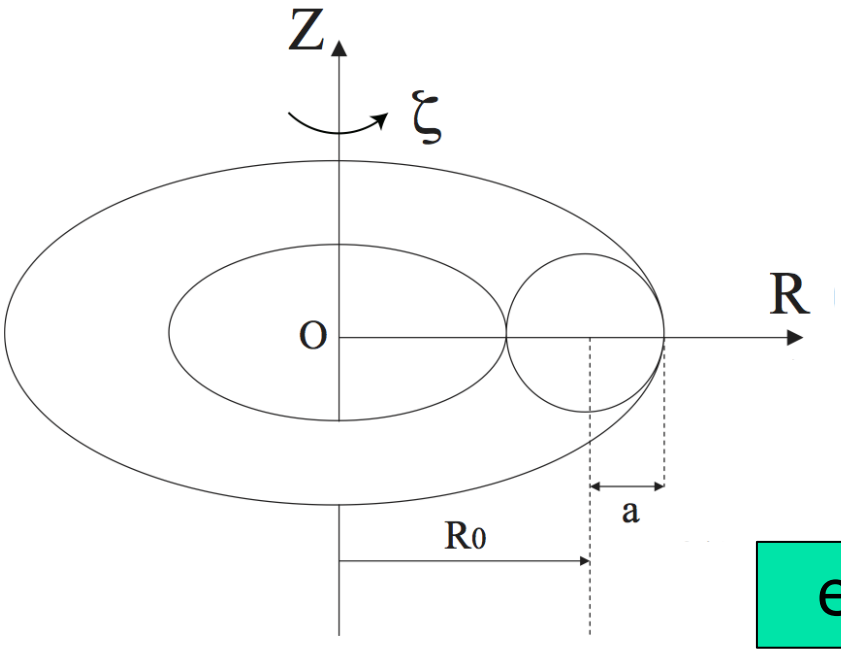
(2) XMP version of GTC-P

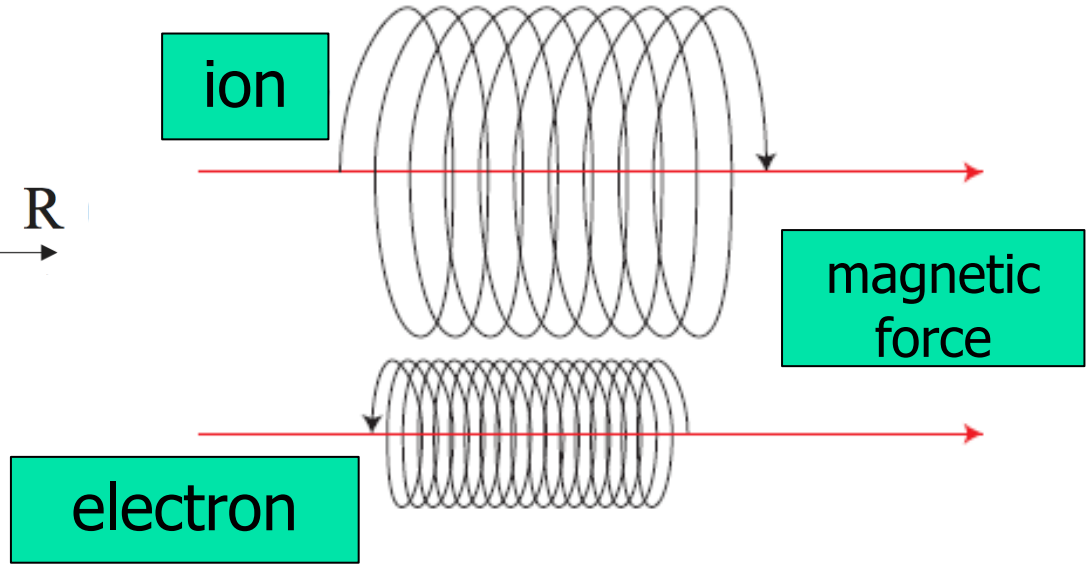
(collaborative work with

JAEA Japan and Princeton U.)

Coordinate System on GT5D

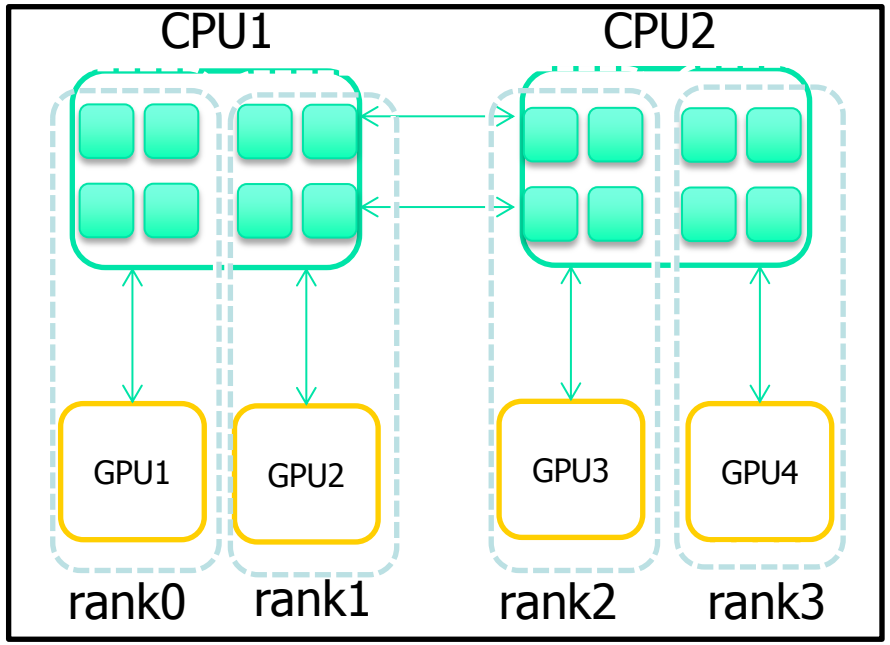
- Torus domain: physical 3D space domain
- Plasma particle movement: 2D velocity domain



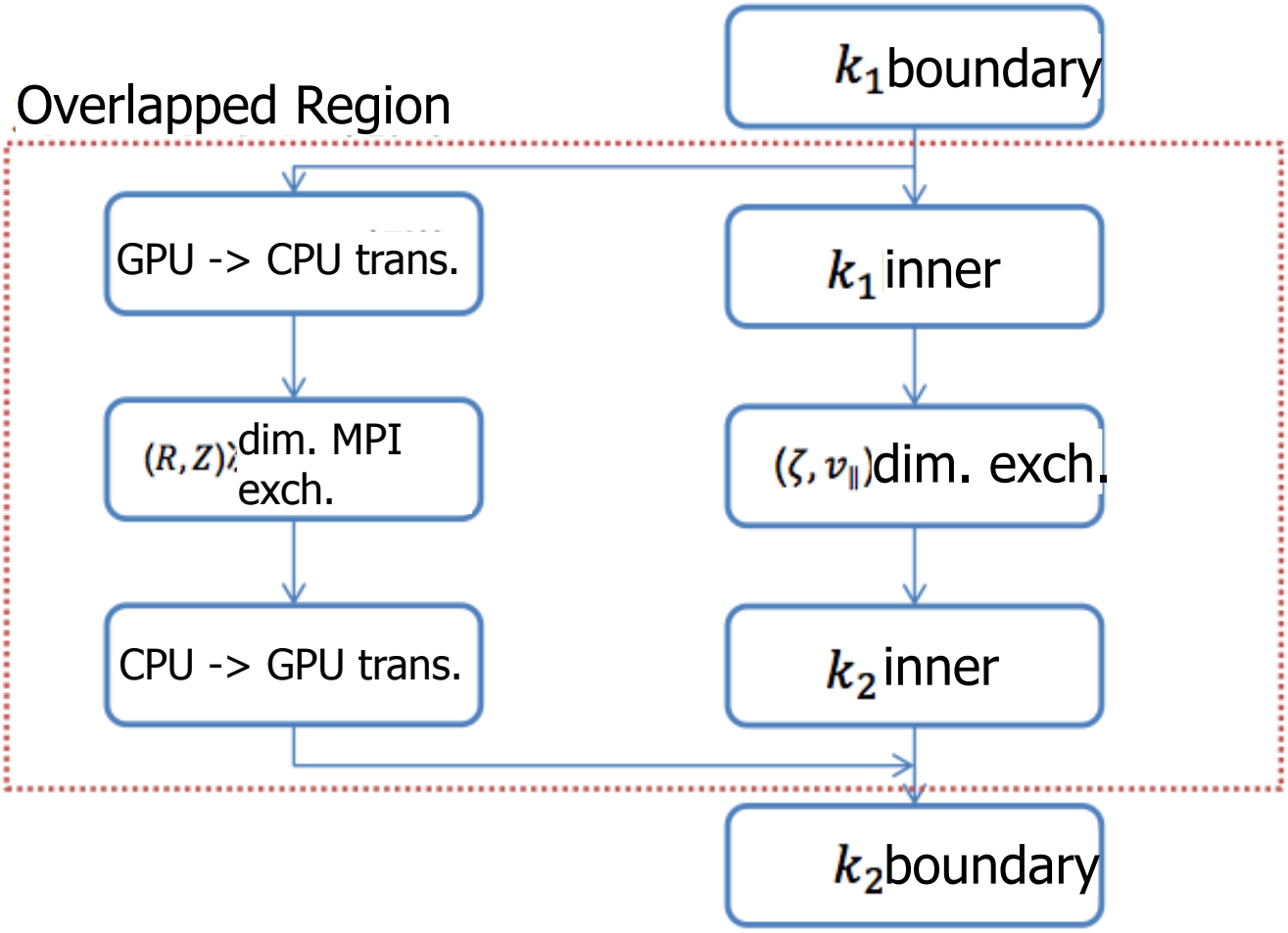


Porting GT5D to HA-PACS

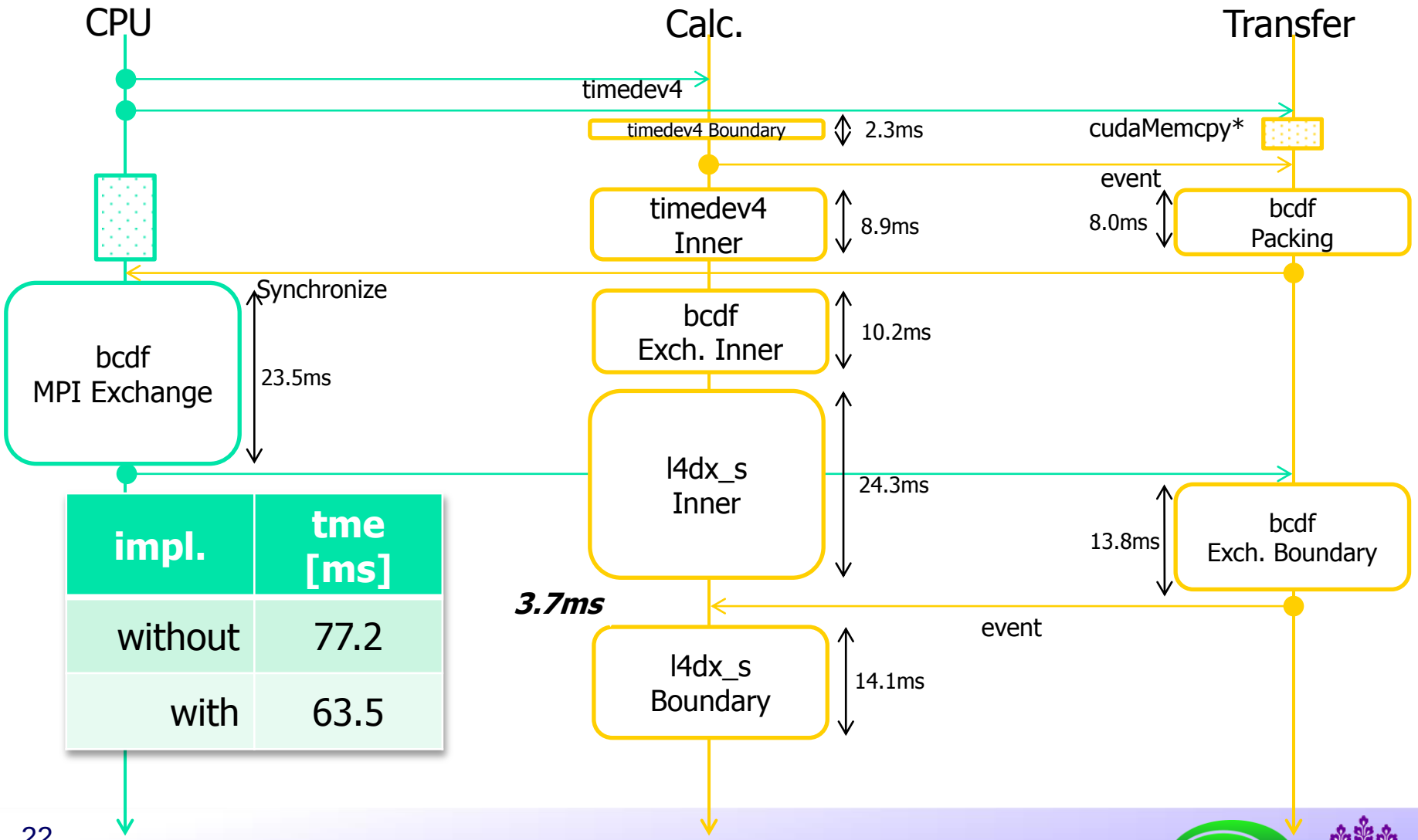
- GT5D Fortran code -> PGI CUDA Fortran
- There are some parts with overlapped execution over CPU and GPU, but basically use CPU only for MPI communication
- Since HA-PACS node has 16 cores (2 sockets) and 4 GPUs, mapping MPI process with 4core : 1GPU and running 4threads (OpenMP) on each MPI process
- Large functions correspond to CUDA Kernels and main body loops are also implemented as "pseudo function" of CUDA Kernels



BCDF function (stencil) implementation



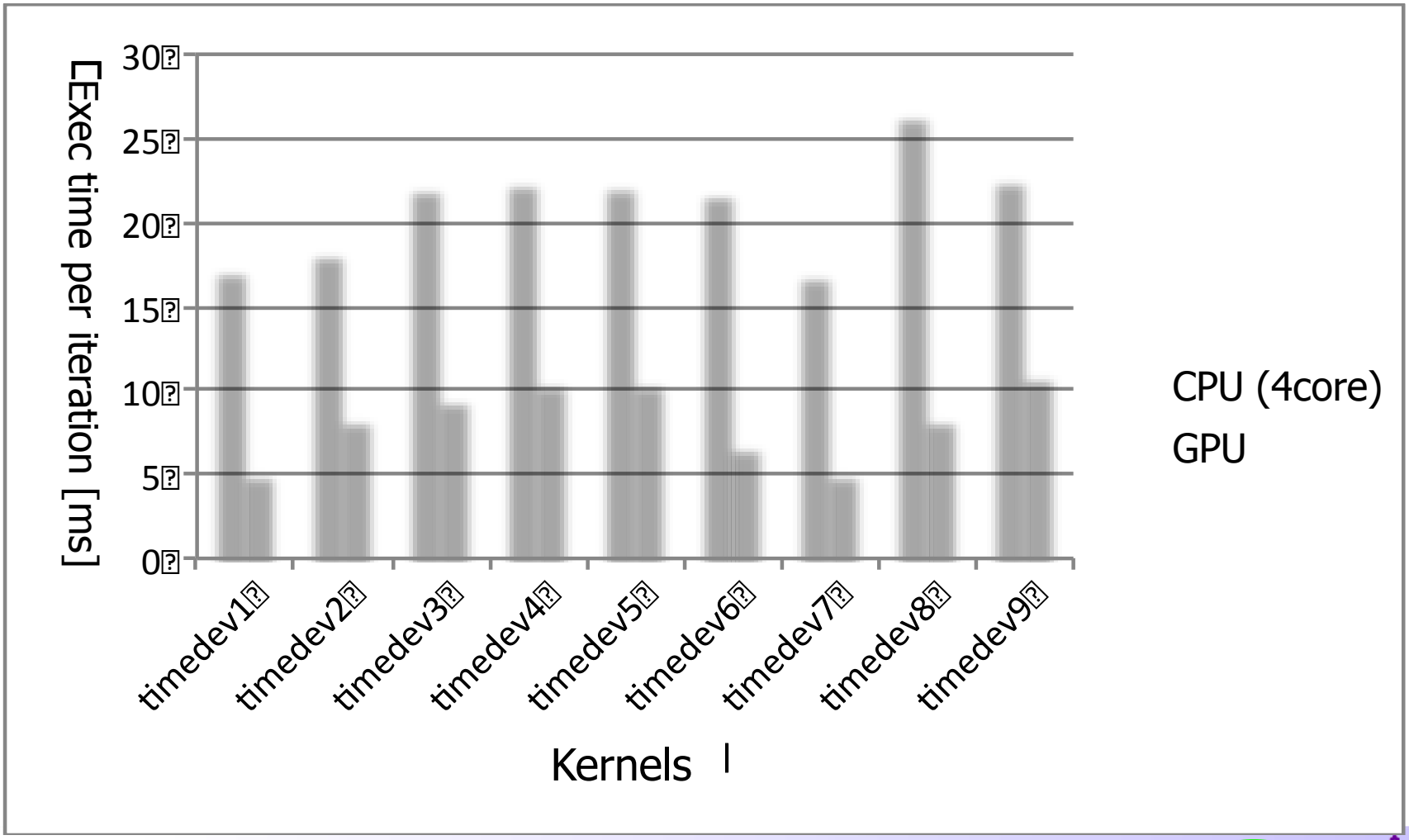
Performance on communication overlapping



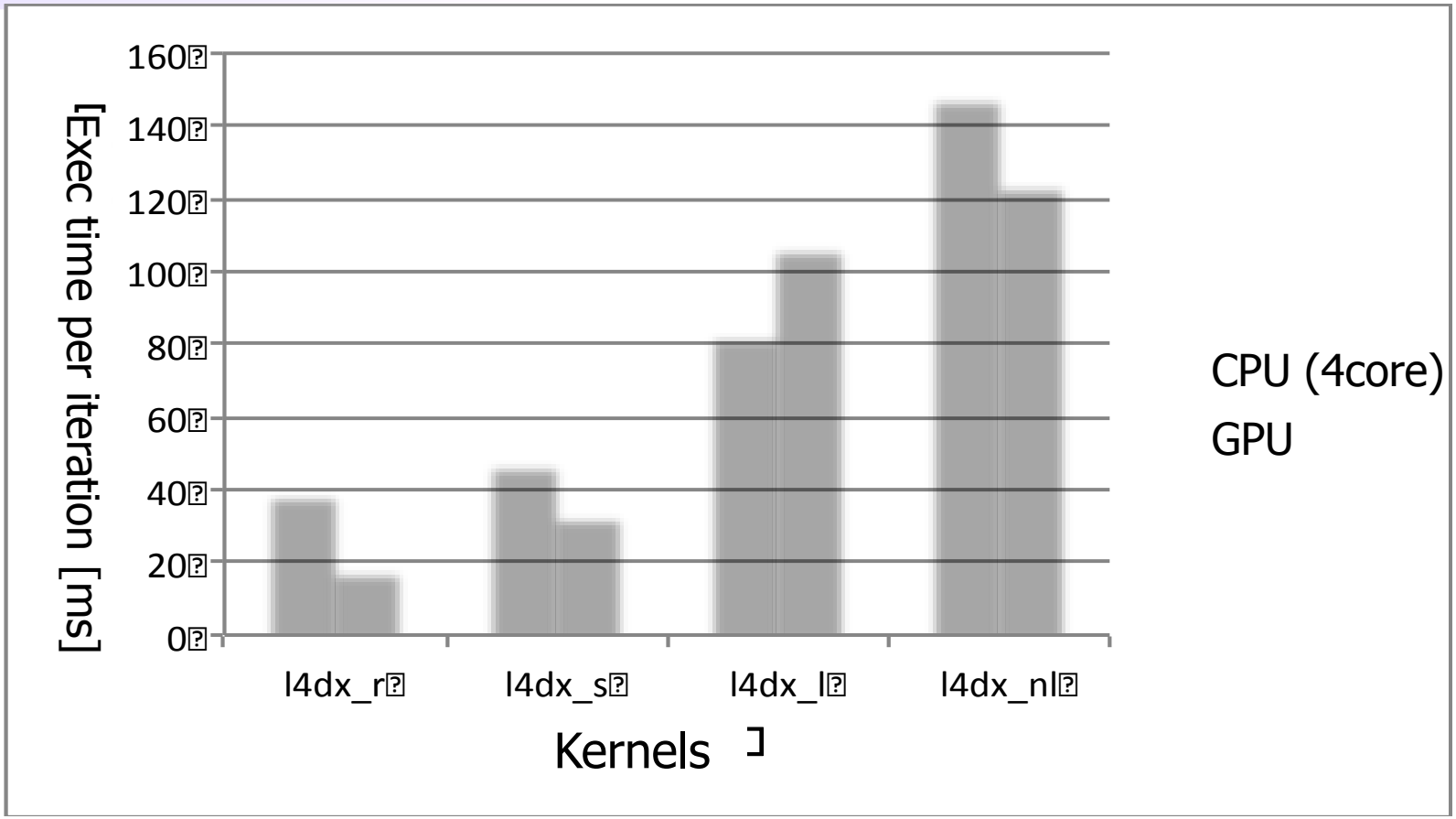
impl.	tme [ms]
without	77.2
with	63.5

performance improvement by GPU

2.5~4x faster than CPU (1 GPU vs 4-core CPU)



function level performance



main function l4dx_r is 2.2x faster than CPU but others are not enough

Overall Performance

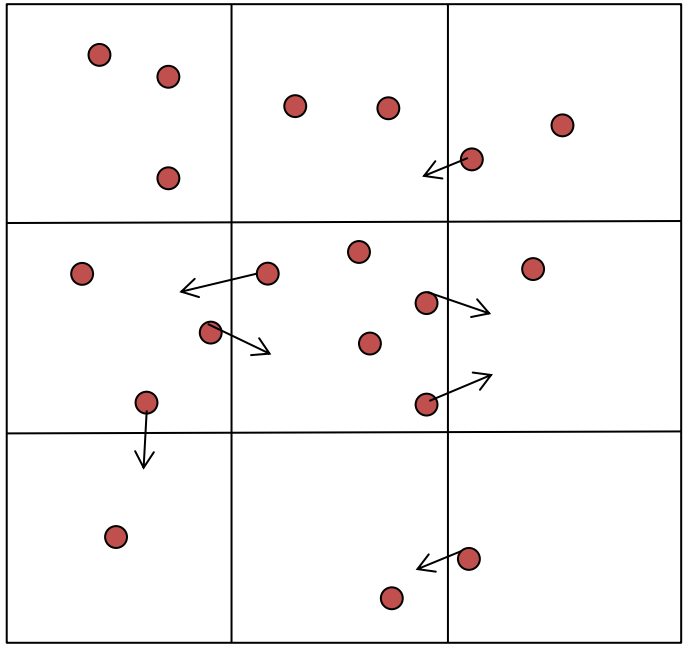
	time/iteration[s]	Speedup to CPU
CPU	33.9	-
GPU (no-Overlap)	14.7	2.31
GPU (Overlap)	13.7	2.47

- Performance comparison with “16 core CPU” vs “4GPU (+16 core CPUs)”
  ⇒ 2.47x speedup
- Main bottlenecks
  - BCDF: MPI call
  - FLD_SFLS: requires MPI comm. and PIC data summation over large field, and currently not using atomic operation
  - LFP: a number of small amount of data copy between CPU and GPU

Porting GTC-P to XMP

- GTC-P: Princeton version of GTC (Gyrokinetic Toroidal Code) for particle oriented fluid dynamics with turbulence
- Since it is a sort of PIC (Particle In Cell) code, we need to treat both variables on mesh-distributed fixed physical domain and particles moving around in these grids at every time step
- Global view (distributed array) is suitable for mesh-distributed domain space to be directly mapped on node grid for minimized communication
- Particle data is difficult to map on nodes statically as well as to localize and bind to local domain data
- Strategy:
  - 3D space domain variables – in global view model
  - Particle data moving around space – in local view model with coarray
- This strategy can be applied commonly for most of PIC code including MD

Image of mixed communication view



```
double f[X][Y];
double p[3][3][N/2], pn[3][3][N/2];
double myp[N];  /* myp is for my particles now */
#pragma xmp align [i][j] with tpl(i,j):: f
#pragma xmp shadow f[1:1][1:1]
#pragma xmp coarray pn:[*,*]

for(t=0; t<TIME; t++){
  /* f (space domain) computation */
#pragma xmp reflect(f)
  /* f, myp, pn computation */
  /* calculate the coordinate of particles in myp
     then pack to p[0][0][*] ~ p[2][2][*] */
  for(i=; i<3; i++)
    for(j=0; j<3; j++)
      if(i!=1 && j!=1){
        pn[2-i][2-j][0:N/2]:[mex+i-1,mey+j-1]
            =p[i][j][0:N/2];
          }
#pragma xmp sync_memory
}
```

Preliminary performance on mixed model