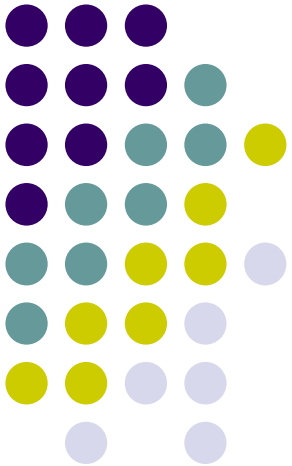
2017
Practical Parallel Computing (実践的並列コンピューティング) No. 15

GPU Programming with CUDA (4)



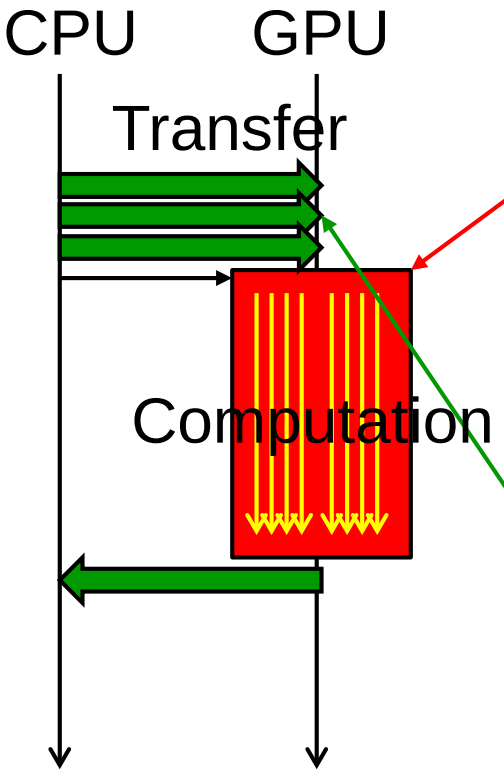
Toshio Endo

School of Computing & GSIC

endo@is.titech.ac.jp

Considering Performance of CUDA Programs

- It is best to reduce of amount of computation & transfer!
- Other approaches are …



Reduce computation time on GPU

- Tuning number of threads
- Reduce non-coalesced access
- Reduce divergent branches

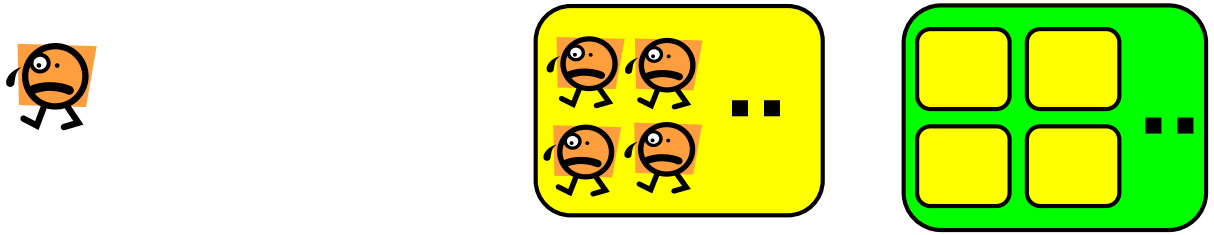
:

Hide transfer time

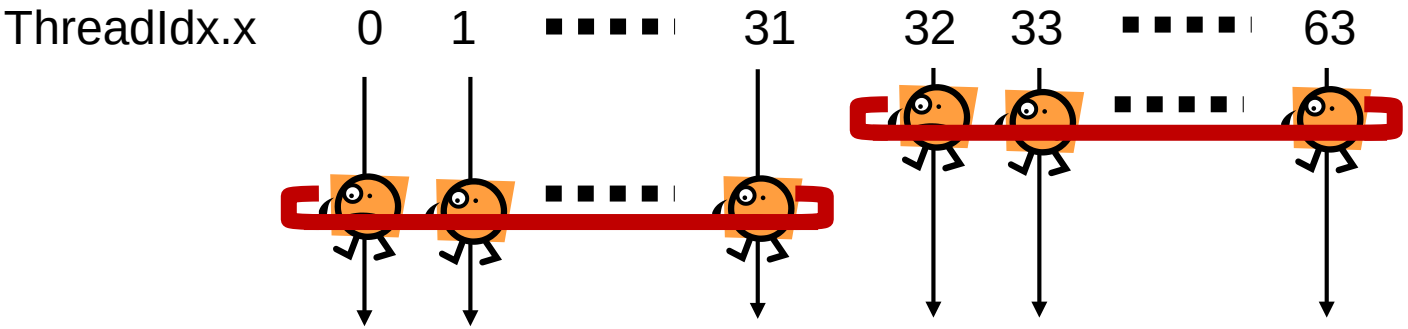
- Asynchronous transfer with cudaStream

Warp: Internal Execution Unit

thread < warp < thread block < grid

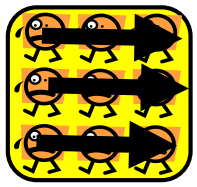


- Threads in a thread block are internally divided into "warp", a group of contiguous 32 threads
- 32 threads in a warp always are executed synchronously
  - They execute the same instruction simultaneously
  - There is only one program counter for 32 threads! → Structure of a GPU core is simplified



Observations due to Warps

- If number of threads per block (blockDim) is not *32 x n*, it is inefficient
  - Even if blockDim=1, the system creates a warp for it
- Characteristics in memory addresses accessed by threads in a warp affect the performance
  - Coalesced accesses are fast
- Characteristics in branch (such as “if”) affect the performance
  - Divergent branches are slow

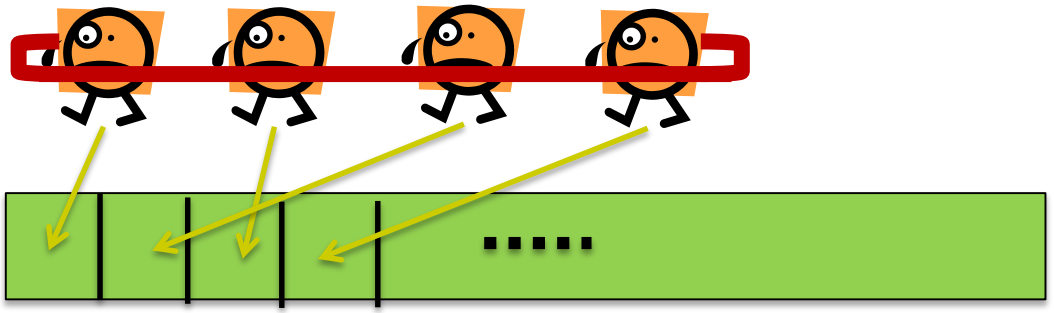


※ In multi-dimensional cases (blockDim.y>1 or blockDim.z>1), “neighborhood” is defined by x-dimension

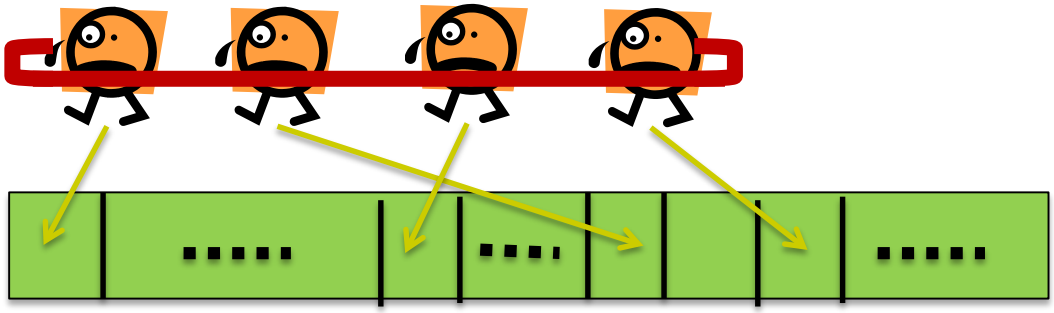
Coalesced Access

- When threads in a warp access “neighbor” address on memory (coalesced access), it is more efficient
  - Here, addresses are “neighbor”, if they are in a 128-byte segment (See Appendix G.4.2 in CUDA Programming Guide)

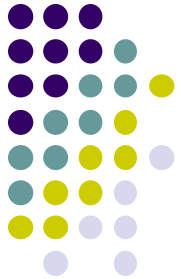
Coalesced access
→ **Faster**



Non-coalesced access
→ **Slower**



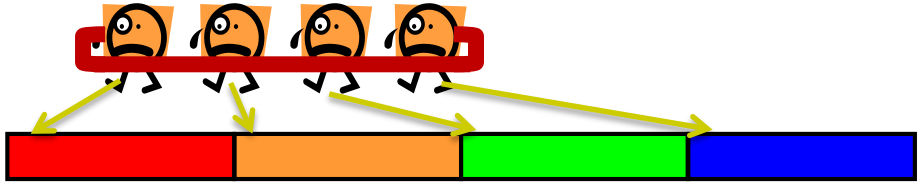
Accesses in Sample Programs



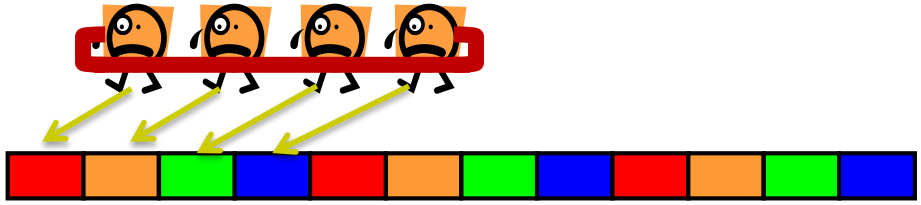
- Accesses in inc-par, inc-rr are coalesced
- Accesses in mm-cuda/mm.cu are coalesced
  - mm-cuda/mm-nc.cu is an example with non-coalesced access (What are different with mm.cu?)

- In CUDA, “cyclic distribution” of arrays is better than “block distribution”, popular in OpenMP

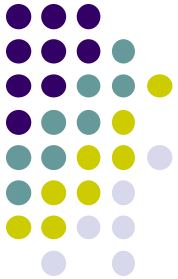
Block distribution
→Slow in CUDA



Cyclic distribution
→Faster in CUDA
(Maybe slower in OpenMP)



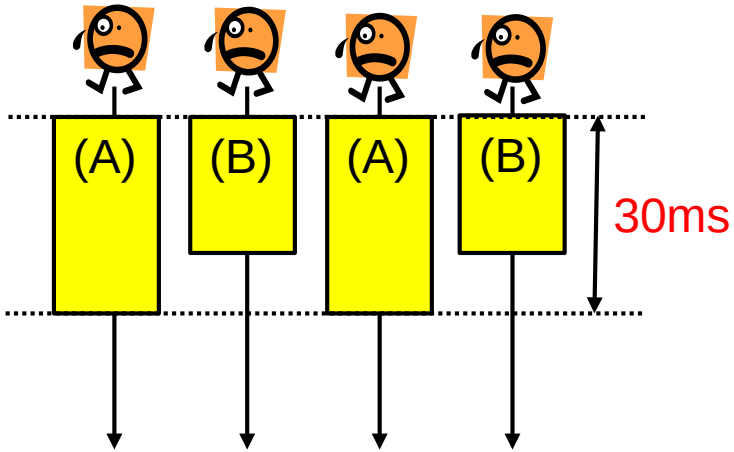
Considering Branches in Parallel Programs



Consider this code. How long is execution time?

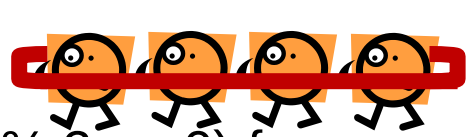
```
if (thread-id % 2 == 0) {
      :  // (A) 30msec
} else {
      :  // (B) 20msec
}
```

On CPU (OpenMP)



On GPU, threads in a warp must execute the same instruction. What happens?

Branches on GPU (1)



```
if (thread-id % 2 == 0) {
```

```
} else {
```

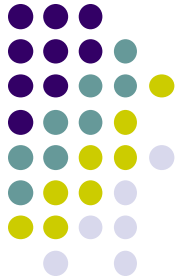
```
}
```

Some threads are made sleep
Both “then” and “else” are executed!

→ Answer to previous question is 50ms !

※ Similar cases happen in for, while…

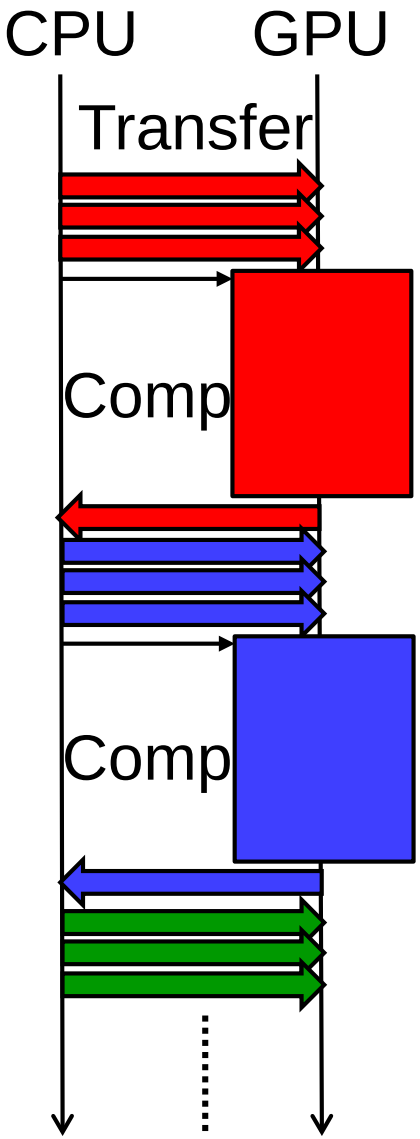
Branches on GPU (2)



- As exceptional cases, if threads in a warp "agree" in branch condition, either "then" part or "else" part is executed → Efficient!
- If there is difference of opinion (previous page), it is called a divergent branch

→ Agreement among buddies is important

Considering Data Transfer Costs



Example case: We are going to multiple matrix multiplications.

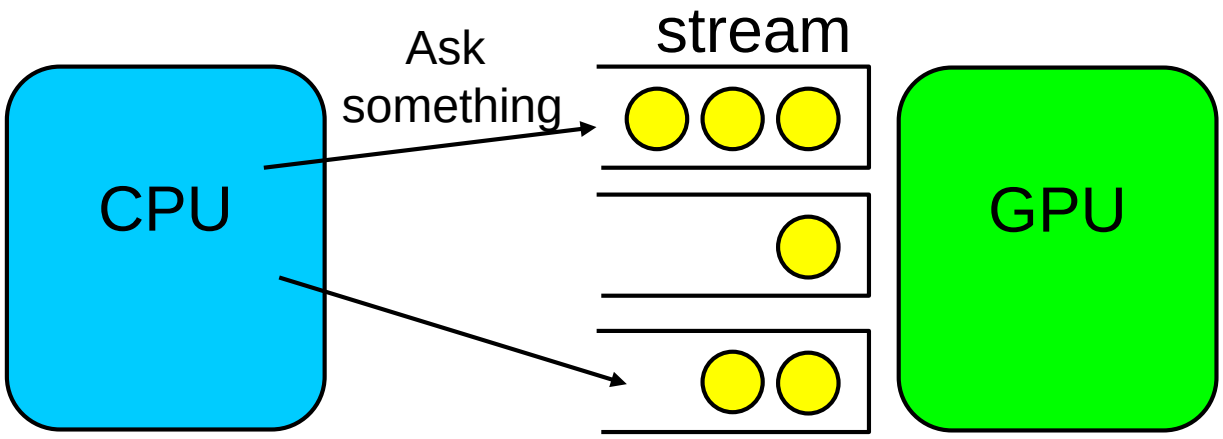
- Input data are on host memory
  - C1 = A1 × B1
  - C2 = A2 × B2
  - ….
  - Cn = An × Bn
- In default, GPU cannot compute during transfer

→ cudaStream is useful for hiding transfer costs

Asynchronous Executions with cudaStream (1)

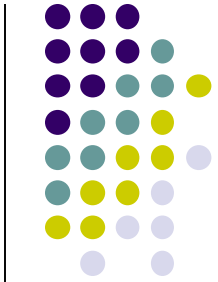
What are streams?

- GPU's "service counters" that accept tasks from CPU
  - Each stream looks like a queue
- "Tasks" from CPU to GPU include
  - Data transfer (Host → Device)
  - GPU kernel function call
  - Data transfer (Device → Host)



All of sample programs are using the "default stream"

Asynchronous Executions with cudaStream (2)



Create a stream

```
cudaStream_t str;
cudaStreamCreate(&str); // Create a stream
```

Data transfer using a specific stream

```
cudaMemcpyAsync(dst, src, size, type, str);
```

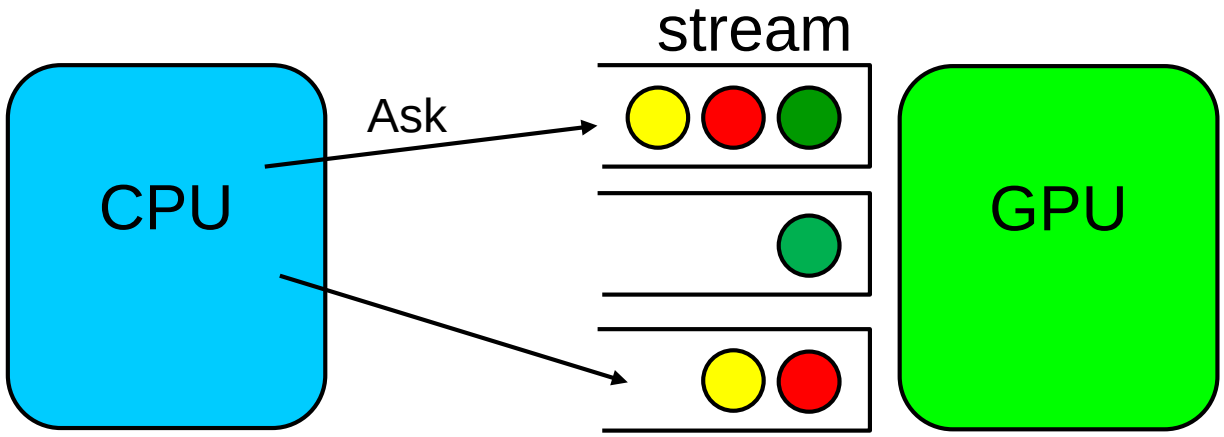
Call GPU kernel function using a stream

```
func<<<gs, bs, 0, str>>>( … );
// 3rd parameter is related to for “shared memory”
```

Wait until all tasks on a stream are finished

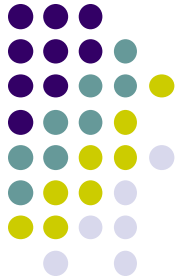
```
cudaStreamSynchronize(str);
```

How GPU does Tasks



- Tasks on the same stream is done in FIFO
- If tasks are in different streams, and have different kinds, they may be done simultaneously
  - Kinds: H→D, kernel, D→H
  - Note: If tasks are in the same kind, no speed up

Speed Up with Overlap of Computation and Transfer



n streams can be used for n independent tasks

- C1 = A1 × B1 (includes H->D, Calc, D->H)
- C2 = A2 × B2

….

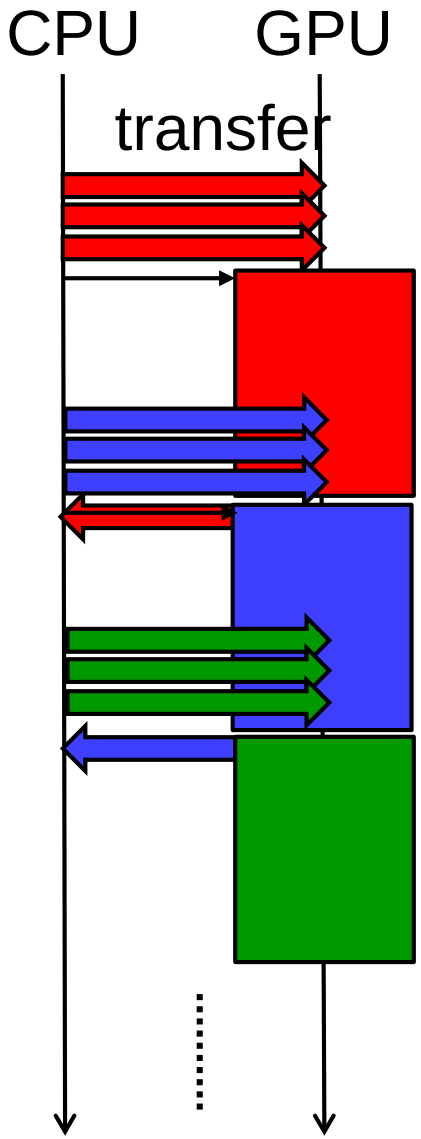
- Cn = An × Bn

→ We will see speed up since

(Total comp time + Total trans time)

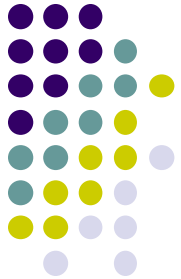
is improved to

max(Total comp time, Total trans time)



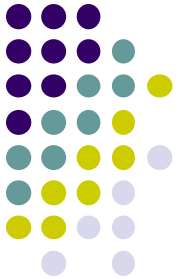
This is not a unique solution;
Use 2 or 3 streams repeatedly → we can save memory and stream resources

More Things to Study

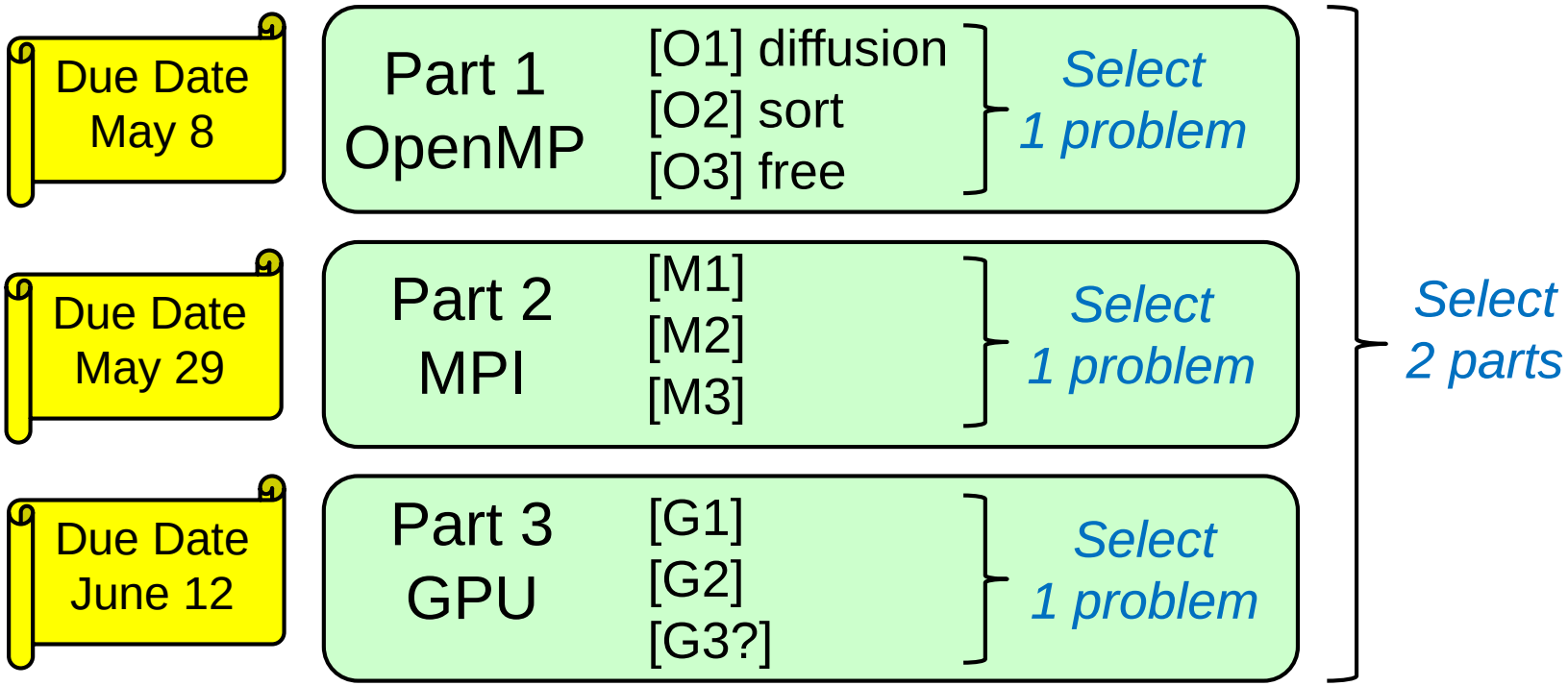


- Using CUDA shared memory
  - fast and small memory than device memory
- Unified memory in recent CUDA
  - cudaMemcpy can be omitted for automatic data transfer
- Using multiple GPUs towards petascale computation
  - MPI+CUDA!
- More and more…

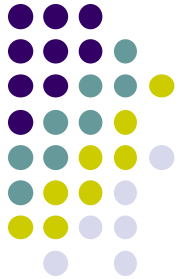
Assignments in this Course



- There is homework for each part. Submissions of reports for 2 parts are required
- Also attendances will be considered



Assignments in GPU Part (Abstract)



Choose one of [G1]—[G3], and submit a report

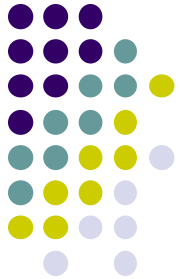
Due date: June 12 (Monday)

[G1] Parallelize “diffusion” sample program by CUDA.

[G2] Evaluate speed of “mm-cuda” in detail.

[G3] (Freestyle) Parallelize *any* program by CUDA.

Notes in Submission



- Submit the followings via OCW-i

  (1) A report document
  - A PDF or MS-Word file
  - 2 pages or more
  - in English or Japanese (日本語もok)

  (2) Source code files of your program
- Report should include:
  - Which problem you have chosen
  - How you parallelized
    - It is even better if you mention efforts for high performance or new functions
  - Performance evaluation on TSUBAME2
    - With varying number of processor cores
    - With varying problem sizes
    - Discussion with your findings
    - Other machines than TSUBAME2 are ok, if available