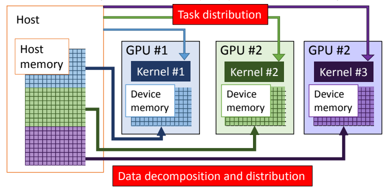
Preliminary Estimation on Automating Multi-dimensional Data Decomposition for Multi-GPU Systems

Ryotaro Sakai[1] Fumihiko Ino[2] Kenichi Hagihara[2]

[1]School of Engineering Science, Osaka University
[2]Graduate School of Information Science and Technology, Osaka University

Background

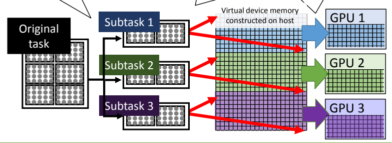
- Multi-GPU systems are useful for accelerating large-scale data processing
- One drawback is more programming efforts needed for data decomposition and task distribution
  - Device memory has a smaller capacity than host memory



Previous system [1]

- A multi-GPU system capable of executing a single-GPU code

• Original task is automatically decomposed into small subtasks	• Runtime access analysis computes upper and lower bounds (UB and LB) on memory range	• Small data segment between UB and LB are transferred to GPUs



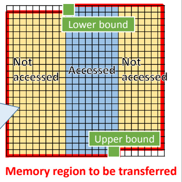
[1] Jungwon Kim, Honggyu Kim, Joo Hwan Lee, Jaejin Lee, Achieving a single compute device image in OpenCL for multiple GPUs, Proc. PPoPP 2011, pp.277-287, 2011.

Issue on runtime access analysis

- LB and UB are given on 1-D memory space though practical data are usually accessed as multi-dimensional data

Ex. 2-D data accessed with a column-wise pattern

- Memory region specified with previous method [1] includes many columns that never accessed



- Performance is limited due to increased amount of CPU-GPU data transfer
- Device memory is wasted due to non-referred region

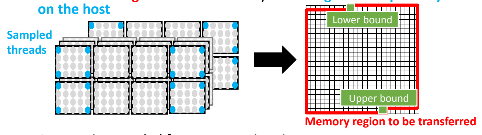
Overview

- **Goal:** an automated framework that facilitates large-scale computation on a multi-GPU system
  - The framework automates data decomposition, CPU-GPU data transfer and task distribution
- **Method:** an extension of previous system [1] such that multi-dimensional data can be efficiently processed on a multi-GPU system
  - A more efficient runtime analysis for multi-dimensional data ← this poster
  - A translator that enables a single-GPU program to run on a multi-GPU system (not yet)
- **Preliminary estimation:** estimated segment size
  - For matrix multiplication, our method reduces the segment size by 88%
  - Multiplication of 40,000 × 40,000 matrices can be done with 1 GB of device memory (instead of 6 GB of device memory)

Previous runtime analysis [1]

A sampling run approach

- Given a task, the runtime estimates memory region to be accessed during kernel execution by executing the task partially on the host

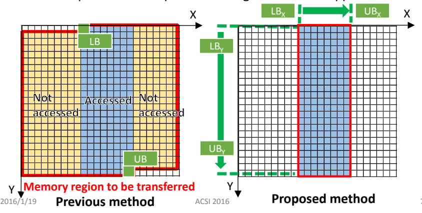


- Assumption needed for correct estimation: memory access pattern must be represented as an affine function
  - Array indexes are given as affine functions comprising thread indexes and thread block indexes

```
A[index] → a_1*l_x + a_2*l_y + a_3*l_z + a_4*w_x + a_5*w_y + a_6*w_z + a_7
```

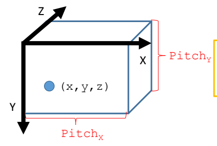
Proposed method

- Basic idea is to extend the previous method [1] to multi-dimensional space
  - Apply the previous method [1] to each dimension to compute lower and upper bounds for every dimension
  - Decompose the data space according to lower and upper bounds



Translating 1-D space to multi-dimensional space

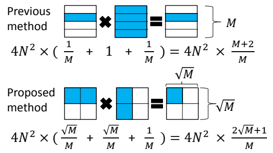
- 1-D space must be translated into *k*-D space to compute lower and upper bounds for each dimension
- Assumption needed for translation: pitch information on each dimension is given by programmers
- Example: translation from `A[index]` to `A[x][y][z]`



```
x = index % Pitch_X
y = index / Pitch_X % Pitch_Y
z = index / Pitch_X / Pitch_Y
```

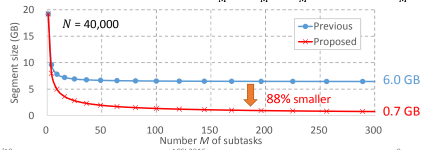
Preliminary estimation

- Segment size of matrix multiplication
  - $N \times N$ matrices each with 4-byte data are decomposed into M subtasks
  - Proposed method reduces segment size by $\frac{M+2}{2\sqrt{M}+1}$



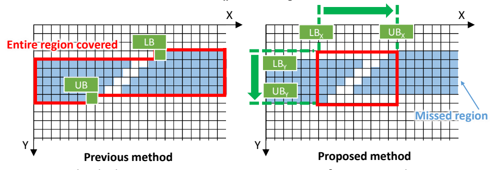
$$4N^2 \times \left(\frac{1}{M} + 1 + \frac{1}{M}\right) = 4N^2 \times \frac{M+2}{M}$$

$$4N^2 \times \left(\frac{\sqrt{M}}{M} + \frac{\sqrt{M}}{M} + \frac{1}{M}\right) = 4N^2 \times \frac{2\sqrt{M}+1}{M}$$



Discussion on assumptions

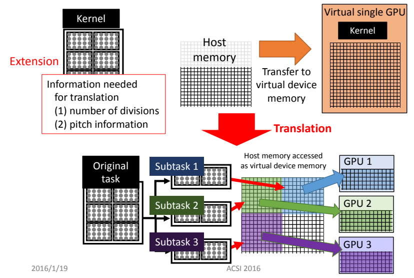
- An additional assumption on memory access pattern is needed to ensure correct decomposition
  - The affine feature is not sufficient due to mod operations
    - `x = index % Pitch_X`
    - `y = index / Pitch_X % Pitch_Y`



- We think this assumption is not too strict for practical apps

Discussion on translator

- Input: extended CUDA code Output: pure CUDA code



Conclusion

- A method for automating multi-dimensional data decomposition for multi-GPU systems
  - An extension of the previous method [1]
  - Data decomposition based on a sampling run approach
- Preliminary estimation for matrix multiplication
  - 88% smaller data segment will be generated
  - An additional assumption is needed to obtain correct decomposition; the affine feature is not sufficient
- Future work
  - Clarify the additional assumption for correct decomposition
  - Performance evaluation using real CUDA programs