Algorithmic Skeleton Framework for the Orchestration of GPU Computations

Ricardo Marques, Herve Paulino, Fernando Alexandre and Pedro D. Medeiros
Universidade Nova de Lisboa

Euro-Par 2013, Aachen, Germany

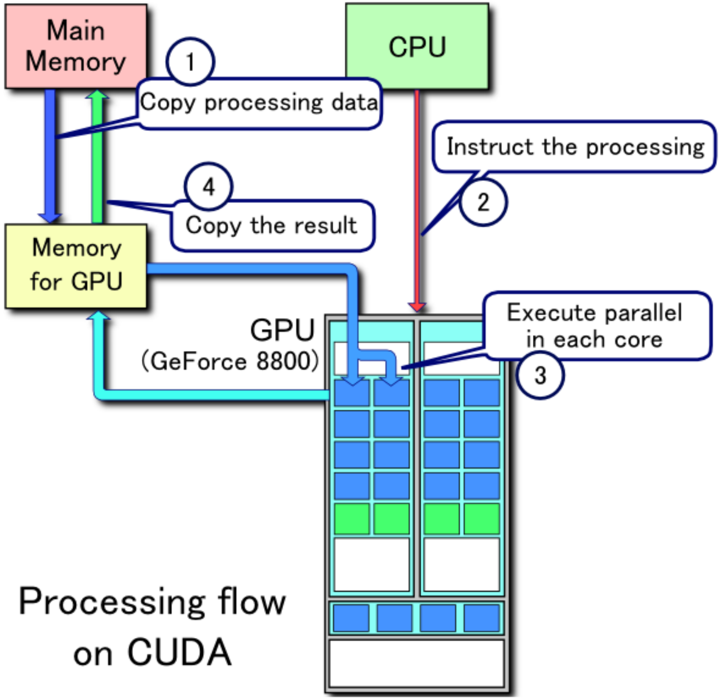
Presented by: Jingjing Du

Introduction

- GPUs (Graphics Processing Unit) are highly parallel computation devices
    - available libraries and languages requires a lot of deep knowledge of the platform
- This work goal is to simplify the management of computation and data transfers on GPU devices

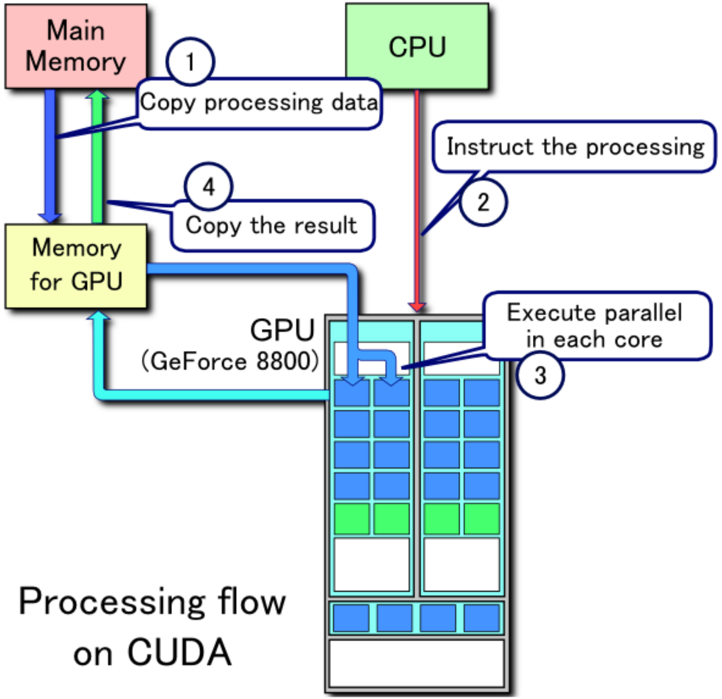
1. Where is the problem?

GPU Programming Flow



cited from http://en.wikipedia.org/wiki/CUDA

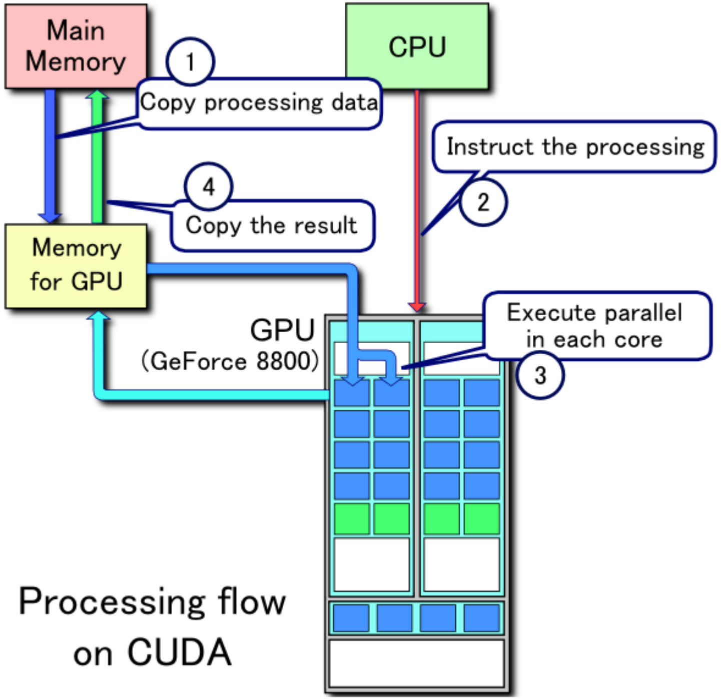
GPU Programming Flow



cited from http://en.wikipedia.org/wiki/CUDA

Modern GPUs allow overlaps of data transfers and kernel executions.

GPU Programming Flow



cited from http://en.wikipedia.org/wiki/CUDA

Modern GPUs allow overlaps of data transfers and kernel executions.

Problem:

not easy to use with current GPU programming frameworks. (synchronization requires a lot of coding)

2. How to solve the problem?

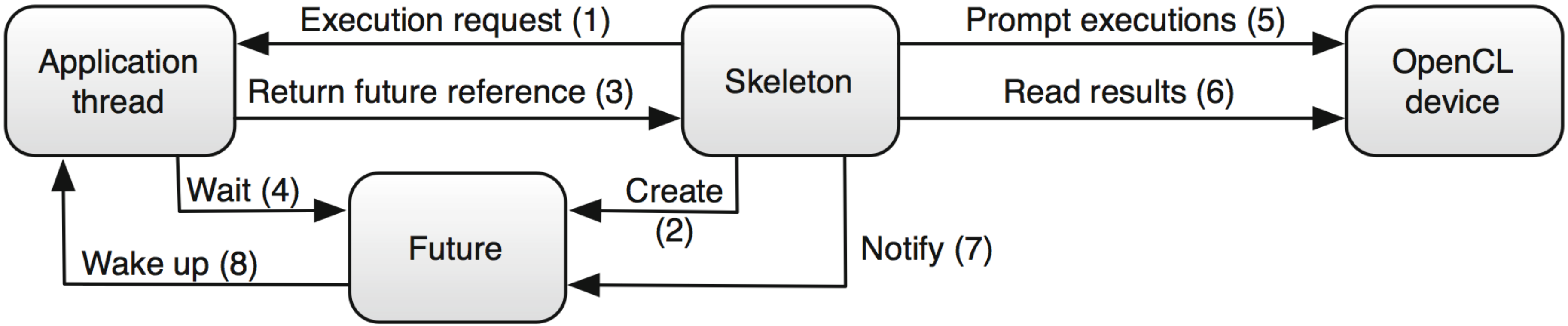
Marrow

An algorithmic skeleton framework(ASkF) to simplify orchestration of OpenCL computations

Main achievement:

Parallelize data transfer and kernel execution.

Marrow Execution Model



Marrow Concepts

- ❑ Nodes
- ❑ Skeletons
  - ❑ Pipeline
  - ❑ Loop
  - ❑ Stream
  - ❑ Map

Nodes

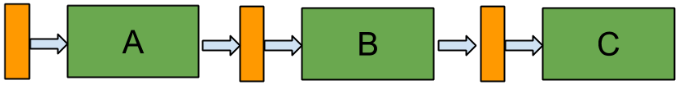
- ❑ Leaf nodes
    - ❑ Only KernelWrapper
- ❑ Inner nodes
    - ❑ Skeletons
- ❑ Root node
    - ❑ Manages execution and synchronization of Inner and Leaf nodes

Skeletons

- Skeletons
  - Organize nodes execution order
  - It is a node itself
  - Can be nested

Skeletons Type

Pipeline

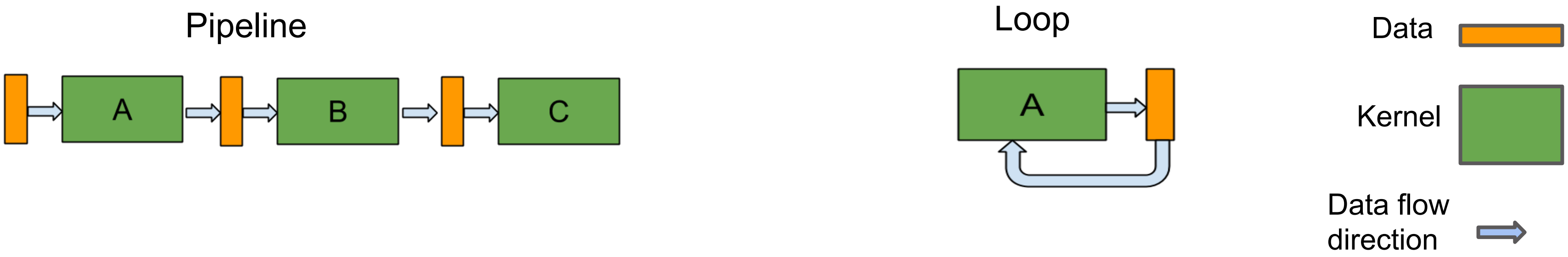


Data 🟧

Kernel 🟩

Data flow direction ⇨

Skeletons Type

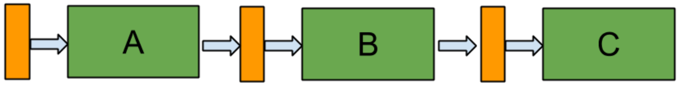


Pipeline

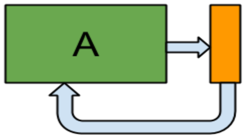
Loop

Skeletons Type

Pipeline



Loop

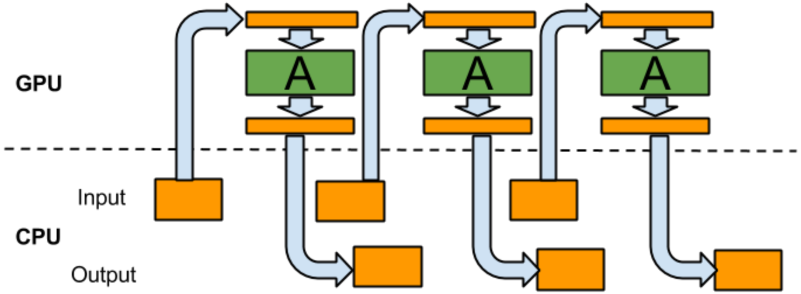


Data 🟧

Kernel 🟩

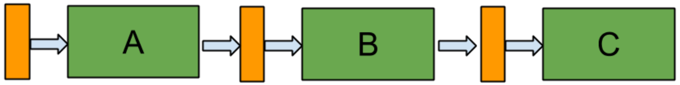
Data flow direction ⇨

Stream
(not nestable)

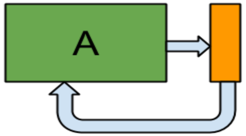


Skeletons Type

Pipeline



Loop

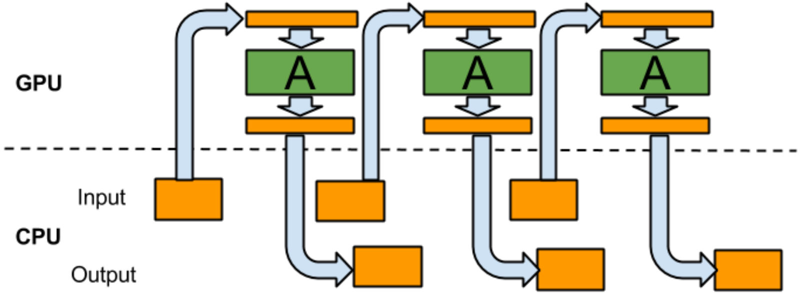


Data 🟧

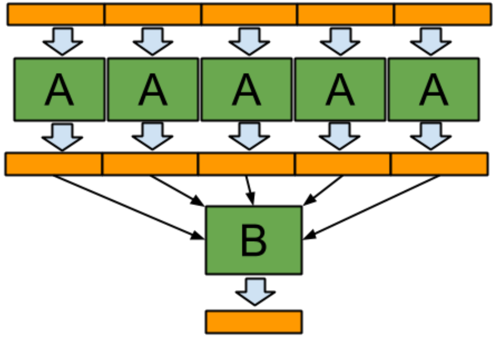
Kernel 🟩

Data flow direction ⇨

Stream
(not nestable)



Map Reduce
(not nestable)



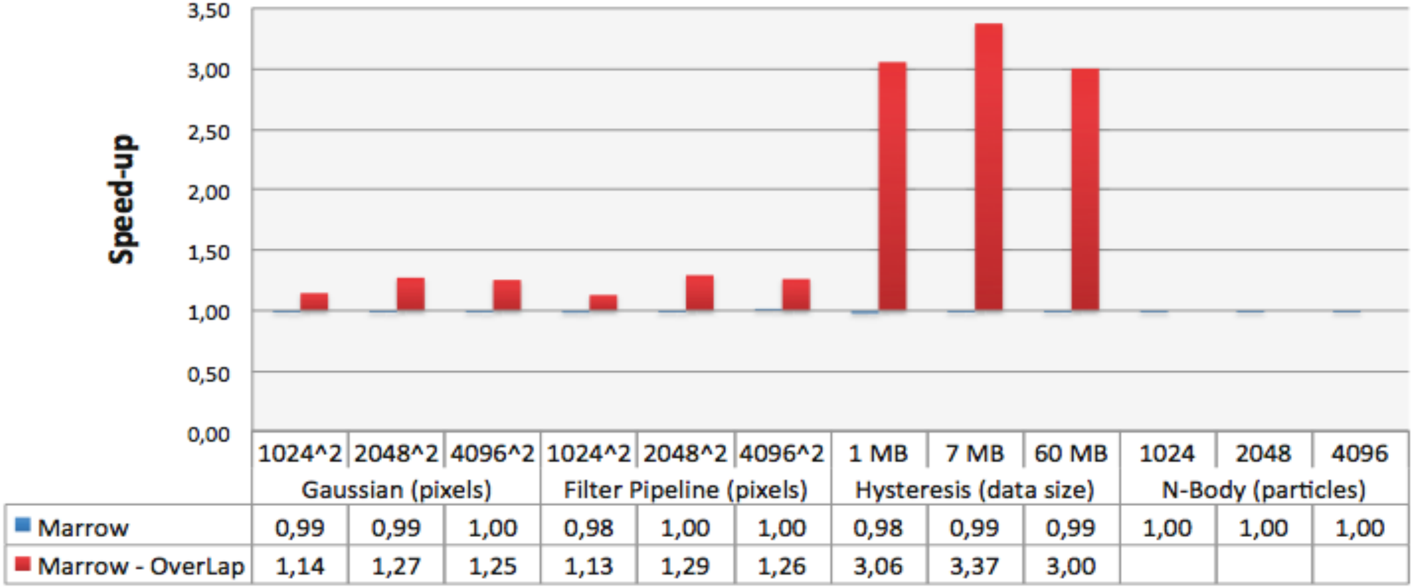
Code Example

```
// … instantiate kernel wrappers
unique_ptr<IExecutable> gaussKernel (new KernelWrapper ( gaussNoiseSourceFile,
        gaussNoiseKernelFunction, inputDataInfo, outputDataInfo, workSize ));
// … instantiate inner skeletons
unique_ptr<IExecutable> p1 ( new Pipeline ( gaussKernel, solariseKernel));
unique_ptr<IExecutable> p2 ( new Pipeline ( p1, mirrorKernel));
// instantiate root skeleton
Stream *s = new Stream (p2, 3); // overlap with 3 concurrent executions
// request skeleton executions
for (int i = 0; i < numberOfSegments; i++) {
        inputValues [0] = …; // offset in the input image
        outputValues [0] = …; // offset in the output image
        futures [i] = s-> write ( inputValues, outputValues);
}
// wait for results ; delete s and resources (e.g the futures)
```

3. Result Analysis

Results

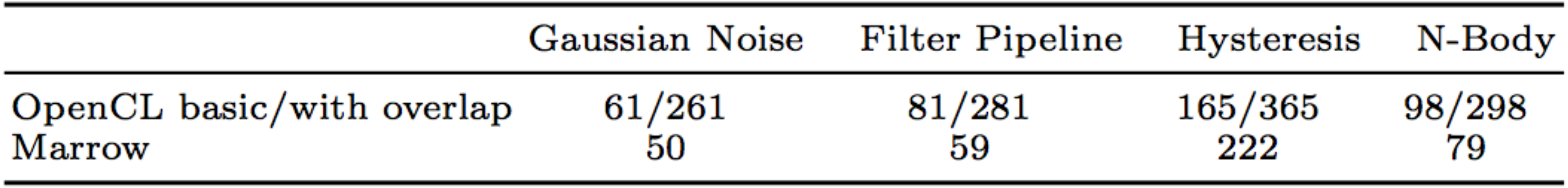
1. Better throughput with overlap



	1024^2	2048^2	4096^2	1024^2	2048^2	4096^2	1 MB	7 MB	60 MB	1024	2048	4096
	Gaussian (pixels)			Filter Pipeline (pixels)			Hysteresis (data size)			N-Body (particles)		
🟦 Marrow	0,99	0,99	1,00	0,98	1,00	1,00	0,98	0,99	0,99	1,00	1,00	1,00
🟥 Marrow - OverLap	1,14	1,27	1,25	1,13	1,29	1,26	3,06	3,37	3,00			

Result

2. Code simplification



	Gaussian Noise	Filter Pipeline	Hysteresis	N-Body
OpenCL basic/with overlap	61/261	81/281	165/365	98/298
Marrow	50	59	222	79

Conclusion

Marrow: a ASkF for the orchestration of OpenCL computations

- ❏ enriching the set of skeletons
- ❏ supporting skeleton nesting
- ❏ easy and efficient overlap programming

https://bitbucket.org/MarrowTeam/marrow/overview

Thank you!