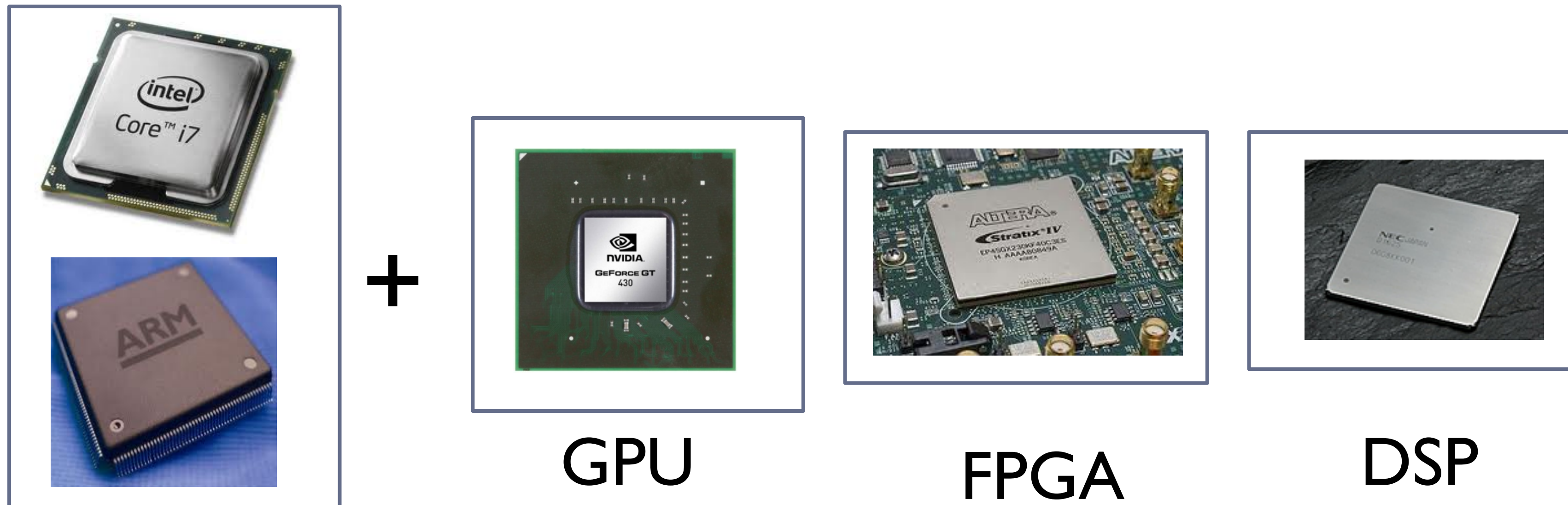
Communication Library to Overlap Computation and Communication for OpenCL Application

Toshiya Komoda, Shinobu Miwa, Hiroshi Nakamura
Univ.Tokyo

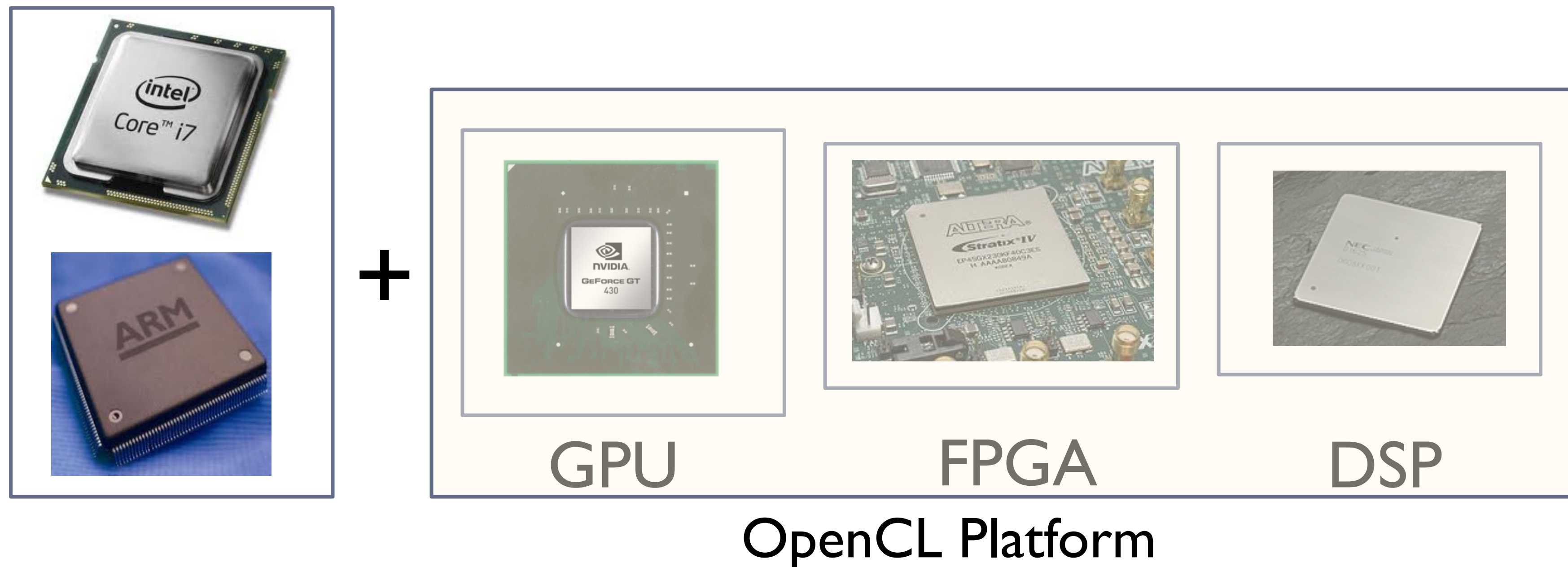
What is today's talk about ?

- Heterogeneous Computing System with data-parallel accelerators.
  - High Performance
  - High Energy Efficiency
- The programming is well known to be difficult.
  - Low portability
  - Low productivity



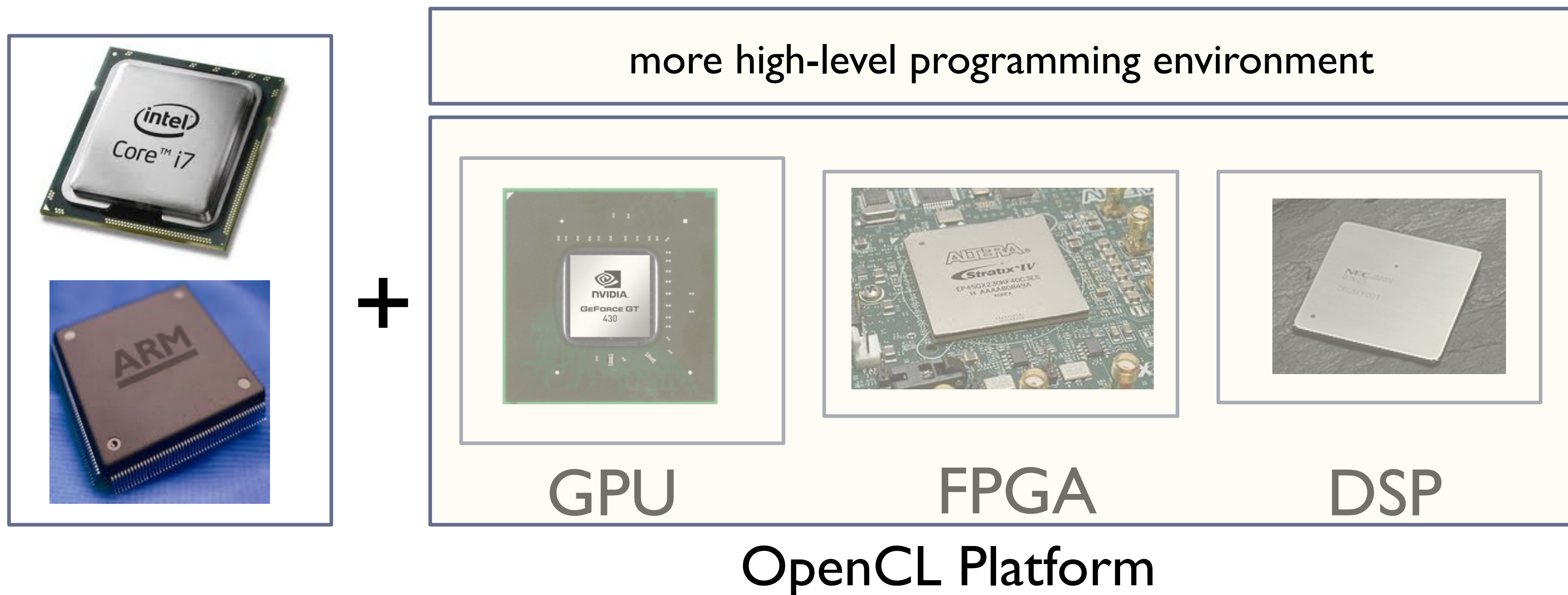
OpenCL

- Enable us to develop portable programs for accelerators
  - can solve the portability problem.
- Productivity is still problem
  - OpenCL provides only very low-level primitives.



OpenCL

- Enable us to develop portable programs for accelerators
  - can solve the portability problem.
- Productivity is still problem
  - OpenCL provides only very low-level primitives.



Programming flow for accelerators

- In most cases, we port an existing codes to accelerator platform
- Required procedure for the porting
  1. write parallel kernel codes (Kernel Code)
     - auto-parallelization technique can help.
  2. write communication codes (CPU Code)
     - memory space of accelerator is isolated from CPU memory.

Programming flow for accelerators

- In most cases, we port an existing codes to accelerator platform
- Required procedure for the porting
  1. write parallel kernel codes (Kernel Code)
     - auto-parallelization technique can help.
  2. write communication codes (CPU Code)
     - memory space of accelerator is isolated from CPU memory.

OpenCL Application(CPU Code)

```
float h_mem1[100];
float h_mem2[100];
cl_mem d_mem1, d_mem2;
initialize(h_mem1);

// Allocate Device Memory
d_mem1 = clCreateBuffer(...);
d_mem2 = clCreateBuffer(...);

// Blocking Host -> Device Transfer

clEnqueueWriteBuffer
    (..,d_mem1,..,h_mem1,..);
```

```
// set input and output device memory.
 clSetKernelArg(kernel,..,d_mem1,..);
 clSetKernelArg(kernel,..,d_mem2,..);

 // Execute on Device
clEnqueueNDRangeKernel(..,kernel,...);
clFinish(cmdQueue);

// Blocking Device -> Host Transfer
clEnqueueReadBuffer
    (..,d_mem2,..,h_mem2,...);

for(i = 0; i< num;i++){
   printf("%d\n" ,h_mem2);
}
```

OpenCL Application(CPU Code)

```
float h_mem1[100];
float h_mem2[100];
cl_mem d_mem1, d_mem2;
initialize(h_mem1);

// Allocate Device Memory
d_mem1 = clCreateBuffer(...);
d_mem2 = clCreateBuffer(...);

// Blocking Host -> Device Transfer

clEnqueueWriteBuffer
    (..,d_mem1,..,h_mem1,..);
```

```
// set input and output device memory.
 clSetKernelArg(kernel,..,d_mem1,..);
 clSetKernelArg(kernel,..,d_mem2,..);

 // Execute on Device
clEnqueueNDRangeKernel(..,kernel,...);
clFinish(cmdQueue);

// Blocking Device -> Host Transfer
clEnqueueReadBuffer
    (..,d_mem2,..,h_mem2,...);

for(i = 0; i< num;i++){
   printf("%d\n" ,h_mem2);
}
```

Almost all codes are related to manual device memory management.

Problems in Communication of OpenCL

- direct manipulation of device memory via low-level memory primitives
  - programmers need knowledge of the underlying memory architecture
  - it takes much time and error-prone
- large overhead of communication
  - because accelerator memory is physically isolated.
  - can be the performance bottleneck.
    - advanced optimization is needed and it increases programming difficulty

Problems in Communication of OpenCL

- direct manipulation of device memory via low-level memory primitives
  - programmers need knowledge of the underlying memory architecture
  - it takes much time and error-prone

- large overhead of communication
  - because accelerator memory is physically isolated.
  - can be the performance bottleneck.
    - advanced optimization is needed and it increases programming difficulty

We need more easy-to use & optmized communication system

Communication Library for OpenCL Applications

- How should we abstract communication ?
- Observation
  - Communication management & optimization process is divided into
    - Analyze communication pattern of the application
      - Which data is used in the task?
    - Mapping the communication pattern to low-level memory functions
      - preparing device memories
      - handling and optimizing data movement according to communication pattern

Communication Library for OpenCL Applications

- How should we abstract communication ?
- Observation
  - Communication management & optimization process is divided into
    - Analyze communication pattern of the application
      - Which data is used in the task?
    - Mapping the communication pattern to low-level memory functions
      - preparing device memories
      - handling and optimizing data movement according to communication pattern

Programmers can do well

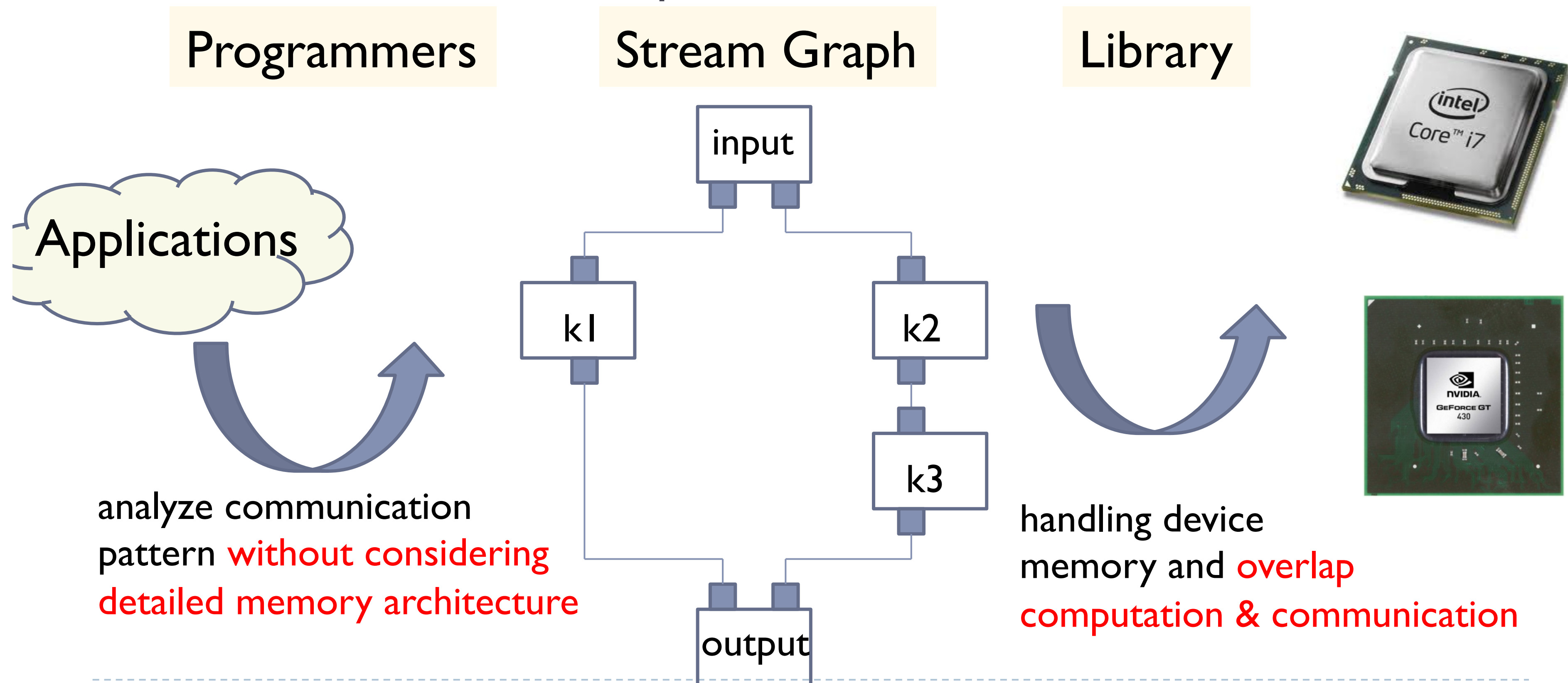
Communication Library for OpenCL Applications

- How should we abstract communication ?
- Observation
  - Communication management & optimization process is divided into
    - Analyze communication pattern of the application
      - Which data is used in the task?
    - Mapping the communication pattern to low-level memory functions
      - preparing device memories
      - handling and optimizing data movement according to communication pattern

System can do well

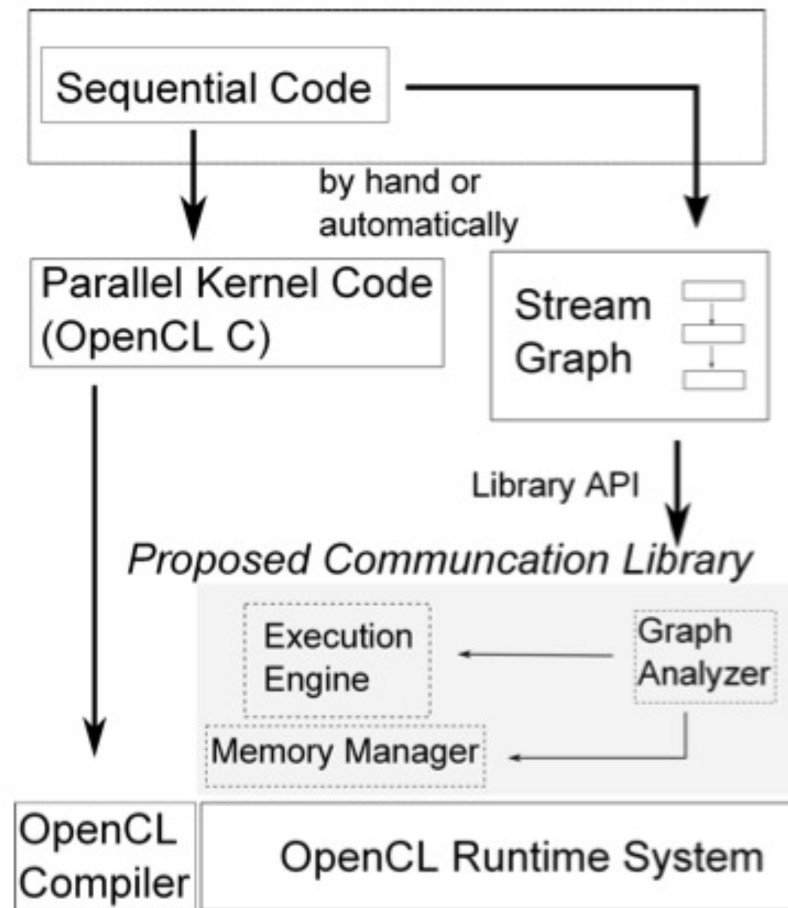
Communication Library for OpenCL Applications

- Stream Graph is Revisited
  - well-studied, simple abstraction to describe producer-consumer relationship between tasks



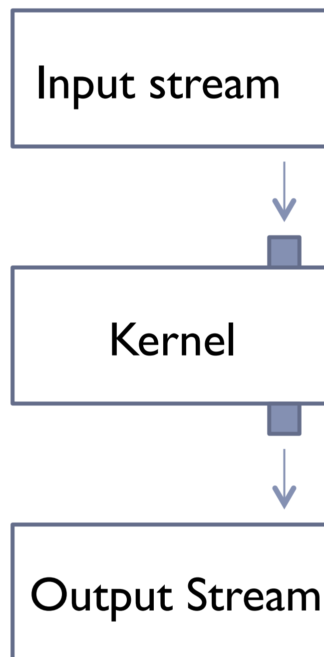
Library Overview

- built on top of the OpenCL platform
  - to keep portability
- data-parallel tasks are written as usual
  - it is beyond the scope of this research



Stream Graph API: Example

Stream Graph



Task Definition

Kernel code

```
__kernel void kernel(
struct p param,
T*  input ,
T* output
)
{

/* operation*/

}
```

Communication Description

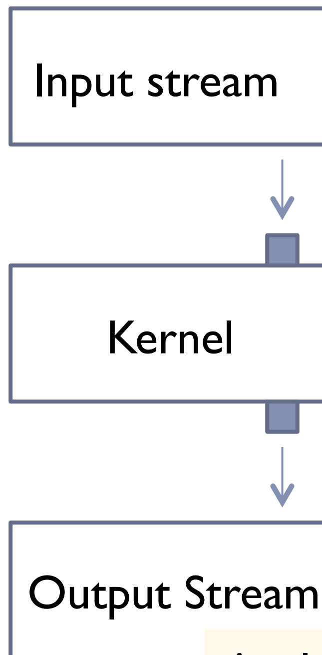
Host Code

```
addInputStream (" i" , total_size, token_size) ;
addOutputStream( "o" , total_size,token_size);

struct kernel_param params;
/* set params for execution */

 addKernel ( "kernel" , params ) ;

connect ( " i" ,  "kernel" , token_size ) ;
connect ( "kernel" , " o" ,  token_size ) ;

initialize();
execute( );
```

Describe a Stream Graph.

Stream Graph API: Example

Stream Graph



Task Definition

Kernel code

```
__kernel void kernel(
struct p param,
T*  input ,
T* output
)
{

/* operation*/

}
```

Communication Description

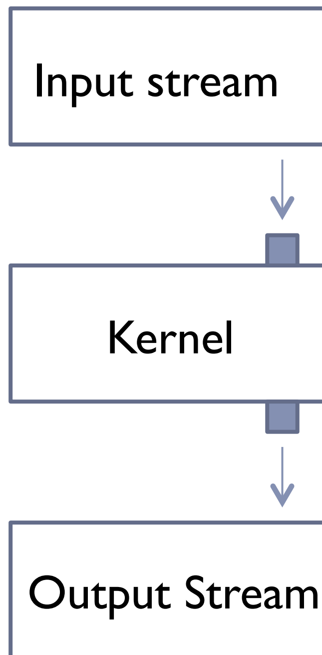
Host Code

```
addInputStream (" i" , total_size, token_size) ;
addOutputStream( "o" , total_size,token_size);

struct kernel_param params;
/* set params for execution */

 addKernel ("kernel" , params ) ;

connect ( " i" ,   "kernel" , token_size ) ;
connect ( "kernel" , " o" ,  token_size ) ;

initialize();
execute( );
```

Analyze a Graph
and map/schedule tasks
and communication.

Stream Graph API: Example

Stream Graph



Task Definition

Kernel code

```
__kernel void kernel(
struct p param,
T*  input ,
T* output
)
{

/* operation*/

}
```

Communication Description

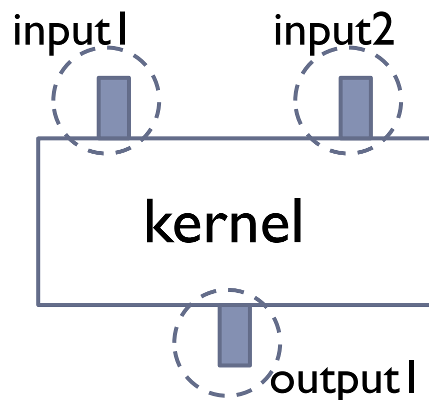
Host Code

```
addInputStream (" i" , total_size, token_size) ;
addOutputStream( "o" , total_size,token_size);

struct kernel_param params;
/* set params for execution */

 addKernel ("kernel" , params ) ;

connect ( " i" ,  "kernel" , token_size ) ;
connect ( "kernel" , " o" ,  token_size ) ;

initialize();
execute( );
```

Execute tasks & communication with asynchronous operation.

Stream Graph API: Task Function Argument

- Small restriction on function argument format
  - Read-only input buffers and write-only output buffers in arguments.



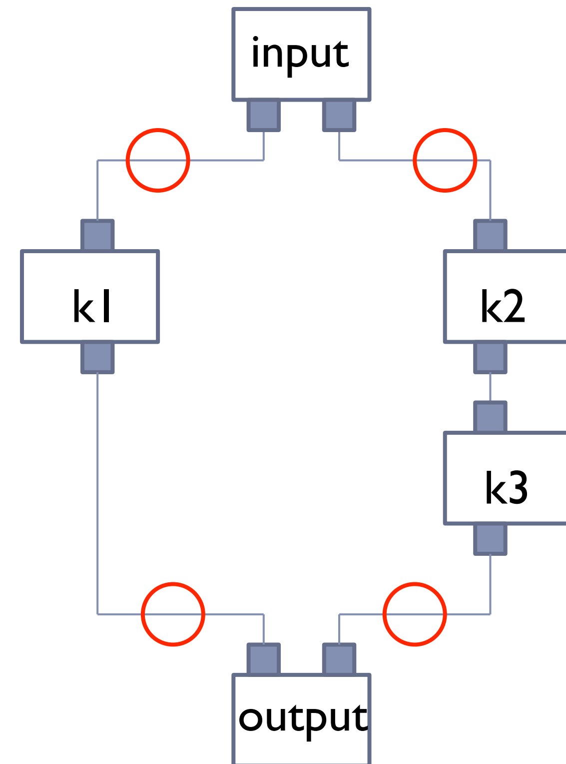
• Support Multiple Buffers

```
__kernel void kernel (
 struct p_kernel param ,
 __global  T* input1 ,
 __global  T* input2,
 __global  T* output1
)
{
  /* read from input buffers */
  /* some data parallel operation */
  /* write results to an output buffer */
}
```

Accelerator Function Example (OpenCL C)

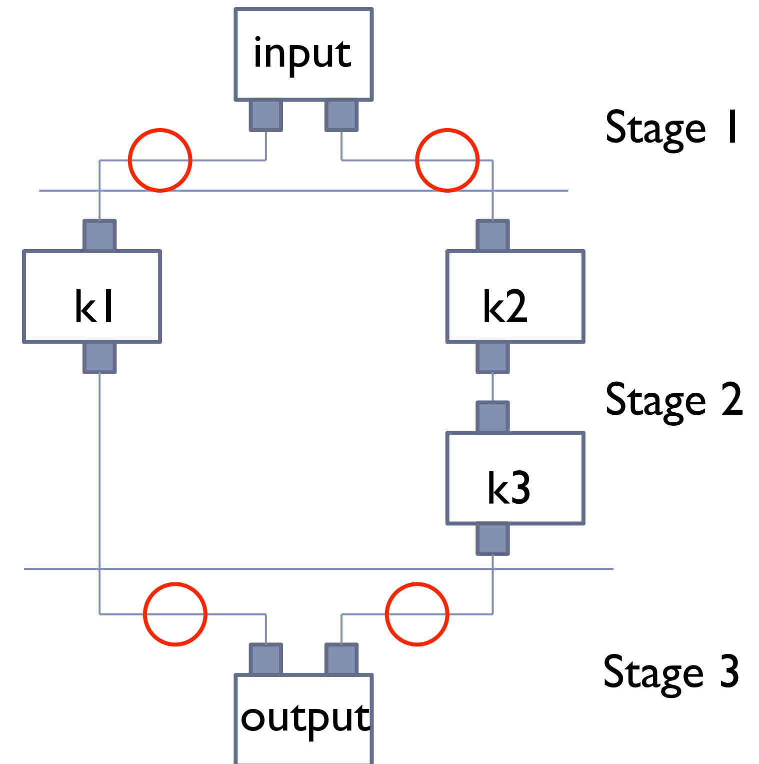
Library Internal

1. Detect communication between CPU & accelerators



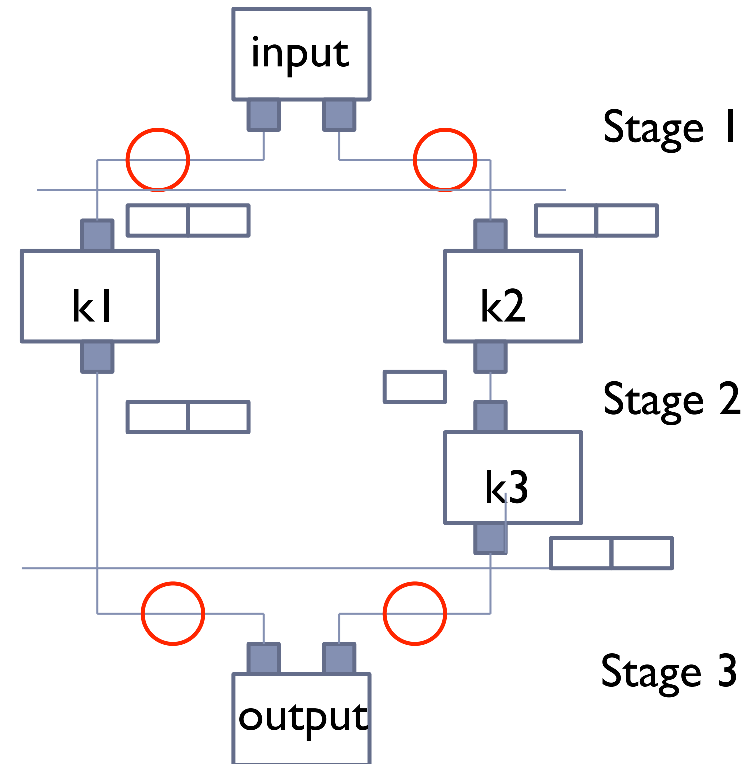
Library Internal

1. Detect communication between CPU & accelerators
2. Determine pipeline stage for tasks and communications
   - DFS based algorithm



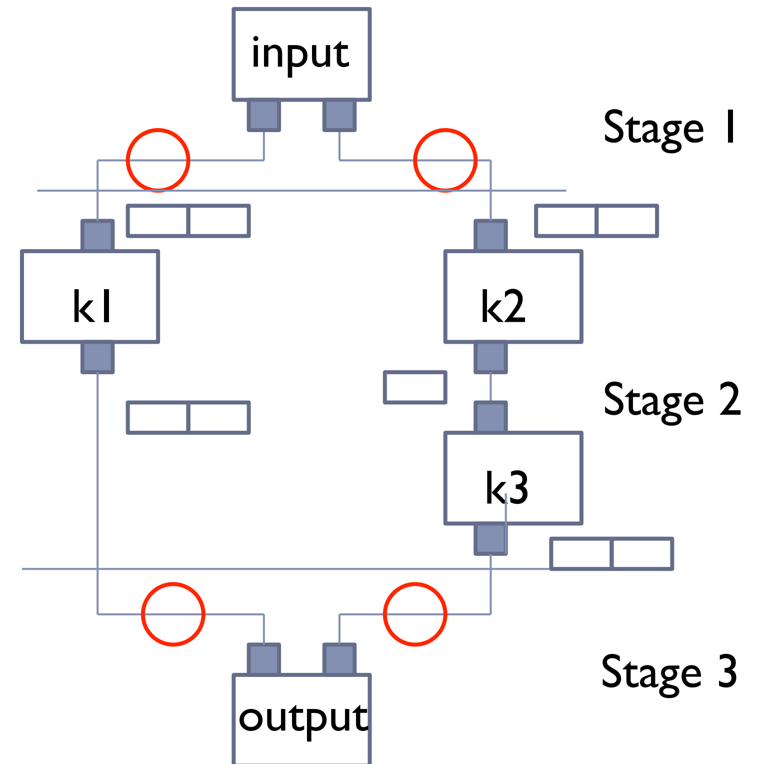
Library Internal

1. Detect communication between CPU & accelerators
2. Determine pipeline stage for tasks and communications
   - DFS based algorithm
3. Prepare device memory
   - some buffers are duplicated for overlap computation & communication



Library Internal

1. Detect communication between CPU & accelerators
2. Determine pipeline stage for tasks and communications
   - DFS based algorithm
3. Prepare device memory
   - some buffers are duplicated for overlap computation & communication
4. Execute tasks and communication in parallel
   - make use of pipeline parallelism



Evaluation Methodology

- Prototype System On Top of OpenCL Platform
- Benchmarks
  - 4 Image Processing Applications
    (from NVIDIA OpenCL Sample Applications)
- Compare Three Versions
  - original : without our library. not use double buffering.
  - handopt : without our library. use double buffering.
  - lib : with our library.

OpenCL Host	Intel(R) Core(TM) i7 CPU X 990
OpenCL Device	Nvidia GeForce GTX 570
CL Driver Version	280.13
OpenCL Platform	NVIDIA CUDA (OpenCL 1.1 CUDA 4.0.1)
Operation System	Linux 2.6

Evaluation : Code Size

- Count Only the Number of Communication Codes
  - exclude comment, white line, and initialization & finalization codes,

	Box	Median	Recursive Gaussian	Sobel
original	41	32	64	33
handopt	54	47	78	47
lib	37	24	53	25

The Number of Code Lines (lines)

Evaluation : Code Size

- Count Only the Number of Communication Codes
  - exclude comment, white line, and initialization & finalization codes,

	Box	Median	Recursive Gaussian	Sobel
original	41	32	64	33
handopt	54	47	78	47
lib	37	24	53	25

↑ due to manual double buffering

The Number of Code Lines (lines)

Evaluation : Code Size

- Count Only the Number of Communication Codes
  - exclude comment, white line, and initialization & finalization codes,

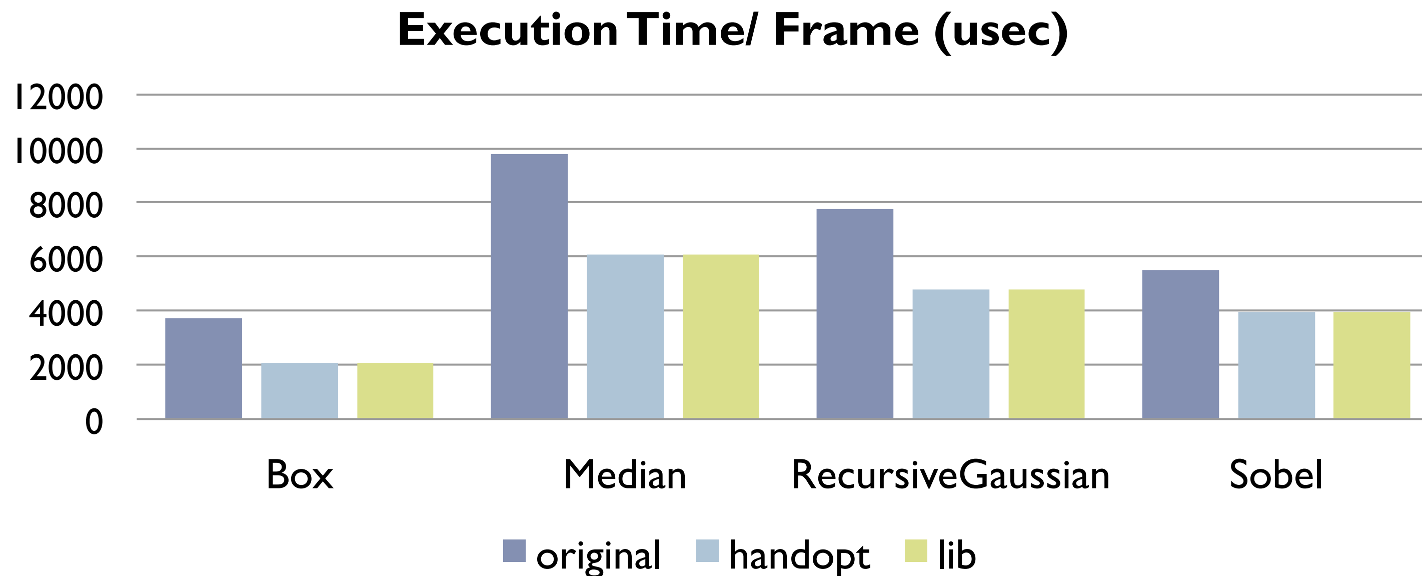
	Box	Median	Recursive Gaussian	Sobel
original	41	32	64	33
handopt	54	47	78	47
lib	37	24	53	25

↓ due to no device memory

The Number of Code Lines (lines)

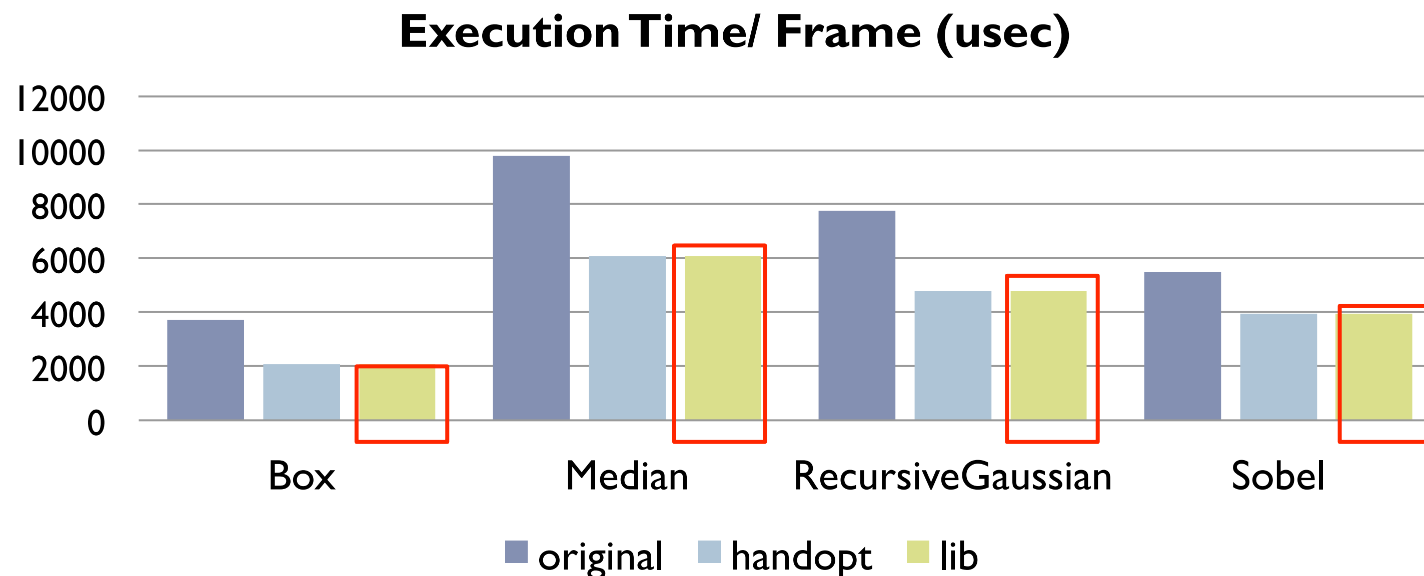
Evaluation : Performance

- Execution Time / frame
  - process 100 frames.
  - In "lib" and "handopt", computation & communication are overlapped.



Evaluation : Performance

- Execution Time / frame
  - process 100 frames.
  - In "lib" and "handopt", computation & communication are overlapped.



System Overhead is Negligible

Related Work

- StreamIT
  - W. Thies, M. Karczmarek, and S. Amarasinghe, "Streamit: A language for streaming applications," in *Proc. International Conference on Compiler Construction(CC'02), Grenoble,* France, Apr 2002.
- StarPU
  - C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. In H. Sips, D. Epema, and H.-X. Lin, editors, *Euro-Par 2009 Parallel Processing, volume 5704* of *Lecture Notes In Computer Science, pages 863–874. Springer Berlin / Heidelberg,* 2009.
- CGCM
  - T. B. Jablin, P. Prabhu, J. A. Jablin, N. P. Johnson, S. R.Beard, and D. I. August, "Automatic CPU-GPU communication management and optimization," in *Proc. the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation(PLDI'11), New York, NY, USA, 2011, pp.*142–151.

Summary & Future Work

- Communication Library for Accelerator Platforms
  - Goals : to hide
    - device memory management
    - large communication latency
  - Stream Graph Abstraction with Static Mapping & Scheduling
    - provide suitable abstraction of communication
    - allow simple mechanism to achieve high-performance
- Future Work
  - Evaluate more realistic applications
    - video-processing system
  - Integrate with stream graph analysis tools
    - build fully automatic communication management system.

System Overhead : initialize()

- Library Initialization Time (usec)
  - Graph Analysis Time
    + Device Memory Allocation Time (exist in direct OpenCL implementations)

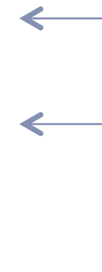
	box	median	recursive gaussian	sobel
Graph Analysis	49	44	69	43
Device Memory Allocation	9421	13411	26437	13427

Needed even when without our library.

System Overhead : execute()

- Execution Time / frame
  - Our Library System Overhead
    + OpenCL System Overhead (exist in direct OpenCL implementations)
  - OpenCL System Overhead is sometimes unpredictable !

	box	median	recursive gaussian	sobel
original	338	320	335	358
handopt	301	358	389	366
lib	304	359	394	358



No Library Overhead

OpenCL Application Example

```
float h_mem1[100];
float h_mem2[100];
cl_mem d_mem1, d_mem2;
initialize(h_mem1);

// Allocate Device Memory
d_mem1 = clCreateBuffer(...);
d_mem2 = clCreateBuffer(...);

// Blocking Host -> Device Transfer

clEnqueueWriteBuffer
    (..,d_mem1,..,h_mem1,..);
```

```
// set input and output device memory.
 clSetKernelArg(kernel,..,d_mem1,..);
 clSetKernelArg(kernel,..,d_mem2,..);

 // Execute on Device
clEnqueueNDRangeKernel(..,kernel,...);
clFinish(cmdQueue);

// Blocking Device -> Host Transfer
clEnqueueReadBuffer
    (..,d_mem2,..,h_mem2,...);

for(i = 0; i< num;i++){
  printf("%d\n" ,h_mem2);
}
```

Describe a Stream Graph : API

- addKernel()
  - add nodes in a Stream Graph
  - arguments
    - task specific parameters, parameters for data-parallel executions.

- addInput(Output)Stream
  - add Start or End nodes in a Stream Graph
  - arguments
    - total data size, token data size
    - Currently, programmers have to manually tune token size.

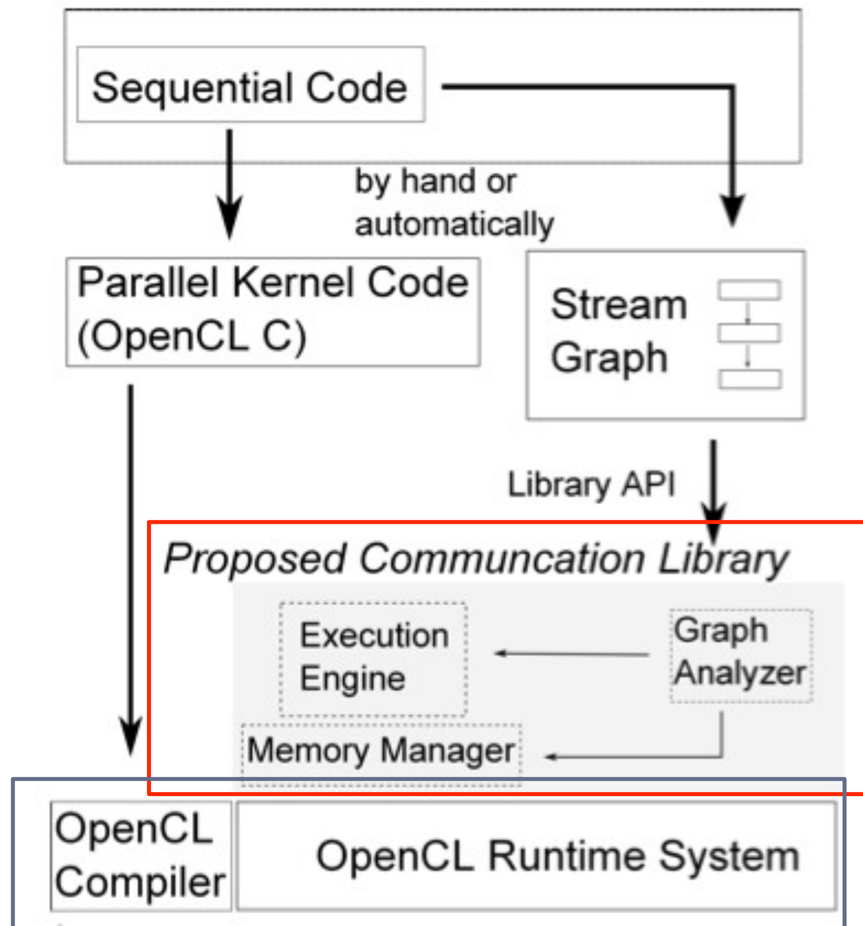
- connect()
  - add an edge between 2 nodes(tasks) in a Stream Graph
    - When a task has multiple input/output buffers, we must explicitly specify the position.
  - arguments
    - task identifier(from), task identifier(to) , token size

Communication Library for OpenCL Application

- Design Goal
  - Hide !
    - Device Memory Management (High Productivity)
    - Communication Latency (High Performance)
- Constraint
  - Can be implemented on top of the OpenCL Platform to keep portability.
- Stream Graph is Revisited
  - provide simple abstraction
  - allow simple internal mechanism to overlap computation and communication

Development Steps with Our Library

System Overview



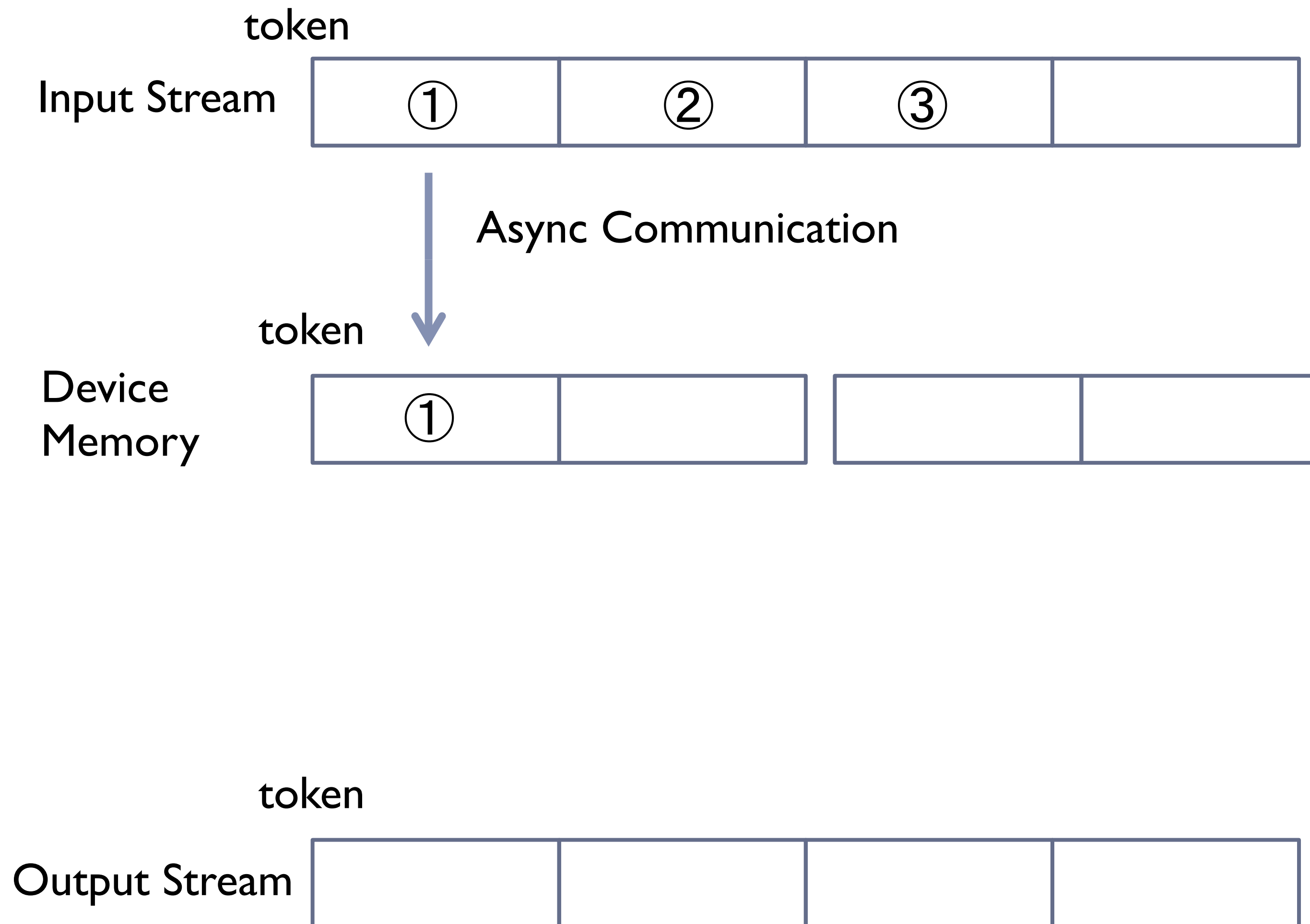
Required Procedure

1. Write Data-Parallel Code
2. Describe a Stream Graph
   - Declare nodes and define communication
   - No device memory & no manual optimization

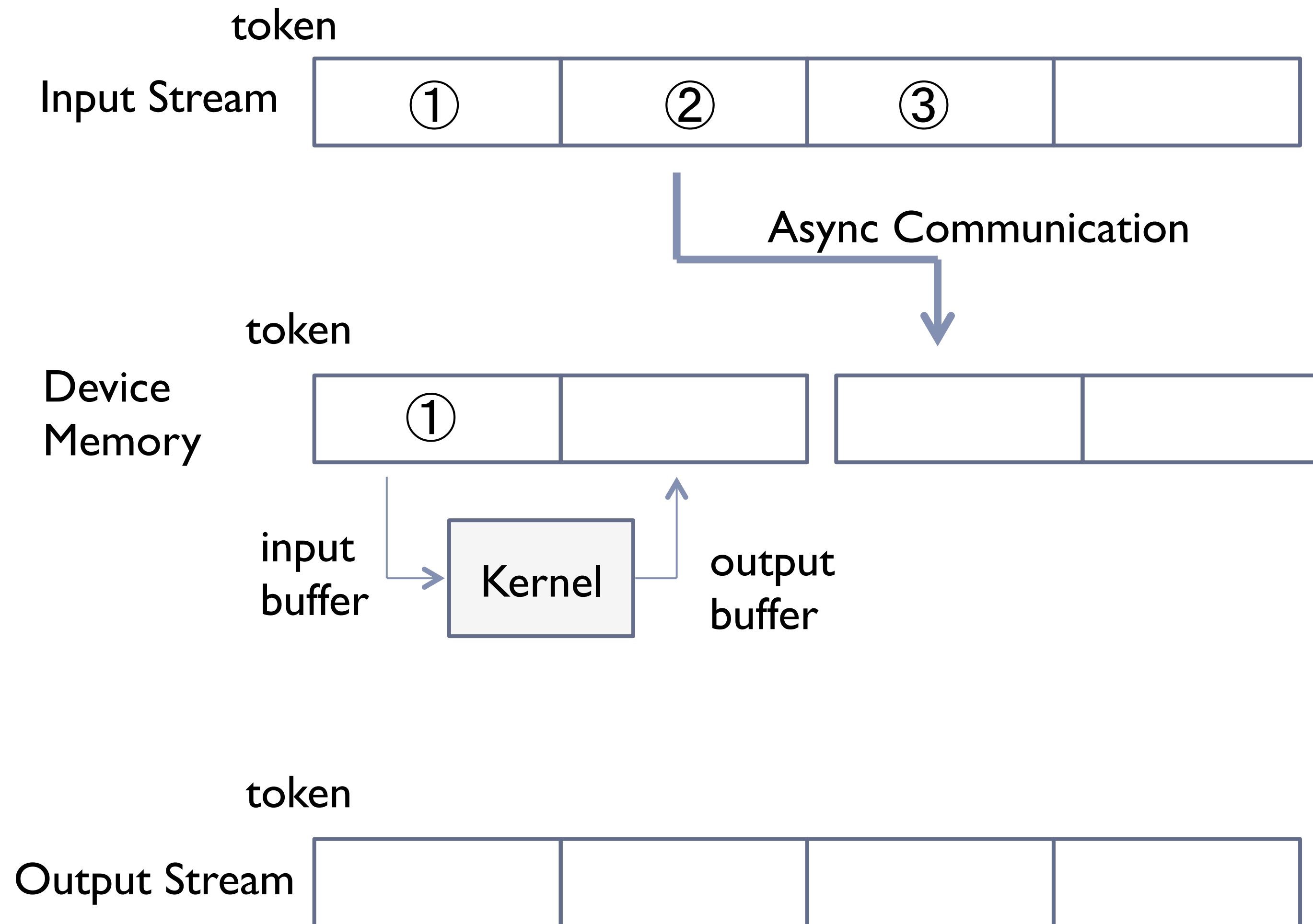
Our Proposal

Hardware Vendor Provide

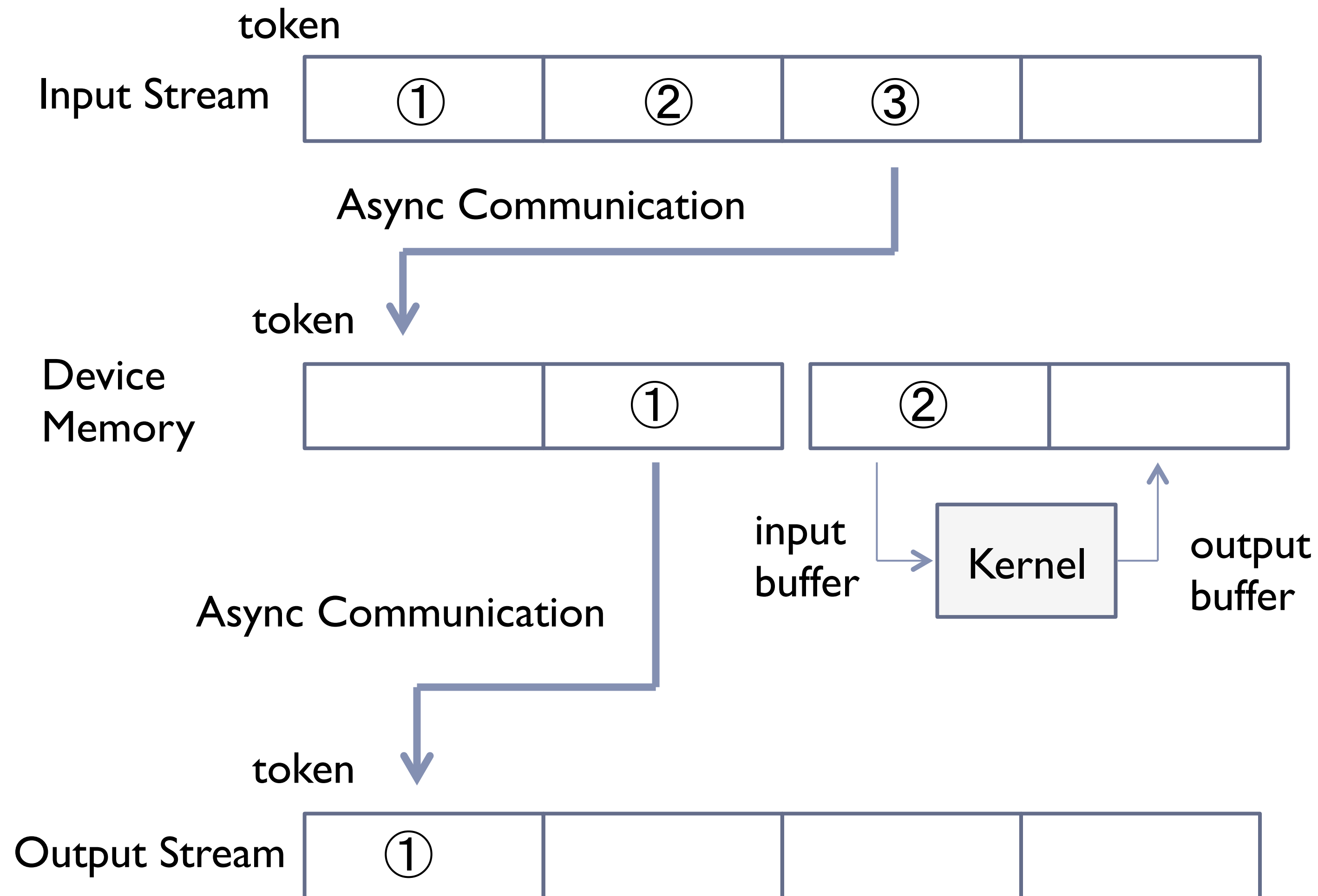
Library Internal : Execution



Library Internal : Execution



Library Internal : Execution



What is the best way to make good communication system？

- Previous Work
  - Use Compiler Technology
    - suffers from ambiguous pointers
  - Use Shared Virtual Memory Technology
    - can automate some part of device memory management.
    - but cannot automate optimization to hide communication latency.

What is the best way to make good communication system ?

- Previous Work
  - Use Compiler Technology
    - suffers from ambiguous pointers
  - Use Shared Virtual Memory Technology
    - can automate some part of device memory management.
    - but cannot automate optimization to hide communication latency.
- Communication Library ← Proposed In This Talk
  - can provide suitable level of communication abstraction

Programming flow for accelerators

- In most cases, we port an existing codes to accelerator platform
- Required procedure for the porting
  1. write parallel kernel codes (Kernel Code)
     - develop parallel codes with data-parallel languages
       - many research exists to (semi-)automatic parallelization

  2. write communication codes (CPU Code)
     - Manually manage data movement between CPU and accelerators
       - also important both for productivity and performance.