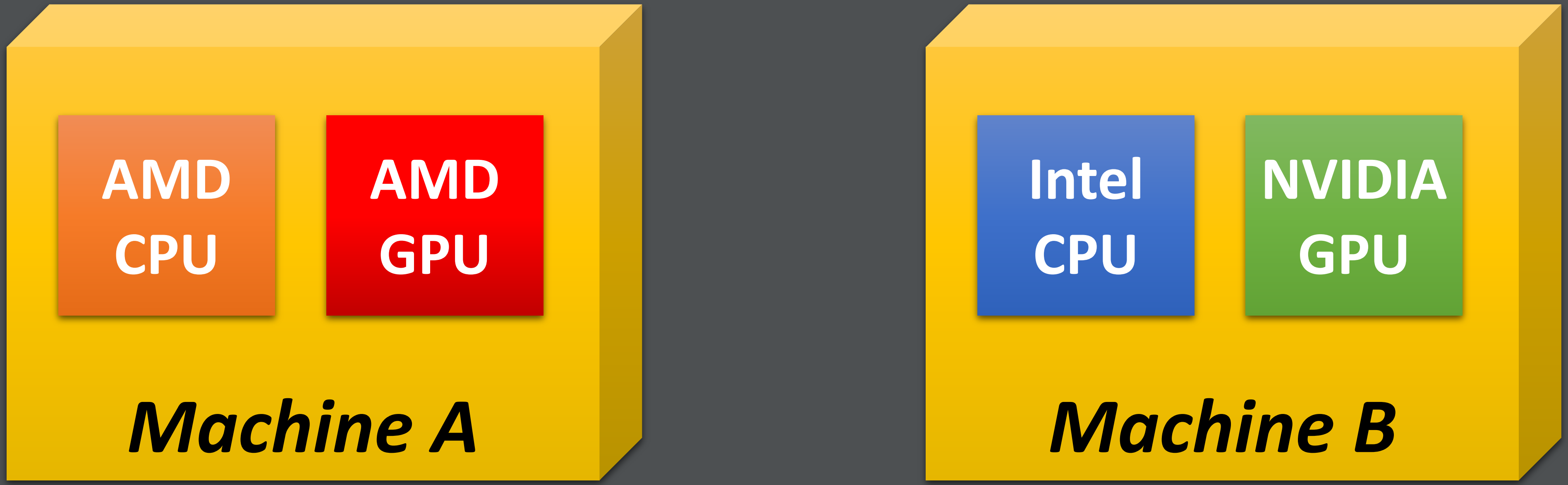
IRIS SAXPY Tutorial

Jungwon Kim

October 21, 2021

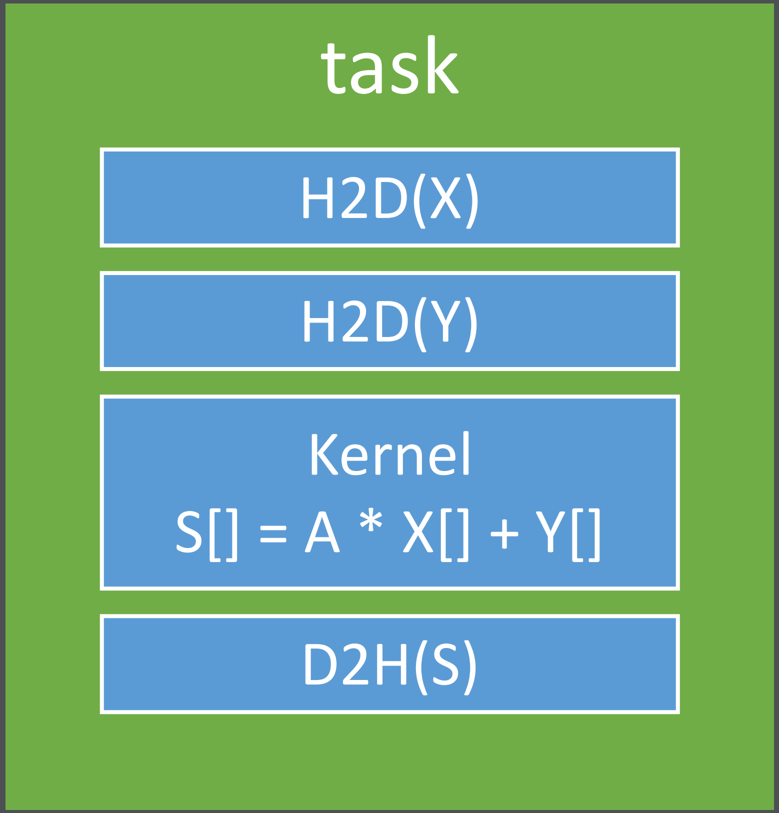
Goal: Portable SAXPY using IRIS

- S[] = A * X[] + Y[]



A Task with Four Commands

- S[] = A * X[] + Y[]



task

H2D(X)

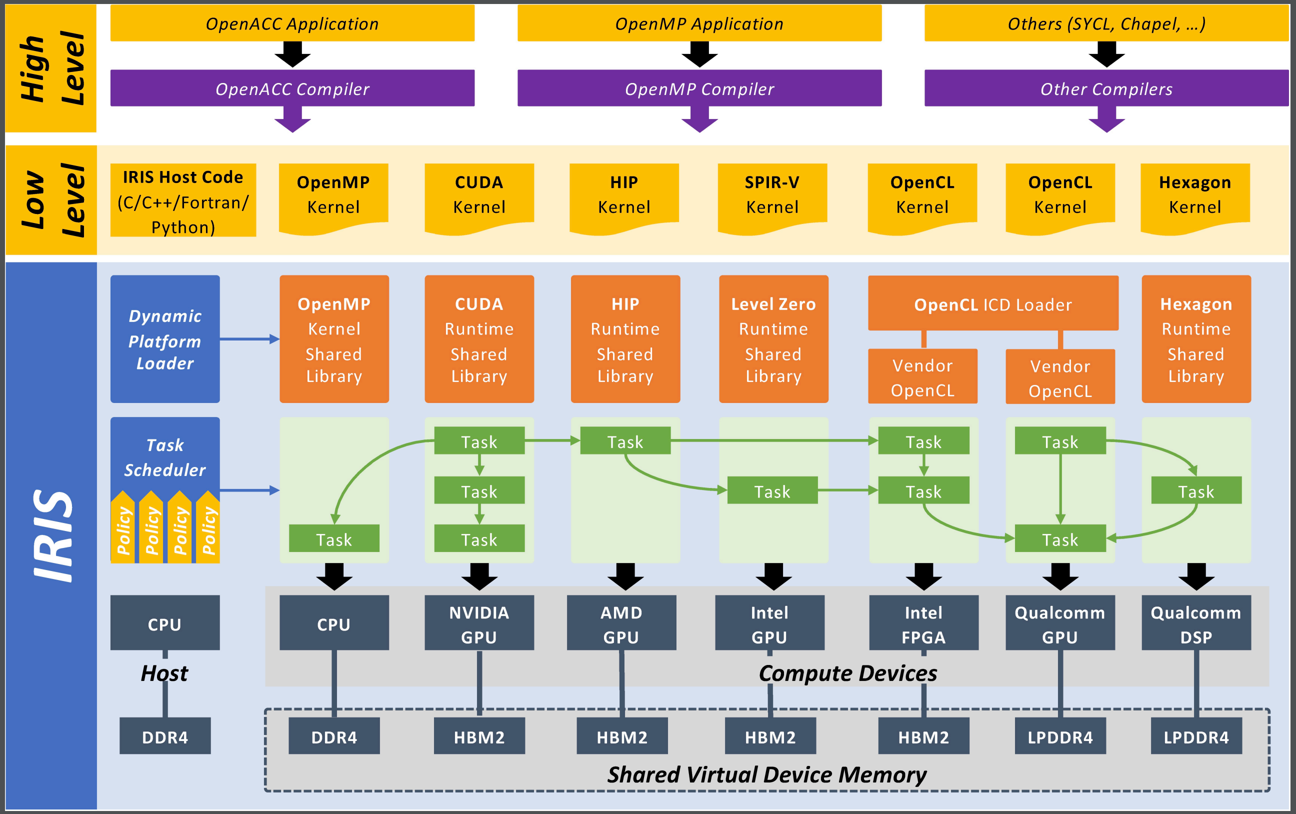
H2D(Y)

Kernel

S[] = A * X[] + Y[]

D2H(S)

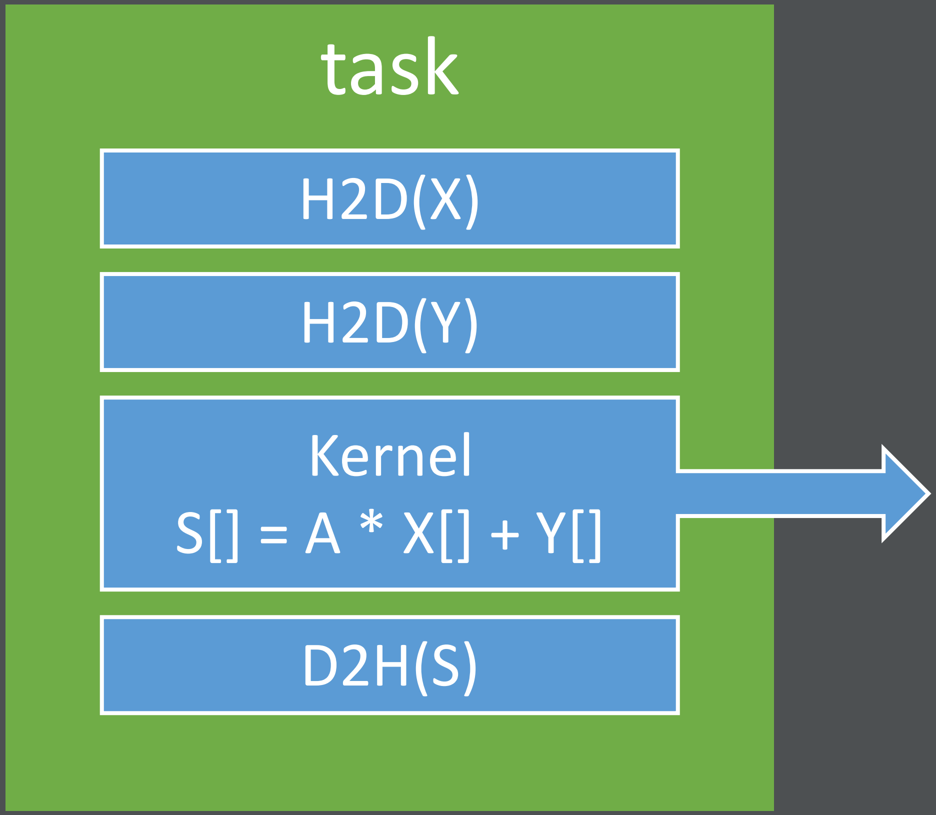
IRIS Overview



https://iris-programming.com

A Task with Four Commands

- S[] = A * X[] + Y[]

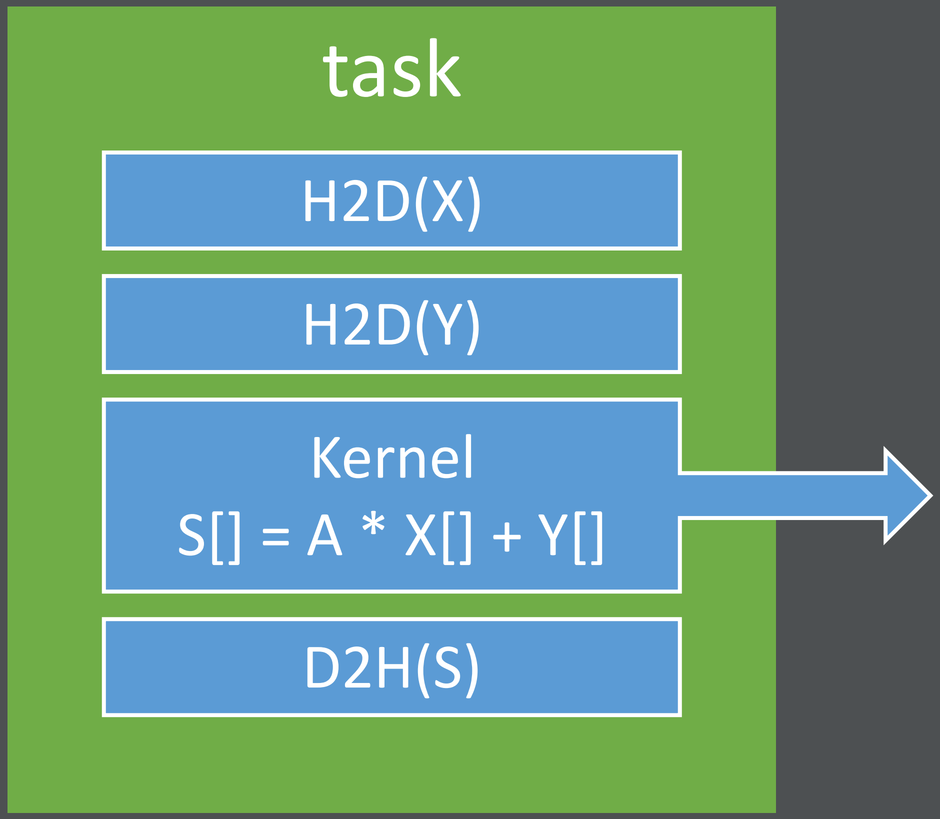


CUDA/HIP Kernel for NVIDIA/AMD GPU

```
extern "C" __global__
void saxpy(float* S, float A, float* X, float* Y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  S[i] = A * X[i] + Y[i];
}
```

A Task with Four Commands

- S[] = A * X[] + Y[]



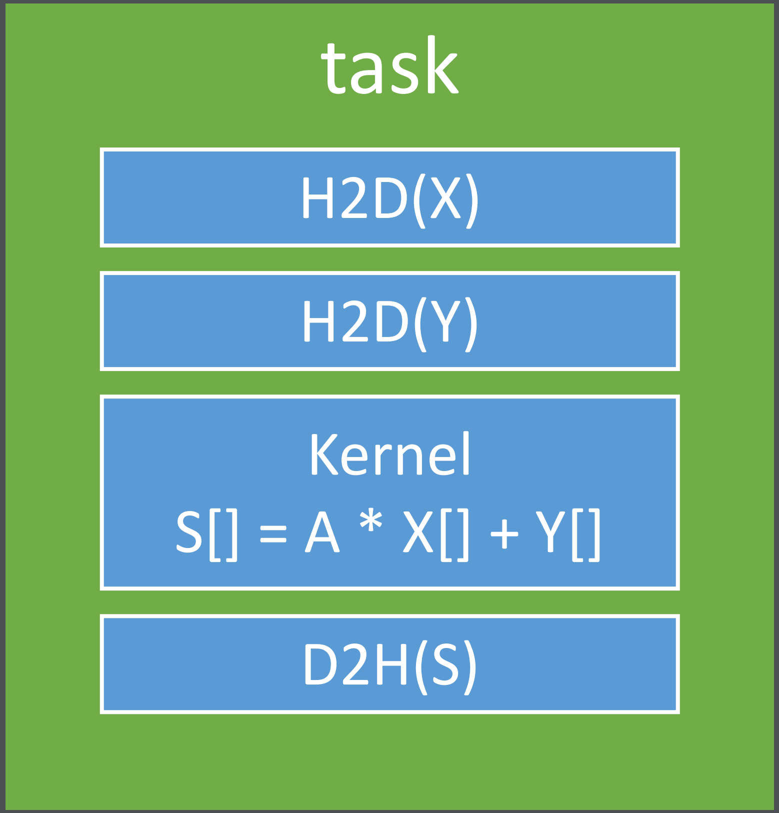
OpenMP Kernel for CPU

```
#include <iris/iris_openmp.h>

static void saxpy(float* S, float A, float* X, float*
Y, IRIS_OPENMP_KERNEL_ARGS) {
  int i;
#pragma omp parallel for shared(S, A, X, Y) private(i)
  IRIS_OPENMP_KERNEL_BEGIN(i)
  S[i] = A * X[i] + Y[i];
  IRIS_OPENMP_KERNEL_END
}
```

A Task with Four Commands

- S[] = A * X[] + Y[]



task

H2D(X)

H2D(Y)

Kernel
S[] = A * X[] + Y[]

D2H(S)

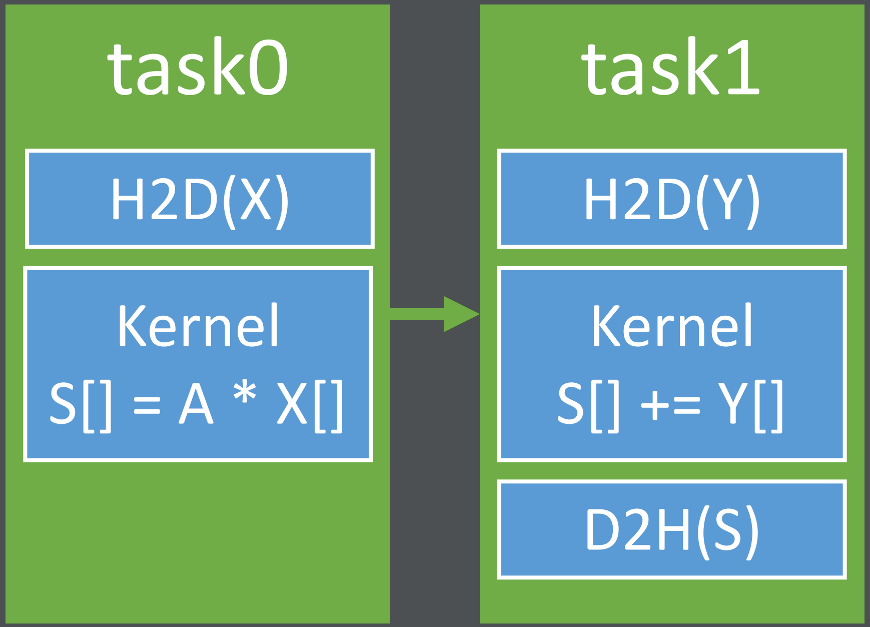
Host code

```
iris_mem mem_S, mem_X, mem_Y;
iris_mem_create(SIZE * sizeof(float), &mem_S);
iris_mem_create(SIZE * sizeof(float), &mem_X);
iris_mem_create(SIZE * sizeof(float), &mem_Y);

iris_task task;
iris_task_create(&task);
iris_task_h2d(task, mem_X, 0, SIZE * sizeof(float), X);
iris_task_h2d(task, mem_Y, 0, SIZE * sizeof(float), Y);
void* saxpy_params[4] = { mem_S, &A, mem_X, mem_Y };
int saxpy_params_info[4] = { iris_w, sizeof(A), iris_r,
iris_r };
iris_task_kernel(task, "saxpy", 1, NULL, &SIZE, NULL,
4, saxpy_params, saxpy_params_info);
iris_task_d2h(task, mem_S, 0, SIZE * sizeof(float), S);
iris_task_submit(task, iris_gpu, NULL, 1);
```

Two Tasks

- S[] = A * X[] + Y[]



task0

H2D(X)

Kernel
S[] = A * X[]

task1

H2D(Y)

Kernel
S[] += Y[]

D2H(S)

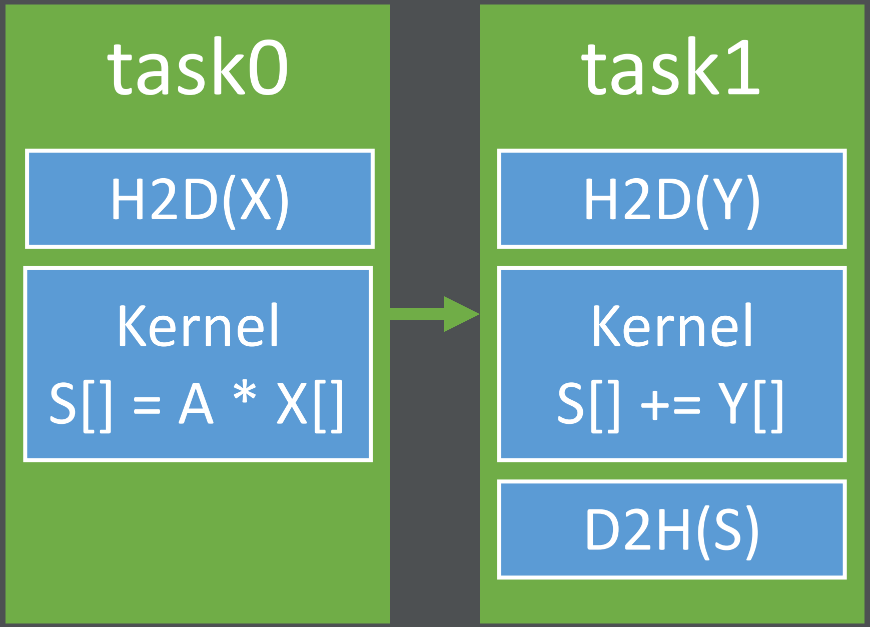
CUDA/HIP Kernel for NVIDIA/AMD GPU

```
extern "C" __global__ void sax(float* S, float A,
float* X) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  S[i] = A * X[i];
}

extern "C" __global__ void spy(float* S, float* Y) {
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  S[i] += Y[i];
}
```

Two Tasks

- S[] = A * X[] + Y[]



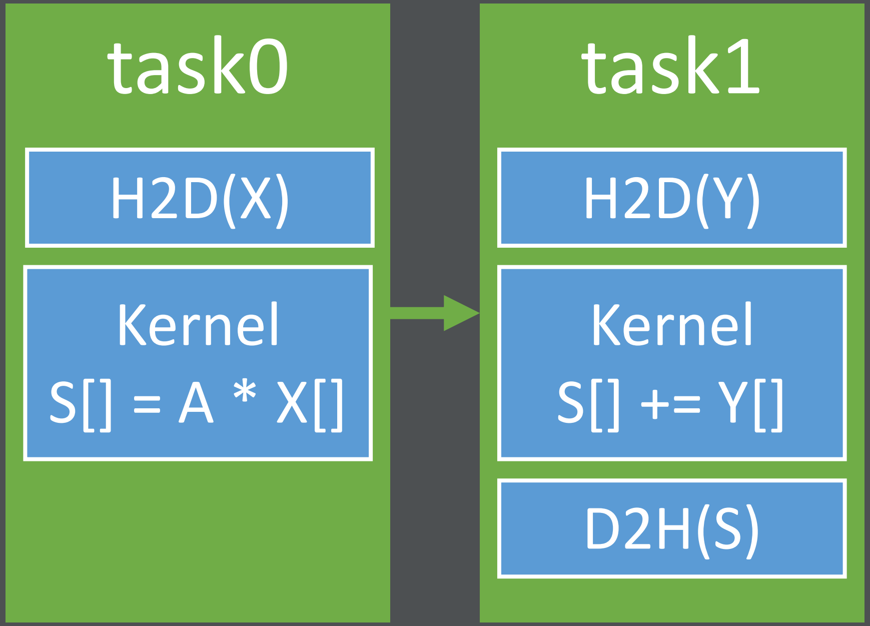
OpenMP Kernel for CPU

```
static void sax(float* S, float A, float* X,
IRIS_OPENMP_KERNEL_ARGS) {
  int i;
#pragma omp parallel for shared(S, A, X) private(i)
  IRIS_OPENMP_KERNEL_BEGIN(i)
  S[i] = A * X[i];
  IRIS_OPENMP_KERNEL_END
}

static void spy(float* S, float* Y,
IRIS_OPENMP_KERNEL_ARGS) {
  int i;
#pragma omp parallel for shared(S, Y) private(i)
  IRIS_OPENMP_KERNEL_BEGIN(i)
  S[i] += Y[i];
  IRIS_OPENMP_KERNEL_END
}
```

Two Tasks

- S[] = A * X[] + Y[]



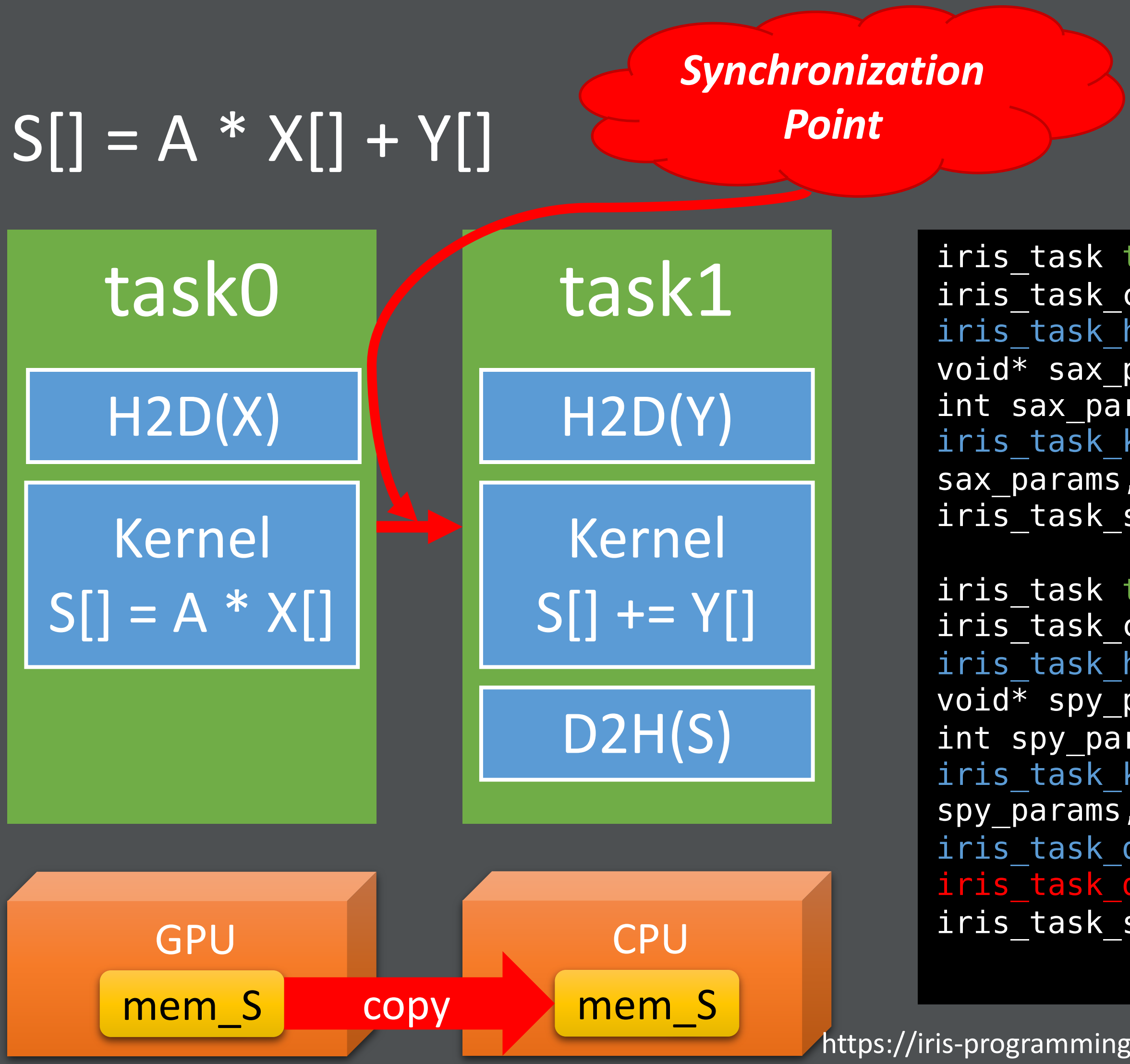
Host Code

```
iris_task task0;
iris_task_create(&task0);
iris_task_h2d(task0, mem_X, 0, SIZE * sizeof(float), X);
void* sax_params[3] = { mem_S, &A, mem_X };
int sax_params_info[3] = { iris_w, sizeof(A), iris_r };
iris_task_kernel(task0, "sax", 1, NULL, &SIZE, NULL, 3,
sax_params, sax_params_info);
iris_task_submit(task0, iris_gpu, NULL, 0);

iris_task task1;
iris_task_create(&task1);
iris_task_h2d(task1, mem_Y, 0, SIZE * sizeof(float), Y);
void* spy_params[2] = { mem_S, mem_Y };
int spy_params_info[2] = { iris_rw, iris_r };
iris_task_kernel(task1, "spy", 1, NULL, &SIZE, NULL, 2,
spy_params, spy_params_info);
iris_task_d2h(task1, mem_S, 0, SIZE * sizeof(float), S);
iris_task_depend(task1, 1, &task0);
iris_task_submit(task1, iris_cpu, NULL, 1);
```

Two Tasks: Relaxed Memory Consistency

- S[] = A * X[] + Y[]



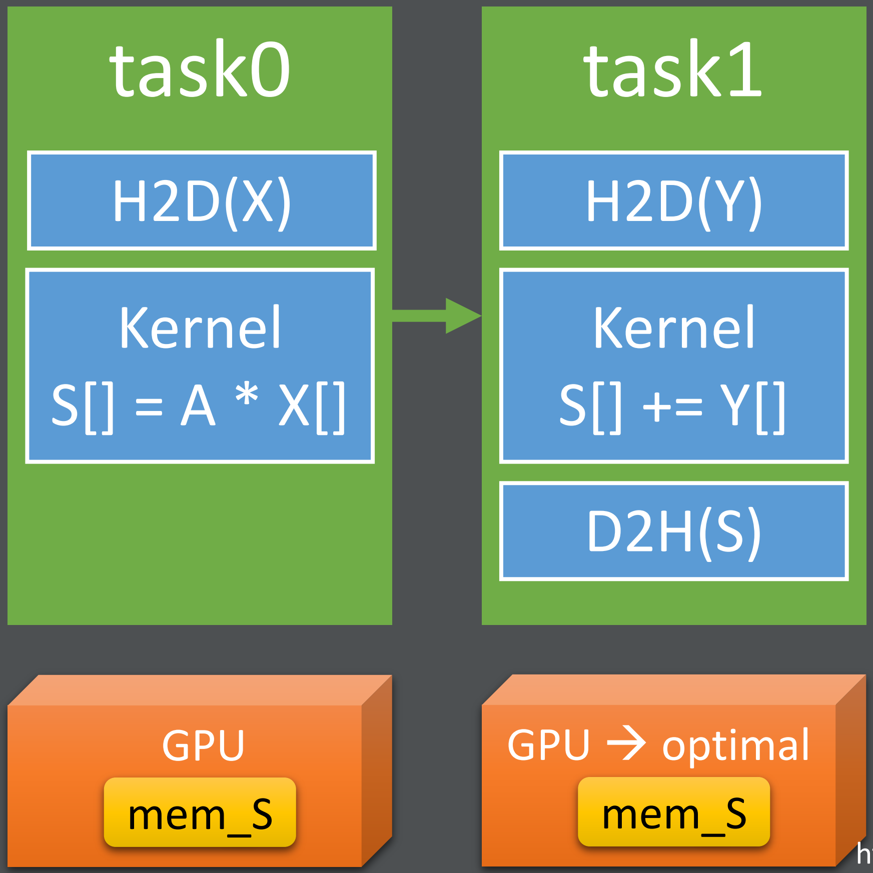
Host Code

```
iris_task task0;
iris_task_create(&task0);
iris_task_h2d(task0, mem_X, 0, SIZE * sizeof(float), X);
void* sax_params[3] = { mem_S, &A, mem_X };
int sax_params_info[3] = { iris_w, sizeof(A), iris_r };
iris_task_kernel(task0, "sax", 1, NULL, &SIZE, NULL, 3,
sax_params, sax_params_info);
iris_task_submit(task0, iris_gpu, NULL, 0);

iris_task task1;
iris_task_create(&task1);
iris_task_h2d(task1, mem_Y, 0, SIZE * sizeof(float), Y);
void* spy_params[2] = { mem_S, mem_Y };
int spy_params_info[2] = { iris_rw, iris_r };
iris_task_kernel(task1, "spy", 1, NULL, &SIZE, NULL, 2,
spy_params, spy_params_info);
iris_task_d2h(task1, mem_S, 0, SIZE * sizeof(float), S);
iris_task_depend(task1, 1, &task0);
iris_task_submit(task1, iris_cpu, NULL, 1);
```

Two Tasks: Intelligent Device Selector

- S[] = A * X[] + Y[]



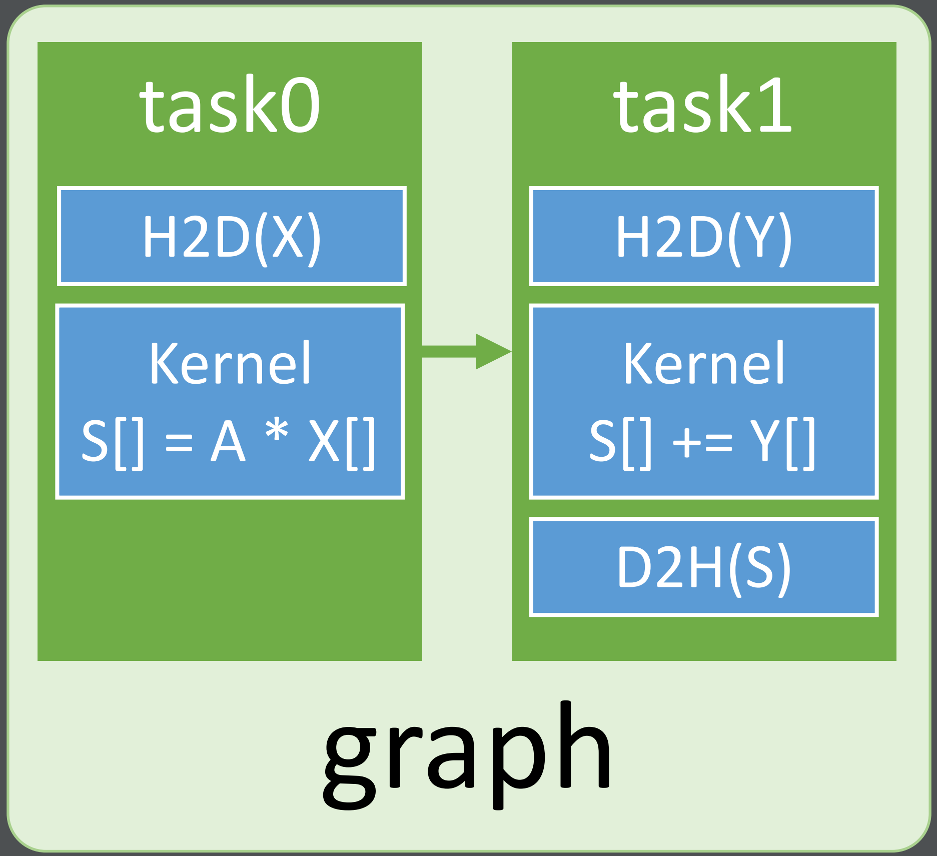
Host Code

```
iris_task task0;
iris_task_create(&task0);
iris_task_h2d(task0, mem_X, 0, SIZE * sizeof(float), X);
void* sax_params[3] = { mem_S, &A, mem_X };
int sax_params_info[3] = { iris_w, sizeof(A), iris_r };
iris_task_kernel(task0, "sax", 1, NULL, &SIZE, NULL, 3,
sax_params, sax_params_info);
iris_task_submit(task0, iris_gpu, NULL, 0);

iris_task task1;
iris_task_create(&task1);
iris_task_h2d(task1, mem_Y, 0, SIZE * sizeof(float), Y);
void* spy_params[2] = { mem_S, mem_Y };
int spy_params_info[2] = { iris_rw, iris_r };
iris_task_kernel(task1, "spy", 1, NULL, &SIZE, NULL, 2,
spy_params, spy_params_info);
iris_task_d2h(task1, mem_S, 0, SIZE * sizeof(float), S);
iris_task_depend(task1, 1, &task0);
iris_task_submit(task1, iris_locality, NULL, 1);
```

A Graph with Two Tasks

- S[] = A * X[] + Y[]



Host Code

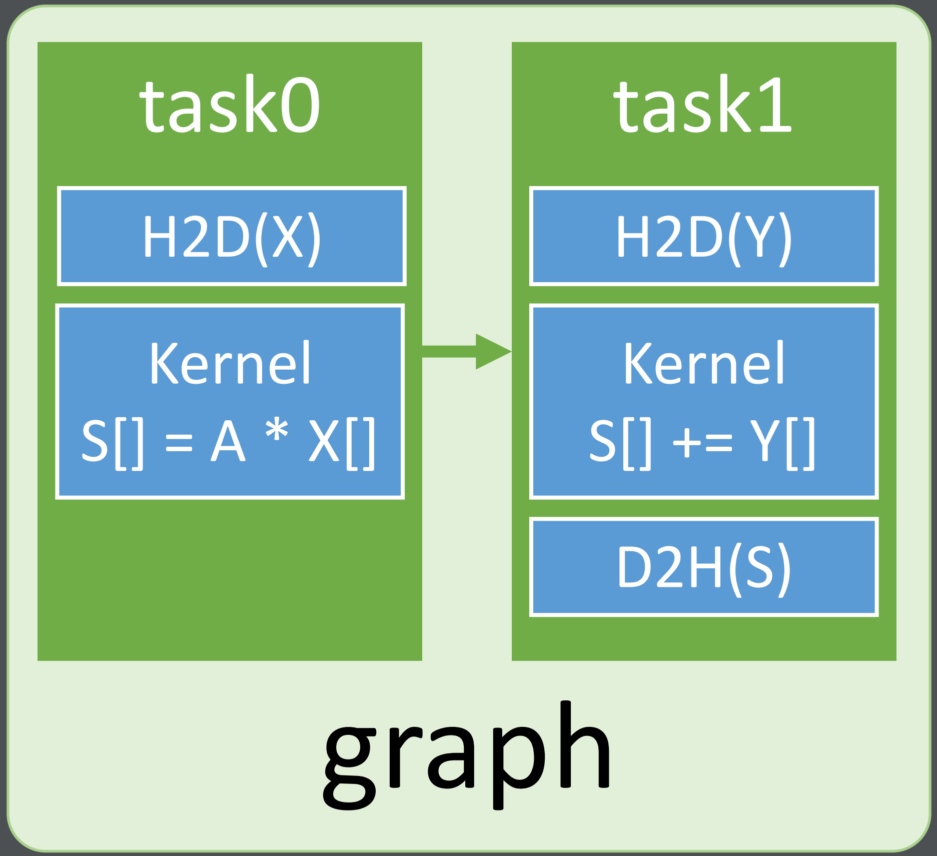
```
iris_graph graph;
iris_graph_create(&graph);

iris_task task0;
iris_task_create(&task0);
iris_task_h2d(task0, mem_X, 0, SIZE * sizeof(float), X);
void* sax_params[3] = { mem_S, &A, mem_X };
int sax_params_info[3] = { iris_w, sizeof(A), iris_r };
iris_task_kernel(task0, "sax", 1, NULL, &SIZE, NULL, 3,
sax_params, sax_params_info);
iris_graph_task(graph, task0, iris_gpu, NULL);

iris_task task1;
iris_task_create(&task1);
iris_task_h2d(task1, mem_Y, 0, SIZE * sizeof(float), Y);
void* spy_params[2] = { mem_S, mem_Y };
int spy_params_info[2] = { iris_rw, iris_r };
iris_task_kernel(task1, "spy", 1, NULL, &SIZE, NULL, 2,
spy_params, spy_params_info);
iris_task_d2h(task1, mem_S, 0, SIZE * sizeof(float), S);
iris_task_depend(task1, 1, &task0);
iris_graph_task(graph, task1, iris_locality, NULL);
iris_graph_submit(graph, iris_default, 1);
```

Building a Graph from JSON

- S[] = A * X[] + Y[]



Host Code

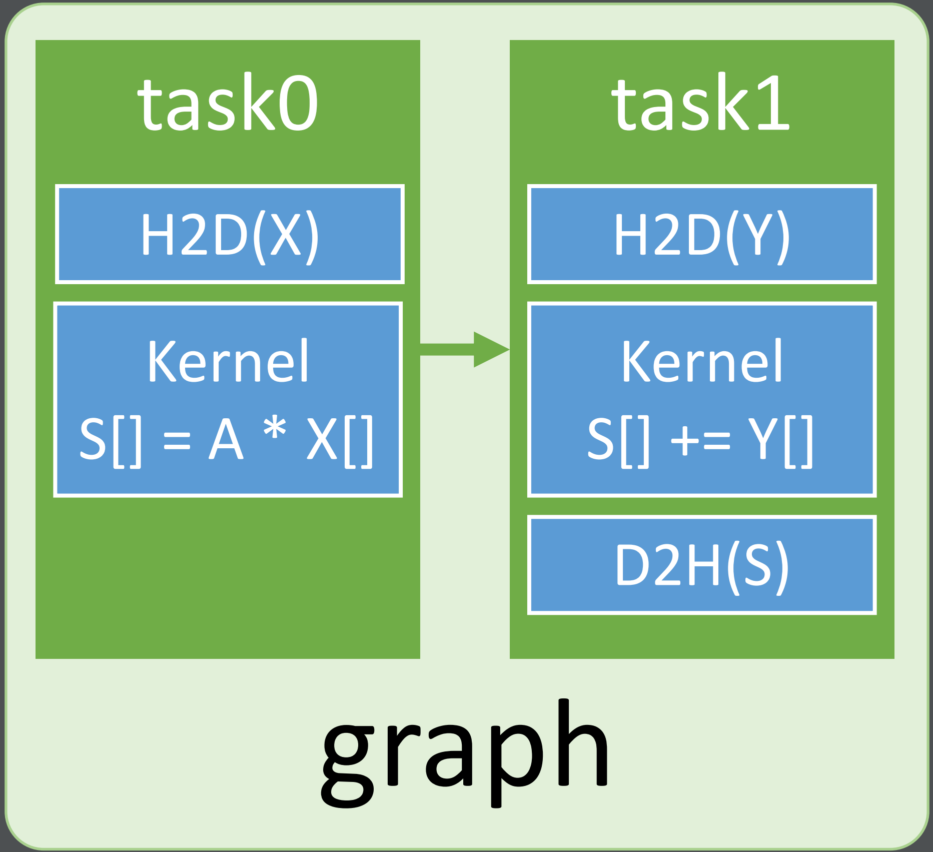
```
void* json_inputs[9] = { &SIZE, &SIZECB, S, &A, X, Y, mem_S,
mem_X, mem_Y };

iris_graph graph;
iris_graph_create_json("graph.json", json_inputs, &graph);

iris_graph_submit(graph, iris_default, 1);
```

Building a Graph from JSON

- S[] = A * X[] + Y[]


task0

H2D(X)

Kernel
S[] = A * X[]

task1

H2D(Y)

Kernel
S[] += Y[]

D2H(S)

graph

JSON

```
{
 "iris-graph": {
  "inputs": [ "user-size", "user-size-cb", "user-S", "user-A",
"user-X", "user-Y", "user-memS", "user-memX", "user-memY" ],
  "graph": {
   "tasks": [
   {
    "name" : "task0",
    "h2d": ["user-memX", "user-X", "0", "user-size-cb"],
    "kernel": ["sax", ["user-size"], ["user-memS", "user-A",
"user-memX"], ["w", "4", "r"] ],
    "target": "iris_gpu"
   },
   {
    "name" : "task1",
    "h2d": ["user-memY", "user-Y", "0", "user-size-cb"],
    "kernel": ["spy", ["user-size"], ["user-memS", "user-memY"],
["rw", "r"] ],
    "d2h": ["user-memS", "user-S", "0", "user-size-cb"],
    "depends": ["task0"],
    "target": "iris_locality"
   }]}}}
```