DESCENDANTS OF THE ABACUS

你也能駕馭GPU！
—淺談CUDA Programming

黃恩明 / Samuel Huang

SITCON'21

黃恩明 / Samuel Huang

- 就讀學校：NTHU CS Freshmen
- 競賽領域：演算法競賽、叢集競賽
- 研究內容：演算法與平行計算

SITCON'21

GPU ARCHITECTURE

SITCON'21

WHAT IS A GPU

- 圖形處理器 - Graphics Processing Unit
- Multi Thread Design
- SIMD Architecture

https://www.nvidia.com/zh-tw/data-center/a100/

Flagship: A100

- FP32: 19.5 TFLOPS
- Memory: 1.555 TB/s
- 1 FP32 = 4 Bytes
- $\frac{19.5 \cdot 4}{1.555} \simeq 50$

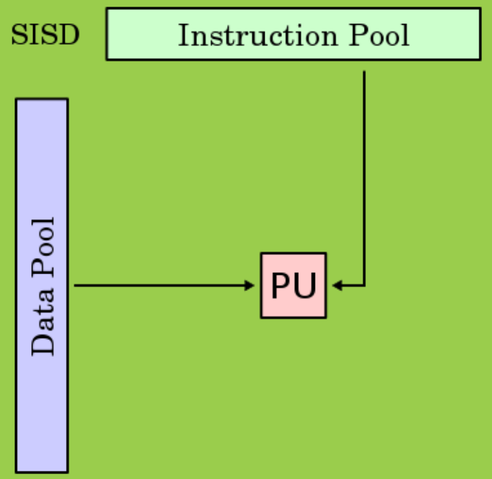
https://www.nvidia.com/zh-tw/data-center/a100/

FLYNN'S CLASSIC TAXONOMY

SISD

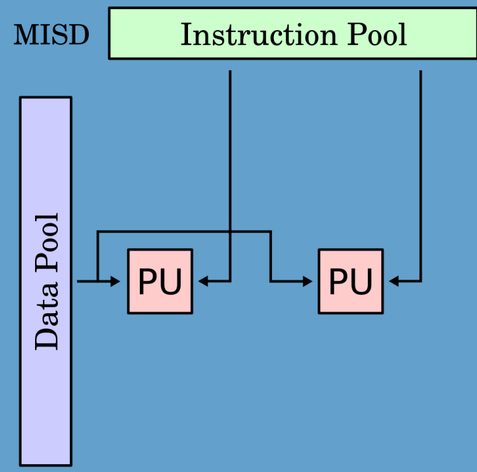
Single Instruction
Single Data

CPU



SIMD

Single Instruction
Multiple Data

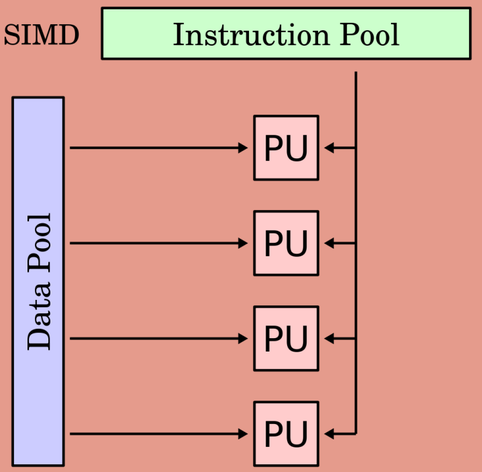


SIMD

Single Instruction
Multiple Data

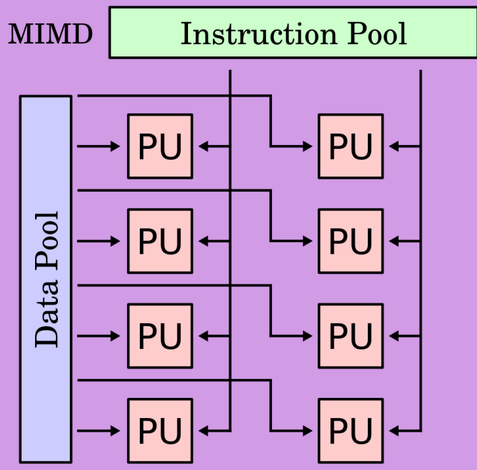
GPU

Vector Processor



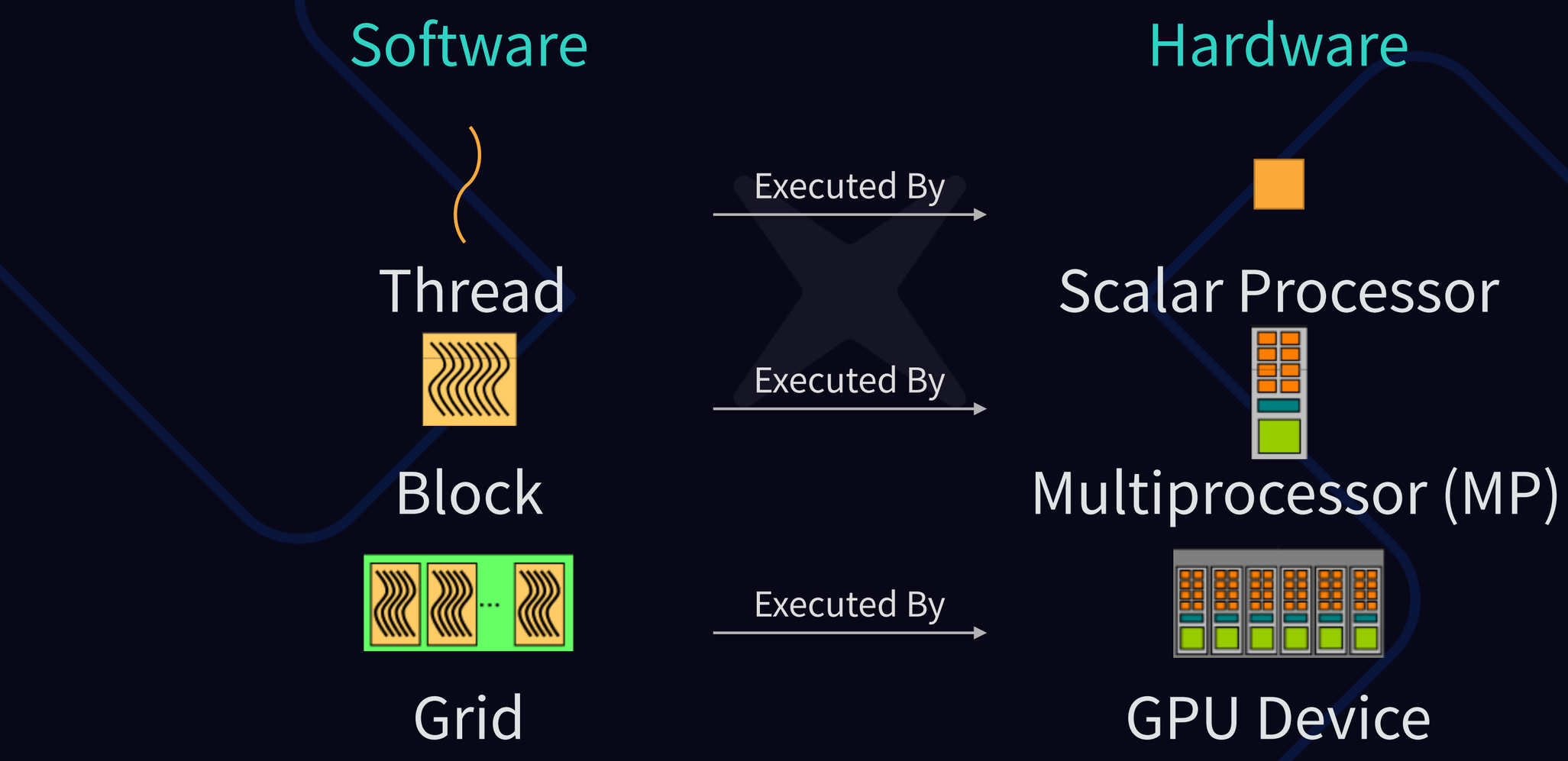
MIMD

Multiple Instruction
Multiple Data



SITCON'21

EXECUTION MODEL

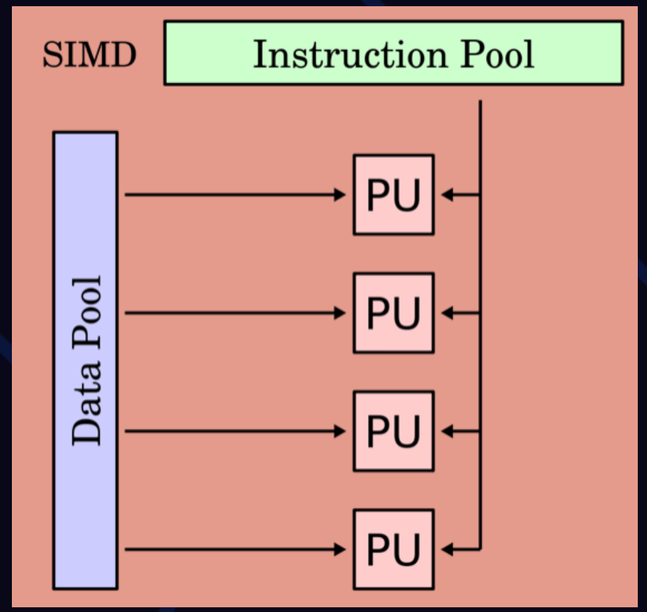


BLOCKS & THREADS

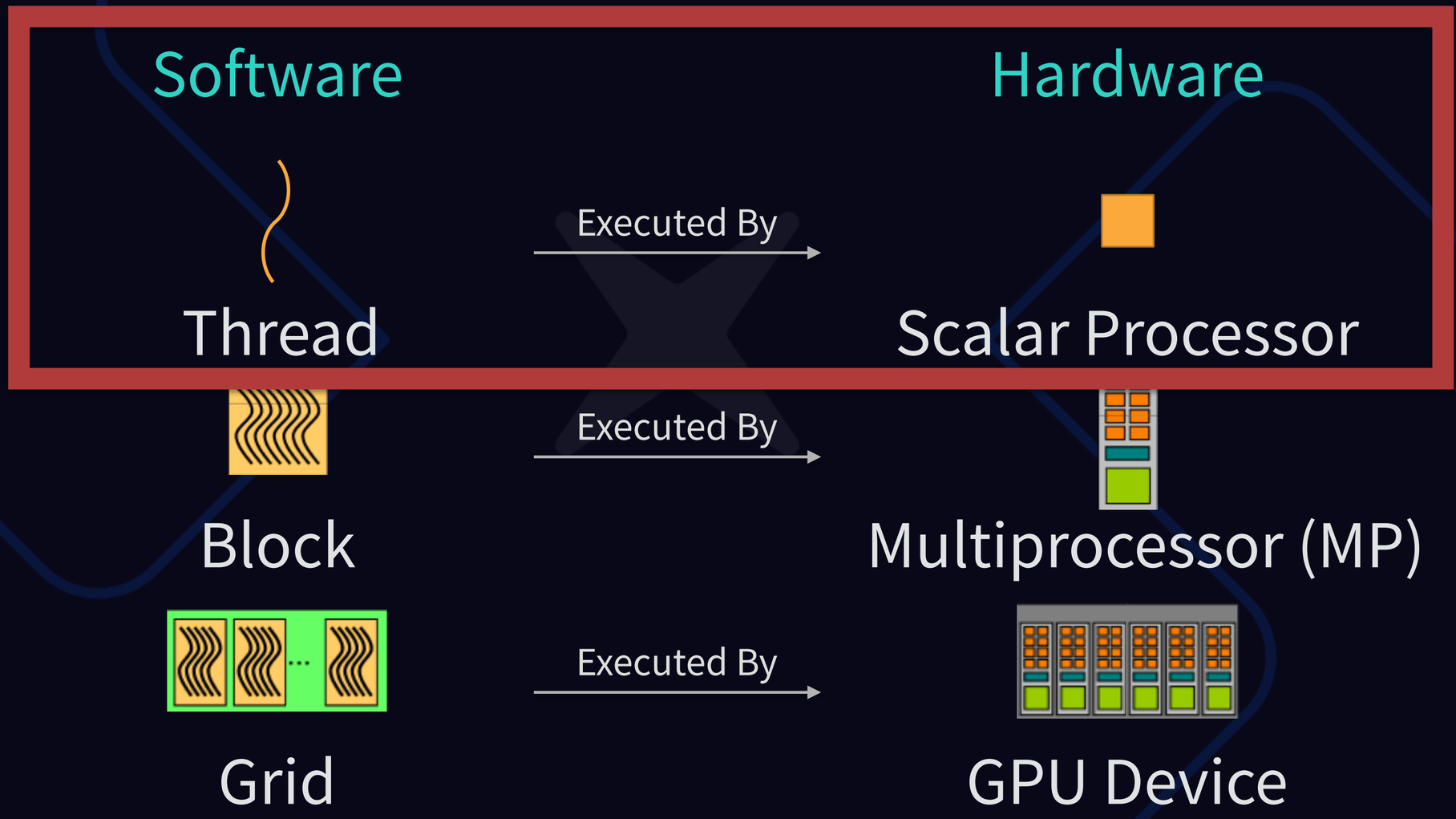
- Threads 會被包裝成 block (ex. block size = 1024)
- 相同 block下的 threads 會被執行在同一個 MP 上
  - 有 shared memory 可以共用資料與溝通
  - Threads 之間可以透過 `__syncthreads()` 彼此同步
- 不同 blocks 沒有執行先後順
  - 無法確定每個 blocks 在哪個 MP 上執行
  - 無法與其他 blocks synchronize

WARPS

- 在一個 block 下，每 32 個 threads 屬於同一個 warp
- 相同 warp 執行相同的 instruction → SIMD 架構

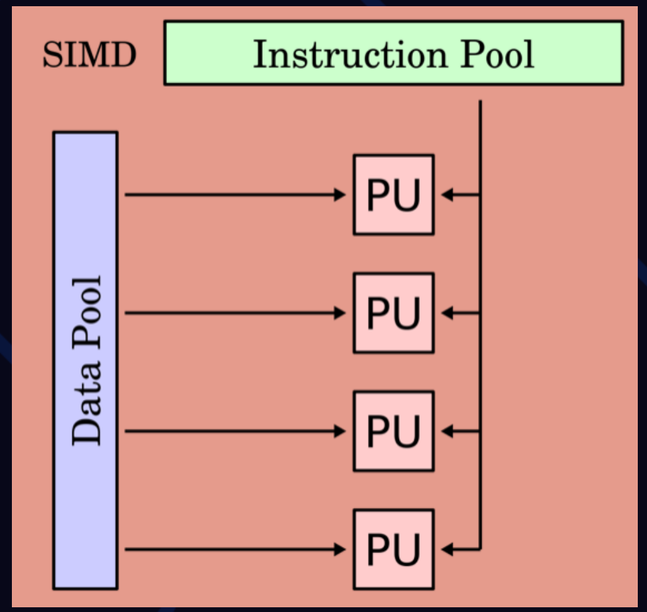


EXECUTION MODEL



WARPS

- 在一個 block 下，每 32 個 threads 屬於同一個 warp
- 相同 warp 執行相同的 instruction → SIMD 架構
- Threads in warp ：物理（時間）上的平行執行
- Blocks and grids ：邏輯上的平行執行



WARP DIVERGENCE

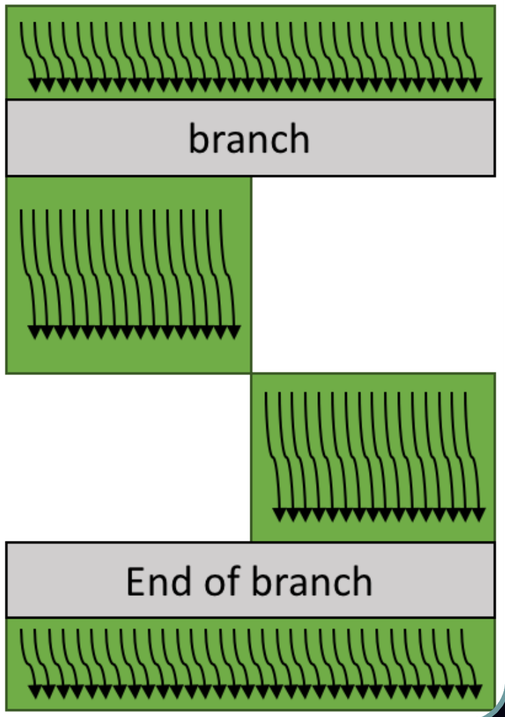
- 同個 warp 裡面的 threads

  有不同的動作

  ex. `if`, `for`, `while`

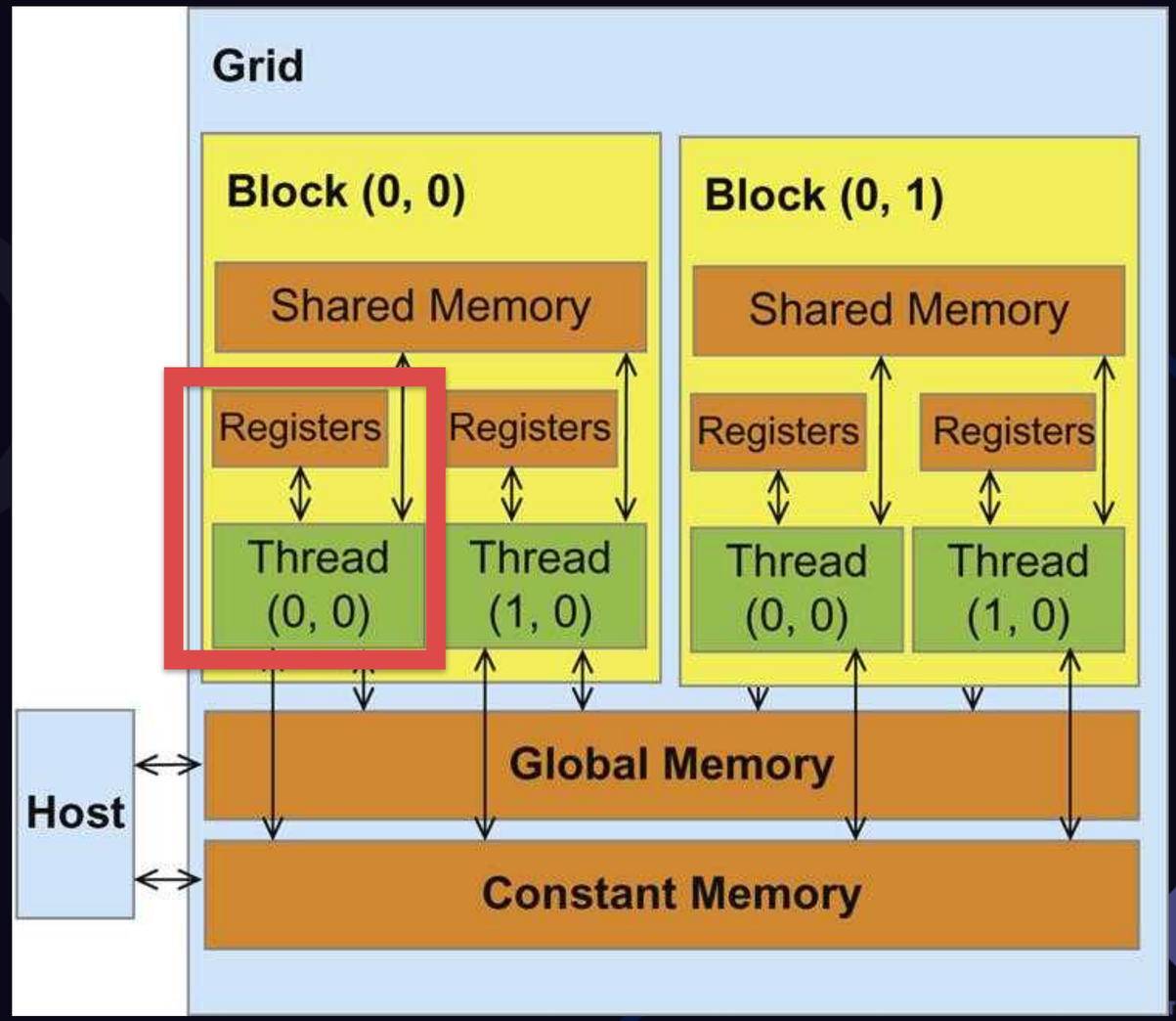
- 不同 warp 則不會出事
- Performance issue!

```
...
if ( threadIdx.x < 16 )
{
    ... A ...
}
else
{
    ... B ...
}
...
```



GPU MEMORY HIERARCHY

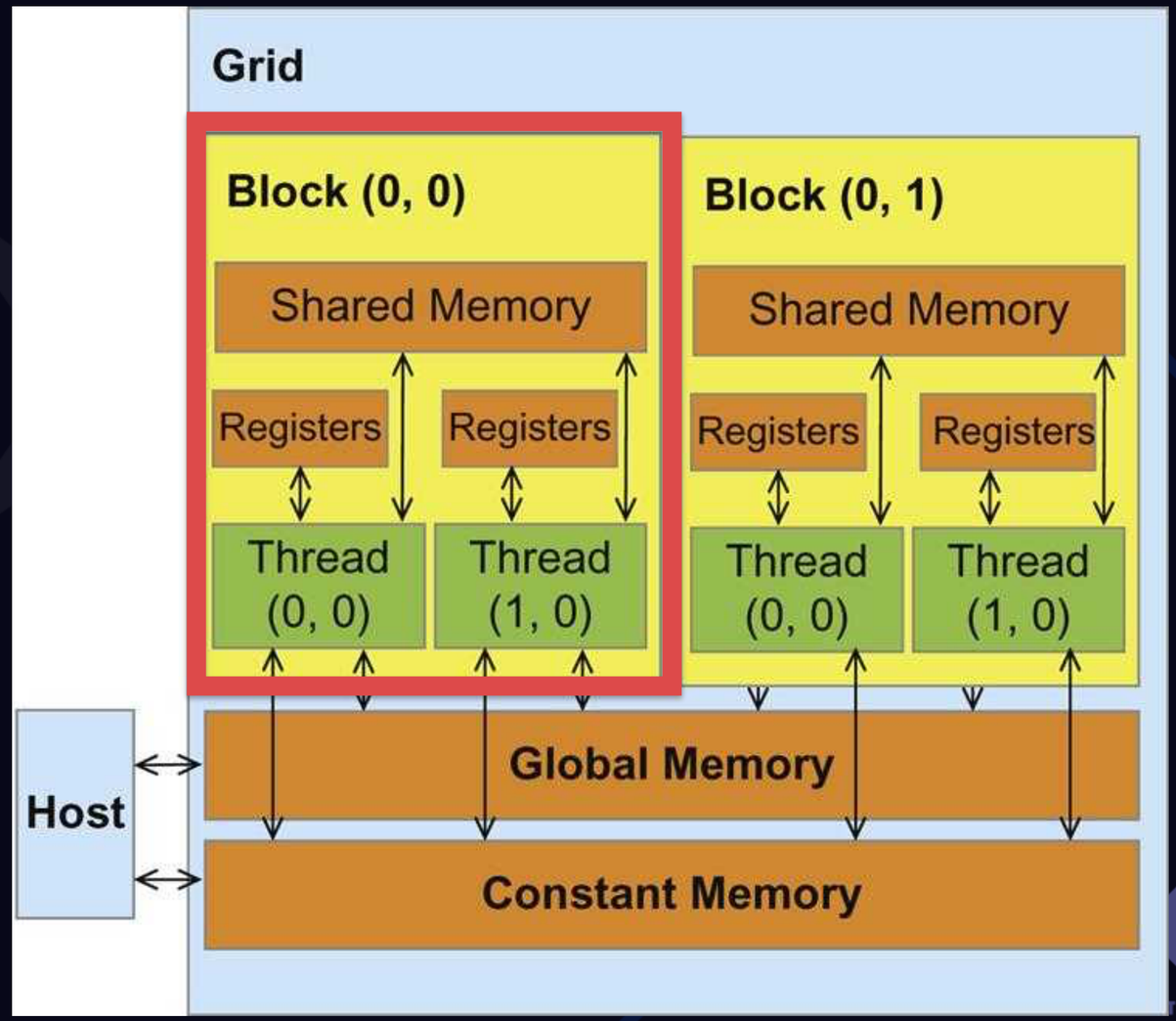
- Per threads / Scalar processor
  - Local registers



https://nichijou.co/static/bacd550fe0b1326f4e4032d2f73592d9/fb77c/memory.jpg

GPU MEMORY HIERARCHY

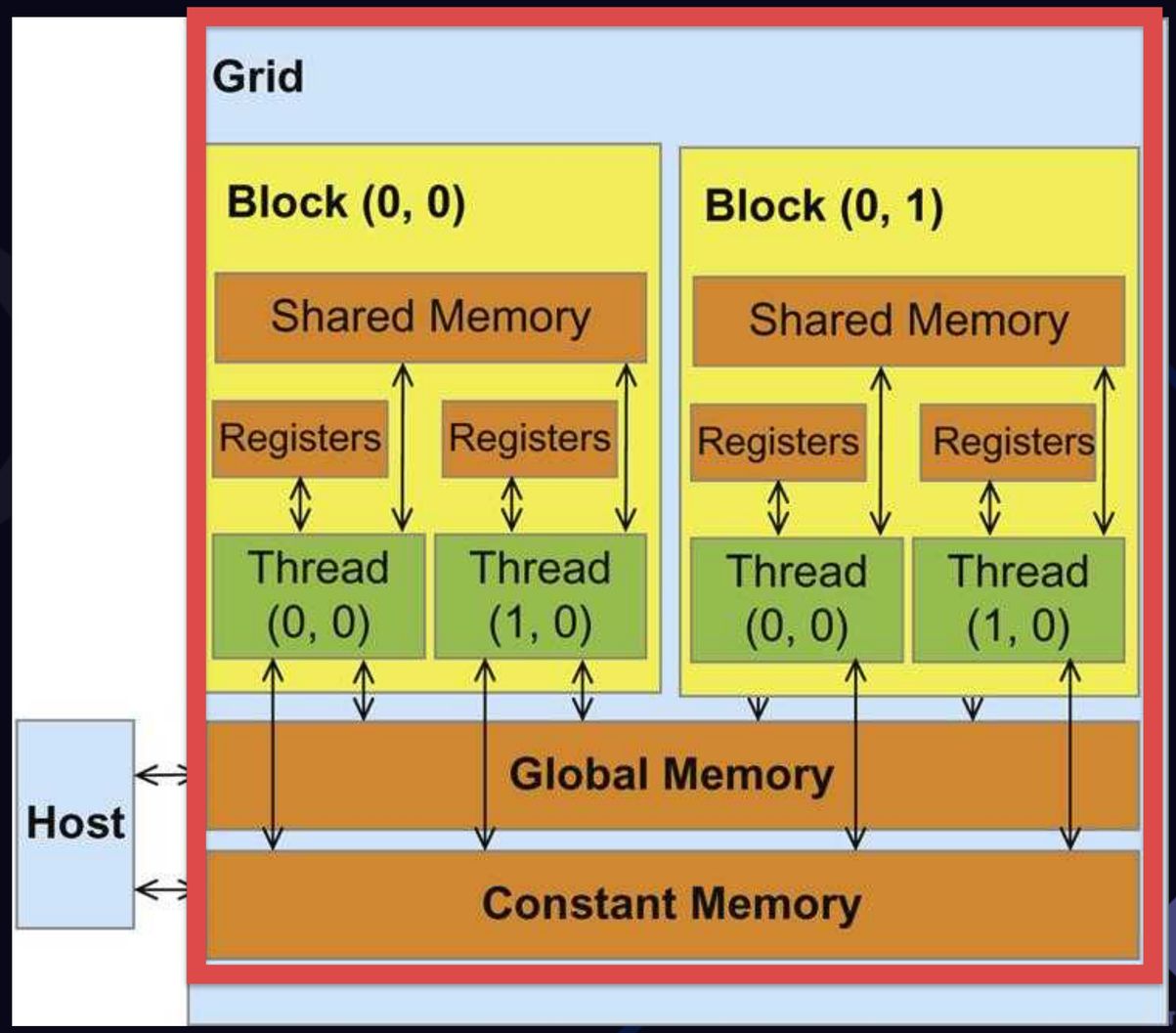
- Per blocks / MP
  - Shared memory
  - L1 cache



https://nichijou.co/static/bacd550fe0b1326f4e4032d2f73592d9/fb77c/memory.jpg

GPU MEMORY HIERARCHY

- Per grid / GPU device
  - Global memory
  - Constant / Texture memory
  - L2 cache



https://nichijou.co/static/bacd550fe0b1326f4e4032d2f73592d9/fb77c/memory.jpg

DEVICE QUERY

- https://github.com/NVIDIA/cuda-samples/tree/master/Samples/deviceQuery
- GeForce RTX 3070

```
➜  deviceQuery git:(master) ✗ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce RTX 3070"
  CUDA Driver Version / Runtime Version          11.2 / 11.2
  CUDA Capability Major/Minor version number:    8.6
  Total amount of global memory:                 7982 MBytes (8370061312 bytes)
  (46) Multiprocessors, (128) CUDA Cores/MP:     5888 CUDA Cores
  GPU Max Clock rate:                            1770 MHz (1.77 GHz)
  Memory Clock rate:                             7001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        102400 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1536
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.2, CUDA Runtime Version = 11.2, NumDevs = 1
Result = PASS
```

LOCAL REGISTERS

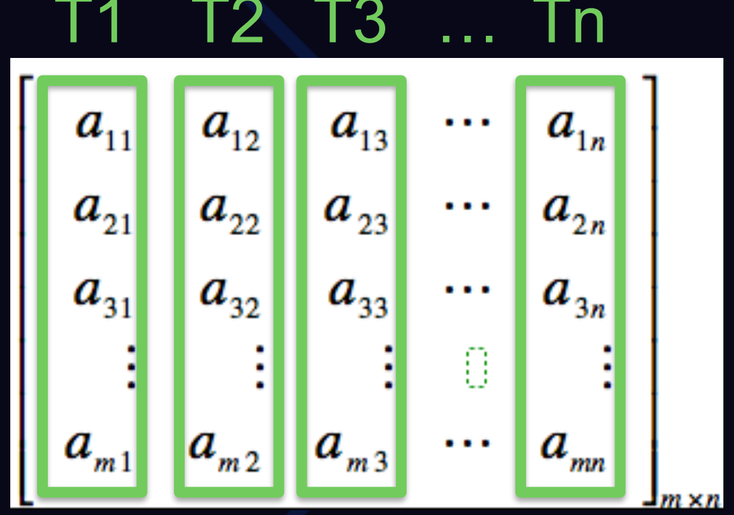
```
Total number of registers available per block: 65536
```

- 同個 block 共用相同 registers 區塊 (4 Bytes)
- Block size = 1024 (threads) → 每個 thread 有 $\frac{65536}{1024} = 64$ 個 registers
- registers不支援連續區塊劃分，無法宣告陣列
  - 若仍宣告陣列，會被分配到global memory位置
  - Performance issue!

SHARED MEMORY

```
Total amount of shared memory per block:       49152 bytes
Total shared memory per multiprocessor:        102400 bytes
```

- 相同 block 下的 threads 可以存取到同樣區塊的shared memory
- 速度約等於registers，例外：Bank conflict
- 資料共享
- Ex: Matrix multiplication



GLOBAL MEMORY

```
Total amount of global memory:                 7982 MBytes (8370061312 bytes)
```

- 容量大、速度慢
- 與CPU的溝通橋樑

CUDA INTERFACE

CUDA

- Compute Unified Device Architecture ： 統一計算架構
- GPGPU (General-purpose GPU) ：通用計算GPU
- Languages: C, C++, Fortran

MEMORY LOCATION

- CPU：RAM
- GPU：Global memory

- Malloc :

```
1   int cpu_arr[LEN];
2   int *gpu_arr;
3   cudaMalloc(&gpu_arr, sizeof(int) * LEN);
```

- Memcpy :

```
4   cudaMemcpy(gpu_arr, cpu_arr, sizeof(int) * LEN, cudaMemcpyHostToDevice);
5   cudaMemcpy(cpu_arr, gpu_arr, sizeof(int) * LEN, cudaMemcpyDeviceToHost);
```

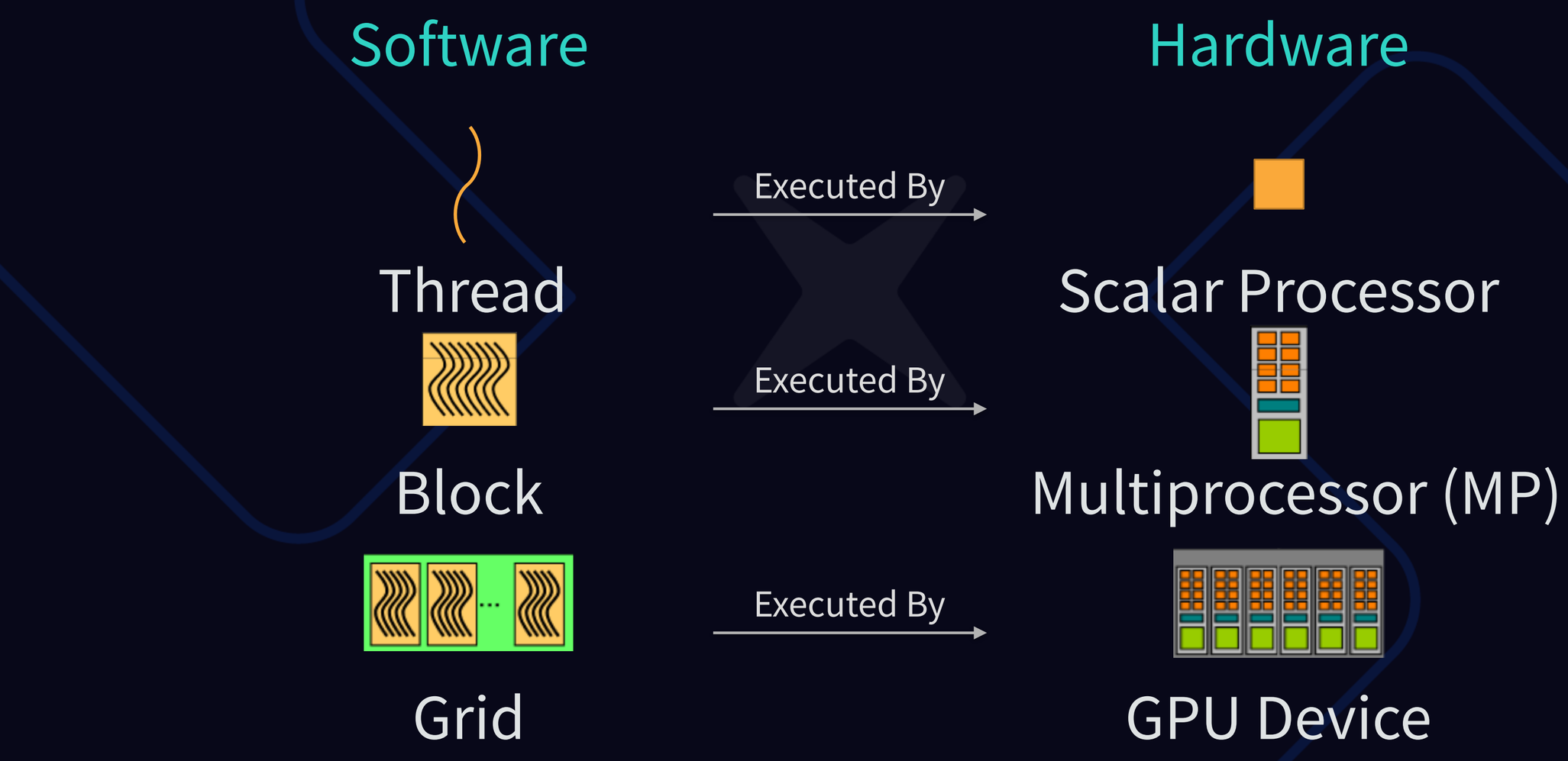
SITCON'21

DECLARE & EXCUTE FUNCTION

- `__global__ void name(…);`
- `name<<<grid_size, block_size>>>(…);`

```
1   #define LEN 1000
2   __global__ void gpu_func(int add, int *arr) {
3       arr[0] += add;
4   }
5
6   int main(int argc, char *argv[]) {
7       int cpu_arr[LEN];
8       int *gpu_arr;
9       cudaMalloc(&gpu_arr, sizeof(int) * LEN);
10      cudaMemcpy(gpu_arr, cpu_arr, sizeof(int) * LEN, cudaMemcpyHostToDevice);
11      gpu_func<<<10, 100>>>(87, gpu_arr);
12      cudaMemcpy(cpu_arr, gpu_arr, sizeof(int) * LEN, cudaMemcpyDeviceToHost);
13  }
14
```

EXECUTION MODEL



DECLARE & EXCUTE FUNCTION

- `__global__ void name(…);`
- `name<<<grid_size, block_size>>>(…);`

Block

```
1   #define LEN 1000
2   __global__ void gpu_func(int add, int *arr) {
3       arr[0] += add;
4   }
5
6   int main(int argc, char *argv[]) {
7       int cpu_arr[LEN];
8       int *gpu_arr;
9       cudaMalloc(&gpu_arr, sizeof(int) * LEN);
10      cudaMemcpy(gpu_arr, cpu_arr, sizeof(int) * LEN, cudaMemcpyHostToDevice);
11      gpu_func<<<10, 100>>>(87, gpu_arr);
12      cudaMemcpy(cpu_arr, gpu_arr, sizeof(int) * LEN, cudaMemcpyDeviceToHost);
13  }
14
```

SITCON'21

DECLARE & EXCUTE FUNCTION

- `__global__ void name(…);`
- `name<<<grid_size, block_size>>>(…);`

Grid

```
1   #define LEN 1000
2   __global__ void gpu_func(int add, int *arr) {
3       arr[0] += add;
4   }
5
6   int main(int argc, char *argv[]) {
7       int cpu_arr[LEN];
8       int *gpu_arr;
9       cudaMalloc(&gpu_arr, sizeof(int) * LEN);
10      cudaMemcpy(gpu_arr, cpu_arr, sizeof(int) * LEN, cudaMemcpyHostToDevice);
11      gpu_func<<<10, 100>>>(87, gpu_arr);
12      cudaMemcpy(cpu_arr, gpu_arr, sizeof(int) * LEN, cudaMemcpyDeviceToHost);
13  }
14
```

SITCON'21

DECLARE & EXCUTE FUNCTION

- `__global__ void name(…);`
- `name<<<grid_size, block_size>>>(…);`

```
1   #define LEN 1000
2   __global__ void gpu_func(int add, int *arr) {
3       arr[0] += add;
4   }
5
6   int main(int argc, char *argv[]) {
7       int cpu_arr[LEN];
8       int *gpu_arr;
9       cudaMalloc(&gpu_arr, sizeof(int) * LEN);
10      cudaMemcpy(gpu_arr, cpu_arr, sizeof(int) * LEN, cudaMemcpyHostToDevice);
11      gpu_func<<<10, 100>>>(87, gpu_arr);
12      cudaMemcpy(cpu_arr, gpu_arr, sizeof(int) * LEN, cudaMemcpyDeviceToHost);
13  }
14
```

總共有10*100=1000個 threads

Block & Thread index

- 每個 thread 都做相同動作？？？

```
__global__ void gpu_func(int add, int *arr) {
    arr[0] += add;
}
```

```
gpu_func<<<10, 100>>>(87, gpu_arr);
```

- blockIdx.x　threadIdx.x　blockDim.x

```
__global__ void gpu_func(int add, int *arr) {
    int bID = blockIdx.x;
    int tID = threadId.x;
    int blockSize = blockDim.x;
    arr[bID * blockSize + tID] += add;
}
```

SITCON'21

Dim3

- thread/block ID 只有 $[0, n)$
- `func<<<1, dim3(10,10)>>>`

```
__global__ void gpu_func(...) {
    int x = threadId.x;
    int y = threadId.y;
}
```

SHARED MEMORY - STATIC

- `__shared__ int s[LEN];`
- `func<<< block, thread >>>();`

```
1   #define LEN 1000
2
3   __global__ void gpu_func(int *arr, int sz) {
4       __shared__ int s[LEN];
5       int id = threadId.x;
6       int bs = blockDim.x;
7       for (int i = 0; i < LEN / bs; i++) {
8           s[i * bs + id] = arr[i * bs + id];
9       }
10      __syncthreads();
11      ...
12  }
13
14  int main(int argc, char *argv[]) {
15      gpu_func<<<1, 100>>>(gpu_arr, LEN);
16      return 0;
17  }
```

SHARED MEMORY - DYNAMIC

- `extern __shared__ int s[];`
- `func<<< block, thread, SM_size >>>();`

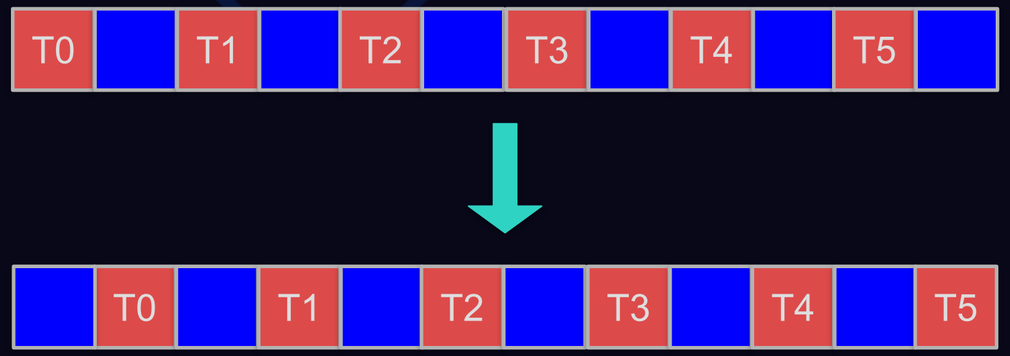
```
1   #define LEN 1000
2
3   __global__ void gpu_func(int *arr, int sz) {
4       extern __shared__ int s[];
5       int id = threadId.x;
6       int bs = blockDim.x;
7       for (int i = 0; i < LEN / bs; i++) {
8           s[i * bs + id] = arr[i * bs + id];
9       }
10      __syncthreads();
11      ...
12  }
13
14  int main(int argc, char *argv[]) {
15      gpu_func<<< 10, 100, sizeof(int)*LEN >>>(gpu_arr, LEN);
16      return 0;
17  }
```

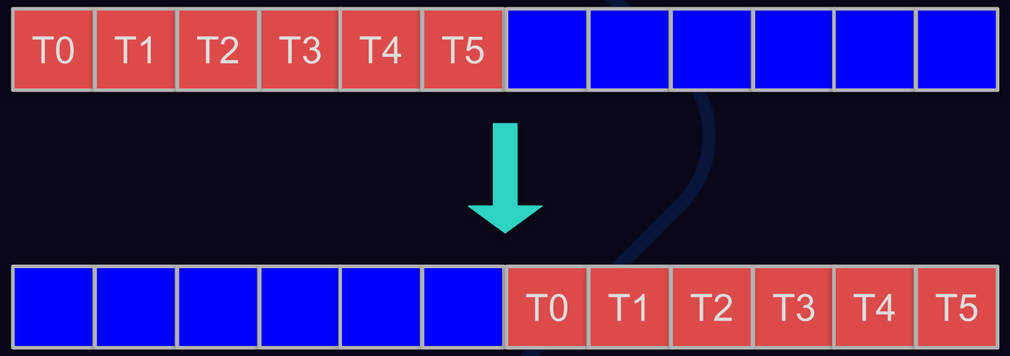
~~`extern __shared__ int *s;`~~

SITCON'21

WRITING FAST GPU PROGRAM

連續記憶體讀取	L1, L2 cache
存取相同記憶體	Shared memory
增加GPU使用率	Large block size
減少溝通次數	Copy larger memory block
Warp divergence	unroll loops…





DEMO

- All-Pairs Shortest Path
- 0/1 Knapsack

SITCON'21

Floyd-Warshall Algorithm

- 全點對最短路徑
- $f_k(i,j) = \min(f_{k-1}(i,j)\ ,\ f_{k-1}(i,k) + f_{k-1}(k,j))$
- 時間複雜度：$\Theta(n^3)$
- Problem size : $n = 2 \cdot 10^4 \rightarrow n^3 = 8 \cdot 10^{12}$

Floyd-Warshall - CPU

```
1   for (int k = 0; k < n; k++) {
2       for (int i = 0; i < n; i++) {
3           for (int j = 0; j < n; j++) {
4               g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
5           }
6       }
7   }
```

Floyd-Warshall - GPU

- 觀察 Data dependency

$$f_k(i,j) = \min(f_{k-1}(i,j)\ ,\ f_{k-1}(i,k) + f_{k-1}(k,j))$$

- 裡面兩層 loop 沒有關聯
- 將裡面兩層迴圈平行化

```
for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
        }
    }
}
```

```
__global__ void gpu(int **g, int n, int k) {
    int id = blockIdx.x * blockDim.x+ threadIdx.x;
    int i = id / n;
    int j = id % n;
    g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
}

// bs * 1024 = n^2
for (int k = 0; k < n; k++) {
    gpu<<<bs, 1024>>>(g, n, k);
}
```

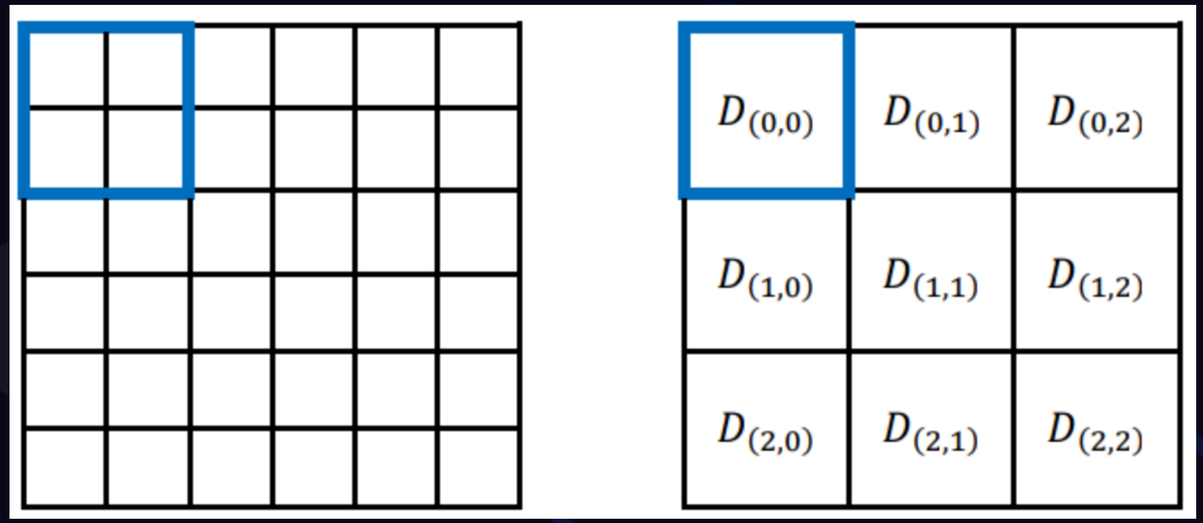
Performance Issue

- 無連續讀取記憶體讀取 `min(g[i][j], g[i][k] + g[k][j]);`
- 無共用記憶體
- 效能瓶頸為記憶體頻寬

Blocked Floyd-Warshall Algorithm

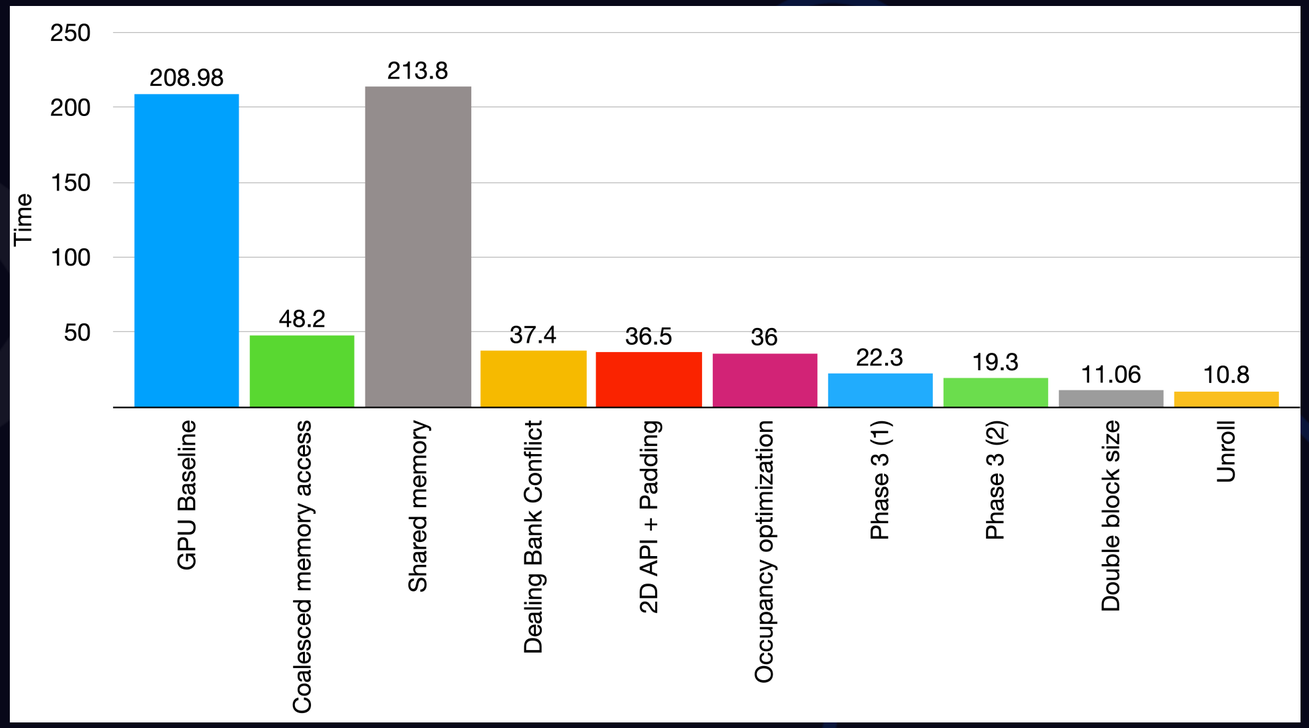
- 改善自 FW Algorithm
- 利用 block 性質，有效利用 cache
- 同時也會有不同 threads 共用相同記憶體資料

  → shared memory 加速



Optimization

- CPU Baseline: $> 10^4$
- GPU Baseline: 209
- GPU Final: 10.8



0/1 Knapsack

- *Optimization of Multi-Class 0/1 Knapsack Problem on GPUs by Minimizing Memory Overhead*
- Submitted to ACM Transactions on Parallel Computing

Model	Archi-tecture	Cores	FP32 peak op per sec	Memory size (GB)	Memory bandwidth	Price (USD)	Release year
RTX3070	Ampere	5888	20.3T @1725MHz	8	448GB/s	500	2020
V100 SXM2	Volta	5120	15.7T @1380MHz	32	900GB/s	>10,000	2017

0/1 Knapsack

- *Optimization of Multi-Class 0/1 Knapsack Problem on GPUs by Minimizing Memory Overhead*

	Original Method	Proposed Method
V100	101	14
RTX 3070	190	10

Conclusion

- 撰寫 GPU 程式需要對硬體有所了解
  - Warp divergence
  - Memory Bandwidth
- 通常 GPGPU 瓶頸在於記憶體頻寬
  - 機器學習、AI 訓練資料量大

Further Optimization Techniques

- CGMA ratio :

  https://www.sciencedirect.com/topics/computer-science/global-memory-access

- Bank conflict :

  https://blog.csdn.net/Bruce_0712/article/details/65447608

- CUDA streams:

  https://developer.download.nvidia.com/CUDA/training/StreamsAndConcurrencyWebinar.pdf