CUDA programming
Shared memory, divergence, synchronization

Oguz Kaya

Maˆıtre de Conf'erences
Universit'e Paris-Saclay and the LRI ParSys team, Orsay, France

Objectives

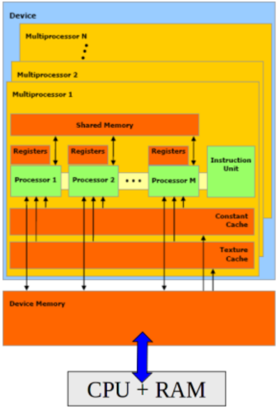
- Zoom on the memory architecture of a GPU
- Use of shared-memory for fast memory access Synchronization of threads
- Divergence concept for efficient branch management

Outline

Outline

GPU memory architecture

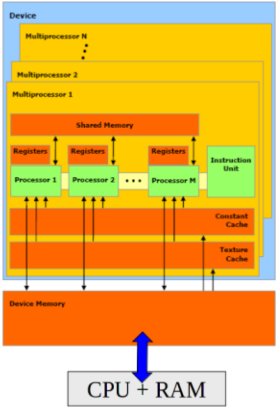
There are many types of memory in a GPU.



- Global memory (RAM): 8-48GB, 500-2000GB/s, 300-400 cycles per access
- Registers: 65536/SM, access immediat (1/cycle)
- L1 cache: 64-192KB/SM, ≈ 4 cycles per access
- L2 cache: 8-40MB, worse latency/bandwidth than L1
- Constant cache: Constant memory by SM

GPU memory architecture

There are many types of memory in a GPU.



- Shared memory: Shared by all threads of a
  - 32-128Kb/SM block
  - Technology same as L1 cache
  - Latency ≈ 4 cycles/access
  - Bandwidth comparable to registers No need
  - for coalescing (as efficient) Can compete
  - with the capacity of the
    L1 (e.g., unified L1+shared memory)

Outline

Example: Power calculation in a table

Given an array A[N] and an integer k, set A[i] to $A[i]^k$.

```
#include <cstdio>
#include "cuda.h"

#define N 1024
#define BLOCKSIZE 128
float A[N];

__device__ float dA[N];

__global__ void powerArray(int n, int k)
{
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  if (i < n) {
    float c = 1.0;
    for (int j = 0; j < k; j++) { c *= dA[i];
    }
    dA[i] = c;
  }
}

int main(int argc, char **argv)
{
  // Initialization
  for (int i = 0; i < N; i++) { A[i] = i; }
  // Copy the table to the GPU
  cudaMemcpyToSymbol(dA, A, N * sizeof(float), 0,
      cudaMemcpyHostToDevice);
  int blockSize = 128;
  int numBlocks = N / blockSize;
  if (N % blockSize) numBlocks++;
  powerArray<<<numBlocks, blockSize>>>(N, 4);
  // Copy the table to the CPU
  cudaMemcpyFromSymbol(A, dA, N * sizeof(float), 0, cuda
      MemcpyDeviceToHost);
  printf("%lf\n", A[2]);
  return 0;
}
```

- 1D blocks/threads
- Each thread updates 1 element with *k* multiplications
- *dA*[*i*] is accède *k* times. Can we make better?
  - Put *dA*[*i*] in a register (i.e., float temp = dA[i];) Use
  - the shared memory

université PARIS-SACLAY

Example: Power calculation in a table

Given an array A[N] and an integer k, set A[i] to $A[i]^k$.

```
#include <cstdio>
#include "cuda.h"

#define N 1024
#define BLOCKSIZE 128
float A[N];

__device__ float dA[N];

__global__ void powerArray(int n, int k)
{
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  // BLOCKSIZE == blockDim.x
  __shared__ float data[BLOCKSIZE];
  if (i < n) {
    data[threadIdx.x] = dA[i];
    float c = 1.0;
    for (int j = 0; j < k; j++) {
      c *= data[threadIdx.x];
    }
    dA[i] = c;
  }
}

int main(int argc, char **argv)
{
  /// ...
  powerArray<<<numBlocks, blockSize>>>(N, 4);
  /// ...
  printf("%lf\n", A[2]);
  return 0;
}
```

- The prefix **__shared__** to define an array in shared memory
- The size must être a known **constant** at compile time (so impossible to use **blockDim.x**)
- dA[i] is accessed **once** and then reused **k times** in shared memory.
- The table is released when the block ends so **there is** no need to **allocate it**

Outline

Connections in a GPU kernel

The threads are executed in groups of 32 in a **warp**.

- All threads execute the same instruction simultaneously. If
- there is a **branch**: if (cond) f(); else g();
  - If all threads in a warp satisfy **cond**, they only execute f() simultaneously.
  - If all threads in a warp fail, they only execute g() simultaneously.
  - If there is **at least one thread** that satisfies **cond** and **at least one thread** that 'fails **cond**
    - First f() is executed by those who satisfy **cond**, the rest fall asleep. Then g()
    - is executed by those who 'fail **cond**, the rest fall asleep. The total execution
    - time of the warp is therefore the sum of the two.

Example: Two types of discrepancies

Given an array A[N] and an integer k, set A[i] to $A[i]^k$.

```
__device__ void kernel()
{
  // ...
  // Divergence 1
  if (threadIdx.x % 2 == 0) {
    f();
  } else {
    g();
  }
  // ...
  // Divergence 2
  if (threadIdx.x < SIZE / 2) {
    f();
  } else {
    g();
  }
  // ...
}
```

- Suppose *f* () and *g* () take *F* and *G* seconds respectively in the execution by a thread.
- Divergence 1 concerns **all warps**, so the total execution time of each warp will be $F + G$.
- Divergence 2 concerns **only one warp** among all, each other warp takes either *F* or *G* seconds in the execution.

Example: Two types of discrepancies

Given an array $A[N]$ and an integer k, set $A[i]$ to $A[i]^k$.

```
__device__ void kernel()
{
  // ...
  // Divergence 1
  if (threadIdx.x % 2 == 0) {
    f();
  } else {
    g();
  }
  // ...
  // Divergence 2
  if (threadIdx.x < SIZE / 2) {
    f();
  } else {
    g();
  }
  // ...
}
```

- For good performance, you should
  - either 'avoid connections as much as possible
  - or to make them in a way that is not too difficult. of warps diverge in the execution.

Contact

Oguz Kaya
Université Paris-Saclay and LRI, Paris, France
oguz.kaya@lri.com
www.oguzkaya.com