Lập trình song song trên GPU

Hoàng Minh Thanh (18424062)

BT4 : Tính tổng tích lũy

Bộ môn Công nghệ phần mềm

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

Contents

Vì máy tính cả nhân của em không có GPU nên bắt buộc em phải sử dụng Google Colab : https://colab.research.google.com/drive/1TeARxOiMfQQ4AT8Gy61KvgNprqs5UIme#scrollTo=KBj0mTbo8JrV

(Thầy có thể vào link Online để xem luồng chạy dễ hơn báo cáo)

Quá trình cài đặt

Chi tiết cài đặt trong file .cu và file Google Colab

Kết quả của toàn bộ các phương pháp :

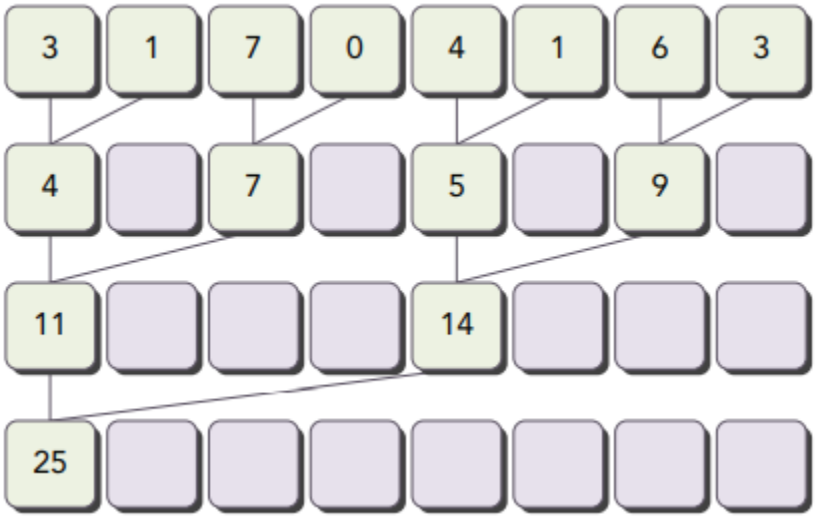
```
############ THÔNG TIN GPU ############
Device 0: Tesla P100-PCIE-16GB
Kích thước mảng : 16777216
Kích thước : <<<Grid (512, 1), Block (32768, 1)>>>
ID| Time          | Sum result      | <<<GridSize, BlockSize >>> | Kernel
1 | 0.049916 sec | 8389084.624453 |            | recursiveReduce-CPU
2 | 0.002044 sec | 8389084.624453 |<<<32768, 512>>> | reduceNeighbored
3 | 0.001141 sec | 8389084.624453 |<<<32768, 512>>> | reduceNeighboredLess
4 | 0.000958 sec | 8389084.624453 |<<<32768, 512>>> | reduceInterleaved
5 | 0.000620 sec | 8389084.624453 |<<<16384, 512>>> | reduceUnrolling2
6 | 0.000382 sec | 8389084.624453 |<<<8192, 512>>> | reduceUnrolling4
7 | 0.000318 sec | 8389084.624453 |<<<4096, 512>>> | reduceUnrolling8
8 | 0.000306 sec | 8389084.624453 |<<<4096, 512>>> | reduceUnrollWarps8
9 | 0.000306 sec | 8389084.624453 |<<<4096, 512>>> | reduceCompleteUnrollWarsp8
10| 0.000305 sec | 8389084.624453 |<<<4096, 512>>> | reduceCompleteUnroll
Sum on CPU : 8389084.624453

Sum on GPU : 8389084.624453
```

Block và Grid được cấu hình theo **1D grid** và **1D block**

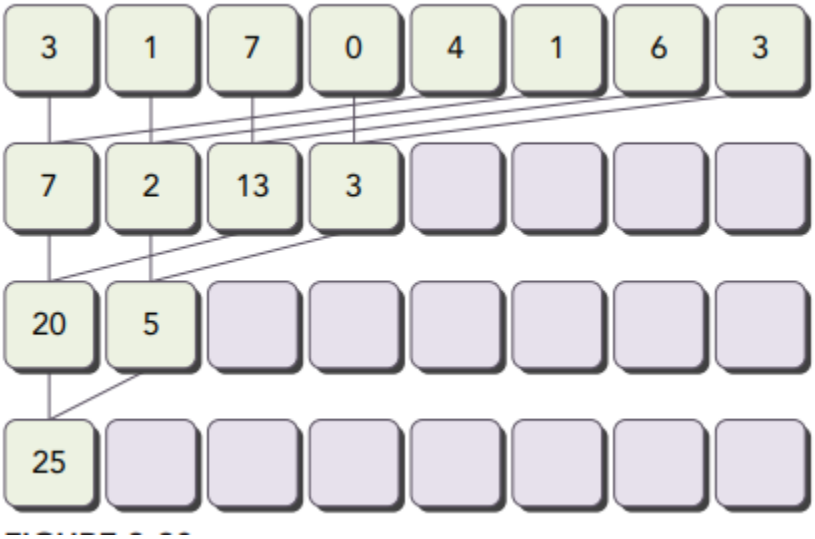
1) Hàm hệ qui reduce trên CPU

Trên CPU ta cài đặt đệ qui để tính toán

Phương pháp tính tổng mảng bằng hàng xóm kế cận



Phương pháp tính tổng bằng phương pháp tính tổng một nửa mảng

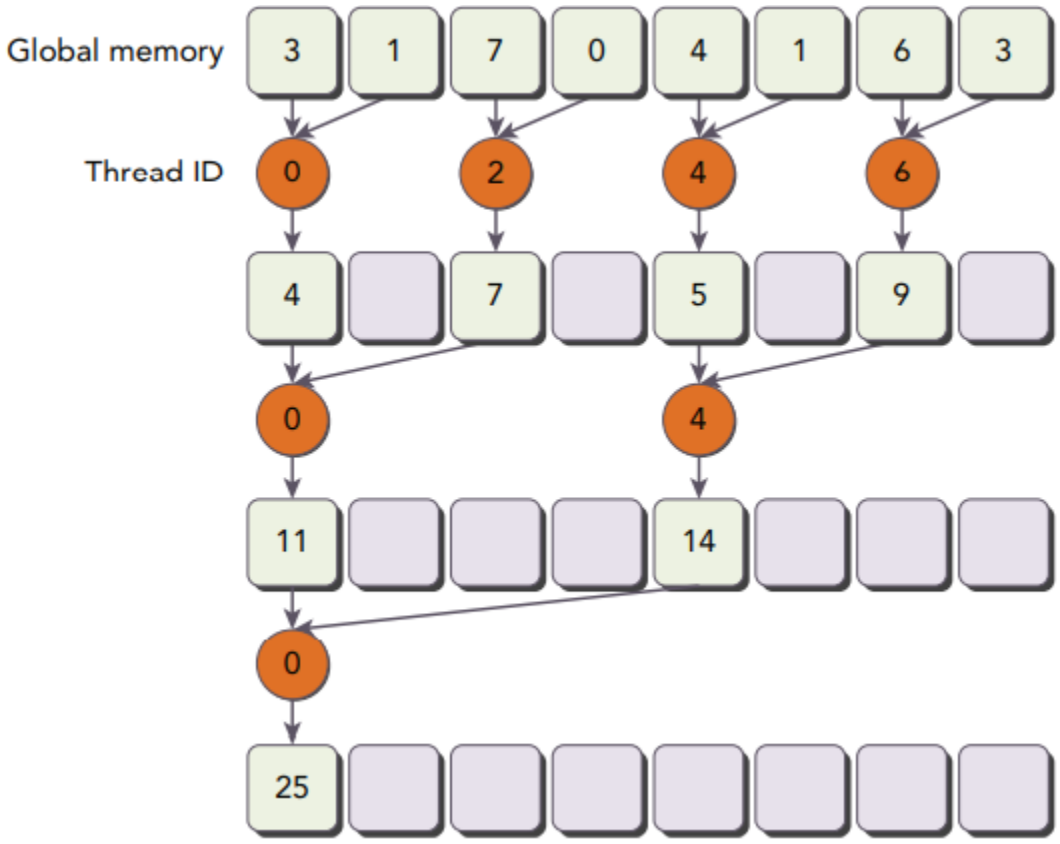


```
48  // ################ Device(CPU) ################
49  // Hàm thực hiện reduce trên CPU
50  double recursiveReduce(double *data, int const size)
51  {
52      if (size == 1) return data[0];
53
54      int const stride = size / 2;
55      for (int i = 0; i < stride; i++){
56          data[i] += data[i + stride];
57      }
58
59      return recursiveReduce(data, stride);
60  }
61
```

2) Hàm reduceNeighbored :

Tương tự như phương pháp trên CPU nhưng ta sẽ chia công việc tính toán ra các thread trong GPU để có tốc độ tính toán cao hơn .

Phương pháp này sử dụng một biến stride tăng dần và tính tổng trên bộ nhớ toàn cục



Cài đặt :

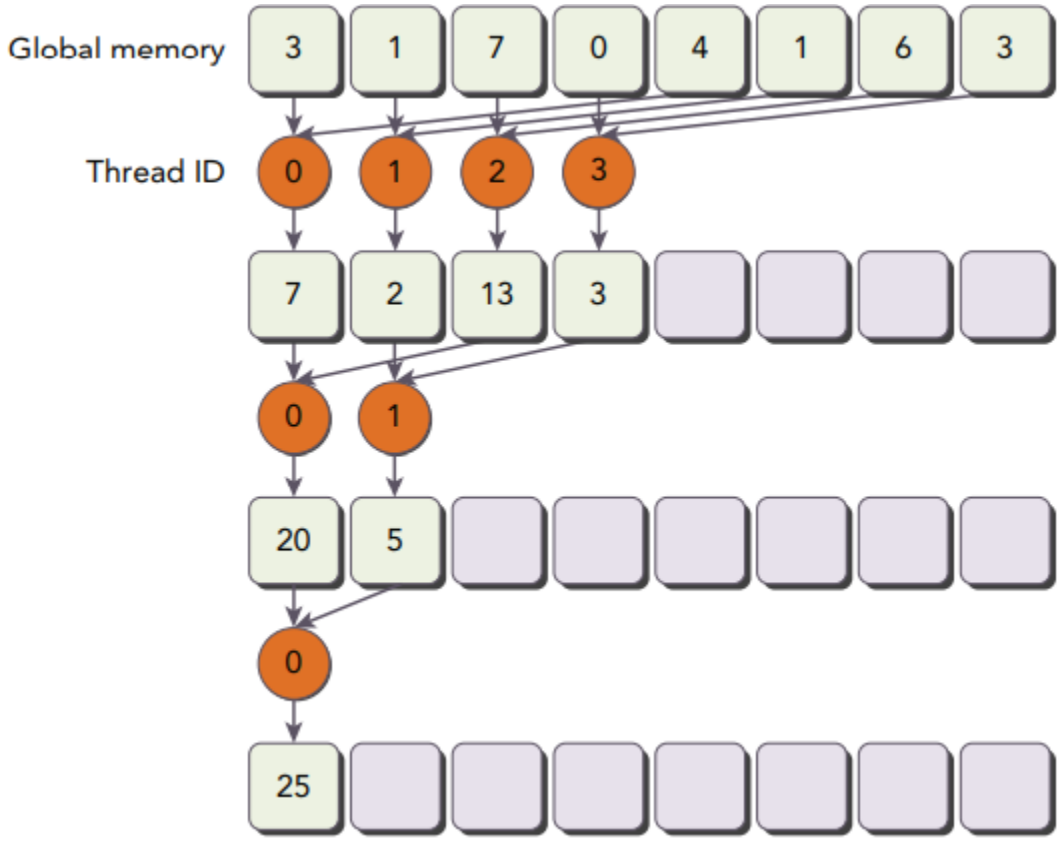
```cpp
// Neighbored Pair phân kỳ
__global__ void reduceNeighbored (double *g_idata, double *g_odata,
    unsigned int n)
{
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;

    // Chuyển từ con trỏ toàn cục sang con trỏ của block này
    double *idata = g_idata + blockIdx.x * blockDim.x;

    // Kiểm tra nếu vượt qua kích thước mảng
    if (idx >= n) return;

    // in-place reduction in global memory
    for (int stride = 1; stride < blockDim.x; stride *= 2)
    {
        if ((tid % (2 * stride)) == 0)
        {
            idata[tid] += idata[tid + stride];
        }

        // Đồng bộ hóa trong một threadBlock
        __syncthreads();
    }

    // Ghi kết quả cho block này vào bộ nhớ toàn cục
    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

3) Hàm reduceNeighboredLess:

Phương pháp này tối ưu hơn phương pháp trên ở chỗ thay vì sử dụng index của toàn cục (**idx**) thì sử dụng index của các thread trên một block (**tid**); giúp quá trình tính toán ngay trên register (register – thanh ghi)

```
92  // Neighbored Pair cài đặt với ít phân kỳ bằng cách thực thi trong một block
93  __global__ void reduceNeighboredLess (double *g_idata, double *g_odata,
94      unsigned int n)
95  {
96      unsigned int tid = threadIdx.x;
97      unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;
98
99      // Chuyển từ con trỏ toàn cục sang con trỏ của block này
100     double *idata = g_idata + blockIdx.x * blockDim.x;
101
102     // Kiểm tra nếu vượt qua kích thước mảng
103     if(idx >= n) return;
104
105     // Thực hiện tính tổng ở bộ nhớ toàn cục
106     for (int stride = 1; stride < blockDim.x; stride *= 2)
107     {
108         // Chuyển tid sang bộ nhớ của một block (register - thanh ghi)
109         int index = 2 * stride * tid;
110
111         if (index < blockDim.x)
112         {
113             idata[index] += idata[index + stride];
114         }
115
116         // Đồng bộ hóa trong một threadBlock
117         __syncthreads();
118     }
119
120     // Ghi kết quả cho block này vào bộ nhớ toàn cục
121     if (tid == 0) g_odata[blockIdx.x] = idata[0];
122 }
```

4) Hàm reduceInterleaved:



Phương pháp này tốt hơn phương pháp trên vì các bộ nhớ không bị phân mảnh, gom chung vào trong một wrap (tuy vẫn bị phân mảnh nhưng đỡ hơn)

Cài đặt :

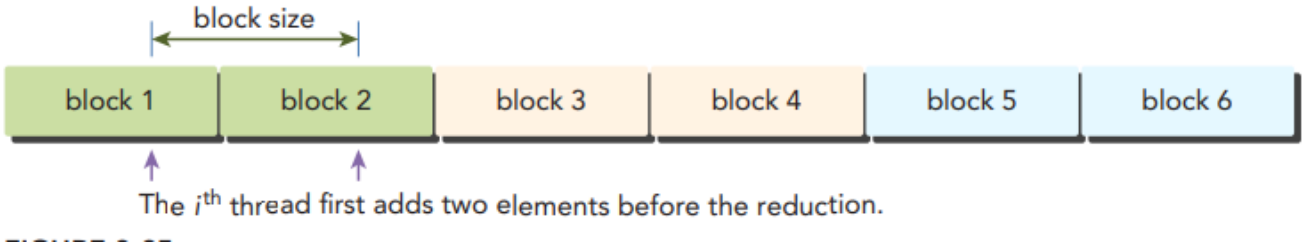
```cpp
// Interleaved Pair Implementation with less divergence
__global__ void reduceInterleaved (double *g_idata, double *g_odata, unsigned int n)
{
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;

    // Chuyển từ con trỏ toàn cục sang con trỏ của block này
    double *idata = g_idata + blockIdx.x * blockDim.x;

    // Kiểm tra nếu vượt qua kích thước mảng
    if(idx >= n) return;

    // Thực hiện tính tổng ở bộ nhớ toàn cục
    for (int stride = blockDim.x / 2; stride > 0; stride >>= 1)
    {
        if (tid < stride)
        {
            idata[tid] += idata[tid + stride];
        }

        __syncthreads();
    }

    // Ghi kết quả cho block này vào bộ nhớ toàn cục
    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

UNROLLING LOOPS :

UNROLLING LOOPS : là nhóm các phương pháp thay vì thực hiện vòng lặp for; thì phải tốn thời gian so sánh if và tăng biến đếm với mỗi lần lặp thì thực hiện thủ công bằng tay; Sự cải tiến này đến từ việc sự cải tiến và tối ưu của instruction, vì compiler unrolled vòng lặp, giúp cung cấp cho warp scheduler nhiều eligible wrap hơn để giúp giảm độ trễ instruction và memory (bộ nhớ)

5) Hàm reduceUnrolling2:

```
unsigned int idx = blockIdx.x * blockDim.x * 2 + threadIdx.x;
int *idata = g_idata + blockIdx.x * blockDim.x * 2;
```



Ta sẽ coi hai block kế cận là một **group data** (một block/nhóm gồm 2 block). Sau đó với mỗi thread ta sẽ cộng thread của block thứ hai vào thread tương ứng của block thứ nhất. Tương tự thực hiện với reduceUnrolling4, reduceUnrolling8 ta sẽ coi 4 hoặc 8 thread là một group data, sau đó ta cộng các thread của 4, 8 block còn lại vào thread đầu tiên.

Sau đó khi tính toán song song xong, ta chỉ cần tính tổng của mỗi thread của block đầu tiên của mỗi **group data**

Cài đặt :

```cpp
__global__ void reduceUnrolling2 (double *g_idata, double *g_odata,
    unsigned int n)
{
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x * 2 + threadIdx.x;

    // Chuyển từ con trỏ toàn cục sang con trỏ của block này
    double *idata = g_idata + blockIdx.x * blockDim.x * 2;

    // unrolling 2 data blocks
    if (idx + blockDim.x < n) g_idata[idx] += g_idata[idx + blockDim.x];
    // Đồng bộ hóa các group data trong 2 thread kết cận
    __syncthreads();

    // Thực hiện tính tổng ở bộ nhớ toàn cục
    for (int stride = blockDim.x / 2; stride > 0; stride >>= 1)
    {
        if (tid < stride)
        {
            idata[tid] += idata[tid + stride];
        }

        // Đồng bộ hóa trong một threadBlock
        __syncthreads();
    }

    // Ghi kết quả cho block này vào bộ nhớ toàn cục
    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

6) Hàm reduceUnrolling4:

Tương tự như reduceUnrolling2; nhưng vì có 4 biến cần tính tổng nên ta khai báo các biến riêng biệt để lưu trữ trên bộ nhớ register của mỗi block

Cài đặt

```cpp
__global__ void reduceUnrolling4(double *g_idata, double *g_odata, unsigned int n){
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x * 4 + threadIdx.x;

    // Chuyển từ con trỏ toàn cục sang con trỏ của block này
    double *idata = g_idata + blockIdx.x * blockDim.x * 4;

    // unrolling 4
    if (idx + 3 * blockDim.x < n)
    {
        double a1 = g_idata[idx];
        double a2 = g_idata[idx + blockDim.x];
        double a3 = g_idata[idx + 2 * blockDim.x];
        double a4 = g_idata[idx + 3 * blockDim.x];
        g_idata[idx] = a1 + a2 + a3 + a4;
    }
    __syncthreads(); // // Đồng bộ hóa các group data trong 4 thread kết cận

    // Thực hiện tính tổng ở bộ nhớ toàn cục
    for (int stride = blockDim.x / 2; stride > 0; stride >>= 1)
    {
        if (tid < stride)
        {
            idata[tid] += idata[tid + stride];
        }

        // Đồng bộ hóa trong một threadBlock
        __syncthreads();
    }

    // Ghi kết quả cho block này vào bộ nhớ toàn cục
    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

7) Hàm reduceUnrolling8:

Tương tự như reduceUnrolling4; ta sẽ khai báo 8 biến và tính tổng vào thread đầu tiên của một group data

```cpp
__global__ void reduceUnrolling8 (double *g_idata, double *g_odata, unsigned int n){
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x * 8 + threadIdx.x;
    // Chuyển từ con trỏ toàn cục sang con trỏ của block này
    double *idata = g_idata + blockIdx.x * blockDim.x * 8;

    // unrolling 8
    if (idx + 7 * blockDim.x < n)
    {
        double a1 = g_idata[idx];
        double a2 = g_idata[idx + blockDim.x];
        double a3 = g_idata[idx + 2 * blockDim.x];
        double a4 = g_idata[idx + 3 * blockDim.x];
        double b1 = g_idata[idx + 4 * blockDim.x];
        double b2 = g_idata[idx + 5 * blockDim.x];
        double b3 = g_idata[idx + 6 * blockDim.x];
        double b4 = g_idata[idx + 7 * blockDim.x];
        g_idata[idx] = a1 + a2 + a3 + a4 + b1 + b2 + b3 + b4;
    }
    __syncthreads(); // Đồng bộ hóa các group data trong 8 thread kết cận

    // Thực hiện tính tổng ở bộ nhớ toàn cục
    for (int stride = blockDim.x / 2; stride > 0; stride >>= 1){
        if (tid < stride){
            idata[tid] += idata[tid + stride];
        }

        // Đồng bộ hóa trong một threadBlock
        __syncthreads();
    }
    // Ghi kết quả cho block này vào bộ nhớ toàn cục
    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

Unrolled Warps

__syncthreads : là dòng lệnh giúp đồng bộ hóa các thread trong một group data từ phương pháp trên

SIMT (Single Instruction Multiple Thread) - một câu lệnh được thực thi đồng thời cho tất các thread trong warp (mỗi thread có dữ liệu riêng của mình)

Từ khóa __ syncthreads sẽ giúp đồng bộ hóa các thread trong một block, để đảm bảo với mỗi vòng tính tổng thì kết quả đã được ghi vào bộ nhớ toàn cục hoàn tất và không có thread nào đang còn ghi để tiếp tục vòng tiếp theo. Phương pháp này sẽ giúp unroll casc thread trong cùng một wrap (32 thread)

Một block có tối đa 1024 thread (chiều x – theo cấu hình Fermi hoặc Kepler), một wrap có 32 thread. Nên nếu chỉ còn lại 32 thread chắc chắn sẽ thuộc một wrap; nên ta sẽ thực hiện có thể unroll.

8) Hàm reduceUnrollWarps8:

Từ khóa **volatile** là từ khóa để đảm bảo giá trị load lên bộ nhớ luôn là mới nhất; để tránh hiện tượng giá trị trong bộ nhớ thay đổi bất thường do quá trình tính toán song song hóa. Nghĩa là có thể có một thread khác đang chạy, và chạy trước và ghi vào bộ nhớ, nếu không có từ khóa **volatile** thì chương trình sẽ lấy kết quả từ bộ nhớ tạm mà không lấy kết quả từ bộ nhớ lưu trữ.

Tham khảo tại : https://ktmt.github.io/blog/2013/05/09/y-nghia-cua-tu-khoa-volatile-trong-c/

Cài đặt :

```
__global__ void reduceUnrollWarps8 (double *g_idata, double *g_odata, unsigned int n)
{
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x * 8 + threadIdx.x;
    double *idata = g_idata + blockIdx.x * blockDim.x * 8;

    if (idx + 7 * blockDim.x < n)
    {
        double a1 = g_idata[idx];
        double a2 = g_idata[idx + blockDim.x];
        double a3 = g_idata[idx + 2 * blockDim.x];
        double a4 = g_idata[idx + 3 * blockDim.x];
        double b1 = g_idata[idx + 4 * blockDim.x];
        double b2 = g_idata[idx + 5 * blockDim.x];
        double b3 = g_idata[idx + 6 * blockDim.x];
        double b4 = g_idata[idx + 7 * blockDim.x];
        g_idata[idx] = a1 + a2 + a3 + a4 + b1 + b2 + b3 + b4;
    }
    __syncthreads();

    // Thực hiện tính tổng ở bộ nhớ toàn cục
    for (int stride = blockDim.x / 2; stride > 32; stride >>= 1)
    {
        if (tid < stride)
        {
            idata[tid] += idata[tid + stride];
        }

        // Đồng bộ hóa trong một threadBlock
        __syncthreads();
    }
```

Ta thay đổi điều kiện stride > 0 bằng stride > 32

```
    // unrolling warp
    if (tid < 32)
    {
        volatile double *vmem = idata;
        vmem[tid] += vmem[tid + 32];
        vmem[tid] += vmem[tid + 16];
        vmem[tid] += vmem[tid +  8];
        vmem[tid] += vmem[tid +  4];
        vmem[tid] += vmem[tid +  2];
        vmem[tid] += vmem[tid +  1];
    }

    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

9) Hàm reduceCompleteUnrollWarsp8:

```
__global__ void reduceCompleteUnrollWarps8 (double *g_idata, double *g_odata,
        unsigned int n){
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x * 8 + threadIdx.x;
    double *idata = g_idata + blockIdx.x * blockDim.x * 8;

    if (idx + 7 * blockDim.x < n)
    {
        double a1 = g_idata[idx];
        double a2 = g_idata[idx + blockDim.x];
        double a3 = g_idata[idx + 2 * blockDim.x];
        double a4 = g_idata[idx + 3 * blockDim.x];
        double b1 = g_idata[idx + 4 * blockDim.x];
        double b2 = g_idata[idx + 5 * blockDim.x];
        double b3 = g_idata[idx + 6 * blockDim.x];
        double b4 = g_idata[idx + 7 * blockDim.x];
        g_idata[idx] = a1 + a2 + a3 + a4 + b1 + b2 + b3 + b4;
    }
    __syncthreads(); // Đồng bộ hóa tất cả các thread trong một block
```

```
    // in-place reduction and complete unroll
    if (blockDim.x >= 1024 && tid < 512) idata[tid] += idata[tid + 512];
    __syncthreads();

    if (blockDim.x >= 512 && tid < 256) idata[tid] += idata[tid + 256];
    __syncthreads();

    if (blockDim.x >= 256 && tid < 128) idata[tid] += idata[tid + 128];
    __syncthreads();

    if (blockDim.x >= 128 && tid < 64) idata[tid] += idata[tid + 64];
    __syncthreads();

    // unrolling warp
    if (tid < 32)
    {
        volatile double *vsmem = idata;
        vsmem[tid] += vsmem[tid + 32];
        vsmem[tid] += vsmem[tid + 16];
        vsmem[tid] += vsmem[tid +  8];
        vsmem[tid] += vsmem[tid +  4];
        vsmem[tid] += vsmem[tid +  2];
        vsmem[tid] += vsmem[tid +  1];
    }

    // Ghi kết quả cho block này vào bộ nhớ toàn cục
    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

Tương tự như hàm trên nhưng ta unroll toàn bộ kích thước tối đa của một block

10) Hàm reduceCompleteUnroll:

Phương pháp này sử dụng một kỹ thuật **template** của C++ để tối ưu; nghĩa là nó tạo ra một bản sao nhiều cái của cùng một hàm với mỗi iBlockSize khác nhau. Khi đó ta chương trình sẽ không phải so sánh kích thước của blockSize mà chỉ gọi hàm với mỗi blockSize khác nhau

```
template <unsigned int iBlockSize>
__global__ void reduceCompleteUnroll(double *g_idata, double *g_odata,
                                     unsigned int n)
{
    unsigned int tid = threadIdx.x;
    unsigned int idx = blockIdx.x * blockDim.x * 8 + threadIdx.x;

    // Chuyển từ con trỏ toàn cục sang con trỏ của block này
    double *idata = g_idata + blockIdx.x * blockDim.x * 8;

    // unrolling 8
    if (idx + 7 * blockDim.x < n)
    {
        double a1 = g_idata[idx];
        double a2 = g_idata[idx + blockDim.x];
        double a3 = g_idata[idx + 2 * blockDim.x];
        double a4 = g_idata[idx + 3 * blockDim.x];
        double b1 = g_idata[idx + 4 * blockDim.x];
        double b2 = g_idata[idx + 5 * blockDim.x];
        double b3 = g_idata[idx + 6 * blockDim.x];
        double b4 = g_idata[idx + 7 * blockDim.x];
        g_idata[idx] = a1 + a2 + a3 + a4 + b1 + b2 + b3 + b4;
    }

    __syncthreads(); // Đồng bộ tất các thread trong một block
```

```
    // in-place reduction and complete unroll
    if (iBlockSize >= 1024 && tid < 512) idata[tid] += idata[tid + 512];
    __syncthreads();

    if (iBlockSize >= 512 && tid < 256)  idata[tid] += idata[tid + 256];
    __syncthreads();

    if (iBlockSize >= 256 && tid < 128)  idata[tid] += idata[tid + 128];
    __syncthreads();

    if (iBlockSize >= 128 && tid < 64)   idata[tid] += idata[tid + 64];
    __syncthreads();

    // unrolling warp
    if (tid < 32)
    {
        volatile double *vsmem = idata;
        vsmem[tid] += vsmem[tid + 32];
        vsmem[tid] += vsmem[tid + 16];
        vsmem[tid] += vsmem[tid +  8];
        vsmem[tid] += vsmem[tid +  4];
        vsmem[tid] += vsmem[tid +  2];
        vsmem[tid] += vsmem[tid +  1];
    }

    // Ghi kết quả cho block này vào bộ nhớ toàn cục
    if (tid == 0) g_odata[blockIdx.x] = idata[0];
}
```

Kết quả thực thi

```
############ THÔNG TIN GPU ############
Device 0: Tesla P100-PCIE-16GB
Kích thước mảng : 16777216
Kích thước : <<<Grid (512, 1), Block (32768, 1)>>>
ID| Time                | Sum result            | <<<GridSize, BlockSize >>> | Kernel
1 | 0.049916 sec        | 8389084.624453        |                | recursiveReduce-CPU
2 | 0.002044 sec        | 8389084.624453        |<<<32768, 512>>> | reduceNeighbored
3 | 0.001141 sec        | 8389084.624453        |<<<32768, 512>>> | reduceNeighboredLess
4 | 0.000958 sec        | 8389084.624453        |<<<32768, 512>>> | reduceInterleaved
5 | 0.000620 sec        | 8389084.624453        |<<<16384, 512>>> | reduceUnrolling2
6 | 0.000382 sec        | 8389084.624453        |<<<8192, 512>>> | reduceUnrolling4
7 | 0.000318 sec        | 8389084.624453        |<<<4096, 512>>> | reduceUnrolling8
8 | 0.000306 sec        | 8389084.624453        |<<<4096, 512>>> | reduceUnrollWarps8
9 | 0.000306 sec        | 8389084.624453        |<<<4096, 512>>> | reduceCompleteUnrollWarsp8
10| 0.000305 sec        | 8389084.624453        |<<<4096, 512>>> | reduceCompleteUnroll
Sum on CPU : 8389084.624453
Sum on GPU : 8389084.624453
```

Tỷ lệ tối ưu về hiệu suất tính toán

TABLE 3-5: Reduction Kernel Performance

KERNEL	TIME (S)	STEP SPEEDUP	CUMULATIVE SPEEDUP
Neighbored (divergence)	0.011722		
Neighbored (no divergence)	0.009321	1.26	1.26
Interleaved	0.006967	1.34	1.68
Unroll 8 blocks	0.001422	4.90	8.24
Unroll 8 blocks + last warp	0.001355	1.05	8.65
Unroll 8 blocks + loop + last warp	0.001280	1.06	9.16
Templatized kernel	0.001253	1.02	9.35

Kết quả tỉ lệ tối ưu của lưu trữ

TABLE 3-6: Load/Store Efficiency

KERNEL	TIME (S)	LOAD EFFICIENCY	STORE EFFICIENCY
Neighbored (divergence)	0.011722	16.73%	25.00%
Neighbored (no divergence)	0.009321	16.75%	25.00%
Interleaved	0.006967	77.94%	95.52%
Unroll 8 blocks	0.001422	94.68%	97.71%
Unroll 8 blocks + last warp	0.001355	98.99%	99.40%
Unroll 8 blocks + loop + last warp	0.001280	98.99%	99.40%
Templatized the last kernel	0.001253	98.99%	99.40%

II. Báo cáo và rút ra nhận xét, kết luận

Có thể thấy để tối ưu quá trình tính toán ta có nhiều cách khác nhau như :

* unroll trên block

* unroll trên một wrap

* sử dụng template function

* sử dụng kỹ thuật stride

* sử dụng kỹ thuật stride để tránh phân mảnh

Mỗi một câu lệnh đều có thể ảnh hưởng đến hiệu suất tính toán.

Có tham khảo mã nguồn tại : https://github.com/hmthanh/ProfessionalCUDACProgramming

(Repo của em)