**GPU**

**Radix sort trên GPU**

Tài liệu này ghi nhận lại báo cáo Đồ án cuối kỳ môn học ***GPU***

*Sinh viên thực hiện:*

**1512416 – Nguyễn Tất Nam Phương**

**1512473 – Trương Ngọc Tài**

C:\Users\tdqua_000\Dropbox\SS-Slides\DeCuong-CDIO\Template CDIO v4.2\Templates\Hinh anh\LogoTruong.png

**Khoa Công nghệ thông tin**

**Đại học Khoa học tự nhiên TP HCM**

**Tháng 1/2019**

Mục lục

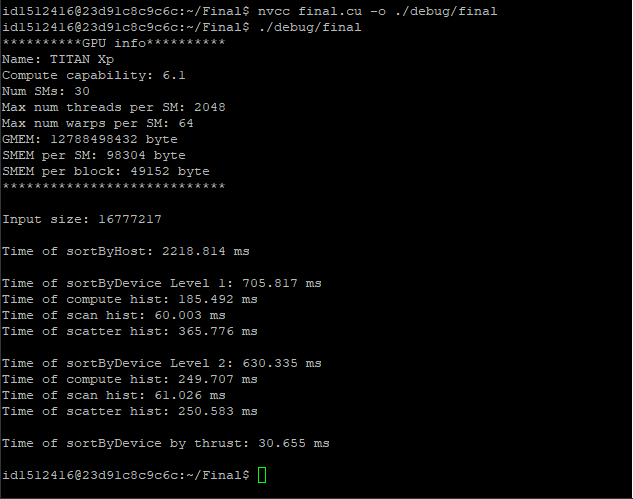
[Kết quả chạy 3](#_Toc534674978)

[Thời gian chạy 4](#_Toc534674979)

[Level 1 5](#_Toc534674980)

[Level 2 7](#_Toc534674981)

# Kết quả chạy



# Thời gian chạy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phiên bản | Thời gian tính hist | Thời gian scan | Thời gian scatter | Tổng |
| Level 1 | 185.492 | 60.003 | 365.776 | 705.817 |
| Level 2 | 249.707 | 61.026 | 250.583 | 630.335 |
| Thrust | | | | 30.655 |

# Level 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phiên bản | Thời gian tính hist | Thời gian scan | Thời gian scatter | Tổng |
| Level 1 | 185.492 | 60.003 | 365.776 | 705.817 |

Duyệt mảng từ “least significant digit” đến “most significant digit” (mỗi digit gồm k bit). Sắp xếp các digit bằng counting sort:

// Compute histogram

computeHistTimer.Start();

    memset(hist, 0, nBins \* numBlks \* sizeof(int));

computeHistByDevice(src, n, hist, nBins, bit, blkSize);

computeHistTimer.Stop();

scanTimer.Start();

int curScan = 0;

for (int binIdx = 0; binIdx < nBins; binIdx++) {

for (int blkIdx = 0; blkIdx < numBlks; blkIdx++) {

int histIdx = blkIdx \* nBins + binIdx;

histScan[histIdx] = curScan;

curScan += hist[histIdx];

}

};

scanTimer.Stop();

    // Scatter

scatterTimer.Start();

    scatterByDevice(src, dst, n, histScan, nBins, bit, blkSize);

scatterTimer.Stop();

    // Swap src and dst

    uint32\_t \* temp = src;

    src = dst;

    dst = temp;

* Tính histogram trên mỗi block. Mảng hist gồm 2k cột và numBlocks dòng. Có thể tính song song được giữa các block.

\_\_global\_\_ void computeHist(uint32\_t \* in, int n, int \* hist, int nBins, int bit)

{

int idx = blockDim.x \* blockIdx.x + threadIdx.x;

if (idx < n) {

int bin = (in[idx] >> bit) & (nBins - 1);

int histIdx = blockIdx.x \* nBins + bin;

atomicAdd(&hist[histIdx], 1);

}

}

* Sau khi có mảng scan, kích thước không quá lơn, có thể scan trực tiếp trên host.
* Với mảng scan, ta tiến hành scatter kết quả. Chỉ có thể song song giữa các block

\_\_global\_\_ void scatter(uint32\_t \* in, uint32\_t \* out, int n, int \* histScan, int nBins, int bit)

{

int idx = blockIdx.x \* blockDim.x;

if (threadIdx.x == 0) {

for (int i = 0; i < blockDim.x; i++){

if (idx + i < n) {

int bin = (in[idx + i] >> bit) & (nBins - 1);

int histIdx = blockIdx.x \* nBins + bin;

out[histScan[histIdx]] = in[idx + i];

histScan[histIdx]++;

}

}

}

}

Kết quả: Giai đoạn scatter chiếm nhiều thời gian nhất. Do chưa song song được giữa các thread trong cung block.

# Level 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phiên bản | Thời gian tính hist | Thời gian scan | Thời gian scatter | Tổng |
| Level 2 | 249.707 | 61.026 | 250.583 | 630.335 |

Cải tiến thời gian scatter bằng cách sắp xếp các giá trị trong block theo digit.

* Dễ dàng tính rank mà k còn bị phụ thuộc giữa các thread với nhau.

Trong hàm tính hist, t sắp xếp lại các giá trị theo digit.

* Sử dụng SMEM để lưu giá trị local của block.

extern \_\_shared\_\_ uint32\_t s\_data[];

int baseCopyIdx = blockDim.x;

int baseHistIdx = blockDim.x \* 2;

Mảng s\_data[] được chia thành 3 phần, phần 1 để lưu giá trị in – đầu vào, phần 2 để lưu giá trị scan phục vụ để sắp xếp và phần cuối để lưu giữ liệu scatter.

1. Đầu tiên cần chép dữ liệu vào s\_data[]

s\_data[threadIdx.x] = in[idx];

\_\_syncthreads();

1. Tiến hành radix sort với k = 1
   1. Tính bit hiện tại

s\_data[baseCopyIdx + threadIdx.x] = (s\_data[threadIdx.x] >> (bit+b)) & 1;

\_\_syncthreads();

Exclusive scan với giá trị vừa tính

* 1. Tính tổng số bit 0 có trong block = Kích thước block – giá trị scan cuối – giá trị bit cuối.

int numZeros = blockSize - s\_data[baseHistIdx + blockSize-1] - s\_data[baseCopyIdx + blockSize-1];

* 1. Tính rank dựa vào bit hiện tại.
  + bit = 0 => rank bằng chỉ số hiện tại trừ đi số bit 1 trước nó

rank = threadIdx.x - s\_data[baseHistIdx + threadIdx.x];

* + bit = 1 => rank bằng số bit 1 trước nó cộng thêm tổng số 0

rank = numZeros + s\_data[baseHistIdx + threadIdx.x];

* 1. Local scatter dựa vào rank

1. Copy lại host

Sau khi dữ liệu trong từng local đã được sắp xếp thì việc tính rank trong lúc scatter giữa các thread không bị phụ thuộc nhau.

int baseIdx = blockIdx.x \* blockDim.x;

int idx = baseIdx + threadIdx.x;

Trước tiên cần scan giá trị localhist. Mảng nhỏ nên có thể scan với 1 thread.

extern \_\_shared\_\_ int localHistScan[];

if (threadIdx.x == 0) {

localHistScan[0] = 0;

for (int binIdx = 1; binIdx < nBins; binIdx++) {

int histIdx = blockIdx.x \* nBins + binIdx - 1;

localHistScan[binIdx] = localHistScan[binIdx - 1] + hist[histIdx];

}

}

\_\_syncthreads();

Mảng histScan của bin trong block cho biết số giá trị nhỏ hơn hoặc bằng bin ở các block trước nó. Còn localHistScan lại cho biết số giá trị có bin nhỏ hơn nó.

* rank = histScan[bin trong block] + threadIdx – localHistScan[bin];

if (idx < n) {

int bin = (in[idx] >> bit) & (nBins - 1);

int histIdx = blockIdx.x \* nBins + bin;

int rank = histScan[histIdx] + threadIdx.x - localHistScan[bin];

out[rank] = in[idx];

}