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

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

Tháng 1/2019

Mục lục

Kết quả chạy

```
id1512416@23d91c8c9c6c:~/Final$ nvcc final.cu -o ./debug/final
id1512416@23d91c8c9c6c:~/Final$ ./debug/final
**********GPU info**********
Name: TITAN Xp
Compute capability: 6.1
Num SMs: 30
Max num threads per SM: 2048
Max num warps per SM: 64
GMEM: 12788498432 byte
SMEM per SM: 98304 byte
SMEM per block: 49152 byte
*******************************

Input size: 16777217

Time of sortByHost: 2218.814 ms

Time of sortByDevice Level 1: 705.817 ms
Time of compute hist: 185.492 ms
Time of scan hist: 60.003 ms
Time of scatter hist: 365.776 ms

Time of sortByDevice Level 2: 630.335 ms
Time of compute hist: 249.707 ms
Time of scan hist: 61.026 ms
Time of scatter hist: 250.583 ms

Time of sortByDevice by thrust: 30.655 ms

id1512416@23d91c8c9c6c:~/Final$
```

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 2^k 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();
```

2. Tiến hành radix sort với k = 1

2.1. Tính bit hiện tại

```
s_data[baseCopyIdx + threadIdx.x] = (s_data[threadIdx.x] >> (bit+b)) & 1;
__syncthreads();
```

Exclusive scan với giá trị vừa tính

2.2. 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];
```

2.3. 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];
```

2.4. Local scatter dựa vào rank

3. 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];
}
```