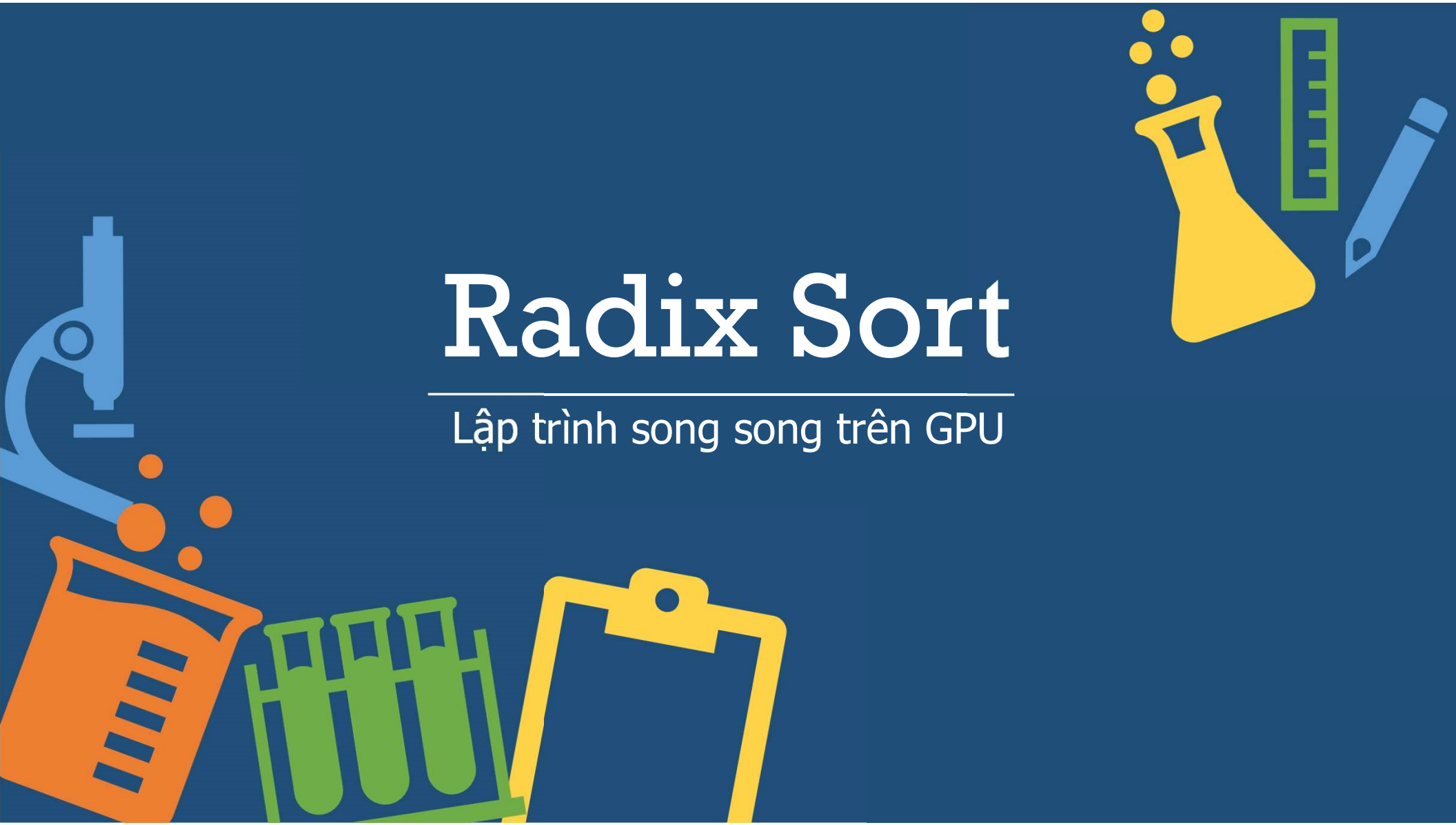
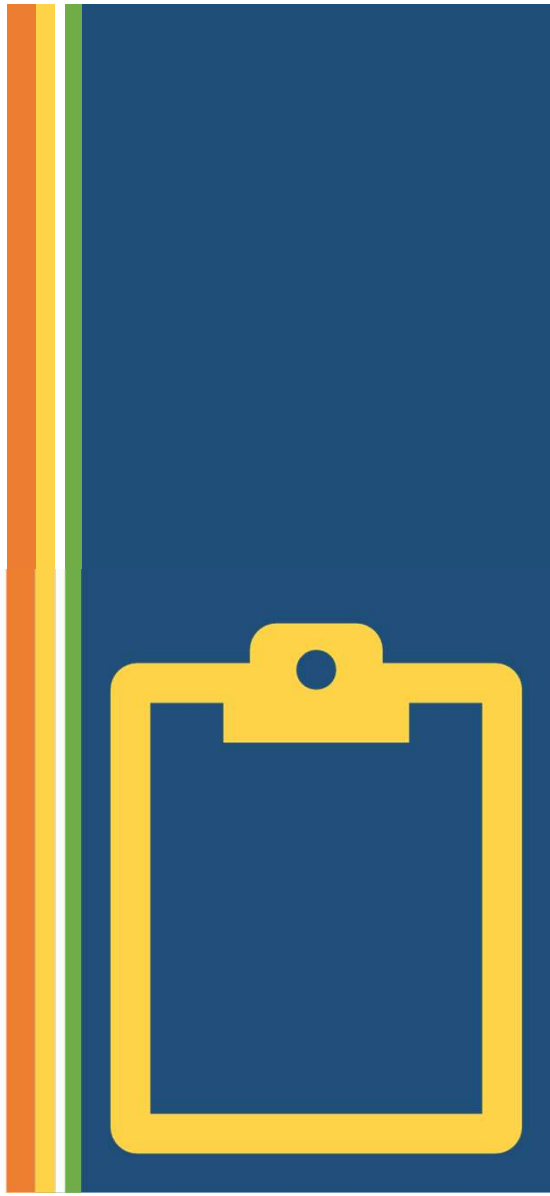

Radix Sort
Lập trình song song trên GPU

Thành viên nhóm

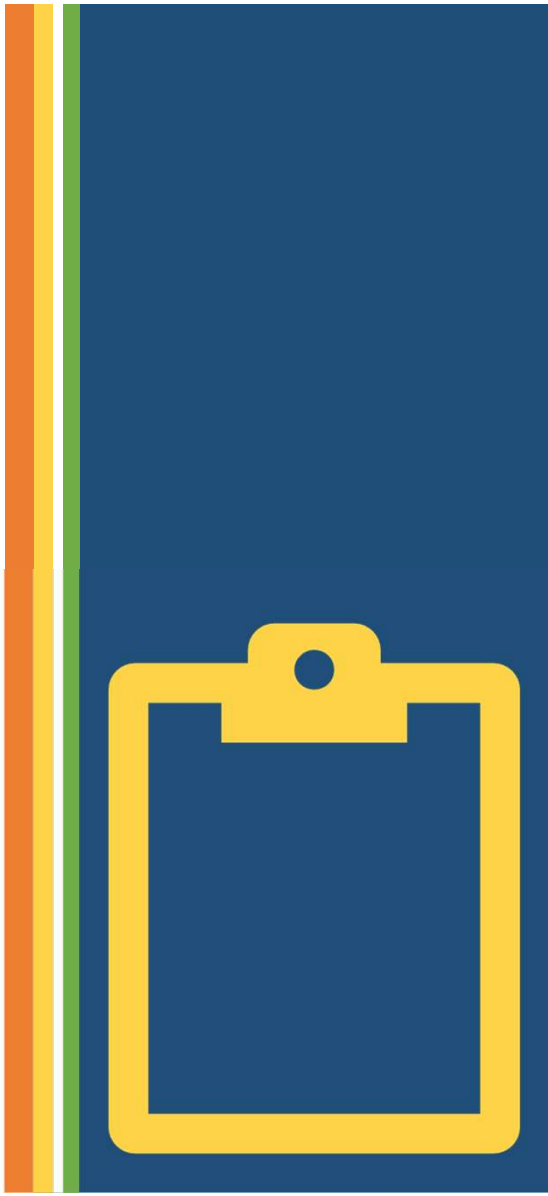
- 1512416 - Nguyễn Tât Nam Phương
- 1512473 - Trương Ngọc Tài



Môi trường cài đặt

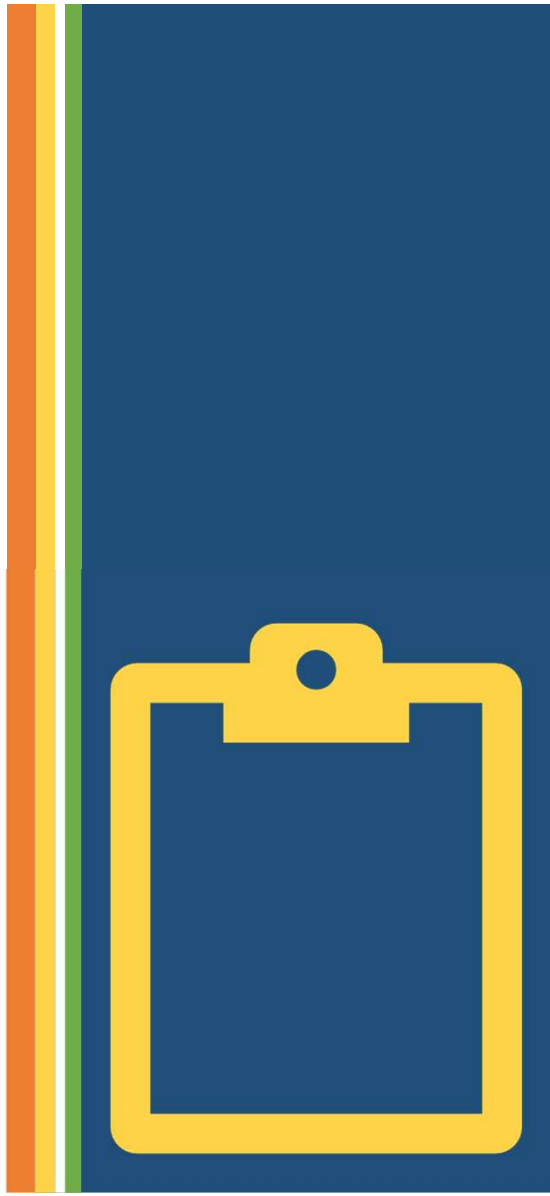
Name : TITAN Xp

Computer capability: 6.1



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



Cài đặt level 1

Cài đặt tuần tự theo bài báo

Chạy vòng lặp, duyệt từ "least significant digit" đến "most significant digit"

Với từng digit, dùng Counting sort

- Tính hist
- Scan local hist
- Scatter

Cài đặt level 1

Tính hist

- Thời gian: 185.492 ms

Ta tính local hist trên từng block.

Mảng hist sẽ có [Số bin] cột và [Số block] dòng. Giá trị hist[blockIdx][bin] cho biết trong block blockIdx có bao nhiêu giá trị bin.

```
// Tính giá trị bin
int bin = (in[idx] >> bit) & (nBins - 1);
// Tính chỉ số của bin hiện tại
int histIdx = blockIdx.x * nBins + bin;
// Cập nhật giá trị hist
atomicAdd(&hist[histIdx], 1);
```

Cài đặt level 1

Scan hist

- Thời gian: 60.003 ms

Ta scan local hist trên từng block.

Mảng scanHist có kích thước tương tự mảng hist. Dùng exclusive scan theo chiều tang dần của bin.

Giá trị hist[blockIdx][bin] cho biết trước block blockIdx có bao nhiêu giá trị nhỏ hơnbin.

Kích thước mảng này không quá lớn, có thể scan trên host.

Cài đặt level 1

Scatter

- Thời gian: 365.776 ms

Dựa vào local scan, ta tính rank của giá trị in và scatter vào out.

Tuy nhiên, việc tính rank tại vị trí idx phụ thuộc vào việc cập nhật mảng histScan từ các vị trí trước. Nên chỉ có thể song song giữa các block với nhau.

```
// Tính giá trị bin và chỉ số của bin
int bin = (in[idx + i] >> bit) & (nBins - 1);
int histIdx = blockIdx.x * nBins + bin;
// Scatter ra mảng out và cập nhật lại mảng histScan
out[histScan[histIdx]] = in[idx + i];
histScan[histIdx]++;
```

Cài đặt level 2

Cải tiến scatter

Phân tích

Trong level trước, thời gian để scatter chiếm nhiều nhất. Do chưa cài đặt song song được các thread trong cùng block.

⇒Tìm cách tính rank mà không bị phụ thuộc giữa các thread.

Cài đặt level 2

Cải tiến scatter

Thiết kế

Ta đã có localHist là giá trị hist có trong block và localScanHist là giá trị scan của nó.

Nhắc lại là, giá trị hist[blockIdx][bin] cho biết trước block blockIdx có bao nhiêu giá trị nhỏ hơn bin.

⇒ Ta cần biết tại vị trí idx trong block có bao nhiêu giá trị có bin nhỏ hơn nó

⇒ Sắp xếp lại giá trị trong từng block theo digit

Cài đặt level 2

Cải tiến scatter

Cài đặt

Bổ sung hàm tính hist, thêm 1 bước sắp xếp các giá trị trong block.

Sử dụng SMEM để lưu giá trị local trong 1 block.

```
extern __shared__ uint32_t s_data[];
int baseCopyIdx = blockDim.x;
int baseHistIdx = blockDim.x * 2;
```

Mảng s_data sẽ chứa 3 phần dữ liệu. Phần đầu để lưu giá trị local, phần 2 để lưu giá trị scan và phần cuối để lưu giá trị của mảng scatter.

Sau mỗi vòng lặp, cập nhật lại giá trị local bằng giá trị scatter.

Lưu giá trị local	Lưu giá trị hist scan	Lưu giá trị để scatter

Cài đặt level 2

Cải tiến scatter

Cài đặt

Đầu tiên, ta load dữ liệu vào SMEM

```
// 1. Input current data to block mem
s_data[threadIdx.x] = in[idx];
__syncthreads();
```

Sau đó tiến hành lặp để sắp xếp mảng theo giá trị digit hiện tại.

Số lần lặp bằng số bit của digit. Sử dụng radix sort với k = 1

```
// 2. Local sort
for (int b = 0; b < nBits; b++) {}
```

Cài đặt level 2

Cải tiến scatter

Cài đặt

Quá trình sort:

- Tính giá trị bit hiện tại
- Exclusive scan cho mảng bit => Giá trị scan tại threadIdx cho biết trước nó có bao nhiêu bit 1
- Tính tổng số bit 0. numZeros = blockSize – Giá trị scan cuối – Giá trị bit cuối
- Tính rank, và dựa vào rank để scatter

```
if (s_data[baseCopyIdx + threadIdx.x] == 0)
        rank = threadIdx.x - s_data[baseHistIdx + threadIdx.x];
if (s_data[baseCopyIdx + threadIdx.x] == 1)
        rank = numZeros + s_data[baseHistIdx + threadIdx.x];
```

Cài đặt level 2

Cải tiến scatter

Cài đặt

Cuối cùng ta load lại dữ liệu đã sắp xếp ra mảng in và tính hist

```
// 3. Copy value to host
in[idx] = s_data[threadIdx.x];
// 4. Compute hist
int bin = (s_data[threadIdx.x] >> bit) & (nBins - 1);
int histIdx = blockIdx.x * nBins + bin;
atomicAdd(&hist[histIdx], 1);
```

Cài đặt level 2

Cải tiến scatter

Cài đặt

Quay lại với bước scatter. Giờ ta đã có mảng localHist, scanLocalHist và dữ liệu trong block đã được sắp xếp theo digit hiện tại.

Với giá trị localHist, ta scan ở từng block. Giá trị localScan sẽ cho biết số giá trị có bin nhỏ hơn tại tại vị trí của nó.

```
localHistScan[0] = 0;
for (int binIdx = 1; binIdx < nBins; binIdx++) {
        int histIdx = blockIdx.x * nBins + binIdx - 1;
        localHistScan[binIdx] = localHistScan[binIdx - 1] +
hist[histIdx];
}
```

Cài đặt level 2

Cải tiến scatter

Cài đặt

Khi đó việc tính rank đơn giản hơn nhiều. Mỗi thread có thể tính rank mà không cần phụ thuộc vào các thread trước.

rank = [Số giá trị có bin nhỏ hơn hoặc bằng ở các block trước] + [Chỉ số hiện tại] – [Số giá trị có bin nhỏ hơn ở block này]

```
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];
```

Sau khi có giá trị rank thì mỗi thread có thể scatter xuống mảng out.

Cài đặt level 2

Cải tiến scatter

Nhận xét

Sau khi cải tiến thì thời gian scatter giảm nhưng thời gian tính hist lại tăng do thêm công đoạn sắp xếp lại mảng.

Cảm ơn thầy đã lắng nghe