

Sắp xếp

Trần Trung Kiên
ttkien@fit.hcmus.edu.vn



KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Nội dung

- ☐ Bài toán sắp xếp
- ☐ Radix sort tuần tự
- ☐ Radix sort song song

Bài toán sắp xếp

in

1	8	5	2	6	4	7	2
---	---	---	---	---	---	---	---

Stable sort

1	2	2	4	5	6	7	8
---	---	---	---	---	---	---	---

Unstable sort

1	2	2	4	5	6	7	8
---	---	---	---	---	---	---	---

Trước mắt, ta chỉ tập trung làm với mảng **số nguyên không dấu**

Nội dung

- ☐ Bài toán sắp xếp
- ☐ Radix sort tuần tự
- ☐ Radix sort song song

Counting sort tuần tự

Rank của $\text{in}[0]$

= Số phần tử bên trái $\text{in}[0]$ mà $\leq \text{in}[0]$ + Số phần tử bên phải $\text{in}[0]$ mà $< \text{in}[0]$
= $0 + 0 = 0$

in	0	1	8	5	2	6	4	7	2
idx	0	1	2	3	4	5	6	7	

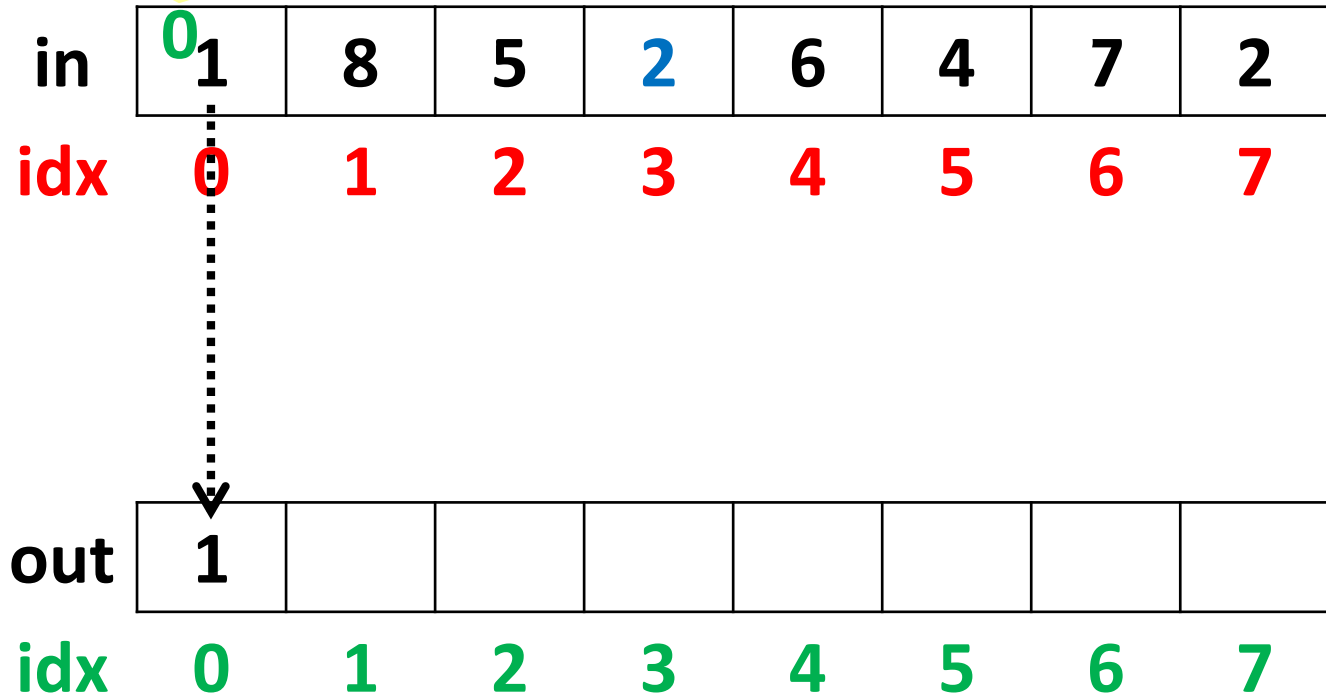
out									
idx	0	1	2	3	4	5	6	7	

Counting sort tuần tự

Rank của $\text{in}[0]$

= Số phần tử bên trái $\text{in}[0]$ mà $\leq \text{in}[0]$ + Số phần tử bên phải $\text{in}[0]$ mà $< \text{in}[0]$

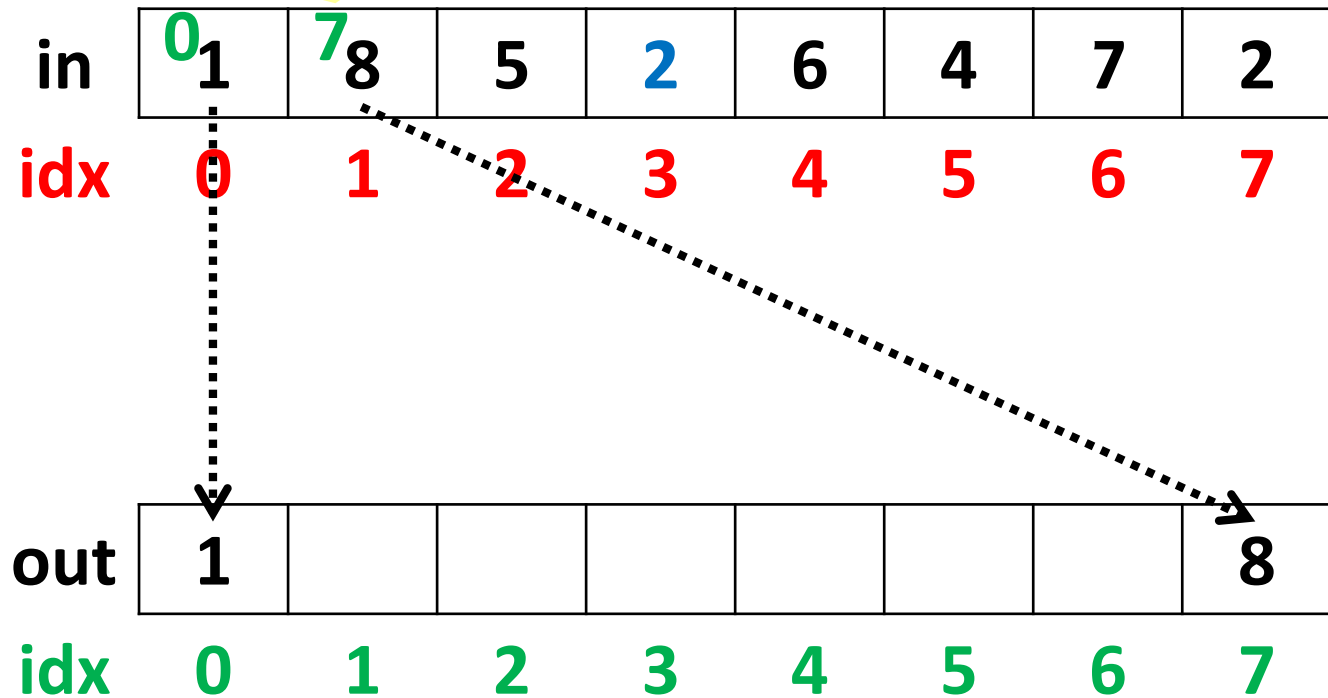
= $0 + 0 = 0$



Counting sort tuần tự

Rank của in[1]

= Số phần tử bên trái in[1] mà \leq in[1] + Số phần tử bên phải in[1] mà $<$ in[1]
= 1 + 6 = 7

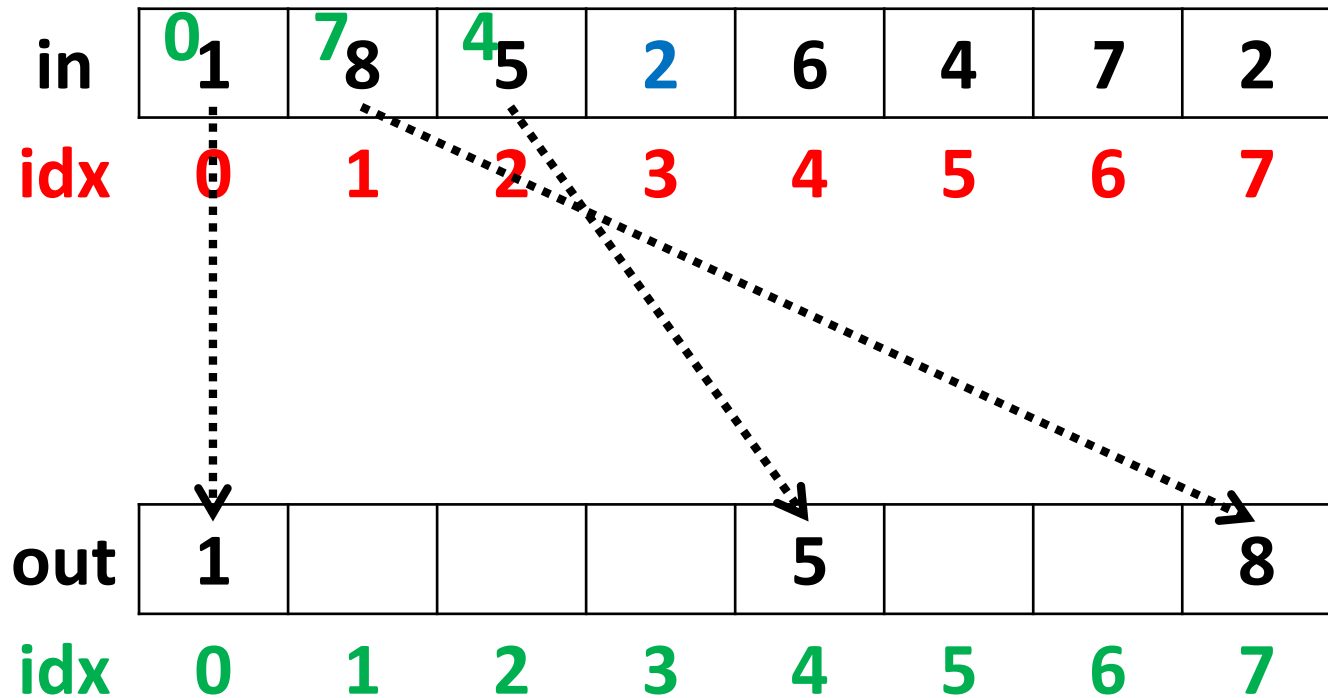


Counting sort tuần tự

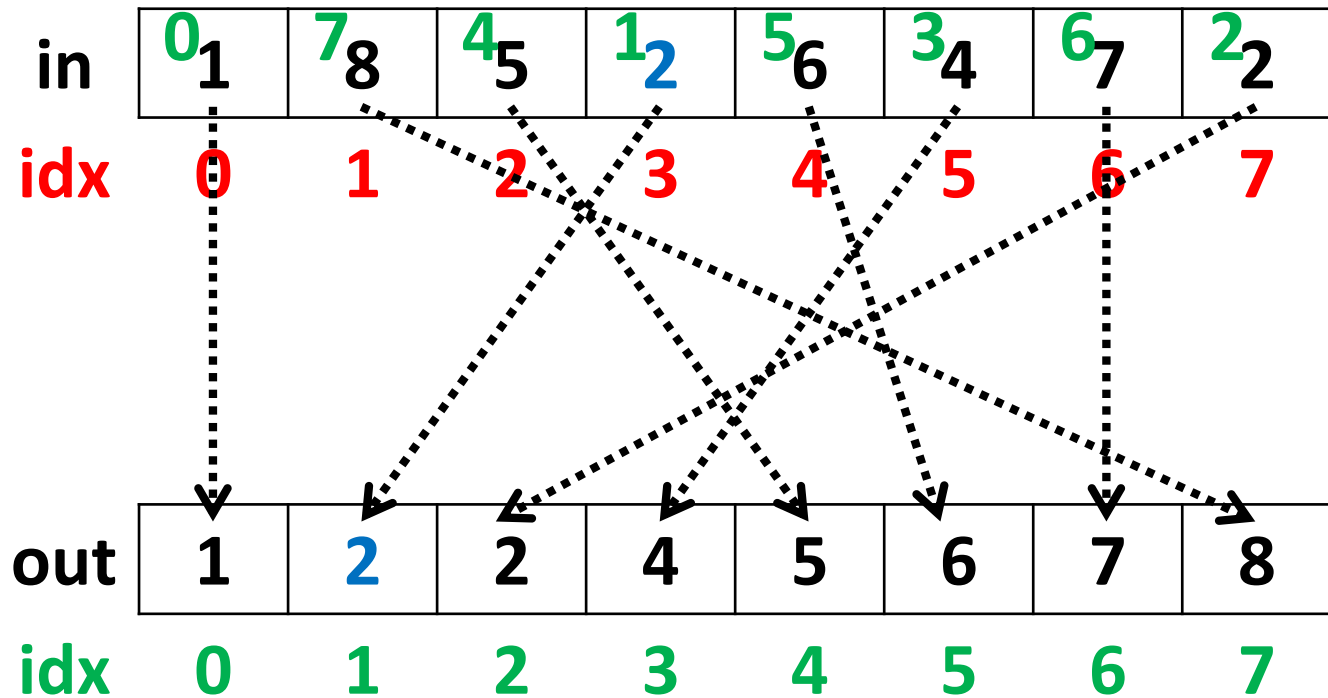
Rank của in[2]

= Số phần tử bên trái in[2] mà \leq in[2] + Số phần tử bên phải in[2] mà $<$ in[2]

= 1 + 3 = 4



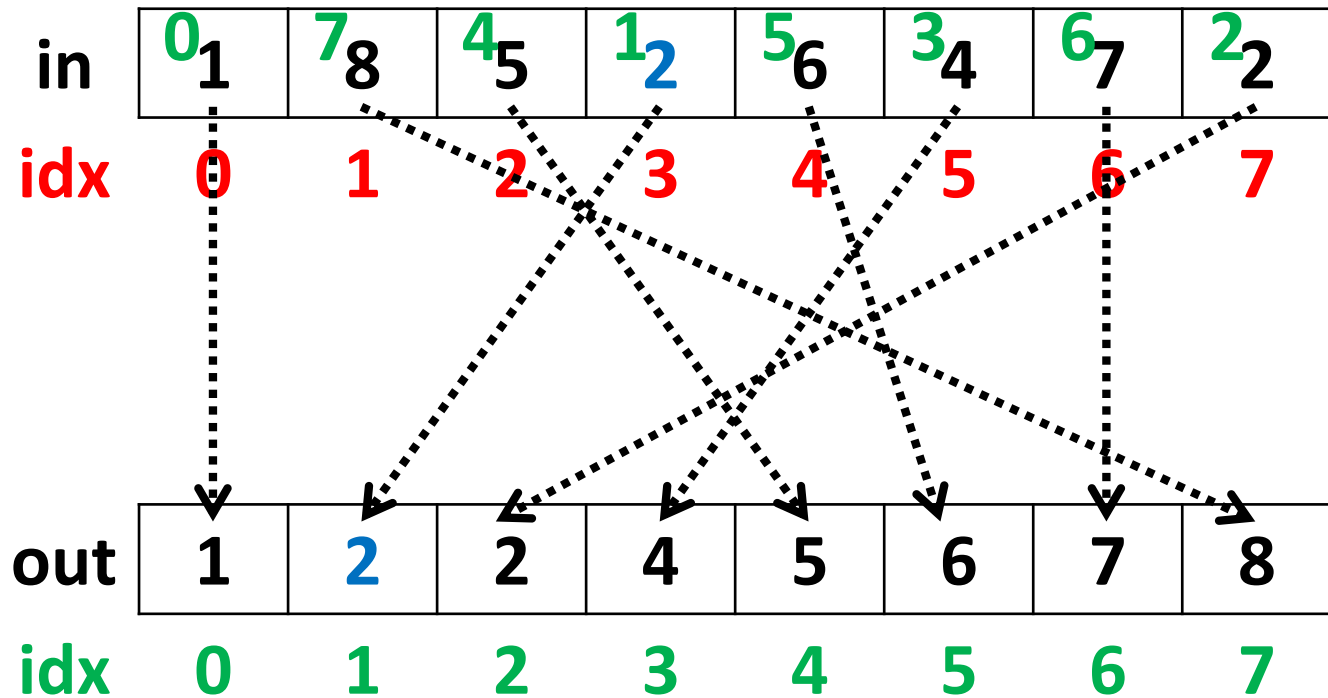
Counting sort tuần tự



Q: Counting sort có stable không?

A: Có!

Counting sort tuần tự

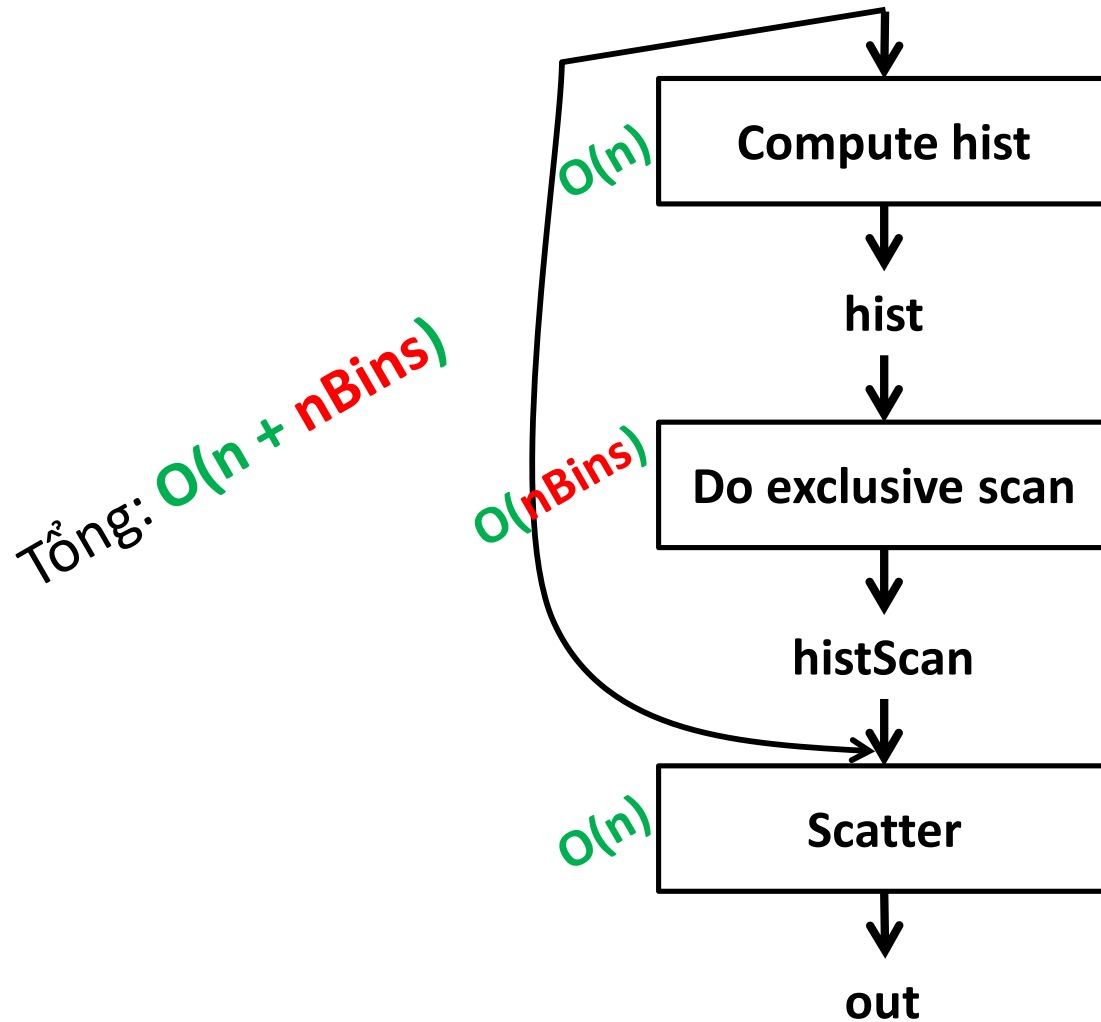


Q: Nếu cài đặt một cách đơn giản thì chi phí tính toán là bao nhiêu?

A: $O(n^2)$

Counting sort tuần tự: làm sao để giảm chi phí?

in (mảng số nguyên không dấu)

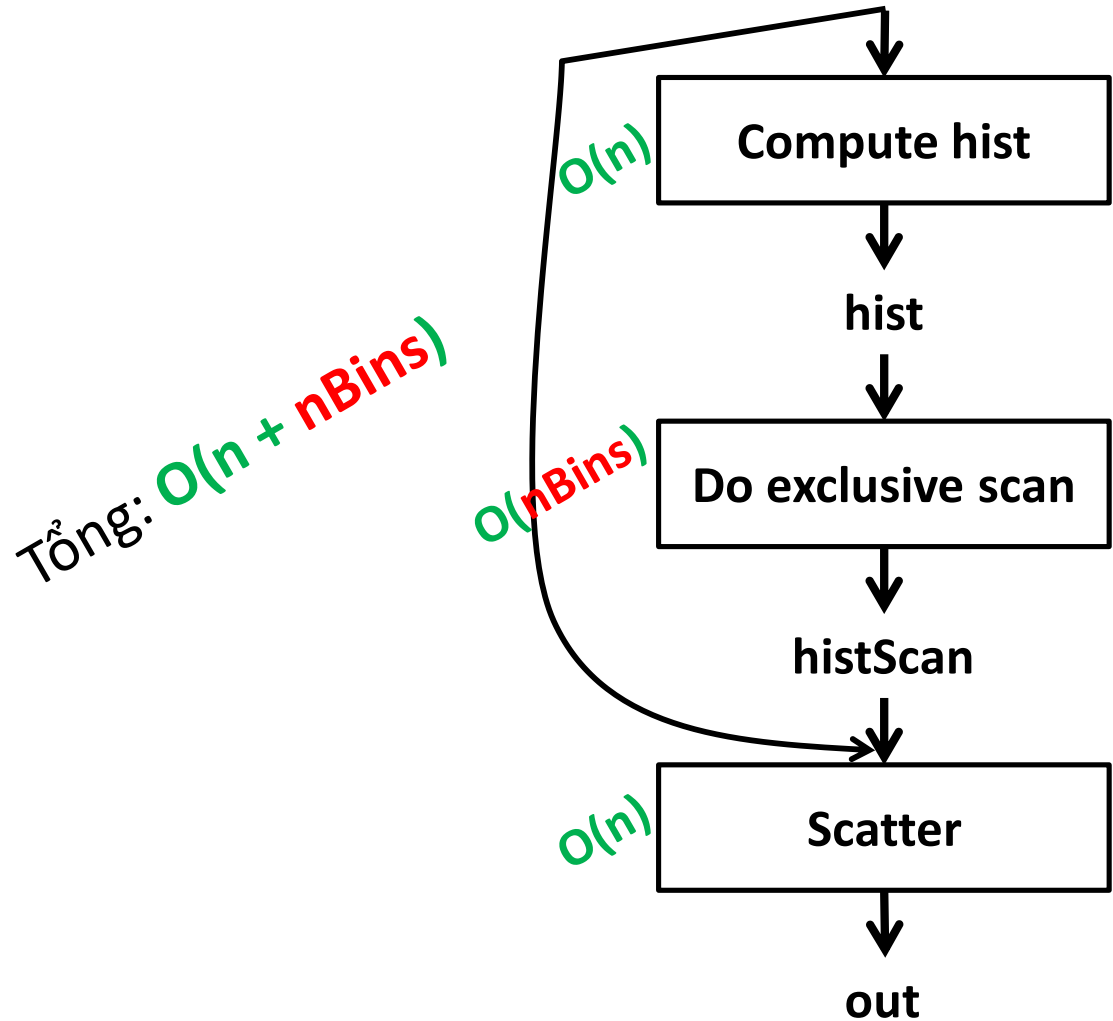


```
// Scatter
for (int i = 0; i < n; i++)
{
    int bin = in[i];
    out[histScan[bin]] = in[i];
    histScan[bin]++;
}
```

Q: Khi nào thì counting sort chạy hiệu quả?

A: Khi **nBins** nhỏ

in (mảng số nguyên không dấu)

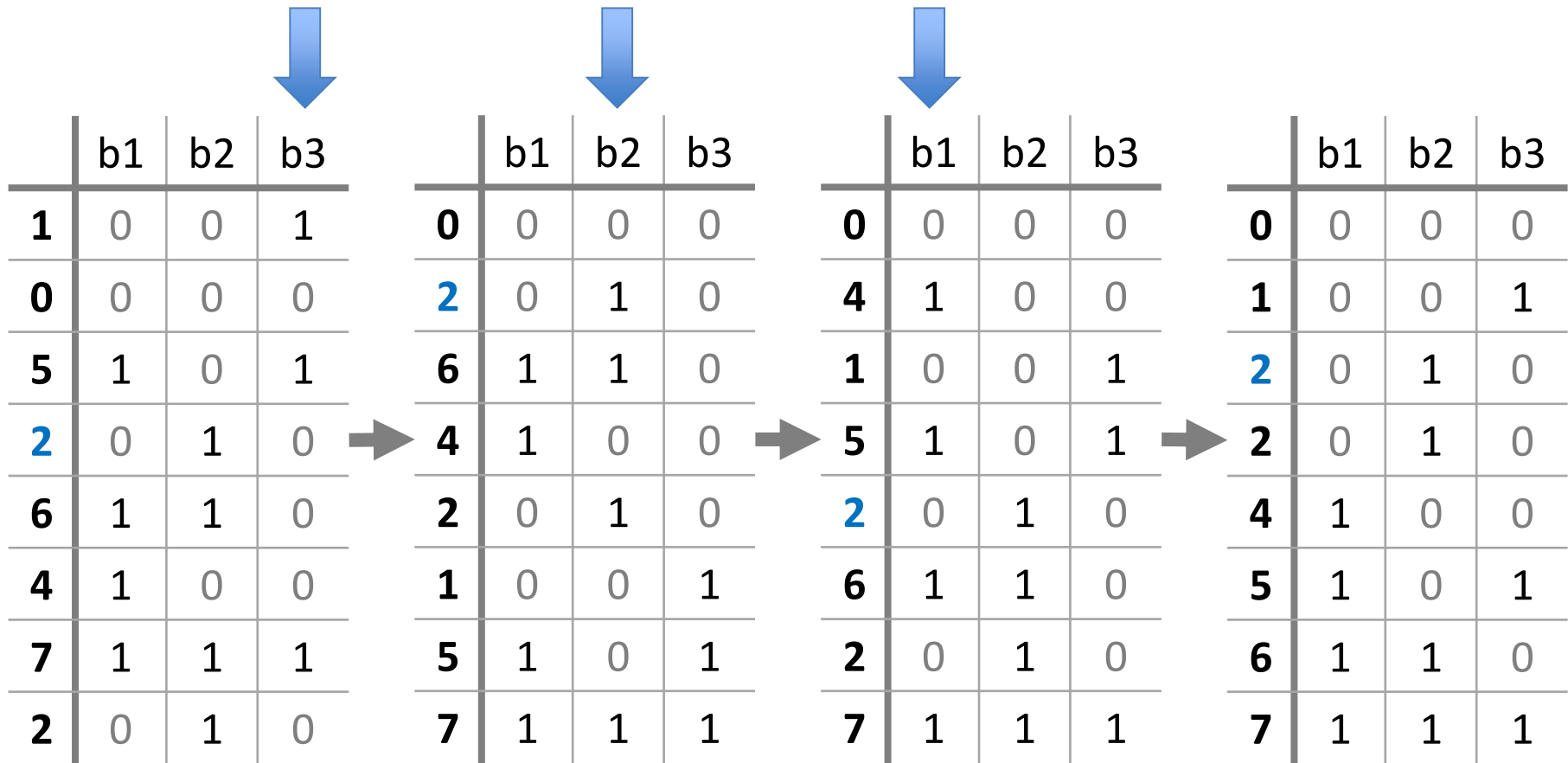


```
// Scatter
for (int i = 0; i < n; i++)
{
    int bin = in[i];
    out[histScan[bin]] = in[i];
    histScan[bin]++;
}
```

Radix sort (tuần tự): làm cho counting sort vẫn chạy hiệu quả khi nBins lớn

Duyệt từ bit b3 (least significant bit) đến b1 (most significant bit)

Với mỗi bit, sort các phần tử theo bit này bằng Counting Sort (tổng quát hơn là một thuật toán sort mà **stable**)



XONG!

Radix sort tuần tự

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm **k** bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

Demo cài đặt radix sort tuần tự

Nội dung

- ☐ Bài toán sắp xếp
- ☐ Radix sort tuần tự
- ☐ Radix sort song song

Radix sort: song song hóa?

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm **k** bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

=====

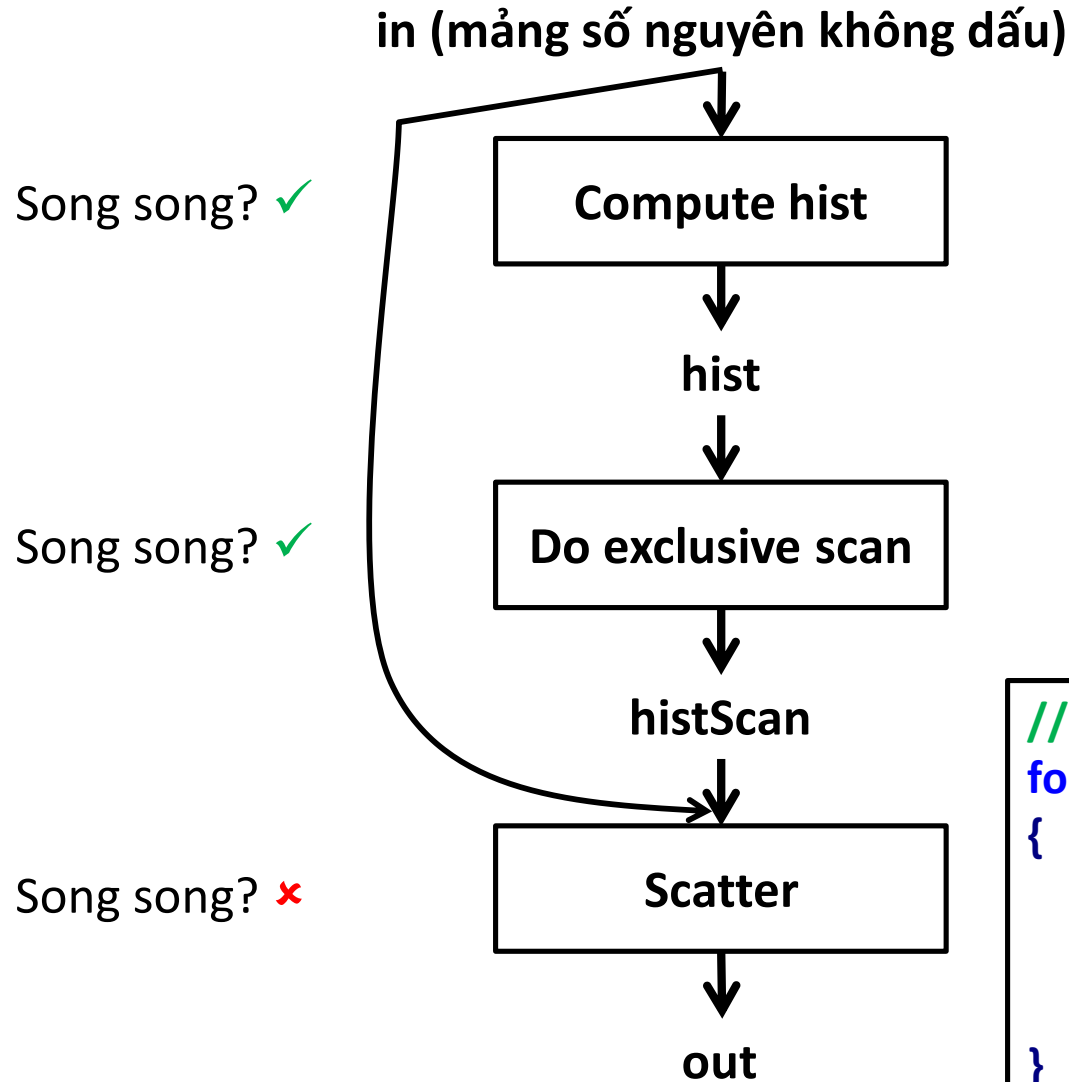
Q: Song song hóa phần duyệt các digit?

A: Khó 😞

Q: Song song hóa phần counting sort?

A: Có hy vọng 😊

Counting sort: song song hóa?



```
// Scatter
for (int i = 0; i < n; i++)
{
    int bin = in[i];
    out[histScan[bin]] = in[i];
    histScan[bin]++;
}
```

Counting sort: song song hóa?

- Có thể cài đặt song song một cách hiệu quả trong trường hợp mảng **in** là mảng nhị phân
- Vd, cho mảng **in** gồm **n** phần tử: 0, 1, 1, 0, 1
 - ▣ Thực hiện exclusive scan được mảng **inScan**:
0, 0, 1, 2, 2
inScan[i] cho biết số lượng số 1 trong mảng **in** mà ở phía trái **in[i]**
 - ▣ Tính số lượng số 0 trong mảng **in**:
$$\text{nZeros} = n - \text{inScan}[n-1] - \text{in}[n-1]$$
 - ▣ Tính **rank**:
 - Nếu **in[i]** là 0 thì **rank** = **i** - ?
 - Nếu **in[i]** là 1 thì **rank** = ?

Counting sort: song song hóa?

- Có thể cài đặt song song một cách hiệu quả trong trường hợp mảng **in** là mảng nhị phân
- Vd, cho mảng **in** gồm **n** phần tử: 0, 1, 1, 0, 1
 - ▣ Thực hiện exclusive scan được mảng **inScan**:
0, 0, 1, 2, 2
inScan[i] cho biết số lượng số 1 trong mảng **in** mà ở phía trái **in[i]**
 - ▣ Tính số lượng số 0 trong mảng **in**:
$$\mathbf{nZeros} = \mathbf{n} - \mathbf{inScan}[\mathbf{n}-1] - \mathbf{in}[\mathbf{n}-1]$$
 - ▣ Tính **rank**:
 - Nếu **in[i]** là 0 thì $\mathbf{rank} = \mathbf{i} - \mathbf{inScan}[\mathbf{i}]$
 - Nếu **in[i]** là 1 thì $\mathbf{rank} = \mathbf{nZeros} + \mathbf{inScan}[\mathbf{i}]$
 - ▣ Thực hiện scatter: $\mathbf{out}[\mathbf{rank}] = \mathbf{in}[\mathbf{i}]$

Radix sort: song song hóa?

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm **k** bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

Với $k = 1$, ta có thể cài đặt song song counting sort một cách hiệu quả 😊
nhưng sẽ cần nhiều lần chạy counting sort 😞

Radix sort

Ý tưởng cài đặt song song với $k > 1$

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm k bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

Ý tưởng:

Thay vì cố gắng cài đặt song song các bước của thuật toán Counting Sort tuần tự (**bước cuối khó cài đặt song song hiệu quả**), ta sẽ thiết kế lại các bước của thuật toán Counting Sort để dễ cài đặt song song hơn

Radix sort

Ý tưởng cài đặt song song với $k > 1$ (level 1)

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm k bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

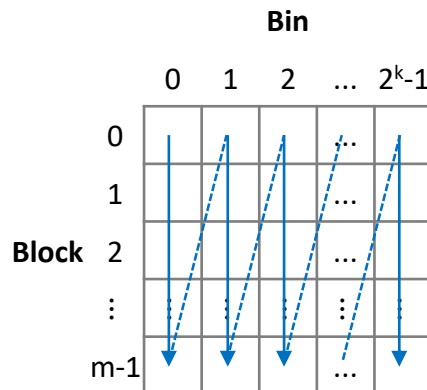
- Mỗi block tính local histogram của digit-đang-xét trên phần dữ liệu của mình
- Với mảng 2 chiều mà mỗi dòng là local hist của một block, thực hiện exclusive scan trên mảng một chiều gồm các cột nối lại với nhau

Kết quả scan ở dòng r và cột c cho biết:

số lượng phần tử có digit-đang-xét nhỏ hơn c

+ số lượng phần tử có digit-đang-xét bằng c trong các block trước block r

rank?



- Mỗi block thực hiện scatter phần dữ liệu của mình xuống mảng output dựa vào kết quả scan ở trên

Radix sort

Ý tưởng cài đặt song song với $k > 1$ (level 1)

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm k bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

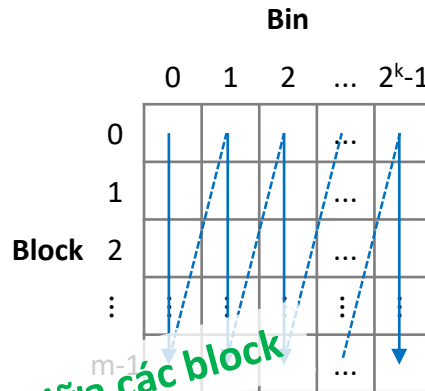
Song song? ✓ Mỗi block tính local histogram của digit-đang-xét trên phần dữ liệu của mình

Song song? ✓ Với mảng 2 chiều mà mỗi dòng là local hist của một block, thực hiện exclusive scan trên mảng một chiều gồm các cột nối lại với nhau

Kết quả scan ở dòng r và cột c cho biết:

số lượng phần tử có digit-đang-xét nhỏ hơn c

+ số lượng phần tử có digit-đang-xét bằng c trong các block trước block r



Song song? ✓ Mỗi block thực hiện scatter phần dữ liệu của mình xuống mảng output dựa vào kết quả scan **Ghi GMEM hiệu quả?** ✗

ít nhất là có thể song song giữa các block

Radix sort

Ý tưởng cài đặt song song với $k > 1$ (level 2)

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm k bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

- Mỗi block tính local histogram của digit-đang-xét trên phần dữ liệu của mình
- Với mảng 2 chiều mà mỗi dòng là local hist của một block, thực hiện exclusive scan trên mảng một chiều gồm các cột nối lại với nhau
- Mỗi block thực hiện scatter phần dữ liệu của mình xuống mảng output dựa vào kết quả scan ở trên
 - Mỗi block sắp xếp cục bộ phần dữ liệu của mình theo digit đang xét (dùng Radix Sort với $k=1$ bit và làm trên SMEM)
 - Mỗi block tính chỉ số bắt đầu (xét cục bộ trong block) của mỗi giá trị digit
 - Mỗi thread trong block tính số lượng phần tử đứng trước mình trong block có digit-đang-xét bằng digit-đang-xét của phần tử mà mình phụ trách
 - Mỗi thread trong block tính rank và thực hiện scatter

Radix sort

Ý tưởng cài đặt song song với $k > 1$ (level 2)

Duyệt từ “least significant digit” đến “most significant digit” (mỗi digit gồm k bit):

Sắp xếp các phần tử theo digit đang xét bằng một thuật toán sắp xếp **stable** như counting sort

Song song? ✓ Mỗi block tính local histogram của digit-đang-xét trên phần dữ liệu của mình

Song song? ✓ Với mảng 2 chiều mà mỗi dòng là local hist của một block, thực hiện exclusive scan trên mảng một chiều gồm các cột nối lại với nhau

Song song? ✓ Mỗi block thực hiện scatter phần dữ liệu của mình xuống mảng output dựa vào kết quả scan ở trên

- Mỗi block sắp xếp cục bộ phần dữ liệu của mình theo digit đang xét (dùng Radix Sort với $k=1$ bit và làm trên SMEM)
- Mỗi block tính chỉ số bắt đầu (xét cục bộ trong block) của mỗi giá trị digit
- Mỗi thread trong block tính số lượng phần tử đứng trước mình trong block có digit-đang-xét bằng digit-đang-xét của phần tử mà mình phụ trách
- Mỗi thread trong block tính rank và thực hiện scatter

Ghi GMEM hiệu quả? ✓

Radix sort với số nguyên có dấu

- Bit dấu là bit MSB (Most Significant Bit)
 - MSB = 0: số dương
 - Số nguyên có dấu = số nguyên không dấu
 - MSB = 1: số âm
 - Số nguyên có dấu = số nguyên không dấu
 - $2^{\text{Số-bit-biểu-diễn-số-nguyên}}$
- Nếu giữ nguyên thuật toán radix sort ban đầu thì sẽ chạy sai
- Một giải pháp là chuyển từ số nguyên có dấu sang số nguyên không dấu, chạy radix sort, rồi chuyển kết quả ngược lại số nguyên có dấu

Radix sort với số thực






- ☐ Cần hiểu về cách biểu diễn bit của số thực
- ☐ Ý tưởng tương tự như số nguyên có dấu: chuyển từ số thực sang số nguyên không dấu, chạy radix sort, rồi chuyển kết quả ngược lại số thực

Nhìn lại các nội dung đã học


Nội dung môn học:

- ☒ Giới thiệu CUDA
- ☒ Các dạng tính toán song song thường gặp **Reduction, Convolution, Scan, Histogram, Sort**
- ☒ Cách thực thi song song trong CUDA
- ☒ Các loại bộ nhớ trong CUDA
- ☐ Quy trình tối ưu hóa chương trình CUDA
- ☐ Các chủ đề mở rộng (nếu có thời gian)

Sau khi học xong môn học này, SV có thể:

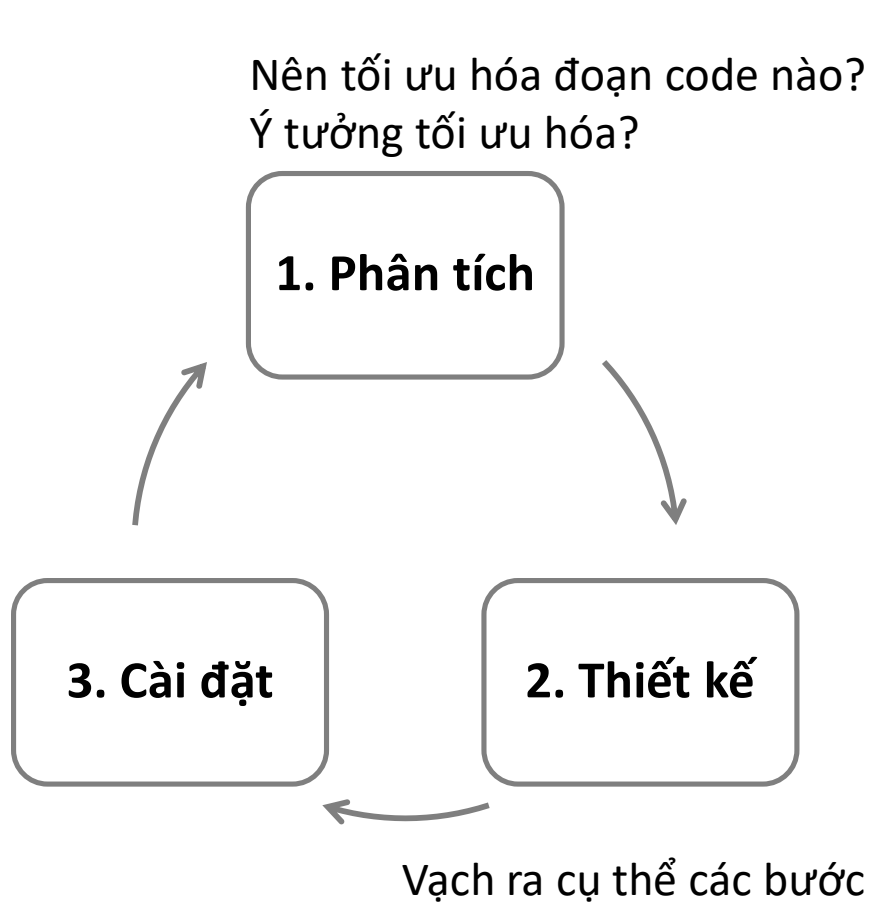
-  Cài đặt được chương trình chạy song song trên GPU bằng CUDA
-  Vận dụng được cách thực thi song song trong CUDA để tăng tốc chương trình
-  Vận dụng được các loại bộ nhớ trong CUDA để tăng tốc chương trình
-  Vận dụng được quy trình tối ưu hóa chương trình CUDA
-  Vận dụng được kỹ năng làm việc nhóm để hoàn thành các bài tập nhóm trong môn học

Đồ án cuối kỳ (50% điểm môn học)

- ☐ **Số lượng SV / nhóm:** 2
- ☐ **Hình thức:** thuyết trình và vấn đáp
- ☐ **Địa điểm:** I81; **thời gian:** ?
- ☐ **Nội dung đồ án:** tối ưu hóa Radix Sort trên GPU (dựa trên ý tưởng trong bài báo của NVIDIA¹ mà mình đã giảng)
 - ☐ Điểm bắt đầu: cài đặt tuần tự trên host
 - ☐ Tối ưu hóa nhiều lần 
 - ☐ Đích đến: cài đặt song song trên device và thời gian chạy tương đương với thuật toán sort của thư viện Thrust
- ☐ **Yêu cầu:**
 - ☐ Tất cả các thành viên trong nhóm đều phải hiểu về đồ án mà nhóm mình làm (tất nhiên là phải hiểu cả code)
 - Phải làm việc nhóm như thế nào để đạt được yêu cầu này?
 - ☐ Nộp: file làm việc nhóm (viết file này khi nào?), slide báo cáo quá trình tối ưu hóa, các phiên bản của file code

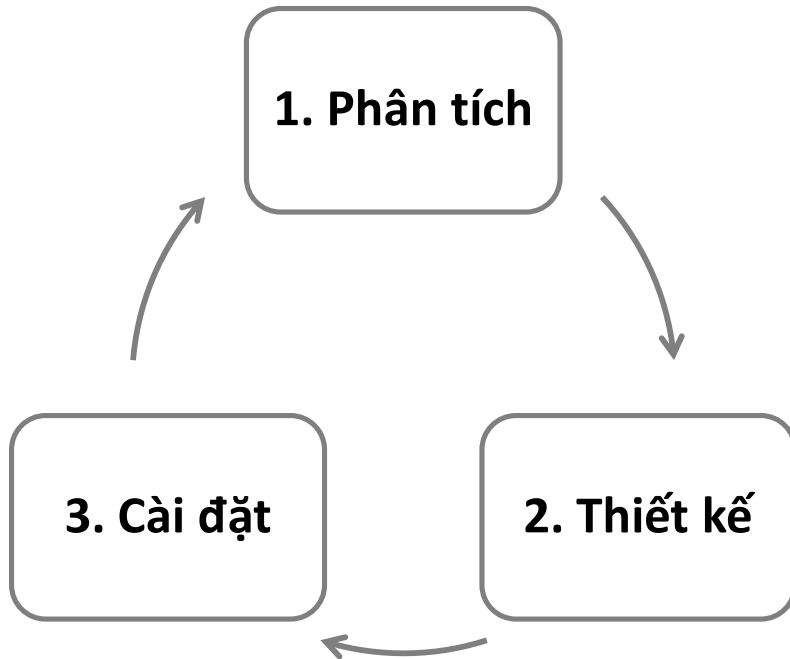
¹Satish et al. Designing efficient sorting algorithms for manycore GPUs. 2009

Quy trình tối ưu hóa chương trình CUDA



- Mỗi vòng lặp sẽ tạo ra một phiên bản cài đặt mới dựa trên các phiên bản trước đó
- Đi từ cài đặt tuần tự đi lên, đi từ đơn giản đi lên

Quy trình tối ưu hóa chương trình CUDA



Làm sao để đi tốt quy trình này?

Một số ý:

- **Giữ tâm tĩnh**
- Giữ code rõ ràng, dễ đọc
- Code nhanh hay code chậm?
- Dùng một editor tốt và học cách để sử dụng hiệu quả

Định hướng tối ưu hóa nói chung

- Đưa đủ số lượng công việc độc lập để tận dụng hết khả năng song song của phần cứng GPU
 - ▣ Đưa đủ số block để tận dụng hết các SM
 - ▣ Trong SM, đưa đủ số lượng câu lệnh độc lập (có thể đến từ trong cùng một warp, hoặc từ các warp khác nhau) để tận dụng các execution pipeline, che độ trễ
- Truy xuất DRAM hiệu quả
 - ▣ Tránh để warp truy xuất đến các địa chỉ nằm “rải rác” trong DRAM
 - ▣ Đưa đủ số lượng câu lệnh truy xuất DRAM độc lập để giữ cho bus luôn “đầy”
 - ▣ Dùng SMEM để giảm số lần truy xuất DRAM, cũng như để truy xuất DRAM hiệu quả
- Giảm thiểu phân kỳ warp

Thư viện Thrust

Xem các file đính kèm:

- ❑ “thrust.cu”: minh họa cách dùng các cấu trúc dữ liệu và thuật toán của Thrust
- ❑ “thrust_bt04_p3.cu”, hàm “sortByDevice”: minh họa cách kết nối giữa các cấu trúc dữ liệu thông thường (vd, con trỏ tới mảng) của chương trình CUDA với Thrust

