**Lập trình song song trên GPU**

**Hoàng Minh Thanh (18424062)**



**BT4 : Tính tổng tích lũy**

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

Bộ môn Công nghệ phần mềm

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên TP HCM

Contents

[I. Quá trình cài đặt 3](#_Toc45824445)

[1) Hàm hệ qui reduce trên CPU 3](#_Toc45824446)

[2) Hàm reduceNeighbored : 5](#_Toc45824447)

[3) Hàm reduceNeighboredLess: 7](#_Toc45824448)

[4) Hàm reduceInterleaved: 9](#_Toc45824449)

[UNROLLING LOOPS : 10](#_Toc45824450)

[5) Hàm reduceUnrolling2: 11](#_Toc45824451)

[6) Hàm reduceUnrolling4: 12](#_Toc45824452)

[7) Hàm reduceUnrolling8: 13](#_Toc45824453)

[Unrolled Warps 14](#_Toc45824454)

[8) Hàm reduceUnrollWarps8: 15](#_Toc45824455)

[9) Hàm reduceCompleteUnrollWarsp8: 16](#_Toc45824456)

[10) Hàm reduceCompleteUnroll: 17](#_Toc45824457)

[II. Báo cáo và rút ra nhận xét, kết luận 22](#_Toc45824458)

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

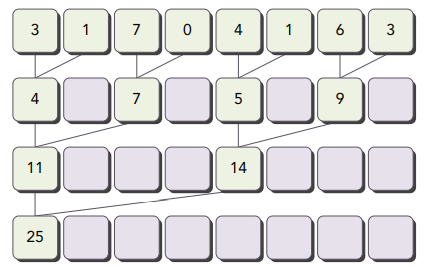
Sum on GPU : 8389084.624453

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

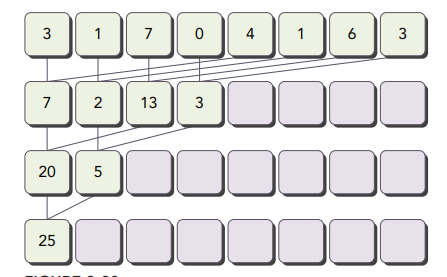
## 1) Hàm hệ qui reduce trên CPU

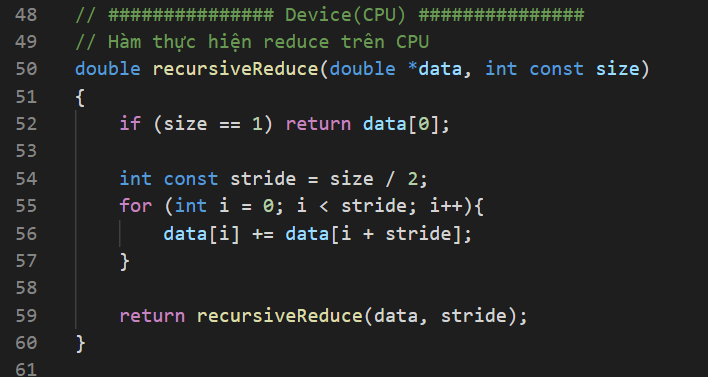
Trên CPU ta cài đặt đệ qui để tính toán

Phương pháp tính tổng mảng bằng hàng xóm kế cận



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

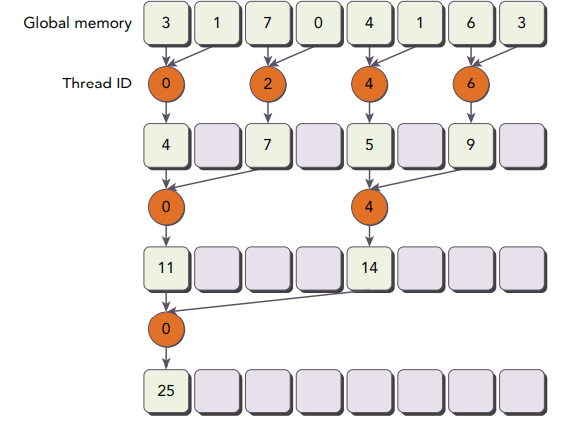




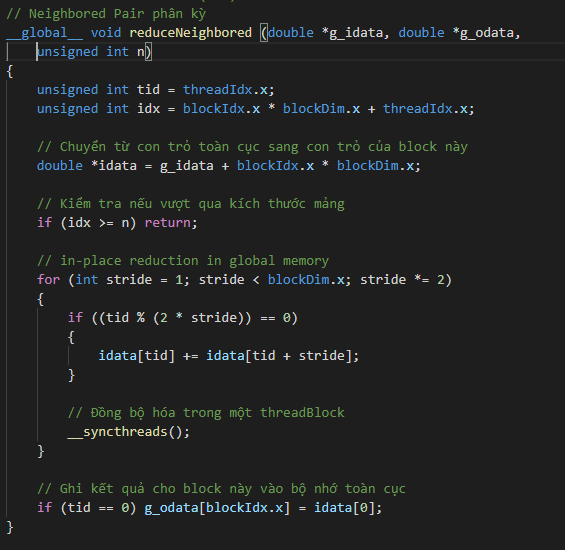
## 2) Hàm reduceNeighbored :

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

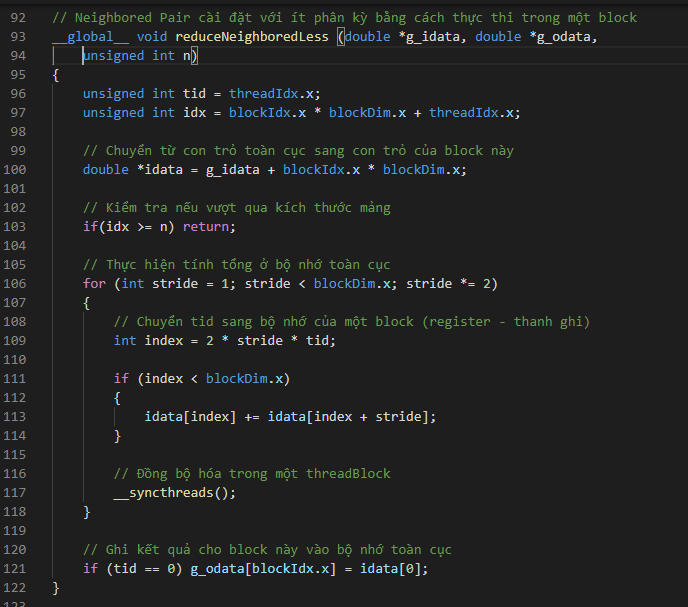


Cài đặt :

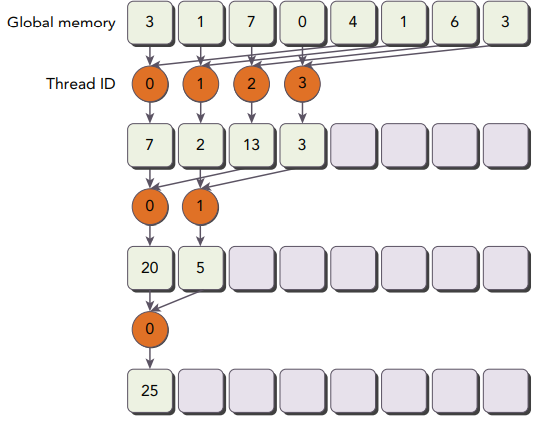


## 3) Hàm reduceNeighboredLess:

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)

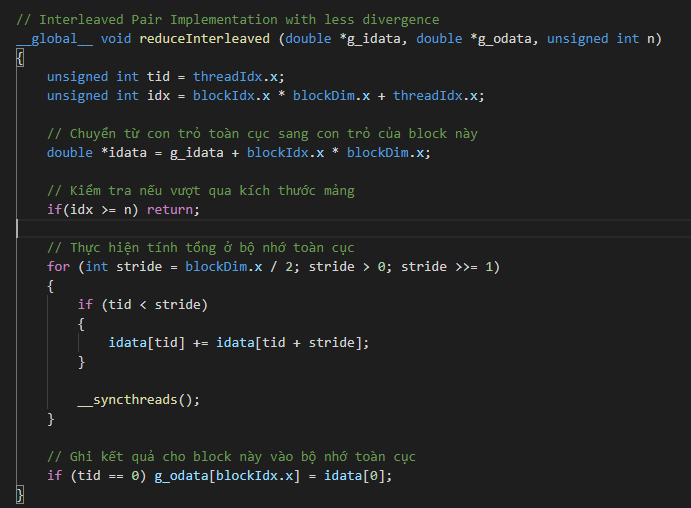


## 4) Hàm reduceInterleaved:



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)

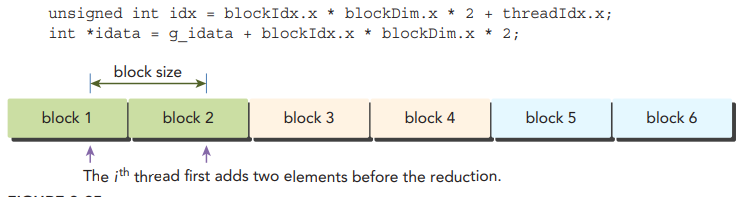
Cài đặt :



## 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)

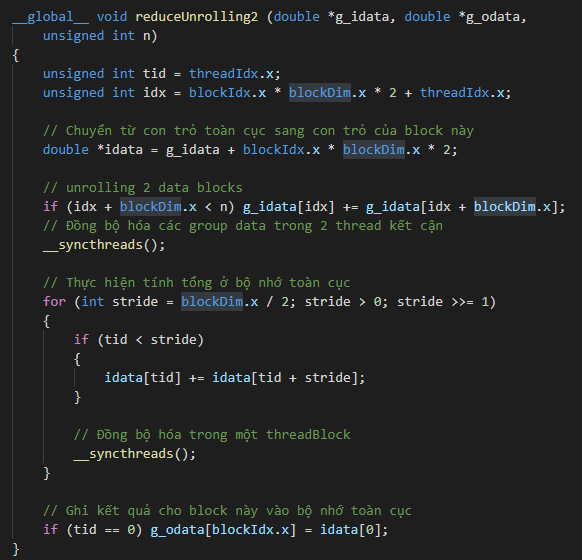
## 5) Hàm reduceUnrolling2:



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**

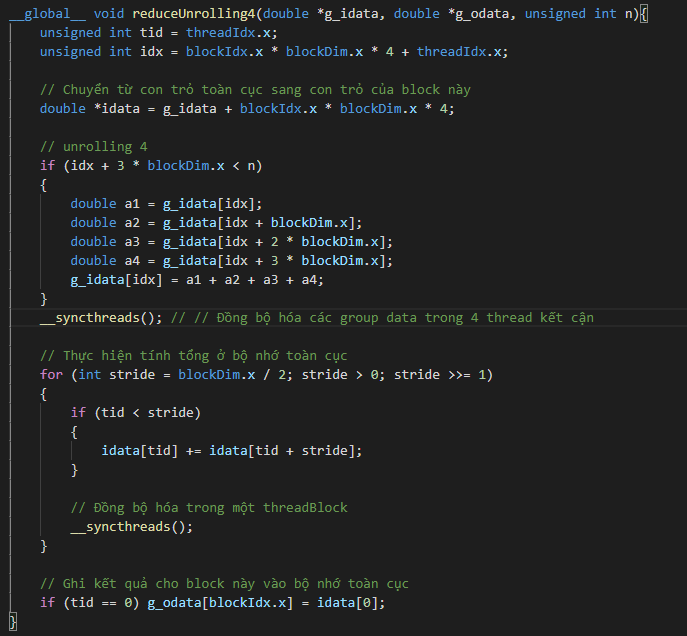
Cài đặt :



## 6) Hàm reduceUnrolling4:

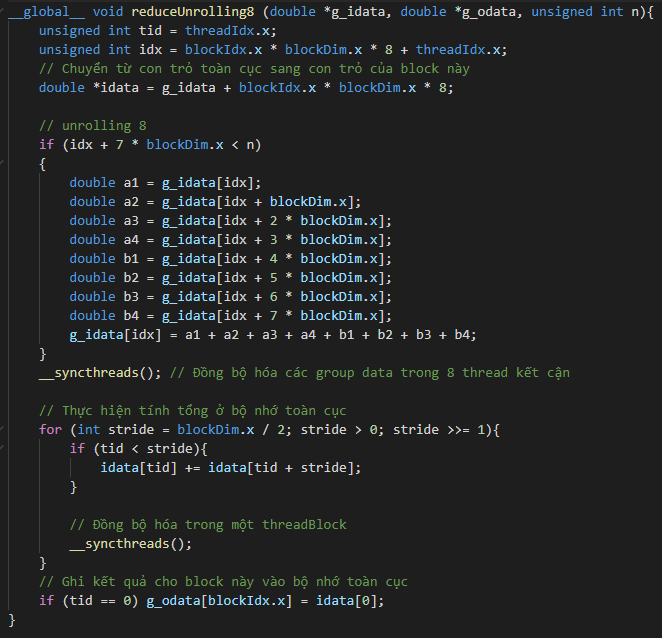
Tương tự như reduceUnrolling2; nhưng vì có 4 biến cần tính tổng nên ta khai báo các biến riêng biệt để lưu trữ trên bộ nhớ register của mỗi block

Cài đặt



## 7) Hàm reduceUnrolling8:

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



## 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)

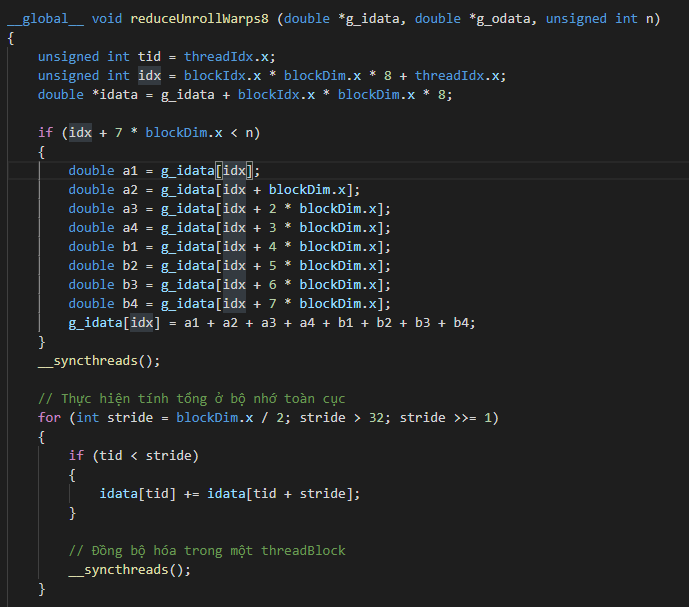
Từ khóa \_\_ syncthreads sẽ giúp đồng bộ hóa các thread trong một block, để đảm bảo với mỗi vòng tính tổng thì kết quả đã được ghi vào bộ nhớ toàn cục hoàn tất và không có thread nào đang còn ghi để tiếp tục vòng tiếp theo. Phương pháp này sẽ giúp unroll casc thread trong cùng một wrap (32 thread)

**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.

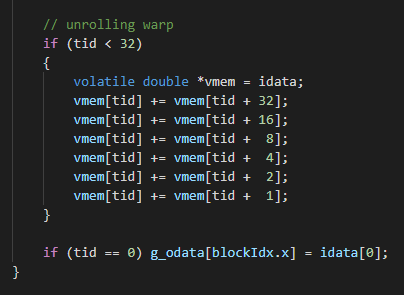
## 8) Hàm reduceUnrollWarps8:

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.

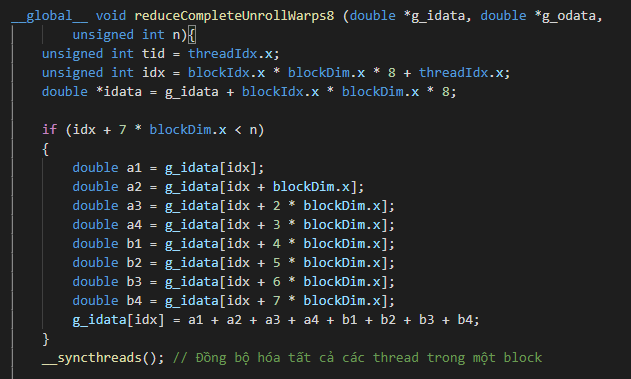
Tham khảo tại : <https://ktmt.github.io/blog/2013/05/09/y-nghia-cua-tu-khoa-volatile-trong-c/>

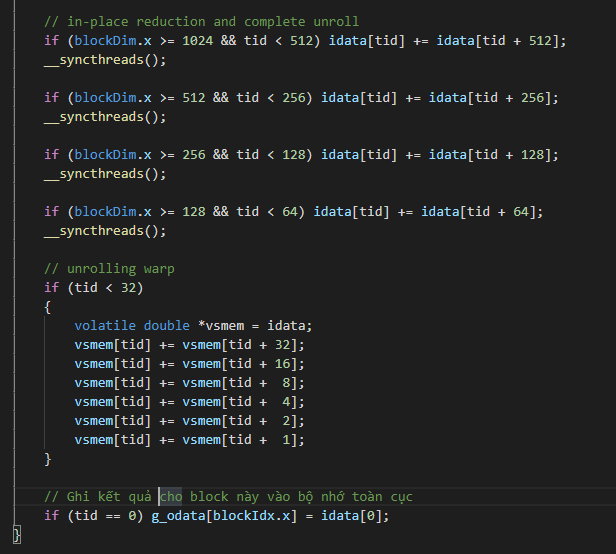
Cài đặt : 

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



## 9) Hàm reduceCompleteUnrollWarsp8:

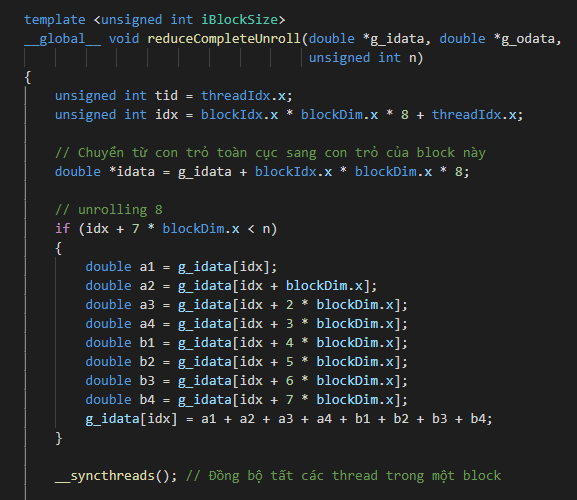


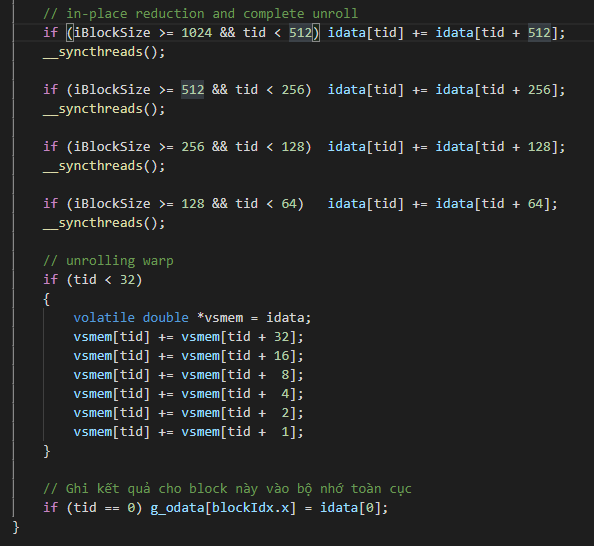


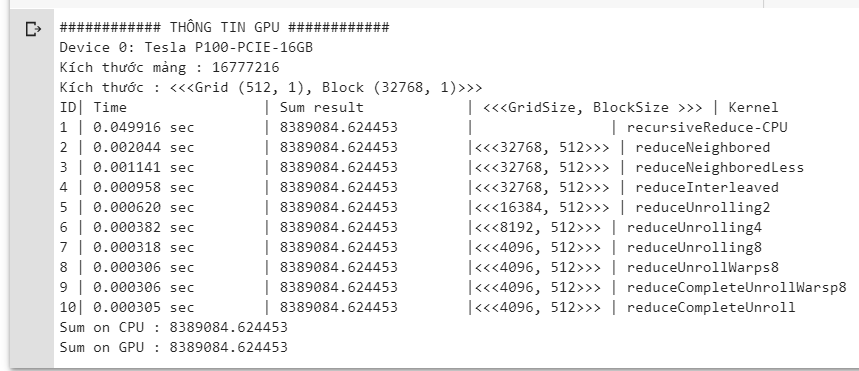
Tương tự như hàm trên nhưng ta unroll toàn bộ kích thước tối đa của một block

## 10) Hàm reduceCompleteUnroll:

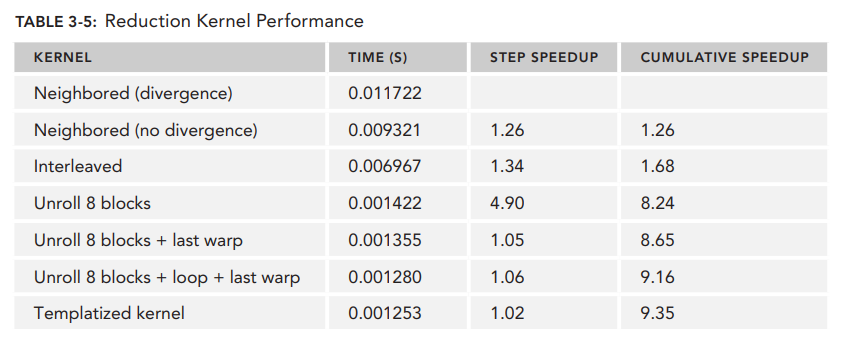
Phương pháp này sử dụng một kỹ thuật **template** của C++ để tối ưu; nghĩa là nó tạo ra một bản sao nhiều cái của cùng một hàm với mỗi iBlockSize khác nhau. Khi đó ta chương trình sẽ không phải so sánh kích thước của blockSize mà chỉ gọi hàm với mỗi blockSize khác nhau



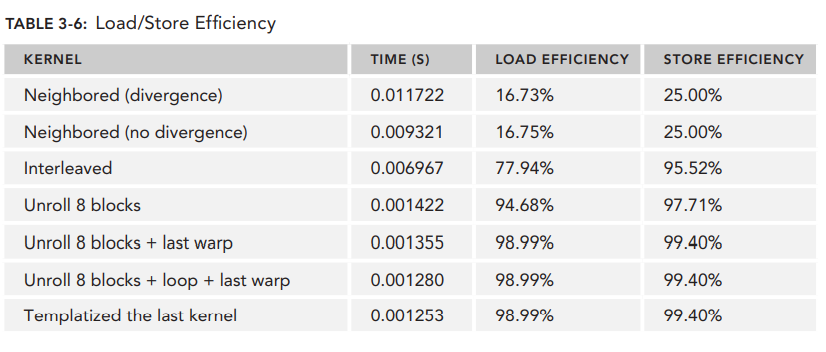


Kết quả thực thi

Tỷ lệ tối ưu về hiệu suất tính toán



Kết quả tỉ lệ tối ưu của lưu trữ



# 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)