

## IMT2112 - Algoritmos Paralelos en Computación Científica

Ordenar en paralelo

Elwin van 't Wout

3 de diciembre de 2019



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

Facultad de Matemáticas • Escuela de Ingeniería

imc.uc.cl

# Clase previa

- Programar con OpenCL en tarjetas gráficas

# Reducción en OpenCL

- En OpenCL hay distintos niveles de datos
  - *private*: datos por cada hilo
  - *local*: datos por cada grupo de trabajo
  - *global*: datos por cada *kernel queue*
- Es posible hacer operaciones de reducción en cada nivel, con distintas características de desempeño
  - dentro de los hilos: eficiente, especialmente si se puede usar instrucciones como  $a[i] += f(x)$
  - entre hilos de un grupo de trabajo: desafíos con el SIMD
  - entre distintos grupos de trabajos: ineficiente, mejor enviar resultados parciales por grupo al *host*

# Agenda

- ¿Como ordenar un areglo en paralelo?

# Parallel prefix

Capítulo 19 del libro de Eijkhout

# Suma cumulativa

- Calcular la suma de un vector se puede parallelizar con un árbol binario
  - la suma es un ejemplo de una operación de reducción
- Calcular la suma cumulativa en paralelo es un mayor desafío
  - la ejecución en serie no requiere mas operaciones que para la suma
  - la suma cumulativa es un ejemplo de una operación de ‘prefijo’ (*prefix*)
- Ejemplo:

$$\mathbf{x} = [4, 2, 1, 3, 1]$$

$$s(\mathbf{x}) = [4, 6, 7, 10, 11]$$

# Operaciones de prefijo

- Para una operación asociativa  $y(x_1, x_2)$ , una ‘operación de prefijo’ está definida como

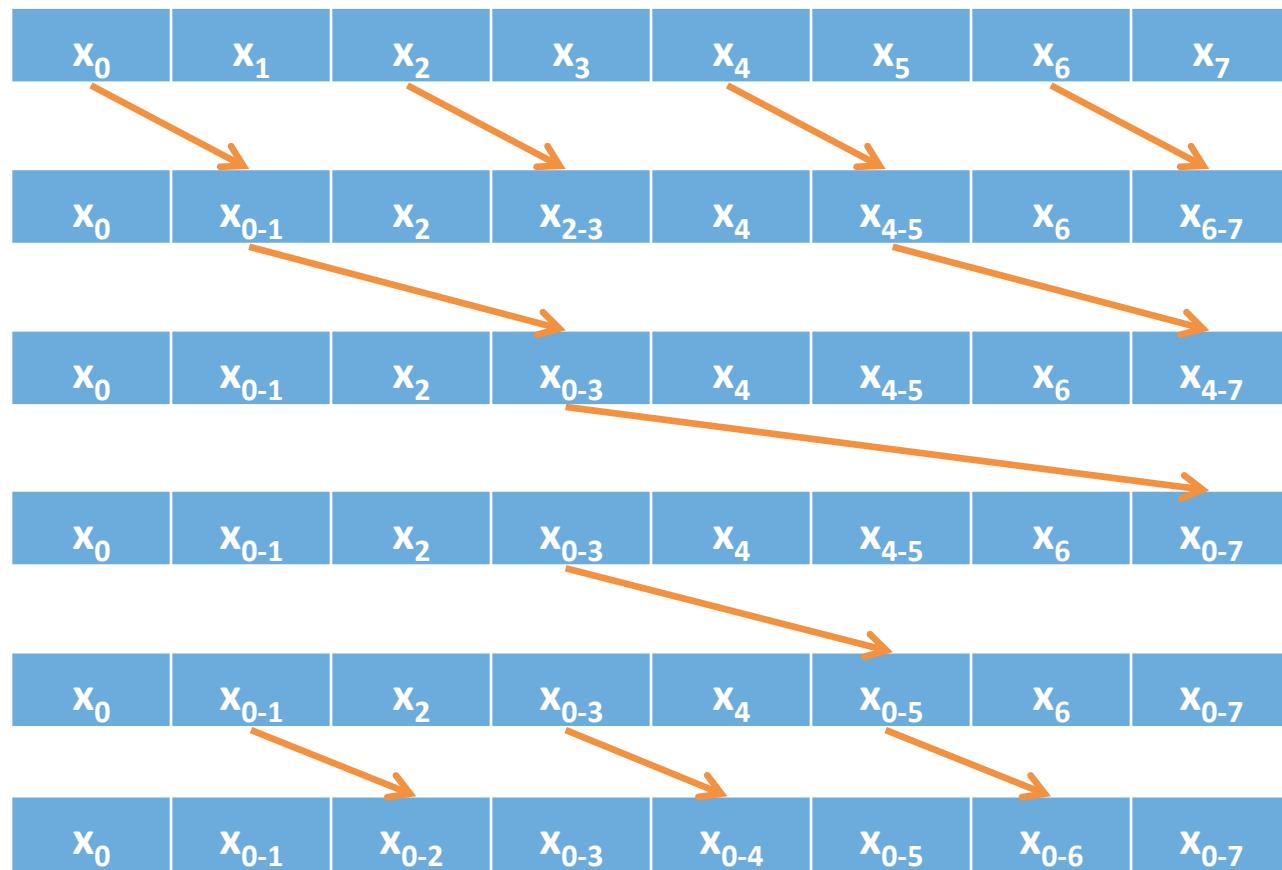
$$Y(x_1, x_2, \dots, x_n) = \begin{cases} y(x_1, x_2), & n = 2 \\ y(Y(x_1, x_2, \dots, x_{n-1}), x_n), & n > 2. \end{cases}$$

- Es una operación recursiva y, por lo tanto, desafiante de paralelizar
- La operación base podría ser una suma, producto, máximo, etc.
  - también sirve para condiciones como por ejemplo el número de elementos menor a un valor dado

# Operaciones de *prefix* en paralelo

- Recuerde que se puede parallelizar una suma con un árbol binario
  - la suma en serie tiene complejidad de  $O(n)$
  - la suma en paralelo tiene complejidad de  $O(n/p \log_2(n))$
- Usar la suma en paralelo para cada elemento de la suma cumulativa es ineficiente
- La idea es utilizar los resultados parciales de la suma en paralelo

# Operaciones de *prefix* en paralelo



# Operaciones de *prefix* en paralelo

- Se puede implementar una operación de *prefix* como dos árboles binarios
  - un árbol hacia adelante
  - otro árbol hacia atrás
- Tiene una complejidad computacional de  $O(2 n/p \log_2(n))$
- Una operación de *prefix* también se llama una operación de *scan*
  - el *exclusive scan* suma los elementos 0 ... n-1
- MPI tiene la funcionalidad disponible: **`MPI_Scan()`**

# Parallel sorting

Capítulo 8 del libro de Eijkhout

# Ordenar datos

- Ordenar los elementos de un vector
- Ejemplo:
$$\mathbf{x} = [4, 2, 1, 3, 1]$$
$$\mathbf{s(x)} = [1, 1, 2, 3, 4]$$
- Los algoritmos *brute force* tiene complejidad  $O(n^2)$ 
  - con poco oportunidad de paralelizar
- Algoritmos eficientes tienen complejidad  $O(n \log(n))$

# Ordenar datos con Quicksort

- Input: un arreglo de  $n$  elementos
- Output: un arreglo con los números ordenados
- Algoritmo de quicksort:
  - Comienza con el arreglo entero
  - Mientras que el arreglo es más larga que un único elemento:
    1. elegir un valor arbitrario como pivote
    2. ordenar la matriz en regiones rojas, blancas y azules
      - a. los elementos rojos son más grandes que el pivote
      - b. los elementos blancos son iguales al pivote
      - c. los elementos azules son más pequeños que el pivote
    3. realizar quicksort sobre los arreglos rojos y azules

# Ordenar datos con Quicksort

- Ejemplo: ordenar [6 1 7 4 0 3 5 2] con un pivote de 4
  1. etiquetar elementos: [r a r b a a r a]
  2. prefijos paralelos: azul [0 0 1 1 1 2 3 3], blanco [0 0 0 0 1 1 1], rojo [0 1 1 2 2 2 2 3]
  3. sumas: [4 1 3]
  4. desplazamientos: [0 4 5]
  5. índices: [5 0 6 4 1 2 7 3]
  6. arreglo azul-blanco-rojo: [1 0 3 2 4 6 7 5]
  7. realizar quicksort en [1 0 3 2] y [6 7 5]

# Quicksort en paralelo

- La eficiencia paralela de quicksort
  1. etiquetar elementos:  $O(n/p)$
  2. prefijos paralelos (*exclusive scan*):  $O(n/p \log(n))$
  3. calcular suma y desplazamiento:  $O(1)$ 
    - usando los resultados del scan
  4. areglo de índices:  $O(n/p)$
  5. crear areglos particionados:  $O(n/p)$ 
    - no cálculo, sino comunicación
- En total, cada iteración de quicksort tiene complejidad de  $O(n/p \log(n))$

# Quicksort en paralelo

- El número de iteraciones de quicksort depende de la elección del pivote
- Versión afortunada: el pivote es siempre la mediana
  - en cada iteración el arreglo siempre se divide en la mitad
  - la complejidad es  $O(\log_2(n))$
- Versión desafortunada: el pivote es siempre el mínimo o el máximo
  - el arreglo nunca se divide y siempre se reduce en uno
  - la complejidad es  $O(n)$

# Quicksort en paralelo

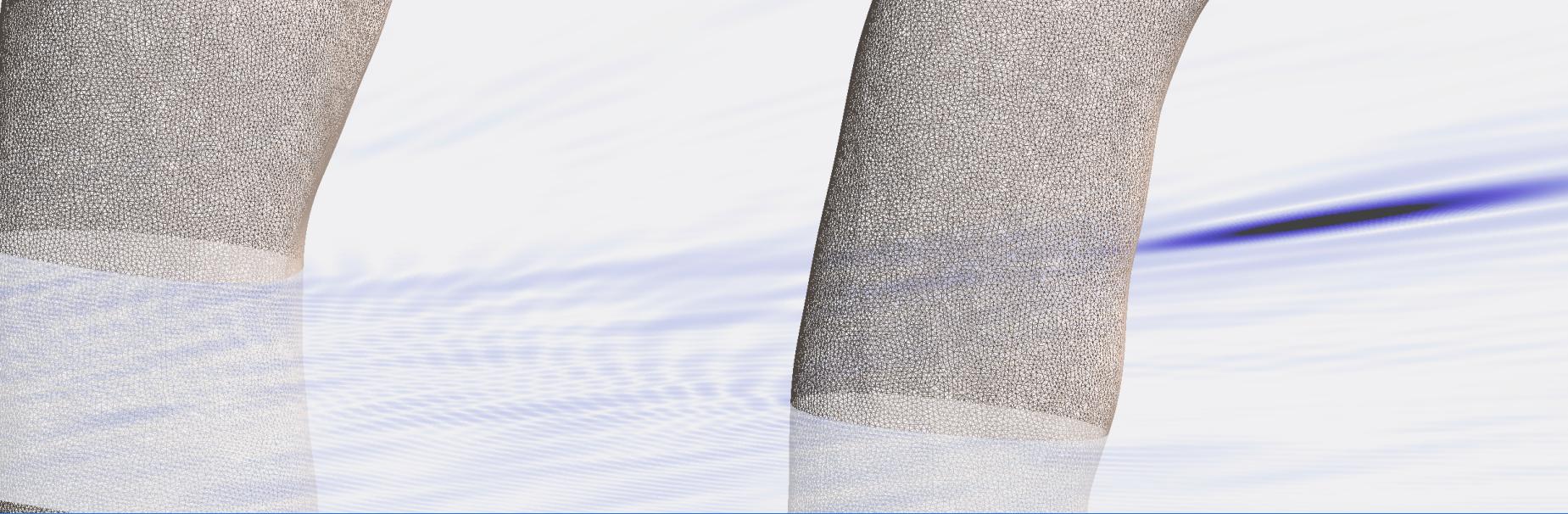
- En la práctica, tomar el primer elemento del arreglo como pivote es una buena opción
- El número de iteraciones *esperado* es de  $O(\log_2(n))$

# Quicksort en paralelo

- En el caso extremo de  $p=n$ , cada procesador tiene un elemento del arreglo
- La complejidad computacional *garantizada* de quicksort es de  $O(n \log_2(n))$
- La complejidad computacional *esperada* de quicksort es de  $O((\log_2(n))^2)$

# Resumen

- Parallel prefix
- Parallel sorting



## IMT2112 - Algoritmos Paralelos en Computación Científica

Ordenar en paralelo

Elwin van 't Wout

3 de diciembre de 2019



PONTIFICIA  
UNIVERSIDAD  
CATÓLICA  
DE CHILE

Facultad de Matemáticas • Escuela de Ingeniería

imc.uc.cl