Heterogeneous
Parallel
Programming

Lecture 4.4

Parallel Computation Patterns

Scan (Prefix Sum)

Wen-mei Hwu - University of Illinois at Urbana-Champaign

Objective

- To master parallel scan (prefix sum) algorithms
  - frequently used for parallel work assignment and resource allocation
  - A key primitive in many parallel algorithms to convert serial computation into parallel computation
  - A foundational parallel computation pattern
  - Work efficiency of kernels
- Reading -Mark Harris, Parallel Prefix Sum with CUDA
  - http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/scan/doc/scan.pdf

(Inclusive) Prefix-Sum (Scan) Definition

Definition: *The* all-prefix-sums *operation takes a binary associative operator* $\oplus$, *and an array of n elements*

$$[x_0, x_1, \ldots, x_{n-1}],$$

and returns the array

$$[x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1})].$$

Example: If $\oplus$ is addition, then the all-prefix-sums operation
on the array [3 1 7 0 4 1 6 3],
would return [3 4 11 11 15 16 22 25].

An Inclusive Scan Application Example

- Assume that we have a 100-inch sausage to feed 10
- We know how much each person wants in inches
  - [3 5 2 7 28 4 3 0 8 1]
- How do we cut the sausage quickly?
- How much will be left

- Method 1: cut the sections sequentially: 3 inches first, 5 inches second, 2 inches third, etc.

- Method 2: calculate prefix sum:
  - [3, 8, 10, 17, 45, 49, 52, 52, 60, 61] (39 inches left)

Typical Applications of Scan

- Scan is a simple and useful parallel building block
  - Convert recurrences from sequential :

```
for(j=1;j<n;j++)
    out[j] = out[j-1] + f(j);
```

  - into parallel:

```
forall(j) { temp[j] = f(j) };
scan(out, temp);
```

- Useful for many parallel algorithms:
  - Radix sort
  - Quicksort
  - String comparison
  - Lexical analysis
  - Stream compaction
  - Polynomial evaluation
  - Solving recurrences
  - Tree operations
  - Histograms, ….

Other Applications

- Assigning camp slots
- Assigning farmer market space
- Allocating memory to parallel threads
- Allocating memory buffer for communication channels
- …

An Inclusive Sequential Addition Scan

Given a sequence $[x_0, x_1, x_2, \ldots]$

Calculate output $[y_0, y_1, y_2, \ldots]$

Such that

$$y_0 = x_0$$

$$y_1 = x_0 + x_1$$

$$y_2 = x_0 + x_1 + x_2$$

…

Using a recursive definition

$$y_i = y_{i-1} + x_i$$

A Work Efficient C Implementation

```
 y[0] = x[0];
 for (i = 1; i < Max_i; i++)
y[i] = y [i-1] + x[i];
```

Computationally efficient:

N additions needed for N elements - O(N)!

Only slightly more expensive than sequential reduction.

A Naïve Inclusive Parallel Scan

- Assign one thread to calculate each y element
- Have every thread to add up all x elements needed for the y element

$$y_0 = x_0$$
$$y_1 = x_0 + x_1$$
$$y_2 = x_0 + x_1 + x_2$$

"Parallel programming is easy as long as you do not care about performance."

To learn more, read
Section 9.1-9.2