Technische Universität Berlin

School IV – Electrical Engineering and Computer Science
Chair Communication and Operating Systems (KBS)
http://www.kbs.tu-berlin.de

Integrated Course

Parallel Programming

J. Schönherr

in Summer Semester 2013

Assignment 3

Issued: Wednesday, 15th May 2013

Due: Wednesday, 29th May 2013

Information

Please upload your solution via the ISIS page of this course until 23:55 of the due date. Your upload should consist of a PDF document covering the theoretical parts and a zip/tar.gz archive with your source files. Please list the names and matriculation numbers of all group members inside your uploaded documents.

Exercise 1 – Parallel Scan

(a) Data Parallel Scan

Consider the following algorithm for a parallel scan with an associative operator $\oplus$:

for j=1 to $\log_2 n$
  for all k from 1 to n in parallel
    if $k \geq 2^{j-1}$
      $x_k = x_{k-2^{j-1}} \oplus x_k$

(Note: This assumes, that all x_k are updated simultaneously.)
(2nd note: Realize, that $\oplus$ is not required to be commutative.)

Discuss the properties of this algorithm and compare it to a sequential scan. How many operations are executed, how long does it take to execute these operations depending on the number of elements n and the number of processors p?

(b) Handling large numbers of elements

Suppose, your programming environment provides you with a parallel scan implementation that – while hiding communication aspects – is only able to handle one element per processor. (This is for instance the case for MPI.)

How can you realize a scan for an arbitrary numbers of elements (executing the provided scan internally just once)?

Assuming that the provided scan works like the scan in (a), compare the properties of your newly developed scan operation to those of the scan in (a).

(c) Recursion

In (b) you devised a scan that internally uses another scan.

Following this line of thought, the structure of the internal scan could be similar. This leads to a recursion until we hit the trivial case of scanning just two elements.

Explore what happens, when we consider the primitive task to be the following: scan two elements, feed second element into next primitive task, fix up first element. (Resulting algorithm, properties, comparison to previous algorithms.)

Exercise 2 – Segmented Scan

Another variant of the scan operation is the segmented scan. That is, an array A has been split by some algorithm into multiple (possibly differently sized) segments, and you are interested in the individual prefix sums of all segments. Such a segmentation of an array A is usually represented by an array F of flags: if f_k=1 then a_k is the start of a new segment.

For example

A = [1 4 2 3 1 6 5 2 1 1]

F = [1 0 0 1 0 1 1 0 0 0]

represents the segments [1 4 2], [3 1], [6], and [5 2 1 1].

What we want to calculate now, are the prefix sums of all segments. For the example above, the result with addition is (together with the same F):

B = [1 5 7 3 4 6 5 7 8 9] (i. e. [1 5 7], [3 4], [6], [5 7 8 9])

Devise an associative operator, so that a regular scan over (*A, F*) will generate the desired result *B*.

Exercise 3 – Implementation

In real life, we also have to take the properties of the actual system into account.

Discuss for each, distributed memory systems, shared memory systems, and GPU-like coprocessors, which scan algorithms can be applied and which one is better suited for which architecture.

Implement either

- a parallel scan for an arbitrary number of elements in OpenCL, or
- a parallel scan (segmented or not) for an arbitrary number of elements in OpenMP, or
- a segmented parallel scan for an arbitrary number of elements in MPI.

For OpenCL solutions, a template will be available to help you to an easy start.

(This is intended to be the OpenCL exercise. However, you have the freedom so skip over the practical part of OpenCL by selecting one of the other two parallel programming environments.)