Weekly Assignment 2
Parallel Functional Programming

Troels Henriksen and Cosmin Oancea
DIKU, University of Copenhagen

December 2021

Introduction

This weekly assignment focuses on advanced parallel programming, with a particular focus on flattening.

The handin deadline is the 9th of December.

The handin is expected to consist of a report in either plain text or PDF file (the latter is recommended unless you know how to perform sensible line wrapping) of 4—6 pages, excluding any figures, along with an archive containing your source code. The report should contain instructions on how to run and benchmark your code.

Task 1: Programming in `ispc`

For this task you will be programming in `ispc`. The code handout contains a `Makefile` and benchmarking infrastructure for several programs implemented in both sequential C and `ispc`. Of these, three `ispc` implementations are blank, and it is your task to finish them.

A program `foo` can be benchmarked with the command `make run_foo`. This will also verify that the result produced by the `ispc` implementation matches the result produced by the C implementation. Feel free to change the input generation if you wish, for example to make the input sets larger or smaller.

Generally, do not expect stellar speedups from these programs. A $\times 2$ speedup over sequential C on DIKUs GPU machines should be considered quite good. By default, the `Makefile` sets `ISPC_TARGET=sse4`, which is a relatively old instruction set. You may get better performance by setting it to `host`, which will tell `ispc` to use the newest instruction set supported on the CPU you are using.

All of the subtasks involve some kind of cross-lane communication. Each subtask contains a list of the builtin functions I found useful in my own implementation, but you do not have to use them, and you are welcome to use any others supported by `ispc`. It is likely that my own solution is not optimal anyway.

Subtask 1.1: Prefix sum (`scan.ispc`)

The task here is to implement ordinary *inclusive* prefix sum, which you should be quite familiar with by now.

Recommended builtin functions: `exclusive_scan_add()`, `broadcast()`

Subtask 1.2: Removing neighbouring duplicates (`pack.ispc`)

This program compacts an array by removing duplicate neighbouring elements. It has significant similarities to the filter implementation in `filter.ispc`.

Recommended builtin functions: `exclusive_scan_add()`, `reduce_add()`, `extract()`, `rotate()`

Subtask 1.3: Run-length encoding (`rle.ispc`)

Run-length encoding is a compression technique by which runs of the same symbol (in our case, 32-bit words) are replaced by a *count* and a *symbol*. For example, the C array

```
{ 1, 1, 1, 0, 1, 1, 1, 2, 2, 2, 2 }
```

is replaced by the array

```
{ 3, 1, 1, 0, 3, 1, 4, 2 }
```

Recommended builtin functions: `all()`

Hints: This task is significantly more tricky than the two others. I advise optimising for the case where each symbol is repeated many times, which can be quickly iterated across in a SIMD fashion, falling back to scalar/single-lane execution when a new symbol is encountered. A rough pseudocode skeleton could be:

```
while at least programCount elements remain to be read:
  read next programCount elements
  if all equal to current element:
    increase count and move to next loop iteration
  else:
    # Use single lane to find the mismatch
    if programIndex == 0:
      ...
```

Task 2: Almost homomorphism fusion

Carry out the last fusion step in the *mss* example from Monday's lecture and the paper *Construction of List Homomorphisms by Tupling and Fusion.*

That is, fuse

$$mss = max^s \circ \pi_1 \circ hom\ (\Delta_1^4 \otimes_i)\ (\Delta_1^4 f_i')$$

to produce

$$mss = \pi_1 \circ hom\ (\Delta_1^4 \otimes_i')\ id$$

where

$$\begin{aligned}
f_1'\ x &= \{x\} \\
f_2'\ x &= [x] \\
f_3'\ x &= [x] \\
f_4'\ x &= [x]
\end{aligned}$$

$$\begin{aligned}
(s_1, i_1, t_1, d_1) \otimes_1 (s_2, i_2, t_2, d_2) &= s_1 \cup s_2 \cup (t_1\ \mathcal{X}_+\ i_2) \\
(s_1, i_1, t_1, d_1) \otimes_2 (s_2, i_2, t_2, d_2) &= i_1 +\!\!+ map\ (d_1+)\ i_2 \\
(s_1, i_1, t_1, d_1) \otimes_3 (s_2, i_2, t_2, d_2) &= map\ (+d_2)\ t_1 +\!\!+ t_2 \\
(s_1, i_1, t_1, d_1) \otimes_4 (s_2, i_2, t_2, d_2) &= d_1 + d_2
\end{aligned}$$

If you read the paper or the slides you can find the answers (the $H = h_1 \times h_2 \times h_3 \times h_4$ to use for the Almost Fusion Theorem and the ultimate definitions of $\otimes'$). For this task you must show the actual steps: write the equations induced by the Almost Fusion Theorem (or in the paper) when fusing *segs* into map^s *sum*. Recall that these take the form

$$\begin{aligned}
&h_j\ ((s_1, i_1, t_1, d_1) \otimes_j (s_2, i_2, t_2, d_2)) = \\
&\quad (h_1\ s_1, h_2\ i_1, h_3\ t_1, h_4\ d_1) \otimes_j' (h_1\ s_s, h_2\ i_2, h_3\ t_3, h_4\ d_2)
\end{aligned}$$

for $j = 1, \cdots, 4$. Show the steps to fully derive $\otimes_j'$ for at least one j. Also show how $h_j \circ f' = id$ for $j = 1, \cdots, 4$.

Hints This LaTeX snippet defines a command for writing $+\!\!+$:

```
\newcommand\concat{\ensuremath{\mathbin{+\mkern-10mu+}}}
```