Recitation 14

PASL

14.1 Announcements

- *DPLab* is due **Tuesday afternoon**.
- *PASLLab* will be released on Tuesday also and will be due at the end of the semester.

14.2 `map_flatten`

If you would like to see the code run on your computer, begin by downloading the files `rec14.hpp` and `rec14-bench.cpp`. You can put these in the top directory of PASLLab once it is released. Then, edit PASLLab's Makefile to add: `rec14-bench.cpp` to the list of programs, i.e.

```
PROGRAMS=\
  sandbox.cpp \
  check.cpp \
  bench.cpp \
  rec14-bench.cpp # add me here.
                  # don't forget the slash on the previous line.
```

Task 14.1. *Using PASL primitives, implement the function*

```
template <class Map_func, class Size_func>
sparray map_flatten(const Map_func& f,
                    const Size_func& g,
                    const sparray& xs);
```

where, at a high-level, the goal is to compute

$$\texttt{flatten}\left\langle f(x) : x \in xs\right\rangle.$$

Begin by thinking of a sequential implementation and then parallelizing it. You should assume that the function arguments are typed as follows, where `f(xs[i])` *is a pointer to the front of an array of length* `g(xs[i])`*.*

$$\begin{aligned}\texttt{f} &: \texttt{value\_type} \rightarrow \texttt{value\_type*}\\ \texttt{g} &: \texttt{value\_type} \rightarrow \texttt{long}\end{aligned}$$

14.3 inject

Throughout the semester, we've largely kept the sequence function inject shrouded in mystery. Let's see how the magic works!

Task 14.2. *Using PASL, implement the function*

```
sparray inject(const sparray& xs,
               const sparray& indices,
               const sparray& updates);
```

which returns the result of injecting into xs*. We require that* indices *and* updates *be the same length, such that for each* i*, we attempt to write* updates[i] *at position* indices[i] *in* xs*. Note that you should not destructively modify* xs*.*

If there are multiple updates specified at the same position, then all except the last should be ignored. (We want to match the behavior of inject *as specified in the 15210 Library.)*

14.4 Benchmarking

Try running some speedup experiments! The two bench arguments are `map_flatten` and `inject`, respectively. For example, the following injects m randomly placed updates into an array length n. In the `map_flatten` benchmark, n is the initial array size, and m is the size of each subarray (so the output is length nm).

```
make rec14-bench.opt rec14-bench.baseline

./prun speedup -baseline "./rec14-bench.baseline" \
-parallel "./rec14-bench.opt -proc 1,5,10,15,20" \
-bench inject -n 100000,1000000 -m 100000000,200000000

./pplot speedup -series n,m
```