Progression Review

Chris Cummins

Review

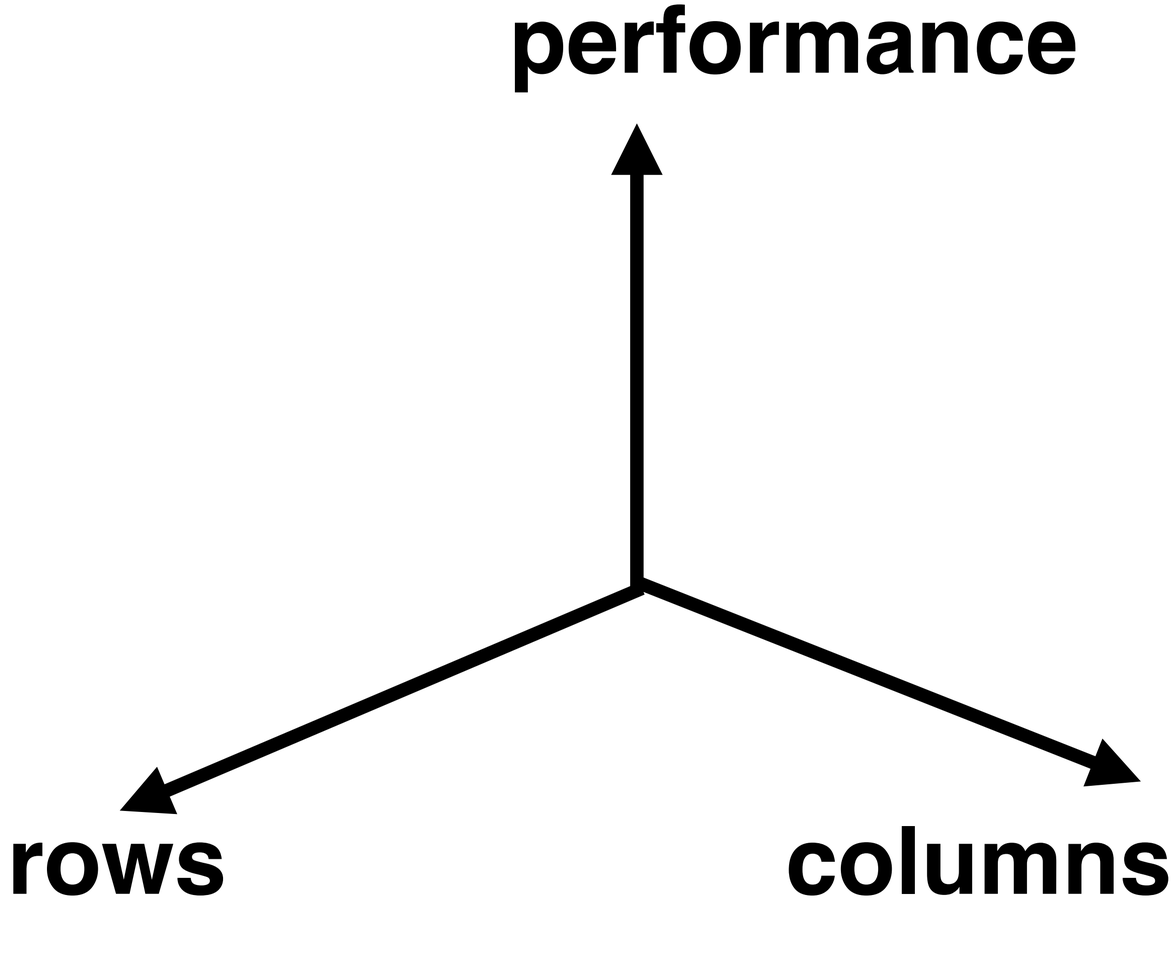
Thesis

Autotuning Stencil Codes with Algorithmic Skeletons

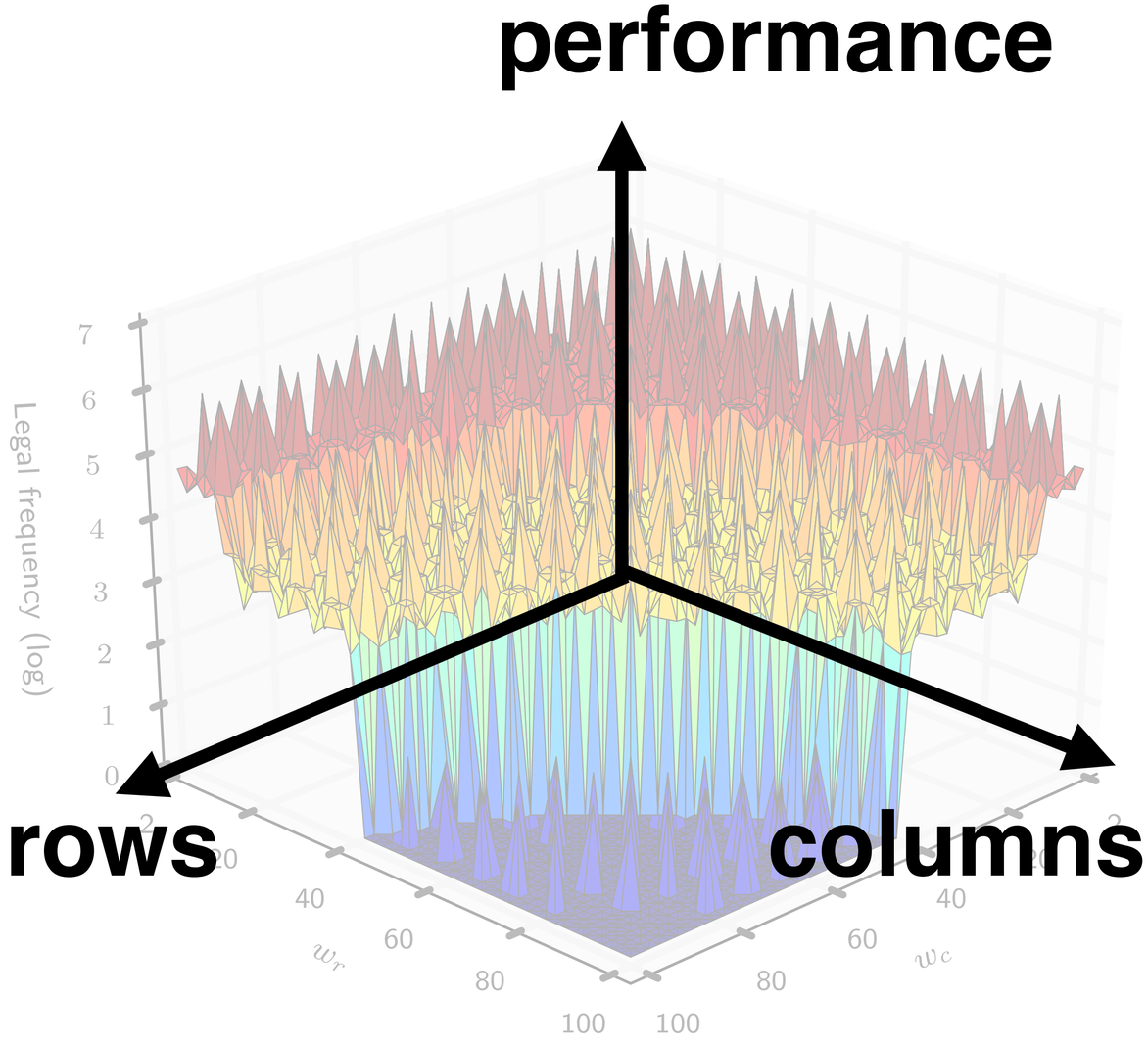
Thesis Autotuning

Stencil Codes with Algorithmic Skeletons

2D parameter space



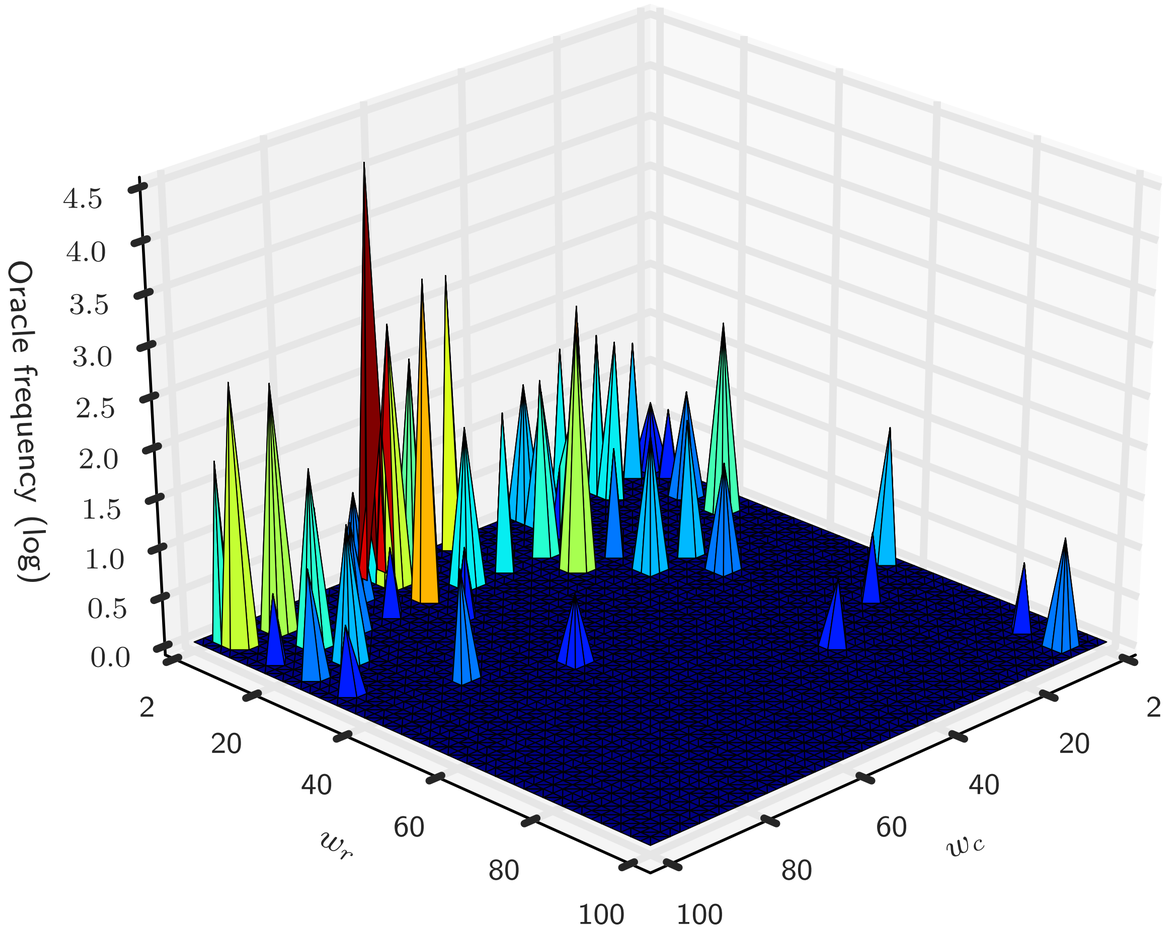
2D parameter space



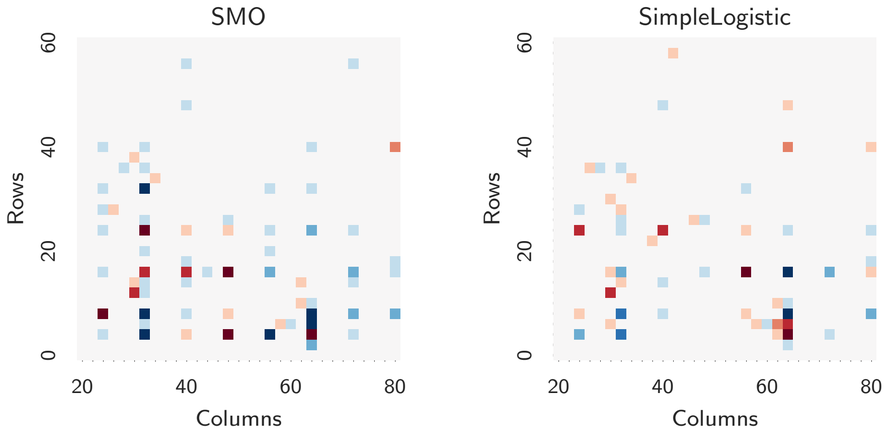
3x Supervised machine learning

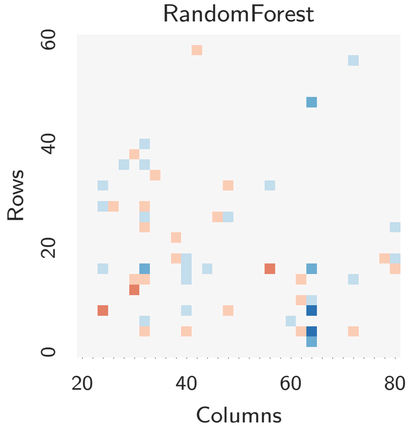
Predict:

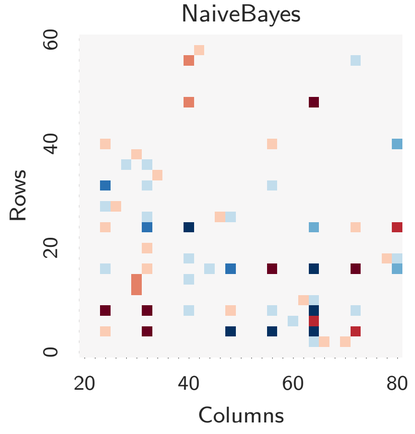
oracle runtime speedup

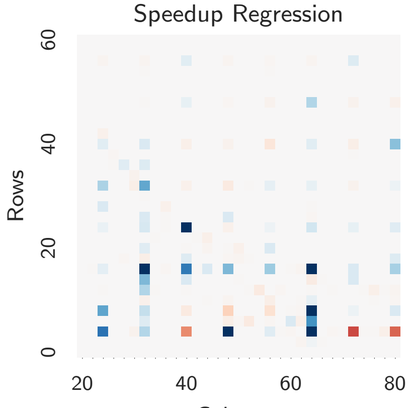


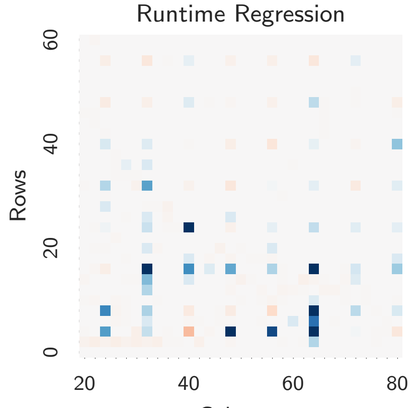
Compare Predictions

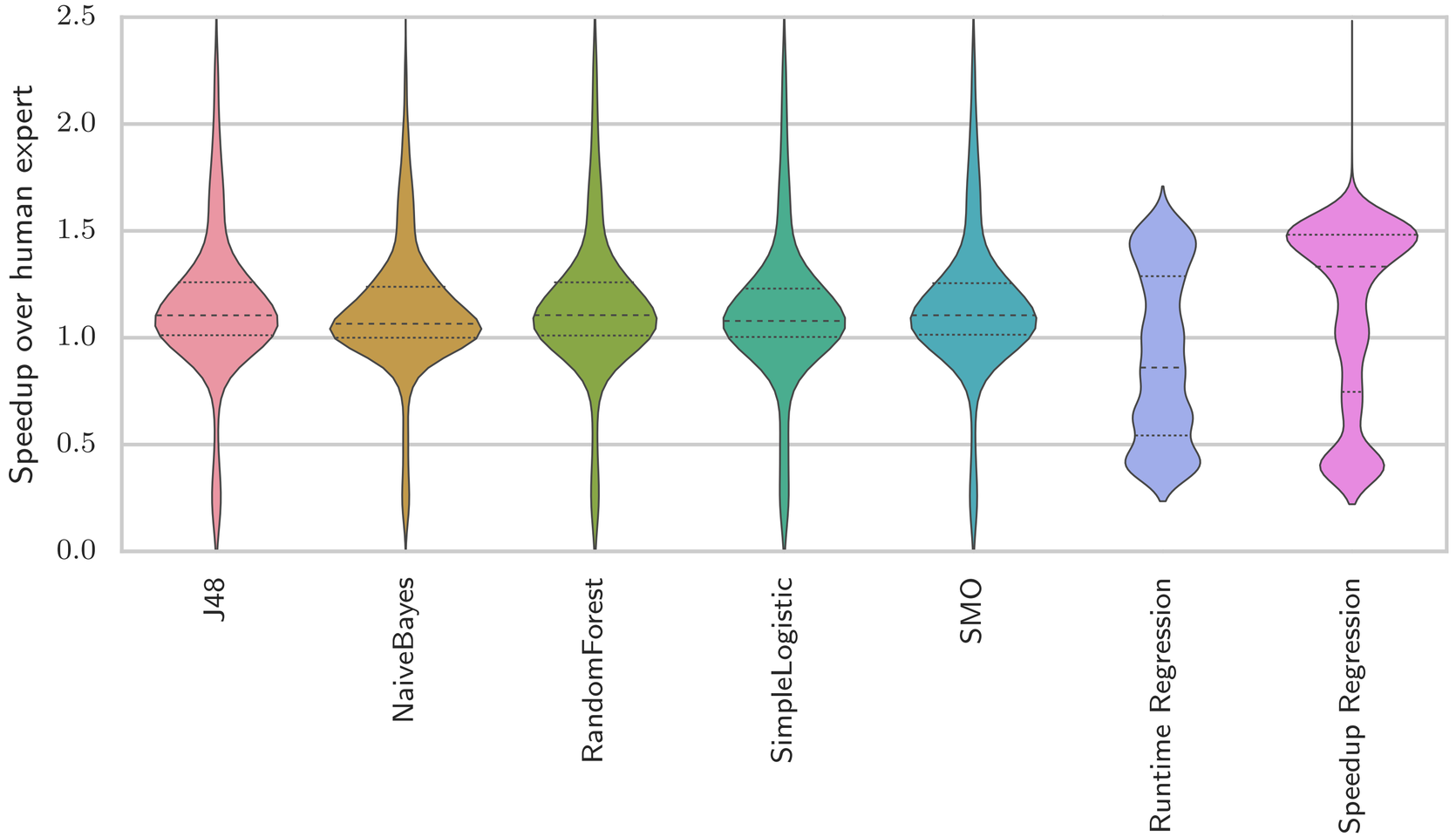












Compare

Speedups

Thesis Autotuning

Stencil Codes with Algorithmic Skeletons

Thesis Autotuning Stencil Codes with Algorithmic Skeletons

2D, iterative operations

```
// "Complex" kernel. Performs lots of trigonometric heavy lifting.
DATA_T func(input_matrix_t *img) {
        DATA_T sum = 0;

        // Iterate over all except outer neighbouring elements.
        for (int y = -SCL_NORTH + 1; y < SCL_SOUTH; y++) {
                for (int x = -SCL_WEST + 1; x < SCL_EAST; x++) {
                        // Do *some* computation on values.
                        DATA_T a = sin((float)getData(img, -1, 0));
                        DATA_T b = native_sin((float)getData(img, 0,
1) * a);

                        sum += getData(img, y, x) * a * (b / b);
                }
        }

        DATA_T out = 0;
        // Loop over horizontal region.
        for (int i = SCL_EAST; i >= -SCL_WEST; i--) {
                // DO *some* computation on values.
```

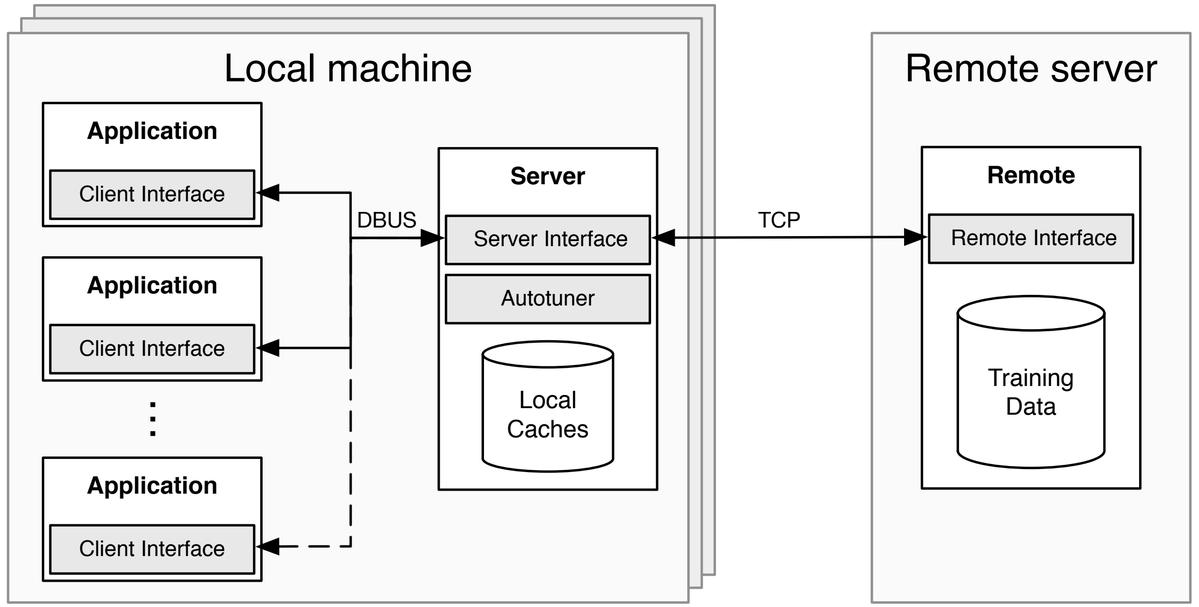
Synthetic benchmark generator

Thesis Autotuning Stencil Codes with Algorithmic Skeletons

Thesis Autotuning Stencil Codes with Algorithmic Skeletons

SkelCL
OpenCL
C++ Templates
GPUs

OmniTune



Thesis Autotuning Stencil Codes with Algorithmic Skeletons

Thesis

Autotuning Stencil Codes with Algorithmic Skeletons

How did it go?

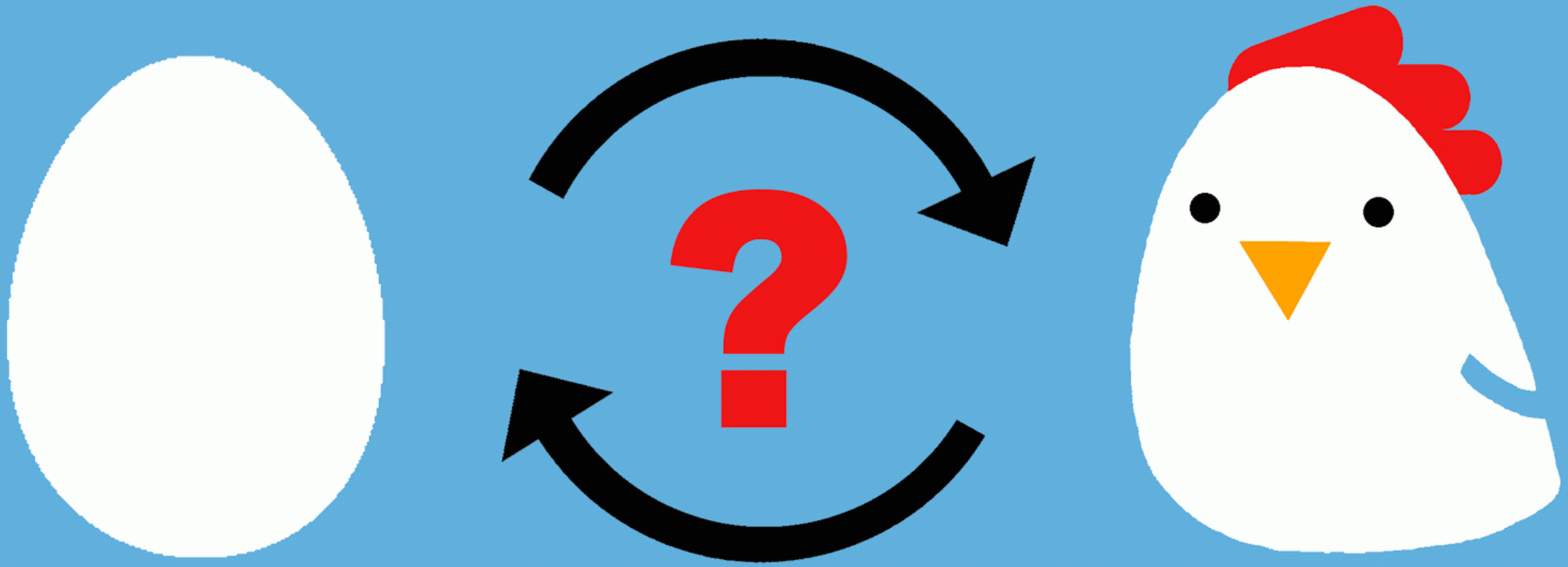
Lessons learned

Scalable experiments

Write up often

Plan for planning

The Plan



Design

Implement

Proposal

Skeleton-aware compilation

There is only so much optimisation that can be performed at the library level.

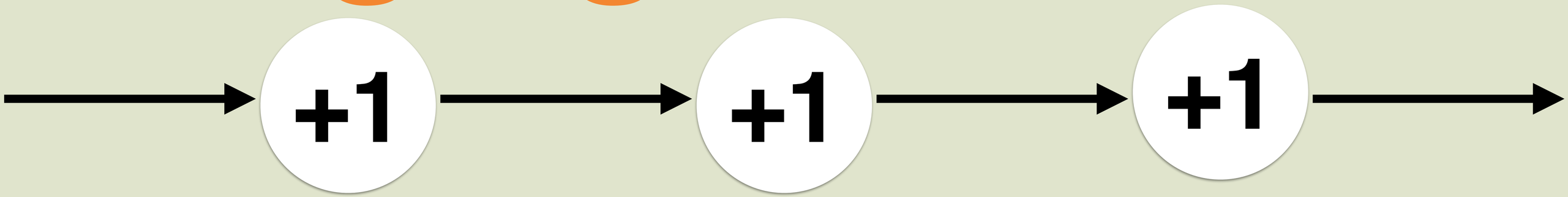
Can’t optimise

Across muscle functions

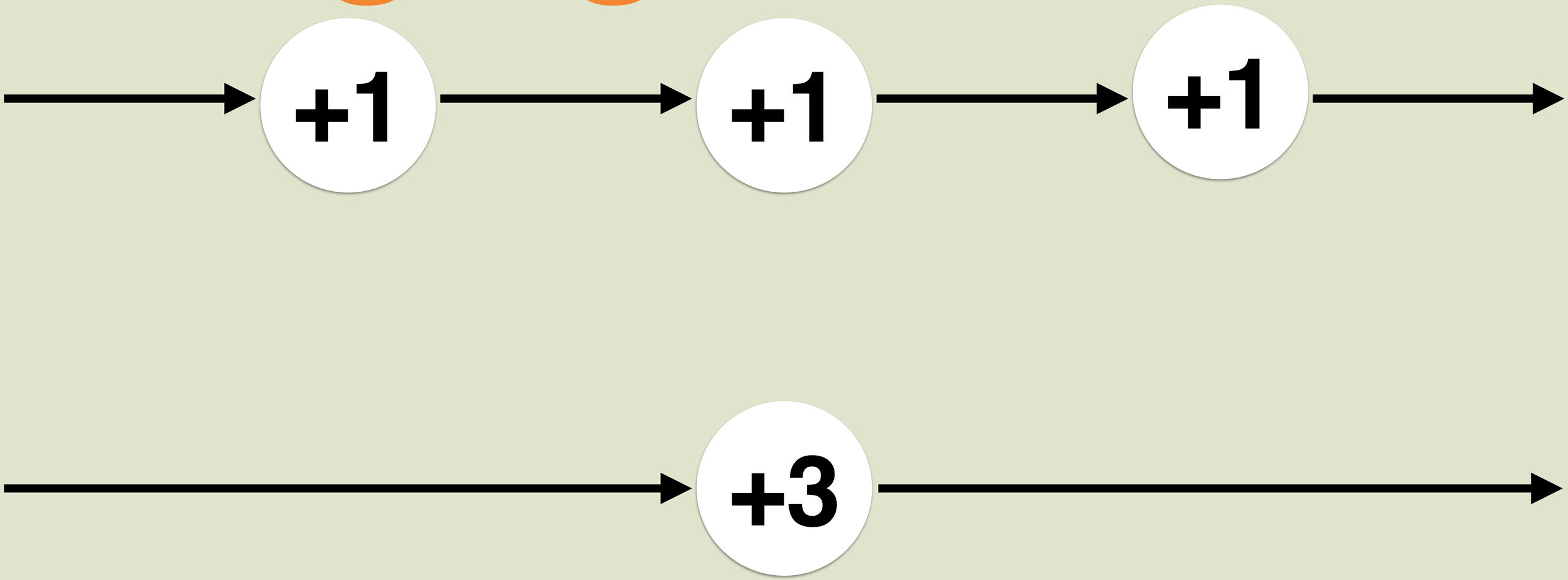
Across skeletons

SKELETON “COARSENING”.
LOAD BALANCING.
NESTED SKELETON RE-ORDERING.
CONSTANT PROPAGATION.

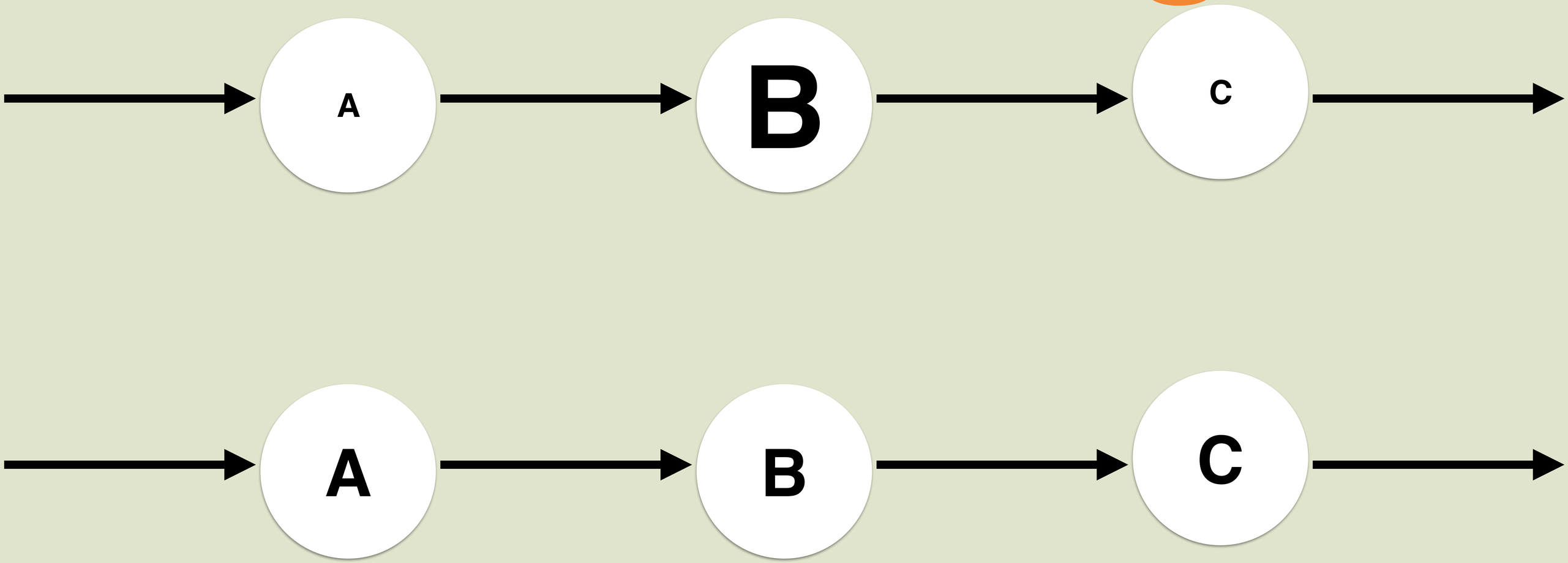
Merging



Merging



Load balancing



What is the simplest first step that would need to be taken?

Enumerate range of what *can* be done.

Evaluate range of what *has* been done.

To test an optimisation

Implement by hand.
Measure performance.
Implement compiler pass.
Measure success.

Composition

```
(pmap f4
  (pmap f3
    (pmap f2
      (pmap f1 x))))
```

Composition

```
(pmap (comp f4 f3 f2 f1) x)
```

Composition

```
(pmap (comp f4 f3 f2 f1) x)
```

1.81x faster

(1e6 floats, 30 samples)

Summary

My TODO list

1. review the literature.
2. hand-test optimisations
3. publish a workshop paper (ADAPT or HLGPGPU)

Skeleton-aware compilation