Approach: Lazy Evaluation and Skeleton Lineages

- For data locality optimization: consider sequences of skeleton invocations to enable tiling across entire sequence
- Inspiration from the Spark framework for big data analytics
  - Locality is even more important on Hadoop clusters where the access times are much higher
  - Solved with lazy evaluation: a lineage (DAG) of transformations on data is built up until an action is required
  - Spark containers are single-assignment, unlike SkePU
  - In SkePU, Maps are "transformations" and other operations are "actions", extensible to other skeletons as well
  - Lineages give run-time information of actual program flow (dynamic rather than static analysis)
- Once an action is required on a container, the lineage is traversed backwards, following dependencies, and its nodes are evaluated starting from the roots
- Skeleton calls may be evaluated globally out-of-order, but still in-order w.r.t. data dependencies

Example Program

```c
Vector<float> v1, v2, v3, v4, v5, v6, v7, v8, v9;
auto add = Map2<<[](float a, float b){ return a+b;}>; // transformation using Map instance
add(v1, v3, v4); // transformation using Map instance
auto copy = Map2<<[](float v){ return v;}>; // transformation using Map instance
copy(v9, v1);
auto mult = Map2<<[](float a, float b){ return a*b;}>; // transformation using Map instance
mult(v2, v1, v3); // transformation using Map instance
auto square = Map2<<[](float v){ return v*v;}>; // transformation using Map instance
square(v1, v2); // transformation using Map instance
auto generate = Map2<<[](float v){ return v;}>; // transformation using Map instance
generate(v6, generate(v6, 5.f)); // transformation using Map instance
for (int i = 0; i < 5; i++)
  add(v8, v8, v8);
reduce(v8); // action point, causes evaluation
```

Selected SkePU Publications