Optimizing Array Locality via Memory Layout Reorganization

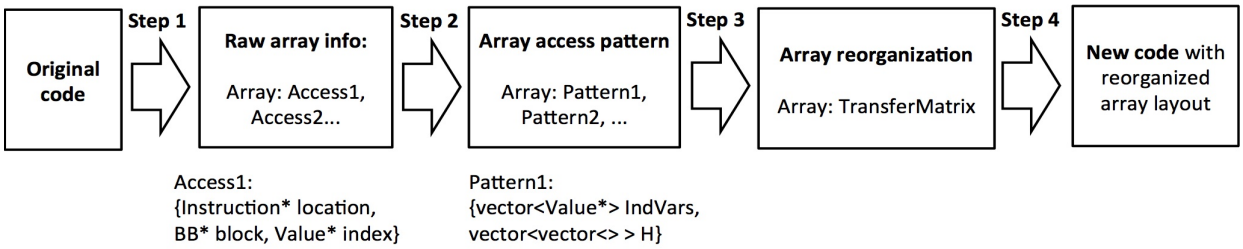
Xuezhi Wang (xuezhiw@cs.cmu.edu), Junchen Jiang (junchenj@cs.cmu.edu)
Project webpage: www.cs.cmu.edu/~junchenj/15745proj/index.html

Major Changes:

We find it hard to automatically measure the actual cache miss rate. Instead we would write code to investigate the indices visited by the array, by running both the original program and the revised program, and compute the cache miss rate by making certain assumptions about the cache (e.g., cache size).

What You Have Accomplished So Far:

We have finished code to detect access patterns of arrays in programs, and also finished code to reorganize memory layout based on different access patterns. The major part of this project has been finished (except the analysis of multiple access patterns) and currently we are able to transform array in code for a better memory layout, thus resulting in better locality. Specifically, the implementation consists of four steps as demonstrated by the figure below.



Step 1 makes a pass on the original code to get **raw access information** for each array. For each array, we record the location (inst, block), access index variable (index), etc of each access of the array.
Step 2 processes the raw information of each access to get the **access pattern**, which includes induction variable matrix (H) and induction variables (IndVars) used by each index access.
Step 3 calculates a **reorganization** for each array based on the access patterns. The reorganization is essentially a linear transformation (through a TransferMatrix) between index of original array and that of the new array.
Step 4 generates the **optimized code** by initializing a new array with new size for each old array and replace all access to the old array to the corresponding new array with index calculated by TransferMatrix.

Core algorithm:
Consider a 2-d array A used in a 2-level nested loop (outer indVar i, inner indVar j), and $[x, y] = [i, j] H^T$. where H is the induction variable matrix. Now, think about a new array B, and element at A[x, y] corresponds to B[x', y'] such that $[x', y'] = [x, y]P$ where P is the **TransferMatrix**.
The key idea of our algorithm is find a TransferMatrix P to achieve: when j increments, if we access the corresponded elements in B rather than A, we could have much better locality, i.e., $[0, \Delta] = [0, 1] H^T P$. *where* Δ *will be as small as possible (smaller than the cache size).*

Meeting Your Milestone:

We think we almost finish the milestone. The only change we want to make is the measure methods, and we hope to finish it before the end of this week.

Surprises:

1. When reorganizing the memory layout by solving linear equations, we find that the solutions do not necessarily make the transformed indices nonnegative. Currently we add an offset (can be computed by the size of the array) to the transformed array so that the transformed indices are guaranteed to be nonnegative.
2. When solving the linear equations for more than three variables, we find it is generally hard to get integer solutions. For two variables we can use the extended Euclidean algorithm, and for three variables we can further extend the algorithm by finding a particular solution of one variable first. However there does not exist general algorithms for more than three variables. Currently we are using integer programming method (searching solutions over a relatively small range), which is able to solve equations involving any number of variables. But it would be relatively slow when the searching range increases or the number of variables is large.

Revised Schedule:

Apr 19 - Apr 21: Add printing code to the program for measurement [Junchen, Xuezhi].
Apr 22 - Apr 27: Implement code for programs that have multiple access patterns. [Junchen, Xuezhi]. Test on the benchmarks. [Xuezhi]
Apr 28 - Apr 30: Writeup final report. [Junchen, Xuezhi]

Resources Needed:

We will basically use LLVM on the same Ubuntu image as in the homework. We will use SPEC benchmark (http://www.spec.org/benchmarks.html) to test our proposed methods.