Class exercise: Improving loops

COMP 40

November 16, 2011

Group

Keeper of the record:
Other group members:

Improving loops

The following loop implements a row-major mapping:

```
struct T {
  int width, height;
  int size;
  UArray_T elements; /* UArray_T of 'height * width' elements,
                        each of size 'size', in row-major order */
   // Element (i, j) in the world of ideas maps to
   //   elements[i + width * j], where the square brackets
   //   stand for access to a Hanson UArray_T
};

void map_row_major(struct T *a2,
    void apply(int i, int j, struct T* a2, void *elem, void *cl), void *cl) {
  assert(a2);
  for (int k = 0; k < a2->width * a2->height; k++) {
    int col = k % a2->width;
    int row = k / a2->width;
    apply(col, row, a2, UArray_at(a2->elements, k), cl);
  }
}
```

Improve the `map_row_major` function using the following procedure:

1. In the loop, draw a rectangle around every expression that *you* believe is invariant. (An invariant expression has the same value on every iteration.)
2. In the loop, draw a circle around every expression that the *compiler* can prove is invariant.
3. In the loop, draw a *dotted* rectangle around every expression that is *nearly* invariant (its value changes relatively infrequently).
4. Rewrite the procedure to exploit the gaps between the rectangles and circles: use your knowledge of invariants to make improvements the compiler cannot make.
5. Restructure the code to reduce the cost of computing expressions that are nearly invariant.