Class Announcements

Here is where we are:

- Third project is due Monday, May 1
- Sixth homework due Tuesday, April 25 (tonight)
- Seventh homework has been posted; due Monday, May 1
- Final (third midterm) exam on Thursday, May 4, noon - 3:00pm timeslot. Location: most likely this room.

Any CONFLICTS with other classes?

Only the top 4 homeworks count for the final grade

Topics covered in third (final) midterm

1. Projects 2 (Scheme) and 3 (Parallelization)

2. Parallel executions, dependence analysis, loop transformation, automatic parallelization and vectorization

3. Basic concepts in quantum computing, qbits, NOT, CNOT and Hadamard Gates
A Simple Vectorizing Compiler

How to vectorize the following loops?

```c
for (i=2; i<100; i++) {
    S1: a[i] = b[i+1] + 1;
    S2: b[i] = a[i] + 5;
}
```

```c
for (i=2; i<100; i++) {
    S1: a[i] = b[i-1] + a[i-1] + 3;
    S2: b[i] = a[i+1] + 5;
}
```

**Simple vectorizer assumptions:**

1. singly-nested loops
2. constant upper and lower bounds, step is always 1
3. body is sequence of assignment statements to array variables
4. simple array index expressions of induction variable (i +/- c or c); can use ZIV or SIV test
5. no function calls
A Simple Vectorizing Source-to-Source Compiler

SKETCH OF BASIC ALGORITHM

Here is a basic vectorization algorithm based on a statement-level dependence graph:

1. Construct statement-level dependence graph considering true, anti, and output dependences; in the final dependence graph, the type of the dependence is not important any more

2. Detect strongly connected components (SCC) over the dependence graph; represent SCC as summary nodes; walk resulting graph in topological order; For each visited node do

   (a) if SCC has more than one statement in it, distribute loop with statements of SCC as its body, and keep the code sequential

   (b) if SCC is a single statement and has no dependence cycle, distribute loop around it and generate vector code; otherwise, mark distributed loop sequential.