Announcements

- Reading for today's lecture is
  - "Commutativity Analysis: A New Framework for Parallelizing Compilers"
- Reading for Thursday:
  - PVM paper (on the web or CS library)
- If you are taking the class for 1 unit
  - please send me mail

Target Environment

- Shared memory multi-processors
- Object oriented programs
  - C^{++} class methods
  - pointer based graph data structures
- Sources of parallelism
  - method invocation
  - methods may be invoked:
    - recursively
    - simple looping constructs (converted to tail recursion)

Analysis

- Determine if two method invocations commute
  - intuitive definition: can be performed in any order
  - a followed by b (a;b) is the same as b then a (b;a)
- Technique
  - symbolic evaluation
    - generate symbolic results of running a;b and b;a
    - like running a method but expressions not data
  - compare two results
    - ivar analysis - are the variables the same
      - need to know basic commutative ops (e.g. addition)
    - sub-method invocation
      - are multi-sets of different invocations the same

Performance Issues

- Method size
  - methods should be the “natural” size
  - too small - not enough work for overhead
  - too large - results in a load imbalance
- Synchronization
  - need to provide mutex over shared data
  - granularity an important parameter
    - too small - lock overhead dominates
    - too large - reduce potential parallelism
  - compiler can change granularity
    - start with one lock per method invocation
    - use lock “coarsing” to merge locks across invocations

Lock Granularity (cont.)

Hard to know correct lock size at compile time

Solution: use runtime adaptation

- Generate multiple versions of methods
  - each using a different lock granularity
  - also provide a way to switch between versions
- Run each version
  - each version runs on a different part of the data
  - gather timing data
  - need to run long enough for a “representative” sample
- Select best choice for rest of execution

Questions about the technique

- Are the speedup numbers good?
  - Martin liked them, but he is the author!
  - 50% parallel efficiency is not bad for an automatic tool
- Is the technique general?
  - has only tried the ideas on two applications
    - these were the target application from the start!
  - works for recursive graph structures
    - how big is this application domain?
- Will it work and play well with other approaches?
  - can data parallelism be used for parts of the computation?