Understanding Communication Characteristics in Shared Memory Applications

David Sheffield, Kurt Keutzer

**Motivation**
- Communication in shared memory programs is expensive
- Avoid communication if possible
- We have developed a suite of tools to measure communication and memory behavior
- Dynamic binary instrumentation with Pin
  - Instrument existing x86 binaries
  - No source modifications required
  - Analysis code is injected into an existing binary at runtime
  - Just-in-time recompilation

**Measuring communication**
- Metadata associated with main memory keeps track of sharers
- Memory operations augmented with analysis code
  - Dynamic binary translation used to inject instrumentation code

**Recording memory behavior**
- Each memory operation stores thread-local metadata
  - Last address
  - Stride automata graph

**Splash-2 FFT**
- 2^20 complex points, 16 processors, six-step 1d FFT algorithm
- Results closely match our communication expectations
  - 2^17 bytes exchanged between processors
  - Each processor owns 2 complex doubles
  - Two all-to-all exchanges with 16 byte operands
  - First transpose is integrated into data distribution from processor 0

**SpMV memory behavior**
- Test matrix
  - CSR format
  - 92 non-zero entries
- Constant stride access to matrix and column indices
  - 4 byte operands used as column indices
  - 8 byte operands used for matrix values
- Access to the dense vector is highly irregular
  - Access pattern is data-dependent

**CS267 programs**
- Particle simulation
  - Shared memory implementation poorly scales due to excess communication
- 0-1 knapsack problem
  - UPC and shared memory implementation have similar communication topology

**Future work**
- Apply tools to real applications
- Splash-2 and CS267 assignments are not very realistic
- Reduce tool memory requirements
- Redundant metadata
- Automatically detect computational patterns?
- We can we cluster computational patterns with memory FSMs?