The Stanford Smart Memories Project: Software

**The Smart Memories Compilation Process**

We have designed our compilation environment to be PCA compatible.
- Will focus on C/C++ for threaded and Brook for streaming apps
- Threaded STL with transactions to facilitate parallel programming
- Java can also be used in a more conventional manner
- PCA high-level compilers to break down source code to SVM/TVM
- **xcc** from Tensilica and dynamic compilers at low level
- OPEN64 xcc compiler is automatically generated by Tensilica tools
- We will add compiler passes to allow additional parallelization functions
- Will leverage existing JRP Java dynamic compilation framework

**SAPIENT Application Analysis Framework**

**Goal:** Efficient transformation of sequential applications for parallel execution on polymorphic architecture

**Parallel Analysis Approach:** Profile and analyze individual regions of application execution to find available parallelism automatically
- Identifies data-level parallel (DLP) and thread-level parallel (TPL) loops
- Profiles loops to determine dynamic dependencies, data access stride, & locality
- Inherent parallelism is detected based on available coverage and concurrency
- **DLP:** Considers costs of parallel thread creation, communication and synchronization
- **TPL:** Looks for limited communication and similarity between loop iterations
- Explores parallel analysis to the programmer and provides feedback
- Enables informed decisions regarding how to apply programmer effort and what kinds of performance tradeoffs must be considered
- Gives ideal and realistic performance estimates of all parallel loop combinations
- Can inform if parallel regions than any human programmer can
- Analysis parameters can be altered to better fit any target parallel machine
- Compares combinations of parallel regions and recommends optimal ones
- Recommends optimal architecture configurations for each combination
- Optional numbers of PCA architecture tiles to allocate to a particular application
- Static allocations of tiles between threaded and data parallel modes for each combination

**Probabilistic Reasoning Algorithms**

**Goal:** Evaluate Smart Memories architectural features with a new generation of leading-edge algorithms
- Demonstrate utilization of reconfigurable components
- Understand the computation and communication requirements of these algorithms

**Algorithm:** Recursive conditioning of Bayesian nets
- Start with a decomposition tree (d-tree) that encodes how a Bayesian net can be divided into independent sub-nets
- Recursively traverse a binary tree representing the network
- At each node, iterate over possible value assignments to variables
- At leaves, look up probability using inherited variable assignments
- Intermediate results may be cached at internal nodes
- Allow memory/exeption transactions to be made
- Loop-level parallelism
  - Parallelize at instruction loop at each internal node
  - Static loop level optimization
  - Synchronization is enabled at the root node
- **Problem:** Load imbalance (expensive dynamic balancing required)

**Results:** From Linux-based modeling of apps...
- Show results with multiple-context processors
- Loop-Level — Inherently limited — Contexts sometimes exaggerate problem
- Task-Level — Inherently limited — Contexts sometimes exaggerate problem

**SVM Simulator & Validation**

- The SVM simulator estimates performance of run-time of application based on machine model performance characteristics
- Used SVM code micro-kernels to evaluate performance characteristics
- Will use larger applications when the R-Stream compiler becomes more robust
- Ran applications on several platforms and compared results:
  - 3 kinds of hardware: Imagine streaming processors, ATI and NVIDIA GPUs
  - Also ran all applications on the SVM simulator with all three machine models
- We found that the SVM simulator predicts hardware performance very well
- Performance trends obtained match those seen in real hardware
- Shows that the SVM simulator will be a good tool for analyzing streaming code
- Indicates that the SVM is a good intermediate representation and compiler target