Easy PRAM-based High-performance Parallel Programming with ICE

**Fady Ghanim**

**Technical Contribution**

- Multi-threaded execution is the norm
- Problem statement: Can we enable tightly-synchronous threading-free programming for multi-threaded execution?
- Current understanding: No. Performance programming must be multi-threaded.
- New result: Yes: Parallel programming can be lock-step.

**Significance**

- Hardware parallelism is increasing.
- Auto parallelization in hardware or software?
- Limited success and scaling. Not for irregular programs -> Parallel algorithms & programming.
- But, how to minimize human effort?
- **Our goal**: Specify what is parallelizable, but nothing else.

**Facts:** Parallel programmer must specify much more. He/she is expected to partition a task into subtasks (threads) so as to meet multiple constraints and objectives, involving data and computation partitioning, locality, synchronization, race conditions, limiting and hiding communication latencies.

- Pain of parallel programming of the available ecosystem: commodity hardware and parallel programming languages.

**Intermediate Concurrent Execution (ICE) Model**

- A parallel algorithm is expressed as a series of time steps of parallel operations.
- Lock-step execution model: A fine step is not executed until all operations of the previous time step are completed.
- Parallel Random Access Machines (PRAM) is the main parallel algorithmic theory.
  - The “Work-Depth” (WD) abstraction. Pseudocode uses “pardo”. defines ICE.
  - PRAM is a large latent knowledge base of algorithms and techniques.
- Uses the XMTC model developed at UMD.
  - Designed with irregular algorithms (like those in PRAM) in mind.
  - Programmed using threaded parallel language called XMTC.
  - XMTC uses “spawn” keyword to create concurrent threads.
- The ICE compiler translates the ICE high level language into XMTC.

**The ICE Language**

- The ICE language is based on the C language.
- Extends C by adding a new keyword “pardo”. Used to specify parallelism as in WD.
- Shared variables are declared outside the pardo block.
- Private variables are declared within the pardo block.

**Translation: ICE to XMTC**

- Threaded model (XMTC) is incompatible with lock-step model (ICE).
- In lock-step, different parallel contexts progress in concert one step at a time.
- Threads each progresses on its own pace regardless of other threads.
- Correct translation requires synchronizing threads by introducing barriers between dependent memory accesses.
- A “pardo” block is split into multiple “spawn” blocks.
- The splitting occurs wherever barriers were added.
- Use temporary variables to communicate data and control flow between different “spawn” blocks.

**Experimental Results**

- Goal: ICE produces XMTC code that has a comparable performance to hand optimized XMTC.
- Developed a benchmark suite consisting of 11 PRAM algorithms.
- The experiment was conducted by:
  - Producing a pseudocode for each algorithm in the suite.
  - Using the pseudocode, two implementations were produced: an XMTC version manually optimized for best performance, and an ICE version.
  - Compile and execute each version on a 64 core XMTC processor.
- ICE achieves comparable performance to optimized XMTC while requiring considerably less effort.
  - Average speedup of ICE across all benchmarks is 0.76%.
  - Maximum slowdown was 2.7%, maximum speedup was 8.3%.
- We do not claim that ICE will provide speedups compared to hand-optimized XMTC.

**Translation: Optimization**

- Splitting a pardo block into multiple spawn blocks causes performance degradation.
- So does using shared memory to communicate information.
- Minimizing the number of splits is crucial to high performance.
- Consolidate unnecessary splits wherever possible.
- Use a list scheduling algorithm to group independent memory accesses into clusters.
- Each cluster becomes a spawn block later on.
- Called clustering algorithm.

**Conclusion**

- Transcribe PRAM algorithms right out of the textbook & go fishing.
- Freeing parallel programmers from current pain points.
- Get the best performance with proper compiler and architecture.
- Was it premature to replace the Parallel Algorithms section by a Knowledge-Based Algorithms section in some standard algorithms texts?
- To be fair, we surprised even ourselves. The XMTC (explicit multi-threading) platform expected a manual workflow: starting from PRAM algorithms produce multi-threaded programs. Not directly-transcribed PRAM.