Transitioning to Multicore

ParaSailing into the Future

S. Tucker Taft

October 2011
Outline of Presentation

- Language Design for Fun and Profit
- Getting Rid of Old Baggage
- Simplify, Unify, Parallelize, Formalize
- New Data Structuring Approaches
- Divide and Conquer
- Conclusions
Language Design for Fun and Profit

- S. Tucker Taft
  - Chief scientist at Intermetrics for 22 years
  - Founder of SofCheck, Inc. in 2002

- Involved with language design and compiler implementation for past 35 years

- LISP, Ada, Embedded C, VHDL, ParaSail

- Focus on high-integrity systems
  - Aerospace
  - High-speed rail
  - Multi-level security

- Compiler and run-time system should do the heavy lifting to increase safety, security, and correctness
Getting Rid of Old Baggage

- Lose the sequential mindset
- Lose the pointer/heap/garbage-collection addiction
- Lose the focus on threads and explicit synchronization
- Adopt a divide-and-conquer mindset
- Think in terms of expandable and shrinkable objects rather than pointers or nodes
- Use a pervasively parallel model where compiler ensures there is no hidden aliasing and no race conditions
- Parallel programming for the rest of us...
Simplify, Unify, Parallelize, Formalize

- **Simplify/Unify**
  - Smaller number of concepts, uniformly applied, all features available to user-defined types
  - Simplify to make *conceptual room* for parallelism and formalism

- **Parallelize**
  - Parallel by default
  - Have to work harder to force sequential execution

- **Formalize**
  - Assertions, Invariants, Preconditions, Postconditions integrated into the syntax
  - All checking at compile-time -- if compiler can’t prove the assertion, program is illegal -- no run-time checking, no run-time exceptions
Expandable Objects Instead of Pointers

- Generalized indexing into containers replaces pointer dereferences
  - Similar to region-based storage management (e.g. Cyclone)
  - Region[Index] analogous to *Index
    - Presuming Index is known to point into Region

- Objects can be declared optional and can grow/shrink
  - Eliminates many of the common uses for pointers
  - Tree has subtrees that are optional
  - Subobjects can be moved or copied to new places, but data is never shared.

- Short-lived references to existing objects are permitted
  - Necessary for user-defined indexing
  - Not re-assignable; reference always refers to same object
    - Target object cannot be destroyed during lifetime of reference
Why and How to Parallelize?

- Computers have stopped getting faster -- they are getting “fatter” -- more cores, not more GHz
- Programmers are lazy -- will take path of least resistance
  → The default should be parallel -- must work harder to force sequential evaluation.
  → Programmer mindset: there are 1000s of threads, and their goal is to use as many as possible.
- Compiler should prevent race conditions, and ideally, deadlock as well.
Why pervasively parallel?
Computers have stopped getting faster

Figure 2. Historical growth in single-processor performance and a forecast of processor performance to 2020, based on the ITRS roadmap. A dashed line represents expectations if single-processor performance had continued its historical trend.

type Node_Kind is Enum < [#leaf, unary, binary] >;
...
for X => Root while X not null loop
  case X.Kind of
  [#leaf] =>
    Process_Leaf(X.Data);
  [#unary] =>
    Process_Unary(X.Data) ||
    continue loop with X => X.Operand;
  [#binary] =>
    Process_Binary(X.Data) ||
    continue loop with X => X.Left ||
    continue loop with X => X.Right;
  end case;
end loop;
Why and How to Formalize?

- Assertions help catch bugs sooner rather than later.
- Parallelism makes bugs much more expensive to find and fix.
  - Integrate assertions (annotations) into the syntax everywhere, as pre/postconditions, invariants, etc.
  - Compiler disallows potential race-conditions.
  - Compiler checks assertions, rejects the program if it can’t prove the assertions.
  - No run-time checking implies better performance, and no run-time exceptions to worry about.
Problems with conventional threads
(courtesy of FSU Prof. Ted Baker)

- Threads implicitly share access to global data
  - encourages undisciplined sharing
  - hides data flow within internal task logic

- Threads mix concerns that should be separable
  - semantics vs. performance

- Threads limit concurrency, ability to use more cores
  - hard coded

- Threads limit fine-grained concurrency
  - single thread of control, heavy weight
Conclusions and Ongoing Work

- It is productive to get rid of old baggage
- Can rethink, simplify, and unify
- Can focus on new issues
  - pervasive parallelism
  - new pointer-free data structuring approaches
  - full compile-time checking

- Ongoing work
  - Prototype ParaSail Compiler and Virtual machine released
  - Refining ParaSail language and implementation from experience and feedback

- Read the blog if you are interested...
  http://parasail-programming-language.blogspot.com
11 Cypress Drive
Burlington, MA 01803-4907

Tucker Taft
tucker.taft@sofcheck.com

http://parasail-programming-language.blogspot.com

+1 (781) 750-8068 x220