REFERENCE COUNTING:

Tired Old Technique, or Part of the Multi-Core Future?

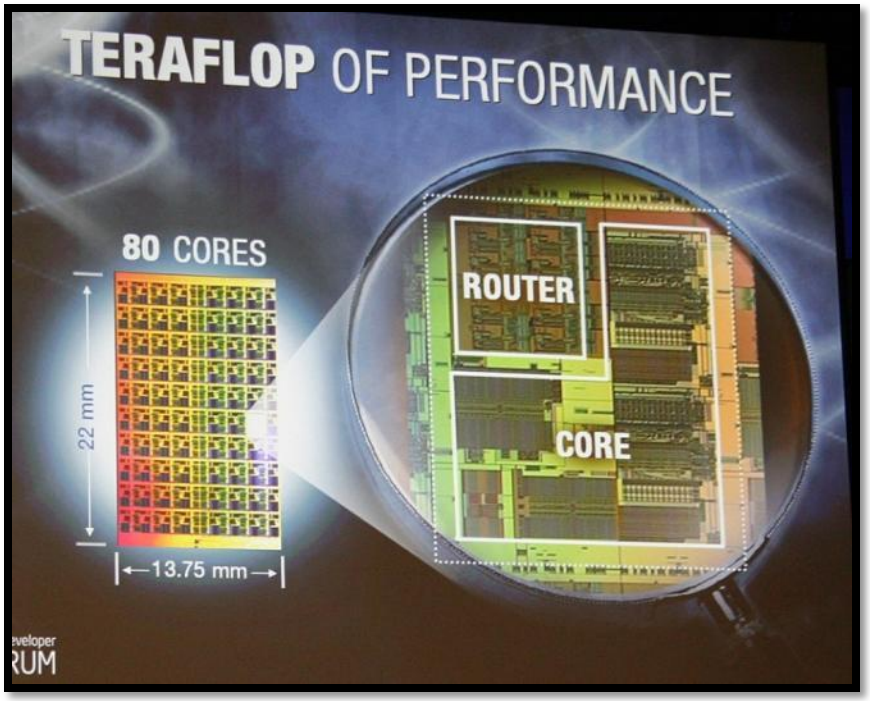
Kevin Hoffman, Purdue University
ISMM'09 WACI

Reference Counting

- The Good
  - Work not proportional to the size of the heap
  - Amenable to very low pause times
  - No global synchronization required in fast paths
- The Bad
  - Potentially high overhead
  - Cache line contention
  - Cyclical data structures

Changing Landscape of Computing

- Heterogeneous architectures
- NUMA becoming more mainstream
- Multicore processing



Wild and Crazy Idea

Non-deferred reference counting can be fast/scalable (?!?)

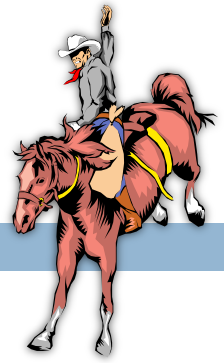
(faster than modern GCs in *some* cases???)

(1) All live data have positive reference counts (2) An object's reference count is zero when it becomes unreachable (3) A zero reference count implies that an object can be reclaimed

Wild and Crazy Idea

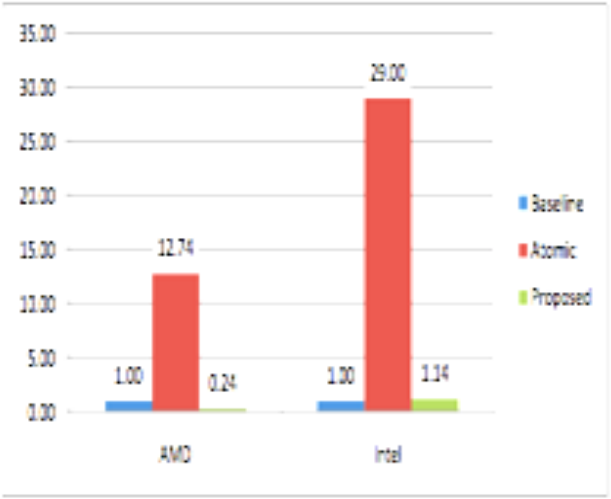
- Atomic instructions (CAS, etc.) are expensive
  - 4x to 32x slowdown on x86-64 (Intel Core 2 Duo, AMD Opteron 865)
  - Goal is to completely avoid them in the fast path
- Use thread-local cache of object reference counts
  - Inc heap refcount only the first time a thread acquires ref
  - Dec heap refcount only the last time a thread releases ref
- Organize thread-local refcounts into 2-level "page table"
  - Mitigates memory overhead so efficiency is bounded by number of objects touched by each thread
  - Can check/update thread-local refcount in ~8 instructions

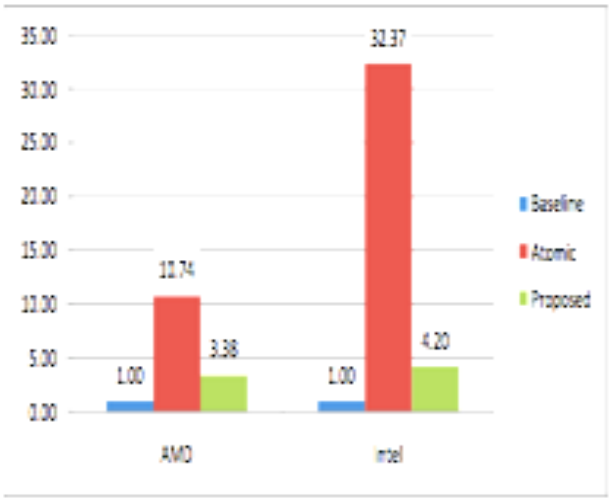
Wild and Crazy Idea

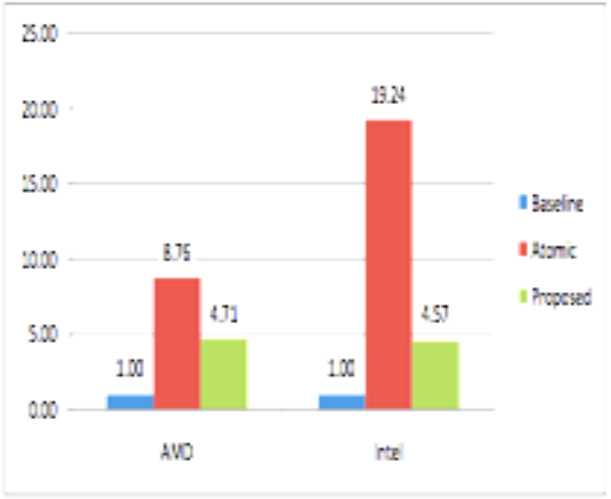


- Add inc/dec operations at the byte-code level
  - Ensures refcounting is not incorrectly modified by JIT opt.
  - Almost all bytecode produced by javac has a very nice single source, single use structure for the operand stack
    - Allows for refcount injection by inserting inc/ref calls at appropriate places (to be later inlined by the JIT)
    - Requires very careful consideration of protocols for references when calling and returning from methods, semantics related to exceptions; too many details to discuss here…
    - Facilitates removal of many pairs of inc/dec operations, such as when we know the same operand value will be incremented before and decremented after the lifetime of the operand

Very Preliminary Numbers







(Shows slowdown on AMD Opteron 865 1.8 Ghz, Intel Core 2 Duo 2.33Ghz)

- Reference count inc/dec injection on SPECjbb2000
  - 71,152 original instructions [660 field puts]
  - Injected 16,156 increments and 10,757 decrements
  - Optimized away 7,745 incs (48%) and 7,133 decs (66%)
  - Additional optimizations are possible (e.g., via [Joisha ISMM'07])