What is the memory model problem?

Memory Model
– Interface for programmer to reason about what values could be returned when a read is performed in a shared memory parallel program.
– Necessary for understanding the semantics of a shared memory parallel program.

What is the problem?
– ISA memory models and programming language memory models have evolved separately and in ad hoc ways.
– Even with significant work in the last 30 years on these problems, current solutions still have bugs, are difficult to understand, and have performance issues. (why we should care)
How should we evaluate memory models?

Programmability
– Easy to explain and teach to undergraduates.
– Should enforce no data races.
– Should enable the expression of "important parallel algorithms and patterns"
– Can multiple programming languages implement the memory model.

Portable Performance
– After all, that is one of the main reasons we do parallel computing in the first place.
– Is the model reasonably supported (efficient and inexpensive) by various computer architecture paradigms

Quick Review of Terminology

Sequential consistency as a memory model
– Possible parallel results can be determined by trying all possible sequential interleavings between instructions in threads. Instructions in a single thread must occur in same order in interleaving as they do in the thread.

Data Race
– When two instructions that could be executed in parallel read or write the same memory location and at least one of the instructions is a write
**Detailed Examples of the Problem**

Figure 1, Dekker’s algorithm

Figure 2, non-determinism in the hardware

Figure 4a and 4b, speculation combined with control and data dependences
   –They say 4a does not have a data race.

Figure 5, do we allow individual program optimizations?

---

**Current Solutions and Remaining Problems**

**Ada**
   –Pro: high-level programming model with support for shared memory parallel programming
   –Con: “did not fully formalize the notion of well-synchronized”

**Java**
   –Pro: threads in the language
   –Con: must deal with data races due to safety guarantees of the language
   –Con: in memory model a “future can determine whether the current access is legal”

**C/C++ and POSIX threads**
   –Pro: simpler than Java memory model, because can leave data races undefined
   –Con: Atomic keyword can break sequential consistency

**Data-race free**
   –Single-thread program optimizations must still be modified
   –Does not deal with data races
   –May not have efficient sequential consistency support in HW
   –Does not "eliminate atomicity violations or non-deterministic behavior"
Author Conclusions and Future Research

The shared memory programming model is important and should be supported with good memory models

– Hardware already supports it (e.g., cache coherence)
– Can pass references to complex data structures, which is much more efficient than copying
– Incremental parallelization is easier
– Do not have to explicitly distribute data structures.

Memory model development should be more disciplined

– Should move away from the test case only based development
– “disciplined shared-memory models”

System architecture and programming languages should enforce no data races

Need SW/HW co-design to successfully evolve and/or reinvent memory models

My Future Research Questions

Composition of programming models and memory models

– Do parallel programming models that we want to compose need to have the same underlying memory model?
– Can we develop memory models so they are composable?

Implementation details surrounding shared memory parallel programming

– Examples include synchronization constructs, atomic, shared, and volatile keywords.
– Is this the way we should be expressing these implementation details?
– Can we make implementation details such as these more orthogonal to the algorithm specification?