Parallel Computing on Uniform-Memory-Access Shared-Memory Architectures: Linear Speed-Ups for Complex Combinatorial Problems

Bernard M.E. Moret
Dept. of Computer Science
U. of New Mexico

Joint work with

David A. Bader
Dept. of Electrical and Computer Engineering
U. of New Mexico
Overview

1. Introduction
2. Parallel Computing To Date
3. Problems with Parallel Computing To Date
4. Shared-Memory Through Hardware
5. Uniform Memory Access in Shared-Memory
6. Advantages of UMA Shared-Memory
7. Linear Speed-Ups: An Example
8. Opportunities
9. Conclusions
Introduction

• Parallel computing has been with us for over 30 years (ILIAC)

• But it has been limited to “embarrassingly parallel” problems (e.g., Monte Carlo simulations) and highly regular computations (e.g., finite-element methods on meshes)

• Massively parallel machines are becoming affordable (512 processors for less than a million)

• But they still will not help much with complex optimization problems (NP-hard problems)
Parallel Computing To Date

- array- or mesh-based specialized engines (ILIAC, systolic computers)
- vector machines (Crays)—very limited form of parallelism
- very high-bandwidth interconnection networks with specialized topologies
  (Connection Machines, Ncube, Intel Paragon, Cray T3D)
- commodity-based (both CPU and network) clusters
- cluster of SMPs (symmetric multiprocessors) for higher-performance
  (Origin)

only SMP clusters have any shared memory
Problems with Parallel Computing To Data

- Topology: interconnecting large numbers of processors remains expensive (typically half the price of the machine)
- Routing: because the topology is not all-to-all, routing is crucial to performance; congestion is a frequent problem
- Latency: accessing nonlocal memory takes time measured in tens to hundreds of microseconds (nearly 3 orders of magnitude slower than local memory)
- Load-Balancing: all computations must be based on local memory, or tasks have to be migrated across processors, at huge expense.
- Fault-Tolerance: expensive to build in, since it requires multiple copies of data and thus more migration
Shared Memory Through Hardware

• Complex: requires very sophisticated approaches for maintaining cache coherence, a lot of hardware to establish all-to-all (or nearly so) connectivity.

• Promising: automatically fault-tolerant in terms of processor failure, automatically load-balanced, no communication at all.

• Problematic: SGI Origins and derivatives (ASCI Blue Mountain) have overall poor performance in terms of shared-memory—latencies for the “most distant” memory locations remain very high.

• Poor price to (raw) performance ratio: the extra hardware is expensive (not yet commodity items).
Uniform Memory Access (UMA) in Shared Memory

- Uniform Memory Access simply says that access from a processor to a memory location is the same for any (processor, location) pair.

- Allows programmer to ignore issues of locality (not possible on an Origin, for instance).

- Enables true shared-memory style of programming.

- Not doable with any reasonable performance through software simulation (as in various “virtual shared memory” systems, such as TreadMarks).

- Note that caching is still crucial to performance.
Characteristics of Memory Access in the Sun E10K
25 Years of PRAM Algorithms

- PRAM: Parallel Random Access Machine, a theoretical model of computation with perfect shared-memory (one memory access = one CPU computation) and perfect (and free) synchronization among processes.

- For these many years, algorithm designers have been publishing very parallel algorithms (linear speed-ups over very large ranges) for combinatorial problems.

- Yet only a tiny percentage has ever been used, because parallel machines to date take the equivalent of hundreds or thousands of operations to access one “distant” memory location.

- Many of computational biology’s most difficult problems rely on combinatorial problems for which we have good PRAM algorithms.
Message-Passing vs. Shared-Memory

- The dominant programming paradigm/environment for today’s parallel machines is message-passing, as embodied by MPI.
- In this model, computations are local and explicit communication (messages) are specified by the programmer to synchronize processes, exchange data, etc.
- In a true shared-memory model, the programmer only specifies what processor $i$ does—and all processors execute exactly the same program modified only by the parameter $i$.
- Thus shared-memory programming is much easier than message-passing programming.
- With the right machine (a fast UMA shared-memory architecture) it can also much faster.
Message-Passing vs. Shared Memory: (Ear Decomposition)
Linear Speed-ups
with PRAM Algorithms

• PRAM algorithms typically seek to achieve linear speed-ups (with a proportionality factor) over a huge range of processor numbers. Most algorithms scale perfectly to using one processor per data word.

• In a real machine, the number of processors is constant and synchronization must be achieved through software.

• We have found that we can achieve linear speed-ups for a large variety of basic discrete algorithms (sorting, basic graph algorithms, basic geometric algorithms).
Linear Speed-ups with PRAM Algorithms: (Ear Decomposition)
Linear Speed-ups
with PRAM Algorithms:
(Ear Decomposition)
Conclusions

- While expensive for now, the SUN E10K architecture offers true UMA shared-memory.
- UMA shared-memory machines can tackle problems that no other single machine can tackle (indeed that single workstations solve better than massive parallel machines).
- Problems abandoned over 20 years ago as too hard to solve can now be revisited.
- A revolution in high-performance computing may be at hand.
- Computational Biology stands to profit more than most other disciplines, most of its crucial problems are combinatorial in nature.