Models of Concurrency

David Chisnall

February 22, 2011
Computers are Slow!

- Buying two computers is cheaper than buying one that’s twice as fast
- Sometimes buying a faster processor is not even possible, but buying twice as many is
Trivial Concurrency

Multiple users, multiple computers
Collaboration Problem

Old problem:

- It takes one woman nine months to produce a baby.
- How many women do you need to produce a baby in one month?
Shared Memory

- Uniform Memory Architecture (UMA)
- Symmetric Multiprocessor (SMP)
- Duplicate the processors, nothing else
SMP Example

![Diagram of SMP Example](image-url)
Problems with SMP

- Memory controller can handle one memory request at a time
- Each CPU needs to wait for others to complete
- Memory starvation
- Cache coherency
Solution: Private Memory

- Non-Uniform Memory Architecture (NUMA)
- Each processor has private memory
- Access to other CPU’s memory is indirect and explicit
NUMA Example

The Problem

Hardware Models

Programmer Models

Memory Controller

Memory Controller

Memory Controller

Memory Controller

CPU

CPU

CPU

CPU

RAM

RAM

RAM

RAM
Emulating UMA with NUMA

- Done by most cluster operating systems and supercomputers
- Single global address space
- Remote memory fetched and cached as required
Problems with Emulating UMA

- Memory in different locations costs different amounts to access
- The programmer doesn't see this
- Reasoning about performance is very hard
Hidden in the Details

What’s this black line?

- HyperTransport / QuickPath Interconnect
- Infiniband
- Gigabit Ethernet
- The Internet
- Carrier Pigeons (see RFC1149)
INMOS Transputer
What Does the Programmer See?

• Shared memory?
• Independent Processes?
• Some combination?
Shared-Everything: Threads

- Private stack
- Everything else shared
Multithreaded Memory Layout

0xffffffff

- stack 1
- stack 2

Heap Memory

Library Code

Static Data

Program Code

0x00000000

0xffffffff

0xffffffff

0xfffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff

0xffffffff
Aside: Coroutines

- The `ucontext()` C library call allows you to create stacks
- And switch between them
- Concurrent programming style, but only one real thread: no true concurrency
Advantages of Threads

- Threads can share pointers
- Data structures are implicitly shared
- Easy to extend serial code to use threads
Disadvantages of Threads

- Everything is shared, but not always safe to access
- Hard to reason about cost of operations
- Hard to model interactions - any byte in memory is a potential interaction point
- Debugging code using threads is incredibly painful
The Opposite Extreme: Shared Nothing

- Separate processes
- Exchange data explicitly
- No implicit sharing
Message Passing

- Explicit data exchange
- Usually asynchronous
- Helps latency hiding
Latency Hiding

- Process sends a message
- Continues working
- Message reply arrives
Contrast with Distributed Shared Memory

- Attempts to access remote memory
- Waits for fetch to local cache
- Continues
Questions?