RETHINKING OPERATING SYSTEM DESIGNS FOR A MULTICORE WORLD

Ken Birman

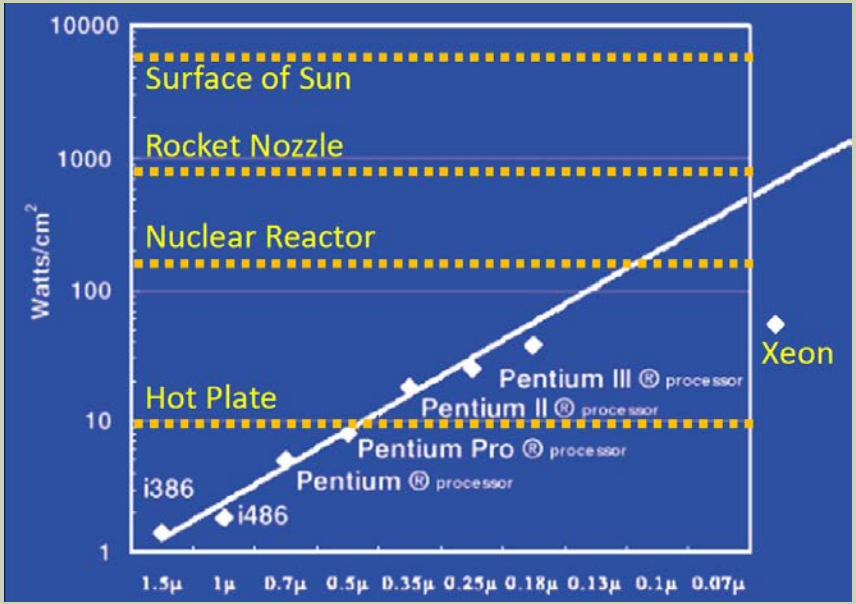
Based heavily on a slide set by Colin Ponce

THE RISE OF MULTICORE CPUS

- Multicore computer: A computer with more than one CPU.
  - 1960-1990: Multicore existed in mainframes and supercomputers.
  - 1990's: Introduction of commodity multicore servers.
  - 2000's: Multicores placed on personal computers.

- Soon: Everywhere except embedded systems?
  - But switched on and off based on need: each active core burns power
  - Debated: Will they be specialized cores (like GPUs, NetFPGA) or general purpose cores? Or perhaps both?

MULTICORE IS INESCAPABLE!



- Clearly, traditional speedup could not continue beyond 2005. or s0
- But we do need speedup or technology progress comes to a halt…

THE END OF THE GENERAL-PURPOSE UNIPROCESSOR

Intel Changes Plans for Pentium 4

Company drops Tejas and plans a dual-core desktop chip for 2005.

By Tom Krazit, IDG News Service and Tom Mainelli, PC World May 7, 2004 4:00 pm

Intel has canceled plans to produce its Tejas processor, the successor to today's Prescott-based Pentium 4 chip. Instead the company has moved up work on an as-yet-unnamed dual-core desktop processor it hopes to launch by 2005.

The surprise move shows Intel has embraced the idea that merely adding more megahertz to its processors is no longer the best way to boost performance, says Kevin Krewell, editor-in-chief of *Microprocessor Report*.

"Faster clocks speeds are just not the way to improve user experiences anymore," he says. "Tejas no longer made a lot of sense."

MULTICORE RESEARCH ISSUES

- The machines have become common, but in fact are mostly useful in one specific situation
  - Cloud computing virtualization benefits hugely from multicore
  - We end up with multiple VMs running side by side, maybe sharing read-only code pages (VM hardware ideally understands that these are “never dirty” and won’t suffer from false sharing). Each VM uses the same cores each time it becomes active (hence good affinity)
  - Offers a very good price/performance tradeoff to Google, Amazon
- But general purpose exploitation of multicore has been hard
  - So the machine on your desk might have 12 cores, yet rarely uses 2…

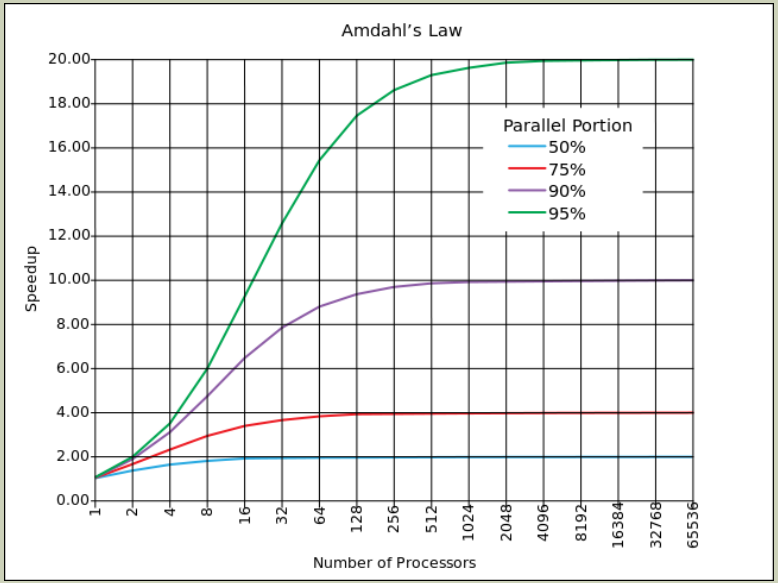
PUZZLE: IS MULTICORE USEFUL?

- To host multiple VMs concurrently, for sure.
  - Any modern multitenant data center exploits this feature
  - VMs “share nothing”, hence ideal for use with multicore servers
- But for general purpose programming, far less evident
  - We see this in mini-project 1: Leveraging multicore parallelism for speedup is very difficult. Slow-down is not uncommon!
  - Problem: any form of sharing seems to be an obstacle to speed
  - Even compilers have serious difficulty with modern hardware models.

BASIC CONCEPTS

- Memory Sharing Styles:
  - Uniform Memory Access (UMA)
  - Non-Uniform Memory Access (NUMA)
  - No Remote Memory Memory Access (NORMA)
- Cache Coherence
  - Many models: barrier, sequential, causal…
- Inter-Process (and inter-core) Communication
  - Shared Memory: At granularity of “cache line”
  - Message Passing: Implemented by OS but shapes what the h/w sees

WRITING PARALLEL PROGRAMS: AMDAHL'S LAW



N: Number of processors
B: Unavoidably sequential portion
T(n): Runtime with N processors

Speedup:

$$S(n) = \frac{T(1)}{T(n)} = \frac{T(1)}{T(1)\left(B + \frac{1}{n}(1-B)\right)} = \frac{1}{B + \frac{1}{n}(1-B)}$$

EXPLOITING PARALLEL PROCESSORS

- Experiment by Boyd-Wickizer et. al. on machine with four quad-core AMD Operton chips running Linux 2.6.25.
- *n* threads running on *n* cores:

```
id = getthreadid();
f = createfile(id);
while(True){
    f2 = dup(f);
    close(f2);
}
```

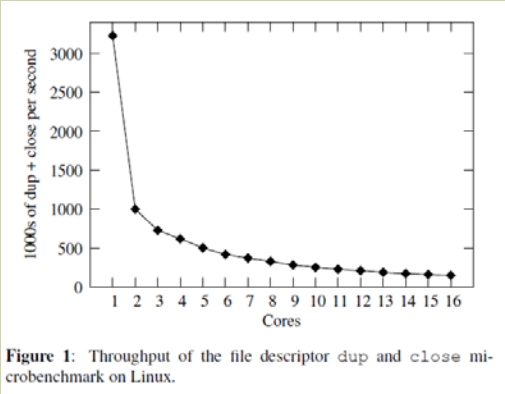


Figure 1: Throughput of the file descriptor `dup` and `close` microbenchmark on Linux.

- Looks embarassingly parallel… so it should scale well, right?

Boyd-Wickizer et. al., “Corey: An Operating System for Many Cores"

LINUX IS *NOT* GOOD AT MULTICORE!

- Application developer *could* provide the OS with hints:
  - Parallelization opportunities
  - Which data to share
  - Which messages to pass
  - Where to place data in memory
  - Which cores should handle a given thread
- Right now, this doesn't happen, except for "pin thread to core"
  - Should hints be architecture specific? What about GPU?

HINTS IN ACTION

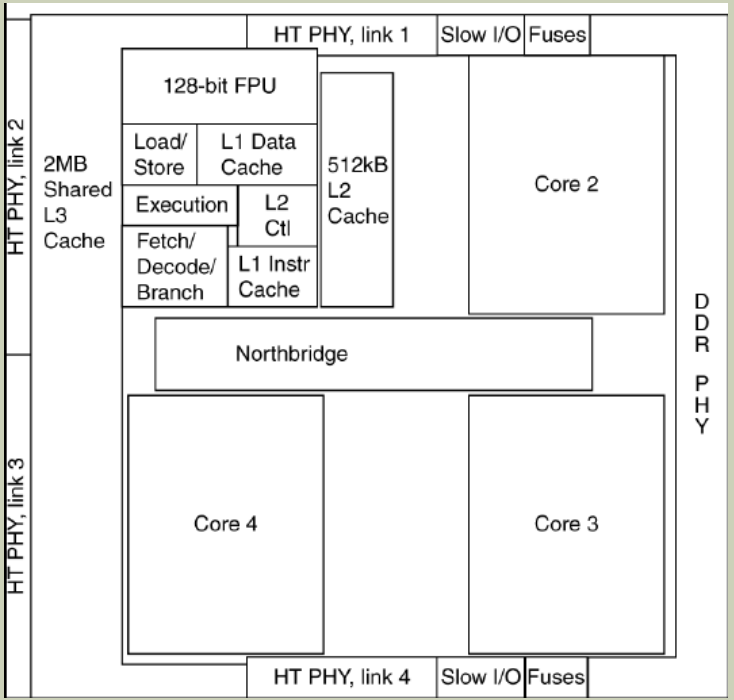
- Example: OpenMP (Open MultiProcessing)

```
#include <iostream>
using namespace std;

#include <omp.h>

int main(int argc, char *argv[])
{
  int th_id, nthreads;
  #pragma omp parallel private(th_id) shared(nthreads)
  {
    th_id = omp_get_thread_num();
    #pragma omp critical
    {
      cout << "Hello World from thread " << th_id << '\n';
    }
    #pragma omp barrier

    #pragma omp master
    {
      nthreads = omp_get_num_threads();
      cout << "There are " << nthreads << " threads" << '\n';
    }
  }

  return 0;
}
```

- Coded in C++ 11
- But the pragmas tell the compiler about intent
- Compiler can then optimize the code for parallelism / speed

IS IT ONE MACHINE? OR MANY?

- Modern machines often have several identical cores
  - But even with identical cores it isn't obvious how to think about these machines
  - Problem: Location of data very much shapes performance of computation on that data
- Here is a simple one-chip AMD 4-core design…

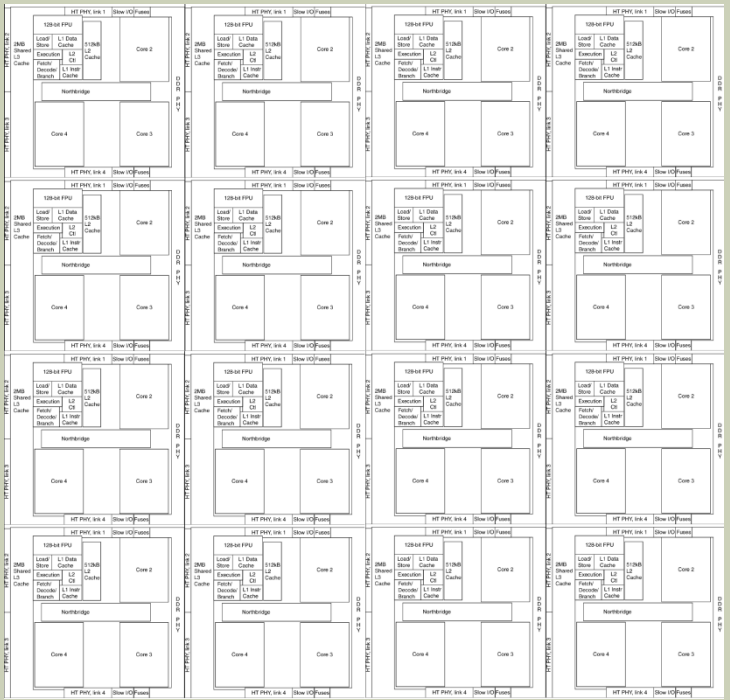


EVEN WITH IDENTICAL CORES…

- With multiple AMD chips in a multisocket CPU board, looks more and more like a distributed computer cluster!
- This illustrates a 16-core system, but looks just like a quad-computer system with each chip being a 4-core AMD processor

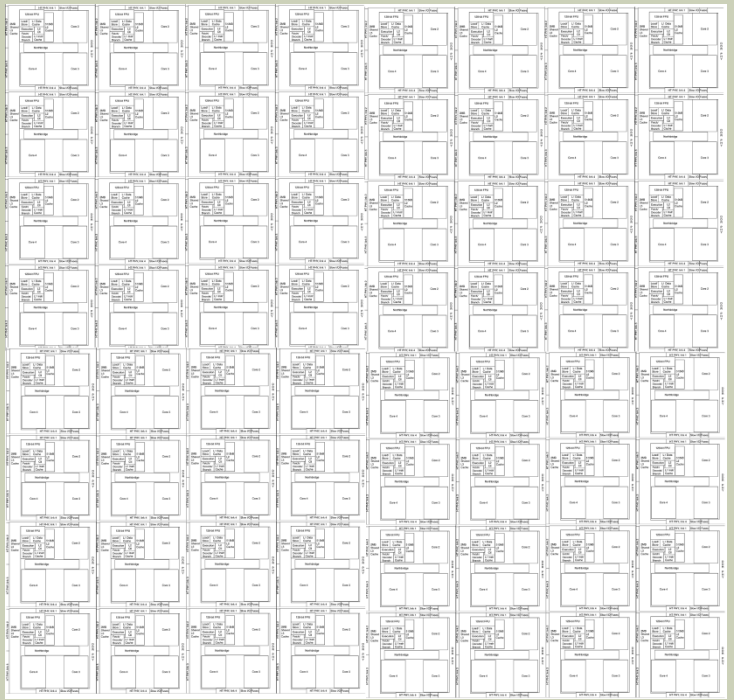
AMD 64-CORE CHIP

- AMD keeps pushing it to larger and larger scale…
- Like a cluster on a chip



AMD 256-CORE CHIP

- Will it ever end?
- Real puzzle: how to harness all the cores



CORE DIVERSITY

- More and more vendors are exploring specialized cores
  - GPU cores for high speed graphics
  - NetFPGA: devices that can process video streams or other streams of data on the network at optical line speeds
  - Computational geometry cores for manipulating complex objects
  - Scientific computing accelerators that offer special functions like DFFTs via hardware support: you load the data, the chip does the operation, and then the outcome is available on the other side
  - Some of these can support complex programs that run on the special processor, but use its own domain-specific programming style

TODAY'S PAPERS: TORNADO

- Context: Need to understand the state of play in late 1990's:
  - Ten years prior, memory was fast relative to the CPU. During the 90's, CPU speeds improved over 5x as quickly as memory speeds.
  - Over the course of the 90's, communication became a bottleneck.

- 1990 was prior to the full multicore revolution. But even in 1990 these issues were exacerbated in multicore systems.
  - Tornado developers saw this as a primary issue

Tornado: Maximizing Locality and Concurrency in a Shared Memory Multiprocessor Operating System"
Ben Gamsa, Orran Krieger, Jonathan Appavoo, Michael Stumm OSDI 1999

INITIAL OBSERVATIONS

- The hardware makes cross-core interactions transparent, but in fact the cost penalty is often high
  - Locking by threads is cheap if on same core, expensive cross-core
  - Memory sharing looks free, but in reality cache-line migration can be very costly (true sharing with writes is the big issue)
  - L2 cache will be cold if a thread is paused, then resumes on a different core than where it ran previously
- So Tornado tries to minimize these costly overheads

TORNADO

- Develops data structures and algorithms to minimize contention and cross-core communication. Intended for use with multicore servers.
- These optimizations are all achieved through replication and partitioning.
  - Clustered Objects
  - Protected Procedure Calls
  - New locking strategy

TORNADO: CLUSTERED OBJECTS

- OS treats memory in an object-oriented manner.
- Clustered objects are a form of object virtualization: the illusion of a single object, but actually composed of individual components spread across the cores called *representatives*.
  - One option is to simply replicate an object so that each core has a local copy, but can also partition functionality across representatives.
  - Exactly how the representatives function is up to the developer.
  - Representative functionality can even be changed dynamically.

TORNADO: PROTECTED PROCEDURE CALLS

- Primary use case: To support parallel client-server interactions.
- Idea is similar to that of clustered objects. Calls pass from a client task to a server task without leaving that core.
  - Benefits from *affinity:* hardware resources accessed by the collection of threads can live local to the core
  - In effect, the OS is structured in a way that matches what the hardware is already good at doing.
- By spreading server representatives over multiple cores, we get parallel speedup without cross-core contention delays

TORNADO: LOCKING

- Locks are kept internal to an object, limiting the scope of the lock to reduce cross-core contention.
- Locks can be partitioned by representative, allowing for optimizations involving mixed coarse and fine-grained uses.
- For intended use (Apache web server), very good match to need, although seems a bit peculiar and not very general…

… TEN YEARS PASSED

- Pollack’s Rule:
  - Thousand Core Chips: A Technology Perspective. Shekhar Borkar

- Pollack's Rule: **Performance** increase is roughly proportional to the square root of the increase in circuit complexity. This contrasts with **power consumption** increase, which is roughly linearly proportional to the increase in complexity

- Implication: Many small cores instead of a few large cores.

BARRELFISH

- A completely new OS, built from scratch that
  - Views multicore machines as networked, distributed systems.
  - *No inter-core communication* except through message-passing.
  - Core OS seeks to be as hardware-neutral as possible, with per-architecture adaptors treated much like device drivers.
  - Replicates entire application state across cores: everything is local.

- In effect, Barrelfish choses not to use features of the chip that might be very slow.

The Multikernel: A new OS architecture for scalable multicore systems. Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schaupbach, Akhilesh Singhania. SOSP 2009

THE WAY OF BARRELFISH

- Presumes that in fact, cores will be increasingly diverse
  - A small data center on a chip, with specialized computers that play roles on behalf of general computers
- And also assumes the goal is really research
  - Not clear that Barrelfish intends to be a real OS people will use
  - More of a prototype to explore architecture choices and impact
- “How fast can we make a multicomputer run”?

THE WAY OF BARRELFISH

- … so, Barrelfish
  - Starts with a view much like that of a virtual computing system
  - Lots of completely distinct VMs. Obvious fit for multicore
- But then offers a more integrated set of OS features
  - So we can actually treat the Barrelfish as a single machine
- And these center on ultrafast communication across cores
  - Not shared memory, but messages passed over channels

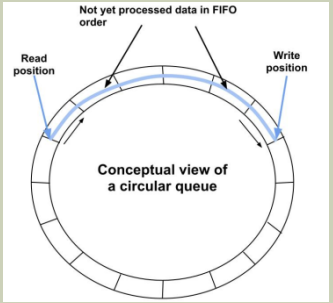
BARRELFISH MESSAGE PASSING

- This is the *only* way for separate cores to communicate.

- Advantages:
  - Cache coherence protocols look like message passing anyways, just harder to reason about.
  - Eases asynchronous application development.
  - Enables rigorous, theoretical reasoning about communication through tools like π-calculus.

HOW IT WORKS

- They design a highly asynchronous message-queue protocol
  - We'll see it again in a few weeks when we discuss RDMA
  - The Barrelfish version is circular



- Basically
  - Wait for a slot in the circular queue to some other processor
  - Drop your message into that slot, and done (no cross-core lock used)
  - Request/reply: You include a synchronization token, and reply will eventually turn up, and wake up your thread

THE MULTIKERNEL

- Operating system state (and potentially application state) is automatically replicated across cores as necessary.
- OS state, in reality, may be a bit different from core to core depending on needs, but that is behind the scenes.
  - Reduces load on system interconnect and contention for memory.
  - Allows us to specialize data structures on a core to its needs.
  - Makes the system robust to architecture changes, failures, etc.
- Claim: Enables Barrelfish to leverage distributed systems research (like Isis2 ☺, although this has never been tried).

ATTEMPT TO BE HARDWARE NEUTRAL

- Separate the OS as much as possible from the hardware. Only two aspects of the OS deal with specific architectures:
  - Interface to hardware
  - Message transport mechanisms (needed for GPUs)
- Advantages:
  - Facilitates adapting an OS to new hardware: “device driver”.
  - Allows easy and dynamic hardware- and situation-dependent message passing optimizations.
- Limitation:
  - Treats specialized processors like general purpose ones…
  - Future world of NetFPGA devices “on the wire” would be problematic

SUMMARY

- Multicore computers are here!
- They work *really well* in multitenant data centers (Amazon)
- But less well for general purpose computing
  - Our standard style of coding may be the real culprit
  - Seems like pipelines of asynchronous tasks are a better fit to the properties of the hardware, but many existing OS features are completely agnostic and allow any desired style of coding, including styles that will be very inefficient