Operating Systems in a Multicore World

Colin Ponce
CS 6210

Professor: Ken Birman

September 3, 2012

The Rise of the Multicore

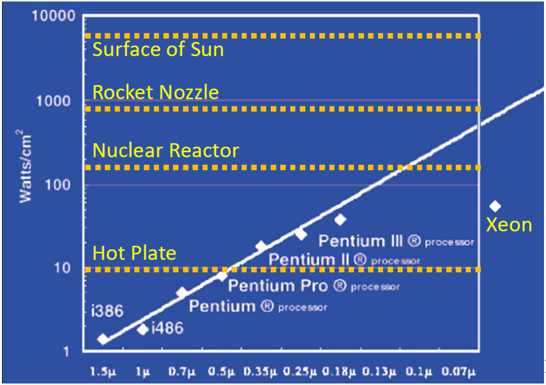
Multicore computer: A computer with more than one CPU.

- 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?

Why Multicore Research Matters

- Multicore machines have been getting more and more general.
- New types of computation introduced at each step of generality.
- Very thorny issues!
- Why deal with these issues?

The Power Wall



[1]

Making computers more powerful also means they run hotter. Can reduce power by reducing voltage, but can only go so far.

[1] Image by Hakim Weatherspoon, CS 3410 Lecture 23

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." [2]

[2] http://www.pcworld.com/article/116053/intel_changes_plans_for_pentium_4.html

Basic Multicore Concepts

Memory Sharing Styles:

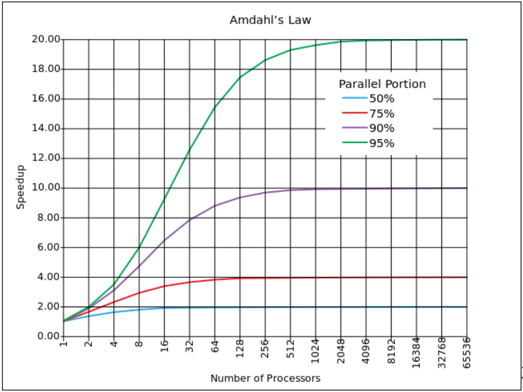
- Uniform Memory Access (UMA)
- Non-Uniform Memory Access (NUMA)
- No Remote Memory Memory Access (NORMA)

Cache Coherence

Inter-Process (and inter-core) Communication

- Shared Memory
- Message Passing

Writing Parallel Programs: Amdahl's Law



[3]

Speedup given by parallel computation: $\frac{1}{(1-P)+\frac{P}{S}}$

[3] http://en.wikipedia.org/wiki/File:AmdahlsLaw.svg

Using Parallel Hardware

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 = get_thread_id();
f = create_file(id);
while(True){
    f2 = dup(f);
    close(f2);
}
```

Using Parallel Hardware

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 = get_thread_id();
f = create_file(id);
while(True){
    f2 = dup(f);
    close(f2);
}
```

Embarrasingly parallel, so it'll scale well, right?

Catastrophe!

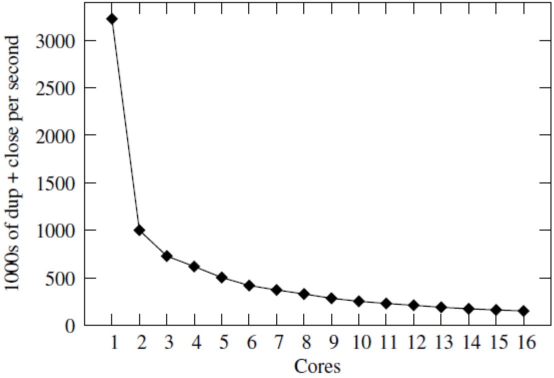


Figure 1: Throughput of the file descriptor `dup` and `close` microbenchmark on Linux. [4]

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

Viewpoint: Hints from the application developer?

Application developer could provide the OS with hints on

- Parallelization opportunities
- Which data to share
- Which messages to pass
- Where to place data in memory
- Which cores should handle a given thread

Should hints be architecture specific?

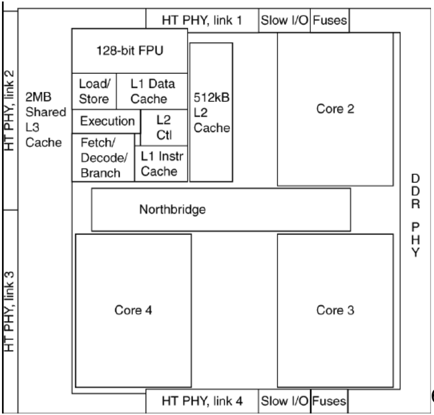
Viewpoint: Hints from the application developer?

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;
}
```
[5]

[5] http://en.wikipedia.org/wiki/OpenMP

Viewpoint: A Single or Distributed System?



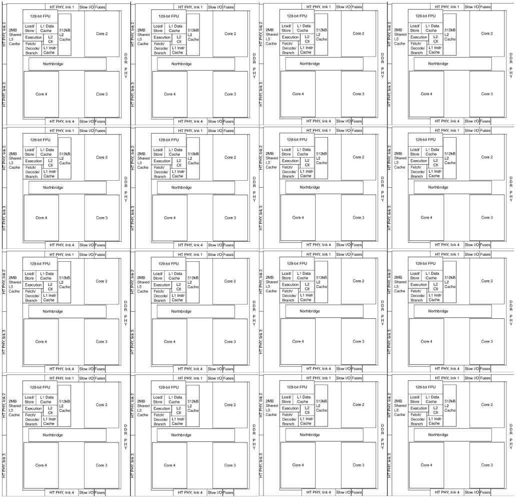
[6]

Figure : AMD Barcelona Quad-core, ca 2007

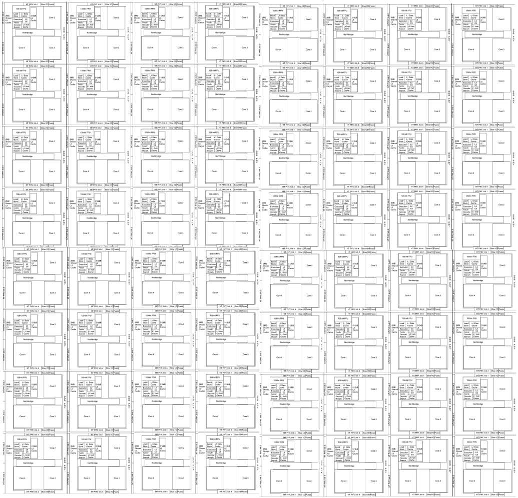
[6] Image by Hakim Weatherspoon, CS 3410 Lecture 23

Viewpoint: A Single or Distributed System?

Viewpoint: A Single or Distributed System?



Viewpoint: A Single or Distributed System?



Tornado

"Tornado: Maximizing Locality and Concurrency in a Shared Memory Multiprocessor Operating System"

Ben Gamsa, Orran Krieger, Jonathan Appavoo, Michael Stumm

OSDI 1999

State of Multicore in Late 90'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 started to become a bottleneck.

These problems are exacerbated in multicore systems.

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: present the illusion of a single object, but is actually composed of individual components spread across the cores called *representatives*.

Tornado: Clustered Objects

OS treats memory in an object-oriented manner.

Clustered objects are a form of object virtualization: present the illusion of a single object, but is actually composed of individual components spread across the cores called *representatives*.

Can simply replicate an object, but can also partition functionality across the representatives.

Tornado: Clustered Objects

OS treats memory in an object-oriented manner.

Clustered objects are a form of object virtualization: present the illusion of a single object, but is actually composed of individual components spread across the cores called *representatives*.

Can simply replicate an object, but can also partition functionality across the representatives.

Exactly how the representatives function is up to the developer.

Representative functionality can even be changed dynamically.

Tornado: Protected Procedure Calls

Applicable during client-server interactions.

Tornado: Protected Procedure Calls

Applicable during client-server interactions.

Virtualized similarly to clustered objects. Calls pass from a client task to a server task without leaving that core.

So a server also has representatives on each core.

Tornado: Locks

Locks are kept internal to an object, limiting the scope of the lock to reduce contention.

Tornado: Locks

Locks are kept internal to an object, limiting the scope of the lock to reduce contention.

Locks can be partitioned by representative, allowing for optimization.

Tornado: Discussion

Single vs Distributed System?

Tornado: Discussion

Single vs Distributed System?

Demands on application developer?

Ten Years Later: 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 complexity.

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

Barrelfish

"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 Schüpbach, and Akhilesh Singhania

SOSP 2009

Barrelfish

- View multicore machines as networked, distributed systems.
- No inter-core communication except through message-passing.
- Create hardware-netural OS.
- Replicate state across cores.

Barrelfish: A Networked System

Architectures are becoming increasingly diverse. Many arrangements are possible in terms of

- Number of cores.
- Sharing of memory and caches.
- Types of cores within a system.

Especially with many-core systems, too difficult to view them as single entities.

So view as distributed systems.

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.

Barrelfish: A hardware-neutral OS

Separate the OS as much as possible from the hardware. Only two aspects deal with specific architectures:

- Interface to hardware
- Message transport mechanisms

Advantages:

- Facilititates adapting an OS to new hardware, as there is only a narrow bridge between OS and hardware.
- Allows easy and dynamic hardware- and situation-dependent message passing optimizations.

Barrelfish: The multikernel

Operating system state (and potentially application state) is to be 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.
- Can leverage distributed systems research.

Barrelfish: Discussion

Sufficiently optimizable?

Burden on the developer?

Thank you!