Lecture 12: Putting It All Together

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Today's Lecture

• What to do with all what you heard about in this course?
• Challenges in hardware and how they affect the software
• Different type of parallelism
• Common problems in parallel programs
• Tools
Keep The Big Picture in Mind
Which Programming Model/Language?

Application & Platform Specific Optimization

Architecture-Specific Optimization

Performance Evaluation

Debugging

Design Points
-- Performance
-- Reliability
-- Availability
-- Cost
-- Power
-- Time to Market
How to choose your programming language/model given an application?
How to Choose?

• Your level of expertise
  – Less experience = higher abstraction
• The concurrency in the application
  – What type of parallelism we have?
• What can you do with the hardware?
• How does the platform interact with you?
• Are you starting from scratch? or from a sequential version of the program?
Today’s Lecture

• What to do with all what you heard about in this course?
• Challenges in hardware and how they affect the software
• Different type of parallelism
• Common problems in parallel programs
Challenges

- Power-Aware Software?
- Reliability-Aware Software
Cooking Aware Computing
Power-Aware Computing

• **Definition:** reducing power without loosing performance

• **Must deal with:**
  – dynamic power
  – static power
  – temperature
What To Do About Dynamic Power

- Stop and go
- DFVS (Dynamic Frequency and Voltage Scaling)
  - At OS level
    - idle time represents energy waste
    - deadlines for interactive programs
  - Offline compiler analysis
    - insert mode-set instructions
    - depends on program phases
    - lowers the voltage for memory-bound sections
  - Online dynamic compiler analysis
    - phase detection
    - binary instrumentation
- Reducing switching activity
Global power/performance data

Power/performance data

Power mode

• Power budget
• High level scheduling

System Software

Global Power Management

Core

Core

Core
Other Techniques for the Multicore

• Migration
  – Moving threads among cores
  – Timescale of order of a millisecond, much slower than DVFS
  – Migration can be used with DVFS

• Migrate critical thread
  – Measure criticality with heat sensor
  – Or with cache misses as a proxy
As A Programmer

• Try to use less-expensive operations (i.e. help the compiler)
• Locality to help caches
• Control DVFS
  – Advanced Configuration and Power Interface (ACPI)
Tools for Programmers

- lm-sensors: http://www.lm-sensors.org/
- To access performance counters: http://icl.cs.utk.edu/papi/overview/index.html
- https://01.org/powertop/
- Intel Power Governor: http://software.intel.com/en-us/articles/intel-power-governor
ACPI Tables
ACPI BIOS
ACPI Registers

Kernel
Device
Driver

Applications

OS Dependent Application APIs

OSPM System Code

ACPI Driver/AML Interpreter

ACPI Registers
ACPI BIOS
ACPI Tables

Platform Hardware

BIOS

Existing industry standard register interfaces to: CMOS, PIC, PITs, ...

ACPI BIOS Interface

ACPI Table Interface

ACPI Register Interface

OS Specific technologies, interfaces, and code.

OS Independent technologies, interfaces, code, and hardware.

- ACPI Spec Covers this area.
- Hardware/Platform
- OS specific technology
- Provided by ACPI CA
Reliability

• Transistors are becoming unreliable
• What will your application do if a core becomes unavailable?
• Can you duplicate some computations for very critical operations?
# define _GNU_SOURCE
#include <pthread.h>

int pthread_setaffinity_np (pthread_t tid, size_t cpusetsize, cpu_set_t *mask);
int pthread_getaffinity_np (pthread_t tid, size_t cpusetsize, cpu_set_t *mask);

You can get it from:
pid_t getpid (void);
or
pid_t gettid (void);

Data structure representing a set of CPUs
typically sizeof(cpu_set_t)

Assigning a thread to a CPU
Important: The OS can override this!
Today’s Lecture

• What to do with all what you heard about in this course?
• Challenges in hardware and how they affect the software
• Different type of parallelism
• Common problems in parallel programs
A Glimpse at Another Type of Parallelism: SIMD/SPMD/STMD

Figure 1.1. Enlarging Performance Gap between GPUs and CPUs.

Courtesy: John Owens

16 highly threaded SM's, >128 FPU's, 367 GFLOPS, 768 MB DRAM, 86.4 GB/S Mem BW, 4GB/S BW to CPU
A Glimpse at A Typical GPU
A Glimpse at A Typical GPU

SPs within SM share control logic and instruction cache

Streaming Processor (SP)

Host

Input Assembler

Thread Execution Manager

Parallel Data Cache

Parallel Data Cache

Parallel Data Cache

Parallel Data Cache

Parallel Data Cache

Parallel Data Cache

Parallel Data Cache

Parallel Data Cache

Texture

Texture

Texture

Texture

Texture

Texture

Texture

Texture

Load/store

Load/store

Load/store

Load/store

Load/store

Load/store

Load/store

Global Memory
A Glimpse at A Typical GPU

- Much higher bandwidth than typical system memory
- A bit slower than typical system memory
- Communication between GPU memory and system memory is slow
BUT …

- GPU is not standalone, it needs CPU
- How to divide your program among multicore and GPU?
- Can you put part of your program into STMD/SPMD/SIMD?
- GPU $\rightarrow$ OpenCL and CUDA
Today’s Lecture

• What to do with all what you heard about in this course?
• Challenges in hardware and how they affect the software
• Different type of parallelism
• Common problems in parallel programs
Common problems in parallel programs

- Parallel programs are subject to the usual bugs
- Plus: new timing and synchronization bugs (race condition, deadlocks, livelocks, ...)
- Parallel bugs often disappear when you add code to try to identify the bug 😞
Common problems in parallel programs: Too Many Threads

- (Fixed amount of work)/(large number of threads) $\Rightarrow$ each thread will do too little + the overhead of threading

- overhead of threading:
  - starting ending threads
  - contention on shared resources $\Rightarrow$ especially when software threads are more than hardware threads
Common problems in parallel programs: Too Many Threads

- The best strategy: limit the number of software threads to:
  - Number of hardware threads
  - Number of outer-level caches
- Also, separate your threads into I/O threads and compute threads
  - Blocked threads are not fighting for time-slice by the OS
- OpenMP takes this burden from the programmer
- Thread pool is another option (no standard one for POSIX threads though)
Common problems in parallel programs: Data Race, Deadlocks, and Live Locks

• Sometimes races are hidden by the language syntax (What may seem like a single instruction may actually be several ones.)

• Several ways to deal with that:
  – Use tools like Intel Thread checker
  – Locks
  – Transactions (which may go down to locks in some implementations!)
Common problems in parallel programs: Data Race, Deadlocks, and Live Locks

- Locks can lead to deadlocks
- Deadlocks occur when the following 4 conditions exist:
  - Access to each resource is exclusive
  - A thread is allowed to hold one resource while requesting another
  - No thread is willing to relinquish a resource it has acquired
  - There is a cycle of threads trying to acquire resources
- Deadlocks can be avoided by breaking anyone of the above four conditions, for example:
  - Replicate a resource if possible
  - Always let threads acquire locks (resources) in the order
Common problems in parallel programs: Heavily Contended Locks

- Locks can become contended → performance degradation

- What to do?
  - Replicating the resource (hence spreading the contention) can help.
  - Consider partitioning the resource and use locks for each part
Conclusions

• Multicore and manycore processors is a work in progress → new techniques/developments almost daily
• No magical recipe → must keep track of the software and hardware
Thank You!