CSCI-GA.3033-009
Multicore Processors:
Architecture & Programming

Lecture 10: 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 do 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 losing 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
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:**
  https://wiki.archlinux.org/index.php/lm_sensors

• **To access performance counters:**
  http://icl.cs.utk.edu/papi/overview/index.html
Reliability

- Transistors are becoming unreliable
- What will your application do if a core becomes unavailable?
- Can you duplicate some computations for very critical operations?
Tools At Your Service!

```c
#define _GNU_SOURCE
#include <pthread.h>

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

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

Streaming Multiprocessor (SM)
A Glimpse at A Typical GPU

SPs within SM share control logic and instruction cache
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) → each thread will do too little + the overhead of threading

- overhead of threading:
  - starting ending threads
  - contention on shared resources → 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
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
Looking Ahead
How Will the Future Look Like?

• "I think there is a world market for maybe five computers."
  - Thomas Watson, chairman of IBM, 1949

• "There is no reason in the world anyone would want a computer in their home. No reason."
  - Ken Olsen, Chairman, DEC, 1977

• "640K of RAM ought to be enough for anybody."
  - Bill Gates, 1981

Predicting the Future is not easy!!
How Will the Future Look Like?

- **Evolution**: just interpolating the current trend
- **Revolution**: a new technology, a paradigm shift → very hard to predict!
The Triangle

• Only the **PROGRAMMER** knows the **ALGORITHM**
  – Pragmas
  – Pointer chasing
  – Partition code, data

• Only the **COMPILER** knows the future (sort of ??)
  – Predication
  – Prefetch/Poststore
  – Block-structured ISA

• Only the **HARDWARE** knows the past
  – Branch directions
  – Cache misses
  – Functional unit latency
Looking Ahead: The Software
The Future of Software: Evolution

• Application types
• Languages
• MPI for multicore?
• Auto-parallelization
What Kind of Apps Need 100s Cores?

• “Who needs 100 cores to run M/S Word?”
  – Need compelling apps that use 100s of cores

• Compelling in terms of likely market or social impact, with longer term potential
Example of Evolving Applications

• Deep learning
• Health-record management
• NLP
• Teleconferences
• Heavy multimedia contents
• More realistic graphics and user interface
• Event-driven (real-time)
• ...

The Future of Software: Evolution

- Application types
- Languages
- MPI for multicore?
- Auto-parallelization
Evolving Languages

- Scripting
- Domain specific languages (e.g. StreamIt, ...)

boost productivity, enable faster development and rapid prototyping

programmability

domain specific optimizations
simple and effective optimizations for domain specific abstractions

enable parallel execution
target tiled architectures, clusters, DSPs, multicores, graphics processors, ...

The Future of Software: Evolution

• Application types
• Languages
• MPI for multicore?
• Auto-parallelization
MPI on Multicore

• One MPI process per core
  – Each MPI process is a single thread

• One MPI process per node
  – MPI processes are multithreaded
  – One thread per core
  – aka Hybrid model
The Future of Software: Evolution

- Application types
- Languages
- MPI for multicore?
- Auto-parallelization
Improvement in Automatic Parallelization

- Compiling for Instruction Level Parallelism
- Prevalence of type unsafe languages and complex data structures (C, C++)
- Automatic Parallelizing Compilers for FORTRAN
- Typesafe languages (Java, C#)
- Demand driven by Multicores?

Source: Saman Amarasinghe, MIT
What Do We Need From the Compiler?

- Compilers are critical in reducing the burden on programmers
  - Identification of data parallel loops can be easily automated, but many current systems require the programmer to do it.
- Reviving the push for automatic parallelization
  - Best case: totally automated parallelization hidden from the user
  - Worst case: simplify the task of the programmer
How Can the Compiler Parallelize a Program?

- Find the dependency in the program
- Try to avoid or eliminate the dependency
- Reduce overhead cost
Challenges of Auto-Parallelization

• There are some established forms that the compiler can detect and deal with.
• In complicated programs, these forms may be scarce.
• Compiler always chooses to be conservative.
The Future of Software: Revolution

• Program design methodology
  – Sketching? → given a specification, synthesize a program meeting this spec

• New programming paradigm

sketch  program = completed sketch
The Hardware
The Future of Hardware: Evolution

• More cores on-chip but constraints increase
  – Dark-silicon becomes more substantial
  – Wire delay
  – Power
  – Reliability

• More widespread heterogeneous multicore
Symmetric-multicore alone will not sustain the multicore era.

Symmetric Multicore Projections

- Target: 18x in 10 years
- Symmetric: 3.4x in 10 years

Speedup vs Year
Why Diminishing Returns?

- Transistor area is still scaling
- Voltage and capacitance scaling have slowed
- Result: designs are power, not area, limited
What belongs in multicore model?

Styles
- Number of Threads, Cache Sizes

Architectures
- Cache & memory latencies, memory bandwidth

Topologies
- Area & Power Budget

Pareto Frontiers
- Area & Power / Performance Tradeoffs

Applications
- PARSEC, Data Use
Dark Silicon

Sources of Dark Silicon:
Power + Limited Parallelism
Wire Delay

• Communication is now more expensive than computation.
• Wire delay became more relevant than before.
• We cannot avoid interaction among cores
Multicore Scaling Trends in Terms of Interconnect

**Today**

- A few large cores on each chip
- Diminishing returns prevent cores from getting more complex
- Only option for future scaling is to add more cores
- Still some shared global structures: bus, L2 caches

**Tomorrow**

- 100’s to 1000’s of simpler cores [S. Borkar, Intel, 2007]
- Simple cores are more power and area efficient
- Global structures do not scale; all resources must be distributed

Source: Jason Miller slides from MIT Carbon Research Group
Can Interconnect Improve Programmability?

A cheap broadcast communication mechanism can make programming easier

– Enables convenient programming models (e.g., shared memory)
– Reduces the need to carefully manage locality
The Future of Hardware: Revolution

- Brain-inspired machines
- Reconfigurable processors
- DNA computing
- Quantum computing
- ...
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
• Heterogeneous computing will be the norm very soon!
Thank You!