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

Lecture 11: Looking Ahead

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

- Content-based image retrieval
- 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, ...)

**Programmability**
- Boost productivity, enable faster development and rapid prototyping
- 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
Message Passing Interface (MPI)

- Message Passing Model
- Allows communication between processes (threads) using specific message-passing system calls.
- Shared data is communicated through messages
- Does not assume shared memory
Message Passing Interface (MPI)

- Allows for asynchronous events
- Does not require programmer to write in terms of loop-level parallelism
- Operates on multicore AND is scalable to very large systems
- Extremely flexible

Why then it was not adopted for multicore as the de facto?
Message Passing Interface (MPI)

- Considered extremely difficult to write
  - Shared-memory models seem more intuitive
- Difficult to incrementally increase parallelism
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
MPICH Features to Support Multicore Architecture

- Shared memory
- Network

Process 0
Process 1
Process 2
Node 0
Node 1
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
• In the next few slides we will see some examples.
Dependence Elimination and Avoidance

• A data dependence between two sections of a program indicates that during execution those two sections of code must be run in certain order
  • anti dependence: READ before WRITE
  • flow dependence: WRITE before READ
  • output dependence: WRITE before WRITE
Data Privatization and Expansion

- Data privatization can remove anti and output dependences.
- These dependences are not due to computation having to wait for data values produced by others. Instead, it waits because it wants to assign a value to a variable that is still in use.
- The basic idea is to use a new storage location so that the new assignment does not overwrite the old value too soon.
Data Privatization and Expansion

- Example:

```plaintext
DO i=1,n
  t = A(i)+B(i)
  C(i) = t + t**2
ENDDO

PARALLEL DO i=1,n
  PRIVATE t
  t = A(i)+B(i)
  C(i) = t + t**2
ENDDO

PARALLEL DO i=1,n
  T(i) = A(i)+B(i)
  C(i) = T(i) + T(i)**2
ENDDO
```
Parallel Loop Restructuring

• A parallel computation usually incurs an overhead when starting and terminating
• The larger the computation in the loop, the better this overhead can be amortized
• Use techniques such as loop fusion, loop coalescing, and loop interchange to optimize performance
Challenges of Auto-Parallelization

• There are some established forms that the compiler can detect and deal with (as we saw in the previous few slides).

• 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
Is increasing number of cores enough?

We have relied on multicore scaling for several years.

How much longer will it be our primary performance scaling technique?

Source: The Dark Silicon Implications for Microprocessors by Karu Sankaralingam et. al.
Symmetric multicore projections show a target speedup of 18x in 10 years, while symmetric multicore projections show a speedup of 3.4x in the same period. This indicates that symmetric multicore processors alone will not sustain the multicore era.
Why Diminishing Returns?

- Transistor area is still scaling
- Voltage and capacitance scaling have slowed
- Result: designs are power, not area, limited
### Device Scaling Projections

From 45 nm to 8 nm:

<table>
<thead>
<tr>
<th></th>
<th>Conservative</th>
<th>Optimistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>32x ↓</td>
<td>32x ↓</td>
</tr>
<tr>
<td>Power</td>
<td>4.5x ↓</td>
<td>8.3x ↓</td>
</tr>
<tr>
<td>Frequency</td>
<td>1.3x ↑</td>
<td>3.9x ↑</td>
</tr>
</tbody>
</table>

[Borkar 2007]    [ITRS 2010]
What belongs in multicore model?

Styles

Number of Threads, Cache Sizes

Topologies

Area & Power Budget

Pareto Frontiers

Area & Power / Performance Tradeoffs

Architectures

Cache & memory latencies, memory bandwidth

Applications

PARSEC, Data Use
Dark Silicon

Sources of Dark Silicon: Power + Limited Parallelism

At 22 nm: 17%, 26%
At 8 nm: 51%, 71%

Conservative
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
Current Multicore Communication Trend

Point-to-Point Mesh Network

Examples: MIT Raw, Tilera TILEPro64, Intel Terascale Prototype

- Neighboring tiles are connected
- Distributed communication resources
- Non-uniform costs:
  - Latency depends on distance
  - Encourages direct communication
- More energy efficient than bus
- Scalable to hundreds of cores
Effect of Interconnect on Multicore Programming Trends

Meshes and small cores solve the physical scaling challenge, but programming remains a barrier

Parallelizing applications to thousands of cores is hard

- Task and data partitioning
- Communication becomes critical as latencies increase
- Increasing contention for distant communication
  - Degraded performance, higher energy
  - Inefficient broadcast-style communication
    - Major source of contention
    - Expensive to distribute signal electrically
Effect of Interconnect on Multicore Programming Trends

For high performance, communication and locality must be managed

• Tasks and data must be both partitioned and placed
  – Analyze communication patterns to minimize latencies
  – Place data near the code that needs it most
  – Place certain code near critical resources (e.g. DRAM, I/O)

• Dynamic, unpredictable communication is impossible to optimize

• Orchestrating communication and locality increases programming difficulty exponentially
Can Interconnect Improve Programmability?

Observations:

- A cheap broadcast communication mechanism can make programming easier
  - Enables convenient programming models (e.g., shared memory)
  - Reduces the need to carefully manage locality

- On-chip optical components enable cheap, energy-efficient broadcast
**Suggestion**

Chemical Broadcast WDM Interconnect

Electrical Mesh Interconnect

ATAC Interconnect

from MIT Carbon research group.
The Future of Hardware: Revolution

- Brain-inspired machines
- Reconfigurable processors
- DNA computing
- Quantum computing
- Non Von-Neumann model
- ...

Conclusions

• Increasing the number of on-chip cores leads to many challenges both in software and hardware.
• Do we really need to increase the number of on-chip cores?
• What if we can no longer increase the number of on-chip cores?