Multi-core Programming

Jay Swaminathan
Senior Application Engineer
Software Solutions Group
Intel
Topics

- **CPU (semiconductor) History**
  - Moore’s Law & Multi-Core
  - Transistor scaling & its impact
  - What then? The new era...

- **Software Parallelization**
  - Performance/ Threading Methodology
  - Software tools
What can we do with faster Computers?

Solve problems faster
- Reduce turn-around time of big jobs
- Increase responsiveness of interactive apps

Get better solutions in the same amount of time
- Increase resolution of models
- Make model more sophisticated
Why Parallel Computing?

“These free lunch is over: A Fundamental Turn Toward Concurrency” – Herb Sutter, Dr. Dobb’s Journal, March 2005

We want applications to execute faster…

Clock speeds no longer increasing exponentially
Computing Landscape

What can we expect in the future?

- **Scalar plus many cores** for highly threaded workloads
- **Large, Scalar cores** for high single-thread performance
- **Many-core array** CMP with 10s-100s low power cores
- **Scalar cores** Capable of TFLOPS+
- **Full System-on-Chip Servers, workstations, embedded...**

**Evolution**

Transformation of Landscape

**Tier-1 compute resource**
A cluster of *multicores*, and soon heterogeneous *manycores*

**Tier-2 compute resource**
General purpose computation using:
- **GPU** (Graphics Processing Unit)
- **FPGA** (Field Programmable Gate Array)
- **Cell** Processor
How can we accomplish this?

Through parallel-computing...

Attempt to speed solution of a particular task by

1. Dividing task into sub-tasks
2. Executing sub-tasks simultaneously on multiple processors

Successful attempts require both

1. Understanding of where parallelism can be effective
2. Knowledge of how to design and implement good solutions
Topics

- CPU (semiconductor) History
  - Moore’s Law & Multi-Core
  - Transistor scaling & its impact
  - What then? The new era...

- Software Parallelization
  - Performance/ Threading Methodology
  - Identifying Threading Issues using Analysis Tools
Moore’s Law

“... the number of transistors on a chip approximately doubles every 24 months ...”

— Gordon Moore
Circa 1975
Historical Driving Forces

**Shrinking Geometry**

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
<th>Transistors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1971</td>
<td>4004 Processor 2300 Transistors</td>
<td></td>
</tr>
<tr>
<td>1978</td>
<td>8008 Processor IBM PC</td>
<td></td>
</tr>
<tr>
<td>1986</td>
<td>i386 Processor 32-bit</td>
<td></td>
</tr>
<tr>
<td>1993</td>
<td>Pentium Processor 3.1M transistors</td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>Montecito 1.7B Transistors</td>
<td></td>
</tr>
</tbody>
</table>

**Increased Frequency**

<table>
<thead>
<tr>
<th>Year</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1970</td>
<td>4004 Processor 2300 Transistors</td>
</tr>
<tr>
<td>1978</td>
<td>8008 Processor IBM PC</td>
</tr>
<tr>
<td>1986</td>
<td>i386 Processor 32-bit</td>
</tr>
<tr>
<td>1993</td>
<td>Pentium Processor 3.1M transistors</td>
</tr>
<tr>
<td>2005</td>
<td>Montecito 1.7B Transistors</td>
</tr>
</tbody>
</table>
Architectural Innovations

- Serial, sequential execution
- Overlapped execution (pipelining)
- Multi-stage, deep pipelining
- Control-speculative execution
- Data-speculative execution
- Super-scalar execution
- Out-of-order execution
- Vector computing
- Addressing extensions

- Application specific instructions
- Multi-level on-chip caching
- Memory disambiguation
- Register renaming
- Score-boarding
- Hardware data prefetching
- ...

Many decades of computer architecture focused on Instruction-Level Parallelism (ILP) enhancement
Processor Resources

- Caches: L0, L1, L2 etc (Different levels of caches)
- General Purpose Registers (For SW programming)
- Segment Registers & TLB (for memory management)
- FP registers, XMM registers
- System Flags
- Control and Data registers, Debug registers
- Many more
Why Multi-core? The Challenges

Power Limitations

Supply Voltage (V)

Diminishing Voltage Scaling

Power = Capacitance \times \text{Voltage}^2 \times \text{Frequency}

Power \sim \text{Voltage}^3

Copyright © 2007, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.
What then?

- **Performance**
- **Power**

Max Frequency

1.00x
Over-clocking

Over-clocked (+20%)

1.73x Performance
1.13x Power

Max Frequency

1.00x
Under-clocking

- **Over-clocked (+20%)**: 1.73x
- **Max Frequency**: 1.00x
- **Under-clocked (-20%)**: 0.87x
Multi-Core Energy-Efficient Performance

Over-clocked (+20%)

Max Frequency

Dual-core (-20%)

Relative single-core frequency and Vcc
Dual core with voltage scaling

**RULE OF THUMB**

<table>
<thead>
<tr>
<th>Frequency Reduction</th>
<th>Power Reduction</th>
<th>Performance Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>15%</td>
<td>45%</td>
<td>10%</td>
</tr>
</tbody>
</table>

**SINGLE CORE**

- Area = 1
- Voltage = 1
- Freq = 1
- Power = 1
- Perf = 1

**DUAL CORE**

- Area = 2
- Voltage = 0.85
- Freq = 0.85
- Power = 1
- Perf = ~1.8

A 15% Reduction In Voltage Yields

**Frequency Reduction**
- 15%

**Power Reduction**
- 45%

**Performance Reduction**
- 10%
A New Era...

THE OLD

Performance Equality Frequency
Unconstrained Power
Voltage Scaling

THE NEW

Performance Equals IPC
Multi-Core
Power Efficiency
Microarchitecture Advancements
Multi-Core: what next?

Many Floating-Point Cores

+ 3D Stacked Memory

Many general-purpose cores

Next research challenge

SRAM

100+ Research Projects Worldwide

Microprocessor

Examples:
Scalable memory
Multi-core architectures
Specialized cores
Scalable fabrics
Energy efficient circuits

Platform

Examples:
3D Stacked Memory
Cache Hierarchy
Virtualization/Partitioning
Scaleable OS’s
I/O & Networking

Programming

Examples:
Speculative Multithreading
Transactional memory
Workload analysis
Compilers & Libraries
Tools

Copyright © 2007, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.
Intel Polaris (80-core)
Multi-core: Architectural Challenges

- Instruction-level parallelism v/s Thread-level parallelism tradeoffs
- Shared resource management (functional units, caches, TLB, BTB)
- Multi-threading v/s Multi-core tradeoffs
- On and Off-chip bandwidth requirements
- Latencies (execution, cache, and memory) reduction
- Memory Coherence/Consistency (for high speed on-die cache hierarchies)
- Partitioning resources (between threads/cores)
- Fault tolerance (at device, storage, execution, core level) (aka reliability)
- On-die interconnect (optimized along latency, HW, modularity, power, ...)

Copyright © 2007, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.
Multi-core: Eco-system challenges

Underlying Software assumptions on resource sharing

• Lack of standard mechanisms to share “resource sharing info” between hw and OS

Lack of “Resource sharing” aware SW

• Compilers, Schedulers, Configuration/Management (Power!) etc

Legacy SW architectural requirements left on Multi-Core CPUs

• Compatibility requirements

Many more...unknowns (to CPU Design world)
Topics

- CPU (semiconductor) History
  - Moore’s Law & Multi-Core
  - Transistor scaling & its impact
  - What then? The new era...

- Software Parallelization
  - The need of the day...
  - Performance/ Threading Methodology
  - Identifying Threading Issues using Analysis Tools
Games & Multithreading

Today

- Few current game platforms have multi-core architectures
- Multithreading pain often not worth performance gain
- Most games are single-threaded (or mostly single-threaded)

Future

- Game platforms will have multi-core architectures (PCs & Game consoles)
- Games wanting to maximize performance will be multithreaded

Need to address this change! Thread for efficiency.
Several Applications can be Threaded

- Parallel implementation of MPEG-2 & H264 Encoding
  - Divide frame into multiple blocks & encode them in parallel
- Threading High Performance Computing Applications
- Threading CAD and S/W Modeling Applications
- Threading Interactive Applications for better UI feel
- Encryption, Game Theory and Number Crunching
- And many more...
Designing Threaded Programs

**Partition**
- Divide problem into tasks

**Communicate**
- Determine amount and pattern of communication

**Agglomerate**
- Combine tasks

**Map**
- Assign agglomerated tasks to created threads

The Problem

Initial tasks

Communication

Combined Tasks

Final Program
Ways of Exploiting Parallelism

- Domain Decomposition (a.k.a data decomposition)
- Task Decomposition (a.k.a functional decomposition)
- Pipelining (a.k.a “assembly line” parallelism)
Domain Decomposition

1. Decide how data elements should be divided among processors

2. Decide which tasks each processor should be doing

Example: Find largest element in an array
Task Decomposition

1. Divide tasks among processors

2. Decide which data elements are going to be accessed (read and/or written) by which processors

Ex: Event handler for GUI
Pipelining

Special kind of task parallelism

a.k.a “Assembly line” parallelism

Ex: 3D rendering in computer graphics

Processing 2 data sets

Stage 1 → Stage 2 → Stage 3 → Stage 4

5 steps
How Do I Parallelize An Application?
Five Major Parallel Methods

Which one do I use?

• Message passing
  • MPI, PVM

• Explicit threading
  • Windows threading API, Pthreads, Solaris threads, Java thread class

• Compiler-directed
  • Automatic parallelization, OpenMP, Intel Threading Building Blocks

• Parallel programming languages
  • HPF, CAF, UPC, CxC, Cilk

• Parallel math libraries
  • ScaLAPACK, PARDISO, PLAPACK, PETsc
Common Performance Issues

Parallel Overhead
• Due to thread creation, scheduling.

Synchronization
• Excessive use of global data, contention for the same synchronization object

Load balance
• Improper distribution of parallel work

Granularity
• No sufficient parallel work

System Issues
• Memory bandwidth, false sharing
What is scalability?

“What is it that we really mean by scalability? A service is said to be scalable if when we increase the resources in a system, it results in increased performance in a manner proportional to resources added.”

-- Werner Vogels
CTO - Amazon.com

Handle growing amounts of work in a graceful manner

What resources might be increased?

• Cores and threads
• Memory capacity
• Data, problem size
Amdahl’s Law

Describes the upper bound of parallel speedup (scaling)

Helps think about the effects of overhead

\[ T_{\text{parallel}} = \left\{ (1-P) + \frac{P}{n} \right\} T_{\text{serial}} + O \]

where, \( n \) = number of processors

Scaling = \( T_{\text{serial}} / T_{\text{parallel}} \)

\[ = \frac{0.5 + 0.25}{0.5 + 0} = \frac{0.75}{0.5} = 1.5 \]

Serial code limits scaling...
Effect of Serial Code on Scalability

Amdahl's Law

Scalability

Number of Processors

0% 10% 20% 30% 50%
Performance Tuning: Top-Down Approach

System Config
  Topology
  Network I/O
  Disk I/O
  Database Tuning
  OS

Application Design
  Threading/ APIs usage
  Data Locality
  Locks/ Heap Contention

Cache Utilization
  Architecture Pitfalls
  Branches and Loop Code
  Execution Efficiency

System

Application

Architecture
Optimization Methodology

1. Collect Data
2. Identify Bottlenecks
3. Identify Alternatives
4. Apply Solution
5. Test

Baseline

Address one issue at a time
Development Cycle – Optimization Tools

Analysis
- VTune™ Performance Analyzer

Design (Introduce Threads)
- Intel® Performance libraries: IPP and MKL
- OpenMP® (Intel® Compiler)
- Explicit threading (Win32*, Pthreads*)

Debug for correctness
- Intel® Thread Checker
- Intel Debugger

Tune for performance
- Intel® Thread Profiler
- VTune™ Performance Analyzer
## Optimization Tools

### Maximize Multi-Threaded Application Performance & Scalability with Intel® Software Development Products

<table>
<thead>
<tr>
<th>Intel® Compilers</th>
<th>The best way to get C++ and Fortran application performance on Intel® multi-core processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ SSE4 support through intrinsics and inline _asm</td>
</tr>
<tr>
<td></td>
<td>+ Code generation optimized for Nehalem</td>
</tr>
<tr>
<td></td>
<td>+ OpenMP improvements</td>
</tr>
<tr>
<td>Intel® VTune™ Performance Analyzers</td>
<td>Identify bottlenecks in source code and optimize multi-core performance</td>
</tr>
<tr>
<td></td>
<td>+ Performance Tuning Utility (PTU) introduces Data access modeling prototype on whatif.intel.com</td>
</tr>
<tr>
<td>Intel® Performance Libraries</td>
<td>Highly optimized, thread-safe, multimedia and HPC math functions</td>
</tr>
<tr>
<td></td>
<td>+ SSE4 used where applicable</td>
</tr>
<tr>
<td></td>
<td>+ Accommodate microarchitecture changes for increased performance</td>
</tr>
<tr>
<td>Intel® Threading Analysis Tools</td>
<td>Find threading errors and optimize threaded applications for maximum performance</td>
</tr>
<tr>
<td>Intel® Threading Building Blocks</td>
<td>C++ template-based runtime library that simplifies writing multithreaded applications for performance and scalability</td>
</tr>
</tbody>
</table>
Rules of Thumb – For Parallel Programming

- Think parallel.
- Program using abstraction.
- Program in tasks (chores), not threads (cores).
- Design with the option to turn concurrency off.
- Avoid using locks.
- Use tools and libraries designed to help with concurrency.
- Use scalable memory allocators.
- Design to scale through increased workloads
End of Session

Thank you

Q&A