Mission-Critical Space Software For Multi-Core Processors

Steve Crago
USC/ISI-East
November 6, 2009

FSW-09
Pasadena, CA
Outline

- Introduction
- Mission Critical Software
- Summary
Multi-Core and Many-Core

- Power density and complexity problems have driven current and next generation processors to have multiple cores on chip
  - Intel Nehalem (4 cores)
  - AMD Phenom (4 cores)
  - Sun UltraSPARC T2 (8 cores)
  - Tilera Tile64 (64 cores), Tile-Gx (100 cores in 2011)

- On-chip parallelism leads to programming challenges and opportunities
Principal OPERA Components

Hardware – Maestro Chip
- 49 cores, IBM 9SF 90 nm CMOS
- 65 GOPS at 440 MHz
  - 32 GFLOPS (theoretical)
- Four - 10 Gbps SERDES XAUI I/O
- Radiation Hard By Design (RHBD)
- Developed by Boeing SSED
  - Uses Tilera Corporation IP
  - Additional third party IP

Software
- Basic compiler tools
  - Complements Tilera’s toolset
- Benchmark code
- Performance and productivity tools
  - Parallel libraries, analysis, debugger, and run time monitor
Maestro Performance Features

- **Tiled Architecture**
  - 2-D mesh of processors, connected by low-latency high-bandwidth register-mapped networks
  - Intra-tile (i.e., intra-core) VLIW performance
  - Multi-tile ILP compilation
  - Inter-module communication acceleration at compile time

- **Processors**
  - Main processor: 3 way VLIW CPU, 64-bit instruction bundles, 32-bit integer operations
  - Static switch processor: 16-bit instructions

- **Memory**
  - L1 cache: 8 KB per core, 2 cycle latency
  - L2 cache: 64 KB per core, 7 cycle latency
  - Limited cache coherency across cores
  - Cores/tiles can access other cores L2 cache (“L3”)
  - Off-chip main memory, ~ 88 cycle latency
  - 32-bit virtual address space per core

- **I/O Interfaces**
  - Four integrated XAUI MACs
  - Two 10/100/1000 Ethernet MACs
Multi-Core Programming

- Multi-core programming is easy because:
  - Cores can run legacy microprocessor programs
  - Multi-core provides gentle slope from existing programming paradigms to those specifically for multi-core
  - Embedded software is already often multi-threaded (the real world is parallel)
  - Unlike FPGAs, programmer need not worry about
    - Timing (clock speed)
    - Bit-level optimization
    - Cycle-level synchronization
    - VHDL / Verilog / C dialects
  - Unlike compute accelerators, programmers need not worry about
    - Mapping to SIMD paradigm
    - Host / compute partitioning
    - Unusual numerical results

- Multi-core programming is hard because:
  - Parallelism can be tricky
    - Mapping / synchronization
    - Debugging
    - Performance analysis
  - Tools for parallelism have never been supported by a large market before (HPC has always been a small market) – This has changed!
Low latency connections between compute tiles expose new software issues (multi-core ≠ Symmetric Multiprocessor ≠ High Performance Computer)

Traditional Multiprocessor

Multi-Core Architecture

<10 cycles latency between tiles
Software Stack for Tile64 / Maestro

- Integrated Development Environment (IDE)
  - Code development
  - Debugging
  - Profiling

- User Libraries
  - iLib, pthreads, TMC (Tilera Multicore Components)
  - Shared memory, message passing, channels

- Operating system
  - Linux SMP 2.6 (Symmetric Multi-Processing)
  - VxWorks 6.7 SMP/AMP (planned)

- Hypervisor
  - Network interfaces
  - Tile protection

Tile64 software stack re-used for Maestro
Integrated Development Environment (IDE)

Based on Eclipse open source IDE (also compatible with VxWorks)
Outline

- Introduction
- Mission Critical Software
- Summary
Mission Critical Software

- **Reliable**
  - Validation and verification
  - *Fault Tolerance*
    - Hard and soft faults
    - Global and local faults

- **Timely**
  - High throughput
  - Low latency
  - *Real-time deadline guarantees*
    - Hard real-time
    - Soft real-time
Fault Tolerance Techniques

- **Hardware**
  - Spatial redundancy: 2, 3, or 4 processors to run all computation
  - Spares: power on extra processor when primary processor fails
  - Voting: difference detection and resolution
  - Error detection and correction: encoding used in memory
  - Radiation hardening for single-event upsets and total ionizing dose

- **Software**
  - Temporal redundancy
  - Checkpoint and restart
  - Error checking (e.g. check that answers are within range)
Real-Time Software Technology

- Real-time operating systems (RTOS)
  - VxWorks
  - RTEMS
  - Real-time Linux
  - QNX

- Deterministic scheduling and interrupt handling
  - Round-robin
  - Fixed priority preemptive
  - Fixed priority non-preemptive
  - Earliest deadline first
Opportunities for Mission Core Software on Multi-core Processors

Processors (cores) are relatively cheap and are getting cheaper

- Moore’s Law continues – number of transistors doubles every 18-24 months
- Number of cores could track number of transistors
- Today: 64 cores on Tile64 / 49 on Maestro

Use dedicated cores for

- Redundancy for fault tolerance
- Real-time tasks
**Fault Tolerance on Multi-Core**

Basic Idea: Use available cores for fault tolerance and adapt redundancy scheme to application needs via software.
Advantages of Fault Tolerance on Multi-Core

- Processing resources are inexpensive
- Coordination between cores is fast
- Redundancy is inherent
- Can be adaptive to application requirements and traded off with performance
## Multi-Core Fault Tolerance: Limitations

<table>
<thead>
<tr>
<th>Limitation</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using cores for redundancy within a processor does not protect against chip or module failures.</td>
<td>Use space or redundant chips as appropriate for application</td>
</tr>
<tr>
<td>Software must be modified to implement fault tolerance.</td>
<td>Develop run-time library to provide fault tolerance services to application software</td>
</tr>
<tr>
<td>Faults may occur outside of core processors</td>
<td>Fault tolerance software must consider all error locations and must account for potential single-points-of-failure</td>
</tr>
</tbody>
</table>
## Parallelism to Exploit for Fault Tolerance

<table>
<thead>
<tr>
<th>Level</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data parallel instructions</td>
<td>SIMD (Single Instruction, Multiple Data) instructions operate on multiple data elements simultaneously</td>
</tr>
<tr>
<td>Instruction bundles</td>
<td>Multiple independent instructions can execute simultaneously</td>
</tr>
<tr>
<td>Processes or threads</td>
<td>Provide parallelism to software that can be executed in parallel on multiple cores or by time-sharing single cores</td>
</tr>
<tr>
<td>Cores</td>
<td>Processing cores can be dedicated to redundancy for fault tolerance.</td>
</tr>
<tr>
<td>On-chip networks</td>
<td>Suitable architecture have multiple on-chip networks that can be exploited by software for fault tolerance.</td>
</tr>
<tr>
<td>Memory data</td>
<td>Critical data can be stored redundantly and EDAC can protect stored data.</td>
</tr>
<tr>
<td>Memory and I/O interfaces</td>
<td>Software must exploit redundant interfaces for fault tolerance.</td>
</tr>
</tbody>
</table>
Adaptive Fault Tolerance

Fault tolerance requirements vary within and between applications. Software can adapt fault tolerance schemes at design time, programming time, or run time.

- **Application:**
  - Processing data from noisy sensors
  - Use watchdog tasks to make sure processing is running, allow upsets in data.
  - Mission critical control.

- **Fault Tolerance Technique:**
  - Processing precise data
  - Duplicate processing tasks and compare for error detection. Use checkpoint and rollback to recover.
  - Triple (or quad) modular redundancy with fault tolerant voting scheme.

Higher Performance  More Fault Tolerance
Real-Time Tasks on Multi-Core

Basic Idea: Use dedicated cores for real-time tasks to avoid context switching overhead and scheduling challenges.
Multiplexing Real-Time Tasks

Traditional uniprocessor time-multiplexing

Multi-core space allocation
Advantages of Spatial Partitioning

- **Interrupt Priority Interrupt**
- **Eliminate Context Switching Overhead**
- **Eliminate Cache Pollution**
- **Simplify Scheduling**
- **Co- Locate Processing Resources with External Resources**
Other Real-Time Issues to Address for Multi-Core

**Challenge:** On-chip network interference may cause unpredictable delay for real-time tasks that are running on multiple cores

**Solution:** Partition architecture to prevent interference between tasks

**Challenge:** Caches introduce variable task execution times, and variability may be exasperated by inter-core interference

**Solution:** Pin critical data in local memory

**Challenge:** Interrupt handlers are run on specific cores

**Solution:** Route interrupts to specified cores based on interrupt level and id
Outline

- Introduction
- Software Architecture
- Summary
Access to this infrastructure subject to sponsor approval. Contact me if you are interested.
**Modified OPERA Tools (IP Release CD - MDE-OPERA/install/bin)**
- Includes everything in the normal MDE, in addition to
- tile-cc (floating point compiler, [http://tilera.east.isi.edu/mywiki/OperaCompiler](http://tilera.east.isi.edu/mywiki/OperaCompiler))
- tile-sim (floating point simulator, [http://tilera.east.isi.edu/mywiki/OperaSimulator](http://tilera.east.isi.edu/mywiki/OperaSimulator))

**Ported Libraries/Applications**
- MPI – [http://tilera.east.isi.edu/mywiki/OperaMpi](http://tilera.east.isi.edu/mywiki/OperaMpi)
- VSIPL – [http://tilera.east.isi.edu/mywiki/OperaVsipl](http://tilera.east.isi.edu/mywiki/OperaVsipl)
- Coaddition – [http://tilera.east.isi.edu/mywiki/Coadd](http://tilera.east.isi.edu/mywiki/Coadd)

**Documentation**
- MDE-Getting-Started-Guide.pdf – quick summary on using tile-eclipse
- MDE-Users-Guide.pdf – in depth tile-eclipse/tile-monitor documentation
- MDE-System-Programmers-Guide.pdf – create customized bootrom, hypervisor, and linux
- iLib-API-Reference.pdf – reference for using iLib API
- [http://tilera.east.isi.edu/mywiki/IPRelease](http://tilera.east.isi.edu/mywiki/IPRelease) – information on OPERA IP
- [http://tilera.east.isi.edu/mywiki/FrontPage](http://tilera.east.isi.edu/mywiki/FrontPage) – information on OPERA tools, libraries, and example applications
Software Summary

- Tilera tool set provides a productive and stable environment for software development
  - iLib provides multiple, high-performance options for mapping code to tiles
  - Tools are robust and easy to use

- OPERA/MAESTRO is providing additional tools that will provide increased application coverage and novel development tools

- There are interesting challenges and opportunities for putting mission-critical software on Maestro and other multi-core processors
Acknowledgements

- USC/ISI: Jungwoo Ha, Dong-In Kang, Mikyung Kang, Rob Kost, Karandeep Singh, Jinwoo Suh, JP Walters
- University of Maryland: Don Yeung (PI)
- Reservoir Labs: Rich Lethin (PI)
- MIT: Anant Agarwal (PI)
- Wind River: Bonnie Coker (PM)