Overview

● (note: Tute 02 this Weds - handouts)
● multicore architecture concepts
  ■ hardware threading
  ■ SIMD vs MIMD in the multicore context
● T2: design features for multicore
  ■ system on a chip
  ■ execution: (in-order) pipeline, instruction latency
  ■ thread scheduling
  ■ caches: associativity, coherence, prefetch
  ■ memory system: crossbar, memory controller
  ■ intermission
  ■ speculation; power savings
  ■ OpenSPARC
● T2 performance (why the T2 is designed as it is)
● the Rock processor (slides by Andrew Over; ref: Tremblay, IEEE Micro 2009)
Hardware (Multi)threading

- recall concurrent execution on a single CPU: switch between threads (or processes) requires the saving (in memory) of thread state (register values)
  - motivation: utilize CPU better when thread stalled for I/O (6300 Lect O1, p9–10)
  - what are the costs? do the same for smaller stalls? (e.g. cache misses)
- idea: have a CPU with multiple sets of registers for each thread
  - each set represents a virtual CPU; what is switching cost now?
- three types of hardware threading:
  1. block: execute thread A until stall occurs, then switch to thread B (starvation issues)
  2. interleaved: execute a single instruction from (available) threads A, B, C, ... in turn on any cycle (e.g. UltraSPARC T1)
  3. simultaneous: execute multiple instrns. from (available) threads on any cycle can be combined with superscalar (e.g. IBM Power PC), or instrns. can be from different hardware threads (UltraSPARC T2)
- OpenSparc Slide Cast Ch 1: p7–9: ST, HMT, CMP + HMT
### SIMD and MIMD in the Multicore Context

- **Flynn’s Taxonomy**

<table>
<thead>
<tr>
<th></th>
<th>Single Instruction</th>
<th>Multiple Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Data</td>
<td>SISD</td>
<td>MISD</td>
</tr>
<tr>
<td>Multiple Data</td>
<td>SIMD</td>
<td>MIMD</td>
</tr>
</tbody>
</table>

- for SIMD, the control unit and processor state (registers) can be shared
- however, SIMD is limited to data parallelism (through multiple ALUs)
  - algorithms need a regular structure, e.g. dense linear algebra, graphics
  - SSE2, Altivec, Cell SPE (128-bit registers); e.g. $4 \times 32$-bit add

\[
\begin{align*}
R_x: & \begin{bmatrix} x_3 & x_2 & x_1 & x_0 \end{bmatrix} \\
+ & \begin{bmatrix} y_3 & y_2 & y_1 & y_0 \end{bmatrix} \\
= & \begin{bmatrix} z_3 & z_2 & z_1 & z_0 \end{bmatrix} \\
\end{align*}
\]

\(z_i = x_i + y_i\)

- design requires massive effort; requires support from a commodity environment
- massive parallelism (e.g. nVidia GPGPU) but memory is still a bottleneck

- multicore (CMT) is MIMD; hardware threading can be regarded as MIMD

  - higher hardware costs also includes larger shared resources (caches, TLBs) needed ⇒ less parallelism than for SIMD
The UltraSPARC T2: System on a Chip

- OpenSparc Slide Cast Ch 5: p79–81,89
- aggressively multicore: 8 cores, each with 8-way hardware threading (64 virtual CPUs)
  - one FPU (+graphics) and SPU (S = security, i.e. supports cryptography)
  - support (almost) full SPARC V9 instruction set;
    - $\approx 2$ KB of register state needed!! (register windows – especially engineered to keep this compact)
  - (moderately) large ITLB & DTLB with hardware tablewalk – reduce misses /miss penalty
  - requires gasket to arbitrate between the threads for L2$ access
- large shared level-2 cache (L2$); 16 way associative to reduce conflicts between threads
  - 8 banks; crossbar between CPUs & L2$ for high bandwidth
- high memory bandwidth via 4 × dual channel memory controllers
- relatively low clock speed and low power consumption
T2: Core Pipeline

- OpenSparc Slide Cast Ch 5: p89–94
- each core has two (integer) execution units, 1 load/store unit, 1 FPU
  - can issue two instrns (from different thread (groups)) per cycle
    \( \leq 2 \) integer/branch, \( \leq 1 \) load/store and \( \leq 1 \) floating point
- shallow instruction pipeline [p90] – simplifies core design
  - (conditional) branches are assumed not taken (no BP tables!); 5 cycle penalty if taken
  - 3 cycle penalty between load and first use
  - main pipeline ‘extends’ into load/store unit and FPU pipelines [p91]
  - 6 cycle latency between dependent floating point instructions [p90]
- latency \( l \) is high, but effect is reduced by \( w \)-way hardware threading (conversely, is amplified by \( w' \)-way superscalar)
  - required ‘instruction gap’ between dependent instructions is \( \frac{l}{w} \) (not \( lw' \))
  - c.f. optimized matrix multiply for \( l = 3 \) \( w' = 4 \)
    - this complexity not needed if run fully-threaded on the T2
T2: Thread Scheduling

- hardware threading introduces a pick stage and per-thread 8-entry instrn. buffers [p91]
- the fetch stage fetches up to 4 instructions for the thread just selected in the pick stage
- threads are divided into two groups of 4; one instruction from each group may be issued [p92] – simplicity/more scalable
  - selected from available threads, using least-recently fetched policy [p93]
    - however, thread may lose priority after a load [p94]
    - e.g. Kongetira et al, Figs 4-5, IEEE Micro ’05
  - thread selection is also based on resource (functional unit or register dependency) conflicts (e.g. $\text{fmuld } %f0,%f1,%f2 \ ; \ \text{fadd}d \ %f2,%f4,%f4$)
  - thread may become unavailable due to cache / TLB miss, long latency instruction (e.g. $\text{fdivd}$), branch misprediction, trap
    - also if instrn. buffer is full!
T2: Intra-Core Memory Sub-system

- Each core has a level-1 data (D$; 8KB, 4-way set associative, 16 bytes lines, virtually indexed - physically tagged) and instrn. (I$; 16 KB, 8-way; why larger?); shared between threads.
  - D$'s write-through policy simplifies coherency issues but increases traffic to L2$ [p96]
  - However, it does not allocate on a store miss.
  - A store buffer is used to mitigate this (can pipeline stores to the same 64-byte L2$ line).
  - 16 byte data transfers to/from the L2$.
  - Load miss queue need only support one pending load miss per thread
    - On non-CMT Sparcs (e.g. V1280 – 2003), these were 8 deep; why?
- Gasket to ensure fair access to L2$ between threads (priority to oldest).
T2: Inter-Core Memory Sub-system

- **crossbar**: provides high bandwidth between cores and L2$ [p101]
  - three cycle arbitration protocol (request, arbitrate and grant); priority to oldest
  - 90 GB/s write, 189 GB/s read (8 load/store requests & 8 returns / acks / invalidations) simultaneously

- L2$ is organized into 8 banks (address bits 8:6 used to select banks)
  - full bandwidth to L2$ can only be achieved if each core is accessing data in a different bank on a given cycle!
  - 26 cycle latency

- L2$ implements atomic instructions (e.g. ldstub [%00], [%01])
  - semantics requires store buffers to be drained (entries for all 64 threads) 1st!
  - effectively stops most other memory operations; becomes effectively more expensive with increasing number of cores (20-30 cycle latency)

- L2$ manages coherency via separate directories to all 16 I$ and D$ [p102]
  - broadcasts an invalidate signal whenever a value is changed

- memory is organized into 4 banks (selected by address bits 8:7) [p86,102], enabling 42 GB/s read, 21 GB/s write; s/w can prefetch data into the L2$
UltraSPARC T2: Speculative Execution

the ultimate instruction level parallelism technique in high power/chip area costs and low on average (single threaded) performance gains, so . . .

UltraSPARC T2: Power Management

CoolThreads: low cooling requirements

p99 CPU: worst case power usage 84W (1.4 GHz); T5120 system: 330W

- can turn cores, DRAM banks on/off via software
- H/W uses clock gating to turn off functional units (e.g. FPUs) not in use
- power throttling: can inject stall cycles to cool chip, or conserve power in low workload periods
- improved reliability with lower and more uniform junction temperatures, on-chip temperature control
OpenSPARC T2: A Open-Source Chip Design

- see www.opensparc.net/opensparc-t2
- includes micro-architecture specification, RTL and verification suite
- supports projects in
  - create new (variant) core designs
  - explore CMT design points for new applications
  - port OS of your choice, explore program/compiler optimizations
  - using software (simulator) and/or single core single thread FPGA
    implementation of OpenSPARC T1
UltraSPARC T2: Performance

- world record SPECint_Rate2006, SPECfp_Rate2006 (single chip) [p85]
  - not optimized for FP; and a low clock speed?
  - for dual-socket systems (Kumar et al, 2008):

<table>
<thead>
<tr>
<th></th>
<th>clock</th>
<th>cores</th>
<th>f.p/cycle</th>
<th>R_peak</th>
<th>SPECfp2006</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5240</td>
<td>1.4 GHz</td>
<td>16</td>
<td>1</td>
<td>22.4 GF</td>
<td>119</td>
</tr>
<tr>
<td>IBM p570</td>
<td>4.7 GHz</td>
<td>4</td>
<td>4</td>
<td>75.2 GF</td>
<td>116</td>
</tr>
</tbody>
</table>

c.f. 3.2GHz Cell has \( R_{peak} \) of \( 3.2 \times 8 \times 4 = 102 \) GF

- reason: FLOPs aren’t everything; memory performance is important!

- world records (single or dual chip) on SPEC web, Java VM, Java App, OLTP etc
  - for SPECweb2005, T5220 rating: 41847 (@426W)
  - nearest: \( 2 \times 4 \)-core Opteron Sun Fire X4240: 32288 (@411W)

- niche is cryptographic applications (SPUs act as on-board accelerators)
  - e.g. out-performs 2.7GHZ quad-core Clovertown by factors of \( \approx 10 \)
UltraSPARC T2: Design for Performance Choices

- aggressively multi-core and threaded design
  - avoid expensive single-threaded motivated features:
    - branch prediction, (wide) superscalar, O-O-E, deep pipeline, speculative ex.
      in favor of a simpler, highly threaded approach
    - can afford higher L & lower GHz; reduced s/w complexity (once threaded); no
      need for deep prefetch buffers; can better tolerate cache misses
      but:
    - pick stage, per-thread instrn. & store buffers, thread scheduling (reduced by
      2 sub-groups), many sets of register files, larger and more associative
      caches, coherency and atomic instruction implementation
  - equal effort put into high bandwidth L2$ and main memory access
    - banking introduces complexities (gasket & crossbar) in h/w (& optimizing s/w)
- could be (easily) improved for floating point (dual FP issue or fused multiply-add),
  but intended for other markets (memory & I/O intensive; encryption)
- supporting full SPARC ISA somewhat of a burden
  - c.f. Terascale chip (80 cores @ 4.3GHz; 32-bit multiply-adder; minimal ISA,
    on-chip memory & I/O; 97W)
- power/performance also important (low peak power & power saving features)