Memory Architectures

- Shared Memory
Memory Architectures

- Shared Memory
- Distributed Memory

Memory Architectures

- Shared Memory
- Distributed Memory
- Distributed Shared Memory
Last time: Quadrics (Elan)

- Trouble shoehorning MPI into a system designed for Elan3lib
- Why not tightly couple hardware with programming model

Tera Programming Model

- MultiThreading Architecture (MTA)
  - One goal: Try to push as much as possible from OS to user-level
Tera’s MTA System

- Shared Memory
- No cache!
  - No ownership concerns
  - In perfectly parallel world, don’t need it
- High bandwidth
  - All 256 processors can issue load/store every clock cycle

Tera’s MTA System

- Lookahead (Software Pipelining)
  - “Latency tolerance”
  - Instruction tells processor how long before a load/store will be required
- Wide instructions
  - Instruction-level parallelism
  - One memory op
  - Two arithmetic ops
Tera’s MTA System

- Protection domains
  - 16 per processor
  - Allows multiple processes on a processor to have distinct address spaces without expensive context switches
- What else?

Hardware Interleaving

- Many program counters (threads/streams)
- Every clock cycle, processor executes instruction from different “thread”
- Thread creation and destruction lightweight
  - No need for OS intervention
  - Go nuts and create them for no reason
Memory State Bits

- Provide lightweight synchronization
- At the word level
  - Data-trap bit: for user-level exceptions
  - Full-empty bit: 3 modes
    - Normal
    - Future (for asynchronous computation)
    - Synch

What to do now?

- We have all these parallel abstraction
- How can we exploit them?
Compilers

- Relies heavily on compilers
- However, loop-level parallelism and instruction-level parallelism are well understood
- No need to worry about cache management or branch prediction

Example: Radix Sort

```c
for (i = 0; i < key_value_bound; i++)
    count[i] = 0;

for (i = 0; i < nkeys; i++)
    count[key[i]]++;

start[0] = 0;
for (i = 1; i < key_value_bound; i++)
    start[i] = start[i - 1] + count[i - 1];

for (i = 0; i < nkeys; i++)
    rank[i] = start[key[i]]++;```

for (i = 0; i < key_value_bound; i++)
    count[i] = 0;

for (i = 0; i < nkeys; i++)
    count[key[i]]++;

start[0] = 0;
for (i = 1; i < key_value_bound; i++)
    start[i] = start[i - 1] + count[i - 1];

for (i = 0; i < nkeys; i++)
    rank[i] = start[key[i]]++;

Example: Radix Sort

for (i = 0; i < key_value_bound; i++)
    count[i] = 0;

for (i = 0; i < nkeys; i++)
    count[key[i]]++;

start[0] = 0;
for (i = 1; i < key_value_bound; i++)
    start[i] = start[i - 1] + count[i - 1];

start$ = (sync int *) start;
#pragma tera assert parallel
for (i = 0; i < nkeys; i++)
    rank[i] = start$[key[i]]++;
Assembly Instructions

```c
for (i = 0; i < nkeys; i++)
    count[key[i]]++;
```

```assembly
(inst 1 (INT_MEM_ADD_INDEX r19 r8 r14) (INT_ADD_IMM r21 r21 8))
(inst 7 (LOAD r14 r21) (NOP) (NOP))
(inst 1 (INT_MEM_ADD_INDEX r19 r8 r20) (INT_ADD_IMM r21 r21 8))
(inst 7 (LOAD r20 r21) (NOP) (NOP))
(inst 1 (INT_MEM_ADD_INDEX r19 r8 r11) (INT_ADD_IMM r21 r21 8))
(inst 7 (LOAD r11 r21) (NOP) (INT_ADD r6 r21 r4))
(inst 1 (INT_MEM_ADD_INDEX r19 r8 r10) (INT_ADD_IMM r21 r21 8))
(inst 7 (LOAD r10 r21) (NOP) (INT_SUB_TEST r0 r6 r7))
(inst 1 (INT_MEM_ADD_INDEX r19 r8 r22) (INT_ADD_IMM r21 r21 8))
(inst 7 (LOAD r22 r21) (NOP) (JUMP_OFTEN IF_ILT cn0 t3))
```

Ugly Hack

- How can you have a statistics thread that sleeps?
  - Interrupts replaced by traps
- Answer: Mask trap-bit
  - Synch access empty word
  - Once retry limit reached, control will return to statistics thread
Other concerns

- Rethink OS design
  - Lends itself to microkernel design
- Debugger easier to implement
  - Don’t affect timing too much

Cray X1 Programming Model

- Shmem
- Co-Array Fortran
- MPI
Cray X1 System

- Distributed Shared Memory
  - NUMA
- No caching between nodes
  - Get around ownership issue
- Continue theme of hierarchical design

- SSP
  - Two 32-stage vector
  - One scalar
- MSP
  - Four SSPs
- Ecache?

Figure 1. Cray MSP module.
Ecache

- Insufficient memory bandwidth to saturate vector units
  - But we are close, within factor of 2
- Sneak a little 2MB cache in there
  - Make it really fast, even for random stride
  - Hope there’s a little data reuse

To MSP or SSP?

- Can use all eight vector units in an MSP when vectorizing a single loop
- Or use in a Tera-like mode
  - Split up work on an unvectorized loop and use SSPs separately
  - Again, requires compiler support for instruction-level parallelism
Continuing the Hierarchy

- **Node**
  - Four MSPs
  - 16 memory controller chips
- **204 Gbyte/s bandwidth**
  - Enough to saturate local MSPs and service remote requests
- **Aggregate bandwidth between nodes**
  - 50 Gbyte/s

More Hierarchy

- **Processor stack**
  - 8 adjacent connected nodes
- **Cabinet**
  - Two Processor stacks
Atomic Operations

- Regular fetch-and-op
- But also ordering instructions at each level of hierarchy
  - Because many times synchronization implies ordering

Transparent Global Load/Store

![Diagram](image)

Figure 3. Gray X1 address translation.
Cray X1 versus Others…

Table 1. Platform configurations.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>SGI Altix</th>
<th>Alpha SC</th>
<th>Earth Simulator</th>
<th>IBM SP4</th>
<th>Cray X1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Interconnect</td>
<td>Intel Itanium 2</td>
<td>Quadrics E360</td>
<td>NEC SX-6</td>
<td>IBM Power4</td>
<td>Cray X1</td>
</tr>
<tr>
<td>Processor speed (MHz)</td>
<td>1,500</td>
<td>667</td>
<td>500</td>
<td>1,300</td>
<td>800</td>
</tr>
<tr>
<td>Memory bandwidth (Gbytes)</td>
<td>512</td>
<td>2</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>L1 cache size (Kbytes)</td>
<td>32</td>
<td>64</td>
<td>NA</td>
<td>32</td>
<td>16 (scalar)</td>
</tr>
<tr>
<td>L2 cache size (Mbytes)</td>
<td>0.256</td>
<td>8</td>
<td>NA</td>
<td>1.5</td>
<td>2 (per MSP)</td>
</tr>
<tr>
<td>L3 cache size (Mbytes)</td>
<td>6</td>
<td>NA</td>
<td>NA</td>
<td>128</td>
<td>NA</td>
</tr>
<tr>
<td>Processor peak performance (MFlops)</td>
<td>6,000</td>
<td>1,334</td>
<td>5,000</td>
<td>1,200</td>
<td>12,000</td>
</tr>
<tr>
<td>Peak memory bandwidth (Gbytes)</td>
<td>6.4</td>
<td>5.2</td>
<td>32 (per each processor)</td>
<td>51 (per module)</td>
<td>26 (per each module)</td>
</tr>
</tbody>
</table>

Figure 4. Stream triad with Co-Array traffic.
Figure 6. MPI internode bandwidth.

Figure 7. Halo-exchange timings.
Figure 8. Allreduce latency.

Figure 9. Performance of the LANL Parallel Ocean Program (POP) version 1.4.3.
Figure 10. Performance of the Gyro Eulerian Gyrokinetic-Maxwell Solver 64-mode GTC benchmark.

Figure 11. Communication fraction for Gyro 64-mode GTC benchmark.