Parallel Architecture

Announcements
– The RamCT merge is done! Please repost introductions.
– Manaf’s office hours
– HW0 is due tomorrow night, please try RamCT submission
– HW1 has been posted

Today
– Isoefficiency
– Levels of parallelism in architecture and programming models
– Leveraging the memory hierarchy
– Understanding performance limits
  – Machine balance
  – Roofline model
– Performance analysis logistics for this semester

Recall Parallel Performance Metrics

**Speedup**
\[ T_s(N) \text{ exec time for efficient serial computation on problem size N} \]
\[ T(N, P) \text{ exec time for parallel version of computation on problem size N with } P \text{ processors} \]

speedup is the serial exec time divided by the parallel exec time
\[ S = \frac{T_s(N)}{T(N, P)} \]

**Efficiency**
efficiency is the percentage of all the processing power that is being used
\[ E = \frac{T_s(N)}{(PT(N, P))} = \frac{S}{P} \]
Isoefficiency

Main idea
– Keep the parallel efficiency the same and increase the problem size.
– The isoefficiency function indicates how much the problem size needs to increase as processors are added.

Example
– Summation is a reduction
– Serial execution time \( T_S(N) = O(N) \)
– Parallel execution time needs to include communication time and idle time, or parallel overhead \( T(N, P) = O(N/P + \log_2(P)) \)
– Isoefficiency function indicates how the problem size must increase as a function of the number of processors to maintain the same efficiency.

\[
N = P \log P
\]

Parallel Architectures

Parallelism
– Many processing units (floating point and integer units, processors)
– Places to store data (memory, cache)
– Various ways the processors are connected to the memory.
– No real “winner” parallel architecture, so variety in programming models.
– Tradeoffs between portable code and efficient code.
– Goal of automation tools (e.g., compilers) is to find the sweet spots
Levels of Parallelism in Architectures

**Instruction Level Parallelism**
- Pipeline parallelism
- Superscalar
- VLIW (very long instruction word)
- Vector processing units

**Shared Memory Parallelism:** multiple processors connected to same memory usually with cache coherency
- Multicore machines like veges
- Node of the cray
- Shared memory for a thread block in a GPU and global memory in GPU

**Distributed Memory Parallelism:** multiple processors each with own memory connected with an interconnect
- Between nodes of the cray
- Clusters

Levels of Parallelism in Programming Models

**Instruction Level Parallelism**
- Mostly handled by the compiler
- Loop unrolling
- MMX, SSE, and AVX

**Shared Memory Parallelism**
- OpenMP
- Threads: pthreads, Java threads, Cilk, TBB

**Distributed Memory Parallelism**
- MPI
- PGAS languages
  - Co-array Fortran, Unified Parallel C, Titanium, …
- New languages with concept of places/locales
  - X10
  - Chapel
Memory Hierarchy

What?
- There is an ever growing hierarchy of caches that lie between main memory and the processing unit. Eg, L1 cache, L2 cache, L3 cache
- In parallel machines the hierarchy causes non-uniform memory access (NUMA) due to subsets of cores sharing caches.

Why?
- It takes 100-1000 cycles to access main memory directly.
- Caches (SRAM-based memory) are fast but expensive and therefore not that large.
Examples of Caching in the Hierarchy

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>What is Cached?</th>
<th>Where is it Cached?</th>
<th>Latency (cycles)</th>
<th>Managed By</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>4-8 bytes words</td>
<td>CPU core</td>
<td>0</td>
<td>Compiler</td>
</tr>
<tr>
<td>TLB</td>
<td>Address translations</td>
<td>On-Chip TLB</td>
<td>0</td>
<td>Hardware</td>
</tr>
<tr>
<td>L1 cache</td>
<td>64-bytes block</td>
<td>On-Chip L1</td>
<td>1</td>
<td>Hardware</td>
</tr>
<tr>
<td>L2 cache</td>
<td>64-bytes block</td>
<td>On/Off-Chip L2</td>
<td>10</td>
<td>Hardware</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>4-KB page</td>
<td>Main memory</td>
<td>100</td>
<td>Hardware + OS</td>
</tr>
<tr>
<td>Buffer cache</td>
<td>Parts of files</td>
<td>Main memory</td>
<td>100</td>
<td>OS</td>
</tr>
<tr>
<td>Disk cache</td>
<td>Disk sectors</td>
<td>Disk controller</td>
<td>100,000</td>
<td>Disk firmware</td>
</tr>
<tr>
<td>Network buffer cache</td>
<td>Parts of files</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>AFS/NFS client</td>
</tr>
<tr>
<td>Browser cache</td>
<td>Web pages</td>
<td>Local disk</td>
<td>10,000,000</td>
<td>Web browser</td>
</tr>
<tr>
<td>Web cache</td>
<td>Web pages</td>
<td>Remote server disks</td>
<td>1,000,000,000</td>
<td>Web proxy server</td>
</tr>
</tbody>
</table>


Harpertown Architecture (veges in department)

```
Machine (16GB)

Socket #0
L2 #0 (6144KB)  L2 #1 (6144KB)
L1 #0 (32KB)    L1 #1 (32KB)  L1 #2 (32KB)  L1 #3 (32KB)
Core #0  PU #0  Core #1  PU #1  Core #2  PU #2  Core #3  PU #3

Socket #1
L2 #2 (6144KB)  L2 #3 (6144KB)
L1 #4 (32KB)    L1 #5 (32KB)  L1 #6 (32KB)  L1 #7 (32KB)
Core #4  PU #4  Core #5  PU #5  Core #6  PU #6  Core #7  PU #7

//commands
setenv PATH /s/bach/e/proj/rtrt/software/bin:$PATH
lstopo --output-format pdf > lstopo-out.pdf
```
Cray Architecture

Each compute node contains 2 processors (2 sockets)
64-bit AMD Opteron “Magny-Cours” 1.9GHz processors
1 NUMA processor = 8 cores
4 NUMA processors per compute node
24 cores per compute node
4 NUMA processors per compute blade
32 GB RAM (shared) / compute node = 1.664 TB total RAM (ECC DDR3 SDRAM)
1.33 GB RAM / core

Source: http://istec.colostate.edu/istec_cray/tutorial_3_30_11.pdf

NVIDIA Tesla (bananas, coconuts, apples, and oranges)

Source: http://www.euroben.nl/reports/web09/tesla.jpg
Data Reuse and Data Locality

Definitions

– *Temporal reuse* is when the same memory location is read more than once.

– *Spatial reuse* is when more than one memory location mapped to the same cache line are used.

– *Temporal and spatial data locality* occurs when those reuses occur before the cache line is kicked out of cache.

Matrix Multiply Example <Draw pictures in class>

```
for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
        C[i][j] = 0;
        for (k=0; k<N; k++) {
            C[i][j] += A[j][k] * B[k][j];
        }
    }
}
```

Sparse Matrix Vector Example <Draw pictures in class>

```
for (j=0; j<N; j++) { Y[i] = 0; }
for (i=0; i<NNZ; i++) {
    Y[row[i]] += val[i]*X[col[i]];
}
```

Arithmetic Intensity and Machine Balance

Arithmetic Intensity

– Ratio of arithmetic operations to memory operations within a computation/loop.

Machine Balance

– Ratio of peak floating point ops per cycle to sustained memory ops per cycle.

– Number of floating point operations that can be performed during the time for an average memory access.

Why?

– If the arithmetic intensity doesn’t match the machine balance then the computation will be memory bound.

– If there is data reuse then it might be possible to store data in scalars (i.e. registers) and raise the arithmetic intensity.
Roofline Model (Will be using this in HW2)

Roofline: An Insightful Visual Performance Model for Multicore Architecture
- By Sam Williams, Andrew Waterman, and David Patterson

Operational Intensity: Operations per byte of DRAM traffic.

Roofline graph per machine
- FLOPS/sec versus operational intensity
- Horizontal line for the peak floating point performance (compute bound)
- Diagonal line for the measured peak memory performance (memory bound)

Placing ceilings to represent how performance optimizations can help
- Improve ILP and apply SIMD (computation bound)
- Balance floating point operation mix (computation bound)
- Unit stride accesses (memory bound)
- Memory affinity (memory bound)

Logistics of Performance Analysis

Multiple observations are necessary
- Execution time will not be the same for every run: other users, slightly different cache alignments, etc.
- Need to plot the execution time average of 5-10 observations with bars for standard deviation. Is 5-10 enough?

Performance Issues for specific architectures
- Throughout the semester post programming techniques that improve performance on the vege, cray, and tesla machines.
Concepts

Isoefficiency

Levels of Parallelism in Arch and Programming Languages
- ILP, shared memory, distributed memory
- Loop unrolling, SSE and AVX instructions, do all loops, SPMD, message passing

Memory Hierarchy
- NUMA
- Data reuse and data locality
- Programming constructs to manage data locality?

Performance limits
- Machine balance
- Roofline model (how to draw the graph)
  - operational intensity
  - Programming techniques to break through ceilings

Next Time

Reading
- Berkeley View

Homework
- HW0 is due Wednesday 1/25/12
- HW1 is due Wednesday 2/1/12

Lecture
- Scientific Applications of Interest
Isoefficiency

\[ P = 160 \]
\[ N = 160 \]

\[ T_s(N) = 16 \]
\[ T(N, P) = \frac{N}{P} + \log P \]
\[ = 1 + 4 = 5 \]

\[ E = \frac{T_s(N)}{P + T(N, P)} = \frac{16}{16 + 5} = 20\% \]

Let \( N > P \)

\[ N = 16, P = 4 \]
\[ T(N, P) = \frac{N}{P} + \log P \]
\[ = 4 + 2 = 6 \]

\[ E = \frac{16}{4 \times 6} = \frac{16}{24} = 66.7\% \]

General Case

\[ E = \frac{N}{P} + P \log P \]

\[ = \frac{1}{1 + (P \log P / N)} \]

\( N \geq P \log P \)
Matrix Multiply

\[ C = AB \]

\[ \begin{bmatrix} \vdots & \vdots & \vdots \\ \hline \\ \vdots & \vdots & \vdots \end{bmatrix} \]

\[ O(N^3) \) \text{ computation} \]
\[ O(N^2) \) \text{ data} \]

\[ \text{reuse} = O(N) \]

SpMV

\[ y = Ax \]
Roofline Model (Opteron X2)

Gflops/sec

- Software prefetching
- Memory affinity
- Unit stride
- Memory access time
- BW

Operational Intensity

\[ \frac{G\text{Flops}}{\text{Byte}} \]

\[ \frac{G\text{Flops}}{\text{sec}} = \text{Peak wall} \times \frac{G\text{Flops}}{\text{Bytes}} \]

\[ \frac{\text{Bytes}}{\text{sec}} \times \frac{G\text{Flops}}{\text{Bytes}} \]