Scaling: Why is it important?

• Over time:
  – computer systems become larger and more powerful
    » more powerful processors
    » more processors
    » also range of system sizes within a product family
  – problem sizes become larger
    » simulate the entire plane rather than the wing
  – required accuracy becomes greater
    » forecast the weather a week in advance rather than 3 days

• Scaling:
  – How do algorithms and hardware behave as systems, size, accuracies become greater?

• Intuitively:
  – “Performance” should scale linearly with cost
Cost

- Cost is a function of more than just the processor.
- Cost is a complex function of many hardware components and software
- Cost is often not a "smooth" function
  - Often a function of packaging
    » how many pins on a board
    » how many processors on a board
    » how many boards in a chassis

Performance

- How does performance vary with added processors?
- Depends on inherently serial portion vs. parallel portion
- Depends on problem size
- Depends on architecture and algorithm
- Depends on computation vs. communication
**Speedup Review**

Let Speedup = \( T_{\text{serial}} / T_{\text{parallel}} \)

- **Amdahl's law**
  
  \[ f = \text{fraction of serial work}; \]
  \[ (1-f) = \text{parallel fraction} \]

- **Speedup with N processors,** \( S(N) = 1 / (f + (1-f)/N) \)
  
  Maximum speedup = \( 1/f \)
  
  Eg. 10% serial work => maximum speedup is 10.

---

**Effect of Problem Size**

- **Amdahl's law assumes constant problem size**
  
  - Or, serial portion grows linearly with parallel portion

- **Often, serial portion does not** grow linearly with parallel portions

- And, parallel processors solve larger problems.

- **Example: N x N Matrix multiplication**
  
  Initialize matrices, serial, complexity N
  
  Multiply matrices, parallel, complexity \( N^3 \)
Problem Constrained Scaling

• User wants to solve same problem, only faster
  – E.g., Video compression & VLSI routing

  \[ \text{Speedup}_{PC}(p) = \frac{\text{Time}(1)}{\text{Time}(p)} \]

• Assessment
  – Easy to do & explain
  – In some cases may not be realistic
  – Doesn’t work well for much larger machine
    (c.f., Amdahl’s Law)

Time Constrained Scaling

• Execution time is kept fixed as system scales
  – User has fixed time to use machine or wait for result

• Performance = Work/Time as usual, and time is fixed, so

  \[ \text{Speedup}_{TC}(p) = \frac{\text{Work}(p)}{\text{Work}(1)} \]

• Assessment
  – Often realistic (e.g., best weather forecast over night)
  – Must understand application to scale meaningfully
    (would scientist scale grid, time step, error bound, or combination?)
**Time Constrained Scheduling**

- Overheads can become a problem:
- For matrix multiply, data set size can be increased by $N^{1/3}$
  \[ \Rightarrow \text{for 1000 x more processors, data size increases by 10.} \]
- Problem grows slower than processors,
- Eventually performance gain will flatten
  - and diminish due to overheads associated with smaller amounts of data per processor.
  - Start with 100 element array $\Rightarrow$ 100 elts. per 1 processor
  - Scale up to 1000 processors $\Rightarrow$ 1 elt. per processor

---

**Memory Constrained Scaling**

- Let problem size scale linearly with number of processors.
  - (assumes memory scales linearly with #processors)
- Scaled Speedup: Is $\frac{\text{Time}(1)}{\text{Time}(p)}$?

\[
\text{Speedup}_{MC}(p) = \frac{\text{Work}(p)}{\text{Time}(p)} \times \frac{\text{Time}(1)}{\text{Work}(1)} = \frac{\text{Increase in Work}}{\text{Increase in Time}}
\]

- Assessment
  - Realistic for memory-constrained programs (e.g., grid size)
  - Can lead to large increases in execution time if work grows faster than linearly in memory usage
Memory Constrained Scaling

- Matrix multiply example:
  \[ f = \frac{N}{(N + N^3)}, \text{ and} \]
  \[ S(1) = 1 \]
  \[ S(10) \approx 10 \]
  \[ S(100) \approx 100 \]
  \[ S(1000) \approx 1000 \]

- BUT, 1000 times increase in problem size =>
  1,000,000 times increase in execution time, even with 1000 processors.

---

How to scale?

- Not just data
- Must consider application constraints
  - ERROR
- Equal Error Scaling
  - Scale all sources of error so they have equal contribution to total error
Example: Barnes-Hut Galaxy Simulation

- Different parameters govern different sources of error
  - Number of bodies \( (n) \)
  - Time-step resolution \( (dt) \)
  - Force calculation accuracy \( (fa) \)

- Scaling Rule
  All components of simulation error should scale at the same rate

- Result: If \( n \) scales by a factor of \( s \)
  - \( dt \) must scale by \( s^{1/4} \)
  - \( fa \) must scale by \( s^{1/4} \)

Demonstrating Scaling Problems

- Small & big Ocean problems on SGI Origin2000

(C) 2001 Mark D. Hill from Adve, Falsafi, Lebek, Reinhardt & Singh  CS/ECE 757
Scaling Down

• Scale down to shorten evaluation time on hardware and especially on simulators

• “Scale up” issues apply in reserve

• Must watch out if problem size gets too small
  – Communication dominates computation (e.g., all boundary elements)
  – Problem size gets too small for realistic caches, yielding too many cache hits
    » Scale caches down considering application working sets
    » E.g., if a on a realistic problem a realistic cache could hold a matrix row but not whole matrix
    » Scale cache so it hold only row or scaled problem’s matrix

The SPLASH2 Benchmarks

• Kernels
  – Complex 1D FFT
  – Blocked LU Factorization
  – Blocked Sparse Cholesky Factorization
  – Integer Radix Sort

• Applications
  – Barnes-Hut: interaction of bodies
  – Adaptive Fast Multipole (FMM): interaction of bodies
  – Ocean Simulation
  – Hierarchical Radiosity
  – Ray Tracer (Raytrace)
  – Volume Renderer (Volrend)
  – Water Simulation with Spatial Data Structure (Water-Spatial)
  – Water Simulation without Spatial Data Structure (Water-Nsquared)
Characterizing Applications

- **Concurrency and Load balance**
  - Measure inherent parallelism for fixed data set (speedup on perfect machine)

- **Working Sets**
  - Measures temporal locality: referenced again soon
  - Can prune design space

- **Communication to computation ratio**
  - Bytes transferred per unit of work
  - Latency and Bandwidth

- **Spatial locality:** nearby items referenced soon
  - Long cache lines
  - Prefetching
  - False Sharing

- **Want to study realistic regions**
  - Don’t use small data sets for large # of processors

---

Concurrency and Load Balance

Speedups for a perfect memory system
Impact of Synchronization

- Load imbalance from large synchronization waits

Working Sets

- Hierarchy of Working Sets
  - Must pick realistic operating points
    - Not small data sets with large caches
  - Understand how WS scales with app parameters
  - Can prune evaluation space

- Most SPLASH-2 fit in 1MB cache...
**Communication to Computation Ratio**

- Traffic is smaller for larger problem
- What must the machine support?

---

**Commercial Workloads**

- An important example of throughput parallelism
- Dominant user of MPs today
- Continuous, ongoing application
  - typically use dynamic load balancing
- Synchronization/communication typically not a problem
- Memory hierarchy effects are critical
Online Transaction Processing (OLTP)

- Banking system
- Updates to random accounts
  - locality within single transactions
- Small transactions, many updates
- Several important tables can be cached in SGA
- Server dedicated to client during transaction

Decision Support Systems (DSS)

- Wholesale supplier data base
- Answer big questions
  
  “select all goods ordered in a given period with a given discount and maximum quantity and compute the total sum of discounts”.
- Will sweep through large blocks of data
Web searching (AltaVista)

- We all pretty much know what this does
- 200 GB index database
- Memory management via OS, not application SW

OLTP and DSS use Oracle Database Engine

- Number of server processes (several per processor)
- #processes chosen to balance I/O
- daemon processes do housekeeping (e.g. update redo log) relatively light load
- Large, shared System Global Area
  - block buffer (software managed disk block buffer)
  - meta data
- Private Program Global Area per process
Hardware Platform

- 4 Alpha 21164 (2-way SS) processors
- 8KB direct mapped I-cache
- 8KB direct mapped D-cache
- 96KB shared 3-way L2 cache (on chip) 7 cycles
- 2MB direct mapped L3 cache (board level) 21 cycles
- 2 GB RAM 80 cycles
- Dirty miss 125 cycles
  (block in another processor's cache)
- Disk subsystem limited
  - requires some problem down-scaling

Study Workload

- OLTP
  - 40 branches, 900 MB data base (memory resident)
  - Oracle SGA 1 GB
  - most of I/O is redo log updates (not limiting)
  - 7 processes per processor
- DSS
  - 500 MB TPC-D database
  - Oracle SGA 1 GB
  - 4 processes per processor
    » (16-way parallelism for table operations)
  - 8 processes per processor
    » (for pipelined table operations)
- AltaVista
  - 10 minute runs from saved query log
Results

- Basic cycle breakdown (Fig. 3)
- CPI = cycles per instruction

![Basic cycle breakdown graph]

Results

- More detailed breakdown (Fig. 4)

![More detailed breakdown graph]
Results

- Cache performance Table 2

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>DSS-Q1</th>
<th>DSS-Q4</th>
<th>DSS-Q5</th>
<th>DSS-Q6</th>
<th>DSS-Q8</th>
<th>DSS-Q13</th>
<th>Altavista</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icache (global)</td>
<td>19.9%</td>
<td>9.7%</td>
<td>8.5%</td>
<td>4.6%</td>
<td>5.9%</td>
<td>3.7%</td>
<td>0.7%</td>
<td>1.8%</td>
</tr>
<tr>
<td>Dcache (global)</td>
<td>42.5%</td>
<td>6.9%</td>
<td>22.9%</td>
<td>11.9%</td>
<td>11.3%</td>
<td>11.0%</td>
<td>12.4%</td>
<td>7.6%</td>
</tr>
<tr>
<td>Scache (global)</td>
<td>13.9%</td>
<td>0.8%</td>
<td>2.3%</td>
<td>1.0%</td>
<td>0.6%</td>
<td>1.0%</td>
<td>1.0%</td>
<td>0.7%</td>
</tr>
<tr>
<td>Bcache (global)</td>
<td>2.7%</td>
<td>0.1%</td>
<td>0.5%</td>
<td>0.2%</td>
<td>0.2%</td>
<td>0.3%</td>
<td>0.3%</td>
<td>0.3%</td>
</tr>
<tr>
<td>Scache (local)</td>
<td>40.8%</td>
<td>3.6%</td>
<td>10.7%</td>
<td>7.7%</td>
<td>3.9%</td>
<td>6.0%</td>
<td>6.1%</td>
<td>7.6%</td>
</tr>
<tr>
<td>Bcache (local)</td>
<td>19.1%</td>
<td>13.0%</td>
<td>21.3%</td>
<td>23.9%</td>
<td>30.7%</td>
<td>27.9%</td>
<td>31.3%</td>
<td>41.2%</td>
</tr>
<tr>
<td>Dirty miss fraction</td>
<td>15.5%</td>
<td>2.3%</td>
<td>2.2%</td>
<td>10.6%</td>
<td>1.7%</td>
<td>8.4%</td>
<td>3.3%</td>
<td>15.8%</td>
</tr>
</tbody>
</table>

OLTP

- CPI = 7.0
  - Relatively few 2-issue cycles
- Very large number of instruction and data stalls
  - 75% of time stalled for memory
  - Half of stalls hit in L2 and L3 caches
- Workload “overwhelms” caches
  - I cache performance especially poor
- Dirty misses 15% of L3 cache misses
- Fraction of dirty misses increases with size of L3 cache and # cpus
DSS

- CPI = 1.5 to 1.9
- L2 cache captures main footprint
  - worst local miss rate is 10%
- Low dirty Bcache misses
- Biggest improvements likely to come from larger L1 caches

AV

- CPI = 1.3
- Instruction working set fits in Icache
- Data working set fits in L2 cache
- Bcache adds little value
  - 40% of references that reach Bcache miss
Review: Characterizing Applications

• Concurrency and Load balance
  – Measure inherent parallelism for fixed data set

• Working Sets
  – Measures temporal locality: referenced again soon

• Communication to computation ratio
  – Bytes transferred per unit of work (Latency & BW)

• Spatial locality: nearby items referenced soon
  – Long cache lines (implicit prefetching, false sharing)

• Want to study realistic regions
  – Don’t use small data sets for large # of processors