New-School Machine Structures (It’s a bit more complicated!)

Today’s Lecture

- **Parallel Requests**
  Assigned to computer
e.g., Search “Katz”

- **Parallel Threads**
  Assigned to core
e.g., Lookup, Ads

- **Parallel Instructions**
  >1 instruction @ one time
e.g., 5 pipelined instructions

- **Parallel Data**
  >1 data item @ one time
e.g., Add of 4 pairs of words

- **Hardware descriptions**
  All gates @ one time

8/8/2011

Summer 2011 -- Lecture #1
Review

• Cloud Computing
  – Benefits of WSC computing for all parties involved
  – “Elastic” pay as you go resource allocation
  – Amazon demonstrates that you can make money by selling cycles and storage

• Warehouse-Scale Computers (WSCs)
  – Power ultra large-scale Internet applications
  – Emphasis on cost, power efficiency
  – PUE - Ratio of total power consumed over power used for computing.
Agenda

• Request Level Parallelism
• MapReduce
• Administrivia
• MapReduce Processing
• Break
• Modern Microarchitectures
Request-Level Parallelism (RLP)

• Lots of independent tasks
  – In a WSC for a large internet app, hundreds to thousands per second.
    • Mostly involve read-only databases
    • Little read-write (aka “producer-consumer”) sharing
    • Rarely involve read–write data sharing or synchronization across requests

• Computation easily partitioned within a request and across different requests
Google Query-Serving Architecture
Anatomy of a Web Search

• Google “Randy H. Katz”
  – Direct request to “closest” Google Warehouse Scale Computer
  – Front-end load balancer directs request to one of many arrays (cluster of servers) within WSC
  – Within array, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
  – GWS communicates with Index Servers to find documents that contain the search words, “Randy”, “Katz”, uses location of search as well
  – Return document list with associated relevance score
Anatomy of a Web Search

• In parallel,
  - Ad system: books by Katz at Amazon.com
  - Images of Randy Katz
• Use docids (document IDs) to access indexed documents
• Compose the page
  - Result document extracts (with keyword in context) ordered by relevance score
  - Sponsored links (along the top) and advertisements (along the sides)
Anatomy of a Web Search

• Implementation strategy
  – Randomly distribute the entries
  – Make many copies of data (aka “replicas”)
  – Load balance requests across replicas

• Redundant copies of indices and documents
  – Breaks up hot spots, e.g., “Justin Bieber”
  – Increases opportunities for request-level parallelism
  – Makes the system more tolerant of failures
Agenda

• Request Level Parallelism
• MapReduce
• Administrivia
• MapReduce Processing
• Break
• Modern Microarchitectures
Data-Level Parallelism (DLP)

- Lots of data that can be operated on in parallel
- We’ve already seen SIMD instructions as a way to exploit this.
- What about larger scale DLP? Lots of data spread across many disks? WSC level quantities of data?
Problem Trying To Solve

• How process large amounts of raw data (crawled documents, request logs, ...) every day to compute derived data (inverted indicies, page popularity, ...) when computation conceptually simple but input data large and distributed across 100s to 1000s of servers?

• Challenge: Distribute and load balance data, tolerate faults.

MapReduce Solution

• Reduce the complexity of an otherwise complex parallel programming task by providing a rigid structure.

• Create a robust implementation
  – Fault tolerant, handles all the pesky details

• To solve problem, simply formulate it in the MapReduce format
  – Surprisingly versatile, can also chain together multiple MapReduce tasks.
Inspiration: cs61a map/reduce

- `(define (square x) (* x x))`
- `(reduce + (map square '(1 2 3 4)))`
- **Map** - Applies the operation “square” to every element in the list (result: ‘(1 4 9 16))
- **Reduce** - Combines the resulting data using the provided operator + (1 + 4 + 9 + 16 => 30).
MapReduce Format

• Apply Map function to user supplied record of key/value pairs
• Compute set of intermediate key/value pairs
• Apply Reduce operation to all values that share same key in order to combine derived data properly
• User supplies Map and Reduce operations
Data-Parallel “Divide and Conquer” (MapReduce Processing)

• Map:
  – Map: \((K, V) \rightarrow \text{list}( (K_1, V_1), (K_2, V_2), \ldots, (K_n, V_n))\)
    • Processes input key/value pair
    • Produces set of intermediate key/value pairs
  – Slice data into “shards” or “splits”, distribute these to workers, compute sub-problem solutions

• Reduce:
  – Reduce: \((K, \text{list}(V_1, V_2, \ldots, V_n)) \rightarrow (K, \text{combinedValue})\)
    • Combines all intermediate values for a particular key
    • Produces a set of merged output values (usually just one)
  – Collect and combine sub-problem solutions
MapReduce Execution

Fine granularity tasks: many more map tasks than machines

2000 servers =>
≈ 200,000 Map Tasks,
≈ 5,000 Reduce tasks
## MapReduce Popularity at Google

<table>
<thead>
<tr>
<th></th>
<th>Aug-04</th>
<th>Mar-06</th>
<th>Sep-07</th>
<th>Sep-09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MapReduce jobs</td>
<td>29,000</td>
<td>171,000</td>
<td>2,217,000</td>
<td>3,467,000</td>
</tr>
<tr>
<td>Average completion time (secs)</td>
<td>634</td>
<td>874</td>
<td>395</td>
<td>475</td>
</tr>
<tr>
<td>Server years used</td>
<td>217</td>
<td>2,002</td>
<td>11,081</td>
<td>25,562</td>
</tr>
<tr>
<td>Input data read (TB)</td>
<td>3,288</td>
<td>52,254</td>
<td>403,152</td>
<td>544,130</td>
</tr>
<tr>
<td>Intermediate data (TB)</td>
<td>758</td>
<td>6,743</td>
<td>34,774</td>
<td>90,120</td>
</tr>
<tr>
<td>Output data written (TB)</td>
<td>193</td>
<td>2,970</td>
<td>14,018</td>
<td>57,520</td>
</tr>
<tr>
<td>Average number servers /job</td>
<td>157</td>
<td>268</td>
<td>394</td>
<td>488</td>
</tr>
</tbody>
</table>
Google Uses MapReduce For ...

- Extracting the set of outgoing links from a collection of HTML documents and aggregating by target document
- Stitching together overlapping satellite images to remove seams and to select high-quality imagery for Google Earth
- Generating a collection of inverted index files using a compression scheme tuned for efficient support of Google search queries
- Processing all road segments in the world and rendering map tile images that display these segments for Google Maps
- Fault-tolerant parallel execution of programs written in higher-level languages across collections of input data
- More than 10,000 MR programs at Google in 4 years
### What if Ran Google Workload on EC2?

<table>
<thead>
<tr>
<th></th>
<th>Aug-04</th>
<th>Mar-06</th>
<th>Sep-07</th>
<th>Sep-09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of MapReduce jobs</td>
<td>29,000</td>
<td>171,000</td>
<td>2,217,000</td>
<td>3,467,000</td>
</tr>
<tr>
<td>Average completion time (secs)</td>
<td>634</td>
<td>874</td>
<td>395</td>
<td>475</td>
</tr>
<tr>
<td>Server years used</td>
<td>217</td>
<td>2,002</td>
<td>11,081</td>
<td>25,562</td>
</tr>
<tr>
<td>Input data read (TB)</td>
<td>3,288</td>
<td>52,254</td>
<td>403,152</td>
<td>544,130</td>
</tr>
<tr>
<td>Intermediate data (TB)</td>
<td>758</td>
<td>6,743</td>
<td>34,774</td>
<td>90,120</td>
</tr>
<tr>
<td>Output data written (TB)</td>
<td>193</td>
<td>2,970</td>
<td>14,018</td>
<td>57,520</td>
</tr>
<tr>
<td>Average number of servers per job</td>
<td>157</td>
<td>268</td>
<td>394</td>
<td>488</td>
</tr>
<tr>
<td>Average Cost/job EC2</td>
<td>$17</td>
<td>$39</td>
<td>$26</td>
<td>$38</td>
</tr>
<tr>
<td>Annual Cost if on EC2</td>
<td>$0.5M</td>
<td>$6.7M</td>
<td>$57.4M</td>
<td>$133.1M</td>
</tr>
</tbody>
</table>
Agenda

• Request Level Parallelism
• MapReduce
• Administrivia
• MapReduce Processing
• Break
• Modern Microarchitectures
Administrivia

• Final Review - Today, **4pm-6pm, 277 Cory.**

• HKN Course Surveys administered during lecture tomorrow.

• Project 3 Face-to-Face grading, Wednesday, 8/10 in 200 SD lab.

• Final Exam - Thursday, 8/11, 9am - 12pm 2050 VLSB
    • Use the back side of your midterm cheat sheet!
  – We’ll be holding some extra OH
     Tuesday/Wednesday, we’ll keep you posted.
CS61c in... Social Media

• “I’ve noticed a pattern. Knowledge of what is generally considered “low-level” programming is waning.”

• Cites important topics that programmers should know, among them: Floating Point, Fixed Point, Cache behavior, Bitwise operators, Branch Prediction.

• Inspired by this, I’ll talk about Fixed Point a bit tomorrow.

• Criticism: Also important to know when low level programming tricks are relevant. Not every task requires a performance programmer.

Agenda

• Request Level Parallelism
• MapReduce
• Administrivia
• MapReduce Processing
• Break
• Modern Microarchitectures
MapReduce Processing
Example: Count Word Occurrences

map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1"); // Produce count of words

reduce(String output_key, Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v); // get integer from key-value
  Emit(AsString(result));
Example: Count Word Occurrences

Distribute

that that is
Map 1
is 1, that 2

is that that
Map 2
is 1, that 2

is not is not
Map 3
is 2, not 2

is that it it is
Map 4
is 2, it 2, that 1

Shuffle

is 1,1,2,2
that 2,2,1
it 2
not 2

Reduce 1
is 6; it 2

Reduce 2
not 2; that 5

Collect

is 6; it 2; not 2; that 5
MapReduce Processing

MapReduce Processing

Shuffle phase
1. MR 1st splits the input files into $M$ “splits” then starts many copies of program on servers

**Shuffle phase**
2. One copy—the master—is special. The rest are workers. The master picks idle workers and assigns each 1 of M map tasks or 1 of R reduce tasks.

Shuffle phase
3. A map worker reads the input split. It parses key/value pairs of the input data and passes each pair to the user-defined map function.

(The intermediate key/value pairs produced by the map function are buffered in memory.)
4. Periodically, the buffered pairs are written to local disk, partitioned into $R$ regions by the partitioning function.
5. When a reduce worker has read all intermediate data for its partition, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.

(The sorting is needed because typically many different keys map to the same reduce task.)
MapReduce Processing

6. Reduce worker iterates over sorted intermediate data and for each unique intermediate key, it passes key and corresponding set of values to the user’s reduce function.

The output of the reduce function is appended to a final output file for this reduce partition.

Shuffle phase
MapReduce Processing

7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. The MapReduce call in user program returns back to user code.

Output of MR is in $R$ output files (1 per reduce task, with file names specified by user); often passed into another MR job.

Shuffle phase
MapReduce Processing Time Line

| Process       | Time ---------------| MapReduce() | ... wait ...
|---------------|---------------------|-------------|-----------------------|
| User Program  | MapReduce()         | Assign tasks to worker machines... | Map 1  | Map 3
| Master        |                     |             | Map 2                |
| Worker 1      |                     |             | Read 1.1  | Read 1.3
| Worker 2      |                     |             | Read 1.2 | Reduce 1
| Worker 3      |                     |             | Read 2.1 | Read 2.2 | Read 2.3
| Worker 4      |                     |             |                    | Reduce 2

- Master assigns map + reduce tasks to “worker” servers
- As soon as a map task finishes, worker server can be assigned a new map or reduce task
- Data shuffle begins as soon as a given Map finishes
- Reduce task begins as soon as all data shuffles finish
- To tolerate faults, reassign task if a worker server “dies”
Example MapReduce Job Running

• ~41 minutes total
  – ~29 minutes for Map tasks & Shuffle tasks
  – ~12 minutes for Reduce tasks
  – 1707 worker servers used
• Map (Green) tasks   read 0.8 TB, write 0.5 TB
• Shuffle (Red) tasks read 0.5 TB, write 0.5 TB
• Reduce (Blue) tasks read 0.5 TB, write 0.5 TB
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec
323 workers; 0 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>0</td>
<td>323</td>
<td>878934.6</td>
<td>1314.4</td>
<td>717.0</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>323</td>
<td>717.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>72.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>145825686</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>506631</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>508192</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>506631</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>1857</td>
<td>1707</td>
<td>878934.6</td>
<td>191995.8</td>
<td>113936.6</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>113936.6</td>
<td>57113.7</td>
<td>57113.7</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>57113.7</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>699.1</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>349.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5004411944</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17290135</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17331371</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17290135</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>5354</td>
<td>1707</td>
<td>878934.6</td>
<td>406020.1</td>
<td>241058.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>241058.2</td>
<td>196362.5</td>
<td>196362.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>196362.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>704.4</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>371.9</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5000364228</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17300709</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17342493</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17300709</td>
</tr>
</tbody>
</table>

8/8/2011

Summer 2011 -- Lecture #28
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>8841</td>
<td>1707</td>
<td>878934.6</td>
<td>621608.5</td>
<td>369459.8</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>369459.8</td>
<td>326986.8</td>
<td>326986.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>326986.8</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>706.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>419.2</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>4982870667</td>
</tr>
<tr>
<td>dups-in-indexmerge</td>
<td>17229926</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17272056</td>
</tr>
<tr>
<td>mr-operator-outsuts</td>
<td>17229926</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>195</td>
<td>305</td>
<td>523499.2</td>
<td>523389.6</td>
<td>523389.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>195</td>
<td>523389.6</td>
<td>2685.2</td>
<td>2742.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.3</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>45.7</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>2313178</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>7936</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>1954105</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>1954105</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>133837.8</td>
<td>136929.6</td>
</tr>
</tbody>
</table>

Counters

Variable               | Value |
-----------------------|-------|
Mapped (MB/s)          | 0.0   |
Shuffle (MB/s)         | 0.1   |
Output (MB/s)          | 1238.8|
doc-index-hits         | 0     |
docs-indexed           | 0     |
dups-in-index-merge    | 0     |
mr-merge-calls         | 51738599|
mr-merge-outputs       | 51738599|
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>263283.3</td>
<td>269351.2</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1225.1</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51842100</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51842100</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>390447.6</td>
<td>399457.2</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>520468.6</td>
<td>520468.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>406</td>
<td>94</td>
<td>520468.6</td>
<td>512265.2</td>
<td>514373.3</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>849.5</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>35083350</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>35083350</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>519781.8</td>
<td>519781.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>498</td>
<td>2</td>
<td>519781.8</td>
<td>519394.7</td>
<td>519440.7</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>9.4</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>394792</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>394792</td>
</tr>
</tbody>
</table>
MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec
1707 workers; 1 deaths

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>519774.3</td>
<td>519774.3</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>499</td>
<td>1</td>
<td>519774.3</td>
<td>519735.2</td>
<td>519764.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1.9</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>73442</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>73442</td>
</tr>
</tbody>
</table>
MapReduce Failure Handling

• On worker failure:
  – Detect failure via periodic heartbeats
  – Re-execute completed and in-progress map tasks
  – Re-execute in progress reduce tasks
  – Task completion committed through master

• Master failure:
  – Could handle, but don't yet (master failure unlikely)

• Robust: lost 1600 of 1800 machines once, but finished fine
MapReduce Redundant Execution

• Slow workers significantly lengthen completion time
  – Other jobs consuming resources on machine
  – Bad disks with soft errors transfer data very slowly
  – Weird things: processor caches disabled (!!)

• Solution: Near end of phase, spawn backup copies of tasks
  – Whichever one finishes first "wins"

• Effect: Dramatically shortens job completion time
  – 3% more resources, large tasks 30% faster
Impact of Redundant Execution, Failure for 10B record Sort using 1800 servers

- no backup tasks
- with backup tasks
- with 200 failures
Agenda

• Request Level Parallelism
• MapReduce
• Administrivia
• MapReduce Processing
• Break
• Modern Microarchitectures
Intel Nehalem

• Look at microprocessor from Intel in servers in your 61C labs and your laptops
• Nehalem is code name for microarchitecture at heart of Core i7 and Xeon 5500 series server chips
  – Intel legal said had to pick names of rivers for code names (vs. “Destroyer”)
• First released at end of 2008
• Die size 263mm² at 45 nm
• 731M transistors
Nehalem River, Oregon
Nehalem System Example:
Apple Mac Pro Desktop 2010

Each chip has three DRAM channels attached, each 8 bytes wide at 1.066Gb/s (3*8.5GB/s).
Can have up to two DIMMs on each channel (up to 4GB/DIMM)

PCI Express connections for Graphics cards and other extension boards. Up to 8 GB/s per slot.

Two Nehalem Chips ("Sockets"), each containing four processors ("cores") running at up to 2.93GHz

"QuickPath" point-point system interconnect between CPUs and I/O. Up to 25.6 GB/s per link.

Disk drives attached with 3Gb/s serial ATA link

Slower peripherals (Ethernet, USB, Firewire, WiFi, Bluetooth, Audio)
Nehalem 12-inch Wafer: 280 dies

Same diameter as medium Domino’s pizza
Nehalem Die Photo

Memory Controller

Core

Core

Core

Core

Shared L3 Cache

13.6 mm (0.54 inch)

18.9 mm (0.75 inch)

8/8/2011

Fall 2010 -- Lecture #38
Core Area Breakdown

Memory Controller

Out-of-Order Scheduling & Instruction Commit

Instruction Decode, Reg Renaming & Microcode

Memory Ordering & Execution

L1 Data cache

L1 Inst cache & Inst Fetch

Branch Prediction

Virtual Memory, TLB

L2 Cache & Interrupt Servicing

L3 Cache

Load Store Queue

Execution Units

Virtual Memory,
In-Order Fetch

In-Order Decode and Register Renaming

In-Order Commit

Out-of-Order Execution

Out-of-Order Completion

2 Threads per Core

8/8/2011 -- Lecture #38
Extending Performance and Energy Efficiency
- Intel® SSE4.2 Instruction Set Architecture (ISA) Leadership in 2008

Accelerated String and Text Processing
- Faster XML parsing
- Faster search and pattern matching
- Novel parallel data matching and comparison operations

Accelerated Searching & Pattern Recognition of Large Data Sets
- Improved performance for Genome Mining, Handwriting recognition, Fast Hamming distance / Population count

New Communications Capabilities
- Hardware based CRC instruction
- Accelerated Network attached storage
- Improved power efficiency for Software I-SCSI, RDMA, and SCTP

STTNI

SSE4 (45nm CPUs)

SSE4.1 (Penryn Core)

SSE4.2 (Nehalem Core)

ATA

STTNI e.g. XML acceleration

ATA (Application Targeted Accelerators)

POPCNT e.g. Genome Mining

CRC32 e.g. iSCSI Application

What should the applications, OS and VMM vendors do?:
Understand the benefits & take advantage of new instructions in 2008.
Provide us feedback on instructions ISV would like to see for next generation of applications
Intel® Hyper-Threading Technology

- Also known as Simultaneous Multi-Threading (SMT)
  - Run 2 threads at the same time per core
- Take advantage of 4-wide execution engine
  - Keep it fed with multiple threads
  - Hide latency of a single thread
- Most **power efficient** performance feature
  - Very low die area cost
  - Can provide significant performance benefit depending on application
  - Much more efficient than adding an entire core
- Intel® Core™ microarchitecture (Nehalem) advantages
  - Larger caches
  - Massive memory BW

Simultaneous multi-threading enhances performance and energy efficiency
µOP is Intel name for internal RISC-like (MIPS) instruction, into which x86 instructions are translated.

Loop Stream Detector (can run short loops out of the buffer)
x86 Decoding

• Translate up to 4 x86 instructions into μOPS (≈MIPS or RISC instructions) each cycle
• Only first x86 instruction in group can be complex (maps to 1-4 μOPS), rest must be simple (map to one μOP)
• Even more complex instructions, jump into microcode engine which spits out stream of μOPS
Branch Prediction

- Part of instruction fetch unit
- Several different types of branch predictor
  - Details not public
- Two-level Branch Table Buffer
- Loop count predictor
  - How many backwards taken branches before loop exit
- Return Stack Buffer
  - Holds subroutine targets
  - Separate return stack buffer for each SMT thread
Loop Stream Detectors save Power

Intel® Core™ 2 Loop Stream Detector

Intel Core Microarchitecture (Nehalem) Loop Stream Detector

8/8/2011
Out-of-Order Execution Engine

Renaming happens at uOP level (not original macro-x86 instructions)
Multithreading effects in Out-of-Order Execution Core

• Reorder buffer (remembers program order and exception status for in-order commit) has 128 entries divided statically and equally between both threads

• Reservation stations (instructions waiting for operands for execution) have 36 entries competitively shared by threads
Nehalem Memory Hierarchy Overview

- Private L1/L2 per core
- Local memory access latency ~60ns
- Each DRAM Channel is 64/72b wide at up to 1.33Gb/s
- Each direction is 20b@6.4Gb/s
- 4-8 Cores
- L3 fully inclusive of higher levels (but L2 not inclusive of L1)
- Other sockets’ caches kept coherent using QuickPath messages
Non-Uniform Memory Access (NUMA)

- FSB architecture
  - All memory in one location
- Starting with Intel® Core™ microarchitecture (Nehalem)
  - Memory located in multiple places
- Latency to memory dependent on location
- Local memory has highest BW, lowest latency
- Remote Memory still very fast

**Ensure software is NUMA-optimized for best performance**
All Sockets can Access all Data

How ensure that get data allocated to local DRAM?

OS doesn’t allocate pages to physical memory after malloc until first access to page. Be sure to touch what each CPU wants nearby.

~60ns

Local Memory Access

~100ns

Remote Memory Access

Such systems called “NUMA” for Non Uniform Memory Access: some addresses are slower than others.
Core’s Private Memory System

Load queue 48 entries
Store queue 32 entries
Divided statically between 2 threads
Up to 16 outstanding misses in flight per core
What to do with So Many Features?

- “Introduction to Performance Analysis on Nehalem Based Processors”, 72 pages

“Software optimization based on performance analysis of large existing applications, in most cases, reduces to optimizing the code generation by the compiler and optimizing the memory access. Optimizing the code generation by the compiler requires inspection of the assembler of the time consuming parts of the application and verifying that the compiler generated a reasonable code stream. Optimizing the memory access is a complex issue involving the bandwidth and latency capabilities of the platform, hardware and software prefetching efficiencies and the virtual address layout of the heavily accessed variables.”
Summary

• Request-Level Parallelism
  – High request volume, each largely independent of other
  – Use replication for better request throughput, availability

• MapReduce Data Parallelism
  – Divide large data set into pieces for independent parallel processing
  – Combine and process intermediate results to obtain final result

• Intel Nehalem
  – Speculative execution: branch prediction, out of order execution, data prefetching
  – Hardware translation and optimization of instruction sequences
  – Opportunistic acceleration (Turbo Mode)
Acknowledgements

• Figures/Info from Intel, David Kanter at Real World Technologies.
• These slides contain material developed and copyright by:
  – Krste Asanovic (UCB)
  – Beeman Strong (Intel)
• UCB material derived from course CS152.