Performance evaluation of hash join algorithms on chip multiprocessors

Spyros Blanas
Hash join in detail

inner

\( h(x) \)
Hash join in detail

inner

$h(x)$
Hash join in detail

inner

$h(x)$
Hash join in detail
Hash join in detail

inner

\[ h(x) \]
Hash join in detail

inner

\[ h(x) \]
Hash join in detail

inner

h(x)
Hash join in detail
Hash join in detail
Hash join in detail
Hash join in detail
Hash join in detail

outer

$h(x)$
Hash join in detail

outer

h(x)

---

Detail of hash join algorithm.
Hash join in detail
Algorithms

- Parallel hash join
- Buffered hash join
- Column-oriented hash join
Prototype

• ~3000 lines of C++ code
  - More than 50% deals with I/O, abstract data type support and instrumentation

• On each run:
  - Reads configuration file with join parameters
  - Reads input in memory
  - Joins data
  - Outputs statistics and join result (for validation)
Dataset

- Used TPC-H, query 5
• Used TPC-H, query 5
Dataset

- Used TPC-H, query 5
Workload characteristics

- Q1 is very asymmetric:
  - Inner: 2,036 tuples each 36 bytes
  - Outer: 6,001,125 tuples each 24 bytes

- Q2 has a “narrow” inner:
  - Inner: 227,597 tuples each 8 bytes
  - Outer: 1,222,276 tuples each 50 bytes
Runtime vs. chunk size

Execution time (msec)

Chunk size (KB)
Runtime vs. number of buckets
Runtime vs. bucket size

![Graph showing the relationship between execution time (msec) and bucket size (KB). The graph indicates that execution time decreases as bucket size increases, reaching a minimum at around 100 KB before stabilizing.](image-url)
Speedup: Pentium4 HT

![Graph showing speedup with two lines, Q1 and Q2, and two markers for each line. The x-axis represents the number of threads, and the y-axis represents speedup. The graph shows a slight decrease in speedup as the number of threads increases.](image-url)
Speedup: Clovertown

![Graph showing speedup vs number of threads for Q1 and Q2. The graph has speedup on the y-axis and number of threads on the x-axis. The lines for Q1 and Q2 show increasing speedup with more threads.]

- Q1 line
- Q2 line
Speedup: UltraSPARC T1

![Graph showing speedup vs. number of threads for Q1 and Q2. The graph indicates a linear relationship between speedup and the number of threads, with Q2 showing slightly better performance at higher thread counts.]
Absolute performance: Q1

- normal
- buffered
- column

<table>
<thead>
<tr>
<th></th>
<th>Pentium4 HT</th>
<th>UltraSPARC T1</th>
<th>Clovertown</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>buffered</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>column</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Absolute performance: Q2

- Normal
- Buffered
- Column

Performance comparison for Pentium4 HT, UltraSPARC T1, and Clovertown (from left to right).
Absolute performance: Q2

- normal
- buffered
- column

Comparison of performance across different processors:
- Pentium4 HT
- UltraSPARC T1
- Clovertown
Prefetching & big pages on UltraSPARC T1
Conclusions

• Hash join can parallelize very well

• Clovertown outperforms both Pentium4 HT and UltraSPARC T1

• Copying data introduces significant overhead and limits hash join performance

• Column-oriented join is a promising solution
  - Also allows for delayed materialization of the result
Future work

• The two input datasets clearly fail to capture every aspect of the problem
  - Will also give additional insight on the factors limiting performance

• Measurement of total execution time for queries with multiple joins
  - Experimenting with pipelined joins