PERFORMANCE OF RDF QUERY PROCESSING ON THE INTEL SCC

Vasil Slavov, Praveen Rao, Dinesh Barenkala, Srivenu Paturi

Department of Computer Science & Electrical Engineering
University of Missouri-Kansas City, USA

Acknowledgements
Single-chip Cloud Computer Research Program, Intel Labs
National Science Foundation (IIS-1115871)
Overview

♦ Introduction
♦ Background
  ♦ RDF and SPARQL
  ♦ Prior work on RDF query processing
♦ Our methodology
♦ Performance evaluation
♦ Conclusion and future work
Introduction (1/2)

♦ The RDF data model
  ♦ Important in domain-specific applications and the WWW
    ♦ Knowledge representation and reasoning
    ♦ E.g., healthcare, defense and intelligence, biopharmaceuticals, Linked Data

♦ Very large RDF datasets are available today
  ♦ DBPedia [WWW ‘07, ISWC ‘07]
  ♦ YAGO2 [WWW ‘11]
  ♦ Billion Triples Challenge (http://challenge.semanticweb.org)
  ♦ Uniprot RDF (http://www.uniprot.org/downloads)
Introduction (2/2)

♦ Pressing need for high performance RDF processing tools

Can a manycore processor boost the performance of RDF query processing through parallel processing?
Background (1/3)

♦ RDF

♦ Triple format: (subject, predicate, object)
♦ Represents a directed, labeled graph

Example

```xml
<Intel_Corporation> rdf:type <wikicategory_Companies_established_in_1968> .
<Intel_Corporation> rdf:type <wikicategory_Motherboard_companies> .
<Intel_Corporation> rdf:type <wikicategory_Multinational_companies> .
<Intel_Corporation> rdf:type <wikicategory_Netbook_manufacturers> .
<Intel_Corporation> rdf:type <wikicategory_Semiconductor_companies> .
<Intel_Corporation> y:created <IA-32_Execution_Layer> .
<Intel_Corporation> y:created <Itanium> .
<Intel_Corporation> y:created <LightPeak> .
```
Background (2/3)

♦ SPARQL
  ♦ A popular query language for RDF

Example

```sparql
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix y: <http://www.mpii.de/yago/resource/> .

SELECT ?a ?n WHERE
{
  ?a rdf:type <wikicategory_Motherboard_companies> .
  ?a y:created ?n .
}
```

<table>
<thead>
<tr>
<th>?a</th>
<th>?n</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Intel_Corporation&gt;</td>
<td>&lt;IA-32_Execution_Layer&gt;</td>
</tr>
<tr>
<td>&lt;Intel_Corporation&gt;</td>
<td>&lt;Itanium&gt;</td>
</tr>
<tr>
<td>&lt;Intel_Corporation&gt;</td>
<td>&lt;Light_Peak&gt;</td>
</tr>
</tbody>
</table>
Background (3/3)

♦ Prior work on RDF query processing
  ♦ Using an RDBMS
    ♦ Sesame [ISWC ‘02]
    ♦ Jena2 [SWDB ’03]
    ♦ RDF_MATCH [VLDB ’05]
    ♦ Vertical partitioning [VLDB ’07]
  ♦ Native RDF databases
    ♦ Hexastore [VLDB ‘08]
    ♦ RDF-3X [VLDB ‘08, VLDB Journal ‘10]
    ♦ BitMat [WWW ‘10]
  ♦ Shared-nothing clusters
    ♦ YARS2 [ISWC ‘07], 4store
    ♦ Tools built using Apache Hadoop and Apache Pig

♦ On the Intel SCC (but not RDF)
  ♦ Parallel AI planning [MARC ‘11]
  ♦ Relational decision support queries [MARC ‘11]
Our Methodology

♦ Programming models
  ♦ Task (inter-query) parallelism
  ♦ Data (intra-query) parallelism

♦ I/O bound queries
  ♦ Small I/O footprint
  ♦ Large I/O footprint

♦ Message Passing Interface (MPI)
  ♦ MPI_Send, MPI_Recv
  ♦ MPI_Barrier
  ♦ MPI_Bcast, MPI_Scatter, MPI_Gatherv
Models

![Diagram with categories: Large I/O and Small I/O, Task parallelism and Data parallelism, with LT, LD, ST, and SD points.]
Task Parallel Programming Model

1. Request a query
2. Send a query
3. Process the query using the index

Master (core 0)
Task pool of queries

SCC die

Worker
Worker
Worker
Worker

Single index shared by all the workers, stored on an NFS mounted filesystem

up to 47 workers
Data Parallel Programming Model (1/2)

1. Send the query to each worker along with a different partition id.
2. Each worker queries a different index to process the query.
3. Collect partial results from workers.

Partitioned indexes, stored on an NFS mounted filesystem.
Data Parallel Programming Model (2/2)

♦ Partition the RDF graph into smaller graphs
  ♦ Extract weakly connected directed subgraphs
  ♦ Apply standard graph partitioning techniques (e.g., METIS [SIAM ‘98])
  ♦ We may miss results

♦ Collect partial results
  ♦ Multiple MPI_Recv
  ♦ Single MPI_Gatherv
Performance Evaluation

♦ RDF-3X [VLDB ‘08, VLDB Journal ‘10]
♦ RCKMPI (a modified MPICH2 for Intel SCC)
♦ Tile_Mesh_DDR: 800MHz, 800MHz, 800MHz
♦ 2GB index size (limit of OS)
♦ Indexes – stored on an NFS mounted filesystem
♦ Cold cache
Datasets

♦ Real datasets
  ♦ YAGO2 (27,331,797 triples)
  ♦ Uniprot (46,972,851 triples)

♦ Synthetic dataset
  ♦ LUBM (35,612,176 triples) [WWW ‘05]

♦ Data partitioning
  ♦ LUBM (based on RDF files)
  ♦ Uniprot (based on protein fragments)
  ♦ YAGO2
    ♦ Star-shaped graphs
    ♦ METIS
## YAGO: Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>I/O footprint</th>
<th>Type</th>
<th>% CPU</th>
<th>Serial time</th>
</tr>
</thead>
<tbody>
<tr>
<td>QY₁</td>
<td>14,756 KB</td>
<td>small</td>
<td>29</td>
<td>4.73 secs</td>
</tr>
<tr>
<td>QY₂</td>
<td>15,004 KB</td>
<td>small</td>
<td>40</td>
<td>9.23 secs</td>
</tr>
<tr>
<td>QY₃</td>
<td>22,832 KB</td>
<td>small</td>
<td>29</td>
<td>6.51 secs</td>
</tr>
<tr>
<td>QY₄</td>
<td>33,492 KB</td>
<td>small</td>
<td>21</td>
<td>9.27 secs</td>
</tr>
<tr>
<td>QY₅</td>
<td>216,564 KB</td>
<td>large</td>
<td>22</td>
<td>82.65 secs</td>
</tr>
<tr>
<td>QY₆</td>
<td>272,848 KB</td>
<td>large</td>
<td>30</td>
<td>120.08 secs</td>
</tr>
<tr>
<td>QY₇</td>
<td>332,944 KB</td>
<td>large</td>
<td>43</td>
<td>218.43 secs</td>
</tr>
</tbody>
</table>
# LUBM: Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>I/O footprint</th>
<th>Type</th>
<th>% CPU</th>
<th>Serial time</th>
</tr>
</thead>
<tbody>
<tr>
<td>QL₁</td>
<td>2,668 KB</td>
<td>small</td>
<td>25</td>
<td>1.4 secs</td>
</tr>
<tr>
<td>QL₂</td>
<td>3,132 KB</td>
<td>small</td>
<td>35</td>
<td>1.47 secs</td>
</tr>
<tr>
<td>QL₃</td>
<td>9,804 KB</td>
<td>small</td>
<td>19</td>
<td>3.5 secs</td>
</tr>
<tr>
<td>QL₄</td>
<td>636,204 KB</td>
<td>large</td>
<td>32</td>
<td>299.99 secs</td>
</tr>
<tr>
<td>QL₅</td>
<td>673,924 KB</td>
<td>large</td>
<td>29</td>
<td>206.58 secs</td>
</tr>
</tbody>
</table>
### Uniprot: Queries

<table>
<thead>
<tr>
<th>Query</th>
<th>I/O footprint</th>
<th>Type</th>
<th>% CPU</th>
<th>Serial time</th>
</tr>
</thead>
<tbody>
<tr>
<td>QU₁</td>
<td>4,468 KB</td>
<td>small</td>
<td>39</td>
<td>2.08 secs</td>
</tr>
<tr>
<td>QU₂</td>
<td>10,344 KB</td>
<td>small</td>
<td>39</td>
<td>6.46 secs</td>
</tr>
<tr>
<td>QU₃</td>
<td>48,020 KB</td>
<td>large</td>
<td>31</td>
<td>19.39 secs</td>
</tr>
<tr>
<td>QU₄</td>
<td>62,188 KB</td>
<td>large</td>
<td>19</td>
<td>15.48 secs</td>
</tr>
<tr>
<td>QU₅</td>
<td>166,808 KB</td>
<td>large</td>
<td>17</td>
<td>43.51 secs</td>
</tr>
</tbody>
</table>
ST Model (small I/O footprint, task parallelism)
ST Model Load Distribution

Mean (# of tasks/worker)

![Graph showing the mean number of tasks per worker across different numbers of cores for YAGO, LUBM, and Uniprot datasets.](image)

Standard deviation

![Graph showing the standard deviation across different numbers of cores for YAGO, LUBM, and Uniprot datasets.](image)
LT Model (large I/O footprint, task parallelism)

**Speedup**

![Speedup Graph](chart)

**Efficiency**

![Efficiency Graph](chart)
LT Model Load Distribution

Mean (# of tasks/worker)

Standard deviation

![Graphs showing mean and standard deviation over the number of cores for YAGO, LUBM, and Uniprot.](image)
CPU Usage

![Graph showing CPU usage for different datasets and core counts.]

- YAGO (ST)
- Uniprot (ST)
- LUBM (ST)
- YAGO (LT)
- Uniprot (LT)
- LUBM (LT)
SD Model (small I/O footprint, data parallelism)

**Speedup**

![Speedup graph](image)

**Efficiency**

![Efficiency graph](image)
LD Model (large I/O footprint, data parallelism)

**Speedup**

- YAGO
- LUBM
- Uniprot

**Efficiency**

- YAGO
- LUBM
- Uniprot
Conclusion and Future Work

♦ Task parallel programming yields good speedup and efficiency
  ♦ For both large I/O and small I/O footprint queries

♦ Data parallel programming yields poor speedup and efficiency due to
  ♦ Load imbalance or I/O contention

♦ Future work
  ♦ New methods to overcome the challenges posed by the data parallel programming model
  ♦ Effect of dynamic voltage and frequency scaling on the performance of RDF query processing
Questions?

♦ Acknowledgements
  ♦ Single-chip Cloud Computer Research Program, Intel Labs
  ♦ National Science Foundation (IIS-1115871), 2011-2014
♦ For further information, contact Prof. Praveen Rao (raopra@umkc.edu)