Architectural Characterization of XQuery Workloads on Modern Processors

Rubao Lee, Bihui Duan, Taoying Liu
Research Centre for Grid and Service Computing,
Institute of Computing Technology, Chinese Academy of Sciences
PO Box 2704, Beijing, China
{lirubao,duanbihui,liutaoying}@software.ict.ac.cn

ABSTRACT
As XQuery rapidly emerges as the standard for querying XML documents, it is very important to understand the architectural characteristics and behaviors of such workloads. A lot of efforts are focused on the implementation, optimization, and evaluation of XQuery tools. However, few or no prior work studies the architectural and memory system behaviors of XQuery workloads on modern hardware platforms. This makes it unclear whether modern CPU techniques, such as the multi-level caches and hardware branch predictors, can support such workloads well enough.

This paper presents a detailed characterization of the architectural behavior of XQuery workloads. We examine four XQuery tools on three hardware platforms (AMD, Intel, and Sun) using well-designed XQuery queries. We report measured architectural data, including the L1/L2 cache misses, TLB misses, and branch mispredictions. We believe that the information will be useful in understanding XQuery workloads and analyzing the potential architectural optimization opportunities of improving XQuery performance.

1. INTRODUCTION
The wide spread of XML storages and web service creates a lot of applications which need to query and process XML documents. More than traditional scientific computing applications and OLTP/DSS workloads [2], such applications run on a larger range of hardware platforms, which include web servers, desktops, and even mobile devices. As XQuery rapidly emerges as the standard for querying XML documents, it is very important to understand the architectural characteristics and behaviors of running such workloads on different hardware platforms.

However, the current architectural research community and the current XQuery research community are not well joining to study and make it clear how XQuery workloads run on modern processors. Architectural researchers have focused on studying the architectural characteristics of database workloads including query processing and transaction processing. Furthermore, architectural optimized database algorithms and schemes [24][25][26] are presented on the basis of insights given by DBMS-characterizing works. Compared with the joint of architecture research and database research, researching XQuery from the viewpoint of architecture is just beginning.

Proceedings of the Second International Workshop on Performance and Evaluation of Data Management Systems (ExpDB 2007), June 15, 2007, Beijing, China.
Copyright 2007 ACM 978-1-59593-773-5/07/06 ...$5.00.

Although XQuery shares many common concepts with SQL, its executions have many specific features differing from them in relational database. Intuitively, XQuery is basically computing and memory bound since (1) XML documents are relatively small (<100MB) in typical applications and as thus the I/O is not the dominating factor, and (2) executions of XQuery queries are often time-consuming, which need to manipulate a lot of nodes resident in memory. Path navigation is the cornerstone of XQuery, which creates different memory-accessing patterns from DBMS's tuple-based query executions. In addition, current XQuery engines are often written in an object-oriented language and running on a virtual machine, such as typical Java-based systems [20][22]. These factors make it necessary to characterize the CPU and memory behavior of XQuery workloads.

This paper is our first step to understand the architectural characteristics of running XQuery workloads on modern processors. We present a detailed characterization of architectural behaviors of XQuery workloads. We examined four XQuery tools on three hardware platforms (AMD, Intel, and Sun) using our designed XQuery queries. We report measured architectural data, including the L1/L2 cache misses, TLB misses, and branch mispredictions. We believe that the information will be useful in understanding XQuery workloads and analyzing the potential architectural optimization opportunities of improving XQuery performance.

The remainder of this paper is organized as follows. Section 2 lists related work very briefly. Section 3 introduces our designed XQuery workloads. In section 4, we introduce the target hardware platforms and corresponding tools. All test results are presented in section 5. We conclude this paper in section 6.

2. RELATED WORK
There are many papers for characterizing architectural behaviors of DBMS queries including OLTP and DSS workloads, such as [1][2][3][4][5][6]. Some papers involve examining Java workloads, such as [7][8][9]. To our knowledge, almost no prior work is specially focused on charactering XQuery workloads. In [10], the authors report the measured architectural characteristics for XML processing on an Intel Xeon platform. In addition, work on benchmarking XQuery is related to this paper, such as [16][17][18][19][28][29].

3. XQUERY WORKLOAD DESIGN

In this section, we introduce the XQuery workloads used in our experiments, which include the data part and the query part.

3.1 Documents

We design three categories of XML documents. Their visual shapes are illustrated by Figure 1.

3.1.1 Rectangle XML

Rectangle XML has two parameters: the *width* and the *height*. The width means how many child nodes the root has, while the height means how many levels the tree has under the root node.

We distinguish two kinds of Rectangle XML: the *wide* one and the *narrow* one. A wide Rectangle XML has a very large width but a very small height, while a narrow one has a very small width but a very large height. Although the narrow Rectangle XML is very rare, we are expecting that the difference between the two shapes can show different memory system behaviors considering their structural difference.

The concrete values of the two parameters are given as the following.

Wide Rectangle XML:

Width: [10000,20000,40000,80000,100000]

Height: [1,2,4,6,8,10]

Narrow Rectangle XML:

Width: [1,2,4,6,8,10]

Height: [500,1000,5000,10000,20000,40000]

In a Rectangle XML document, the root is labeled *root*. Nodes at the level *x* are all labeled *tx*. No node contains attribute.

3.1.2 Triangle XML

In a Triangle XML document, each non-leaf node has two child nodes, so the tree is a complete binary tree. Triangle XML has only one parameter: the *height*. The concrete values of heights are given as the following.

Triangle XML:

Height: [15,16,17,18,19,20]

In a Triangle XML document, the root is labeled *t1*. Nodes at the level *x* are all labeled *tx*. No node has attribute.

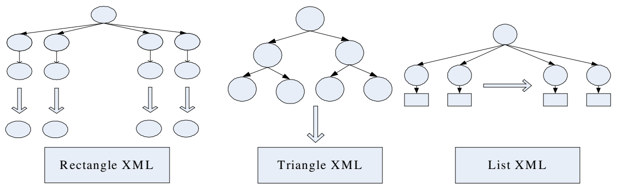

Figure 1: visual shapes of tree kinds of XML documents. In the graphs, a circle means an element node, while a rectangle means a text node (only existing in List XML).

3.1.3 List XML (300k.xml)

Unlike the above Rectangle XML and Triangle XML, List XML has payloads and it is defined as the following:

(1) The root node has 300000 child nodes and each child node has a text child node whose value is a random integer between 1 and 300000.

(2) The root node is labeled *result* and all 300000 child nodes are labeled *t*.

3.2 Queries

In our experiments, we only use three queries which contain basic operations of XQuery: path navigation, selection and sorting, considering our major goal is not to evaluate language supporting capabilities of concrete tools, but to understand and analyze architectural behaviors of basic XQuery workloads.

Q1: Retrieving all leaf nodes This query is actually an XPath query. It is executed on Rectangle XML and Triangle XML. The query has the form of "*doc("xmlname")//tn*", in which "*n*" is the level of all leaf nodes in the corresponding document "*xmlname*".

Q2: Selection This query is executed only on List XML. The selectivity of the where clause in the query is 50%.

```
for $t in doc("300k.xml")/result//t
where number($t/child::node()) < 150000
return  $t
```

Q3: Sorting This query is executed only on List XML.

```
for $t in doc("300k.xml")/result//t
order by number($t/child::node())
 return  $t
```

Table 1: Characteristics of three CPUs. (The data is from corresponding official websites [11][14][15].)

Characteristic	AMD Sempron 2500+	Intel Pentium P4 2.8GHz	Sun UltraSPARC T1
L1 cache organization	Split instruction/data caches	Split instruction/data caches	Split instruction/data caches
L1 cache size	64 KB each for instructions/data	8KB for data, 96KB trace cache	16KB instruction cache per core; 8KB data cache per core
L1 cache associativity	2-way set associative	4-way set associative	4-way set associative
L1 block size	64 bytes	64 bytes	32 bytes(instruction cache) 16 bytes(data cache)
L2 cache organization	Unified (instruction and data)	Unified (instruction and data)	Unified (instruction and data)
L2 cache size	256 KB	512 KB	3 MB (shared by cores)
L2 cache associativity	16-way set associative	8-way set associative	12-way set associative
L2 block size	64 bytes	128 bytes	64 bytes

4. HARDWARE & SOFTWARE

4.1 Hardware Platforms and Measuring Tools

Our experiments were executed on three hardware platforms. The specifications of involved CPUs are briefly given in Table 1.

4.1.1 AMD

This system contains an AMD Sempron 2500+ CPU, 2048MB DDR400 RAM. The operating system of this system is FreeBSD 6.2. We use the pmcstat tool [12] to count hardware events.

This system is our primary platform in the experiments, on which we measure the following architectural characteristics:

(1) L2 miss rate
(2) Frequency of branch mispredictions
(3) L1 I-TLB misses per 1000 instructions
(4) L1 D-TLB misses per 1000 instructions
(5) L1 I-Cache misses per 1000 instructions
(6) L1 D-Cache misses per 1000 instructions

To measure these targets, we need to count events listed in Table 2. We use the indirect method [13] to measure L2 cache request/misses and calculate L2 miss rate.

Table 2: events used by pmcstat on AMD Sempron

Characteristics	Event name (used by the pmcstat tool)
branches	Branches (common alias)
Branch mispredictions	Branch-mispredicts (common alias)
instructions	Instructions (common alias)
L1 I-TLB misses	K8-ic-l1-itlb-miss-and-l2-itbl-hit + K8-ic-l1-itlb-miss-and-l2-itlb-miss
L1 D-TLB misses	K8-ic-l1-dtlb-miss-and-l2-dtlb-hit + K8-ic-l1-dtlb-miss-and-l2-dtlb-miss
L1 I-Cache misses	K8-ic-refill-from-l2 + K8-ic-refill-from-system
L1 D-Cache misses	K8-dc-refill-from-l2 + K8-dc-refill-from-system
L2 Cache accesses	L1 I-TLB misses + L1 D-TLB misses + L1 I-Cache misses + L1 D-Cache misses
L2 Cache misses	K8-ic-refill-from-system + k8-dc-refill-from-system + K8-ic-l1-itlb-miss-and-l2-itlb-miss + K8-ic-l1-dtlb-miss-and-l2-dtlb-miss

4.1.2 Intel

This system contains an Intel Pentium P4 2.8GHz CPU, 512MB DDR266 memory. Although this CPU supports Hyper-Threading technology, we disabled this feature in our experiments. Like the above AMD system, we also run FreeBSD 6.2 operating system and use pmcstat tool on this system.

On this system, we only measure the following architectural characteristics using the events listed in Table 3 [11]:

(1) L2 miss rate
(2) Frequency of branch mispredictions

Table 3: events used by pmcstat on Intel P4

Characteristics	Event name (used by the pmcstat tool)
L2 cache accesses	P4-bsq-cache-reference
L2 Misses	P4-bsq-cache-reference,mask=rd-2ndl-miss+wr-2ndl-miss
branches	Branches (common alias)
Branch mispredictions	Branch-mispredicts (common alias)

4.1.3 Sun UltraSPARC T1

This system is a Sun Fire T1000 Server with 8 cores, 8GB RAM, running Solaris 10. On this system, we use the *collect* and *er_print* command provided by Sun Studio to measure instructions, and misses of L1/L2 instruction/data cache [27].

4.2 Software

We examined the following XQuery tools:

- Berkeley DB XML v2.3.10 (Bdb-xml for short) [23]

We only ran this software on our AMD/Intel systems. We downloaded the source tarball which includes sources of all the components: Berkeley DB, Xerces, and XQilla, and build them on our FreeBSD systems. The query execution command was "*dbxml –s ourquery*", in which the "*ourquery*" was a script containing a *query* command and a *quit* command. For our experiments, we did not use the container concept but build our queries on XML files directly.

- Galax v0.5.0. (Galax for short) [21]

We ran it on all systems. We used the Linux binary version on our FreeBSD systems without any modifications. And on the Sun system, we used the Solaris binary version downloaded from the official website. The query executing command was "*Galax-run –print-xml off our.xquery*".

- Saxon-B 8.9 for Java. (Saxon for short) [20]

This is written in Java. We ran it only on the Sun system. The query executing command was "*java net.sf.Saxon.Query our.xquery*".

- Gnu Qexo v1.9.1. (Qexo for short) [22].

This is also written in Java. We ran it only on the Sun system. We downloaded the executable jar file and executed it directly, and the command was "*java –jar kawa.jar our.xquery*".

5. RESULTS

In this section, we first report results on AMD and Intel systems for Bdb-xml and Galax, which include the following three parts:

(1) The L2 Cache miss rates on both AMD and Intel systems (subsection 5.1)
(2) The frequency of branch mispredictions on both AMD and Intel systems (subsection 5.2)
(3) The L1 Cache misses and TLB misses on only AMD system (subsection 5.3)

Then, we report results of running Q1 to wide Rectangle XML documents on the Sun UltraSPARC T1 system for Galax, Saxon, and Qexo (subsection 5.4), which include the following parts:

(1) The total count of instructions
(2) The L1 I-Cache/D-Cache misses per 1000 instructions
(3) The L2 I-Cache/D-Cache misses per 1000 instructions

5.1 L2 Cache Miss Rate

5.1.1 AMD

On this system, we measure Sempron's L2 Cache miss rates for running Q1 on Rectangle XML and Triangle XML, and running Q2/Q3 on List XML.

5.1.1.1 Rectangle XML

Figure 2 shows L2 miss rates of executing Q1 on wide Rectangle XML in Bdb-xml, while Figure 3 shows them in Galax. By comparing these two graphs, we can see that Bdb-xml has a better L2 miss rate than Galax for wide XML documents. Moreover, when the width increases from 10000 to 100000, the L2 miss rate decreases for Bdb-xml but increases for Galax (not including height 1/2).

Figure 4 shows L2 miss rates of executing Q1 on narrow Rectangle XML in Bdb-xml, while Figure 5 shows them in Galax. We can see that Bdb-xml has a very bad L2 miss rate for this situation, especially when the height is larger than 5000 (up to 50%). What is interesting is that the miss rate in Bdb-xml is minimized when the height equals to 1000.

As shown in the graphs for Galax, no matter whether the Rectangle XML is wide or narrow, Galax has a relatively close L2 miss rate (8% -13%). However, Bdb-xml cannot fit narrow XML shapes as well as to wide shapes (ranging from 2% to 50%).

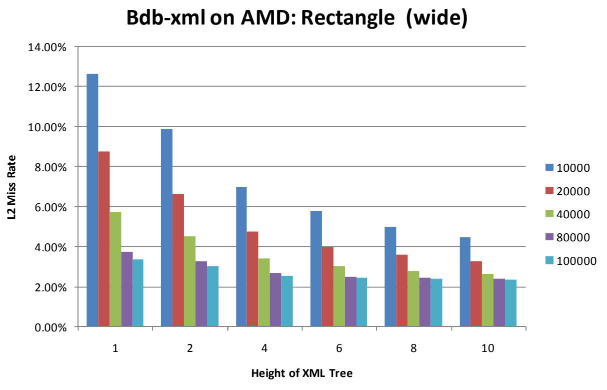

Figure 2: L2 miss rates of executing Q1 on wide Rectangle XML documents in Bdb-xml (varying width and height)

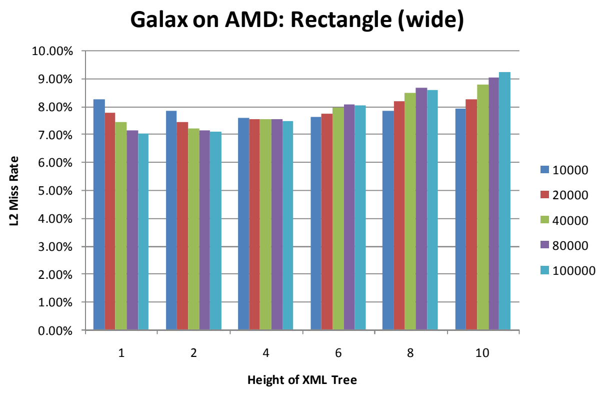

Figure 3: L2 miss rates of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

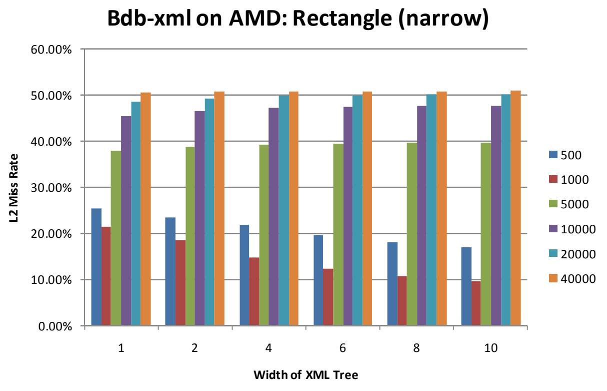

Figure 4: L2 miss rates of executing Q1 on narrow Rectangle XML documents in Bdb-xml (varying width and height)

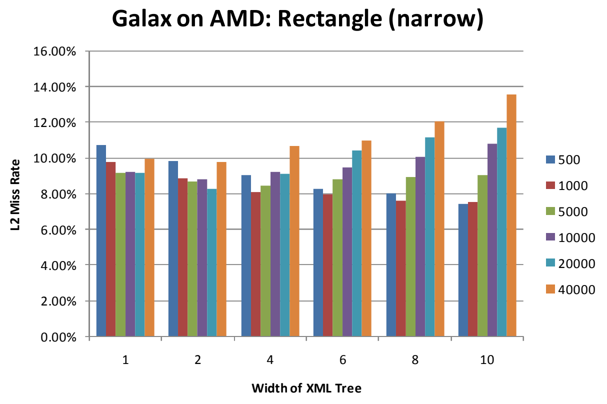

Figure 5: L2 miss rates of executing Q1 on narrow Rectangle XML documents in Galax (varying width and height)

5.1.1.2 Triangle XML

Figure 6 shows that Bdb-xml and Galax have completely converse L2 cache miss behaviors for Triangle XML when the height of the tree increases. Bdb-xml fits Triangle XML better than Galax, as it does for wide Rectangle XML.

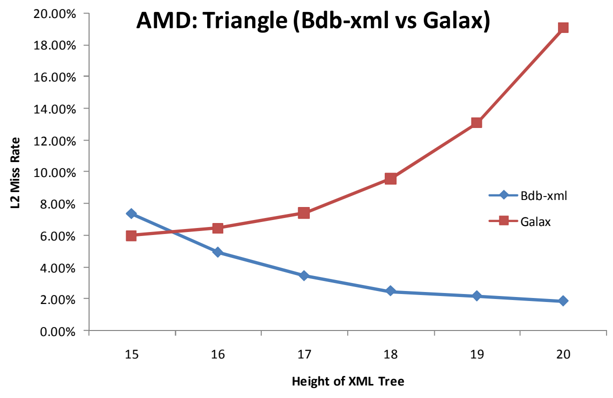

Figure 6: L2 miss rates of executing Q1 on Triangle XML documents in Bdb-xml and in Galax (varying height)

5.1.1.3 List XML

Figure 7 shows L2 cache miss rates when executing Q2 (selection) and Q3 (sorting) on List XML in Bdb-xml and Galax. Although Bdb-xml has a very low L2 cache miss rate (<2%) for selection, its sorting algorithm has to suffer a high L2 cache miss rate (>14%). Galax has converse behaviors for the two operations.

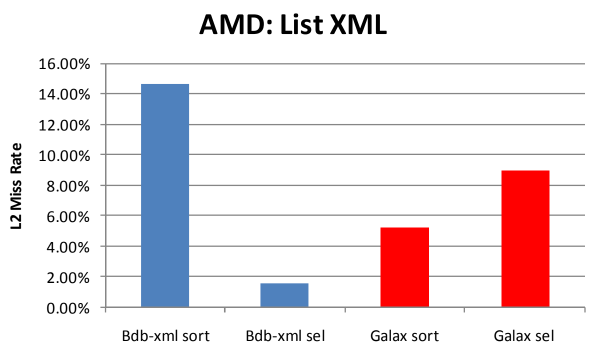

Figure 7: L2 miss rates of executing Q2 (selection) and Q3 (sorting) on List XML document in Bdb-xml and in Galax.

5.1.2 Intel

On this system, we only report results of executing Q1 on Rectangle XML. Figure 8 and Figure 9 show L2 miss rates of Bdb-xml and Galax for executing Q1 on wide shapes. Figure 10 and Figure 11 show them on narrow ones. We summarize them as following:

(1) The four graphs show similar L2 miss rate trends as corresponding AMD versions (in subsection 5.1.1.1).

(2) Except the case of Bdb-xml for narrow Rectangle XML, P4 shows lower L2 miss rates than Sempron, which is the benefit of increased L2 cache size (512KB vs 256KB).

(3) Amazingly, for executing Q1 on narrow Rectangle XML in Bdb-xml, Figure 10 presents a worse L2 miss rate for P4 than Figure 4 for Sempron, despite P4's larger L2 cache. For the maximized document (width=10, height=40000), the L2 miss rate of P4 is up to 87%.

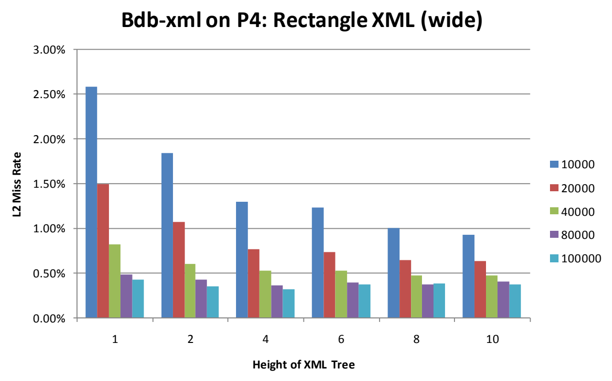

Figure 8: L2 miss rates of executing Q1 on wide Rectangle XML documents in Bdb-xml (varying width and height)

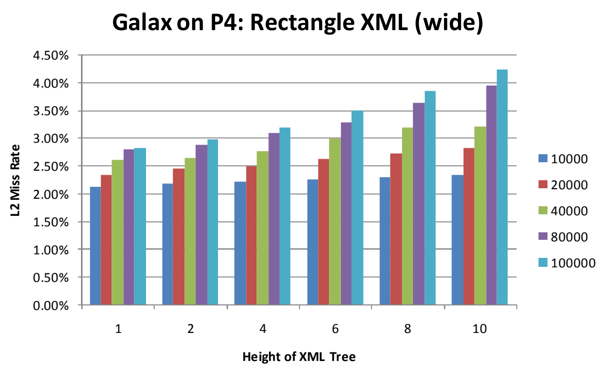

Figure 9: L2 miss rates of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

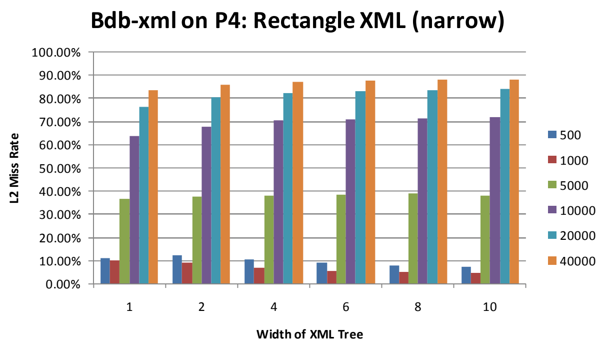

Figure 10: L2 miss rates of executing Q1 on narrow Rectangle XML documents in Bdb-xml (varying width and height)

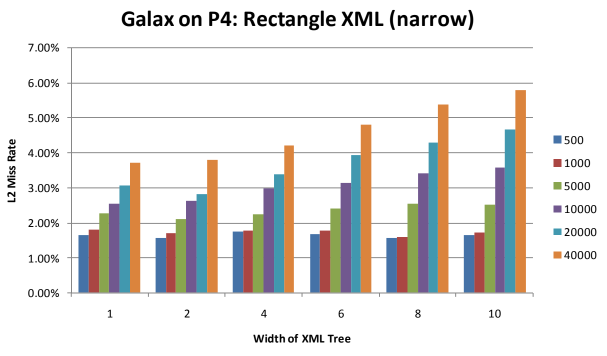

Figure 11: L2 miss rates of executing Q1 on narrow Rectangle XML documents in Galax (varying width and height)

5.2 Branch Mispredictions

5.2.1 AMD

5.2.1.1 Rectangle XML

Figure 12-15 shows branch misprediction rates of executing Q1 on wide Rectangle XML and narrow one in Bdb-xml and in Galax on our AMD system, correspondingly. We summarize the results as follows:

(1) Whether the XML document is wide or narrow, running Galax has similar behaviors of branch mispredictions. (comparing Figure 13 with Figure 15)

(2) However, the XML shape has a significant influence on Bdb-xml. By comparing Figure 12 with Figure 14, we can see that it is better for Bdb-xml to process narrow documents than wide ones.

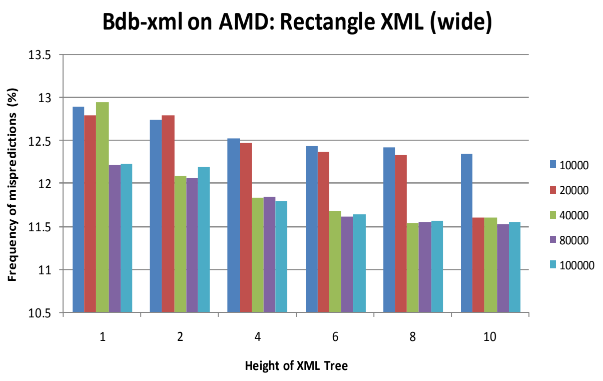

Figure 12: branch misprediction rates of executing Q1 on wide Rectangle XML documents in Bdb-xml (varying width and height)

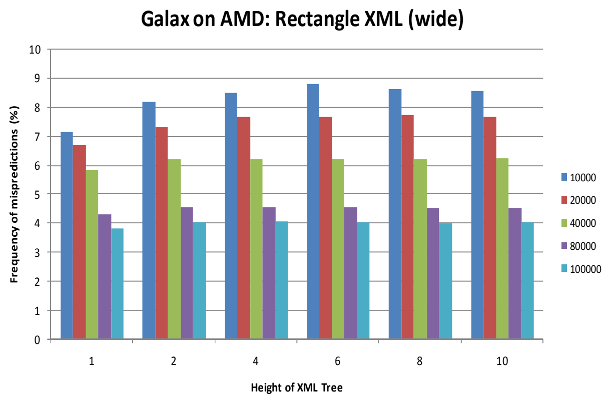

Figure 13: branch misprediction rates of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

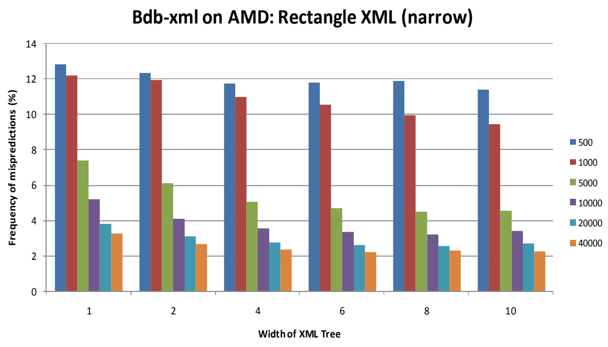

Figure 14: branch misprediction rates of executing Q1 on narrow Rectangle XML documents in Bdb-xml (varying width and height)

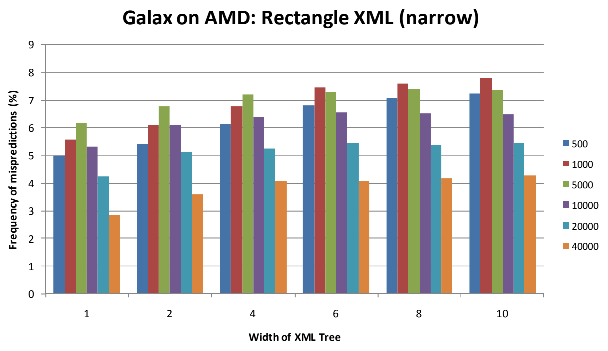

Figure 15: branch misprediction rates of executing Q1 on narrow Rectangle XML documents in Galax (varying width and height)

5.2.1.2 Triangle XML

Figure 16 shows the difference between Bdb-xml and Galax when executing Q1 on Triangle XML. From the graph, we can see, Bdb-xml has higher branch misprediction rates than Galax in this situation.

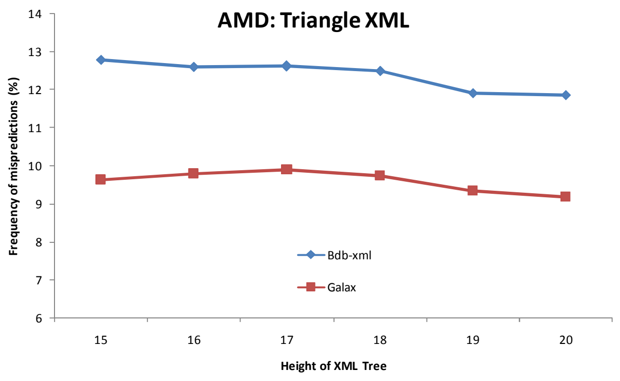

Figure 16: branch misprediction rates of executing Q1 on Triangle XML documents in Bdb-xml and in Galax (varying width and height)

5.2.1.3 List XML

Figure 17 shows the branch information on List XML. Bdb-xml has larger misprediction rates than Galax for both selection and sorting.

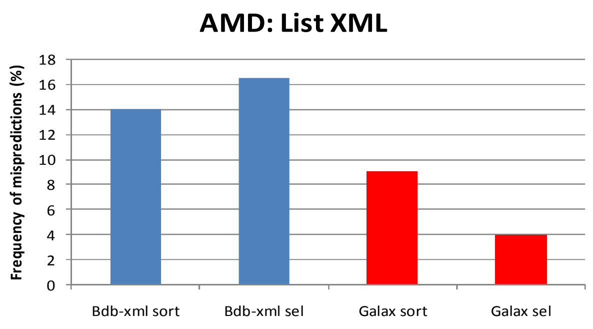

Figure 17: branch misprediction rates of executing Q2 (selection) and Q3 (sorting) on List XML document in Bdb-xml and in Galax.

5.2.2 *Intel*

On this system, we only report results for executing Q1 on Rectangle XML. Figure 18-21 show branch misprediction rates of executing Q1 on wide Rectangle XML and narrow one in Bdb-xml and in Galax on our P4 system, correspondingly. There graphs are similar with corresponding AMD Sempron versions, only with the differences of lower misprediction rates, which show that Pentium P4 has a more effective branch predictor than AMD Sempron.

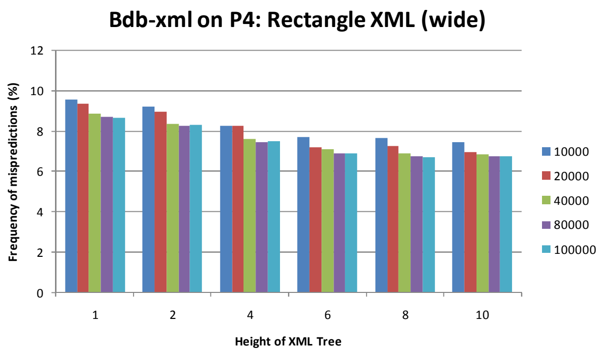

Figure 18: branch misprediction rates of executing Q1 on wide Rectangle XML documents in Bdb-xml (varying width and height)

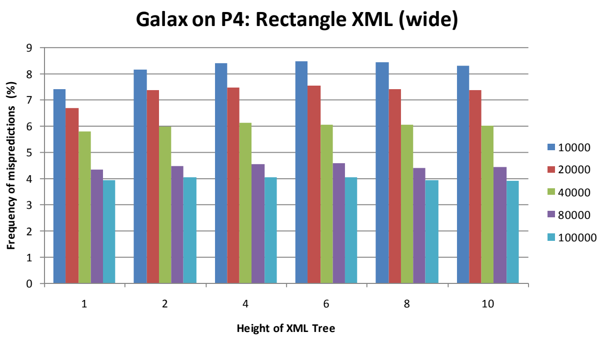

Figure 19: branch misprediction rates of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

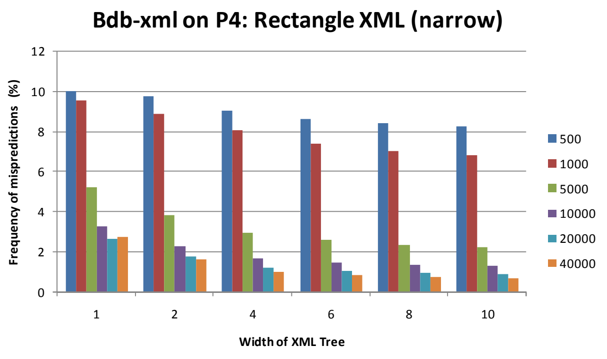

Figure 20: branch misprediction rates of executing Q1 on narrow Rectangle XML documents in Bdb-xml (varying width and height)

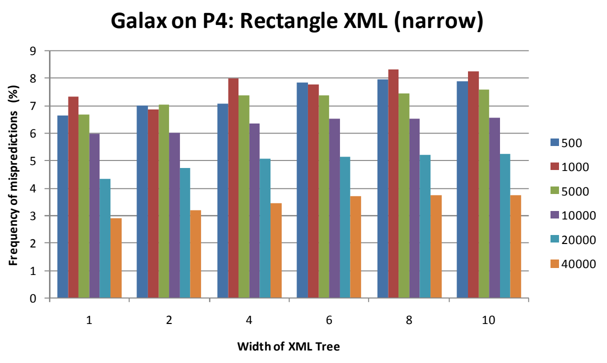

Figure 21: branch misprediction rates of executing Q1 on narrow Rectangle XML documents in Galax (varying width and height)

5.3 L1 Cache & TLB Misses

In this subsection, we present our measured results about L1 cache and TLB behaviors of Bdb-xml and Galax on our AMD system. We examine five fixed queries in turn:

(1) *wide*: Executing Q1 on the maximized wide Rectangle XML document. The width is 100000 and the height is 10.

(2) *narrow*: Executing Q1 on the maximized narrow Rectangle XML document. The width is 10 and the height is 40000.

(3) *triangle*: Executing Q1 on the maximized Triangle XML document. The height is 20.

(4) *selection*: Executing Q2 on the List XML document.

(5) *sorting*: Executing Q3 on the List XML document.

We measure five architectural characteristics available on our AMD system: total instructions, L1 instruction cache misses per 1000 instructions, L1 data cache misses per 1000 instructions, L1 instruction TLB misses per 1000 instructions, and L1 data TLB misses per 1000 instructions.

Figure 22 shows the count of total instructions for executing the five queries in Bdb-xml and in Galax. We can see that, except the query "*narrow*", Bdb-xml needs fewer instructions than Galax, especially for the query "*wide*" and "*sorting*".

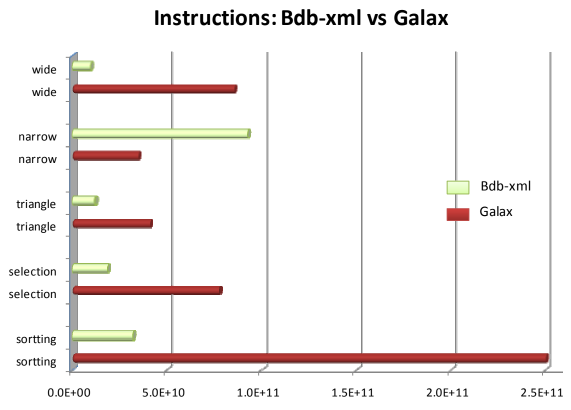

Figure 22: count of instructions of executing corresponding query in Bdb-xml and in Galax on Sempron

Figure 23 shows the difference between Bdb-xml and Galax for the query "*wide*". Except L1 data cache misses, Galax has a better behavior than Bdb-xml.

Figure 24 shows the astonishing difference between Bdb-xml and Galax when executing the query "*narrow*". The graph reveals two points. First, for both tools, L1 instruction cache misses and L1 instruction TLB misses are very few. Second, Bdb-xml has a very bad L1 data cache behavior in this situation. This result is consistent with the above L2 cache miss rates for Bdb-xml on both AMD and Intel systems, which reflects the bad memory-access pattern of Bdb-xml for narrow Rectangle XML documents.

Figure 25 shows the compared results for query "*triangle*", which is similar with the query "*wide*".

Figure 26 shows the difference between Bdb-xml and Galax when executing "*selection"* on List XML. From the graph, we can see that Bdb-xml has a bad L1 instruction cache behavior (more than 25 misses per 1000 instructions).

Figure 27 shows the compared results for query "*sorting*". For each aspect shown in the graph, Bdb-xml is worse than Galax. However, as shown in Figure 22, Bdb-xml needs much fewer instructions to finish sorting than Galax (3.2E+10 vs 2.5E+11).

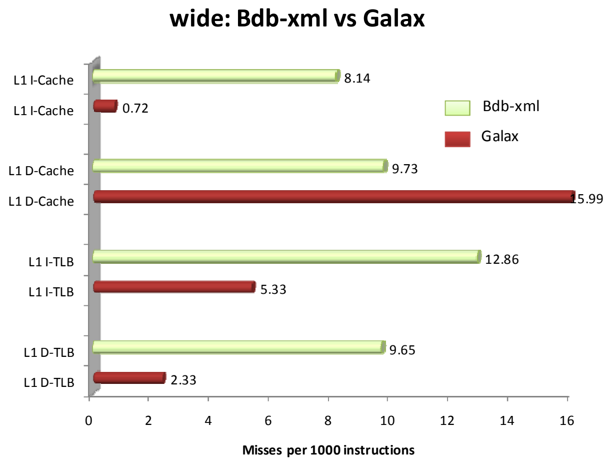

Figure 23: L1 behaviors of executing "wide" in Bdb-xml and in Galax on Sempron

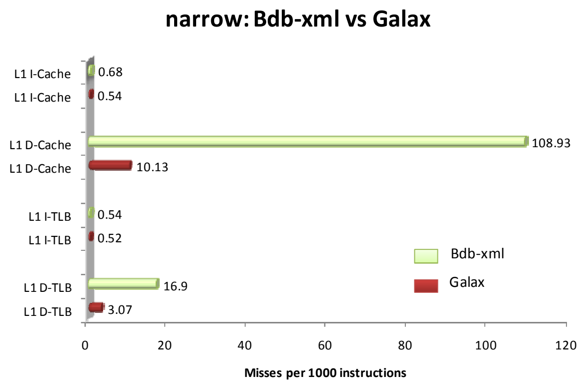

Figure 24: L1 behaviors of executing "narrow" in Bdb-xml and in Galax on Sempron

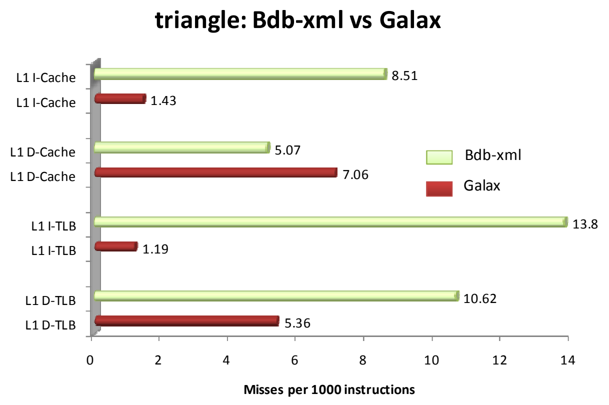

Figure 25: L1 behaviors of executing "triangle" in Bdb-xml and in Galax on Sempron

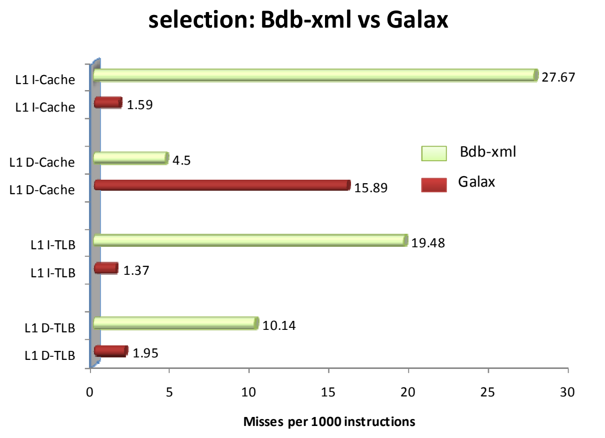

Figure 26: L1 behaviors of executing "selection" in Bdb-xml and in Galax on Sempron

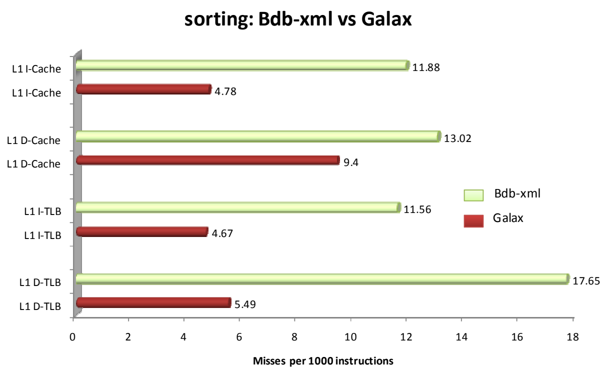

Figure 27: L1 behaviors of executing "sorting" in Bdb-xml and in Galax on Sempron

We summarize this group of measures into three points. First, from the viewpoint of architecture, Bdb-xml is worse than Galax, especially in the respect of being aware of instruction locality. Bdb-xml has more misses than Galax in L1 I-Cache and L1 I-TLB. Second, from the viewpoint of algorithms, however, Bdb-xml is more effective than Galax (except the case for narrow

Rectangle XML). Bdb-xml needs fewer instructions than Galax for finishing corresponding queries. Last, Bdb-xml cannot fit deep XML documents (with large heights) as shown in Figure 4/10/24—it suffers significant data cache misses (at both L1 and L2).

5.4 Results on UltraSPARC T1

In this subsection, we present measured results on UltraSPARC T1 for Saxon, Qexo, and Galax, in turn. The data are only for executing Q1 on wide Rectangle XML documents. Limited by the tool we use on the system, we report the following characteristics:

(1) Count of total instructions
(2) L1 instruction cache misses per 1000 instructions
(3) L1 data cache misses per 1000 instructions
(4) L2 instruction cache misses per 1000 instructions (unavailable for Galax)
(5) L2 data cache misses per 1000 instructions

5.4.1 Saxon

Figure 28 shows count of instructions needed by Saxon for wide Rectangle XML documents with varied dimensions.

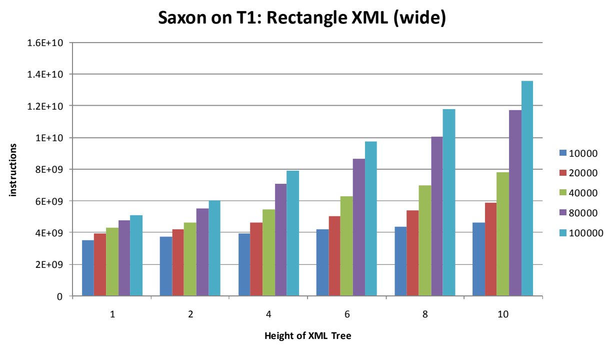

Figure 28: count of instructions of executing Q1 on wide Rectangle XML documents in Saxon (varying width and height)

Figure 29 and Figure 30 show the L1 instruction cache misses and L1 data cache misses of Saxon. By comparing the two graphs, we can see that the width has a more dramatic influence on the misses of instruction cache than on the misses of data cache.

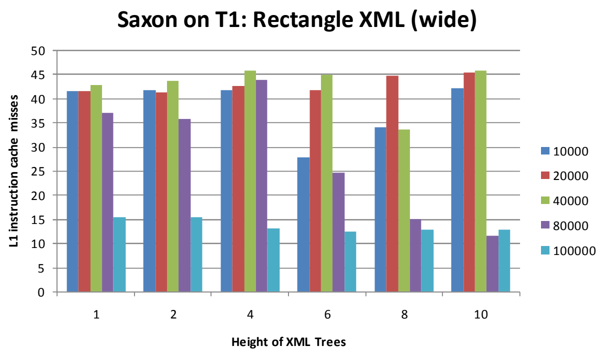

Figure 29: L1 instruction cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Saxon (varying width and height)

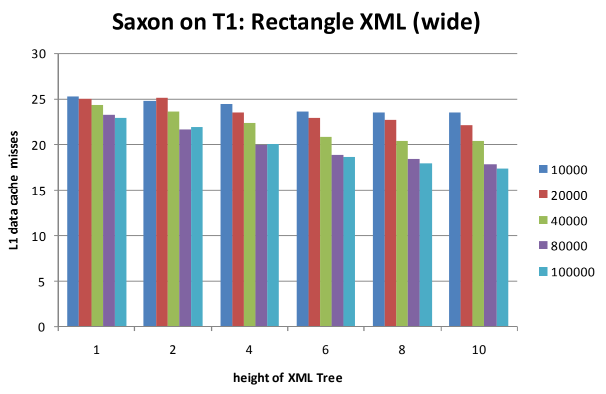

Figure 30: L1 data cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Saxon (varying width and height)

Figure 31 and Figure 32 show the L2 cache behavior of Saxon. They reflect expectable results since UltraSPARC T1 provides a 3MB L2 cache so that misses on this level cache are few.

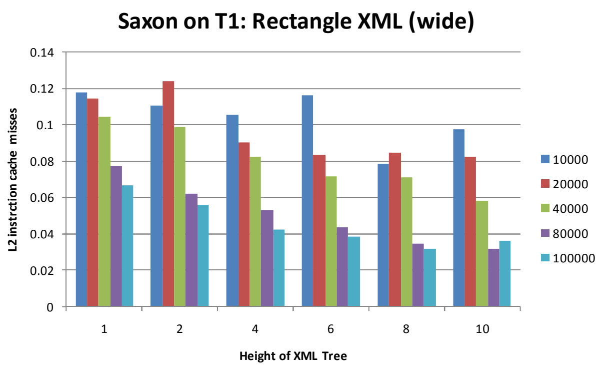

Figure 31: L2 instruction cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Saxon (varying width and height)

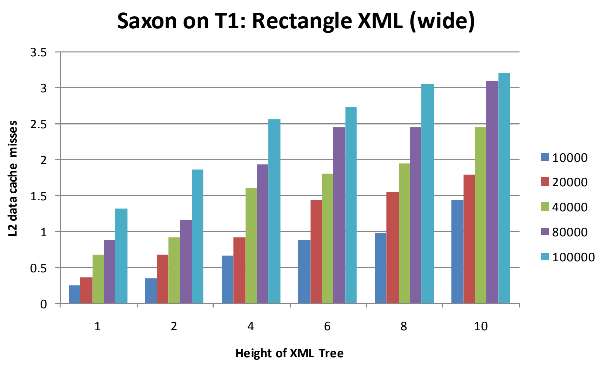

Figure 32: L2 data cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Saxon (varying width and height)

5.4.2 Qexo

Figure 33 shows the count of instructions of Qexo.

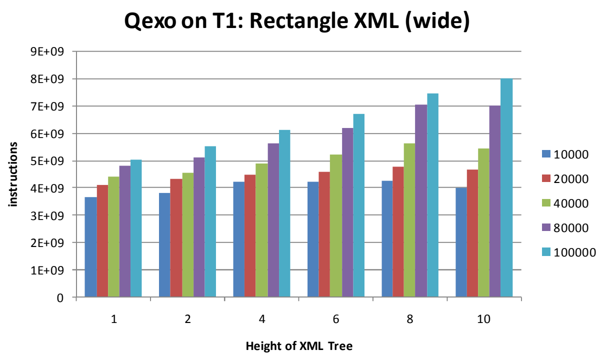

Figure 33: count of instructions of executing Q1 on wide Rectangle XML documents in Qexo (varying width and height)

Figure 34 and Figure 35 show the L1 cache behaviors of Qexo. Comparing Figure 34 and the above Figure 29 for Saxon, we can see that Qexo has fewer L1 instruction cache misses than Saxon, and Qexo is less sensitive to the change of the width than Saxon. As shown in Figure 35 and Figure 30, Saxon and Qexo have similar L1 data cache behaviors, although Saxon suffers more misses than Qexo, correspondingly.

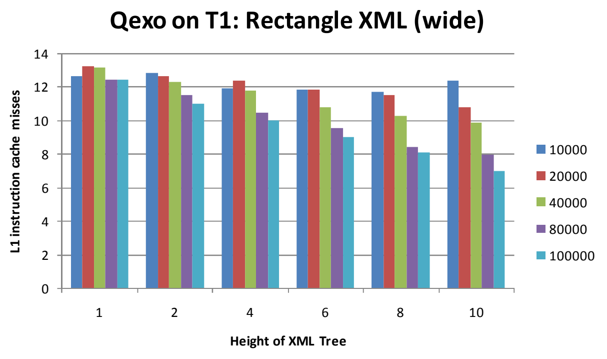

Figure 34: L1 instruction cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Qexo (varying width and height)

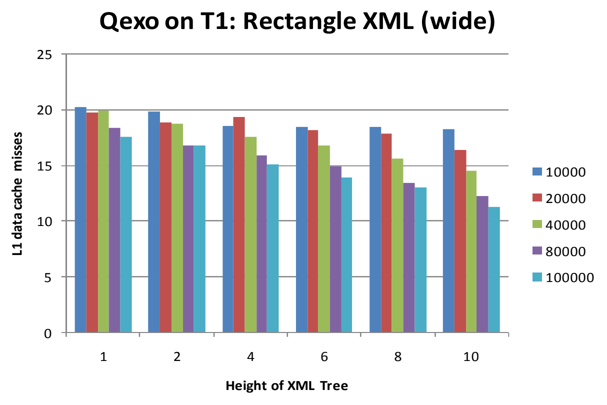

Figure 35: L1 data cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Qexo (varying width and height)

Figure 36 and Figure 37 shows the L2 cache behaviors of Qexo, for instruction cache and data cache respectively. As shown in Figure 31 and Figure 32, misses in this level cache are rare.

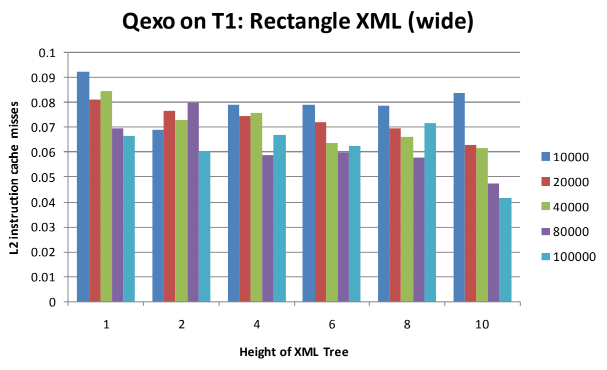

Figure 36: L2 instruction cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Qexo (varying width and height)

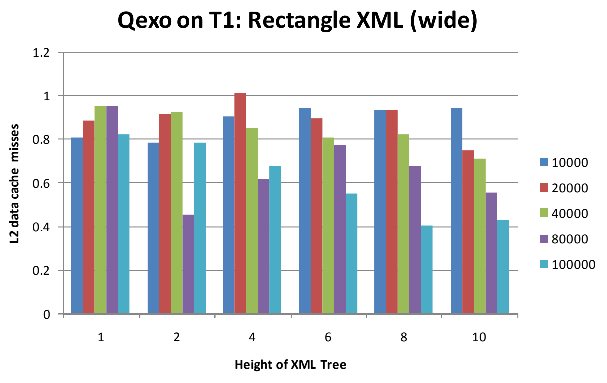

Figure 37: L2 data cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Qexo (varying width and height)

5.4.3 Galax

Figure 38 shows the count of instructions of Galax. Compared with Saxon and Qexo, we find that Galax needs the most instructions to finish the query executions. In addition, the change of width has a more significant influence on Galax than on Saxon and Qexo.

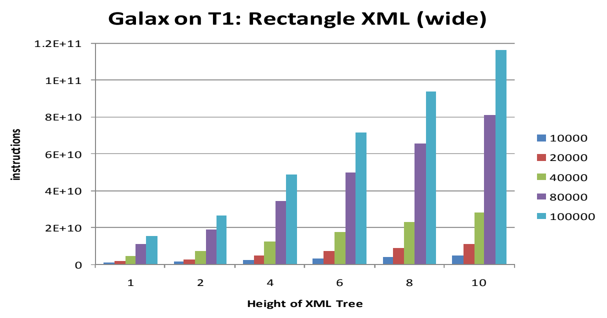

Figure 38: count of instructions of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

Figure 39/40 show L1 cache behaviors of Galax. Galax has fewer instruction cache misses but more data cache misses than Saxon and Qexo.

We compare Galax on T1 with it on Sempron for only the case of maximized wide Rectangle XML documents in Table 4. The results show the benefit of larger L1 cache size (both instruction and data) in Sempron than T1.

Table 4: Galax: UltraSPARC T1 vs Sempron (wide Rectangle XML: width=100000, height=10)

Metrics	T1	Sempron
Count of instructions	1.16E+11	8.6E+10
L1 IC misses per 1k instructions	3.47	0.72
L1 DC misses per 1k instructions	36.5	15.99

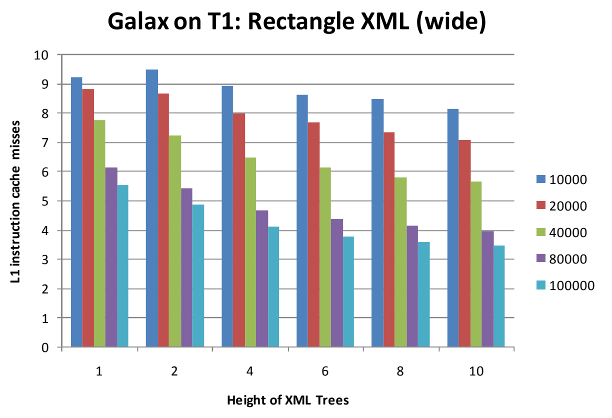

Figure 39: L1 instruction cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

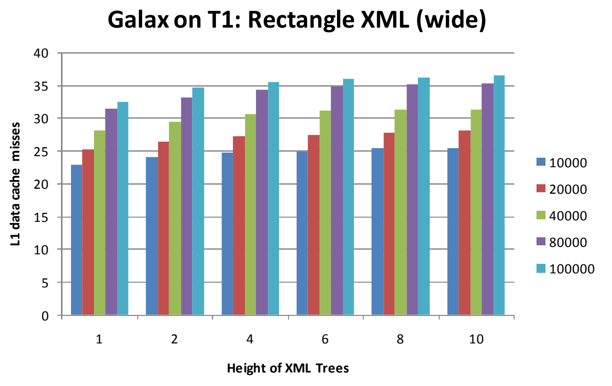

Figure 40: L1 data cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

Figure 41 shows L2 data cache behaviors of Galax. There are distinct differences when the width increases from 40000 to 80000.

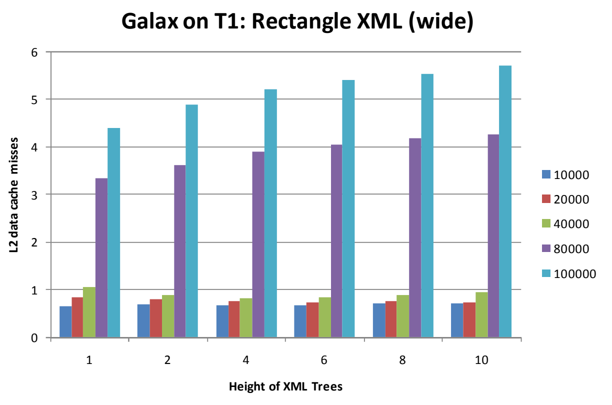

Figure 41: L2 data cache misses per 1000 instructions of executing Q1 on wide Rectangle XML documents in Galax (varying width and height)

6. CONCLUSION

This paper is our first step to understand the architectural behaviors of XQuery workloads on modern processors. In this paper, we are focusing on report detailed measures of architectural characteristics for executing basic XQuery operations on three modern hardware platforms including AMD's Sempron, Intel's Pentium P4, and Sun's UltraSPARC T1. We examine four XQuery tools: Berkeley DB XML, Galax, Saxon-B, and GNU Qexo. Our measured architectural behaviors include L1 cache misses, L2 cache misses, TLB misses, and branch misprediction rates. We believe that these data can be useful in understanding the specific features of XQuery, comparing XQuery workloads and RDBMS query workloads, and analyzing potential optimizing opportunities for XQuery implementations.

Our future work covers the following aspects:

(1) Testing more operations, more complex xml structures, and more software: Currently we only consider path navigation, selection, and sorting of XQuery, and we only study Rectangle XML, Triangle XML, and List XML. We will further consider existing XQuery application-benchmarks [16][17][29] and micro-benchmarks [19][28], and make more wide measures.

(2) Testing concurrent XQuery workloads with write operations: We are planning to utilize concurrent XQuery workloads to study the architectural behaviors on platforms with the new chip-multiprocessor and simultaneous multithreading technology. Although XQuery is a query-oriented language, many tools (e.g. Berkeley DB XML) support modification and transaction processing. We hope to understand the behaviors of shared L2 cache by multiple cores when executing concurrent XQuery instances.

7. ACKNOWLEDGMENTS

We thank Professor Kai Li and Dr. Zhiwei Xu for their valuable suggestions, and thank the ExpDB workshop reviewers for their comments. This work is supported in part by the National Science Foundation of China (Grant No. 90412010), China Ministry of Science and Technology 863 Program (Grant No. 2006AA01A106), and the China National 973 Program (Grant No. 2005CB321807).

8. REFERENCES

[1] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a modern processor: Where does time go? In *Proc.* VLDB, 1999.

[2] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory System Characterization of Commercial Workloads. In *Proc.* ISCA, 1998.

[3] R. J. Eickemeyer, R. E. Johnson, S. R. Kunkel, M. S. Squillante, and S. Liu. Evaluation of multithreaded uniprocessors for commercial application environments. In *Proc.* ISCA, 1996.

[4] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance characterization of a quad Pentium pro SMP using OLTP workloads. In *Proc.*ISCA, 1998.

[5] J. L. Lo, L. A. Barroso, S. J. Eggers, K. Gharachorloo, H. M. Levy, and S. S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Proc.* ISCA, 1998

[6] P. Trancoso, J.L. Larriba-Pey, Z. Zhang, and J. Torellas. The memory performance of DSS commercial workloads in shared-memory multiprocessors. In Proc. HPCA, 1997

[7] M. Karlsson, K. E. Moore, E. Hagersten, and D. A. Wood. Memory System Behavior of Java-Based Middleware. In *Proc.*HPCA, 2003.

[8] Y. Luo and L. K. John. Workload Characterization of Multithreaded Java Servers. In IEEE International Symposium on Performance Analysis of Systems and Software, 2001.

[9] Y. Shuf, M. J. Serrano, M. Gupta, and J. P. Singh. Characterizing the Memory Behavior of Java Workloads: A Structured View and Opportunities for Optimizations. In *Proc.* SIGMETRICS, 2001

[10] P. Apparao, R. Iyer, R. Morin, N. Nayak, and M. Bhat. Architectural Characterization of an XML-centric Commercial Server Workload. In *Proc.*ICPP, 2004

[11] IA-32 Intel(R) Architecture Optimization Reference Manual, http://developer.intel.com/design/pentium4/manuals/

[12] PMC based Performance Measurement in FreeBSD, http://people.freebsd.org/~jkoshy/projects/perf-measurement/

[13] Basic Performance Measurements for AMD Athlon™ 64 and AMD Opteron™ Processors, http://developer.amd.com/articles.jsp?id=90&num=1

[14] AMD Sempron Processor Family, http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_11599,00.html

[15] OpenSPARC T1 Documents, http://opensparct1.sunsource.net/

[16] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, M. J. Carey, I. Manolescu, and R. Busse. Why and How to Benchmark XML Databases. SIGMOD Record, 3(30):27-32, 2001.

[17] B. Yao, T. Ozsu, and N. Khandelwal. XBench benchmark and performance testing of XML DBMSs. In *Proc.*ICDE,2004.

[18] S. Manegold. An Empirical Evaluation of XQuery Processors. In ExpDB, 2006

[19] I. Manolescu, C. Miachon, and P. Michiels. Towards micro-benchmarking XQuery. In ExpDB, 2006.

[20] The Saxon project web site. http://saxon.sourceforge.net

[21] The Galax web site. http://www.galaxquery.org

[22] The Qexo web site. http://www.gnu.org/software/qexo/

[23] The Berkeley DB XML web site. http://www.oracle.com/database/berkeley-db/xml/index.html

[24] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki. QPipe: A Simultaneously Pipelined Relational Query Engine. In *Proc.* SIGMOD, 2005.

[25] S. Harizopoulos and A. Ailamaki. Improving instructions cache performance in OLTP. In TODS, 31(3): 887-920, 2006

[26] S. Chen, A. Ailamiaki, P. B. Gibbons, and T. C. Mowry. Improving Hash Join Performance through Prefetching. In *Proc.* ICDE, 2004.

[27] Sun Studio Performance Analyzer. http://developers.sun.com/sunstudio/analyzer_index.html

[28] K. Runapongsa, J. M. Patel, H. V. Jagadish, and S. AI-Kalifa. The Michigan Benchmark: A Microbenchmark for XML Querying Systems. In EEXTT, 2002.

[29] S. Bressnan, G. Dobbie, Z. Lacroix, M. Lee, Y. Li, U. Nambiar, and B. Wadhwa. X007: Applying 007 benchmark for XML Querying Tool. In *Proc.*CIKM, 2001.