Energy Efficiency, Performance and Scalability of XML Parsing Using SIMD

ABSTRACT

XML is a data format designed for documents as well as the representation of data structures. The simplicity and generality of the rules make it widely used in web services and database systems. Traditional XML parsers have been built around the byte-at-a-time model, in which they process every character token in the file in a sequential fashion. Unfortunately, the byte-at-time sequential model is a fundamental hindrance on performance and in some cases can add up 100% overhead to the database queries themselves.

In this paper, we propose a new XML parser, Parabix, based on parallel bit stream technology, which converts the character strings into bitstreams and then exploits SIMD operations prevalent on modern CPUs. The first generation parser that we developed, Parabix1, uses the bitscan and bitlevel sequencing SIMD operations to emulate much of the parsers functions. Unfortunately operations like bitscan are inherently sequential nature and Parabix1's speedup is limited. We present a second generation parser, Parabix2, that fully parallelizes the parsing operations using using parallel bitlevel logic provided in modern SIMD extensions like SSE2. We evaluate Parabix1 and Parabix2 against two widely-used XML parsers, James Clark's Expat and Apache's Xerces on three generations of x86 machines, including the new Intel Sandy Bridge. We show that Parabix2's speedup is 2×—8× over Expat and Xerces. Across the different Intel machine generations, Parabix rides the scalability curve of SIMD operations whose performance inherently scales better than traditional sequential thread performance. Comparing Intel's new Sandy Bridge core with the Core i3 we observed performance improvement between 20—60% for our Parabix parsers while sequential parsers like Xerces improve by <20%. We measure real CPU power to demonstrate that Parabix also brings with itself significant energy efficiency. On the core i3, Parabix consumes ≃4nJ per byte parsed while Xerces consumes ≃20nJ per byte parsed. Finally, we perform a case study of the Intel's new 256-bit wide AVX instructions, and demonstrate that it provides X speedup over 128 bit SSE2 instruction set.

1. Introduction

Extensible Markup Language (XML) is a core technology standard of the World-Wide Web Consortium (W3C) that provides a common framework for encoding and communicating structured information of all kinds. In applications ranging from Office Open XML in Microsoft Office to NDFD XML of the NOAA National Weather Service, from KML in Google Earth to Castor XML in the Martian Rovers, from ebXML for e-commerce data interchange to RSS for news feeds from web sites everywhere, XML plays a ubiquitous role in providing a common framework for data interoperability world-wide and beyond. As XML 1.0 editor Tim Bray is quoted in the W3C celebration of XML at 10 years, "there is essentially no computer in the world, desk-top, hand-held, or back-room, that doesn't process XML sometimes."

With all this XML processing, a substantial literature has arisen addressing XML processing performance in general and the performance of XML parsers in particular. Nicola and John specifically identified XML parsing as a threat to database performance and outlined a number of potential directions for potential performance improvements [17]. The nature XML APIs was found to have a significant affect on performance with event-based SAX (Simple API for XML) parsers avoiding the tree construction costs of the more flexible DOM (Document Object Model) parsers [18]. The commercial importance of XML parsing spurred developments of hardware-based approaches including the development of a custom XML chip [15] as well as FPGA-based implementations [13]. As promising as these approaches may be for particular niche applications, however, it is still likely that the bulk of the world's XML processing workload will be carried out on commodity processors using software-based solutions.

To accelerate XML parsing performance in software, most recent work has focused on parallelization. The use of multicore parallelism for chip multiprocessors has attracted the attention of several groups [16, 20, 21], while SIMD (single-instruction multiple data) parallelism has been of interest to Intel in designing new SIMD instructions [14] as well as to our group in developing parallel bit stream technology [7, 9, 10]. Each of these approaches has shown considerable performance benefits over traditional sequential parsing following the byte-at-a-time model.

With a focus on performance, however, relatively less attention has been paid to reducing energy consumption. For example, in addressing performance through multicore parallelism, one generally has to pay an energy price for performance gains because of the increased processing required for synchronization. A focus on reduction of energy consumption is a key topic in this paper, in which we study the energy and performance characteristics of several XML parsers across three generations of x86-64 processor technology. A compelling result is that the performance benefits of parallel bit stream technology translate directly and proportionately to substantial energy savings.

The remainder of this paper is organized as follows. Section 2 presents background material on XML parsing and traditional parsing methods. Section 3 then reviews parallel bit stream technology as applied to XML parsing in our Parabix1 and Parabix2 parsers. Section 4 then introduces our methodology and approach for the performance and energy study tackled in the remainder of the paper. Section 5 presents a detailed performance evaluation on a Core i3 processor as our primary evaluation platform, addressing a number of microarchitectural issues including cache misses, branch mispredictions, SIMD instruction counts and so on. Section 6 then looks at scalability and performance gains through three generations of Intel architecture culminating with performance assessment on our one week-old Sandy Bridge test machine. Section 7 looks specifically at issues in applying the new 256-bit AVX technology to parallel bit stream technology and notes that the major performance benefit seen so far is a result of the change to 3-operand instruction form. Section 8 concludes the paper with a discussion of ongoing work and further research directions.

2. Background

This section provides a brief overview of XML and traditional and parallel XML processing technology, and describes the key design and performance aspects of successive generations of the Parabix parallel XML processing technology.

In 1998, W3C officially adopted XML as the standard platform-

independent data interchange format. The defining characteristics of XML was that it could represent virtually any type of information through the use of self-describing markup tags and could easily store semi-structured data in a descriptive fashion. XML markup encodes a description of an XML document's storage layout and logical structure. Because XML is intended to be ubiqutious, XML markup tags are often verbose by design [6]. For example, a typical XML file could be:

```
<?xml version="1.0"?>
<Products>
 <Product ID="0001">
  <ProductName Language="English">Widget</ProductName>
  <ProductName Language="French">Bitoniau</ProductName>
  <Company>ABC</Company>
  <Price>$19.95</Price>
 </Product>
 ...
</Products>
```

Figure 1: Simple XML Document

XML files tend to be classified as either "documents" or "data". XML-Documents are usually designed to be human readable, such as Figure 1; XML-Data files are intended to be parsed by machines and omit any "human-friendly" formatting techniques, such as the use of whitespace and descriptive "natural language" naming schemes. Although the XML specification does not distinguish between "XML for documents" and "XML for data" [6], the latter often requires the use of an XML parser in order to utilize the information within them. The role of an XML parser is to transform the text-based XML data into an application-ready format.

2.1 Traditional XML Parsers

Traditional XML parsers are sequential byte-at-a-time parsers. Using this approach, an XML parser processes a source document by serially scanning through it in a top-down manner. Each character of text is read to distinguish between the XML-specific markup, such as an opening angle bracket '<', and the data held within the document. As the parser moves through the source document, it alternates between markup scanning and data validation operations. At each processing step, a parser scans the source document and locates the expected markup fields or reports an error and terminates. In other words, traditional XML parsers are complex finite-state machines that use per-character comparisons to transition between data- and metadata-type states. Each state transition indicates the context in which to interpret the subsequent characters. Unfortunetly, textual data tends to consist of variable-length items in generally unpredictable patterns [9]; thus any character could be a state transition until deemed otherwise. Two such parsers are Expat and Xerces-C. Both are C/C++ based open-source XML parsers. Expat was originally released in 1998; it is currently used in Mozilla Firefox and Open Office [12]. Xerces-C was released in 1999 and is the foundation of the Apache XML project [2].

The major disadvantage with sequential XML parsers is that every character requires at least one conditional branch. Branch mispredictions have been shown to degrade performance in proportion to the markup density of the source document [10] (i.e., the proportion of XML-markup vs. XML-data).

2.2 Parallel XML Parsing

Parallel XML processing generally comes in one of two forms: multithreading and SIMD. Multithreaded XML parsers take advantage of parallism by first quickly preparsing the XML file to locate the key markup entities and determine the best workload distribution in which process the XML file using n-cores [21]. SIMD XML parsers leverage the SIMD registers to overcome the performance limitations of the sequential paradigm and inherently data dependent branch misprediction rates [9]. Two such SIMD XML parsers, Parabix1 and Parabix2, utilizes parallel bit stream processing technology. With this method, byte-oriented character data is first transposed to eight parallel bit streams, one for each bit position within the character code units (bytes). These bit streams are then loaded into SIMD registers of width W (e.g., 64-bit, 128-bit, 256-bit, etc). This allows W consecutive code units to be represented and processed at once. Bitwise logic and shift operations, bit scans, population counts and other bit-based operations are then used to carry out the work in parallel [11].

2.2.1 Parabix1

Our first generation parallel bitstream XML parser—Parabix1—uses employs a less conventional approach of SIMD technology to represent text in parallel bitstreams. Bits of each stream are in one-to-one-correspondence with the bytes of a character stream. A transposition step first transforms sequential byte stream data into eight basis bitstreams for the bits of each byte. Bitwise logical combinations of these basis bitstreams can then be used to classify bytes in various ways, while the bit scan operations common to commodity processors can be used for fast sequential scanning. At a high level, Parabix1 processes source XML in a functionally equivalent manner as a traditional processor. That is, Parabix1 moves sequentially through the source document, maintaining a single cursor scanning position throughout the parse. However, this scanning operation itself is accelerated significantly which leads to dramatic performance improvements, since bit scan operations can perform up to general register width (32-bit, 64-bit) finite state transitions per clock cycle. This approach has recently been applied to Unicode transcoding and XML parsing to good effect, with research prototypes showing substantial speed-ups over even the best of byte-at-a-time alternatives [9–11].

2.2.2 Parabix2

In our second generation XML parser—Parabix2—we address the replacement of sequential parsing using bit scan instructions with a parallel parsing method using bitstream addition. Unlike the single cursor approach of Parabix1 and conceptually of traditional sequential approach, in Parabix2 multiple cursors positions are processed in parallel. To deal with these parallel cursors, three additional categories of bitstreams are introduced. Marker bitstreams are used to represent positions of interest in the parsing of a source data stream [9]. The appearance of a 1 at a position in a marker bitstream could, for example, denote the starting position an XML tag in the data stream. In general, the set of bit positions in a marker bitstream may be considered to be the current parsing positions of multiple parses taking place in parallel throughout the source data stream. A further aspect of the parallel method is that conditional branch statements used to identify syntax error at each each parsing position are eliminated. Instead, error bitstreams are used to identify the position of parsing or well-formedness errors during the parsing process. Error positions are gathered and processed in as a final post processing step. Hence, Parabix2 offers additional parallelism over Parabix1 in the form of multiple cursor parsing as well as significanlty reduces branch misprediction penalty.

3. Parabix

Describe key technology behind Parabix Introduce SIMD; Talk about SSE Highlight which SSE instructions are important TAlk about each pass in the parser; How SSE is used in every phase... Benefits of SSE in each phase.

The results of [10] showed that Parabix, the predecessor of Para-

bix2, was dramatically faster than both Expat 2.0.1 and Xerces-C++ 2.8.0. It is our expectation is that Parabix2 will outperform both Expat 2.0.1 and Xerces-C++ 3.1.1 in terms of energy consumption per source XML byte. This expectation is based on the relatively-branchless code composition of Parabix2 and the more-efficient utilization of last-level cache resources. The authors of [3–5] indicate that such factors have a considerable effect on overall energy consumption. Hence, one of the foci in our study is the manner in which straight line SIMD code influences energy usage.

4. Methodology

In this section, we describe our methodology for the measurements and investigation of XML parsing energy consumption and performance. In brief, for each of the XML parsers under study we propose to measure and evaluate the energy consumption required to carry out XML well-formedness checking, under a variety of workloads, and as executed on three different Intel cores.

To begin our study, we propose to first investigate each of the XML parsers in terms of the PMCs hardware events as listed in the following subsection. Based on previous key works [3–5], we have chosen several key hardware performance events for which the authors indicate have a strong correlation to energy consumption. From these data, we hope to gain insight into the XML parser execution characteristics which most significantly contribute to overall energy consumption. Secondly, using the Fluke i410 current clamp meter, we plan to measure the total energy consumption required to complete XML well-formedness checking for each XML parser, on each hardware platform, and for each of a number of XML source files.

The foundational work by Bellosa in [3] as well as more recent work in [4, 5] show that hardware-usage patterns has a significant impact in the energy consumption of a particular application; [3–5] further show that there is a strong correlation between specific performance events and energy usage—but the authors of each differ slightly in opinion as to which performance monitoring counters[1] (PMCs) to use.

The following subsections describe the XML parsers under study, XML workloads, the hardware architectures, PMC hardware events selected for measurement, and the Fluke i401 current clamp meter. The expected outcomes of this section are hardware performance counter measurements and total energy consumption measurements for each of XML parser, XML source file, and hardware combination.

4.1 Parsers

The XML parsing technologies selected for this study are the Parabix2, Xerces-C++, and Expat XML parsers. Parabix2 [19] (parallel bit streams for XML) is the second generation Parabix parser. Parabix2 is an open-source XML parser that leverages the SIMD capabilities of modern commodity processors; it employs the new parallelization techniques using parallel parsing with bit stream addition to deliver dramatic performance improvements over traditional byte-at-a-time parsing technology. Xerces-C++ version 3.1.1 (SAX) [2] is a validating open source XML parser written in C++ by the Apache project. Expat version 2.0.1 [12] is a non-validating XML parser library written in C.

4.2 Workloads

Distinguishing between "document-oriented" XML and "data-oriented" XML is a popular way to describe the two basic classes of XML documents. Data-oriented XML is used as an interchange format. Document-oriented XML is used to impose structure on information that rarely fits neatly into a relational database–particularly information intended for publishing. Data-oriented XML are characterized by a higher markup density. Markup density is defined as the ratio of the total markup contained within an XML file to the total XML document size. This metric may have substantial influence on the performance of XML parsing. As such we choose workloads with distinguishable markup densities.

Table 1 shows the document characteristics of the XML input files selected for this performance study. The jawiki.xml and dewiki.xml XML files represent document-oriented XML inputs, containing three-byte and four-byte UTF8 sequence. The remaining files are data-oriented inputs and consist of only ASCII characters. [10]

Describe parameters; what each parameter means.

4.3 Platform Hardware

4.3.1 Intel Core 2

Processor	Intel Core 2 Duo processor 6400 (2.13GHz)
L1 Cache	32KB I-Cache, 32KB D-Cache
L2 Cache	2MB
Front Side Bus	1066 MHz
Memory	2GB
Hard disk	SCSI
Max TDP	65W

Table 2: Core 2

4.3.2 Intel Core i3

The Intel Core i3 is a Nehalem based processor produced by Intel. The intent of this processor is to serve as a low end server processor. Table 4 gives the hardware description of the Intel Core i3 based machine selected.

Processor	Intel Clarkdale I3-530 (2.93GHz)
L1 Cache	32KB I-Cache, 32K D-Cache
L2 Cache	256KB
L3 Cache	4-MB
Front Side Bus	1333 MHz
Memory	4GB
Hard disk	SCSI 1TB
Max TDP	73W

Table 3: Core i3

4.3.3 Sandy Bridge

Processor	Intel Core I5-2300 (2.80GHz)
L1 Cache	192 KB
L2 Cache	4 X 256KB
L3 Cache	6-MB
Front Side Bus	
Memory	
Hard disk	
Max TDP	95W

Table 4: Sandy Bridge

[1] Performance monitoring counters (PMCs) are special-purpose registers that are included in most modern microprocessors; they store the running count of specific hardware events, such as retired instructions, cache misses, branch mispredictions, and arithmetic-logic unit operations to name a few. They can be used to capture information about any program at run-time, under any workload, at a very fine granularity.

File Name	dewiki.xml	jawiki.xml	roads.gml	po.xml	soap.xml
File Type	document	document	data	data	data
File Size (kB)	66240	7343	11584	76450	2717
Markup Item Count	406792	74882	280724	4634110	18004
Markup Density	0.07	0.13	0.57	0.76	0.87

Table 1: XML Document Characteristics

4.4 PMC Hardware Events

Each of the hardware events selected relates to the energy consumption due to one or more hardware units. For example, total branch miss predictions corresponds to the use of the branch misprediction unit.

Initial PMC hardware event set:

- Processor Cycles
- Branch Instructions
- Branch Mispredictions
- Integer Instructions
- SIMD Instructions
- Cache Misses

4.5 Measurement Hardware

The Fluke i410 current clamp meter is an electrical tester that combines a voltmeter with a clamp type current meter. Like the multimeter, the clamp meter has transitioned through the analog period and into the digital era. Created primarily as a single purpose test tool for electricians, the Fluke i410 have incorporated more measurement functions and accuracy [1].

5. Evaluation on Corei3

5.1 Cache behavior

Core i3 has a three level cache hierarchy. The miss penalty for each level is about 4 cycles, 11 cycles, and 36 cycles. Figure 2, Figure 3 and Figure 4 show the L1, L2 and L3 data cache misses of all the four parsers. Although XML parsing is not a memory intensive application, the cost of cache miss for Expat and Xerces can be about half cycle per byte while the performance of Parabix is hardly affected by cache misses. Cache miss isn't just a problem for performance but also energy consumption. L1 cache miss cost about 8.3nJ; L2 cache miss cost about 19nJ; L3 cache miss cost about 40nJ. With a 1GB input file, Expat would consume more than 0.6J and Xerces would consume 0.9J on cache miss.

5.2 Branch Mispredictions

Despite years of improvement, branch misprediction is still a significant bottleneck of performance. The penalty of a branch misprediction is generally more than 10 CPU cycles. As shown in Figure 6, the cost of branch mispredictions for Expat can be more than 7 cycles per byte, which is as much as the processing time of Parabix2 on the same workload.

Reducing the branch misprediction rate is difficult for text-based applications due to the variable-length nature of syntactic elements. Therefore, the alternative solution of reducing branches becomes more attractive. However, the traditional byte-at-a-time method of XML parsing usually involves large amount of inevitable branches. As shown in Figure 5, Xerces can have an average of 13 branches for each byte it processed on the high markup density file. Parabix substantially eliminate the branches by using parallel bit streams. Parabix1 still have a few branches for each block of 128 bytes (SSE) due to the sequential scanning. But with the new

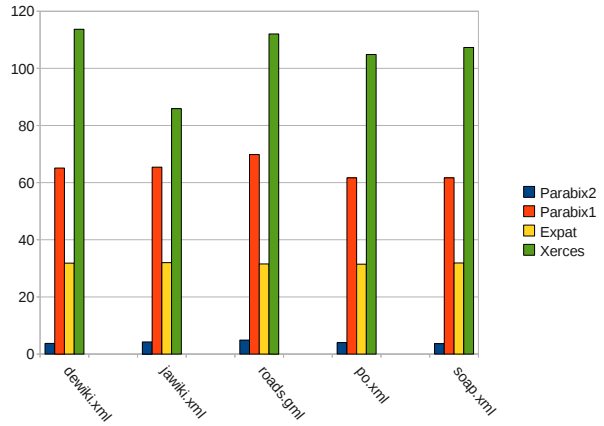

Figure 2: L1 Data Cache Misses/ KB on core i3

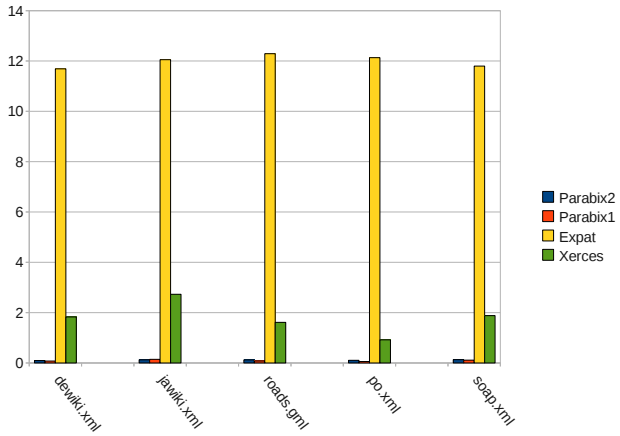

Figure 3: L2 Data Cache Misses/ KB on core i3

parallel scanning technique, Parabix2 is essentially branch-free as shown in the Figure 5. As a result, Parabix2 has much less dependencies on markup density of the workloads.

5.3 SIMD/Total Instructions

Parabix gains its performance by using parallel bitstreams, which are mostly generated and calculated by SIMD instructions. The ratio of executed SIMD instructions over total instructions indicates the amount of parallel processing we were able to achieve. We use Intel pin, a dynamic binary instrumentation tool, to gather instruction mix. Then we adds up all the vector instructions that have been executed. Figure 7 and Figure 8 show the percentage of SIMD instructions of Parabix1 and Parabix2 (Expat and Xerce do not use any SIMD instructions). For Parabix1, 18% to 40% of the executed instructions consists of SIMD instructions. By using

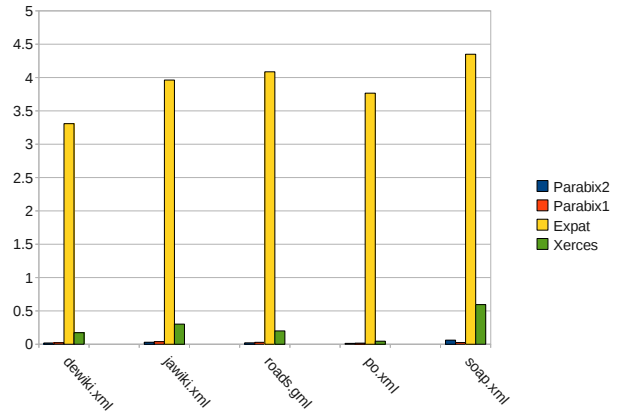

Figure 4: L3 Cache Misses/ KB on core i3

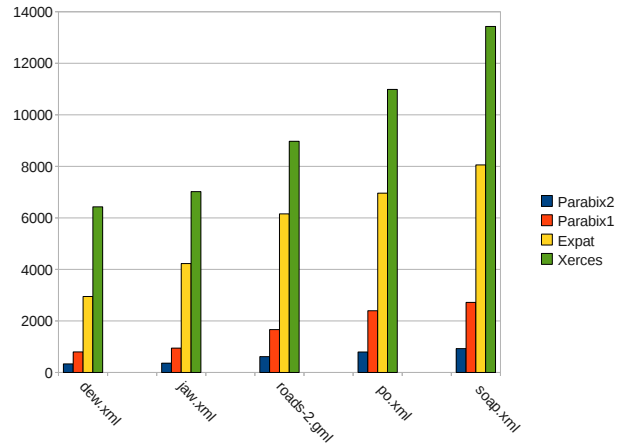

Figure 5: Branches / KB on core i3

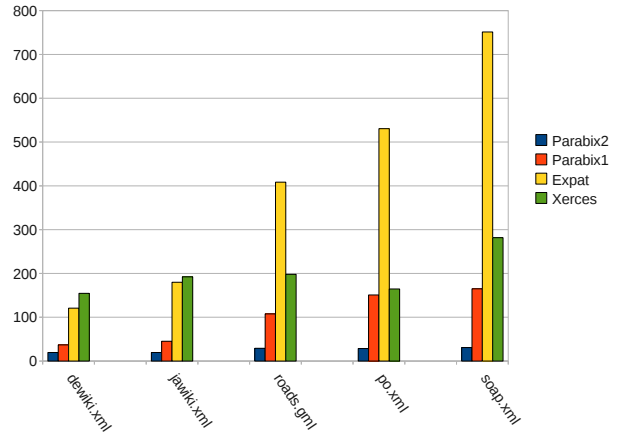

Figure 6: Branch Mispredictions/ KB on core i3

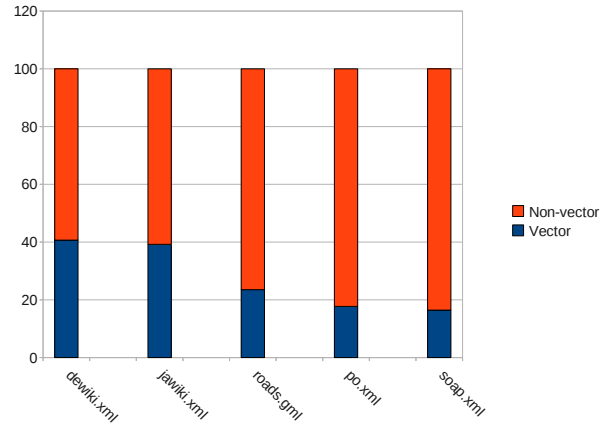

Figure 7: Vector instruction vs. non-vertor instruction for Parabix1 on core i3

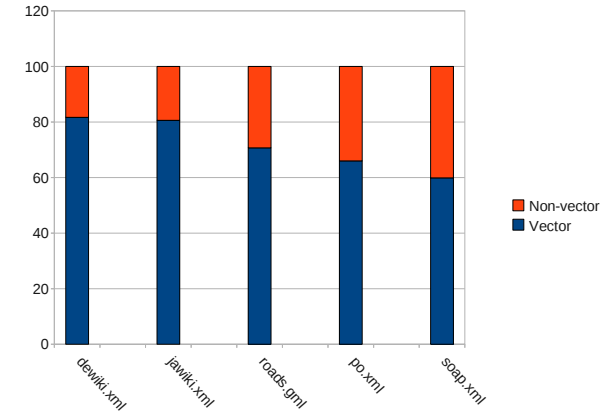

Figure 8: Vector instruction vs. non-vector instruction for Parabix2 on core i3

bistream addition for parallel scanning, Parabix2 uses 60% to 80% SIMD instructions. Although the ratio decrease as the markup density increase for both Parabix1 and Parabix2, the decreasing rate of Parabix2 is much lower and thus the performance degradation caused by increasing markup density is smaller.

5.4 CPU Cycles

Figure 9 shows the result of the overall performance evaluated as CPU cycles per thousands input bytes. Parabix1 is 1.5 to 2.5 times faster on document-oriented input and 2 to 3 times faster on data-oriented input compared with Expat and Xerces. Parabix2 is 2.5 to 4 times faster on document-oriented input and 4.5 to 7 times faster on data-oriented input. Traditional parsers can be dramatically slowed down by higher markup density while Parabix with parallel processing is less affected. The comparison is not entirely fair for Xerces that transcodes input into UTF-16, which typically takes several cycles per byte. However, transcoding using parallel bitstreams can be much faster and it takes less than a cycle per byte to transcode ASCII files such as road.gml, po.xml and soap.xml [8].

5.5 Power and Energy

There is a growing concern of power consumption and energy efficiency. Chip producers not only work on improving the per-

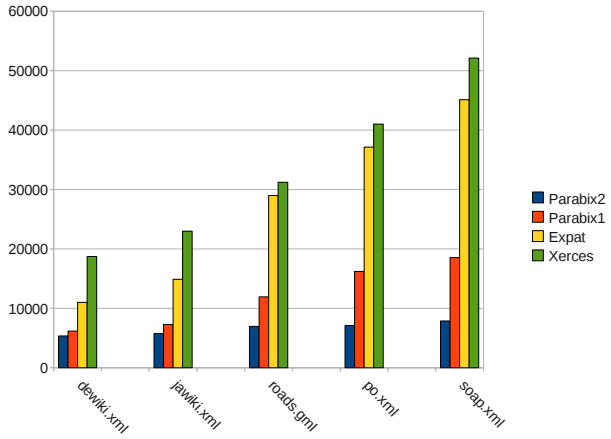

Figure 9: Total CPU Cycles/ KB on core i3

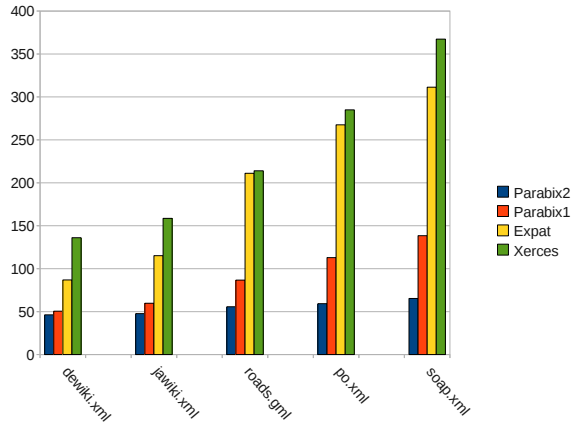

Figure 11: Energy consumption on core i3 (nJ/B)

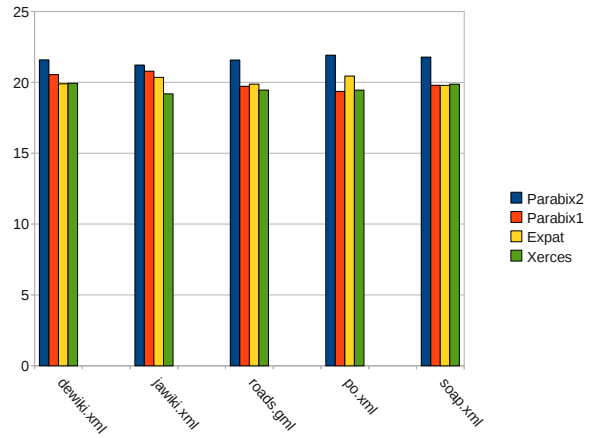

Figure 10: Average Power on core i3 (watts)

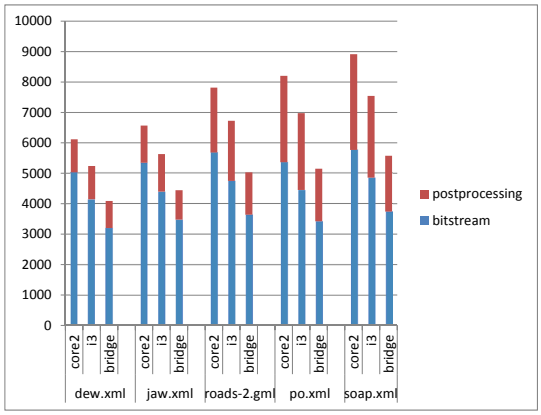

Figure 12: Total CPU cycles /KB of Parabix2

formance but also have worked hard to develop power efficient chips. We studied the power and energy consumption of Parabix in comparison with Expat and Xerces on corei3. We use a clamp to measure the real current of CPU power supply line and a meter to sample and record the results every 10ms.

Figure 10 shows the average power consumed by the four different parsers. The average power of corei3-530 is about 21 watts. This model released by Intel last year has a good reputation for power efficiency. Parabix2 dominated by SIMD instructions uses only about 5% higher power than the other parsers. The power range of SIMD instructions .....

Figure 11 shows the energy consumption of the four different parsers. Although Parabix2 needs slight higer power, its processing time is much shorter and therefore consumes much less energy. Parabix2 consumes 50 to 75 nJ per byte while Expat and Xerces consumes 80nJ to 320nJ and 140nJ to 370nJ per byte seperately.

6. Scalability

6.1 Performance

Figure 12 shows the performance of Parabix2 on three different cores: core2, corei3 and sandybridge. The processing time, which is evaluated as CPU cycles per thousand bytes, is divided up by bitstream parsing and byte space postprocessing. Bitstream parsing, mainly consists of SIMD instructions, is able to achieve 17% performance improvement moving from core2 to corei3; 22% performance improvement moving from corei3 to sandybridge, which is relatively stable compared to postprocessing, which gains 18% to 31% performance moving from core2 to corei3; 0 to 17% performance improvement moving from corei3 to sandybridge.

As comparison, we also measured the performance of Expat on all the three cores. The results are shown in Figure 13. The performance improvement is less than 5% by running Expat on corei3 instead of core2 and it is less than 10% by running on sandybridge instead of corei3.

Parabix2 scales much better than Expat and is able to achieve an overall performance improvement up to 26% simply by running the same code on a newer core. Further improvement on sandybridge with AVX will be discussed in the next section.

6.2 Power and Energy

The newer processors are not only designed to have better performance but also more energy-efficient. Figure 14 shows the average power when running Parabix2 on core2, corei3 and sandybridge with different input files. On core2, the average power is about 32 watts. Corei3 saves 30% of the power compared with core2. Sandybridge saves 25% of the power compared with corei3 and consumes only 15 watts.

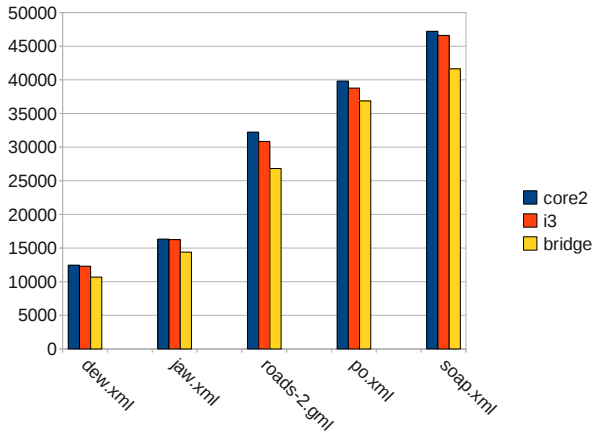

Figure 13: Total CPU cycles /KB of Expat

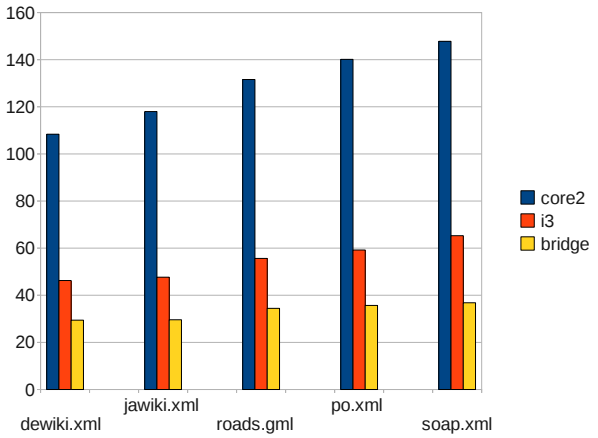

Figure 15: Energy consumption of Parabix2 (nJ/B)

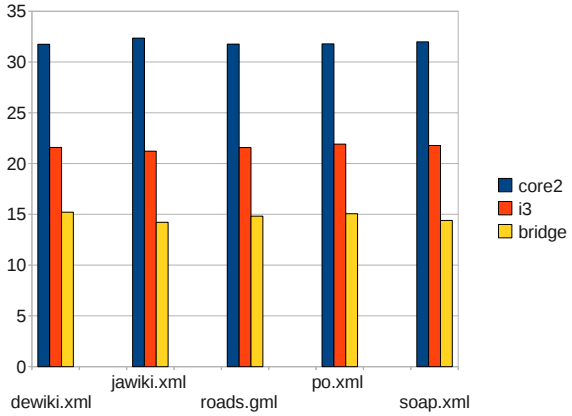

Figure 14: Average Power of Parabix2 (watts)

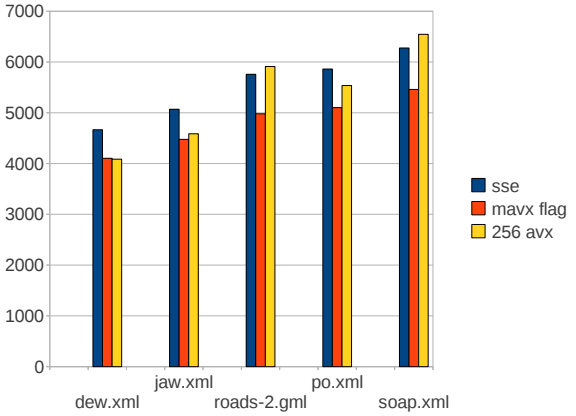

Figure 16: Total CPU cycles /KB on AVX

The energy consumption is further improved by better performance, which means a shorter processing time, as we moved to the newer cores. As a result, Parabix2 on sandybridge cost 72% to 75% less energy than Parabix2 on core2.

7. AVX

7.1 Three Operand Form

7.2 256 bits Operations

8. Conclusion

9. References

[1] Fluke Clamp Meters. http://www.fluke.com/.

[2] Xerces C++ Parser. http://xerces.apache.org/xerces-c/.

[3] F. Bellosa. The case for event-driven energy accounting. Technical Report TR-I4-01-07, University of Erlangen, Department of Computer Sciece, June 2001.

[4] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade. Decomposable and responsive power models for multicore processors using performance counters. In *Proceedings of the 24th ACM International Conference on Supercomputing*, ICS '10, pages 147–158, New York, NY, USA, 2010. ACM.

[5] W. Bircher and L. John. Complete system power estimation: A trickle-down approach based on performance events. In *Performance Analysis of Systems Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 158 –168, Apr. 2007.

[6] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 (fifth edition). W3C Recommendation, 2008.

[7] R. Cameron, K. Herdy, and E. Amiri. Parallel bit stream technology as a foundation for XML parsing performance. In *International Symposium on Processing XML Efficiently: Overcoming Limits on Space, Time, or Bandwidth*, Aug. 2009.

[8] R. D. Cameron. A case study in simd text processing with parallel bit streams: Utf-8 to utf-16 transcoding. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, PPoPP '08, pages 91–98, New York, NY, USA, 2008. ACM.

[9] R. D. Cameron, E. Amiri, K. S. Herdy, D. Lin, T. C. Shermer, and F. P. Popowich. Parallel parsing with bitstream addition: An xml case study. Technical Report TR 2010-11, Simon Fraser University, School of Computing Sciece, October 2010.

[10] R. D. Cameron, K. S. Herdy, and D. Lin. High performance XML parsing using parallel bit stream technology. In *CASCON '08: Proceedings of the 2008 conference of the center for advanced studies on collaborative research*, pages 222–235, New York, NY, USA, 2008. ACM.

[11] R. D. Cameron and D. Lin. Architectural support for swar text processing with parallel bit streams: the inductive doubling principle. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support for programming languages and operating systems*, pages 337–348, New York, NY, USA, 2009. ACM.

[12] J. Clark. The Expat XML Parser. http://expat.sourceforge.net/.

[13] Z. Dai, N. Ni, and J. Zhu. A 1 cycle-per-byte XML parsing accelerator. In *FPGA '10: Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 199–208, New York, NY, USA, 2010. ACM.

[14] Z. Lei. XML parsing accelerator with intelÂő streaming SIMD extensions 4 (intelÂő SSE4). http://software.intel.com/en-us/articles/xml-parsing-accelerator-with-intel-streaming-simd-extensions-4-intel-sse4/, 2008.

[15] M. Leventhal and E. Lemoine. The XML chip at 6 years. In *International Symposium on Processing XML Efficiently: Overcoming Limits on Space, Time, or Bandwidth*, Aug. 2009.

[16] X. Li, H. Wang, T. Liu, and W. Li. Key elements tracing method for parallel xml parsing in multi-core system. *Parallel and Distributed Computing Applications and Technologies, International Conference on*, 0:439–444, 2009.

[17] Nicola, Matthias, and John, Jasmi. XML Parsing: A Threat to Database Performance. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, New Orleans, Louisiana, 2003.

[18] Perkins, E., Kostoulas, M., Heifets, A., Matsa, M., and Mendelsohn, N. Performance Analysis of XML APIs. In *XML 2005*, Atlanta, Georgia, Nov. 2005.

[19] e. a. Robert D. Cameron. Parabix2. http://parabix.costar.sfu.ca/.

[20] B. Shah, P. Rao, B. Moon, and M. Rajagopalan. A data parallel algorithm for xml dom parsing. In Z. BellahsÃĺne, E. Hunt, M. Rys, and R. Unland, editors, *Database and XML Technologies*, volume 5679 of *Lecture Notes in Computer Science*, pages 75–90. Springer Berlin / Heidelberg, 2009.

[21] Y. Zhang, Y. Pan, and K. Chiu. Speculative p-dfas for parallel xml parsing. In *High Performance Computing (HiPC), 2009 International Conference on*, pages 388–397, Dec. 2009.