An overview of XML Parsing using Prefetching algorithm

Ms.Y.S.Alone,Ms.V.M.Deshmukh
Department of Computer Science and Engineering, Sant Gadage Baba Amravati University, India
Email:msvmdeshmukh@rediffmail.com,ysalone29@rediffmail.com

***Abstract*:** *An XML parser is the component that deciphers the XML code .Extensible Markup Language (XML) has become a widely adopted standard for data representation and exchange.However; its features also introduce significant overhead threatening the performance of modern applications. XML parsing is the process of reading an XML document and providing an interface to the user application for accessing the document In this paper we present a study on XML parsing through different classic prefetching algorithms. Without a parser, your code cannot be understood. Computers require instruction. An XML parser provides vital information to the program on how to read the file. Parsers come in multiple formats and styles This paper is an overview of the various issue involved in XML parsing through different prefetching algorithms.*

Keywords: *XML Parsing,SAX,DOM, Prefetching,*

1 INTRODUCTION

XML has become much more than just a data format for information exchange[3]. Enterprises are keeping large amounts of business critical data permanently in XML format. Data centric as well as document and content centric businesses in virtually every industry are embracing XML for their data management and B2B needs[5].Although XML is prevalent with many benefits, due to its verbosity and descriptive nature, XML parsing has introduced heavy performance overhead [1,2]. Generally,XML parsing is both memory and computation intensive.It consumes about 30% of processing time in web service applications [4], and has become a major performance bottleneck in database servers [5]. Extensible Markup Language (XML) is emphasized for its language neutrality, application independency and flexibility, and thus has been adopted as the standard in data exchange and representation.. This is only going to get even worse as XML dataset get larger and more complicated.To improve the performance of XML processing, most existing proposals are dedicated to make acceleration from computation side. However, in this paper, we demonstrate that memory access acceleration is equally (if not more) important compared to computation acceleration. Therefore, different from previous computation acceleration studies, we propose to accelerate XML parsing from the memory-side with the incorporation of data prefetching techniques[15]. memory-side acceleration is generic and can be applied irrelevant of the parsing model underneath. In addition, its combination with computation-side acceleration will largely relieve the performance pressure incurred by XML parsing.

XML parsing is the process of reading an XML document and providing an interface to the user application for accessing the document. An XML parser is a software apparatus that accomplishes such tasks. In addition, most XML parsers check the well-formedness of the XML document and many can also validate the document with respect to a DTD (Document Type Definition) or XML schema[20]. This core should eventually be one of multiple specialized cores in a heterogeneous many-core chip, and acts as the Data Exchange Frontend(DEF), efficiently[11] (in terms of power and performance) parsing the incoming XML data, and then passing the output to other cores for further processing[16].

2. THE XML PARSING PROCESS

XML parsing is a process that scans through the input XML documents, breaks them into small elements, and builds corresponding inner data representation. It is a pre-requisite for any processing of an XML document because an XML document has to be parsed before any other operations can be performed. However, XML parsing is also very expensive due to the high overhead incurred by both computation and memory access.

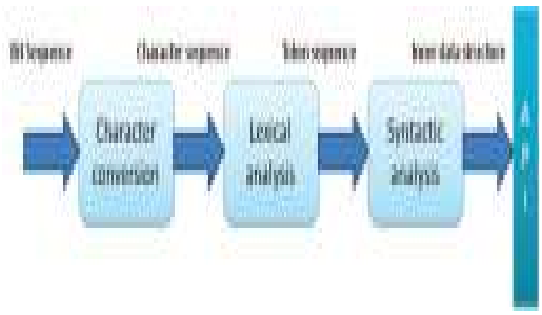

Figure1: XML Parsing Process

Usually, XML data parsing consists of three steps: *character conversion, lexical analysis* and *syntactic analysis*, as shown in Figure 1[1].The first parsing step, *character conversion:*The first parsing step involves converting a bit sequence from an XML document to the character sets the host programming language understands. For example, documents written in Western, Latin-style alphabets are usually created in UTF-8, while Java usually reads characters in UTF-16. In most cases, a UTF-8 character can be converted to UTF-16 by simply padding 8-bit leading zeros. For example, the parser converts "<" "a" ">" from "3C 61 3E" to "003C 0061 003E" in hexadecimal representation. *lexical analysis*: The second parsing step involves partitioning the character stream into subsequences called tokens. Major tokens include a start element, text, and an end element, A token can itself consist of multiple tokens. Each token is defined by a regular expression in the World Wide Web Consortium (W3C) XML specifications.For example, a start element consists of a "<", followed by an element name, zero or more attributes preceded by a space-like character, and a ">".partitions the character sets into subsequences called tokens, like *start element*, *text,* and *end element*. Each token is defined by a regular expression in the World Wide Web Consortium (W3C) XML specifications [8]. The third parsing step, *syntactic analysis*, verifies the structure of tokens by checking that they have been properly nested. It is usually implemented by *pushdown automaton* (PDA)[14]. After syntactic analysis, the PDA organizes tokens into different data representations available for subsequent accesses or modifications via various application programming interfaces (APIs) provided by different parsing models. The first two steps stay the same among different parsing models. However, the third step, *syntactic analysis:* exhibit variable behaviors when different parsing model is applied [6]. The third parsing step involves verifying the tokens' well-formedness, mainly by ensuring that they have properly nested tags. The pushdown automaton (PDA) the following transition rules:

1. The PDA initially pushes a "$" symbol to the stack.
2. If it finds a start element, the PDA pushes it to the stack.
3. If it finds an end element, the PDA checks whether it is equal to the top of the stack.
   - If yes, the PDA pops the element from the stack. If the top element is "$", then the document is "well-formed." Done! Otherwise, the PDA continues to read the next element.
   - If no, the document is not "well-formed." Done!

3. XML PARSING MODELING

Most XML parsers can be classified into two broad categories, based on the types of API that they provide to the user applications for processing XML documents: event-driven parser and tree-based parser[1]. On one hand, event-driven parser simply parses the document and associates any tag it finds along the way with corresponding event, including the start and end of the document, finding a text node, finding child elements, and hitting a malformed element. It transmits and parses XML info sets sequentially at runtime[12]. The parser itself does not store any information of the XML document, so that the application can just access partial data before parsing is completed. As a result, event-driven parser has an enviably small memory footprint and low latency, making it suitable for streaming or forward-only applications. Event-driven model can be further divided into two classes: pull parser and push parser, according to the parser- application interaction. Simple API for XML (SAX) [7,1] adopts the push model, which uses callback functions to report all the events from the parser to the application. In contrast, Sax [18] adopts the pull model, in which clients pull XML data when it is needed so that it can skip uninterested events. As shown in upper part of Figure 2, SAX parses the XML document and then pushes the XML information into application in terms of SAX events. On the other hand, tree-based parser reads the entire content of an XML document into memory and creates an in-memory tree structure to represents parent-childsibling information. Only after parsing is complete, constructed trees can be navigated freely and parsed arbitrarily for the duration of the document processing, which makes this parser suitable for massive and frequent updates. This flexibility, however, comes at a great cost of potentially large memory requirement and significant access delay, especially when large document is processed. Document Object Model (DOM) [8] is the

official W3C standard for tree-based parser. As shown in bottom part of Figure 2, DOM parser processes XML data, creates an object-oriented hierarchical representation of the document and offers the full access to the XML data. In this study, we focus on the two most popular parsing models, namely, SAX and DOM[19].

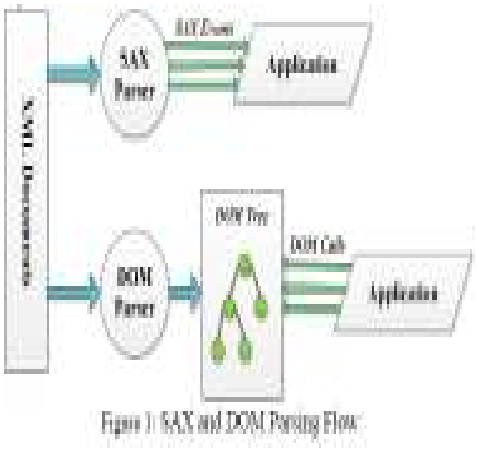

Figure 2: SAX and DOM Parsing Flow

4.PREFETCHING TECHNIQUES

Data prefetching has been proposed as a speculative technique to bridge the speed gap between CPU and memory subsystem[18,1].It alleviates the performance degradation from the long-latency memory accesses by predicting the memory access pattern of the application and speculatively prefetching data that would be used in future computation. Considering that the CPU memory performance gap is on the order of hundreds of processor clock cycles, prefetching is an attractive way to remove the affect of long latency memory accesses.

5.CLASSIC PREFETCHING ALGORITHM

Prefetching techniques has been well studied and lots of algorithms have been proposed. We list some classic prefetching algorithms below.

Sequential prefetching prefetchs the block or blocks that follow the current demanded block, and is fit for the programs with the consecutive memory access pattern [1]. As an improvement, *Sequential tagged prefetching* [1] issues a prefetch upon a cache miss as well as when a prefetched block is referenced for the first time, thus it requires an extra bit per block to mark the prefetch state. The *Sequential prefetching* family increases the performance on a broad range of applications at a low cost, however, at the expense of many useless prefetches.

Stride prefetching makes prefetch requestes according to the observed strides that separate memory addresses flow. Conventional *stride prefetching* uses a record table indexed by the program counter (PC) that associates strides to the loads following this kind of memory access pattern[21]. If address *a* is referenced by a load that hits in the table, the matching entry indicates that the load is following a stride pattern, then prefetcher issues there quest for addresses *a+s*, where *s* is the associated stride.

Strem Prefetching traces a sequence of nearby misses when their addresses follow the same positive or negative direction in a small memory region . In some design,there always exsites a streaming buffer to store the fetched data. *Correlating prefetching* predicts future addresses from tables that record the past memory program behavior .Usually, it generalizes the stride table by registering the stream of addresses associated either to the load PC or to an address that misses in the corresponding cache level.

6.SOFTWARE PREFETCHING Vs. HARDWARE PREFETCHING

According to how prefetching is implemented, it can be classified into two classes: software prefetching and hardware prefetching.

- Software Prefetching

Software prefetching]need to introduce new prefetching instructions into the instruction set architecture (ISA), which could bring data at specified memory addresses into cache. It is assisted by compiler algorithms to insert software prefetching instructions into proper places of the source code. In the preprocessing stage, compiler gets the global information about memory data access pattern, locates those data-sets that are lean towards cache misses and calculates the positions to insert the prefetching instruction. In Intel® Pentium® 4 processor, it enables using the four prefetch instructions introduced with Streaming SIMD Extensions (SSE). These instructions are hints to bring a cache line of data in to various cache levels. Since software prefetching gets the assistance from compiler or programmer, it can acquire a globule map of data accesses, handle irregular access patterns and make more precise prefetchings. However, the insertion of the prefetching instruction is statically determined so software prefetching can not adapt to the phase change of the application. Since new instructions need to be added, recompilation is required, so these do not benefit the scenarios where recompilation is inconvenient.

- Hardware Prefetching

Different from software prefetching that statically inserts prefetching instructions by compiler, hardware prefetching frees the need to expand instruction set architecture and frees the compiler from revising the source code of applications. It automatically determines the data
accesses that might cause cache misses and then make prefetching requests. Its decision is based on the recorded
history information so that it can adapt to the phase change of application. However, it must consume extra
hardware resource and is unable to gain a complete picture of the whole memory pattern. Therefore, it does not suit for the case of irregular data access and short arrays for the penalty of history start-up. In our study, we focus on hardware prefetching for its advantage of no revise of the source code

7.CONCLUSIONS

Different from previous research work which focused on computation acceleration of XML parsing, we first study process of XML parsing, classic prefetching algorithms. We then proposed to make acceleration for XML parsing.

8. FUTURE WORK

The next step of this research project is to integrate memory-side and computations-side accelerators of XML
parsing into a single core, and optimize its performance and power consumption. Then, ultimately, we are going to integrate this core onto many-core architectures to act as a Data Exchange Frontend (DEF).

REFERENCES

[1] Jie Tang, Shaoshan Liu, Chen Liu, Zhimin Gu, Jean-Luc Gaudiot, "Acceleration of XML Parsing Through Prefetching," *IEEE Transactions on Computers*, 19 April 2012. IEEE computer Society Digital Library. IEEE Computer Society, http://doi.ieeecomputersociety.org/10.1109/TC.2012.88

[2] Web Caching and XML Prefetching for Accessing Social Sites from Mobile Environment Renuka Suryawanshi1 ,Prof. Amit Savyanavar2 Pune University, Computer Department,Pune, Maharashtra 411038, India2 Pune University, Computer Department Pune, Maharashtra 411038, India International Journal of P2P Network Trends and Technology- Volume2Issue1-2012 http://www.internationaljournalssrg.org

[3] K.Chiu,M.Govindaraju, and R. Bramley. Investigating the limits of soap performance for scientific computing. In Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002

[4] M. R. Head, M. Govindaraju, R. van Engelen, andW. Zhang.Grid scheduling and protocols benchmarking xml processors for applications in grid web services. In SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, page 121,New York, NY, USA, 2006. ACM Press.

[5] P. Apparao, R. Iyer, R. Morin, and et al., "Architectural characterization of an XML-centric commercial server workload," in 33rd International Conference on Parallel Processing, 2004 P.Apparao and M. Bhat. A detailed look at the characteristicsof xml parsing. In BEACON '04: 1st Workshop on Buildin

[6] M. Nicola and J. John, "XML parsing: A threat to databaseperformance," in Proceeding of the 12th International Conference on Information and Knowledge Management, 2003

[7] International HapMapb Project:http://hapmap.ncbi.nlm.nih.gov/

[8] SAX Parsing Model: http://sax.sourceforge.net

[9] International HapMap Project: http://hapmap.ncbi.nlm.nih.gov/
SAX Parsing Model: http://sax.sourceforge.net

[10] Jie Tang, Shaoshan Liu, Chen Liu, Zhimin Gu, and Jean-Luc Gaudiot:"Acceleration of XML Parsing Through Prefetching" *Fellow, IEEE2012*

[11] Dr. Andrew Blyth, Dr. Daniel Cunliffe, Dr. Iain Sutherland, "Security analysis of XML usage and XML parsing" *in Journal of Computers & Security, Volume 22, Issue 6, September 2003, Pages 494-505.*

[12] B. Naga malleshwar Rao, N. Samba Siva Rao, V. Khanaa, " Exploiting XML Dom for Restricted Access of Information" *in International Journal of Recent Trends in Engineering, Vol. 2, No. 4, November 2009.*

[13] Chen Rongxin, Weibin Chen, "A Parallel Solution to XML Query Application" *in the Proceedings of the International Conference on Computer Science and Information Technology IICCSIT), Vol. 6, pages 542-546,IEEE, 2010*

[14] Kai Ning, Luoming Meng," Design and Implementation of the DTD-based XML Parser", *in Proceedings* of ICCT 2003.

[15] Kotsakis Evangelos, Klemens Böhm, "XML Schema Directory: A Data Structure for XML Data Processing", *in First International Conference on Web Information Systems Engineering (WISE'00),Proceedings, pp 62-69, June 19-21, 2000, Hong Kong, China,* IEEE CS Press

[16] Pan Y., W. Lu, Y. Zhang, and K. Chiu" A Static Load-Balancing Scheme for Parallel XML Parsing on Multi-core CPUs", *in 7th International Symposium on Cluster Computing and the Grid*, IEEE Brazil, May 2007.

[17] Mark E. Williams, Gary R. Consolazio, Marc I. Hoit," Data storage and Extraction in Engineering software using XML", *in Journal of Advances in Engineering Software, Volume 36, Issues 11-12, November-December 2005, Pages 709-719.*

[18] Zacharia Fadika 1, Michael R. Head 2, Madhusudhan Govindaraju ,"Parallel and Distributed Approach for Processing Large-Scale XML Datasets", 10th International Conference on Grid Computing, IEEE/ACM 2009.

[19] Zhang W. and R. van Engelen " A Table-Driven Streaming XML Parsing Methodology for High-Performance Web Services", *in IEEE International Conference on Web Services (ICWS'06), pages 197–204,* 2006.

[20] Zhang W. and R. van Engelen," High-Performance XML Parsing and Validation with Permutation Phrase Grammer Parsers",*in International Conference on Web Services* (ICWS'08) IEEE,2008

[21] Zhang Xiucheng, PAN Zhongshi, XIANG Lei," DOM-Based Research for FOM Parser" in World Congress on Software Engineering," IEEE, 2009.

[22] Zhang Ying, Yinfei Pan, Kenneth Chiu, "Speculative p-DFAs for Parallel XML Parsing". In the Proceedings of International Conference on High Performance Computing (HiPC), pages 388-397, IEEE, 2009.
[23] Zhou Dong, " Exploiting Structure Recurrence in XML Processing", in Eighth International Conference on We Web Engineering, IEEE, 2008