VTD-XML Parsing Performance Optimization based on Helper Thread Sampling Prefetching

Jianxun Zhang
College of Information technology engineering
Tianjin University of Technology and Education
Tianjin, China
zhangjx@tute.edu.cn

Xinyu Qiao Binghui Lin
College of Information technology engineering
Tianjin University of Technology and Education
Tianjin, China

***Abstract*—** With the development of cloud computing, big data and AI technologies, the volume of data is expanding rapidly, and the size of XML documents is also increasing. How to improve the processing efficiency of large-scale XML document data has become a new research hotspot. At present, under the multi-core processor platform, the research work on XML parsing and query is mainly focused on parallel parsing and query, and the performance of XML parsing and query has been greatly improved. However, due to the semi-structured characteristic of XML document and the irregular memory access mode, there are a large number of cache misses in the parsing and query programs, and the access delay becomes the bottleneck of the continuous improvement of performance. To solve this problem, this paper proposes a sample-based helper thread prefetching technology. By using idle multi-core resources, the data required by the main thread is predictably prefetched to the last level of shared cache in advance, so as to hide the delay caused by memory access operations, and achieve the goal of optimizing performance. The experimental results show that the cache miss is reduced by more than 80% after helper thread prefetching, and the total XML parsing performance are improved about 10%, the hot function's performance is improved about 30%.

Keywords—Helper Thread; Parse of XML; Data Prefetch; Performance Optimization

I. INTRODUCTION

With the development of big data and cloud computing technology, the amount of data is expanding exponentially. The calculation and storage of a large number of unstructured and semi-structured data has brought huge challenges to the performance of computer systems. XML (eXtensible Markup Language) is a data standard released by the W3C in 1998. As a flexible semi-structured language independent of the platform, it has the characteristics of high scalability and rigorous structure. XML provides a unified method to describe and exchange structured data independent of applications, and has become the de facto standard for data storage and information exchange. While XML language is widely used in e-commerce, scientific data and digitization of various resources, the scale of XML documents is also growing [1-4]. For XML applications, XML parsing is the key part of XML data processing, and the memory access overhead it introduces directly affects the performance of XML applications. At the same time, XML documents contain a lot of data information. How to quickly find the required data is also one of the research hotspots.

At present, many researchers are committed to improving the processing performance of XML documents by using the parallel processing capability of the processor. Many parallel algorithms to improve the processing speed of XML have been proposed [5-6], the improvement effect of XML parsing was quite good. However, in XML document parsing and XML data query, a large number of memory accesses are required, and memory access delay has become the bottleneck which restricts the further improvement of XMP parsing performance. These parallel algorithms have not solved the memory access delay problem. Data prefetching [7] is a technology of memory access latency hiding. By predicting the memory access behavior of applications, data required by the program is prefetched into the cache in advance. And this can improve the cache hit rate so as to reduce the memory access latency, thus data prefetching can improve the application's performance. Traditional data prefetching is mainly based on the principle of temporal and spatial locality. For applications such as scientific computing, biological computing, and XML data processing, the data structure of these applications is complex, and the memory access mode has no obvious temporal and spatial locality. Therefore, the traditional prefetching method is not applicable to these applications. The helper thread prefetching technology [8-10] is a thread-level software data prefetching technology proposed under the Chip Multi-Processor environment. It makes full use of the idle multi-core resources of the Chip Multi-Processor. Helper thread prefetching uses the multi-thread execution technology to improve the data locality of the program by using the algorithm locality of the source program [11]. A large number of studies and experiments have proved that [8-10] helper thread technology is an effective method to prefetch irregular data.

II. RELATED WORKS

In order to extract data from XML documents, XML documents need to be processed. As the basis of XML document data processing, the parsing operation of XML documents is computationally intensive and storage-intensive. At present, many researchers have studied XML parsing from different aspects, mainly including software and hardware implementation.

The software XML parsing research main includes the following aspects. First, by studying the binary representation

of XML, it can avoid the bottleneck of XML processing. For example, the VTD-XML parser [12] creates 64-bit VTD records in binary format for XML documents during the parsing process. Its advantages are fast parsing and traversing, and it avoids occupying a large amount of memory space. The disadvantage is that XML applications cannot directly use the parsed binary data. For example, Guo et al. [13] proposed a parallel parsing scheme which divides XML documents into several isolated fragments, then they use multithreading data prefetching technology to reduce cache miss rate to optimize parsing performance. Secondly, there are some researches which Use the parallel processing ability of multi-core processors to optimize the analytical performance. For example, W. Lu[5] and Y. Pan [6] et al. have done a lot of parallel parsing research, and proposed a DOM parallel parsing method for XML documents. The parsing process is divided into two stages which are pre-parsing and parallel parsing. In the pre-parsing stage, a framework is generated, which is used to guide the data partitioning and final merging process. On this basis, they proposed parallel optimization methods, such as Meda-DFA method[14] and parallel Transducers method[15]. The framework generated in the pre-parsing stage can effectively guide data partitioning and simplify the complexity of merging, but the entire algorithm needs to read the XML document twice. Thirdly, R. Chen et al. [16] proposed the method of dividing XML into subtrees for parallel parsing. After the parallel parsing is completed, multiple subtrees are merged into the final result. The whole process does not need to generate a parsing framework, and supports arbitrary division of XML data. The XML document only needs to be read once in the whole parsing process, but the subtree merging process is relatively complex. Fourth, Michael R [18] proposed a solution to divide the parsing process into several stages, and it use different threads to execute different parsing stages in a pipeline way, thus expanding the parallelism of parsing. This method all need to read the entire XML document first, and some of the serial execution stages will cause performance bottlenecks.

As far as the XML hardware research is concerned, there are also some researches to improve the XML parsing performance. Fox example, Both J. Moscola [17] and Z. Dai [18] use FPGA to implement XML parsers. The former proposes to automatically map conventional expressions to FPGA hardware. However, since XML syntax rules are not a conventional language, this method is not suitable for promotion. The latter can perform XML rule syntax checking, schema validation, tree construction and other operations, but is only applicable to tree-based parsing models. B. Nag et al. [19] propose a XOE engine to speed up parsing by handing some basic parsing operations to the XOE engine offline. J. Tang et al. [20] proposed a storage-side acceleration method combining data prefetching, which uses the exclusive core for parsing operations. This storage acceleration method combines the storage access characteristics of XML, and further tailors the exclusive core in the isomorphic system. It can cover the computing side and accelerate the XML parsing process at the computing side and the storage side. These methods can effectively speed up the parsing process, but additional hardware logic needs to be added.

Compared with hardware accelerated parsing, software parsing is easier to implement without adding complex hardware logic devices, but requires adding additional code, which increases the complexity of the program. This research is the same as the previous research, and uses the CMP platform. The difference is that it uses the helper thread prefetching method to optimize the XML performance, and proposes the helper thread prefetching method based on sampling to speed up the XML parsing.

III. VTD-XML PARSING MODEL

VTD-XML parser is a lightweight XML parsing tool. It has the advantages of fast parsing speed and small memory space. Its memory size is about 1.3-1.5 times that of XML documents. Because XML documents are stored in memory, it can frequently operate on XML documents. VTD-XML combines the advantages of DOM parsing and SAX parsing. Studying VTD-XML parsing has an important impact on the development of XML parsing technology.

This paper proposes a helper thread prefetching technology to optimize the performance of VTD-XML parsing. By analyzing the VTD-XML parsing performance and memory access characteristics, helper thread prefetches speed always lay behind the main thread. Therefore, sampling prefetch was proposed to solve the prefetch timeliness problem. In other words, helper thread doesn't always working for data prefetching, it prefetches data sampling so as to cache up the main thread.

VTD-XML[12] (Virtual Token Descriptor) is an XML parsing method without extraction. In the parsing process, XML documents are read into memory in binary form without encoding and decoding. After the basic configuration of the parser is completed, the parser enters the finite state machine mode for parsing, scans from the beginning of the document, and obtains the relevant information of each character object (Token). The information is stored in VTD-Record with a 64-bit long integer, and the parent-child relationship between Tokens is recorded through Location Cache. When the XML content is extracted, the position and other information in the record are used to decode the original bit array and return the string.

VTD is a 64-bit fixed-length numeric record, which records the starting position (offset), length (length), depth (depth) and type of token (element label) of each element.

As shown in Fig. 1, the VTD format records the location and type information of each element, and all operations on XML are based on this data structure.

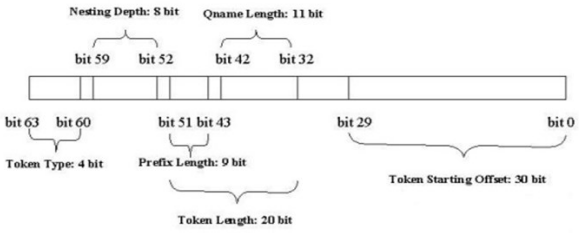

Fig. 1 VTD Format[12]

Location Cache LC (Location Cache) is simply a tree-shaped table structure built with VTD's depth as the standard. The LC entry is also a 64-bit long numeric type. The first 32 bits represent the index of a VTD, and the last 32 bits represent the index of the first child of the VTD. Using this information, the location of any element can be calculated, and the specified location can be reached in a few steps. The process of base traversal presents an irregular data-intensive memory access feature.

IV. XML HELPER THREAD PREFETCHING TECHNOLOGY BASED ON SAMPLING

Helper-threading technology uses other idle core resources in the multi-core processor to prefetch data and execute in parallel with the main thread. Since the helper thread is a lightweight thread extracted by the main thread, the helper thread must be ahead of the main thread in the execution process, so the data needed in the main thread can be prefetched to the shared cache in advance. When the main thread accesses the data, the data has been prefetched to the cache to avoid the delay caused by the main thread accessing the memory data, thus speeding up the execution speed of the main thread.

A. Helper Thread Prefetching Model

In view of the memory access characteristics of irregular data-intensive applications, we established the help thread prefetch model based on irregular memory access applications as follows:

Suppose C_m is the execution core required by the main thread on the multi-core platform. C_h is the execution core required by helper thread. The application program is composed of multiple instruction sequences s_i, which represents by $\Sigma = \{s_1, s_2, ...s_n\}$. $Rv(s_i)$ is the set of read-only variables in the instruction sequence s_i and $Wv(s_i)$ is the set of write variables in the instruction set s_i.

Definition 1 Parallel execution: $s_i \parallel s_j$

For $s_i, s_j \in \Sigma$, if there is no read-write correlation and write-write correlation between the variable sets operated by two instruction sequences, that is, if it satisfied $Rv(s_i) \cap Wv(s_j) = \Phi$, $Rv(s_j) \cap Wv(s_i) = \Phi$ $Wv(s_i) \cap Wv(s_j) = \Phi$, it is said that s_i can be paralleled with s_j, and it is recorded as $s_i \parallel s_j$. That is to say, it has nothing to do with the execution order between s_i and s_j.

Definition 2 Hotspot module M: For any module M in application A, if there is a threshold ε_1 and ε_2, and meet the two conditions respectively which are $Cycle(M)/Cycle(A) \geq \varepsilon_1$ and $LLC\_Miss(M)/LLC\_MISS(A) \geq \varepsilon_2$, then M is called a hotspot module. Among them, Cycle (X) and LLC_ Miss (X) represents the number of program running CPU clocks and the number of LLC misses obtained by profiling program X.

Definition 3 Hot cycle H (L): For any loop L in hot module M, which include nested loops, if the maximum number of iterations $Iterations(L) \geq \varepsilon_3$, L is called a hot loop.

If a hot program block $\Gamma = (\gamma_1, \gamma_2, ...\gamma_n)$ is composed of multiple instruction sequences γ_i, and the instruction sequence meets the conditions $Rv(\gamma_i) \neq Rv(\gamma_{i+1})$ and $Wv(\gamma_i) \neq Wv(\gamma_{i+1})$, the main thread execution core C_m executes the program in sequence γ_1, γ_2, $\ldots \gamma_n$, then the help thread execution core C_h executes a compact version $\Gamma' = (h_1 h_2 ... h_n)$ of the program Γ and satisfies the conditions in definition 2, then there is $\Gamma' \parallel \Gamma$, therefore the helper thread is a non-traditional parallel technology. However, the difficulty of the helper thread prefetch control is how to ensure that the helper thread execution core has already executed the instruction sequence h_j before the main thread execution core is executing γ_j. That is, the helper thread prefetch has completed the prefetch program sequence before the main thread a specified distance.

In order to ensure the timeliness of helper thread prefetching, the selection of parameter values such as helper thread prefetching distance and prefetching workload becomes the key factor to helper thread prefetching quality.

B. Helper thread Prefetching based Sampling

The prefetch effect of the helper thread is different for different programs, even reducing program performance. For example, in a program with a small workload of computation, a large amount of time between the helper thread and the main thread is spent on accessing data. It is possible that the helper thread does not cache up the main thread's execution. In this scenario, it will not only reduce the memory access delay, but also cause shared cache resource competition and affect performance. For programs with large amount of computation, because the helper thread is lightweight, the helper thread will prefetch a large amount of data into the cache. Before the main thread uses the data, it is possible to replace the prefetched data by the helper thread. Although the helper thread has done a lot of prefetching work, it has not played a role in accelerating performance, but has caused a lot of cache pollution. Therefore, on the basis of considering the calculation and memory access time, the sampling helper thread technology proposes to use three parameters (K, P, B) to control the prefetch process. Through the combination of different parameters, the prefetch distance, prefetch workload and synchronization distance of the helper thread prefetch can be controlled.

Fig. 2 shows the sampling helper thread prefetch model on the linked data structure. The K, P and B parameters are respectively SKIP_ SIZE、PUSH_ SIZE and BLOCK_ SIZE

The actual amount of data prefetched by the helper thread is controlled by the parameter Push_size. On the whole, the helper thread presents the characteristics of sampling prefetching, which is called the sample-based helper thread prefetching model in this paper. skip_size means the prefetch distance, we name skip_size + push_size as a block.

BLOCK_size is the number of blocks to synchronize between main thread and helper thread.

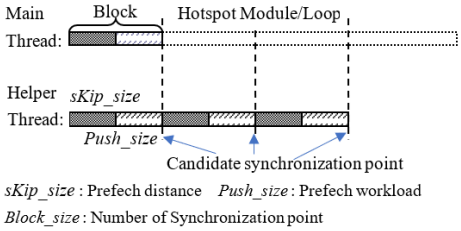

Fig. 2 Schematic diagram of helper thread prefetch based on sampling

Sampling prefetching technology can be deepened into different prefetching methods by controlling different combinations of three parameters, such as when SKIP_SIZE =0, it is suitable for the case that the main thread has sufficient computing workload, which is non-sampling prefetching; When BLOCK_SIZE=1, each block is synchronized once; When sKip_size=1 and BLOCK_ SIZE=∞, indicating that the helper thread is pre-traversal prefetch, there is no synchronization between main thread and helper thread.

C. *Sample-based helper thread parameter definition*

In the helper thread prefetch technology, the parameter value affects the prefetch distance and prefetch effect of the helper thread. The parameter definition is as follows.

Parameter K refers to the prefetch distance, which means that the data prefetch will not start immediately after the helper thread is triggered. After skipping K loop iterations, the helper thread will start prefetching data. K is represented by SKIP_ SIZE in Fig. 2.

Parameter P represents the prefetch workload which is also counted by loop iterations. Parameter K is used to guarantee the helper thread run faster than main thread, it is also necessary to specify how much data the helper thread needs to prefetch. The workload of prefetch is expressed in P. Although, the prefetch workload cannot be too much so as to avoid cache pollution. P is represented by PUSH_SIZE in Fig. 2.

Parameter B represents the synchronization distance. And we regards (K+P) as a data prefetch block. In order to control the distance between the helper thread and the main thread within a certain range, it is necessary to synchronize the main thread and the helper thread. Parameter B is used to control the synchronization distance. For example, if B=2, it means that the synchronization will occur once after two data blocks. B is represented by B_ SIZE in Fig. 2.

By adjusting the three parameters, the helper thread can do data prefetching for all instructions, or skip some loops to complete data prefetching for some instructions. The helper thread prefetch workload depends on the ratio of hot loop calculation time to memory access time. As for those applications with small amount of computation workload, the main thread can undertake a part of the data loading work by adjusting parameter K, so as to guarantee the helper thread faster than the main thread. As for those applications with large amount of calculation workload, by adjusting parameter B to ensure that the helper thread can't run far away from the main thread. The three parameters can effectively control the distance between the helper thread and the main thread, as a result, the effectiveness of prefetching is improved.

D. *Construction of XML Parsing Helper Thread*

Aims to construct a lightweight helper thread for the XML parser, the main steps include the selection of hot spots, slicing to obtain effective loop code, and generation of helper the thread.

(1) Selection of hot spots

A large number of experiments show that the cache miss mainly occurs in the loop, so the selection of hot spots is mainly focus on the selection of the loop. By using the Vtune performance analyzer[21], we can find the main cache miss statement is temp=vg ->XMLDoc [vg ->offset]. This statement is to obtain characters from the array. This statement is contained in the getChar function shown in Fig. 3. After finding the cache miss location, we can find the loop containing the cache miss statement from inside to outside. Because the parsing process is a finite state machine, the getChar function will be called in each state. In some states, getChar is also nested in the loop. To simplify the construction of the helper thread, we select the outermost loop as shown in Fig. 4 as the hot loop.

```
Static int getChar(VTDGen *vg){
    ...
    case FORMAT_UTF8
      temp=vg->XMLDoc[vg->offset];
      if(temp<128){
            vg->offset++;
            return temp;
      }
    return handle_utf8(vg,temp)
    ...
}
```

Fig. 3 Hot Function of Cache Misses

```
while (TRUE){
  switch(parser_state){
     case STATE_LT_SEEN:
        vg->temp_offset = vg->offset;
        vg->ch = getChar(vg);
        if (XMLChar_isNameStartChar(vg->ch)){
              vg->depth++;
              parser_state = STATE_START_TAG;
        } else{
              ...
        }
     case STATE_START_TAG;
        while(TRUE){
           vg->ch = getChar(vg);
              ...
        }
     ...
  }
  ...
}
```

Fig. 4 Area of Hot Iterations.

(2) Construction of helper thread

After selecting the hot spot area, we can obtain effective loop code through slicing. In order to construct a lightweight helper thread, we only select the statementes that affect the calculation of data access address and loop control statements. We delete the statements for writing operations in the main program. The process of parsing is to scan the string array in sequence. Therefore, some control statements such as jump

instructions can be ignored when obtaining the slice, so as to achieve a streamlining version thread code. Frequent call of functions will also affect the program performance. Therefore, when we construct the effective loop code, we directly select the main cache missing statement in the getChar function to replace the call of the getChar function.

(3) Trigger of helper thread

When generating a helper thread, two important issues muse be considered, one is the triggering of the helper thread and the other is synchronization between the helper thread and the main thread. Because the array address in memory needs to transfer to the helper thread, main thread needs to pass the first address of the array to the helper thread. Therefore, it is necessary to configure the parameters of the VTD-XML parser before generating the helper thread. In order to ensure that the helper thread prefetches data before the main thread, once the helper thread is generated, the helper thread starts to run directly and blocks to wait for the trigger of the main thread.

The control mode between the helper thread and the main thread is called sampling prefetch. After the helper thread starts, the first address of the prefetch object must be obtained from the main thread. Then the helper thread can execute independently of the main thread. After prefetching certain data, the helper thread actively synchronizes with the main thread.

V. EXPERIMENTAL EVALUATION AND ANALYSIS

A. *Experimental platform*

The processor used in this experiment is the Q6600 processor, which is a CMP-based product launched by Intel. It consists of four cores. Each core is equipped with a dedicated L1 cache. The four cores are divided into two groups. Each group of cores shares a 4M L2 cache. The Q6600 processor supports hardware prefetch and software prefetch instructions. When using the helper thread to prefetch data, the main thread and the help thread must be assigned to the same group of cores, so that the main thread can use the L2 cache data prefetched by helper thread. The performance analyzer used in this experiment is the Vtune tool which can conduct comprehensive performance testing. It can be used to determine the hot modules, functions, threads, codes, etc. In the experiment, the memory access module in the Vtune performance analyzer can be used to analyze the impact of memory access problems on program performance during program operation, and event-based sampling (EBS) can be used for analysis. The sampling events concerned in the experiment are mainly include CPU_CLK_UNHALTED and MEM_LOAD_RETIRED.L2MISS. CPU_CLK_UNHALTED. CORE refers to the number of CPU clocks corresponding to each instruction. MEM_LOAD_RETIRED.L2MISS is the number of L2 cache misses caused by each LOAD instruction.

B. *XML Parsing Bechmark*

This experiment uses the XMark-1[22] test benchmark to generate four XML documents with the same structure but different file sizes, and analyzes and tests them on the platform. Table I shows the specific XML document size. Because VTD-XML parses documents very fast, in order to better observe the experimental results and reduce the impact of the environment on the program execution time, we test the time of 100 times parsing execution and the number of L2 cache misses in every test benchmark.

TABLE I. TEST BENCHMARK

Test Benchmark	Size（MB）
test0	222.9
test1	167.1
test2	111.1
test3	55.3

C. *VTD-XML parsing time analysis*

In the experiment, the prefetching effect of the helper thread is measured by analyzing the parsing time and cache loss of XML documents of different sizes in the test benchmark given in the TABLE II.

Fig. 5 is the parsing execution time of the four XML documents in the test benchmark which collected by Vtune. It can be seen from the figure that the parsing time increases with the increase of the size of the XML document. It can be observed that the parsing time is directly proportional to the size of the test file.

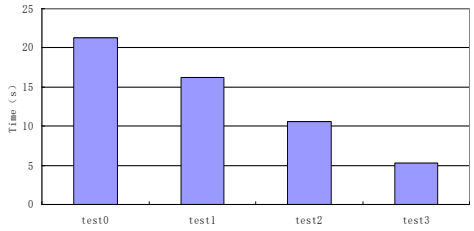

Fig. 5 Paring time of test Benchmark

D. *VTD-XML parsing cache missing analysis*

Fig. 6 shows the number of MEM_LOAD RETIRED.L2_MISS. The result shows that the number of L2 miss increases with the size of XML document. It's also proportional to the size of the test file.

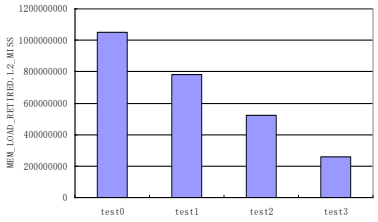

Fig. 6 MEM_LOAD_RETIRED.L2_MISS of Test Benchmark

Fig. 7 shows the comparison of total number of CPU_CLK UNHALTED. CORE between the hot function and total functions. The execution time of getChar function is about 34.8% of total execution time in the four documents. The

number of clock cycles spent in the parsing process is large. When optimizing the program performance, we can reduce the L2 cache miss of the getChar function, so as to speed up the parsing performance.

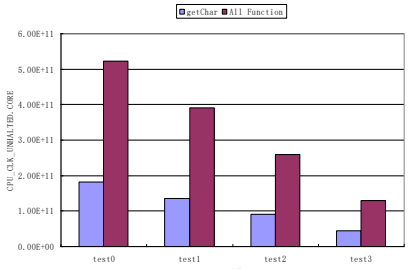

Fig. 7 Comparison between hot function and all functions

E. Execution Time analysis

Table II shows the execution time and speedup of the four test sets before and after the helper thread prefetching. Push_time is the execution time of the program after joining the helper thread, Ori_time is the execution time of the original program. It can be seen from the table that compared with the original program, the speedup of test1 is 1.12 and other three XML document's speedup is about 1.09 after the helper thread prefetching. The performance of the program is improved in terms of program execution time. However, if we just consider the hot function, the speedup of the getChar is about 1.3. The reason for it is that the total program includes the spawn overhead of helper thread. In addition, in the process of testing, the parameter values (K, P and B) are determined according to the parsing process of test1 documents, so the speedup obtained in test1 is the highest, while the speedup of other documents is relatively lower. It indicates that the parameter values have an impact on the prefetch effect of XML documents of different sizes.

TABLE II. SPEEDUP OF ORIGINAL XML PARSING

Test Set	Push_time(s)	Ori_time(s)	speedup
test0	19.54449	21.30836	1.090249
test1	14.46391	16.26688	1.124653
test2	9.778164	10.62295	1.086395
test3	4.858249	5.308914	1.092763

VI. CONCLUSIONS

This paper proposes a sampled helper thread prefetching scheme which aims for the performance of VTD-XML parsing. By constructing a lightweight helper thread according to XML parsing characteristics, the cache miss of the main program is greatly reduced, and the speed of XML parsing is improved. Subsequent work includes analyzing the influence of hardware prefetcher on helper thread, and the influence of different combinations of hardware prefetcher on helper thread prefetching. In this paper, single-thread VTD-XML parsing is used in the experiment, and the helper thread prefetching technology can be applied to multi-thread VTD-XML parsing in the future research.

ACKNOWLEDGMENT

Thank the anonymous reviewers for their comments and suggestions on this paper. The work of this paper has been supported by the Tianjin University Science and Technology Development Fund (JWK1618).

REFERENCES

[1] Gannon D,Krishnan S,Fang L,et a1.On Building Parallel&Grid Applications:Component Technology and Distributed Services[C].Proceedings of the Second International Workshop on Challenges of Large Applications in Distributed Environments, 2004.

[2] eBay Developers Program. http://developer.ebay.com/developercenter/soap/.

[3] Singh G,Bharathi S,Chervenak A,et a1.A Metadata Catalog Service for Data Intensive Applications [C].Proceedings of the ACM/IEEE Conference on Super Computing,2003.

[4] Takase T,Miyashita H,Suzumura T,et a1.An Adaptive,Fast,and Safe XML Parser Based on Byte Sequences Memorization[C].Proceeding of the 14th International Wor1d Wide Web Conference,2005:692—701.

[5] Lu W,Chiu K，Pan Y.A Parallel Approach to XML Parsing[C].7th IEEE/ACM International Conference on Grid Computing,2006:223-230.

[6] Yinfei Pan, Wei Lu, Ying Zhang, Kenneth Chiu A Static Load-Balancing Scheme for Parallel XML Parsing on Multicore CPUs[C].7th IEEE International Symposium on Cluster Computing and the Grid,2007:356-365.

[7] Steven P Vanderwiel, David J Lilja. Data Prefetch Mechanisms[J]. ACM Computer Surveys, 2000, V32(2):174-199.

[8] Jianxun Zhang, Zhimin Gu, Yan Huang, Ninghan Zheng, Xiaohan Hu. Helper Thread Prefetching Control Framework for Chip Multi-processor [J]. International Journal of Parallel Programming, Vol 41(6),2013.

[9] Jianxun Zhang, Zhimin Gu, Yan Huang, Min Cai, Xiaohan Hu. Solving parameter selection problem of helper thread prefetching via realtime hardware performance monitoring[C]. In Proceedings of 13th International Conference on Parallel and Distributed Computing, Applications, and Technologies, PDCAT2012, 65-70, 2012.

[10] Yan Huang, Zhimin Gu, Min Cai, Jianxun Zhang, Ninghan Zheng. Estimating Effective Prefetch Distance in Threaded Prefetching for Linked Data Structures [J].International Journal of Parallel Programming. 2012, 40(5):465-487.

[11] B. Jacob, S. W. Ng, David T. Wang. Memory System Cache, DRAM, Disk［M］. Morgan Kaufmann Publishers, Elsevier Incoration, 2008: 64-65.

[12] VTD-XML:the future of XML processing[EB/OL].[2022-10].http://vtd-xml.sourceforge.net.

[13] XianYong Guo,Xingyuan Chen,Yadan Deng. VTD-XML Parsing Performance Optimization Based Chip Multiprocessor[J]. Journal of Frontiers of Computer Science and Technology,2013,7(8):736-746.

[14] Y. Pan, Y. Zhang, K. Chiu, Parallel XML parsing using meda-DFAs[C].the 3rd IEEE International Conference on e-Science and Grid Computing, 2008:237-244.

[15] Y. Pan, Y. Zhang, K. Chiu, Simultaneous transducers for data-parallel XML Parsing[C]. In Proceeding of the 2008 IEEE International Parallel & Distributed Processing Symposium, 2008:1143-1154.

[16] RongXin Chen,Husheng Liao,WeiBin Chen. XML Parsing Schema Based on Parallel Sub-tree Construction[J]. Journal of Computer Science,2011,38（3）,191-195.

[17] J. Moscola,J. W.Lockwood,Reconfigurable Content-basedRouter using Hardware-Accelerated Language Parser [J]. In the ACM Transactions on Design Automation of Electronic Systems, 13(2), 2008:1-25.

[18] Zefu Dai, Nick Ni, Jianwen Zhu. A 1 Cycle-Per-Byte XML Parsing Accelerator[C]. FPGA'10, 2010:199-208.

[19] B. Nag. Acceleration Techniques for XML processors[C]. In XML Conference & Exhibition, November 2004.

[20] Jie Tang, Shaoshan Liu, Chen Liu, et al.Acceleration of XML Parsing Through Prefetching [J]. IEEE Transactions on Computers,62(8):2013:1616-1628.

[21] https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html

[22] http://xml-benchmark.org/