软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(6):1196−1206[doi: 10.3724/SP.J.1001.2013.04281]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种混合并行 XML 解析方法*

方跃坚[1], 余枝强[2], 翟 磊[2], 吴中海[1,3]

[1](北京大学 信息科学技术学院,北京 100871)

[2](英特尔上海软件研发中心,上海 200333)

[3](北京大学 软件与微电子学院,北京 102600)

通讯作者: 吴中海, E-mail: wuzh@pku.edu.cn

摘 要: 设计了一种混合并行 XML 解析方法.该方法由轻量级事件划分、事件级并行解析和后处理三阶段组成.使用 SIMD 指令来加速事件划分.阶段级处理使用软件流水线并行技术.同时使用了事件级数据并行技术和流水线并行技术,所以该方法是一种混合并行方法.与其他方法相比,该方法具有高效并行解析和低通信开销的优势.在基于 8 核 Intel Xeon X7560 CPU、Linux 操作系统机器上的测试结果表明,与现有其他方法相比,该方法能够达到更高的加速以及更好的可扩展性.

关键词: XML;混合并行处理;轻量级预处理;SIMD;事件流

中图法分类号: TP301 **文献标识码**: A

中文引用格式: 方跃坚,余枝强,翟磊,吴中海.一种混合并行 XML 解析方法.软件学报,2013,24(6):1196−1206. http://www.jos.org.cn/1000-9825/4281.htm

英文引用格式: Fang YJ, Yu ZQ, Zhai L, Wu ZH. Hybrid parallel method for XML parsing. Ruan Jian Xue Bao/Journal of Software, 2013,24(6):1196−1206 (in Chinese). http://www.jos.org.cn/1000-9825/4281.htm

Hybrid Parallel Method for XML Parsing

FANG Yue-Jian[1], YU Zhi-Qiang[2], ZHAI Lei[2], WU Zhong-Hai[1,3]

[1](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Intel Software Developing Center, Shanghai 200333, China)

[3](School of Software and Microelectronics, Peking University, Beijing 102600, China)

Corresponding author: WU Zhong-Hai, E-mail: wuzh@pku.edu.cn

Abstract: This paper presents a hybrid parallel method for XML parsing, which consists of a lightweight events partition stage, followed by an event-level parallel parsing stage, and a final post-processing stage. SIMD instructions are used to speed up the processing in the events partition stage. Software pipelined processing is achieved at stage level. The study combined event-level data parallel parsing technique and pipelined processing technique to create a hybrid parallel method. Compared to other parallel solutions, the method has the advantage of a much more efficient parallel processing with low synchronization overhead. The method is tested on a Linux machine with Intel Xeon X7560 CPU for 8 cores, and the results show the method can achieve a much higher speed up and better scalability than other software implementations done to date.

Key words: XML; hybird parallel processing; lightweight preparing; SIMD; event stream

可扩展标记语言(extensible markup language,简称 XML)广泛应用在网络服务、数据库和文件处理等领域.XML 具有文档内容和结构完全分离、互操作性强、规范统一、支持多种编码和可扩展性好等特点.XML 文档可包括多层嵌套的数据结构,因此,XML 解析通常变得比较复杂.在缺乏高效 XML 解析方法的情况下,XML 解

* 基金项目: 国家科技支撑计划(2012BAH06B01)
收稿时间: 2011-12-26; 修改时间: 2012-05-18; 定稿时间: 2012-06-25

析成为 XML 应用的性能瓶颈.已有很多改进 XML 解析性能的研究成果,如模式引导的解析、冗余编码转码、表格驱动方法和硬件加速等[7−15].

在处理器的发展方面,多核处理器已经得到广泛应用.基于多核结构设计并行 XML 解析方法,可以显著提高 XML 解析速度.Lu[1]提出一种基于预处理的并行解析方法,该方法首先对 XML 文件进行预处理,然后完全地并行解析 XML 数据.在后续的研究中,Lu[2]对预处理并行化以提高预处理速度.Michael[3]将 XML 解析分为 3 个阶段,包括数据载入、XML 解析和结果产生,通过对线程进行调度,使 3 个阶段以流水线方式运行.Parabix[4]使用并行位流技术,通过基于英特尔架构的 SSE 向量指令来加速 XML 解析. Pan[5]提出一种混合并行 XML 解析方法,该方法由 4 个阶段组成,阶段级处理上使用流水线技术,并且在第 1 阶段和第 3 阶段并行处理数据.在第 1 阶段即预处理阶段,XML 数据被分割成相同大小的多个块,使用 meta- DFA[2]技术来进行并行预处理.Li 等人[6]提出一种基于关键元素的并行解析方法,使预处理与解析并行运行.在预处理中,首先选择关键元素,然后抽取出元素间的依赖关系并进行数据划分.

尽管现有方法利用并行处理,但很多方法由于增加了额外的复杂性,阻碍了并行处理的潜在性能提升.比如在有些现有并行方法中,解析程序先将 XML 划分为相同大小的多个数据块,这导致一个 XML 元素的内容可能分布在两个不同的块中,解析程序只能以推测的方式解析数据.而且在并行解析多个数据块时,需要检查各个数据块之间的数据依赖关系,这导致并行解析线程间的频繁通信.当线程等待通信应答时,将暂停运行或减缓处理速度,从而减少了并行处理的潜在速度优势.

现有的一些并行方法产生层次结构的输出,比如 DOM 结构(document object model).在这类结构中,父节点和子节点间需要建立连接关系.当并行解析线程产生的解析结果被合并为这类结构时,会产生额外通信开销,从而降低解析效率.

现有的其他一些解析方法中,首先对 XML 数据进行预处理.这类方法能够准确划分数据块边界,但预处理仍然有些复杂,耗费时间较多,成为性能瓶颈.

本文的贡献在于设计了一种混合并行 XML 解析方法,该方法由轻量级事件划分阶段、事件解析阶段和后处理阶段组成.在事件划分阶段,事件边界被准确快速识别.SIMD 指令用来加速该识别过程;事件解析阶段为数据并行解析阶段,独立地对多个数据块进行并行解析,产生子事件流作为输出;后处理阶段检查和解决子事件流中的未解决事件,并将子事件流连接为完整的事件流.软件流水线并行技术用于阶段级处理,以进一步加速解析过程.本文同时使用了事件级数据并行技术和流水线并行技术,所以本文方法是一种混合并行方法.

本文的创新点在于:

(1) 设计了一种准确、快速的 XML 事件边界识别预处理方法,同时采用 SIMD 指令加速该预处理过程.预处理准确划分事件边界,根据事件边界信息划分多个数据块,在事件解析阶段能够独立地并行解析这些数据块,也极大地减少事件解析阶段的通信开销;

(2) 事件级数据并行解析方式使得线程间同步和通信开销很小;

(3) 从整体上设计了一种新的混合并行 XML 解析方法,实现高效 XML 解析.

1 并行 XML 解析研究背景

并行性可分为流水线并行、任务并行和数据并行.流水线并行方法将处理过程分为多个阶段,给每个阶段分配一个或多个核.一个阶段的输出成为另外一个阶段的输入.任务并行方法将处理过程分为多个可同时运行的任务.数据并行方法将数据分为多个数据块,然后将每个数据块分到不同的核上进行处理.

对于 XML 解析,目前数据并行处理的主要难题在于多个数据块之间最好能相互独立.因为 XML 数据内在的串行性,设计快速准确的 XML 预处理方法,将 XML 数据划分为多个独立的数据块,是一个具有挑战性的难题.

将流水线并行与数据并行方法结合起来,设计一种混合并行的方法,能提高灵活性与负载均衡.例如,可以设计这样一种混合并行方法:数据间的依赖性可以在流水线的第 1 阶段通过严格线性处理得到解决,该阶段可以对输出的数据间的顺序依赖关系进行标记;在第 2 阶段,以数据并行的方式,独立地同时处理多个数据块,并根

据第 1 阶段的标记信息来解决数据块间顺序相关的处理问题.这种方法要求第 1 个阶段能够快速地解决数据间的依赖关系,将更多的计算密集的处理放在数据并行处理阶段.

对于纯流水线并行方法,当增加一个核时,需要增加一个新的阶段,对软件进行重新设计以达到负载均衡;而对于混合并行方法,当增加一个新的核时,只需简单将核分配给数据并行阶段即可.因此,混合并行方法具有更好的灵活性.

对于纯流水线并行方法,当有任何一个阶段处理较慢时,由于阶段之间是线性的,进行后续阶段处理的核会被阻碍;而对于混合并行方法,由于数据并行处理阶段计算任务重,比其他阶段处理计算量大很多,当有一个核空闲时,只需要将该核分配到数据并行处理阶段.因此,混合并行方法能很好地解决负载均衡问题.

2 方法描述与实现

2.1 整体解析流程

本文设计的 XML 整体解析流程如图 1 所示.输入 XML 数据流后,第 1 阶段是事件划分阶段.在该阶段,事件边界被准确快速识别;第 2 阶段是事件解析阶段,并行解析划分的多个数据块,产生子事件流;第 3 阶段是后处理阶段,后处理模块对多个子事件流中未解决的事件进行检查,产生最后的结果事件流.

因为第 2 阶段占用 XML 解析的主要时间,而在第 2 阶段可以对多个数据块进行并行解析,因此能加速解析过程.多个数据块从第 1 阶段~第 3 阶段的处理以软件流水线方式进行,这也缩短了解析时间.

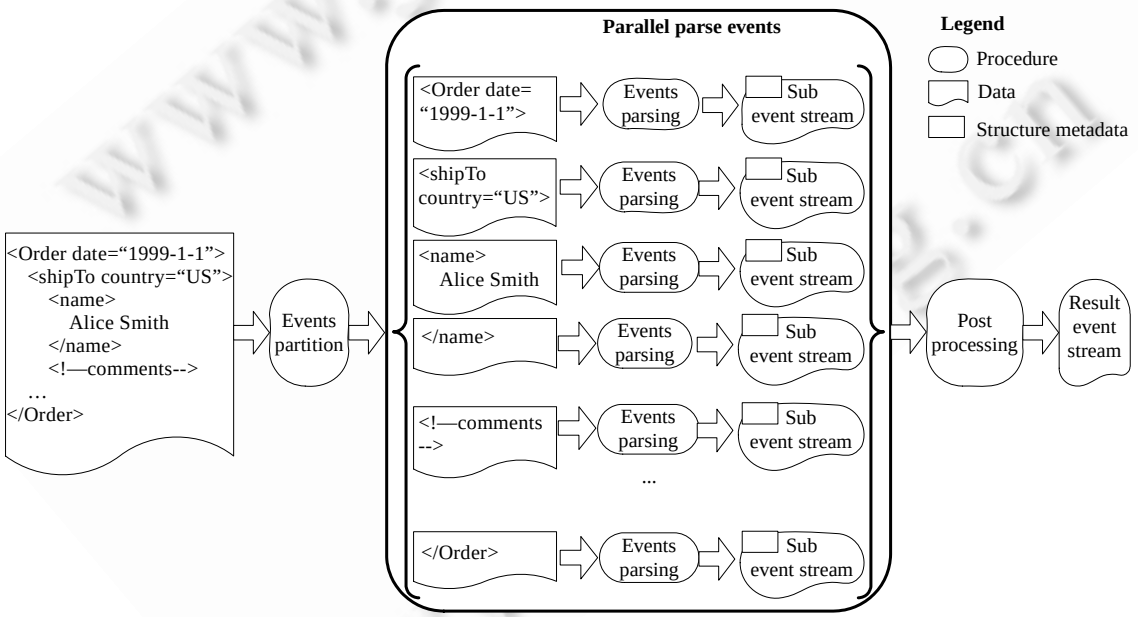

Fig.1 Overall process of XML parsing

图 1 XML 解析过程

2.2 事件流介绍

事件流类似 SAX(simple API for XML)[18],它是 XML 解析结果的一种内部表示,在 Intel 公司 XML 软件套件中得到应用.事件流由一系列缓存组成,包含 XML 数据信息,如元素、属性、字符内容、评论等.如图 2 所示,事件流开始是一个开始元素(start of element or SE),元素名为“Order”.接着是一个属性(attribute or A),属性数据为 date “1999-1-1”.字符数据段以“CD”(character data)表示,如“Alice Smith”是名为“name”元素的字符数据段.结束元素以“EE”(end of element)表示.评论以“C”(comment)表示.

事件流具有简洁高效的特性,因此适合要求高性能低内存的应用.与 DOM 结构不同,事件流中不同事件之间是相互独立的,能支持高效并行处理.DOM 结构中需要建立节点间的相互连接,如父节点、子节点以及邻居节点间的连接,这导致在解析过程中产生较大开销.另外,使用 DOM 结构对于大的文件会消耗较大的内存,使得处理大的文件时比较困难.

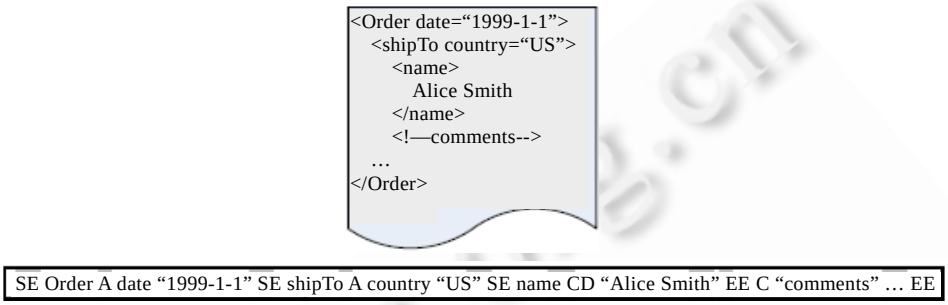

Fig.2 Event stream example

图 2 事件流示例

2.3 事件划分阶段

2.3.1 事件划分流程

通过分析 XML 标准,我们发现,XML 数据中"<"的位置可分为以下两种情况:

a) "<"为一个开始元素或者结束元素或者空元素的开始字符,或者为评论或者字符数据段(CDATA section)或者处理命令(processing instruction)中的开始字符;

b) "<"为评论或者字符数据段或者处理命令中的一个内容字符.

基于以上分析,如果我们设计一种预处理过程来识别所有"<"字符,那么就可以标记 XML 事件的边界.基于预处理的结果信息,我们可以进一步对 XML 数据进行事件级并行的解析.设计的识别"<"过程如图 3 所示.

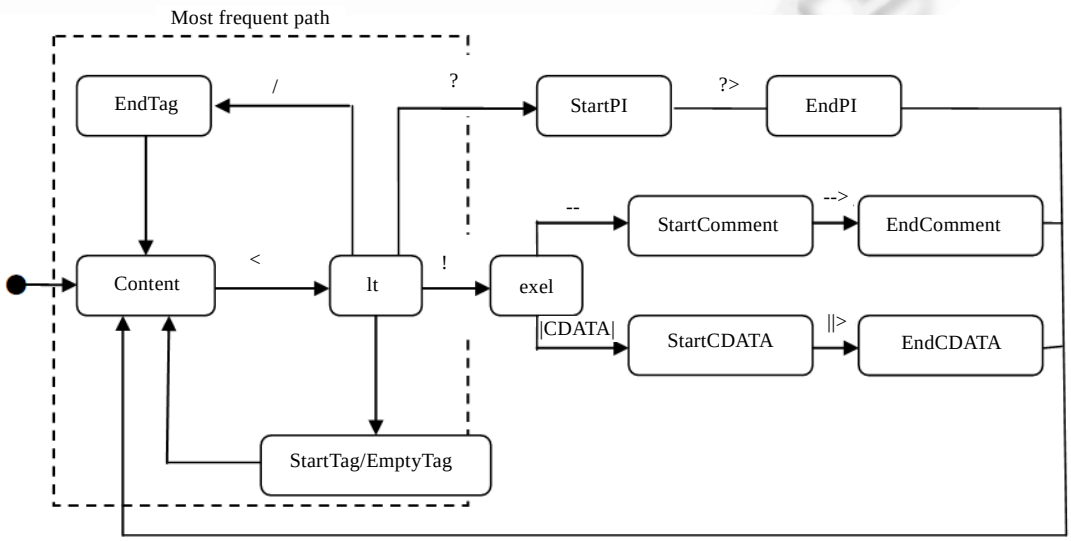

Fig.3 Events boundary marking process

图 3 事件边界识别流程

在图 3 中,因为每个 XML 标签都以"<"字符为开始字符,所以预处理过程从识别第 1 个"<"开始.在识别到

“<”后,根据紧跟在“<”之后的字符来判断 XML 标签的类型.如果紧跟着的是“/”或者“?”或者“!”,则做进一步特殊处理;否则,认为找到一个开始元素.如果紧跟的是“/”字符,则表示找到一个结束元素.以上过程结束后,预处理过程状态转为内容状态,然后重复上述过程.

在大多数情况下,“<”会被识别为开始元素或者结束元素的开始字符;然而在少数情况下,比如碰到评论、处理命令和字符数据段的时候,则不能简单寻找下一个“<”,因为“<”可能仅仅为这些特殊元素中的一个内容字符,比如下面的一个评论的例子.

<!--This element indicates whether age<40.-->

因此,遇到以上特殊情况时,需要识别特殊元素的类型并寻找该元素的结束标志.如果紧跟着的是“?”,则认为找到一个处理命令,识别过程往前搜索,一直搜索到“?>”字符串,则认为该处理命令结束;如果紧跟着的是“!”,则进一步识别是字符数据段或者评论,并往前搜索直到该字符数据段或评论结束.

预处理中,最频繁的识别路径是“<”被识别为开始元素或者结束元素的开始字符.XML 数据中,内容字符占大多数,因此预处理大部分时间处于内容状态.在内容状态下,预处理只需搜索第 1 个碰到的“<”字符,对搜索的每个字符,只需一次字符比较操作来判断该字符是否为“<”,因此预处理具有轻量级的特性.我们将标识事件边界的字符或者字符串称为 BCS(boundary character/string).

根据 BCS 标记的边界位置,XML 数据流被划分为多个数据块.每个数据块可能包含一个或多个事件.在事件解析阶段进行并行解析.

对于一个数据块,我们先开辟一个固定大小的缓冲区.缓冲区空间分成两个部分:第 1 部分大小为 $L1$,第 2 部分大小为 $L2$.当我们需要新的 XML 数据时,读入 $L2$ 大小的 XML 数据到缓冲区的第 2 部分.在事件划分阶段,当开始处理该数据块数据时,我们查找前一个数据块最后一个 BCS.如果前一数据块中没有 BCS,则继续往前查找,直至找到至少包含一个 BCS 的数据块.该 BCS 后的字符数据为一个新元素的内容.我们把前一个数据块的最后一个 BCS 及该 BCS 后面的字符数据拷贝至当前数据块缓冲区的第 1 部分.如果 $L1$ 小于这些字符数据的长度,则重新开辟一个缓冲区.数据拷贝的作用是使要处理的数据从一个事件的边界开始且数据在存储空间上是连续的,令实现时编程变得简洁.然后,我们从当前数据块缓冲区第 1 部分空间的第 1 个有效字符处进行事件划分处理.每识别一个 BCS,记录该 BCS 的类型、所在块号和块内相对位置信息.

2.3.2 SIMD 指令的使用

我们使用 SIMD(single instruction,multiple data)指令来加速预处理.本文中,我们使用英特尔 SSE 4.2 指令集.在 SSE 4.2 指令集中,有 4 条指令用来加速字符串和文本处理.这些指令可在文本片段上一次执行 256 个操作.每个文本片段可为 16 字节.下面举例说明我们如何使用这些指令.由于在预处理中我们多数时间识别一个字符是否为“<”,所以在预处理最开始或者处于内容状态时,我们使用 PCMESTRM 指令来定位从 XML 数据流当前位置开始的第 1 个“<”.当第 1 个“<”被识别后,如果紧跟着的是“!--”,我们可以用 PCMESTRM 指令来定位后面的第 1 个“-->”结束字符串.

2.4 事件解析阶段

事件解析阶段,事件解析模块产生子事件流.根据第 1 个 BCS 的类型分析得出两个相邻 BCS 之间的事件类型,见表 1.

Table 1 Events between neighboring BCSs

表 1 相邻 BCS 之间的事件

第1个BCS的类型	两个相邻BCS间的事件(*表示可能的事件)
StartTag	开始元素,属性(*),字符数据(*)
EndTag	结束元素,字符数据(*)
EndPI,或EndComment,或EndCDATA	字符数据(*)
StartPI	处理命令
StartComment	评论
StartCDATA	字符数据段

对于事件数据结构,当事件包含 XML 数据信息时,不需要开辟新的数据缓存再将 XML 数据从缓存块缓存拷贝至该缓存,而只需记录 XML 数据在缓存块中的开始和结束位置.

事件解析过程很大程度上跟传统 XML 解析过程一致,我们采用基于堆栈的解析处理.与传统 XML 解析过程是:当一个解析的数据块不完整时,如没有找到匹配的开始元素或结束元素,解析程序不认为解析出错,而是将不完整信息记录在事件数据结构中,以在后处理中解决.

当一个数据块被解析后,我们依次搜索子事件流中的未解决元素,将这些元素连接成一个未解决元素链表.进行后处理时,我们只需要检查未解决元素链表中的元素,这样使后处理的搜索时间大为减少.

2.5 后处理阶段

在后处理阶段,后处理模块对未解决链表中的每个项进行检查与处理.我们设计一个全局的包含未解决开始元素的堆栈(Global unresolved start element stack,简称 GUSES).对于一个未解决项,如果该项为未匹配项,则判断该项是否为结束元素,如果是,则从 GUSES 弹出一个开始元素项,与该结束元素进行匹配比较.如果该项的前缀未解决,我们将从 GUSES 中的顶部开始元素起向底部依次进行搜索和比较,直至该属性的前缀得到解决为止.如果该项为一个未解决的开始元素,创建一个包含相同信息的未解决项推入到 GUSES 中.为减少内存开辟开销,我们同时创建一个空闲元素链表(free element list,简称 FEL),从 GUSES 弹出的元素加入到 FEL 中,当 GUSES 需要加入一个新的项时,先从 FEL 中查看是否有空闲项,如果有,则 FEL 取一个空闲项使用.

当一个数据块的未解决项都被解决后,我们将前一个数据块子事件流与该数据块子事件流相连接.最后,当所有数据块处理完毕,我们认为 XML 文件是完整的,并得到一个完整的连续结果事件流.

2.6 解析线程的运行流程

在我们的设计中,线程以数据块为处理单元.每个数据块经过从第 1 阶段~第 3 阶段的依次处理.每个线程可以处理任意阶段的任务.一个数据块可能在不同阶段被不同的线程处理.因为数据块的运行时信息是关键信息,使用全局锁可以防止多个线程同时修改这些信息.我们对所有 XML 数据块建立一个双向链表,以先进先出(FIFO)模式检索该链表.线程运行时,依次检查链表中未处理完的数据块是否需要执行以下任务之一:数据载入、事件划分、事件解析和后处理,然后根据检查结果执行相应任务.线程运行循环流程具体描述如下:

循环开始:如果 XML 文件未解析完毕,则将当前数据块设置为数据块链表中第 1 个未解决的数据块;如果解析完毕,则结束循环.

步骤 1. 如果当前数据块的 XML 数据已经完成载入,转至步骤 2;如果当前数据块的 XML 数据没有完成载入且没有正在被别的线程载入,且如果该块为 XML 文件的第 1 个数据块或者该块的前一个数据块的数据已经完成载入,则执行该数据块的数据载入功能,完成后跳至循环开始处;如果当前数据块的 XML 数据没有完成载入且没有正在被别的线程载入,但该块不为 XML 文件的第 1 个数据块且该块的前一个数据块的数据没有完成载入,则跳至循环开始处;如果当前数据块的 XML 数据已经完成载入,向前搜索至链表中第 1 块未完成全部处理的数据块,跳至循环开始处.

步骤 2. 如果当前数据块的事件划分已经完成,转至步骤 3;如果当前数据块的事件划分没有完成且没有正在被别的线程执行,且如果该块为 XML 文件的第 1 个数据块或者该块的前一个数据块的数据已经完成事件划分,则对该数据块进行事件划分,完成后跳至循环开始处;否则,向前搜索至链表中第 1 块未完成全部处理的数据块,跳至循环开始处.

步骤 3. 如果当前数据块的事件解析已经完成,转至步骤 4;如果当前数据块的事件解析没有完成且没有正在被别的线程执行事件解析,则对该数据块进行事件解析,完成后跳至循环开始处;如果当前数据块的事件解析已经完成,或者当前数据块的事件解析没有完成且正在被别的线程执行事件解析,则搜索至链表中第 1 块未完成全部处理的数据块,跳至循环开始处.

步骤 4. 如果当前数据块的后处理已经完成,转至循环开始处;如果当前数据块的后处理没有完成且没有正在被别的线程执行,如果该块为 XML 文件的第 1 个数据块或者该块的前一个数据块的数据已经完成后处理,

则对该数据块进行后处理,完成后跳至循环开始处;如果当前数据块的后处理已经完成,或者当前数据块的后处理没有完成且正在被别的线程执行,向前搜索至链表中第 1 块未完成全部处理的数据块,跳至循环开始处.

该流程的设计使数据载入、事件划分、事件解析和后处理以流水线方式运行,并且在事件解析阶段,多个数据块的解析可以并行运行.因此,我们设计的是一种混合并行的方法.在数据载入阶段,将 XML 数据从 XML 文件读入 XML 数据块的缓冲区.如果数据载入被考虑作为 XML 解析程序整体性能的一部分,它作为一个单独的阶段加入流水线处理以提高解析性能.

2.7 内存和缓冲区管理

在本文中,解析程序最后产生 XML 文件的完整事件流,以在 XML 软件中进一步使用.对于内存和缓冲区管理,如果我们对每个 BCS 或事件都进行内存开辟的操作,在多线程环境下性能会变得很差.改进方案是:一次开辟一定数量的 BCS 或事件内存项,如果对于一个数据块一次开辟不够,则再次开辟一定数量的内存项,直至够用为止,这样使内存开辟的操作大为减少.尽管该方案可能会浪费一定内存资源,但是显著提高了程序的整体性能.

3 实验结果及分析

我们在基于 Intel Xeon X7560 CPU、频率为 2.27GHz 的 8 核 Linux 机器上测试解析程序.使用 GCC 4.4.4 编译器,编译选项为–O3.测试文件从 DBLP XML 测试文件库[17]获得.缓存块的第 1 部分 $L1$ 设置为 1KB.本文算法的实验结果与 Libxml2 SAX2 和 HPSAX[5]进行对比.

本文的特点在于实现事件级数据并行和流水线的混合并行处理,解析结果为事件流,简称本文方法为 HPES(hybrid method producing event stream).目前,我们检索到的类似研究为 HPSAX[5].目标是研究算法真正解析 XML 数据的时间,即 XML 数据在内存准备好以后至全部结果产生之间的时间间隔.为了测试文件在内存缓存之后在的解析速度,HPSAX[5]采取丢弃第 1 次测试结果、每个测试程序运行 5 次求平均值的方式.本文程序采取了和 HPSAX[5]同样的做法,并重复运行多次取平均值,以得到稳定的平均运行时间.

3.1 整体性能

首先测试本文方法和已有研究在整体性能上的比较.我们选择开源 XML 软件 Libxml2 SAX2 作对比,Libxml2 SAX2 为单线程程序.另外,产生类似事件流形式作为输出结果的 XML 并行解析算法研究工作为 HPSAX[5].在对比实验中,测试的 XML 文件大小为 32MB.数据块缓冲区第 2 部分 $L2$ 大小设置为 100KB.

我们测得 Libxml2 解析时间为 507ms(millisecond).在没有改进内存管理的情况下,本文的单线程解析程序比 Libxml2 SAX2 慢.但当改进内存管理后,本文的单线程解析程序的解析时间为 301ms.

然后,将本文设计方法与 HPSAX 作测试比较.

首先将 HPES 方法与 HPSAX 的解析时间作比较,如图 4 所示.在单线程情况下,HPSAX 方法解析需要 816ms;在 8 线程时,HPSAX 方法解析时间为 183ms,而 HPES 解析时间为 58ms.在 8 线程下,HPES 方法比 HPSAX 方法加速约 3 倍.

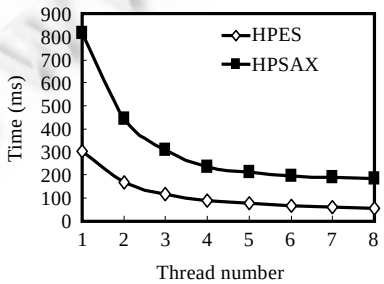

Fig.4 Time of HPES vs. HPSAX

图 4 HPES 与 HPSAX 解析时间比较

HPES 方法和 HPSAX 方法的加速比如图 5 所示.与单线程相比,HPES 方法在 8 线程下加速比达 5.05,而 HPSAX 方法的最大加速比为 4.46.在线程数小于 5 时,HPES 的加速比与 HPSAX 接近.当线程数大于 5 时,HPES 加速比大于 HPSAX 的加速比.HPES 比 HPSAX 方法具有更好的可扩展性.

HPES 方法和 HPSAX 方法的解析速度比较如图 6 所示.单线程解析 test.xml 时,HPES 方法解析速度为 106.31MB/s,而 HPSAX 的解析速度为 39.21MB/s.因为我们使用快速准确的预处理方法,而 HPSAX 使用复杂的 meta-DFA 方法进行预处理,HPES 方法在单线程下比 HPSAX 方法解析速度快很多.在 8 线程时,HPES 的解析速度达 551.72MB/s,而 HPSAX 的解析速度为 174.86MB/s.

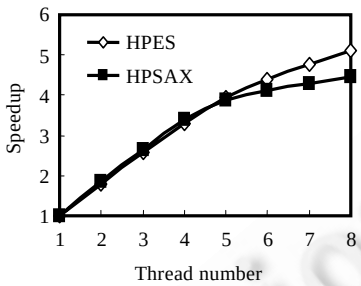

Fig.5 Speedup comparison of HPES vs. HPSAX

图 5 HPES 与 HPSAX 的加速比较

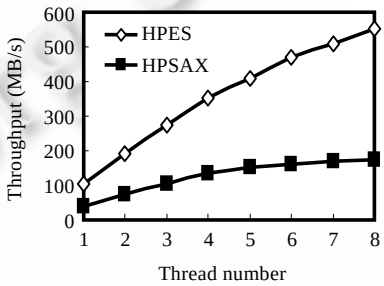

Fig.6 Throughput comparison of HPES vs. HPSAX

图 6 HPES 与 HPSAX 的解析速度比较

3.2 影响性能因素分析

根据文献[21]中的块同步并行计算模型(bulk synchronous parallel model,简称 BSP),本文方法与此模型相似.BSP 模型为

整体性能=计算时间+临界区开销+通信开销.

我们从这几方面对本文方法的性能进行测试与分析.

3.2.1 计算时间

我们设计了一个测试程序来测量每个阶段总共所需的时间,该包含以下几个线程:一个数据载入线程把 XML 文件读入数据块的缓冲区中;一个预处理线程进行事件划分;多个事件解析线程处理事件解析;一个后处理线程进行后处理.以上 4 种线程依次运行,当一种线程全部处理完后,后一种线程才启动运行.只有多个事件解析线程可以并行运行.我们以 32 MB XML 文件作为测试源.测量各线程运行时间如下:

数据载入线程运行时间为 40ms.预处理线程在不使用 SIMD 指令的情况下运行时间为 51ms,在使用 SIMD 指令的情况下运行时间为 25ms,使用 SIMD 指令使预处理时间减少了约一半.后处理线程运行时间为 8ms.

不同数目的事件解析线程的运行时间如图 7 所示,加速比如图 8 所示.单个事件解析线程运行时间为 205ms.当事件解析线程数目为 8 时,运行时间为 40ms.我们估计,在大于 8 核的机器上运行更多线程,运行时间还能继续减少.

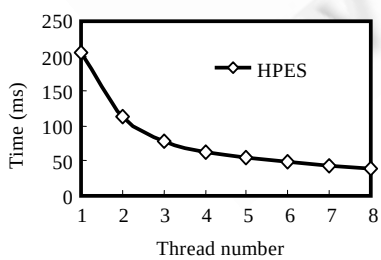

Fig.7 Time of parallel events parsing

图 7 并行解析线程运行时间

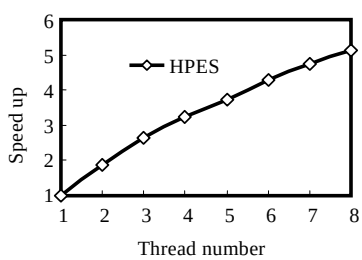

Fig.8 Speed up of parallel events parsing

图 8 并行解析线程加速

从以上测试来看,预处理和后处理阶段所需的时间占整体解析时间的比例较低,主要运算为事件解析阶段.事件解析阶段通过数据并行处理来降低所需解析时间.阶段级流水线方式使各阶段处理并行运行,可进一步减低整体解析时间.

3.2.2 临界区开销+通信开销

本文解析程序数据块缓存和事件流均为全局缓存数据结构,各线程通过共享缓存方式进行通信.

我们测试解析程序的同步开销,测量得到所有线程在临界区耗费的时间为 3ms.本文方法中,当一个线程离开临界区后开始执行某个解析阶段的任务时,该线程所需信息已经全部产生,因此在执行该阶段处理的过程中,它不需要与任何其他运行线程进行通信.线程执行某一阶段处理完毕后,产生的结果数据可以被该线程或其他线程在下一次进行某阶段的处理时直接使用.以上测试和分析表明,本文方法的同步和通信开销很低.

因为本文设计的方法各阶段处理可以以流水线方式运行,同步和通信开销很低,在理想状况下,解析程序运行时间应当接近各阶段的运行时间中的最长时间.所以我们估计,当有足够数量的核时,如果我们设计足够的线程数,可以使解析时间接近理想值.而目前我们测得的总体解析时间离该理想值还有一定差距,因此我们估计本文方法可扩展至多于 8 核的架构上.然而,由于预处理是线性处理过程,预处理时间已难以进一步减少,因此我们估计,当核的数目达到一定数目之后,本文方法难以实现更进一步的加速.

3.2.3 其他影响性能因素

这里,我们测试在不同 XML 文件和不同 *L*2 大小情况下 HPES 方法的性能.

我们选择 3 个不同的 XML 文件 test1.xml,test2.xml 和 test3.xml.test1.xml 均由小的简单元素组成,test2.xml 均由中等大小的普通元素组成,Test3.xml 均由大的较深嵌套层次的元素组成.每个文件在不同 *L*2 大小的情况下进行测试.测试中我们发现:如果 *L*2 太小,加速效果相对较差,只有在 *L*2 大于一定阈值后加速才比较明显;阈值如果太小,会使未解决的元素项较多,使后处理开销增加;同时,内存事件项的开辟次数增加,导致临界区处理时间增加,从而加速相对较差.我们设置阈值为 32KB.数据块大小在阈值以上的测试结果如图 9~图 11 所示.

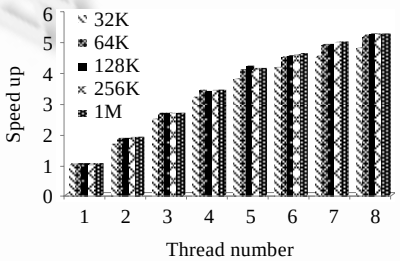

Fig.9 Speedup of HPES with test1.xml

图 9 对 test1.xml 文件 HPES 方法的加速

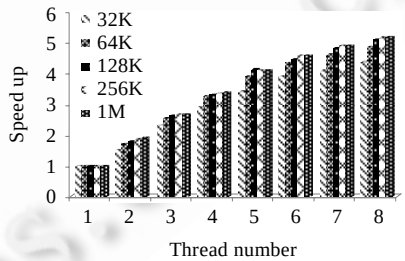

Fig.10 Speedup of HPES with test2.xml

图 10 对 test2.xml 文件 HPES 方法的加速

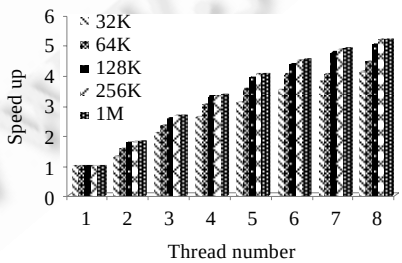

Fig.11 Speedup of HPES with test3.xml

图 11 对 test3.xml 文件 HPES 方法的加速

从图中可以看出,本文算法对于不同类型文件和数据块在阈值以上的不同大小的加速差距比较细微,趋于

基本一致,说明数据块大小在阈值以上的前提下,本文算法对于不同类型文件和不同数据块大小具有较好的适应性,以及良好的稳定性.

4 总结与展望

基于多核架构设计并行方法能够显著提高 XML 处理性能.在本文的方法中,我们提出一种混合并行 XML 解析方法,该方法由轻量级事件划分阶段、事件解析阶段和后处理阶段组成.在事件划分阶段,通过识别"<"进行事件划分.SIMD 指令用来加速预处理.然后在事件解析阶段并行解析多个数据块.该阶段未解决项的信息,如名字空间的解决或者开始元素和结束元素的匹配,被记录在未解决事件链表中.在后处理中检查和处理未解决事件链表.流水线技术用于阶段级处理.如果 XML 文件完整,我们将最后得到完整的结果事件流.本文使用了事件级数据并行技术和流水线并行技术,从整体上设计一种新的混合并行方法.实验结果表明,与以往的方法相比,本文的方法能够实现更快的解析速度、更高的加速比,且具有更好的可扩展性.

未来研究展望方面,可以进一步探索具有更好可扩展性的并行 XML 解析方法.

References:

[1] Lu W, Chiu K, Pan YF. A parallel approach to XML parsing. In: Proc. of the Grid 2006. 2006. 223−230. [doi: 10.1109/ICGRID.2006.311019]

[2] Pan YF, Zhang Y, Chiu K, Lu W. Parallel XML parsing using Meta-DFAs. In: Proc. of the 3rd IEEE Int'l Conf. on e-Science and Grid Computing. 2007. 237−244. [doi: 10.1109/E-SCIENCE.2007.55]

[3] Head MR, Govindaraju M. Approaching a parallelized XML parser optimized for multi-core processor. In: Proc. of the Workshop on Service-Oriented Computing Performance: Aspects, Issues, and Approaches. 2007. [doi: 10.1145/1272457.1272460]

[4] Cameron RD, Herdy KS, Lin D. High performance XML parsing using parallel bit stream technology. In: Proc. of the CASCON 2008. 2008. [doi: 10.1145/1463788.1463811]

[5] Pan YF, Zhang Y, Chiu K. Hybrid parallelism for XML SAX parsing. In: Proc. of the IEEE Int'l Conf. on Web Services (ICWS 2008). 2008. [doi: 10.1109/ICWS.2008.107]

[6] Li XS, Wang H, Liu TY, Li W. Key elements tracing method for parallel XML parsing in multi-core system. In: Proc. of the 2009 Int'l Conf. on Parallel and Distributed Computing, Applications and Technologies. 2009. 439−444. [doi: 10.1109/PDCAT.2009.64]

[7] Kostoulas MG, Matsa M, Mendelsoh N, Perkins E, Heifets A, Mercaldi M. XML screamer: An integrated approach to high performance XML parsing, validataion and deseriallization. In: Proc. of the WWW 2006. 2006. 93−102. [doi: 10.1145/1135777.1135796]

[8] Takage T, Miyashita H, Suzumura T, Tatsubori M. An adaptive, fast and safe XML parser based on byte sequence memorization. In: Proc. of the WWW 2005. 2005. 692−701. [doi: 10.1145/1060745.1060845]

[9] Zhang W, van Engelen RA. TDX: A high-performance table-driven XML parser. In: Proc. of the ACM SE 2006. 2006. 726−731. [doi: 10.1145/1185448.1185606]

[10] Zhang W, van Engelen RA. High-Performace XML parsing and validation with permutation phase grammer parsers. In: Proc. of the IEEE Int'l Conf. on Web Services (ICWS 2008). 2008. 286−294. [doi: 10.1109/ICWS.2008.101]

[11] Tang N, Wang G, Yu JX, Wong KF, Yu G. Win: An efficient data placement strategy for parallel XML database. In: Proc. of the 11th Int'l Conf. on Parallel and Distributed Systems (ICPADS 2005). 2005. 349−355. [doi: 10.1109/ICPADS.2005.298]

[12] IBM. Datapower. http://www-01.ibm.com/software/integration/datapower/

[13] van Lunteren J, Engbersen T, Bostian J, Carey B, Larsson C. XML accelerator engine. In: Proc. of the 1st Int'l Workshop on High Performance XML Processing. 2004.

[14] Dai ZF, Ni N, Zhu JW. A 1 cycle-per-byte XML parsing accelerator. In: Proc. of the 18th Annual ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays. 2010. [doi: 10.1145/1723112.1723148]

[15] El-Hassan F, Peterkin R, Abou-Gabal M, Ionescu D. A high-performance architecture of an XML processor for SIP-based presence. In: Proc. of the 6th Int'l Conf. on Information Technology: New Generations (ITNG 2009). 2009. 90−95. [doi: 10.1109/ITNG.2009.50]

[16] Intel® XML software suite. http://www.softmartonline.com/intel/XSS/xml_products.html

[17] DBLP XML records. http://dblp.uni-trier.de/xml/

[18] SAX. http://www.saxproject.org

[19] W3C. Document object model (DOM). http://www.w3.org/DOM

[20] LibXML2. http//www.xmlsoft.org/

[21] Valiant LGA. A bridging model for parallel computation. Communications of the ACM, 1990,33(8):103−111. [doi: 10.1145/79173.79181]

方跃坚(1981－),男,湖南长沙人,博士生,主要研究领域为高性能 XML 处理,并行计算,嵌入式系统.

E-mail: fangyuejian@163.com

余枝强(1978－),男,硕士,主要研究领域为浏览器/HTML5 相关技术研发,高性能 XML 处理,并行计算.

E-mail: zhiqiang.yu@intel.com

翟磊(1981－),男,高级工程师,主要研究领域为高性能 XML 处理,云计算安全解决方案.

E-mail: lei.l.zhai@gmail.com

吴中海(1968－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为情景感知服务,云服务与安全,嵌入式软件与系统,多媒体与人机交互.

E-mail: wuzh@pku.edu.cn