软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(6):1376−1389 [doi: 10.3724/SP.J.1001.2013.04280]

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

面向多核的并行离散事件仿真服务优化*

唐文杰, 姚益平

(国防科学技术大学 计算机学院,湖南 长沙 410073)

通讯作者: 唐文杰, E-mail: tangwenjie@nudt.edu.cn

摘 要: 处理器发展已进入多核时代.现有并行仿真内核常常以多进程方式使用多核资源,存在较大的同步和通信开销,无法深入发掘多核处理器潜能.基于层次化并行仿真内核(HPSK)模型,重点对时间管理服务和事件管理服务进行优化,支持多线程架构下进行高效能仿真:(1) 基于混合时间推进模式,提出最小发送时戳(EETS)计算协议,可根据仿真应用特点灵活配置为异步 EETS 算法以支持高效的全局同步,并证明了计算协议的正确性;(2) 基于并行仿真事件交互的特点,提出无锁创建、异步提交和指针通信的事件管理算法,最小化线程之间的锁开销并减少了内存的消耗.实验结果表明,采用上述优化服务的 HPSK 能够在各种条件下获得很好的加速效果.

关键词: 多核处理器;并行离散事件仿真;多线程;时间管理;事件管理

中图法分类号: TP303 **文献标识码**: A

中文引用格式: 唐文杰,姚益平.面向多核的并行离散事件仿真服务优化.软件学报,2013,24(6):1376−1389. http://www.jos.org.cn/1000-9825/4280.htm

英文引用格式: Tang WJ, Yao YP. Multicore-Oriented service optimization of parallel discrete event simulation. Ruan Jian Xue Bao/Journal of Software, 2013,24(6):1376−1389 (in Chinese). http://www.jos.org.cn/1000-9825/4280.htm

Multicore-Oriented Service Optimization of Parallel Discrete Event Simulation

TANG Wen-Jie, YAO Yi-Ping

(College of Computer, National University of Defense Technology, Changsha 410073, China)

Corresponding author: TANG Wen-Jie, E-mail: tangwenjie@nudt.edu.cn

Abstract: The development of CPU has made its way into the era of multicore. The current parallel simulation kernel utilizes multicore resource by a multiprocess, which leads to inefficiency in synchronization and communication. This study has optimized two services based on hierarchical parallel simulation kernel (HPSK) model to support high performance simulation in multithread paradigm. First, the paper proposes a protocol of EETS computation based on hybrid time management, which can be configured flexibly as asynchronous EETS algorithm according to application's characteristics. Second, the study proposes an event management algorithm based on characteristics of events interaction, which can create events lock-free, commits events asynchronously, and transfers events based on pointers, to eliminate the overhead of locks and to reduce the usage of memory. Experimental results from phold show that the optimized HPSK works well on different conditions.

Key words: multicore; parallel discrete event simulation; multithread; time management; event management

由于系统能耗、存储性能和指令集并行度等多方面限制,通过提高处理器主频来获取更高性能的方法不再可行,单芯片集成多个处理核心成为处理器发展的新趋势.多核时代的摩尔定律预测,芯片内部集成的核数将呈指数增长,在不久的将来,单个芯片内部就能够集成上百的处理器核[1].处理器微体系结构发展方式的转变,极大地影响了软件设计的思路.软件依靠处理器计算能力提升而直接获得更高性能已成为历史.并行处理成为程序设计时需要考虑的主要问题,这是继面向对象技术之后,软件开发方式的深刻变革[2].

* 基金项目: 国家自然科学基金(61170048); 高等学校博士学科点专项科研基金(201243070017)
收稿时间: 2011-07-13; 修改时间: 2012-04-17; 定稿时间: 2012-06-25

建模与仿真是设计和研究复杂系统的重要工具[3].随着仿真应用在规模上不断扩展、粒度上不断细化和模型复杂度不断提高,使得仿真很难在单机平台完成.并行与分布式仿真是解决这一问题的唯一手段[4,5].作为支撑仿真运行的基础平台,现有的并行仿真内核可运行在多核处理器上,但其设计基础是以单核处理器为构件的紧耦合巨型机或松耦合机群.这种将多核处理器完全等同于多个处理器的方式忽略了多核体系结构的独有特点,难以合理利用和分配多核资源以获得最佳性能.

针对这一问题,课题组初步实现了一个基于逻辑进程范型的多线程层次化并行仿真模型[6],但其运行细粒度仿真应用时存在可扩展性弱、加速比低等问题.本文的主要贡献在于,针对多核处理器和并行仿真应用特点,基于HPSK模型对时间管理和事件管理服务进行优化:(1) 基于混合时间推进模式,提出EETS计算协议,可根据仿真应用特点灵活配置为异步EETS算法以支持高效的全局同步;(2) 基于并行仿真事件交互的特点,提出无锁创建、异步提交和指针通信的事件管理算法,最小化线程之间的锁开销和减少了内存的消耗.

本文第1节介绍相关研究工作.第2节介绍并行离散事件仿真及逻辑进程范型的形式化定义,提出面向多核的层次化仿真内核模型及相关的仿真服务.第3节针对时间管理和事件管理两种服务,分别提出相应的优化方法.第4节是实验与分析.第5节对本文工作进行总结和展望.

1 相关研究工作

物理系统可以视为一组物理进程及其之间的交互.在离散事件仿真中,通常使用一个逻辑进程(logical process,简称LP)来模拟一个物理进程,而物理进程之间的交互则通过在对应逻辑进程之间交换带时戳的事件来表示.每个LP的任务就是按照时戳序进行执行事件,事件的执行会改变LP的某些状态并产生一些新的事件.并行离散事件仿真通过将多个LP分布到不同的计算节点(或进程)并行推进以减少执行时间,使得研究更大规模更细粒度的复杂系统成为可能.并行离散事件仿真通常也被称为并行仿真.

如何构建高效的并行仿真平台一直是过去20年仿真领域的研究热点.由于多核出现的时间尚短,大部分并行仿真平台在设计的时候没有考虑多核特性.例如:美国喷气推进实验室开发的SPEEDES[6]、乔治亚理工学院的GTW[7],ROSS[8],PARSEC[9]和musik[10]等等;国防科学技术大学计算机学院研究的YH-SUPE[11],这些并行仿真平台可以运行在多核处理器上,但其设计的目标平台是SMP和集群,大都是以多进程架构实现并行(GTW的SUN版本基于多线程架构,但仅支持SMP).通过将仿真系统的多个逻辑进程分配到各个进程以降低执行时间,进程之间通过共享内存,MPI或TCP等方式进行消息通信.这种处理方式将多核处理器等同于多个处理器,忽略了多核的独有特点,存在较大的同步和通信开销.另外,多核化作为一种发展趋势,还必须以可扩展的眼光重新审视软件设计模型.相比于多进程模型,多线程模型具有如下优势:

(1) 在并行离散事件仿真中,逻辑进程之间存在大量消息交互,涉及事件的发送、撤销和回滚.同一进程内部的线程共享地址空间,可以基于指针完成事件交互,通信效率比进程之间通信更高;

(2) 随着处理器核数的增多,每个处理核心拥有的内存相对减少.当事件需要在两个进程间传递时,每个进程都要在内存中为事件分配空间,而多线程方式则避免了这种额外的内存开销.就事件的储存而言,多进程模型的内存消耗量至少是多线程模型的2倍;

(3) 负载平衡是影响并行程序的关键因素,而实现高效的负载平衡要求仿真平台提供低开销的任务迁移机制.在并行离散事件仿真中,迁移的基本单位是逻辑进程,涉及到状态变量、事件队列等诸多对象.在多个进程之间迁移这些对象的开销较大,而线程之间能够通过共享地址空间,以很小的开销实现迁移.

当然,相比于多进程模型,多线程模型需要更多的逻辑控制来保证程序的正确性,而且过多的锁操作会严重影响程序的性能.必须针对并行离散事件仿真的特点,充分利用多线程架构带来的好处,尽可能消除锁开销,以构建高效能的多线程并行离散事件仿真平台.

近年来,也出现了一些针对多核的工作,如苏年乐等人[12]的工作,通过将并行仿真的思想引入桌面平台,以利用多核处理器获得仿真加速.但他们同样采用多进程模型,与传统并行仿真平台没有太大区别.本课题组的陈

莉丽等人[13]提出了一种基于分布式队列的全局调度机制,从负载平衡的角度来提高多核处理器上的仿真平台性能.但为实现高负载均衡能力可能导致逻辑进程的频繁迁移,产生大量的 Cache 失效.WarpIV[14]的 HyperWarpSpeed 技术利用多核并发处理事件分支,加速 Monte Carlo 多样本仿真.但优化的效果与应用紧密相关,不适合通用仿真支撑环境.一个更相关的工作是由 Miller[15]完成的——WARPED[16]的线程化版本.ThreadWarped 采用 Master-Worker 模式,一个管理线程负责分配事件到多个工作线程,通过并行处理多个事件获得加速比.为确保时序关系的正确性,工作线程需要在每轮事件执行后进行同步.如果事件粒度不均匀,总存在处理核心空闲.而本文工作采用对称的结构,每个处理核心都要进行逻辑进程调度、事件处理等工作,并且只进行必要的非阻塞协同.

综上所述,针对多核的并行仿真平台研究还处于探索阶段.考虑到并行仿真平台作为一种基础平台,在发掘底层硬件特性时,需向上支持仿真应用的开发与运行.而且经过多年发展,并行仿真领域也形成了一些成熟的技术.因此,支持原有的仿真应用开发方式和最大可能继承原有技术十分必要.基于传统的逻辑进程范型,针对多核进行仿真服务优化研究是一种比较合适的选择.

2 面向多核的并行离散事件仿真服务

2.1 并行离散事件仿真及LP范型的形式化定义

并行离散事件仿真的核心问题是如何确保所有 LP 按照时戳顺序来处理事件.目前主要有两类时间管理协议解决这一问题:保守协议和乐观协议.在保守协议下,只有在确保不会违反时戳序的条件下,事件才能被执行.对于那些具有良好并行性的仿真应用,保守协议能够很好地工作.但是保守协议过于严格,限制了某些并不会违反时戳序的事件执行,造成处理器的空闲等待从而影响效率;乐观协议则放松了执行限制,LP 按当前可见的时戳顺序执行事件.当时戳乱序确实发生后,通过回滚等方法来修复错误.相比于保守协议,乐观协议更能发掘仿真应用潜在的并行性,但如果回滚开销过大,也会影响性能甚至得不偿失.为方便描述,下面分别给出仿真时间、仿真事件(消息)和 LP 范型的形式化定义.

定义 1(仿真时间). 仿真时间 T 定义在 $R_0^+ \times N$ 上,即 $t=(\tau,c),\tau \in R_0^+,c \in N$.对 T 定义两种关系,<和=,分别为:

1) $(\tau_1,c_1)<(\tau_2,c_2) \Leftrightarrow (\tau_1<\tau_2) \vee (\tau_1=\tau_2 \wedge c_1<c_2)$;

2) $(\tau_1,c_1)=(\tau_2,c_2) \Leftrightarrow (\tau_1=\tau_2) \wedge (c_1=c_2)$.

仿真时间是仿真的基本概念,用于区分事件的先后顺序,其直观理解是:第 1 位对应物理系统的物理时间,第 2 位用于区分同一物理时间下具有因果关系的事件.若没有特别指出,本文中时戳就是指仿真时间.

定义 2(仿真事件). LP 执行任务的基本单元.LP 之间进行事件传递时被称为发消息,下文中不加区分地使用消息和事件.

- *ts*(*e*):事件的时戳,即事件执行的仿真时间;
- *sign*(*e*):表示事件的正负,反事件是在 LP 回滚时产生的、用于取消因错误执行所产生的事件.

定义 3(逻辑进程). 逻辑进程(logical process,简称 LP)定义为七元组结构 $LP=\langle s,f,r,PEL,FEL,AntiE,seq\rangle$:

- *FEL*:按时戳升序排列的事件链表,用于记录未处理事件;
- *PEL*:按时戳升序排列的事件链表,用于记录已处理事件;
- *AntiE*:用于缓存先于正消息到达的反消息;
- $s \in S$:LP 的当前状态,*ts*(s)表示该状态的仿真时间;
- $seq \in SEQ$:按时戳升序排列的检查点序列,用于记录 LP 在各个历史时间点的状态和发送的事件集合,乐观 LP 可利用检查点序列恢复到最近的正确状态;对于每个检查点 $se=(s,\theta) \in seq, s \in S, \theta \in 2^E$;
- $f: S \times E \to S \times 2^E$:事件处理函数,修改 LP 的状态,并产生一组新的事件(也可以产生 0 个事件);
- $r: S \times E \times SEQ \to S \times 2^E \times SEQ$:回滚处理函数,将 LP 的状态恢复到输入事件的时戳之前,并把所有错序执行事件对应的反事件发送出去.

图 1 描述了 LP 执行的处理流程,每次从 *FEL* 中取出下一事件,根据事件类型和时戳来进行相应的处理.由于保守 LP 不会遇到落伍消息和反消息,不需要对历史状态进行保存,仅仅执行第 4 行、第 5 行.需要说明的是,如果 *AntiE* 不为空,每个发送到 LP 的消息都必须同 *AntiE* 中的反事件进行匹配:如果匹配成功,互相抵消;若不成功,则加入 LP 的 *FEL* 中.从算法可以看出,为保证 LP 的顺利执行,还需要底层提供一些基础服务,如 *send*()发送消息、全局同步等等.

```
1.  LP 从 FEL 中取出下一事件 e_next
2.  if sign(e_next)=1, then                          //正事件且非落伍
3.     if ts(e_next)>ts(s_now), then
4.        (s_next,E_next):=f(s_now,e_next)
5.        s_now:=s_next, send(E_next)
6.        seq:=seq+{(s_now,E_next)}, PEL:=PEL+{e_next}
7.     else                                          //落伍的正事件
8.        (s_back,E_anti,seq_back):=r(s_now,e_next,seq_now)
9.        s_now:=s_back, send(E_anti)
10.       seq_now:=seq_back, PEL:=PEL−E(ts(e)>ts(e_next))
11.    endif
12. else
13.    if  e_next^-1 ∈ PEL                           //反事件,对应正事件已被执行
14.       (s_back,E_anti,seq_back):=r(s_now,e_next,seq_now)
15.       s_now:=s_back, send(E_anti)
16.       seq_now:=seq_back, PEL:=PEL−E(ts(e)>ts(e_next))
17.    else                                          //反事件,对应正事件未被执行
18.       AntiE:=AntiE+{e_next}
19.    endif
20. endif
```

Fig.1 Procedure of LP's event execution

图 1 LP 执行事件的处理流程

2.2 并行仿真服务和面向多核的并行仿真内核模型

并行仿真内核为 LP 提供一系列服务,以支持 LP 之间的通信和协同,从而确保整个并行仿真的正确和高效.这些服务从下到上分为 3 类,包括基础服务、PDES 服务和优化扩展服务:

(1) 基础服务:搭建并行系统的基本架构,支持一群实体(LP)进行消息交互,包括命名服务、事件管理服务;

(2) 并行离散事件仿真服务:搭建并行离散事件仿真框架,包括时间管理服务、回滚服务等;

(3) 优化扩展服务:针对应用特点提供性能优化,包括动态迁移服务、常用科学计算算法服务等.

为了充分利用多核 CPU 资源,通常采用多线程方法分担计算负载,以优化软件性能.文献[17]中提出了一种层次化并行仿真内核模型(hierarchical parallel simulation kernel,简称 HPSK),该模型针对多核集群计算节点内外交互能力的差异,采用多进程/多线程混合的平台架构.在计算节点之间、仿真内核之间以多进程方式通信与协同;计算节点内部,则采用多线程方式优化通信,并透明地实现多核并行化,如图 2 所示.这样,从系统的角度看,HPSK 同现有仿真平台完全一样;而以单个节点的角度看,HPSK 将多个仿真调度处理核心集成于一个进程内,分为两层:第 1 层称为进程核(ProcessKernel),负责控制第 2 层所有的线程核(ThreadKernel)的推进,包括产生、初始化、启动、停止等功能.进程核启动后,不再占用任何计算资源,仅仅提供一些全局变量供同步使用.当所有线程核执行结束后,CPU 执行权交还进程核,由其完成善后工作;第 2 层由一组线程核组成,每个线程核可视为一个简化版的仿真内核,负责 LP 的调度、事件的执行与发送等.每个线程核与操作系统线程一一对应,最大值可设定为 CPU 的处理核数.为了支持节点间通信,一组通信逻辑进程(communication logical process,简称 CLP)以代理方式负责与对应节点的通信.CLP 之所以被称为逻辑进程,是因为可以将 CLP 已发送事件列表和待发送事件列表映射为 LP 的 PEL 和 FEL,事件执行定义为发送到目标 LP 的事件.这样,CLP 被纳入到 LP 范型的框架内,节点间消息发送还可以采用乐观或保守方式控制.这组 CLP 被统一放置在 0 号线程核上,所有节点间消息被发送到 0 号线程核转发,节点间通信从逻辑上转化为线程核间或核内通信.

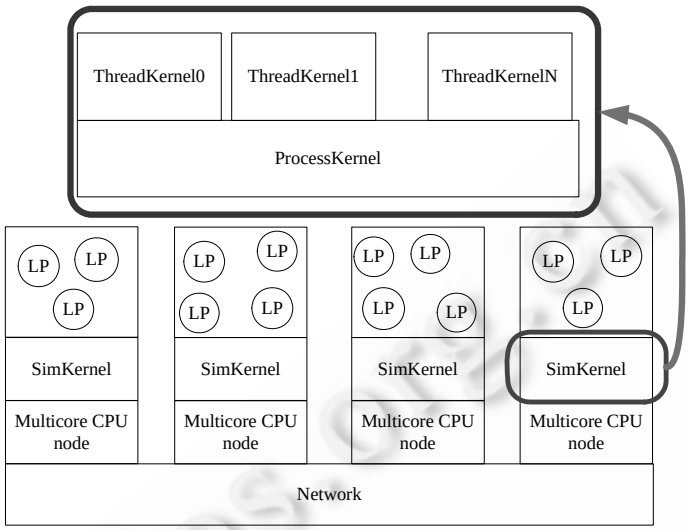

Fig.2 Architecture of hierarchical parallel simulation kernel model

图 2 层次化仿真内核模型体系结构

3 面向多核的并行仿真服务优化

3.1 时间管理服务优化

时间管理是并行离散事件仿真的核心服务.在多核时代,处理器的并行度将持续增长,但每个处理核心所拥有的内存可能减小.采用乐观与保守混合的时间管理协议,可综合利用乐观协议并行度高、保守协议所需内存小的优点.因此,本文采用混合时间管理协议,参考 Jha 等人提出的框架[18],将时间管理服务分为全局控制机制和局部控制机制两个部分.其中,局部控制机制负责 LP 调度,全局控制机制则负责全局时间同步.

LP 调度是指在 LP 之间合理地分配 CPU,以保证仿真快速正确地推进.每次循环开始时,线程核需要选择一个 LP,并为其设定可推进的时间范围,然后将 CPU 执行权交与 LP 进行仿真推进.本文采用的调度算法同文献[10]中类似.所不同的是,由于 HPSK 采用的异步消息发送方式(详见后文第 3.2 节),可能有消息存在于缓存区中,需要清空缓存区中的消息.调度算法的可分为如下两个阶段:

(1) 保守阶段:线程核中存在可推进的保守 LP.从中选择下一事件具有最小时戳的保守 LP,将其推进到设定时限之后;

(2) 乐观阶段:线程核中不存在可推进的保守 LP.选择下一事件具有最小时戳的乐观 LP,将其推进到设定时限之后.

这种调度方式以 LP 为单位,通过最大化每次 LP 推进的范围,减少 LP 切换的次数,减小开销.需要说明的是,调度算法中提到了两个设定时限,前者是当前保守 LP 可推进的时戳上限,与下文的全局时间同步紧密相关;而后者可根据应用的具体情况设定,目前默认为线程核中其他 LP 的最小下一事件时戳.

全局时间同步的具体工作是计算所有逻辑进程的未来可能处理事件的最小时戳,这里记为(EIT).对于保守 LP 来说,该值作为 LBTS 使用,定义了目前可安全执行(不会导致回滚)事件的时戳上限;而对于乐观 LP 而言,该值作为 GVT 使用,是提交事件,完成内存释放和 I/O 交互的时戳下限.计算 EIT 时,可以选用已有 LBTS 栅栏算法和 GVT 算法[19−22].但是无论哪种算法,都需要参与仿真的进程提交其最小发送时戳(EETS).在多进程仿真内核模型中,每个进程内部只有一个调度中心,确定 EETS 十分简单.但 HPSK 中存在多个调度中心,如何在不干扰线程核正常推进的前提下计算 EETS 值相对复杂.从表面上看,通过移植基于共享内存模型的 GVT 算法[22]能够解决这一问题.算法通过一个全局变量 *GVTFlag* 来控制同步,由各个处理器异步地检测 *GVTFlag* 状态并参与同步

值计算.但算法是面向纯粹乐观模式的,不支持混合时间推进模式;而且为了最优化仿真效率,通常要求在不同时机发起全局时间同步计算,而单个全局变量难以满足不同的同步需求;第三,算法没有考虑到节点间通信对全局同步值的影响.针对这些问题,本文提出了一种计算协议,可被灵活配置成EETS算法,支持混合状态的线程核,且能在不阻塞线程核正常推进的情况下计算EETS,如图3所示.

变量定义

$tTTS_i$=min{$ts(e)|e\in TKFEL_i$ 或 $e\in EB_i$}:线程核 i 的线程时戳,用于记录当前线程核中事件时戳的最小值;

$TKFEL$ 是线程核上所有LP的 FEL 集合,EB 是缓存区(包括事件和反事件)

$tOETS_i$:线程核 i 的局部变量,用于记录所有发送到计算态线程核事件的时戳最小值

$tEETS_i$:线程核 i 的最早发送事件时戳

$EETS_{ag}$:算法计算得到的进程核最早发送事件时戳

$EETS(wt)$:墙钟时间为 wt,通过系统快照得到的最早发送事件时戳(真实的EETS)

线程核的计算协议

1. 如果线程核 i 需要计算 $EETS$,提交 $tEETS_i$=min{$tTTS_i$,$tOETS_i$},更新 $tOETS_i$=$SimTime$::$MaxTime$;
2. 当线程核 i 发送事件 e 时,如果目标线程核 j 已经提交了 $tEETS_j$ 且 $ts(e)<tOETS_j$,则更新 $tOETS_i=ts(e)$;
3. 最后一个提交 $tEETS_i$ 的线程核 i 负责计算 $EETS_{ag}$,$EETS_{ag}$=min{$tEETS_i$};
4. 如果存在多个进程进行仿真,由0号线程核最后提交 $tEETS_0$.

Fig.3 Protocol of EETS computation

图3 EETS计算协议

只要保证进程中的所有线程核都提交了,$tEETS$ 就能得到可接受的EETS值.协议为EETS计算提供了一个灵活的框架,使得各个线程核可以根据实时需求合理地选择参与 EETS 计算的时机.假设线程核在墙钟时间 wt_1~wt_2 之间依次提交 $tEETS$,并由最后提交的线程核负责统计最终的进程 $EETS_{ag}$ 值,$EETS_{ag}$ 与真实的EETS的关系可由定理1和定理2得出.线程核提交 $tEETS$ 的墙钟时间构成一个截断,如图4所示.截断之前为预备状态,截断之后为计算状态.因此,可以将事件的发送接收方状态分为4类:E_1:预备到计算;E_2:预备到预备;E_3:计算到预备;E_4:计算到计算.

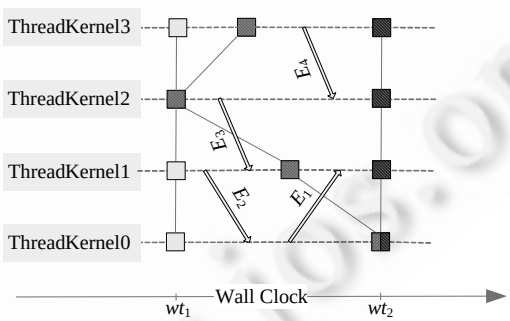

Fig.4 Four types of events

图4 4种不同事件类型

定理1. 如果所有线程核都参与了EETS计算,而且在计算过程中没有收到来自其他进程的消息,那么,按基本算法规则1~规则3计算得到的EETS值,满足 $EETS(wt_1)\leqslant EETS_{ag}\leqslant EETS(wt_2)$.

证明:

(1) 首先证明 $EETS_{ag}\leqslant EETS(wt_2)$

假设 $e_1\in E_1$ 是满足 $ts(e_1)$=min$_i${$tOETS_i$}的事件,j 是满足 $tTTS_j$=min$_i${$tTTS_i$}的线程核,

那么 $EETS_{ag}$=min{$ts(e_1)$,$tTTS_j$}.

(1.1) 如果 $ts(e_1)\leqslant tTTS_j$,往证存在 k,使得 $ts(e_1)\leqslant tTTS_k(wt_2)$;

若不然,则存在 $e_2\in E_4$,满足 $ts(e_1)>ts(e_2)$.

可以分为两种情况分析:

情况 I. e_2 有一个祖先事件 $e_3 \in E_1, ts(e_3)<ts(e_2)$.根据 e_1 的定义,$ts(e_1) \leqslant ts(e_3)$.所以 $ts(e_1) \leqslant ts(e_2)$,与 $ts(e_1)>ts(e_2)$矛盾;

情况 II. e_2 不存在属于 E_1 的祖先事件 e_3,则必有一个祖先事件 e_4, $e_4 \in TKFEL_k^{ag}$ 或 $e_4 \in EB_k^{ag}$.所以 $ts(e_4)<ts(e_2)$,根据 $tTTS_k$ 的定义,有 $tTTS_j \leqslant tTTS_k \leqslant ts(e_4)$.那么根据情况(1.1)的假设,有 $ts(e_1) \leqslant ts(e_4)<ts(e_2)$,矛盾.

所以,如果 $ts(e_1) \leqslant tTTS_j$,存在 k,满足 $ts(e_1) \leqslant tTTS_K(wt_2)$;

(1.2) 如果 $ts(e_1)>tTTS_j$,往证存在 k,使得 $tTTS_j \leqslant tTTS_k(wt_2)$.

- 如果 $tTTS_j \leqslant tTTS_j(wt_2)$,结论显然成立;
- 若 $tTTS_j>tTTS_j(wt_2)$,则存在事件 $e_5 \in E_1 \cup E_4$,满足 $ts(e_5) \leqslant tTTS_j$,在线程核 j 提交 $tEETS_j$ 之后收到:

  如果 $e_5 \in E_1$,则 $ts(e_1) \leqslant ts(e_5) \leqslant tTTS_j$,和 $ts(e_1)>tTTS_j$ 矛盾;

  如果 $e_5 \in E_4$,同样可分为两种情况分析:

  情况 I. e_5 有一个祖先事件 $e_6 \in E_1, ts(e_1) \leqslant ts(e_6) \leqslant ts(e_5) \leqslant tTTS_j$,和 $ts(e_1)>tTTS_j$ 矛盾;

  情况 II. e_5 不存在属于 E_1 的祖先事件,则必有一个祖先事件 e_7, $e_7 \in TKFEL_j^{ag}$ 或 $e_7 \in EB_j^{ag}$,所以 $ts(e_7)<ts(e_5)$.根据 $tTTS_j$ 的定义,有 $tTTS_j \leqslant ts(e_7)<ts(e_5)$,和 $ts(e_5) \leqslant tTTS_j$ 矛盾.

因此,如果 $ts(e_1)>tTTS_j$,存在 k,使得 $tTTS_j \leqslant tTTS_k(wt_2)$.

综合情况(1.1)和情况(1.2)

$$EETS_{ag}=\min_i\{tEETS_i\}=\min_i\{\min\{tTTS_i, tOETS_i\}\} \leqslant \min_i\{tEETS_k(wt_2)\} \leqslant EETS(wt_2).$$

(2) 然后证明 $EETS(wt_1) \leqslant EETS_{ag}$

根据定义,显然有,对于任意 i,$EETS(wt_1) \leqslant tTTS_i$.

取前文定义的 e_1,如果产生 e_1 的墙钟时间大于 wt_1,必有 $EETS(wt_1)<ts(e_1)$;如果产生 e_1 的墙钟时间小于 wt_1,但因为发送 e_1 的墙钟时间大于 wt_1,所以 e_1 的父亲事件 e_8 在 wt_1 时正在执行(即未完成).

所以,$EETS(wt_1) \leqslant ts(e_8)<ts(e_1)=\min_i\{tOETS_i\}$,有 $EETS(wt_1) \leqslant EETS_{ag}$.

综上所述,$EETS(wt_1) \leqslant EETS_{ag} \leqslant EETS(wt_2)$. □

如果在计算 EETS 的过程中收到了来自其他进程的远程消息,设 e^r 在这类事件中具有最小时戳,如果 $ts(e^r)>EETS(wt_1)$,那么定理 1 依然成立.而如果 $ts(e^r) \leqslant EETS(wt_1)$,则有:

定理 2. 如果所有线程核都参与了 *EETS* 计算,且 $ts(e^r) \leqslant EETS(wt_1)$,那么按基本算法规则 1~规则 4 计算得到的 *EETS* 值,满足 $ts(e^r) \leqslant EETS_{ag} \leqslant EETS(wt_2)$.

证明:根据 e^r 的定义,又因为 $ts(e^r) \leqslant EETS(wt_1)$,显然有 $ts(e^r) \leqslant EETS_{ag}$ 和 $ts(e^r) \leqslant EETS(wt_2)$.

按照计算协议 4,由线程核 0 最后提交 *tEETS*,此时墙钟为 wt_2.e^r 或被相应的 CLP 进行转发,或被缓存在 CLP 的 *TKFEL* 中.不妨设 $dest(e^r)=j$,那么:

- 如果被转发,则 $e^r \in E_1 \cup E_2$:
  - 情况 I. $e^r \in E_1$,根据规则 2,有 $EETS_{ag} \leqslant ts(e^r)$,所以 $ts(e^r)=EETS_{ag}$,那么 $ts(e^r)=EETS_{ag} \leqslant EETS(wt_2)$;
  - 情况 II. $e^r \in E_2$:
    - 若 e^r 已执行,$tTTS_j \geqslant ts(e^r)$,所以 $ts(e^r) \leqslant EETS_{ag}$;
    - 若 e^r 未执行,$tTTS_j=ts(e^r)$,同样有 $ts(e^r) \leqslant EETS_{ag}$;

  所以,$ts(e^r)=EETS_{ag} \leqslant EETS(wt_2)$.
- 如果缓存在 *TKFEL* 中,按 e^r 定义,有 $tTTS_0=ts(e^r)$.

  所以,$ts(e^r)=EETS_{ag}=EETS(wt_2)$.

综上所述,$ts(e^r) \leqslant EETS_{ag} \leqslant EETS(wt_2)$. □

根据定理 1 和定理 2,$EETS_{ag}$ 被限定在一个合理的范围内.考虑到 $EETS_{ag}$ 被用来计算 *EIT*,且 $ts(e^r) \geqslant EIT$,规则 4 是充要的.计算协议定义了一个灵活的框架,用户可以根据仿真应用特点指定各个线程核参与 *EETS* 计算,

以配置需要的全局同步算法,如图 5 所示.如果仿真应用乐观执行的风险较大,线程核可以在进入乐观模式后马上提交 *tEETS*;如果仿真应用乐观执行的风险较小,则让线程核的乐观推进到一定时戳上限后提交 *tEETS*.函数 *NeedUpdateEETS*()为用户提供了一个可配置的接口,用于判断线程核是否需要提交 *tEETS*.但由于 HPSK 采用的是以 LP 为单位的调度策略,两次调用 *NeedUpdateEETS*()的间隔可能会很长,从而推迟 EETS 计算并延缓 *EIT* 计算.线程核发送事件后,当检测到目标线程核已提交后,便可立刻参与 EETS 计算.但如果线程核长期不发送消息,也就无法检测是否可以提交 *tEETS*,这时就需要 *NeedUpdateEETS*()来协助.两种检测机制相互配合,从不同粒度上控制线程核参与 EETS 计算,以实现高效的全局同步.

```
进程核变量(所有线程可见):                          线程核变量:
int EETSflag;   /*标识是否正在进行 EETS 计算*/      SimTime tOETS;
SimTime tEETS[NUM_THREAD];  /*线程核提交的 EETS*/   SimTime tTTS;
SimTime EETS;   /*最终计算的 EETS*/                 bool tEETSflag;   /*标识线程核是否正进行 EETS 计算*/

发送事件 e 后,与 EETS 计算相关的代码
if (ChkTargettEETSflag()=true && tEETSflag=false)      /*检查目标线程核的 tEETSflag 是否为真*/
      tOETS:=min(tOETS,ts(e));                         /*且线程核本身尚未提交 tEETS 计算*/
      if (TKmode=true && immediate=true)               /*线程核已进入乐观模式且需要立刻更新 EETS*/
            tEETS[thread_id]:=min(tOETS,tTTS);
            tEETSflag:=true, EETSflag:=EETSflag−1;
      endif
endif

线程核推进主程序中的 EETS 计算代码
while (Simulation is not over) do
      if (NeedUpdateEETS()=true)      /*线程核判断是否需要发起 EIT 计算*/
            EETSflag:=EETSflag−1;
            tEETS[thread_id]:=min(tOETS,tTTS);
            tEETSflag:=true;
      endif
      if (EETSflag=0)                 /*最后提交 tEETS 的线程核负责计算进程 EETS,并开始节点间规约算法*/
            EETS:=min(tEETS[NUM_THREAD]);
            EETSflag:=NUM_THREAD;
      endif
endwhile
```

Fig.5 Pseudo code of EETS algorithm

图 5 EETS 算法伪代码

3.2 事件管理服务优化

事件管理服务属于基础服务,主要为仿真系统提供事件创建、传递和提交等服务.在并行离散事件仿真中,LP 之间通常存在大量的事件交互,且每个事件都要经历创建、传递和提交这一过程.事件管理服务是否高效,对整个仿真平台的性能有重要影响.

事件创建和提交是一组对偶操作,分别对应于内存的 new 和 delete 操作.虽然存在支持多线程应用的通用内存分配器[23],但由于 HPSK 独特的系统架构和应用特点,对事件分配回收机制进行针对性设计有助于获得最佳性能:

- 通用内存分配器则需要复杂的机制来支持线程数目动态变化的应用.而在 HPSK 中,仿真运行后线程数目保持固定;
- 通用内存分配器希望同时最小化 new 和 delete 的延迟.而在 HPSK 中,事件创建后需要尽快返回地址指针,实时性要求高;而事件提交的实时性要求相对较低;
- 相比于事件的数目,事件类型的数目很小;
- 线程核内部和线程核之间的事件由不同的线程执行,通过物理上分隔内存可以减少 cache 的假共享冲突.

针对这些特点,本文设计了一种具有线程局部性的双列多级栈结构的事件管理器,采用一种无锁创建、异步提交的机制来解耦和线程之间的关系,以实现高效的事件服务.如图 6 所示,每个线程核预先分配两个私有的事件管理堆,分别用于管理线程核内部和线程核之间两种类型事件.每个事件管理堆由若干个栈构成,每个事件栈保存一组指向相同大小内存块的事件指针,内存块在物理上是连续的.事件的创建和回收都由源 LP 所在线程核完成.当 LP 需要创建事件时,线程核首先检查事件的目的地,确定事件管理堆;然后根据事件的大小查找对应的事件栈,弹出栈顶指针,获得事件.由于事件管理堆是线程局部的,这一过程是无锁的.当需要提交事件时,线程核将事件发送到源 LP 所在线程核的事件回收站中,由线程核定时的从事件回收站中读取并将内存归还到原来的事件管理堆中.这一过程可参照下文提到于事件的传递方式实现.

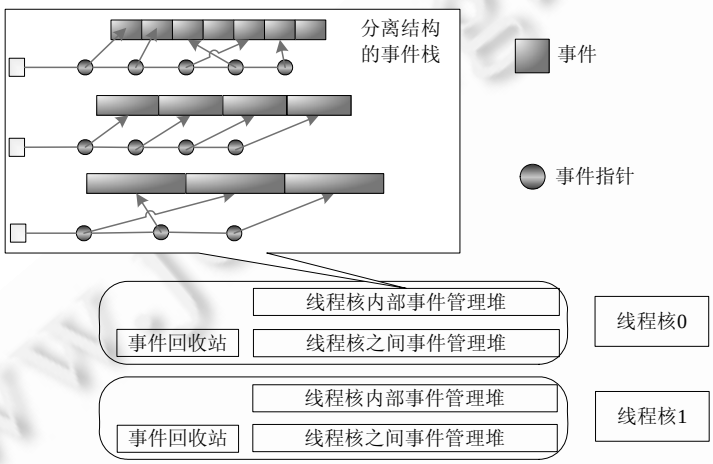

Fig.6 Structure of event allocator

图 6 事件分配器结构

HPSK 中存在 3 种类型通信,线程核内部、线程核之间与进程之间.线程核内部通信十分简单,直接将消息插入目标 LP 的未来事件队列(FEL)中即可.进程之间通信等于两次线程核间通信加上一次网络通信,但多核处理器不能改善网络通信.因此,本文主要针对线程核之间通信进行优化.由于进程内部的所有线程能够共享地址空间,可以利用指针进行事件传递.这将显著地提高通信效率,并减少内存的消耗.然而在仿真执行过程中,各个线程并行推进,同步通信会造成线程的互相干扰,从而极大地影响仿真效率.本文提出了一种事件缓存机制支持基于指针的异步通信,并采用环式队列结构缓存事件,分离事件发送和接收操作,实现线程核之间的高速通信.

图 7 展示了两个线程核之间事件传递的整个过程.当线程核 i 需要向线程核 j 发送事件,它不是直接将事件插入目标 LP 的 *FEL* 中,而是将事件指针存入事件缓存区中;同样,如果线程核 i 向线程核 j 发送反事件,它将事件指针缓存在反事件缓存区中.由于不存在网络通信延迟,事件必然先于反事件存入缓存区.线程核 j 在每次调度循环开始时将两个缓存区的事件指针读入.正事件可直接插入目标 LP 的 FEL 中;对于反事件,线程核 j 可根据下面两种情况做出相应处理:

(1) 如果事件指针存在于目标 LP 的 FEL 中,即事件已收到但未执行,直接从 FEL 中删除事件指针即可;

(2) 如果事件指针存在于目标 LP 的 PEL 中,即该事件已被执行,需要对 LP 进行回滚操作,从而使系统恢复到正确的状态.

在读入事件时(正或反),线程核 j 可能会进行回滚,导致长时间占有缓存区,阻塞其他线程核发送事件.因此,本文采用一种环式队列构建缓存区,如图 7 中圆形内部所示.两个队列轮流作为可写入队列供发送线程核使用.当线程核 i 发送事件时,首先获取可写入队列的序号,插入事件指针.当线程核 j 读入事件时,获取并更改可写入队列的序号,然后读取原可操作队列的事件,执行相应的操作.线程核 i 和 j 通过读写锁控制对可写入队列序号的

访问.

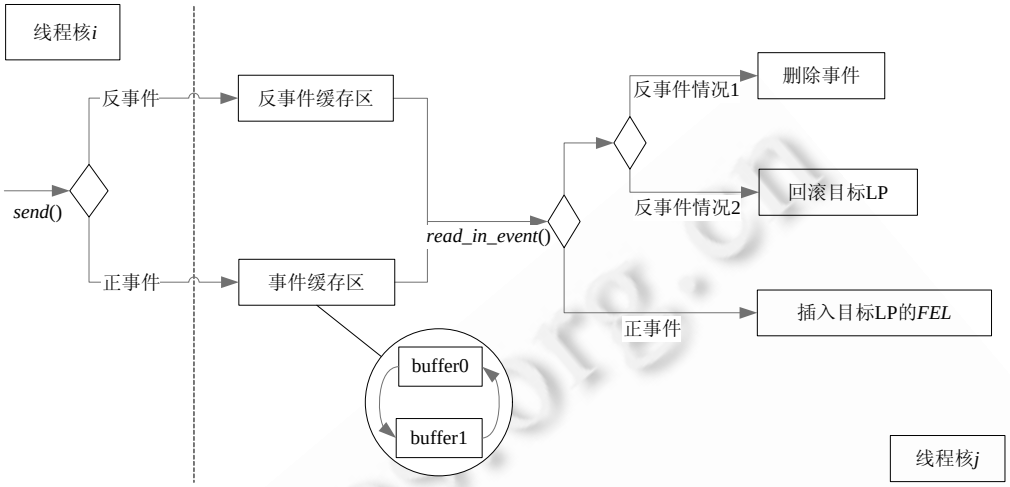

Fig.7 Process of event transfer between ThreadKernels

图 7 线程核之间的事件传递过程

4 实验分析

4.1 硬件环境和Phold模型

并行仿真内核经过服务优化后,能否获得性能上的提升以及能否获得可扩展的提升,是本文所关注的问题.测试平台是一个两路四核的服务器,CPU 为 2.53GHz QuadCore Xeon 处理器,内存 8G,操作系统为 Redhat Server 3.1,内核版本 2.6.18,gcc 版本为 4.1.2.

测试用例选择 phold 模型.Phlod 是离散事件仿真领域中经典的 benchmark,无论在理论上[24]还是实践中[8−10,13],都经常作为仿真内核的标准测试用例使用.其基本描述是:仿真系统由 N 个 LP 组成,这些个 LP 被平均的分配到所有的线程核上.初始化时,每个 LP 产生 R(事件密度)个事件,事件目标按一定规则随机选定.在仿真过程中,各个 LP 接收来自其他或自身 LP 的消息事件,在完成一定计算任务后产生一个新事件,并将事件发送到某个随机选择的目标(可能是自己).事件局部率用来控制线程核内通信和线程核间通信的比例.新产生事件的时戳等于所执行事件的当前时戳加上随机确定的时间增量.该模型可视为对离散事件仿真系统的一个抽象表示,可通过设定不同参数模拟各种特征的仿真应用.

4.2 性能评估

LP 数目(NLP)、前瞻值(lookahead)和事件局部率(locality)是描述仿真应用特征的重要参数.NLP 可视为仿真规模的度量;前瞻值定义为某个 LP 影响其他 LP 的最小仿真时间间隔,可用于衡量 LP 在时间域上的相关性;事件局部率则用于衡量 LP 在空间域上的相关性.下文将从上述 3 个参数入手,深入分析优化效果.

实验 1. 可扩展性分析.

仿真内核能否随着处理器核数和逻辑进程数目的增加,提供可持续的性能增长是衡量服务优化质量的重要指标.图 8 展示了 10 000 个 LP、40 000 个初始事件在不同的事件局部率和前瞻值下,使用不同处理器核数所获得加速比.由于使用单个处理核心时消除了核间通信和同步,无法准确刻画应用和平台特征,这里采用 2 个处理核心性能的一半作为比较标准.可以看出,服务优化的加速效果十分明显,加速效率在 60%以上.尤其在事件局部率达到 50%时,可获得接近线性的加速比.当事件局部率偏低时,通信开销是影响整体性能的主要因素,而处理器核数的增加难以缓解通信开销,导致加速比相对较低.图 9 展示了不同 LP 数目下,仿真内核处理单个事件所需的平均时间.在不同的事件局部率下,虽然 LP 数目按指数增长,但事件执行时间可维持基本平衡(相同条件下扰

动不超过 1.5μs).因此可得出结论:优化的 HPSK 具有较好的可扩展性.

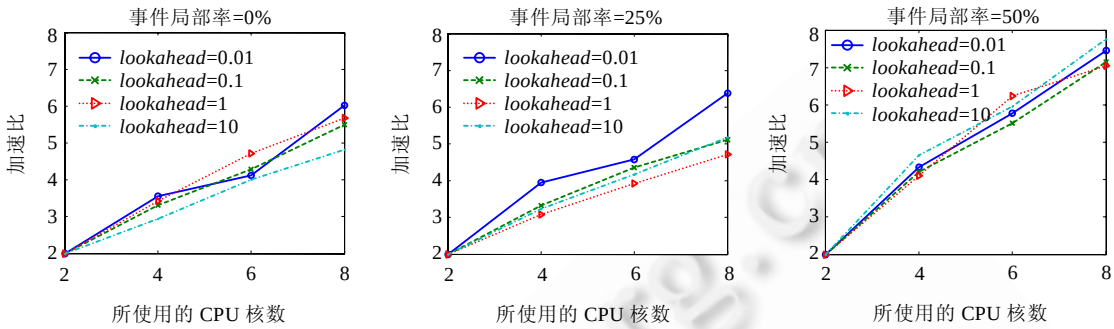

Fig.8 Speedup on different number of cores (N=10000, R=4)

图 8 随 CPU 核数的变化的加速比(N=10000,R=4)

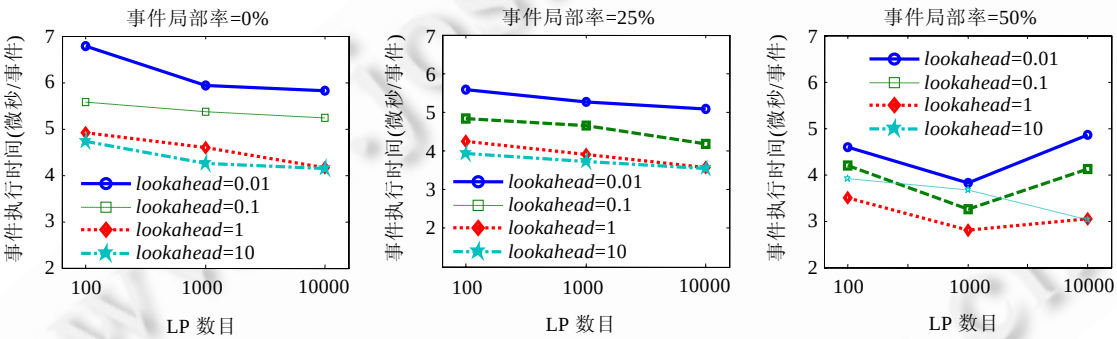

Fig.9 Speedup on different number of logical processes

图 9 事件执行时间随 NLP 变化的情况

实验 2. 绝对性能对比.

musik 是目前唯一支持混合时间管理的基于多进程架构的并行仿真内核,以其高效性和优良的可扩展性闻名,支持共享内存、MPI 和 TCP 多种通信方式.为检验 HPSK 的绝对性能,实验 2 选择 musik[10]进行对比测试.采用其性能最好的共享内存方式作为对比,phold 模型参数同实验 1 一样,处理核数设定为 4 个.从图 10 可以看出,HPSK 优于 musik.因为事件管理服务优化主要针对线程核之间的交互.在相同 lookahead 条件下,当 locality 较小时,线程核间通信概率较大,HPSK 的优势较为明显;当 locality 较大时,线程核间通信概率较小,HPSK 优势不太明显,但仍优于 musik.时间管理服务优化则是提供一个低开销的同步协议.在相同 locality 条件下,HPSK 优于 musik.当 lookahead 值较小时,同步次数较多,优势更明显.

实验 3. 事件密度 R 的影响.

实验 1 和实验 2 反映了特定事件密度(R=4)下的仿真内核性能.如果 R 的取值能对仿真性能产生重要影响,那么实验 1 和实验 2 的结论将缺乏说服力.因此,实验 3 将测试不同事件密度 R 下仿真内核的性能,核数设定为 4 个.从图 11 可以看出,单个事件执行时间随 R 增大略有降低(性能增加),整体上趋于平稳,即事件密度 R 对系统的影响不大.因此,可以认为实验 1 和实验 2 的结论具有一定的普适性.

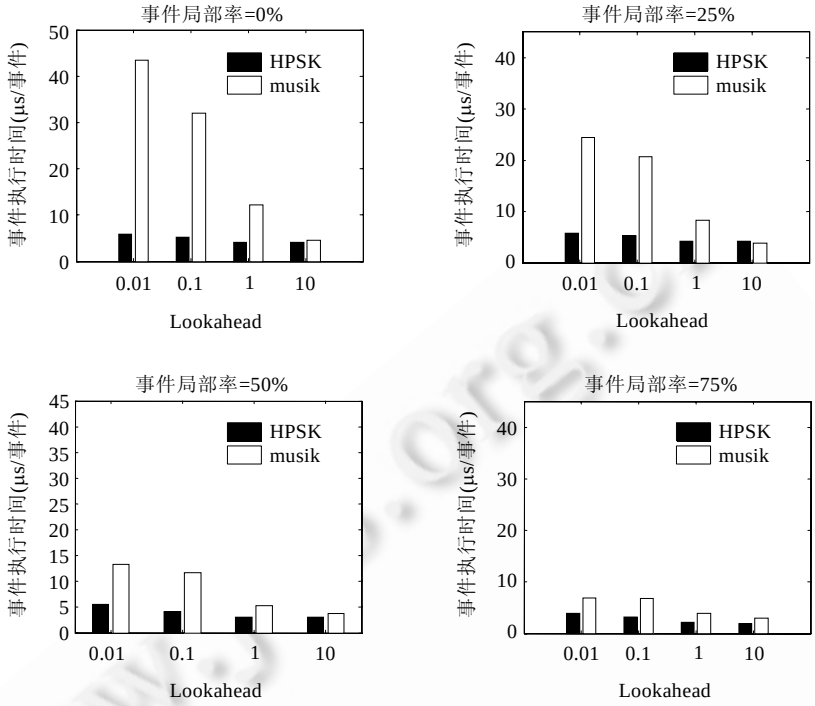

Fig.10 Performance comparison of HPSK and musik (N=10000, R=4)

图 10 HSK 和 musik 的性能比较(N=10000,R=4)

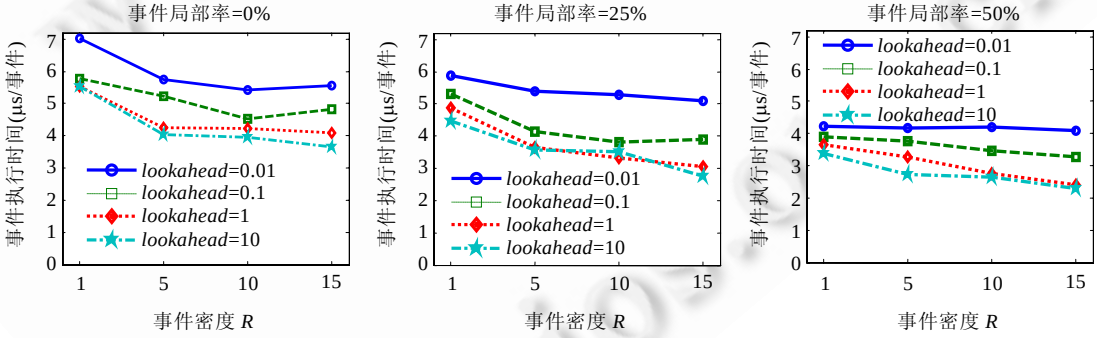

Fig.11 Execution time per event under different event population

图 11 不同事件密度下事件执行时间

5 结束语

随着多核技术的不断进步,可执行的核心数目越来越多,将彻底改变高性能计算机的发展方向.就并行离散事件仿真领域而言,面临应用需求扩张和底层硬件变革的双向压力,发展适应多核的并行仿真技术是一种迫切的现实需求.虽然现有并行仿真平台能够不加修改地运行在多核处理器上,但由于对进程内并发性以及多核体系结构发展的忽视,限制了性能进一步提升的空间.本文提出了一种层次化并行仿真内核,以多线程架构进行仿真调度和事件执行.以此为基础,重点优化了时间管理服务和事件管理服务,从逻辑正确性和效能上对仿真内核提供有力支持.实验表明,通过服务优化的仿真内核能够在多核平台上获得很好的加速效果,且性能明显优于musik 平台.

异构多核是多核技术发展的主要方向,如何充分考虑处理核心的多样性,最大化处理器的整体效能,将是我

们下一步工作的重点.

References:

[1] Asanovic K, Bodik R, Catanzaro B, Gebis J, Husbands P, Keutzer K, Patterson D, Plishker W, Shalf J, Williams S, Yelick K. The landscape of parallel computing research: A view from Berkeley. Berkeley: University of California, 2006.

[2] Akhter S, Roberts J. Multi-Core Programming: Increasing Performance through Software Multi-threading. Intel Press, 2006.

[3] Law AM, Kelton DW. Simulation Modelling and Analysis. McGraw-Hill Education, 2000.

[4] Fujimoto RM. Parallel and Distributed Simulation Systems. New York: John Wiley & Sons Inc., 2000.

[5] Korniss G, Novotny MA, Guclu H, Toroczkai Z, Rikvold PA. Suppressing roughness of virtual times in parallel discrete-event simulations. Science, 2003,299:677−679. [doi: 10.1126/science.1079382]

[6] Steinman JS. Speedes: A unified approach to parallel simulation. In: Proc. of the 6th Workshop on Parallel and Distributed Simulation. 1992. 75−83.

[7] Das S, Fujimoto R, Panesar K, Allison D, Hybinette M. Gtw: A time warp system for shared memory multiprocessors. In: Proc. of the 26th Conf. on Winter Simulation. San Diego: IEEE, 1994. 1332−1339. [doi: 10.1109/WSC.1994.717527]

[8] Carothers CD, Bauer D, Pearce S. Ross: A high-performance, low memory, modular time warp system. Journal of Parallel and Distributed Computing, 2002,62:1648−1669. [doi: 10.1016/S0743-7315(02)00004-7]

[9] Bagrodia R, Meyer R, Takai M, Chen YA, Zeng X, Martin J, Song HY. Parsec: A parallel simulation environment for complex systems. IEEE Computer, 1998,31:77−85. [doi: 10.1109/2.722293]

[10] Perumalla K. μsik—A micro-kernel for parallel/distributed simulation systems. In: Proc. of the 19th Workshop on Principles of Advanced and Distributed Simulation. 2005. 59−68. [doi: 10.1109/PADS.2005.1]

[11] Yao YP, Zhang YX. Solution for analytic simulation based on parallel processing. Journal of System Simulation, 2008,20(24): 6617−6621 (in Chinese with English abstract).

[12] Su NL, Wu XY, Li Q, Wang WP, Zhu YF. Optimistic parallel discrete event simulation based on multi-core platform. Journal of System Simulation, 2010,22(4):858−863 (in Chinese with English abstract).

[13] Chen LL, Liu YS, Yao YP, Peng SL, Wu LD. A well-balanced time warp system on multi-core environments. In: Proc. of the 25th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2011). 2011. 1−9.

[14] Steinman JS. The WarpIV simulation kernel. In: Proc. of the 2005 Principles of Advanced and Distributed Simulation Conf. 2005.

[15] Miller RJ. Optimistic parallel discrete event simulation on a Beowulf cluster of multi-core machines [MS. Thesis]. Cicinati University, 2010.

[16] Martin DE, McBrayer TJ, Wilsey PA. Warped: A time warp simulation kernel for analysis and application development. In: Proc. of the 29th Hawaii Int'l Conf. on System Sciences, Vol.1. Washington: IEEE Computer Society, 1996. 383−386. [doi: 10.1109/HICSS.1996.495485]

[17] Tang WJ, Yao YP. HPSK: A hierarchical parallel simulation kernel for multicore platform. In: Proc. of the 9th IEEE Int'l Symp. on Parallel and Distributed Processing with Applications (ISPA 2011). 2011. 19−24. [doi: 10.1109/ISPA.2011.42]

[18] Jha V, Bagrodia RL. A unified framework for conservative and optimistic distributed simulation. In: Proc. of the Workshop on Parallel and Distributed Simulation. 1994. 12−19. [doi: 10.1145/182478.182480]

[19] Samadi B. Distributed simulation, algorithms and performance analysis [Ph.D. Thesis]. Los Angels: University of California, Los Angels, 1985.

[20] Mattern F. Efficient algorithms for distributed snapshots and global virtual time approximation. Journal of Parallel and Distributed Computing, 1993,18(4):423−434. [doi: 10.1006/jpdc.1993.1075]

[21] Fujimoto RM, Hybinette M. Computing Global virtual time in shared-memory multiprocessors. ACM Trans. on Modeling and Computer Simulation, 1997,7(4):425−446. [doi: 10.1145/268403.268404]

[22] Perumalla K, Fujimoto R. Virtual time synchronization over unreliable network transport. In: Proc. of the Workshop on Parallel and Distributed Simulation. 2001. 129−136. [doi: 10.1109/PADS.2001.924629]

[23] Berger ED, McKinley KS, Blumofe RD, Wilson PR. Hoard: A scalable memory allocator for multithreaded applications. In: Proc. of the 9th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems. 2000. 117−128. [doi: 10.1145/356989.357000]

[24] Nutaro J. On constructing optimistic simulation algorithms for the discrete event system specification. ACM Trans. on Modeling and Computer Simulation, 2008,19(1):1−21. [doi: 10.1145/1456645.1456646]

附中文参考文献:

[11] 姚益平,张颖星.基于并行处理的分析仿真解决方案.系统仿真学报,2008,20(24):6617−6621.

[12] 苏年乐,李群,王维平,朱一凡.基于多核平台的乐观并行离散事件仿真.系统仿真学报,2010,22(4):858−863.

唐文杰(1984－),男,湖南长沙人,博士生,主要研究领域为并行离散事件仿真.
E-mail: tangwenjie@nudt.edu.cn

姚益平(1963－),男,博士,教授,博士生导师,主要研究领域为并行与分布仿真.
E-mail: ypyao@nudt.edu.cn