DOI:10.7544/issn1000-1239.2015.20140104
52(1):3-15,2015

计算机系统模拟器研究综述

刘雨辰[1,2] 王 佳[1,2] 陈云霁[1] 焦 帅[1,2]

[1](计算机系统结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)

[2](中国科学院大学 北京 100049)

(liuyuchen@ict.ac.cn)

Survey on Computer System Simulator

Liu Yuchen[1,2], Wang Jia[1,2], Chen Yunji[1], and Jiao Shuai[1,2]

[1](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)

[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Computer system simulator has long been a useful tool for researchers. It is applied in many different areas, from code design to software programming. In the development of simulators, performance has always been the main focus of researchers, and the improvement of performance will in return benefit the performance of real computers. A number of optimization work has been proposed in both serial and parallel simulations, such as threaded code, binary translation, FPGA accelerator, simulation separation techniques in serial simulation and the solution for the load balance, synchronization and communication in parallel simulation. In this paper, we provide several basic rules and structures that are used in common simulator design, and summarize recent studies of serial and parallel simulation and simulators. First, we introduce the current development of simulators, including current research results, technical problems and challenges. Then, we talk about the structure and the classification of current simulators. After that, the technique in serial simulators is introduced, and the optimization work in parallel simulation is also organized, according to the problems they tend to solve. Some mature simulators as well as simulation platforms are presented later in the paper. At last, potential issues and future work are also introduced.

Key words simulator; serial simulation; parallel simulation; parallel discrete-event simulation; load balance

摘 要 计算机系统模拟器已经成为计算机系统结构领域研究中不可或缺的工具,真实计算机系统的不断发展对模拟器的性能要求也越来越高,模拟器的性能提升也促进了真实计算机结构和性能上的进步.为了提升性能,模拟器的发展经历了从串行单线程模拟到多处理单元并行模拟的发展趋势.串行模拟器和并行模拟器分别针对各自的模拟目标和模拟过程提出了各种优化方案,串行模拟器研究者提出了交织码、二进制翻译、FPGA 加速、模拟分离等加速技术,而并行模拟器在串行模拟器基础上针对自己特有的支撑架构以及负载均衡、同步机制和通信机制等问题提出了各种解决方案.

关键词 模拟器;串行;并行;并行离散事件模拟;负载均衡

中图法分类号 TP337

收稿日期:2014-10-13;**修回日期**:2014-11-12

基金项目:国家自然科学基金项目(61100163,61133004,61222204,61221062,61303158,61432016,61472396,61473275);国家"八六三"高技术研究发展计划基金项目(2012AA012202);中国科学院战略性先导科技专项基金项目(XDA06010403);中国科学院对外合作重点项目(171111KYSB20130002)

计算机系统模拟器是进行计算机系统结构领域研究中不可或缺的工具,它和其他模拟器如气象模拟器、地震模拟器不同,计算机系统模拟器运行在宿主主机上面,从硬件和(或)软件方面模拟目标主机的架构与运行过程,从而方便我们对目标主机的研究和分析.使用模拟器可以有效地减少硬件成本,能够实现对软硬件的配置和观察,方便计算机研究人员完成对硬件和软件的设计和测试.

各种各样的模拟器和模拟技术被不断地提出和修改,而对这些技术和方案的评价主要来自以下 5 个方面:1)模拟结果的准确性应该得到保证,这是模拟器设计的重要考虑因素;2)模拟的速度应尽可能快,相比于功能模拟,时序模拟的速度更难提高,如何在尽量短的时间内完成时序模拟工作对模拟器研究人员提出了巨大挑战;3)模拟器占用的内存应当尽量小,其下限是原始程序在真实机器上执行时所占用的内存空间;4)模拟器部署实现起来应尽量简单易用;5)待模拟的系统和程序只需要进行少量修改或者不进行修改就可以在模拟器上正常运行.

现有的模拟器从实现的角度来看主要分为两类:串行模拟器和并行模拟器,本文也将分别从这两个角度入手对于现有的模拟器研究状况进行总结.

串行模拟器的设计思路较为自然,并且实现起来较为简单,在面对众多指令集架构和操作类型时,串行模拟器也是人们首先想到的解决方案.并且解释执行、交织码、二进制翻译加速等技术的提出使串行模拟器的设计方案日臻完善,成熟的串行处理器也有很多成功的案例.而支持更多的架构、更快的执行效率、更高的准确度一直是模拟器追求的目标,有关串行模拟器的研究工作也从未停止.而并行处理器由于需要处理各个逻辑处理单元(logic processor)之间的负载均衡、同步协作、交互通信等问题,虽然研究者已经提出了很多优秀的加速技术和解决方案,但是模拟时间过长和内存占用等问题都凸显出来.并且随着众核时代的到来,传统单进程的串行模拟器已经无力支撑这样的模拟工作,而并行模拟由于可以在多处理器上完成模拟工作,并且无论从时间上还是存储空间上都具有很大的优势,成为人们在众核处理器模拟工作中的首选,因此更多着眼于众核模拟器设计的并行模拟研究也纷纷展开[1-2],国内的李越等人在分布式并行模拟技术上也有相当大的贡献[3].很多研究工作为了解决并行模拟中关键问题,如负载均衡、同步机制、通信机制等,都取得了相当多的成果.此外,针对乱序超标量处理机设计研究中常用的精确仿真存在的困难,如观察处理器微结构参数变化所带来的性能变化趋势以及获取参数之间的关系需要大量的精确仿真实验,文献[4]提出了基于机械建模与时序模拟结合的模拟方法.

1 模拟器基本结构

模拟器根据模拟对象和功能的不同,可以分为以下 3 个基本模块,如图 1 所示:

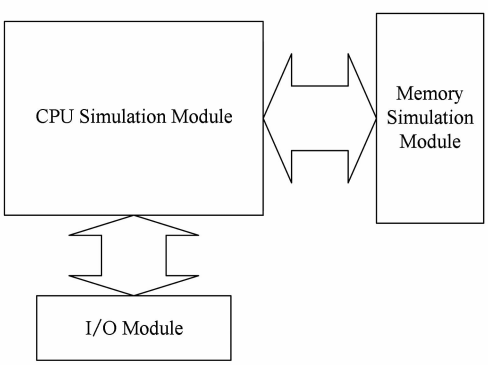


Fig. 1 Base structure of simulators.

图 1 模拟器基本结构

1.1 CPU 模拟部件

CPU 模拟部件是整个模拟器最重要、最复杂的部分.因其结构比较复杂,有很多系统模拟器会选择在高度模块化和定制化的 CPU 模拟部件的基础上进行扩展,期望专注于特定机器的性能和效率.设计 CPU 模拟部件最直接的方法就是将目标机器的二进制代码中的每一条指令都能转换为同样语义的宿主机器的指令,在宿主机器上执行.通过将目标机器的每一个寄存器和标识位对应一个变量,目标机器的所有逻辑操作都可以被间接地翻译为变量的运算和各种各样的简单或者复杂的算法操作,这样目标机器上程序的所有操作都可以在原始机器上以软件模拟的形式复现出来.这种翻译的方式实现起来比较简单,在模拟一些不太复杂的机器和系统时表现尚可,被多数的模拟器所采用.但是如果目标机器的 CPU 复杂度较高,与宿主机器的 CPU 复杂度相仿甚至更复杂,那么逐条指令进行翻译这种方式所产生的速度瓶颈就会逐渐浮现出来.解决这一问题的方案主要采用动态重编译的方式.

1.2 内存模拟部件

最原始的方法就是采用整片字节阵列的存储空间来模拟目标主机的内存空间,但是这种简单的存储控制机制在目标主机有着复杂的逻辑地址和物理

地址映射机制的情况下就会显得束手无策.有些目标主机会有内存管理单元(memory management unit)来控制逻辑地址和物理地址的相互转换,这种结构都说明不会有整块的地址被映射到 ROM,因此上面提到的阵列方法并不实用.作为替代,多数模拟器都会采用多种方式来实现内存读写,保证目标地址的逻辑地址和物理地址的一致性.

1.3 I/O 部件

为了提升设计效率与通用性,I/O 部件往往会遵循统一的标准,在已经定义好的 I/O 的应用程序接口(API)基础上进行开发.这样做的好处在于,对于不同的模拟设备模拟器所要做的只是对标准 I/O 部件模块进行定制和修改,而不需要重新设计,这会大大提升设计效率.并且采用通用 API 的好处在于,第三方的虚拟部件可以很容易地被应用到新型模拟器的开发与设计中.通用的 I/O 的 API 并不需要完全照搬实际模拟器件和总线的设计结构,例如,总线的物理设计往往需要遵循各种电路设计的限制条件,并需要考虑总线上各个部件的并发管理,这在模拟器的软件实现中都是可以忽略的.尽管通用的 API 下每一个 I/O 部件都被认为是特殊的实现,但是模拟器当中还是有一些结构符合通用的设计原则,如中断控制器的设计应当方便 CPU 查询是否发生中断以及发生何种中断,物理内存的读写应当与逻辑内存的读写方式相同.

2 模拟过程的分类

在模拟过程中,根据模拟的对象、过程、目的、需求的不同,不同的模拟过程对模拟器也有着不同的需求.而很多模拟器对于不同的模拟需求都提供了详尽的解决方案,有些模拟器在划分时会被归为多个分类的范畴.

2.1 系统模拟与局部模拟

有些模拟希望在整个系统层面上进行执行,这些模拟过程需要对 CPU、内存、缓存、中断控制器、I/O 设备和磁盘控制器等设备进行模拟,系统对于各个部件都要提供完整的接口和实现,这一般也会使该模拟器能够启动完整的操作系统,甚至是不加修改的操作系统.系统模拟的主要目的在于帮助对系统的设计方案进行验证和修改,也可以实现对程序和操作系统等应用程序开发的调试工作.常见模拟器有 SimOS[5],GEM5[6],QEMU[7] 等.局部模拟器是指对某个部件进行模拟,可以更改相关参数,观察统计器件在模拟过程中的相关信息.局部模拟不能以真实程序作为输入,而一般采用程序执行信息(称为跟踪文件)作为输入.常见的局部模拟器有缓存模拟器 CMP $im[8]、磁盘模拟器 DiskSim[9]、内存模拟器 DRAMSim[10] 等.

2.2 功能级模拟与时钟级模拟

功能模拟部件在模拟正确的基础上尽量简化内部结构,方便模拟器的开发并提高模拟器的运行速度.一般功能模拟用于跟踪文件的获取和系统软件的开发.而时钟级模拟需要对目标系统内部的详细部件进行模拟,不仅要保证模拟结果正确,还要保证各器件的参数对于运行结果的影响.时钟级模拟的执行速度较慢,但能真实模拟目标系统,主要用于分析系统和程序的参数对性能的影响.一般模拟器都会提供这两种模拟方式,并且大多支持在两种模拟方式之间进行切换,用户可以在模拟时自行进行选择,在需要关注系统性能时采用时钟级模拟,而在需要加速模拟进度关注模拟结果时采用功能级模拟.常见的同时支持功能级模拟和时钟级模拟的模拟器有 GEM5[6],SimOS[5] 等.

2.3 用户态模拟与全系统模拟

待模拟的程序往往都需要执行系统调用,如果模拟器本身能够提供,那么程序可以直接在模拟器上执行,这种情况下模拟器不需要模拟目标系统的特权指令(中断调用等),称之为用户态模拟.而另外一些模拟方式需要首先模拟启动操作系统,然后在操作系统上执行程序,这种方式称为全系统模拟.相比于全系统模拟,用户态模拟(或称为系统调用模拟)可以不需要启动操作系统,也不需要目标系统模拟特权指令直接模拟应用程序的执行过程,但是模拟器需要为用户态模拟提供内存管理和异常处理等各种机制,并且对于牵涉到操作系统的系统调用(如用于新建进程的 fork)等,模拟器是无能为力的.常见的支持用户态模拟的模拟器有 QEMU[7],Mambo[11],PTLsim[12] 等.

3 串行模拟关键技术

3.1 传统解释执行技术

解释执行技术最容易实现,也能够实现对目标主机的完全兼容.其过程遵循“取指—译码—执行”的步骤:首先根据程序计数器(program counter, PC)

从模拟内存中取出将要执行的一条指令,译码之后找到模拟器在原始机器上实现的替代函数,执行该函数之后更新 CPU 状态,如果执行指令之后有中断、异常等发生,那么系统模拟器还需要进行处理,最后 PC 加 1,指向下一条指令,如此循环. 这一执行过程非常直接简明,只要模拟器能够提前将目标系统的指令集以高级语言的函数形式实现,那么就可以完美模拟目标主机的所有行为. 但是这一过程的缺点也非常明显:访存开销和译码开销较大,尤其是后者,对于每一条指令都要有大量的位处理获取该指令的特定位域,以便确定选择怎样的模拟函数,指令集的大小是有限的,但是被模拟的程序指令数是非常多的,对于重复指令的繁复译码操作成为整个模拟过程效率提升的最大阻碍.

3.2 交织码技术

交织码(threaded code)技术是文献[13]提出的,其主要思想是对译码之后的执行函数进行缓存操作. 首先将目标指令以及其操作数翻译为中间代码块,然后对中间代码块进行缓存,执行代码. 之后在模拟器执行目标代码时模拟器会首先检查代码缓存,如果在其中发现当前指令已经被翻译过,那么就从缓存中取出代码块进行执行,如果在缓存当中没有发现被翻译过的代码块,那么就对当前指令翻译成代码块并根据一定的替换策略在缓存中存储. 这一过程将会大大减少译码的频度,当然额外增加了判断指令是否已经翻译的执行过程,但是整体而言,模拟器的效率会得到具体的提升. Simics[14] 就采用了交织码技术.

3.3 二进制翻译加速

二进制翻译是通过将源指令集中的代码翻译成目标指令集的指令序列进行模拟的技术. 在指令集模拟情况下,源指令集和目标指令集相同,翻译之后的指令序列中会加入测试和调试的断点等额外信息获得对程序的更多控制. 二进制翻译有两种实现形式:静态二进制翻译(static binary translation)和动态二进制翻译(dynamic binary translation).

静态二进制翻译方式是指在程序执行之前将源指令集下的二进制可执行程序直接翻译成目标指令集对应的程序,通过这种技术要提高翻译的准确性是非常困难的,并不是所有的代码都能够被模拟器静态检测到. 例如:有些程序片段可能只有在程序运行时发生间接跳转时才会被执行到,而这些跳转的地址也只有在实际执行时才能得到. 文献[15]已经实现了通过静态二进制翻译技术完成的模拟器,只需要很少的翻译代价并且在目标指令集上有着很好的表现.

动态二进制翻译技术专注于每次执行到的代码块,翻译该段代码块缓存其执行序列,跳转指令也会被引导到已经被翻译和存储的代码块. 动态的二进制翻译不同于传统的简单模拟执行,消除了传统模拟中的取指—译码—执行过程,而这正是性能的瓶颈所在,动态二进制翻译技术可以通过多次执行进行多次翻译记录代码执行序列,逐步减少性能瓶颈.

更进一步,对于一次执行的代码块,动态二进制翻译技术实际是没有任何提升和帮助的,反而只会引入动态分析等额外开销而降低性能. 理想情况下,模拟器应当对于已经执行的代码块进行频率统计,找出其中开销较大的代码块应用交织码技术和二进制翻译技术,方能提高效率. 利用二进制代码或者源代码插桩等技术实现的分析制导(Profiling)就是这样一种动态分析工具,根据数据粒度不同可以分为事件分析器、统计分析器、插桩分析器等等. 在动态二进制翻译过程中,例如模拟器 FX!32[16] 中就有这样一个称为执行分析器(execution profiler)的部件,每次执行之后该部件会产生一些分析信息,包括所有调用指令的跳转地址、指令的源地址和目的地址和对内存的非对齐引用等. 正是通过这些分析制导技术才能对热点代码块进行二进制翻译加速优化等操作,弥补动态分析引入的统计工作等额外代价. 优秀的动态分析优化部件是二进制翻译的必不可少的一部分,这对于提高二进制翻译的效率是非常重要的. 动态翻译技术在工业界用途广泛,SimOS[5], QEMU[7] 等诸多厂商都在自己的产品中使用了动态的二进制代码翻译技术,来实现并加速对不同指令集的翻译.

3.4 FPGA 加速

FPGA 加速模拟技术(FPGA-accelerated simulation technologies, FAST)[17] 是采用现场可编程门阵列(field-programmable gate array, FPGA)技术设计的加速器,它把模拟过程分为两部分:功能模拟(functional model)和时序模拟(timing model),前者采用了定制的全系统模拟器用来模拟目标主机的指令集和外部接口,并执行可执行代码、操作系统和基本输入输出系统(BIOS)指令;后者采用了 FPGA 硬件实现部分目标 CPU 的组件用来模拟微结构、模拟分支预测和 cache 状态等. 功能模拟模块和时

序模拟模块之间有缓冲区连接,功能模拟能够提供正确的运行结果,而时序模拟模块可能会产生错误的分支预测,此时时序模拟可以通过从缓冲区中获得功能模拟产生的信息来同步自己的分支预测.这样就可以产生一个模拟准确、效率优良的模拟器.

多处理器加速器(research accelerator for multiple processors, RAMP)是 Patterson 等人组织的一系列研究和项目的组合[18].RAMP 利用 FPGA 实现对全系统的模拟,现有的模拟器一般对于 16 核以上的系统的模拟难以实现,RAMP 的主要目的就是实现对多核的系统实现模拟,并且研究多核系统中的系统、编译器、操作系统、应用软件等的开发.

3.5 分离模拟技术

未来处理器设计工作中将会越来越多地考虑到多线程程序的实现,线程的交织方式决定了多线程程序能否正确地并行执行,而在实际系统中,系统的时序顺序决定了线程的并发执行方式.如何在模拟器中实现这些时序相关的线程是整个模拟器成败的关键.上面提到的 FAST[17] 就采用了类似的方法,将时序模拟和功能模拟相分离,用时序模拟提高模拟的精度,用功能模拟提高模拟的准确度.

3.6 快速转发与检查点技术

有些情况下的模拟只是针对于其中一段程序感兴趣,而不是对整个程序的过程感兴趣.这时采用传统的模拟方式很难快速到达模拟起始点,有些模拟器就提供了快速转发(fastforward)机制和检查点(checkpoint)机制来解决这个问题.快速转发技术是指在模拟程序启动时采用简单配置的功能模拟的方式快速推进到感兴趣的程序片段的起点,然后切换成复杂的配置状态进行详细的模拟.这种方法虽然速度快,但是存在着一些问题.正常情况下,程序运行到待观察片段起始位置时,缓存和分支预测等部件都有了很多的先验信息,而采用快速转发技术利用功能模拟推进到待观察点时,缓存等部件都是空的,模拟器可以选择对这些部件进行预热(warm-up),这样可以提高一定的准确率,也可以直接开始模拟,但这种方法会导致程序片段起始状态的不准确.为了更方便地对程序片段进行模拟分析,有些模拟器还采用了检查点技术.模拟器在运行到感兴趣的程序片段起点时进行检查,详细记录模拟器内部各个部件的状态信息,然后下次执行时,模拟器可以根据上次记录的部件信息进行现场恢复,继续模拟.这种方法可以对感兴趣的程序片段进行多次模拟分析,大大提高了模拟器的效率.Gem5 模拟器[6] 对于这 2 种技术都有比较良好的支持.

4 模拟并行化

在串行模拟技术越来越成熟化时,人们逐渐把目光投向了并行离散事件模拟技术(parallel discrete-event simulation, PDES)[19].

4.1 并行模拟技术的提出

以下因素促使了并行模拟技术的提出和发展:

1) 如果待模拟的程序是离散事件并且规模较大计算较复杂,那么在支持并行执行的计算机上进行模拟能够实现较快的运行速度;

2) 待模拟程序中有些执行序列有潜在的可并行化能力;

3) 待模拟系统会在离散的时间点受事件驱动改变运行状态,例如被模拟的计算机网络的某个节点接收到了特定消息;

4) 模拟本身关注的就是分布式的系统或者程序,在这些模拟当中并不会有全局时钟来同步每一个事件.

4.2 并行模拟系统

一个典型的并行离散模拟事件系统是比较复杂的,它需要维护 3 方面的信息:1)状态变量信息——记录模拟器当前状态;2)事件队列——记录待处理事件;3)全局时钟变量——维护不同事件之间的同步和因果关系.在并行系统中每个事件都会获得一个时间戳,一般来说时间戳较小的时间会对系统状态造成影响,从而影响到时间戳较大事件的执行,因此一般系统要求所有事件按照时间戳不递减的顺序来处理[20].在一般的串行系统当中,模拟器总会不断将事件队列中时间戳(time stamp)最小的事件取出来进行处理.这种处理过程不仅会对事件本身的结果造成影响,还会对系统的状态造成影响,并且驱动模拟器的控制单元,调度新的事件来处理,以维持系统的因果关系.而在并行系统中事件集合被分配到不同的子系统中执行,这个子系统被称为逻辑处理单元(logical processor, LP).在单个逻辑处理单元的内部,事件队列按照时间戳的非递减顺序进行处理,这可以保护逻辑处理单元的局部因果关系.而在并行处理中,如何维持多个逻辑处理单元之间事件的因果关系要比维持单个逻辑处理单元内部的因果关系复杂.并行模拟中没有全局时钟,传统离散模拟中所有时间的全序关系退化为局部的偏序关系,

不同的逻辑处理单元之间通过传递时间戳消息避免出现因果关系混乱的情况.

4.3 并行模拟系统架构

并行模拟系统的架构包括两个部分:目标器件模拟部分和模拟系统支撑部分.前者是指对于目标系统的 CPU、内存、I/O 等部件的模拟;而后者则是指为了实现这些模拟,模拟器需要提供的功能,例如支持待模拟器件之间的通信与调度等问题.前者在串行模拟系统和并行模拟系统中的实现并没有太大差异,而后者则是并行模拟系统主要考虑部分,它不仅需要考虑到不同逻辑处理部件之间的协同工作,以实现功能模拟的正确,还要考虑到对模拟器进行各种优化,防止出现模拟效率慢于串行模拟等极端情况.而在模拟系统支撑部分中需要重点考虑以下 3 个问题:

1) 负载均衡.在串行系统中所有的目标器件和任务都在同一个处理器上,但是在并行系统中由于有多个逻辑处理单元的存在,任务集合需要被划分到不同的处理单元上.如果划分得不合适,那么很可能产生有些处理单元一直忙碌而有些处理单元经常空闲的情况,这不仅仅会浪费资源,还会导致并行模拟整体效率的下降、模拟时间的上涨.优秀的负载均衡策略应当注意:①所有处理单元的负载尽可能相等;②不同处理单元之间的通信开销之和尽可能小.该问题可以转换为无向加权图的划分问题,这是 NP 问题,但是目前有一些启发式方式可以给出比较满意的近似求解[21].

2) 同步机制.模拟系统不同部件之间的执行需要维持统一的进度,在串行系统中由于各个部件被调用时遵循一致性的关系,所以不存在同步的问题.而在并行模拟系统中没有全局的时钟来自然维持各个部件的进度,所以需要一定的同步机制来实现同步,PDES 同步机制[22-23]是常见的解决方法.

3) 通信机制.串行模拟环境下所有的部件都在同一个进程内部,其通信机制可以使用单纯的函数调用来实现,而并行模拟系统中不同的器件之间的通信需要传递消息来实现.并行系统中多个部件之间可以同时发送和接收消息,如何实现高效、完善、并发、安全的通信机制是并行模拟系统设计者需要考虑的问题.

4.4 并行模拟系统常见问题

并行模拟优化技术主要可以从负载均衡、同步机制和通信机制 3 个方面来分析,其技术框架如图 2 所示:

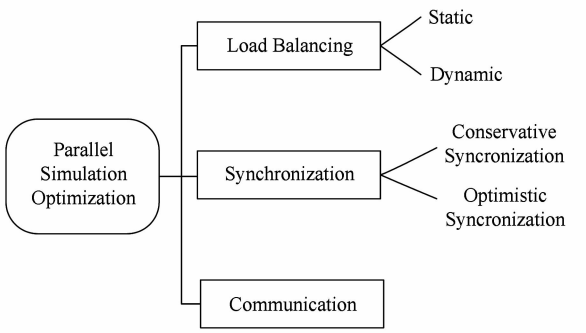


Fig. 2 Framework for optimization of parallel simulation.

图 2 并行模拟优化技术框架

4.4.1 负载均衡

模拟器的并行化带来的主要开销是各个并行处理单元与并行器件之间的同步开销,为了提升并行模拟的性能,模拟器有必要通过任务划分来减少不同处理单元之间的同步操作.

根据 PDES 保守同步机制[22],处理单元在遇到时钟阻塞时会向周围部件发送时钟更新请求,对方收到请求会将最新时间戳以空消息形式返回.如果各个处理单元之间任务划分不均衡,会导致各个部件的推进速度不一致.为了完成同步,速度较快的部件会发送大量空消息,而速度较慢的部件接收到空消息时也会回复大量的空消息,这将会导致系统资源被大量占用,系统性能大大下降.因此如何将任务在各处理部件之间均衡分配是并行模拟器需要首先考虑的问题.

当然上面提到的任务分配的目的是为了让各个处理部件之间的进度尽量统一,时间戳尽量同步,避免过多的同步开销.如果在串行模拟中所有的事件都被分配到同一个处理单元中,就不存在通信开销,但是在并行模拟中,由于分配到不同处理单元的事件之间需要进行通信,这样的通信开销是之前串行模拟中不曾遇到的,如何分配任务使得通信开销尽量减少也是负载均衡的目的之一.

一般负载调度的方法分为两种:静态负载均衡(static load balancing)和动态负载均衡(dynamic load balancing).静态负载均衡方法提前对模拟程序进行分析,在模拟开始之前就对所有的任务在逻辑处理单元之间进行划分.而动态负载均衡方法则允许在并行模拟的执行过程中处理单元进行任务和负载的调整.静态负载均衡需要事先对程序的细节有详细的了解,一旦确定好负载划分之后,在程序执行的过程中就不再变化,不会在程序执行过程中引入额外的开销;而动态负载均衡技术不需要提前对

程序有详细了解,模拟器会在程序执行过程中根据程序的运行状态和各个逻辑处理单元的状态来调整负载分配,这将会引入额外的开销,但是消除了程序开始之前的分析阶段.另外有些程序特征无法静态分析得到,只有在程序动态执行中才能得到并不断变化,这也是动态均衡方法更加适用的原因.

无论是静态负载均衡还是动态负载均衡,在模拟程序启动之前都要通过对程序进行分析后划分.这是典型的图划分问题:给定一个图 G,它带有 m 个带权重的定点和 n 个带权重的边,需要将它划分为 p 个集合,使得每个集合中所有定点的权值尽可能相等,而集合之间的边的权值之和尽可能减少.这个问题下面一些启发式算法可以给出在给定配置之下的近似最优解[21].

1) 静态负载均衡技术

早先的静态负载均衡方法中的分配算法是对程序的所有可能执行路径进行聚类,找出其中的关键执行路径[24].

文献[25]采用了文献[22]中使用的同步协议(空消息协议).算法对负载图进行了随机的初始划分,程序运行时对可能会产生的逻辑单元之间的负载转移进行计算与评估,使用收益函数来对逻辑单元之间的通信进行考量,来确定特定负载转移是否有效.这一算法要求模拟器对程序的通信开销和计算开销有着详细的先验知识.

文献[26]对于数字逻辑模拟的并行化问题也采用了静态负载均衡技术.文中采用的是二分迭代的方法,将数字电路不断进行二分,使得两个子电路之间的通信开销最小,直到每一个子电路都能达到单个模拟电子部件的规模,并能够被单个逻辑处理单元模拟为止.Briner 还对该方法作出了一些变化,使得该算法能够处理子电路之间的通信带有权重的情况,这种改进算法对于其他非数字电路模拟器设计有很大的参考价值.

一般采用静态负载划分的程序都会分析程序执行片段,得到反映事件之间依赖性的划分图,然后根据该图得到任务在逻辑处理单元上的最优划分.这种离线的分析方法能够使得并行模拟的执行获得更好的效率,相比而言,有些动态划分的额外开销甚至会超过它带来的效能提升.但是静态划分方法需要提前对程序的详细执行信息有充分了解,这始终是瓶颈所在,动态的负载均衡算法就应运而生.

2) 动态负载均衡技术

一般使用的动态负载均衡技术多是基于乐观同步机制.但是在动态负载均衡技术中,乐观同步机制中的两个问题需要谨慎地对待.

高的处理单元负载率并不代表高效率.这是因为很多在逻辑处理单元已经处理过的事件后都有可能遇到回滚和撤销的情况,这也是乐观同步机制的特征之一.这种情况下将那些可能会被撤销的任务分配给高负载率的处理单元反而是更合适的选择,这些任务在高负载率的处理单元中会被分配更少的计算资源,以后即便被撤销也不会浪费非常多的计算资源.为了实现这一方案,Reiher 等人提出了称为有效处理单元使用率(effective processor utilization)的评价机制[27],这个度量反映了处理单元所处理的任务中最终会被提交而不会撤销的任务所占用的计算时间,而把那些执行最终会被撤销的任务所占用的时间看作空闲时间(idle time).根据这一定义或者类似定义,Reiher 等人提出了将任务从高有效使用率的处理单元向低有效使用率的处理单元迁移的一般实现方案.

在动态规划和任务迁移的过程中,时钟弯曲(time warp)[23]中还存在一个具体的实现问题.由于处理单元往往会维护大量的历史信息,处理单元在迁移时也需要对这些历史信息进行处理,这将会提高动态规划的开销.Reiher 等人提出的解决方案是将任务的执行在发生任务迁移时划分为多个片段(phase).当任务迁移出现时,老的片段(包含其历史信息)依然存在于原来的逻辑处理单元上,而新的片段则出现在新的逻辑处理单元上,新的片段则是旧有片段的拷贝,只不过根据新的逻辑处理单元更改片段里的状态参数等信息.这样在任务迁移时就不需要对大量的历史信息进行转移,而只需要记录任务迁移时刻旧有片段的状态信息.

Glazer 在文献[28]中也研究时钟弯曲机制的负载转移,其目的是减少回滚次数而提升系统运行速率.Glazer 以任务运行的时间作为判断标准来分发时间片,在各个逻辑单元运行时分配合适长短的时间片以保证各单元的推进速度的一致.Boukerche 等人[29]提出了基于保守 PDES 的动态负载均衡方法,其主要方法是在模拟器中额外增加负载平衡线程来对全局负载信息进行收集和判断,增加负载迁移线程来对需要迁移的节点之间进行操作和执行.其中的负载平衡线程对全局负载信息的收集工作使用了令牌簇的策略,即将底层的多个节点看成一簇,负载平衡线程向其发送令牌,在令牌收集到簇内所有节点信息之后返回.这样做的好处是能够避免负载均衡

线程成为系统执行的瓶颈,但是这种复杂操作对于一般模拟执行开销过大.

4.4.2 并行同步机制

常见的维持不同逻辑处理单元之间因果关系的并行机制有以下两种:

1) 保守策略

文献[22]提出了 CMB(Chandy-Mista-Bryont)算法,这是有关并行模拟的最早算法,并且在之后的有关保守同步机策略的后续研究中也都以该算法作为扩展.

例如处理单元 P 包含未处理的时间 E_1,该事件的时间戳为 T_1,而 T_1 应当为该处理单元所拥有事件中最小的时间戳,即在 P 处理时间 E_1 之前它不能接收时间戳小于 T_1 的事件.

具体实现起来遵从以下 3 个原则:

① 静态地将事件分配到各个处理单元中,并确定事件之间的直接相连;

② 确保每个处理单元中事件的时间戳顺序能够反映事件本身的先后执行顺序;

③ 每一个处理单元的执行过程总是需要维护一个时钟,来反映当前需要处理的事件队列中的头事件的时间戳信息,如果事件队列为空的情况下,该时钟需要反映该处理单元最近接收的消息的时间戳信息.

每一个处理单元内部的处理流程如下:

① 找到所有处理队列连接中时间戳最小的一个队列;

② 如果队列中有待处理事件,则取出事件进行处理,更新该队列时间戳,转入①;

③ 如果队列为空,那么阻塞等待,直到该连接上有新的外部消息到达,更新该队列时间戳,转入①.

但当系统中出现了空队列时就会出现死锁(deadlock)现象,因为此时队列中没有信息,处理单元处于等待状态,但是如果多个队列同时处于相互等待状态就会出现死锁现象.

在基本的 CMB 算法中研究人员提出了发送空消息来消除死锁的办法,空消息是指携带时间戳信息的消息,发送空消息机制如下:

① 每个队列会以固定周期发送带有时间戳 T_i 的空消息,表明时间戳小于 T_i 的消息能够在该处理单元中安全处理.

② 接收到空消息的逻辑单元提取其时间戳,将本消息队列中时间戳小于 T_i 的事件发送出去. 如果不存在时间戳小于 T_i 的事件,那么该消息队列堵塞.

③ 如果系统中所有队列同时为空就会出现死锁现象,发送空消息的方式不能解决死锁现象. 解决方法是从所有消息队列中取得时间戳最小的消息进行处理.

在上述基本的保守策略基础上还能有提升,例如引入 *lookahead* 机制[22]. *lookahead* 指的是从下一个从该处理单元发出的消息的时间戳的下限与当前时钟的差值. 例如,一个处理单元的当前时钟是 T,而其 *lookahead* 值为 L,这说明该处理单元以后发出的消息的时间戳都要大于 $T+L$. 保守同步机制在系统上的效率受到 *lookahead* 值的影响非常大. 极端情况下,*lookahead* 值为 0,那么这就是传统的保守同步策略. 原始的 CMB 算法存在大量的空消息,如前所述,这种现象要求任务分配合理达到负载均衡. 当然面对这一现象也有很多其他的解决方法,CMB 算法中面对死锁的原始解决方法是死锁避免,当然也有上面提到的死锁检测和死锁恢复,这种方法对于 *lookahead* 为 0 的情况下可能会出现的死锁现象也能进行有效的处理.

保守策略的缺点也是显而易见的:对于抢占式资源分配方式不适用;模拟器需要对进程进行静态规划;使用者需要对待模拟系统和程序有充分的了解.

2) 乐观策略

与保守策略尽力避免违反局部因果关系的宗旨不同,乐观策略则是允许这种特例出现,但是能够检测与恢复. 与保守策略相比,乐观策略能够将待模拟程序的并行化程度提高,如果两个事件可能会相互影响,但是两个事件的处理结果证明这种担心是多余的,那么乐观同步策略可以并行处理这两个事件,而保守策略则会根据局部因果关系顺序处理这两个事件. 另外,保守策略需要依赖程序的详细信息来决定事件的执行顺序,而乐观同步则不需要这些详细的信息来确保并行模拟的正确性,对于待模拟程序来说,乐观策略比保守策略更加透明,简化了程序的编写与编译. 但是乐观同步会消耗较多的资源和事件用于计算可能会回滚(rollback)的事件,这会导致一定的性能损失.

文献[23]提出的时钟弯曲策略是乐观同步的代表. 其思想是当逻辑处理单元在接收到时间或者消息时,如果该事件的时间戳小于逻辑处理单元队列中最小的时间戳,逻辑单元就会回滚到接收到的时间戳. 回滚不仅需要恢复该逻辑处理单元将本队列中之前执行过的时间戳大于接收到时间戳的事件,还需要撤销(unsend)这些事件之前向其他处理单元

发出的消息,实现这种功能的机制成为反消息(anti-message)机制.反消息是之前已经发出的正消息(positive message)的拷贝,如果发现消息队列中同时拥有两个完全一样的配对消息,那么这两个消息都要被删除.如果一个处理单元接收到反消息,那么就应该根据匹配机制删除它已经接收到的正消息,这可能会引起一系列的相关联消息的回滚操作,这一迭代过程直到所有的错误消息都被删除之后才会结束.

这一过程在付诸实践之前有两个问题需要解决:①有些操作是无法回滚的,例如 I/O 操作;②由于处理单元要维护已经执行的事件的历史队列,这对于内存的消耗无疑是巨大的.这两个问题都可以通过全局虚拟时间(global virtual time, GVT)来解决,GVT 代表了未来会发生回滚所牵涉所有时间戳的下界,这就意味着时间戳小于 *GVT* 的事件状态和消息都可以安心处理,不需要为了回滚而被记录下来.所有未处理事件和仅部分被处理事件的时间戳的最小值就给出了 *GVT* 的值,一旦 *GVT* 的值被确定,所有时间戳小于 *GVT* 的 I/O 操作都可以放心提交,所有时间戳小于 *GVT* 的存储空间都可以被复用.

单纯的时钟弯曲策略会受到过于乐观执行引发问题的困扰,有些处理单元可能会领先其他处理单元太多时间导致存储不足问题以及过长的回滚问题.为了解决这些问题,文献[30]提出了限制乐观执行跨度的方法,文献[31]提出事件处理后要延后提交,直到 GVT 推进到该事件被模拟的时间戳,这可以减少不必要的连锁回滚和撤销消息的发送.虽然有了上面的一些解决方法,但是乐观策略还是需要内存来存储已经提交的事件来实现回滚的操作,为了解决这一存储问题,文献[32]提出回滚计算可以复用之前用过的存储,文献[33]认为没有必要在每一次事件之后都记录状态,有些时间戳大于 GVT 的状态所占用的内存空间也可以被复用.有些对时钟弯曲方法的改进使用了用户定义的参数,例如上面提出滑动时间窗,传统情况下都是进行分析实验找出较优的参数值固定下来然后进行模拟.后续工作,如文献[34]等都提出在模拟过程中采用自适应的方法来调整这些参数的值,从而提升模拟效能.

4.4.3 通信机制

计算机模拟中模块之间的耦合非常密切,线程之间的通信也非常频繁,通信系统的效率显得尤为重要.

同步通信的过程中发送和接收信息的双方在同步过程中都需要作出等待,一般高性能模拟中采用异步方式进行通信.异步方式需要在消息的并发存取过程中进行加锁等互斥操作,否则消息的通信将会出现混乱,但互斥操作都将会给模拟系统带来巨大的通信开销.

同一个进程内部的通信可以采用锁避免的方式进行通信,忽略了缓存的过程,提升了通信的效率.一般方法是为该进程维持一个先进先出消息队列,消息发送者直接将消息放入消息队列,而消息接收者从消息队列中获取消息,消息传递过程中只需要维持消息队列指针的拷贝,而不需要对被传播的消息本身进行复制.该算法是由 Lamport 在文献[20]中提出的,但是文献[20,35]也都分别指出这种方法只能在内存符合顺序一致性的情况下才适用,也就是说,面对乱序执行的系统,这种模拟方法是无能为力的.并且,对于同一个消息队列被处在不同进程上的生产者和消费者读写时,也会出现多进程的 cache 假共享问题,因此需要对 PDES 同步操作定制特有的通信机制来提升通信效率.

PDES 中逻辑单元经常会发送空消息,这是因为在某个逻辑单元执行进度由于时钟约束无法前进处于等待状态时就会向其他处理单元发送空消息请求,收到空消息的逻辑单元会将最新的时间戳放到空消息中进行返回.这样的操作占用了并行模拟的相当一部分通信开销,发送空消息的请求部件和发送应答消息的应答部件都会产生大量的通信.请求部件发送空消息时可以将自己的时间戳加入到空消息中,如果应答部件发现自己的时间戳小于接收到的空消息所携带的时间戳,就不作应答,否则返回的应答消息不能推动请求部件的执行,请求部件仍然会发送空消息请求.另外应答部件在应答时,如果发现刚刚向请求部件发送了带有最新时间戳的消息,那么对空消息进行应答所携带的时钟消息将是冗余的,通信系统面对这一情况可以直接选择放弃应答,或者如果遇到带有冗余信息的应答信息就直接丢弃.这两种懒惰发送(lazy send)的方式都能够减少同步机制的消息传递,增加了通信的效率.

5 模拟器实例

5.1 串行模拟器

1) Gem5. Gem5[6]是集成 M5 模拟器[36]和 GEMS 模拟器[37]的优点而高度定制化的模拟框架. Gem5

使用C++和python共同开发，其中C++编写底层功能单元，python定制上层模拟器系统，整个开发方式是面向对象的，用户定制也非常方便. Gem5有4种可以相互替换的CPU模拟单元：AtomicSimpleCPU，TimingSimpleCPU，InOrderCPU和O3CPU. 其中AtomicSimpleCPU是最简单的CPU模型，仅能完成功能模拟部分；TimingSimpleCPU可以分析内存操作的时序；InOrderCPU能够模拟流水线操作；O3CPU(OutOfOrderCPU)则能模拟流水线的乱序、并发多线程、超标量等时序模拟. 在指令集架构(ISA)方面，Gem5支持目前主流的X86，ALPHA，ARM，SPARC，MIPS等架构. Gem5的模拟启动方式分为两种，一种是系统仿真调用(simulated emulation)，可以直接模拟二进制应用程序；另外一种是全系统模拟，可以直接启动未经修改的Linux2.4/2.6等操作系统. Gem5的模块化设计使得它在多核模拟上也体现出一定的优势，Gem5不仅可以模拟各个部件，也能模拟部件之间的互连，这使得Gem5模拟器近来受到越来越多的重视.

2) Simics. Simics[14]是全系统模拟的平台，希望在模拟的准确性和效率上保持平衡. Simics支持众多的指令集，并且能够直接运行无修改的Linux，Solaris，Windows等操作系统. 比较特殊的是Simics能够实现反向执行(excute in backward direction)，帮助分析程序的错误(bug)和例外(exception)是如何出现的. 例如在模拟类似Linux的操作系统时，在删除文件之后如果采用反向执行，那么之前删除的文件将会再次出现.

5.2 并行模拟器

1) MPI-SIM. MPI-SIM[38]将传统的空消息传递机制、条件事件协议组合在一起，并且采用了能够减少同步开销的优化手段. MPI-SIM可以根据目标系统的参数给出MPI程序的模拟结果. MPI-SIM是在传统MPI库的基础上加入了同步和统计的功能，使得传统MPI程序不需要进行任何修改就能够调用MPI-SIM库进行执行. MPI-SIM库会将原有的串行程序并行化，主要是将原来的全局变量变为本地变量数组，每个处理单元在访问这些数组时向其传递自己的ID，使得每个处理器访问全局变量时相互不受影响. MPI-SIM中的逻辑处理单元是由多个进程构成，每个处理单元都会维护消息队列、时钟信息等. 其缺点在于只能模拟ISA与宿主机器相同的程序，并且需要提前获得程序的源码.

2) BGLSim. BGLSim[39]是IBM开发的Blue-Gene/L的机器指令级模拟器，运行在真实的Linux机群上，能够模拟多节点的BlueGene/L超级计算机的硬件，在此基础上运行BlueGene/L的Linux系统，其上又移植了BlueGene/L的MPI库，调用这些库的应用程序就可以在被模拟的虚拟机群上运行. BGLSim是全系统模拟器，能够模拟处理器及浮点处理单元、内存映射、设备互连、中断控制器以及互联网络. BGLSim在模拟时需要维持3类线程：bglsim(模拟BlueGene/L的单个节点)，ethgw(处理被模拟节点之间以及被模拟节点和真实网络之间的互连通信)以及idochip(处理被模拟节点和模拟器控制单元之间的互连通信). BGLSim能够在真实机器上完成对512台计算机构成的机群的模拟，并且能够帮助对Blue-Gene/L超级计算机上的软硬件进行模拟和调试.

5.3 模拟平台

1) HLA. HLA(high level architecture)[40]是由美国国防部提供的旨在综合不同种类的模拟实现的服务架构，包含时序并行、空间并行、乐观同步、保守同步等诸多离散事件模拟. HLA的目的是建立开放、可重用的仿真模拟架构，其理念在于在运行时刻架构的基础上提供服务性质的访问接口，这使得HLA的模拟模块和底层服务可以独立开发调优而不需要影响上层的调用，提高整个模拟系统的可重用性. HLA主要偏重于对大规模松耦合系统进行模拟，对于细粒度的高性能模拟并不适用，并且HLA并没有提供实现进程调用的原语和接口，这些接口是细粒度事件驱动模拟效率的关键所在.

2) μsik. μsik[41]的目的在于将PDES技术中的不变部分独立成微核，继而在这些微核上面实现旧有的或者新兴的模拟技术. 这种机制在面对新模拟技术时不需要作出整个模拟系统层面的改变. μsik的思路借鉴于微内核操作系统(microkernel operating system)，微内核系统只提供最基本的服务(进程识别、地址空间等)，而外围的系统服务则在以后添加进来(文件系统、网络协议).

6 研究展望

随着一般处理器和系统中核的数量的增加，未来的模拟工作必将更加艰巨，这也是对模拟器研究方向的指导. 未来的模拟器研究工作会从以下3个方面着手：

1) 整体架构. 现有的串行模拟器还需要不断改

进以提升模拟器的效率,特别是减少时钟级模拟的运行时间,支持更多的架构和操作系统也是串行模拟器不断发展的制导方向.并行模拟器在面临众核系统的模拟工作时,保证模拟的效率是一定要考虑的问题,如何维持处理单元之间的执行顺序也是需要认真思考的问题,这对于程序调试、错误容忍等工作都有着十分重要的价值.另外在模拟器的设计中加入足够多的接口并反馈足够多的调试信息,方便使用者对模拟器进行定制,并深入理解程序执行的各种细节信息及统计结果,这是模拟器设计者需要考虑的问题.

2) 设计实现.由于串行模拟技术的成熟,以及并行模拟对于多线程程序和多核系统模拟中的良好表现,越来越多的研究工作专注于并行模拟器的开发与并行技术的优化.其中,负载均衡的机制虽然已经被广泛地研究,但是对于各种负载均衡机制对于同步的影响,例如静态负载均衡对于时钟弯曲中撤销和空消息数量的影响就是一个非常需要思考的问题.何时使用静态负载均衡,何时使用动态负载均衡,如何确定这样的均衡方案也需要一定的工作支撑.而关于同步机制的研究兴趣也丝毫未减,特别是对于乐观同步策略,如何减少回滚算法在时间和空间上的额外开销是很有吸引力的研究方向,现有的策略包括复用存储[32]、选择记录[33]、设立滑动窗口等都被提出,如何在实际模拟过程中动态选择这些策略,或者自适应调整参数[34]已经成为不少研究者的着眼点,国内的单磊等人在文献[42]中提出的基于指令语义映射的原子指令模拟技术对于实现逻辑单元之间的同步也有着独到的见解.分布式计算机发展得如火如荼,并行模拟器的多处理单元模型在结构上与分布式计算机也有众多相似之处,如何将分布式计算机上关于负载均衡和消息通信的机制应用到并行模拟中也是值得思考的问题.

3) 模拟器在其他研究领域内的应用.在软件调试方面,如多线程程序的确定性重放工作是近年来的研究热点,而模拟器由于软硬件架构便于修改、调试信息丰富等优势可以成为确定性重放的重要研究工具.现有的确定性重放机制分为软件实现和硬件实现,对于硬件实现方案,由于模拟器在时序模拟过程中能够提供足够详细的硬件信息,并且模拟器的模拟部件也可以进行模块化的定制,这都为确定性重放的硬件方案提供了诸多便利.而在系统调试方面,现在的异构多核处理器的优化设计是一些研究者关注的重心,真实实验中定制的异构多核芯片需要从底层进行设计制造并且代价相当昂贵,而无论是并行模拟器还是串行模拟器都能够对处理器核心的参数进行独立的定制,对于异构多核芯片的设计方案是非常方便的验证方案,模拟器设计者可以考虑提供更加开放的接口和可定制性,甚至可以设计专有的异构多核系统模拟器来验证异构多核模拟器设计方案,或者设计并优化专有的片上网络模拟器来比较各种新兴的片上网络通信算法等.

7 结束语

本文首先对模拟器现有的发展进行简介,然后介绍了模拟器的架构和分类,接着从串行模拟和并行模拟两个方面对现有的模拟优化技术进行总结归纳,最后对模拟器的发展前景进行了展望.

未来对于多核乃至众核系统的模拟工作必将成为研究的热点方向,研究人员需要考虑从并行的实现技术出发提升模拟效率,如减少现有时钟弯曲机制的回滚次数或者提出更好的同步策略;另外,模拟器研究人员与设计者也需要考虑将模拟技术和现有的体系结构研究前沿技术相结合,方便研究工作的展开,反过来实现中遇到的问题与解决方案也能完善和优化模拟器的结构设计.

参 考 文 献

[1] Bell S, Edwards B, Amann J, et al. tile64-processor: A 64-core soc with mesh interconnect [C] //Proc of ISSCC'2008. Piscataway, NJ: IEEE, 2008: 88-598

[2] Howard J, Dighe S, Hoskote Y, et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos [C] //Proc of ISSCC'2010. Piscataway, NJ: IEEE, 2010: 108-109

[3] Li Yue, Qian Depei. A distributed parallel network simulator based on NS [J]. Acta Electronica Sinica, 2004, 32(2): 246-249 (in Chinese)
(李越,钱德沛. 基于NS的分布式并行网络模拟器[J]. 电子学报, 2004, 32(2): 246-249)

[4] Eyerman S, Eeckhout L, Karkhanis T, et al. A mechanistic performance model for superscalar out-of-order processors [J]. ACM Trans on Computer System, 2009, 27(2): 3:1-3:27

[5] Rosenblum M, Bugnion E, Devine S, et al. Using the simos machine simulator to study complex computer systems [J]. ACM Trans on Modeling Computer Simulation, 1997, 7(1): 78-103

[6] Binkert N, Beckmann B, Black G, et al. The gem5 simulator [J]. SIGARCH Computer Architecture News, 2011, 39(2): 1-7

[7] Bellard F. Qemu, a fast and portable dynamic translator [C] //Proc of the 14th USENIX Annual Technical Conf. Boston, USA: FREENIX Track, 2005: 41-46

[8] Jaleel A, Cohn R, Luk C K, et al. Cmp $im: A pin-based on-the-fly multi-core cache simulator [C] // Proc of the 4th Annual Workshop on Modeling, Benchmarking and Simulation. Piscataway, NJ: IEEE, 2008: 28-36

[9] Ganger G, Worthington B, Patt Y. The disksim simulation environment version 2.0 reference manual [R]. Pittsburgh, PA: Carnegie Mellon University, 1999

[10] Wang D, Ganesh B, Tuaycharoen N, et al. Dramsim: A memory system simulator [J]. ACM Sigarch Computer Architecture News, 2005, 33(4): 100-107

[11] Bohrer P, Peterson J, Elnozahy M, et al. Mambo: A full system simulator for the powerpc architecture [J]. ACM SIGMETRICS Performance Evaluation Review, 2004, 31(4): 8-12

[12] Yourst M. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator [C] //Proc of the 8th Performance Analysis of Systems & Software. Piscataway, NJ: IEEE, 2007: 23-34

[13] Bell J. Threaded code [J]. Communications of the ACM, 1973, 16(6): 370-372

[14] Magnusson P, Christensson M, Eskilson J, et al. Simics: A full system simulation platform [J]. IEEE Computer, 2002, 35(2): 50-58

[15] Bansal S, Aiken A. Binary translation using peephole superoptimizers [C] //Proc of OSDI'2008. New York: ACM, 2008: 177-192

[16] Hookway R, Herdeg M. Digital fx!32: Combining emulation and binary translation [J]. Digital Technical Journal, 1997, 9(1): 3-12

[17] Chiou D, Sunwoo D, Kim J, et al. Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators [C] //Proc of MICRO'2007. Piscataway, NJ: IEEE, 2007: 249-261

[18] Patterson D. Ramp: Research accelerator for multiple processors—A community vision for a shared experimental parallel hw/sw platform [C] //Proc of ISPASS. Piscataway, NJ: IEEE, 2006: 1-1

[19] Fujimoto R. Parallel discrete event simulation [C] //Proc of the 21st Conf on Winter Simulation. New York: ACM, 1989: 19-28

[20] Lamport L. Time, clocks, and the ordering of events in a distributed system [J]. Communications of the ACM, 1978, 21(7): 558-565

[21] Andersen R, Lang K. An algorithm for improving graph partitions [C] //Proc of SODA 2008. New York: ACM, 2008: 651-660

[22] Chandy K, Misra J. Distributed simulation: A case study in design and verification of distributed programs [J]. IEEE Trans on Software Engineering, 1979, 5(5): 440-452

[23] Jefferson D. Virtual time [J]. ACM Trans on Programming Languages and Systems, 1985, 7(3): 404-425

[24] Nicol D. Automated Partitioning of Simulations for Parallel Execution [M]. Virginia: Department of Computer Science, School of Engineering and Applied Science, University of Virginia, 1985

[25] Nandy B, Wayne M L. An algorithm for partitioning and mapping conservative parallel simulation onto multicomputers [C] //Proc of the 6th Workshop on Parallel and Distributed Simulation. San Diego: Society for Computer Simulation, 1992: 139-146

[26] Briner J. Parallel mixed-level simulation of digital circuits using virtual time [D]. NC, USA: Duke University, 1990

[27] Reiher P, Jefferson D. Virtual time based dynamic load management in the time warp operating system [J]. Trans of the Society for Computer Simulation, 1990, 7(2): 91-120

[28] Glazer D. On process migration and load balancing in time warp [J]. IEEE Trans on Parallel and Distributed Systems, 1993, 4(3): 318-327

[29] Boukerche A, Das S. Reducing null messages overhead through load balancing in conservative distributed simulation systems [J]. Journal of Parallel and Distributed Computing, 2004, 64(3): 330-344

[30] Fujimoto R. Exploiting temporal uncertainty in parallel and distributed simulations [C] //Proc of the 13th Workshop on Parallel and Distributed Simulation. Piscataway, NJ: IEEE, 1999: 46-53

[31] Dickens P, Reynolds P. SRADS with Local Rollback [M]. Charlottesville, Virginia: Institute for Parallel Computation, University of Virginia, 1990

[32] Lin Y B, Preiss B. Optimal memory management for time warp parallel simulation [J]. ACM Trans on Modeling and Computer Simulation, 1991, 1(4): 283-307

[33] Palaniswamy A, Philip A W. An analytical comparison of periodic checkpointing and incremental state saving [J]. ACM SIGSIM Simulation Digest, 1993, 23(1): 127-134

[34] Das S, Fujimoto R. Adaptive memory management and optimism control in time warp [J]. ACM Trans on Modeling and Computer Simulation, 1997, 7(2): 239-271

[35] Gharachorloo K, Phillip B G. Detecting violations of sequential consistency [C] //Proc of the 3rd Annual ACM Symp on Parallel Algorithms and Architectures. New York: ACM, 1991: 316-326

[36] Binkert N, Dreslinski R, Hsu L, et al. The m5 simulator: Modeling networked systems [J]. IEEE Microsoft, 2006, 26(4): 52-60

[37] Martin M, Sorin D, Beckmann Bradford, et al. Multifacet's general execution driven multiprocessor simulator (gems) toolset [J]. ACM SIGARCH Computer Architecture News, 2005, 33(4): 92-99

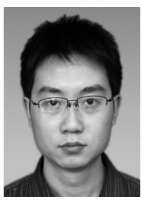
[38] Prakash S, Bagrodia R. Mpi-sim: Using parallel simulation to evaluate mpi programs [C] //Proc of the 30th Conf on Winter Simulation. Los Alamitos, CA: IEEE Computer Society, 1998: 467-474

[39] Almasi G, Bellofatto R, Brunheroto J, et al. An overview of the bluegene/l system software organization [J]. Parallel Processing Letters, 2003, 13(4): 561-574

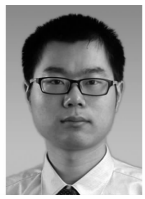
[40] Dahmann J, Fujimoto R, Weatherly R. The department of defense high level architecture [C] //Proc of the 29th Conf on Winter Simulation. Los Alamitos, CA: IEEE Computer Society, 1997: 142-149

[41] Perumalla K S. μsik-a micro-kernel for parallel/distributed simulation systems [C] //Proc of the 19th Workshop on Principles of Advanced and Distributed Simulation. Piscataway, NJ: IEEE, 2005: 59-68

[42] Shan Lei, Zhao Tianlei, Tang Yuxing, et al. Research on efficient execution technique of atomic instructions in parallel simulation [C] //Proc of the 16th CCF Annual Conf on Computer Engineering and Technology. Hunan: National University of Defense Technology Press, 2012: 548-553 (in Chinese)
(单磊, 赵天磊, 唐遇星, 等. 并行模拟器中原子指令的高效执行技术研究[C] //第十六届计算机工程与工艺年会暨第二届微处理器技术论坛论文集. 长沙: 国防科学技术大学出版社, 2012: 548-553)



Liu Yuchen, born in 1987. PhD candidate. His main research interests include high performance computer architecture and machine learning.



Wang Jia, born in 1990. Received his master's degree from the Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include high performance computer architecture and machine learning.

Chen Yunji, born in 1983. Professor and PhD supervisor in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include computer microarchitecture, machine learning and parallel computing.

Jiao Shuai, born in 1984. PhD candidate. His research interests include high performance computer architecture and parallel system.