体系结构模拟器的技术和发展

许建卫 杨伟 潘晓雷 郑规 赵健博 陈明宇

摘要 模拟器是体系结构研究和设计的重要工具，在学术界和工业界得到广泛使用。软件全系统模拟器具有开发速度快、成本低、易于修改等特点，因此在计算机系统设计、验证，系统软件开发、调优等方面都可以发挥重要作用。理想的模拟器应该具有运行速度快、模拟精度高、配置修改灵活等特点，但这三个要素是相互矛盾的，很难兼得。

模拟器的三个要素中，速度尤为重要，是制约模拟器使用的最主要的因素，因此人们从模拟器的开发和模拟器的运行两个层次采取技术措施来提高模拟器速度。本文重点对这些技术进行介绍。除了介绍模拟器加速技术外，本文还对提高模拟器灵活性、提高模拟器精度的技术做了描述，并分析了一些常见的模拟器，从中可以看出各种技术在实际模拟器中的使用。最后，文章对目前模拟器的发展方向做了简单描述。

1 概述

模拟器在计算机系统的设计开发过程中有非常重要的作用：设计初期，模拟器可以用来对各种设计方案进行粗粒度模拟，通过比较模拟结果来选择最优设计方案；产品开发期间，模拟器用来对各种微结构设计进行评估，对一些选择进行折衷；产品开发后期，模拟器主要用来进行目标系统的系统软件开发，使得软硬件开发可以同时进行，加快系统开发速度；系统完成之后，模拟器可以取得丰富的踪迹（trace）信息，从而对系统进行瓶颈分析和性能优化。由于模拟器具有上述重要作用，学术和工业界都开发了大量的模拟器。常见的研究用的模拟器有 SimpleScalar[1]、SimOS[2]、M5[3]、Liberty[4]等，各个公司也都开发自己的模拟器，比如 IBM 公司的 Mambo[5]模拟器、AMD 公司的 SimNow[6]等。

模拟器有三个非常重要的衡量指标：速度、精度和灵活性。速度指模拟器执行模拟任务的快慢，通常用模拟器上执行应用和在宿主机上执行相同应用的时间比来表示，这个时间比通常称为减速比(slowdown)；精度指模拟器模拟出的目标系统和实际系统的接近程度，CPU 模拟器常用模拟得到的每条指令所用周期数（CPI，cycles per instruction）和实际值偏离的百分比来表示；灵活性指模拟器通过配置来模拟不同结构的方便程度及是否具有多种执行模式和灵活可配置的数据收集、显示方式。一个理想的模拟器要求执行速度快、模拟器精度高并且可以灵活地配置出想要的结构，但实际中速度、精度和灵活性这三者存在相互制约的关系，很难同时兼得。通常来说，抽象程度越高、模拟的速度就越快，但模拟的精度也越差；增加模拟精度则增大了模拟负载，带来模拟速度的下降，并且由于模拟的粒度变细，灵活性也会受到影响。因此，根据不同的需要，模拟器在这三个因素中往往有所侧重。一般来说，设计初期，灵活性要求较高，而精度要求较低。在系统的后期设计开发过程中，对精度的要求逐渐提高，而对灵活性的要求则逐渐降低。但模拟器的速度则贯穿整个设计开发的始末，也是制约模拟器使用的最重要的因素。

软件模拟器具有模拟精度高、开发成本低、修改配置灵活、开发周期短等特点而受到研究和开发者的喜爱。软件模拟器中全系统模拟器还支持开发运行操作系统、获取应用程序丰富的踪迹等功能，因此全系统模拟器得到了广泛的应用。本文重点对全系统软件模拟器进行分析，并对常见的硬件模拟器和局部模拟器做介绍。

2 常见术语

2.1 系统/局部模拟器

系统模拟器对整个计算机系统都进行模拟，通常模拟处理器（CPU）、缓存（Cache ）、内存管理单元（MMU，Memory Management Unit）、内存、中断控制器、磁盘以及直接内存存取（DMA，Direct Memory Access）控制器、以太网等，由于系统模拟器除了提供完整的指令集架构（ISA，instruction set architecture）外，也可对外设模拟，所以一般都可以启动操作系统，很多甚至可以启动不加修改的操作系统。系统模拟器除了可以用在对设计方案进行选择之外，还可以进行产品开发、系统调优等。常见的系统模拟器包括 SimOS[1]、SimICS[7]、Mambo[5]、QEMU[8]、SandUPSim[9]、SimOS-Goodson[10]等。

局部模拟器指对整个计算机系统的某个部件进行细致模拟，提供多种配置参数，可以模拟多种类型的部件。局部模拟器由于无法运行真实的应用，所以通常以踪迹为输入。常见的局部模拟包括缓存模拟器[11]、磁盘模拟器 DiskSim[12]、网络模拟器 ns2[13]、能耗模拟器 Wattch[14]等。

2.2 功能级/时钟级模拟

功能级模拟器指模拟器仅保证目标系统的指令集架构的模拟正确，认为所有的指令（包括访存指令）都是一个模拟周期执行完毕，忽略访存延迟、缓存以及现在微处理器指令级并行(ILP，Instruction Level Parallelism)技术的影响。由于功能级模拟器不需要提供详细的指令执行周期信息，所以可以省去缓存等模块的模拟，并且对 CPU、内存的模拟可以简化，因此方便了模拟器的开发，并提高了模拟器的运行速度。功能级模拟器主要用在系统软件开发、踪迹信息获取等方面。

时钟级模拟则要在保证指令集架构正确的基础上，对处理器微结构中的流水线、多发射、乱序执行、分支预测等操作和对内存系统中的缓存读写操作进行模拟，每条指令的执行周期随系统的状态不同而变化。时钟级模拟比较真实地模拟了目标系统的运行状态，主要用在系统性能分析等方面。

功能级模拟追求的是速度，时钟级模拟则更注重精度。为了达到二者的统一，很多模拟器同时提供功能级模拟和时钟级模拟两种模式，并且这两种模式在运行时可以动态切换。使用者可以在不需要时钟相关信息的阶段使用功能级模拟，需要时再切换到时钟级模拟。常见的同时提供功能级模拟和时钟级模拟的模拟器包括 SimOS[2]、M5[3]、RSIM[15]等。

2.3 执行/踪迹驱动

执行驱动（一些文献中也称作程序驱动）是指模拟器的输入是目标系统二进制代码，因此可以在执行驱动的模拟器上运行完整的操作系统或应用程序。而踪迹驱动则是指模拟器的输入是一些事先得到的踪迹，这些踪迹可以是真实系统或执行驱动的模拟器上运行程序的踪迹，也可以是按某种分布函数产生的伪踪迹。一般全系统模拟器都是执行驱动的，而局部模拟器往往是踪迹驱动。

踪迹驱动的优点在于可以对局部进行细致地研究而不用模拟其他无关部分，可以提高模拟速度。但踪迹本身过滤了系统的动态信息，如分支误预测等，因此无法观察到系统的动态特征。而且由于踪迹的数据是一些固定序列，当模拟器的参数改变时无法提供反馈信息。

2.4 时钟/事件驱动

前述的执行/踪迹驱动主要指模拟器的输入，这里的时钟/事件驱动主要是指模拟器的推进方式。时钟驱动是在模拟器中维护一个软件时钟，每个时钟周期调用部件模拟函数模拟一步，比如功能级模拟器的 CPU 模拟模块在每个周期读取一条指令，然后解码执行。事件驱动则是事先将所有的事件按时间戳排列成一个列表，每次从表头取出一个事件进行处理，处理过程中产生的新的事件按时间戳顺序重新插入到列表中。

时钟驱动模拟的方式适合于系统时钟步进比较连续的模拟器，比如常见的全系统模拟器都是采用时钟驱动的方式。而事件驱动的模拟器则适合于系统时钟推进不连续的模拟器或模拟框架，比如 Asim[16]等。

2.5 用户态模拟

用户程序的执行过程中需要操作系统提供系统调用支持，如果模拟器可以提供系统调用支持，那么用户程序就可以直接在模拟器上运行。在这种情况下，模拟器不需要模拟目标系统的特权指令，因此称之为用户态模拟。用户态模拟的优点是不用模拟特权指令，不用启动操作系统就可以运行应用；缺点是需要在模拟器上为用户程序进行内存管理、维护页表等，而且有些依赖于操作系统的系统调用无法模拟，如 fork 等。支持用户态模拟的模拟器包括 Augmint[17]、Mambo[5]、QEMU[8]、SESC[18]、g88[19]等。

3 模拟器实现技术

如前所述，模拟器的速度、精度和灵活性是模拟器的三个要素，人们在这三个方面都在采取技术措施来进行提高，下面分别讨论这三方面的工作。

3.1 加速技术

模拟器的速度是制约模拟器使用的最主要因素，因此人们提出了很多模拟器的加速手段，主要包括单机加速、并行模拟、现场可编程门阵列（FPGA）加速等。

3.1.1 单机加速

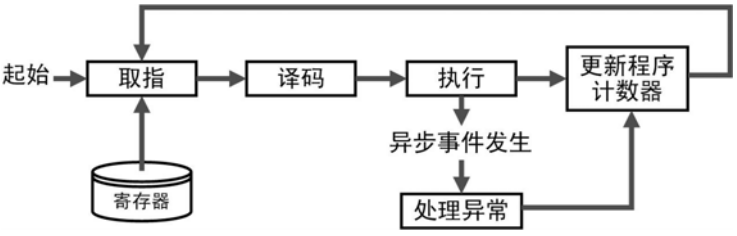

图1. 解释型模拟器

传统模拟器通常是用高级语言（比如 C 或者 C++）写成的解释器，具有很好的灵活性和可移植性。解释器易于实现，而且可以做到和目标机较好的兼容，但是它的速度很慢，不适合运行大型应用。如图 1 所示，解释器循环执行“取指→译码→执行”的步骤：首先根据程序计数器（Program Counter，PC）从模拟内存当中取得一条指令，对它进行译码找到对应的模拟函数，然后调用这个函数来更新状态。系统级模拟器在每条指令模拟完毕后还要判断是否有中断、异常等异步事件发生，若有则需要处理它，否则修改 PC 使它指向下一条指令，进入下一次解释循环。

这种方式的优点是非常直观，和指令语义有着严格的对应关系，简单易实现，适合快速实现一个原型系统。它的缺点是开销大，速度慢。开销主要包括：

(1) 每次取指令至少需要一次访问模拟内存：首先取得程序计数器的值，接着从模拟内存当中读入程序计数器指向的指令。如果是系统级模拟，每次内存操作还需要经过

极为费时的虚实地址转换，开销就更大了；

(2) 指令译码：对于复杂指令集的每条指令，每次都要从中提取出需要的位域，需要不少的逻辑/移位运算；

(3) 对于大的 switch-case 语句，编译器可能会把它编译成跳跃表的形式，这个时候就会多了一次边界检查和一次内存操作才能取到 case 程序段的入口地址；

(4) 一次跳转操作，到对应的 case 程序段，可能还需要一次函数调用来模拟指令功能；

(5) 在 case 程序段执行完毕后，需要再进行一次跳转操作，回到循环开头。

在上述过程中，真正有效的是指令模拟部分（步骤 4），但指令调度部分（步骤 1,2,3,5）占据了很大的比例。调度部分产生的机器指令中有很多的跳转指令和内存操作指令。在现代处理器当中，一般设计有很深的流水线，跳转操作会打断流水线，对性能影响很大。此外内存操作也是十分费时的，所以这类模拟器性能优化的重点在于减少内存操作和跳转指令。经常采用的技术有：

穿线码技术[1]（threaded code）

threaded code 技术通常用于基于解释执行的模拟器，是由贝尔（J.R. Bell）[20]在 1973 年提出的一种程序设计技术，可以用来实现高效的解释器。穿线码的主要思想是针对上述解释型模拟器的缺点，采用一次翻译多次执行的方式来避免重复的取值、译码过程。

如图二所示，穿线码技术首先将目标指令翻译成中间代码，中间代码中存放的是该指令的解释函数的地址和该指令译码后的操作数。翻译后的中间代码以基本块[2]的方式放入穿线码引擎中。在之后模拟器执行过程中，对于一条指令的模拟首先在穿线码引擎中查找是否已经存在，如果不存在，说明这条指令以前没有翻译过，则调用翻译过程对该基本块进行翻译并存入穿线码引擎中。如果已经存在，则直接进入穿线码引擎中执行该基本块。

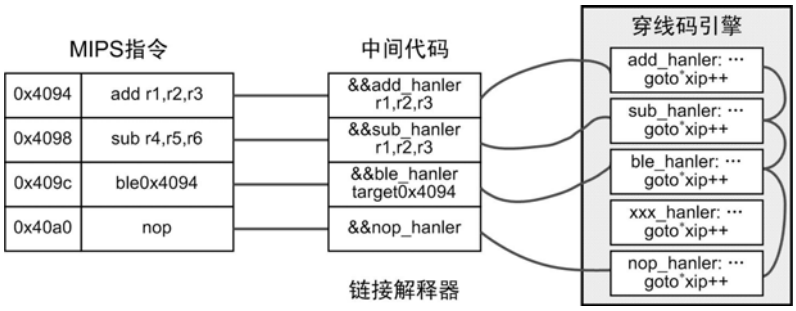

图2. 穿线码模拟器

穿线码技术将指令翻译一次之后多次执行，并且仅在每个基本块执行结束的时候才切换到解释态，判断下一个基本块是否已经翻译过，相比纯解释型模拟器减少了状态切换次数，大大提高了执行效率。由于每条指令的模拟仍通过解释的方式来实现，因此穿线码技术保持了解释型模拟器易读、移植性好并可以取得丰富的踪迹的特点，并取得了很低的减速比。采用这种技术的代表性模拟器有 Talisman[21]和 SimICS[7]。

二进制翻译

二进制翻译（binary translation）是把一个体系结构上的二进制代码翻译成另外一个体系结构上的二进制代码的过程，它已经不再是解释型模拟器，而是直接使用宿主机的一条或几

[1] thread code 迄今为止没有权威的中文翻译，有译作“链接代码”的，本文作者译为“穿线码”，欢迎读者提出自己的译法。意见可通过本刊封底的地址和电子邮箱发送给我们—编者。

[2] 基本块：具有单一入口和出口的一段连续指令码

条指令来模拟目标系统的一条指令。基本原理如下：

(1) 将一条目标机指令分成若干微操作，精心定义微操作集，使得这些微操作的数目不多，但是能够描述指令的基本动作。定义每条指令和这些微操作之间的映射关系，这部分和体系结构密切相关；

(2) 使用简单的 C 函数实现每个微操作，并用 C 语言编译器（gcc）编译成目标文件。然后分析这些目标文件，产生一个动态代码产生器, 动态代码产生器的作用是遇到一条目标指令时，将其对应的微操作组合成对这条指令的模拟；

(3) 分析工具分析每个微操作的代码，去除其函数头尾不需要的部分，找出真正需要的代码，告诉代码产生器如何拷贝、连接代码；

(4) 代码产生器将微操作代码以基本块为单位拷贝到代码缓存中，以后的执行都在代码缓存中进行。

二进制翻译技术相比穿线码更进一步提高了模拟器的执行速度，但由于每条目标指令的执行使用了宿主机指令来代替，因此好多踪迹信息无法收集。并且针对不同的宿主机指令集架构，二进制翻译模块都要做相应的支持，可移植性也进一步降低。使用二进制翻译技术的模拟器有 SimOS[2]、Qemu[8]等。

直接执行

如果模拟的目标机和宿主机指令集架构相同，则可以采用直接执行的方法来加速模拟器运行。直接执行是指将模拟器中的指令模拟直接调用宿主机的指令来实现。由于直接使用宿主机的指令来执行目标指令，所以直接执行的优点在于执行速度大大加快，但缺点是由于模拟器是用户态程序，无法直接执行特权指令，所以对特权指令仍需要进行模拟。而且模拟器运行时在直接执行态和模拟态之间的切换也将带来开销。由于直接执行需要目标机和宿主机指令集架构相同，所以模拟器的可移植性差。直接执行技术可以看作是二进制翻译技术的一种特例，使用直接执行技术的模拟器有 vmware[22]、simICS[7]等。

剖析信息指导（profile directed）

程序运行时的初始化代码可能仅仅执行一次，对这些仅执行一次的代码采用穿线码和二进制翻译技术是没有用处的，并且翻译开销会降低模拟器的性能。因此基于“加速最常用的部分”的基本原理，人们采用运行时分析（runtime profiling）的方法选择执行频率高的基本块作为翻译对象。分析（Profiling）工具收集程序执行信息，如基本块执行频率，分支行为，访存模式等，用于发现热点基本块，然后模拟器再将这些热点基本块进行线索化或者二进制翻译，从而有效地提高模拟器的模拟速度。

时序优先（Timing-first）

前述技术仅仅适用于功能级模拟器，忽略了模拟器模拟的目标系统的时间信息。时钟级模拟器如果将功能模拟和时序模拟结合在一起，则模拟器的实现过于复杂，调试维护难度增大。因此，卡尔（Carl, J.M.）[23]等提出了将功能模拟和时序模拟分离开来，时序模拟主要是提高模拟器的精度，而功能模拟主要是保证模拟的正确性。主要思想是时序模拟模块模拟每个部件的微结构，根据微结构当前状态向功能级模拟器发出执行命令，然后功能模拟模块完成具体的功能模拟。时序优先技术将功能模拟和时序模拟分离，优点是可以利用现有的功能级模拟器快速开发一个时钟级模拟器，并且使得模拟器的结构更加清晰，减轻了开发和调试的负担。使用时序优先技术的模拟器有 TFsim[23]，MASE[24]等。

3.1.2 并行加速

近年来，计算机系统规模正在发生重大变化，单处理器向多核方向迅猛发展，超级计算机节点数目也在指数级增加。因此如果使用传统的串行模拟器来进行大型复杂系统的详尽模拟往往需要几周甚至几个月才能完成一次模拟；如果要缩短模拟时间，则需要降低对模拟粒度和准确度的要求，或者辅之以分析手段，在模拟时间和保真度之间难以达到可以接受的平衡。另一方面，运行模拟器可用的宿主机规模越来越大，需要模拟的目标系统也越来越大，传统的串行模拟器已经成为中间的瓶颈，因此亟需建立并行模拟器以充分利用硬件资源，在可接受的时间内完成特定规模和精度要求的模拟。

并行模拟器的难点在于模拟的各个逻辑单元之间的同步消耗了大量时间。当目标系统规模变大，逻辑单元数目大大增加，事件同步消耗的时间可能远大于计算所需时间。对于并行模拟器的同步，藤本（Fujimoto）[25, 26]等提出了并行离散事件仿真（PDES， Parallel Discrete Event Simulation）同步算法，但尽管人们已经给出了很多有关并行离散事件仿真的算法，只有少部分算法理论上是真正可扩展的，而其中被实践验证可行的则更少。

并行模拟器模拟的目标系统也是并行系统，而运行在并行系统上的应用程序本身维持了同步关系来确保程序运行的正确性。因此，功能级模拟器不需要各并行部分之间的同步就可以实现。BGLsim[27]就是采用这种技术实现了大规模并行模拟器。

3.1.3 现场可编程门阵列加速

由于现场可编程门阵列具有易于编程、运行速度较快的特点，因此可以用来进行局部模拟或系统模拟。其优点是既保留了软件模拟器的灵活性，又弥补了软件模拟器运行速度较慢的不足。但缺点是开发实现复杂，并且相比软件模拟器价格较高。

目前使用现场可编程门阵列进行全系统模拟的有 RAMP[28]，使用现场可编程门阵列进行局部模拟的有 STIMUL[29]、FAST[30]等。

3.1.4 其他加速技术

人们使用模拟器往往是为了对程序运行过程中关键的部分进行研究，对程序运行的其他阶段并不感兴趣。为了快速达到感兴趣的点，人们使用了快速推进（FFW，Fast Forwarding）和检查点重启动（checkpoint restart）技术。

快速推进

快速推进技术是在模拟器运行初期使用功能级模拟的模式，使模拟器快速推进到感兴趣的点，然后切换到详细模拟的模式进行详细的模拟。由于详细模拟对系统的分支预测单元、缓存等模块都进行模拟，而快速推进阶段并没有对这些模块进行预置（warmup 或 warming），因此详细模拟阶段的初始状态可能是不正确的，快速推进技术的描述见 [31, 32]。

检查点重启动

借助于检查点重启动技术，可以在模拟器运行到感兴趣的点之前设置检查点，然后修改模拟的参数后让模拟器从当前点重新执行。巴尔（Barr）[31]提出了内存时间戳记录（memory timestamp record, MTR）技术来加速多处理器模拟中的缓存和目录的预置（warmup）过程。

3.2 模拟器的灵活性

模拟器的灵活性需求有两个方面：一是期望用一个模拟器在一个范围内能尽可能地模拟各种不同类型不同规模的目标系统；另一是对于某一个特定的目标系统，如果各模块或者组件可灵活地替换成不同保真度的版本，则模拟器使用者就可以在模拟时间和精确度之间进行权衡。为了实现模拟器的灵活性，通常采用下列技术：

模块化设计

将模拟器的每个模拟模块都遵照严格的接口来定义，每个模块内部结构的改变不影响其对外接口。这样可以方便地用同一模块实现不同详尽程度的模拟，还可进一步把模块的实现加入软件模拟库中，需要配置一个新的系统时，只需要从库中选择需要的模块组在一起，就可以形成想要的系统。这种方法和实际硬件类似，一个大的系统通过一系列标准的组件组合而成。这种方法开发的模拟器和硬件具有天然的相似性，设计中都非常注重模块化，如Liberty[4]、Asim[16]、GEMS[33]等均是如此。

踪迹采集和分析分离

收集踪迹是模拟器的一个重要功能，但原始的踪迹一般数据量大，信息也晦涩难懂。如果将踪迹分析功能集成在模拟器中，会使得模拟器代码复杂，模拟器的运行速度变慢，并且给踪迹信息的定制带来了困难。SimOS[2]中由模拟器来产生踪迹信息，使用离线的踪迹分析工具来对踪迹进行分析。这样用户可以方便地定制踪迹信息，并可以将信息以更加易读的形式表现出来。而 Mambo[5]中则由模拟器产生数据到专门的内存区，由其他线程从内存区中读出数据并进行分析。

并行模拟框架

对于时钟级精确的并行模拟器，各个模拟单元之间需要进行通信和同步，而通信和同步功能和模拟器模拟的具体内容关系比较松散，因此可以将并行模拟器中的通信和同步等功能单独实现，其他的部件模拟模块通过调用并行框架提供的应用程序接口（API，Application Programming Interfaces）来实现并行模拟。这种方法使得模拟器编写者不需要考虑并行模拟器如何部署、同步和通信，并且模拟模块的修改不会对并行模拟框架带来影响，具有非常好的灵活性。常见的并行框架有 USSF[34]、HLA[35]等。

3.3 模拟器精度

模拟器的精度表示该模拟器的模拟执行结果和模拟所得到的性能与真实机器接近的程度。模拟的精度受两方面影响：模拟器的模型是否与真实设计相匹配以及模型的输入数据是否真实。对于已有系统，模拟器的精度可以通过和实际系统运行结果对比，对于正在实现的系统可以通过和 VHDL（very high speed integrated circuit hardware description language，超高速集成电路硬件描述语言）代码模拟结果对比，但模拟器更多模拟的是未来的系统，因此无法对模拟器的精度进行验证。然而模拟器的一个重要应用是对各种设计方案进行比较，而易（Joshua J.Yi）[36]指出，一个模拟器的绝对误差是无法避免的，但模拟结果和真实系统的偏差是稳定的，因此同一个模拟器对不同系统的模拟结果的相对值仍旧可以相对反映这些目标系统性能上的差异。

影响模拟器运行结果的因素很多，Lipasti[37]对三种重要的因素进行了分析：1)操作系统的影响。操作系统不仅会对商业应用的模拟结果有较大的影响，而且会影响 SPEC 这样的基准测试集；2）那些被不正确预测的路径对模拟结果影响不如人们预期的那么大，因为预测错误对性能的影响有正有负；3）是否正确地模拟 I/O（输入/输出）行为可以影响模拟的准确性，例如在单处理器并且 I/O 带宽足够的情况下，一个可以正确反映由于 I/O 写操作造成的缓存行无效的直接内存存取引擎模型也会对缓存的不命中率模拟结果产生 2%左右的影响，对性能产生 1%的影响。

3.4 模拟器并行化技术

为了充分利用现有单 CPU 多核或多处理器系统资源，模拟器需要并行。但由于模拟器是细粒度并行程序，所以并行最大的困难在于多线程或多进程之间的同步。

一种方法是使用一个管理者模块来统一管理系统时钟，每个线程更新局部时钟之后通知管理者，管理者等到所有线程都更新了时钟之后，推进全局时钟，并把全局时钟广播到系统中所有线程。这种方法每个线程更新时钟后都要等待系统时钟的更新，等待延迟较大。而且集中控制的方式容易使得管理者线程成为系统的瓶颈，扩展性不高。

藤本[26]等提出了并行离散事件仿真技术，其核心思想是系统中如果任何一个局部都是同步的，则整个系统是同步的。如图三所示，假定节点 m 仅和节点 i、j、k 相连，那么节点 m 仅需要和节点 i、j、k 保持同步即可。这样整个系统中的任何一个部分都是同步的，那么全系统也就同步。这种方式避免了集中控制的缺点，可扩展性好。

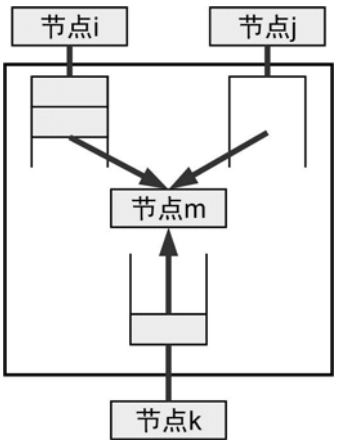

图3. PDES 同步

为了实现系统同步，那么节点 m 每次从节点 i、j、k 的输入队列中挑选一个时间戳最小的消息进行处理，并按时间戳保序的方式向外发送消息。但如果处理过程中发现一个节点输入队列为空，如图三中的节点 j，那么节点 m 是否可以从节点 i、k 的输入队列中选择一个消息进行处理？可以有两种决策方式：保守同步和乐观同步。保守同步严格按照事件的因果关系来执行，因此对于图三的情况，要等待节点 j 发送一个消息或主动发送查询消息；乐观同步则采用“错误检测和回滚”方法，对于图三所示的情况，节点 m 继续进行处理。如果之后节点 j 发来一个过时消息，则对已经处理的消息进行回滚。

这两种同步，各有很多的优化和提高的机制。总的来说，保守机制适用于节点的 lookahead[3]值较大，并且系统状态保存开销大的情况，而乐观机制则刚好相反。由于模拟器状态保存开销较大，所以大部分使用保守并行离散事件仿真机制。使用并行离散事件仿真技术的模拟器有 WWT[38]和 WWTII[39]等。

4 模拟器运行技术

4.1 常用基准测试程序

早期模拟器更多地用来对 CPU 微结构进行研究，因此常用 SPEC CPU[40]来作为测试程序。近来模拟器也逐渐用来模拟大规模系统。对于大规模系统，人们常用 NPB[41]来作为测试程序，对于共享内存的系统，也使用 SPLASH[42]测试程序。

SPEC CPU

SPEC（Standard Performance Evaluation Corporation）发布的 CPU 测试集是体系结构研究者最常用的基准测试程序（benchmark），包括 SPEC CPU 89、92、95、2000 和 2006。这些基准测试程序来自硬件厂商、软件厂商、大学以及用户等，是从真实应用中抽取出的核心例程，因此非常具有代表意义。

SPEC CPU 不仅仅测试 CPU 的性能，还测试内存和编译系统的性能，但不测试 I/O、网络和图形设备。为了实现这个目标，SPEC CPU 测试程序入选标准非常严格。比如对于目前

[3] lookahead：一个节点预测从当前时刻起到下一次发送消息的时间

最常用的 SPEC CPU 2000，为了避免使用磁盘，所有程序运行时对内存的需求都小于 200M，这样就使得 256M 内存配置的计算机运行这些测试集时不需要换页；另一方面，几乎所有测试集的内存需求都大于缓存容量。这样严谨的遴选条件使得 SPEC CPU 测试集有别于那些随手编写的测试集。

完整地运行 SPEC CPU 测试集需要较长的时间。为了缩短运行时间，克雷恩奥索斯基（KleinOsowski）[43]使用 simplescalar 对 SPEC CPU 运行行为做记录分析，然后使用多种方法来对不同的基准测试程序减小输入数据集，从而得到一个可以代表原 SPEC CPU 运行的新测试集 MinneSPEC，MinneSPEC 已经被 SPEC 组织吸纳。

范萨尔卡（Phansalkar）[44]等使用可编程计数器（programmable counter array， PCA）的方法对 SPEC CPU 89、92、95 和 2000 进行对比，发现不同版本间的指令数差别较大，数据的局部性也有很大程度的变化，但对于分支和指令级并行特征变化不大。

NPB

美国国家航空航天局（National Aeronautics and Space Administration，NASA）发布的 NPB（NAS Parallel Benchmarks，数值空气动力学模拟并行基准测试程序）主要用来测试并行超级计算机。它的核心程序算法来自计算流体力学，由八个测试程序组成。

NPB 主要分为 NPB1、NPB2、NPB3 三个版本，其中 NPB1 是最原始的算法，使用者可以根据机器特色对算法和编程模型重新实现。NPB2 则是由美国国家航空航天局对 NPB1 进行补充，提供了 MPI[4]版本的 NPB，使用者不需要修改或仅需要稍微修改就可以运行。NPB2 分为 2.3 和 2.4 两个版本，2.4 是在 2.3 基础上增加了 D 规模测试集。相比于 C 规模测试集，D 测试集数据集扩大了大约 16 倍，负载扩大了大约 20 倍。NPB3 是在 NPB2.3 的基础上开发的，增加了 OpenMP、HPF 和 Java 等支持。并行模式分为粗粒度和细粒度两种模式，粗粒度并行使用 MPI 进行通信，细粒度并行使用 OpenMP。

4.2 参数选择

在模拟器中，不同的参数对性能的影响是不同的。为了区分不同的参数对性能的影响，普拉克特与伯曼设计（Plackett and Burman design，PB）[45]提供了一种基于统计的方法来评价体系结构参数对性能的影响程度。它的主要流程如下：

- 列出所有关心的参数，并确定 PB 矩阵。假如一共有 X 个参数，那么 PB 矩阵由 X 行和 X-1 列组成。其中每一行代表一种配置，每一列代表每一个参数在每种配置下的取值（-1 或+1）；
- 对 PB 矩阵进行初始化，首先确定矩阵第一行，这一行是由一系列-1 和+1 组成的，由文献[61]给出。接下来的 X-2 行的每一行的确定都是由上一行右移一位而形成，最后一行全部由-1 组成；
- 为参数选择“高”和“低”值，其中“高”值对应 PB 矩阵中的+1，这个数值通常要高出该参数所取得的正常范围内的数值；而“低”值对应 PB 矩阵中的-1，这个数值通常要低于该参数所取得的正常范围内的数值；
- 在各种配置下执行模拟，并记录模拟结果，然后根据模拟结果计算影响程度。

根据这种方法，就可以将所有参数对性能的影响因子计算出来了。这种方法的优点在于客观地评价了各种参数对模拟结果产生的影响，减少了模拟过程中产生的误差，可以帮助体

[4] Message Passing Interface，一种基于消息传递的并行程序设计标准

系结构设计者更快地发现误差。而且该方法在确定参数对性能影响的同时，还能表达出一些参数之间的交互关系。

4.3 运行加速

虽然模拟器开发时采用了很多加速优化手段来提高模拟器的运行速度，但目前软件结构模拟器的运行速度通常比其模拟的硬件慢 2~4 个数量级，因此，在模拟器上运行整个基准测试程序集可能需要很长时间。为了快速得出结果，一种方案是减小输入数据集，但应用程序在不同的数据集下的行为可能相差很大，因此这种方法存在缺陷；另一种方案就是使用采样技术，采样技术的基本做法是不完全模拟整个基准测试程序，而是仅仅模拟整个基准测试程序的一部分、一个片断来代表整个基准测试程序的运行。但如何对程序代码进行采样，使得采样的代码片断可以代表整个基准测试程序就成为了采样技术的关键。目前主要常用的采样技术包括下面几种：

头部采样

选取足够大的数值 N，仅运行基准测试程序的前 N 条指令，以这 N 条指令的执行结果代表整个基准测试程序的行为[46]。由于程序开始的前 N 条指令是初始化部分，而程序在运行的不同阶段特征是不同的，所以这种方法往往不具有代表性。

随机采样

在基准测试程序的动态执行踪迹中采样若干代码片断，每个代码片断的规模为固定常数条指令，采样代码段之间的距离是随机的。然后对这些采样后代码片断进行模拟，并将其结果同整个基准测试程序的运行结果进行比较，如果采样不具有代表性，则重新采样，并增加到采样集中，重复进行验证采样是否具有代表性。这个过程一直进行，直到采样的代码片断具有代表性。对于如何验证采样的好坏，加里（Gary）[47]给出了一种方案。该方案将整个验证过程分成两步：1）快速检验（fast checks），主要决定采样集的量是否够；2）最终检验（final checks），决定采样集是否正确。其中快速检验包括检验每类指令的执行频率、每个基本块的执行频率以及缓存模拟中的缓存缺失率，最终检验则是检测采样的代码是否具有整个基准测试程序的行为特点，即是否具有代表性。

虽然随机采样的方法最终得到的采样结果可以代表整个基准测试程序，但这种方法也存在明显的缺陷，首先是确定采样集时反复采样、验证非常耗时，其次所取得的采样集不一定是最小集合。

统计采样

SMARTS[48]（Simulation via Rigorous Statistical Sampling，通过严格统计采样进行的模拟）是一种新的基于统计的采样方法，它通过对变化差异进行测量从而使得采样出的代码能够反映出程序的变化，并采用统计的方法使得采样集为可以代表整个程序的最小采样集。为了使模拟器在运行采样代码前缓存、分支预测单元等部件处于正确的状态，SMARTS 不仅仅对采样后的代码片断进行模拟，同时对采样代码片断之间的代码也进行模拟，只是对这两个阶段模拟的详细程度不同。SMARTS 的一个采样样本通常包括大量的小的采样单元，每个小的采样单元可以小到只有 1000 条指令。SMARTS 技术中，模拟过程包括功能模拟和详细模拟两个步骤。功能模拟主要为即将到来的详细模拟构建正确的体系结构状态，如缓存、变换索引缓冲（TLB）[5]等部件的内容；详细模拟阶段所有的体系结构的状态都将被模拟出

[5] translation lookaside buffer，亦有译成“旁路转换缓冲”，即存放虚拟地址到物理地址的转换结果的页表.

来。为了加速功能模拟，陈（Shelley Chen ）[32]提出了将功能模拟采用直接执行的方法。

舍伍德（Sherwood）[49]提出了基于基本块采样的方法，将程序执行过程中基本块的执行频率记录下来，然后把基本块的执行频率乘以基本块中的指令条数作为结果存储在 BBV（basic block vector）中。接着对 BBV 向量降低维度，并计算向量间的欧拉距离。将距离近的向量作为一组，再从每组中选出一个向量作为代表，就形成了整个采样集。

5 常见模拟器

5.1 单节点模拟器

5.1.1 simplescalar

simplescalar[1]是一套完整的工具集，包括编译器、汇编器、连接器和库等，由威斯康辛（Wisconsin）大学 1992 年开发，1995 年开始发布 1.0 版本。发布之后，simplescalar 迅速成为研究处理器设计的重要工具。据统计，2000 年计算机顶级会议的所有文章中有三分之一都用到了它。

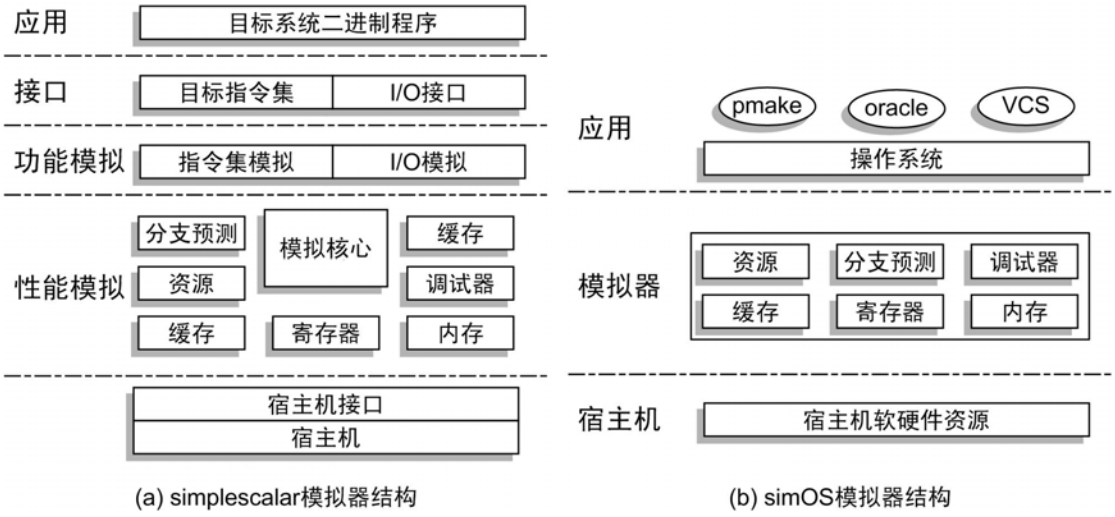

图4. 主流模拟器

simplescalar 模拟的范围非常广泛，从非流水的单处理器到具有动态调度指令和多级存储模型的多处理器系统。它支持多种指令集，主要包括 Alpha，PowerPC，X86 和 ARM，还支持多种运行模式，包括 sim-safe，sim-fast，sim-profile，sim-bpred，sim-cache，sim-fuzz 和 sim-outorder 等，其中最常用的是 sim-safe，sim-fast 和 sim-outorder。sim-safe 模拟最为简单，仅仅模拟指令集；sim-fast 则在 sim-safe 的基础上增加了指令对齐、访存保护等机制；sim-outorder 最为复杂，包括动态调度、投机执行等处理器技术，并且模拟了多级存储系统。

simplescalar 采用了功能模拟和时序模拟分离的方法，如图四(a)所示。功能模拟模块提供指令集和 I/O 模拟，性能模拟部分提供模拟器运行的性能模型。指令集模拟部分采用解释型执行的方式，将目标系统指令集解释执行。指令集解释器采用易于理解的目标定义语言，清晰地描述了指令执行过程中对寄存器或内存的修改，并使用预处理来判断依赖关系并产生 simplescalar 需要的代码。simplescalar 的 I/O 模拟模块提供了丰富的 I/O 功能，可以满足通常的系统调用需要，对于专门的体系结构，simplescalar 还提供了专门的 I/O 模拟，比如康柏（compaq）的 Ipaq 设备等。性能模拟部分的核心是一个时钟计数器，它循环接收功能模拟器部分传递来的指令，根据指令类型来计算指令的执行周期。通常对于非访存指令，执行

周期都设置为 1，而对于访存指令，则根据所模拟的内存级别做相应的计算。

simplescalar 的最近版本 MASE[24]（4.0）已经发布。相比之前的版本，MASE（4.0）在以下方面做了增强：1) 功能模拟部分维持不变，性能模拟部分不再简单地计算延迟，而是使用一个微核心来执行指令。微核心中执行的指令根据系统的动态特征进行分支预测；2）在微核心中执行的指令的提交阶段和功能模拟部分的执行结果进行验证，对于不正确的指令，则从上一次正确的地方重新执行。这样性能模拟部分就可以更好地给出系统的动态性能特征。之前的版本只有分支预测失败时才可以回滚到分支预测指令，而 MASE 则可以回滚到任意一条指令，这样可以有效支持猜测读、处理器精确中断等技术；3) 前期版本的延迟是固定的，MASE 中增加了可变延迟的支持，更好地体现了系统的动态特征，比如系统动态随机存储器（DRAM）对访存指令进行重排所带来的访存延迟影响。

simplescalar 重点对 CPU 模拟，因此是研究 CPU 微结构的理想工具。但 simplescalar 4.0 以前的版本不能启动操作系统，所有的应用需要用工具包中的工具进行编译链接才能执行。因此，simplescalar 上可以运行的应用程序受限，并且无法体现出操作系统对系统的影响。

5.1.2 SimOS

SimOS[2]是斯坦福（stanford）大学 1998 年开发的模拟器，可以模拟单 CPU 系统或对称式多处理器架构（Symmetric Multi-Processing，SMP）系统。如图四（b）所示，SimOS 模拟器整个系统，包括 CPU，存储系统、外设等。SimOS 模拟的主要结构是 SGI 公司的 MIPS 和 DEC 公司的 alpha 处理器，由于 SimOS 对系统各个部件的模拟都非常详细，所以可以启动和运行商用的操作系统。已经移植的操作系统包括 IRIX 和 digital UNIX、Linux（alpha）等。由于 SimOS 上可以运行完整的操作系统，所以在标准操作系统上可以运行的程序，如数据库和万维网服务器（web server）等复杂的应用都可以不加修改地在 SimOS 上运行。

为了达到速度和精确性的统一，SimOS 提供了三种运行模式，分别是 Embra、Mipsy 和 MXS。Embra 模式也叫做定位模式（positioning mode），它忽略等待时间，所有的 I/O 操作立即得到响应，而且采用动态二进制翻译的方法将目标指令集直接翻译成宿主机的指令集。这种模式通常用于操作系统和应用程序启动，或者其他用户不关心的执行部分，保证以最快的速度完成这一过程；Mipsy 模式也叫粗特性模式（rough characterization mode），它是在 Embra 模式基础上增加模拟模块，主要包括缓存模拟，固定延迟的外设模拟等，可以在达到功能的基础上获取系统的存储系统或 I/O 系统的粗略信息。MXS 模式也叫精确模式（accurate mode），这种模式下系统中的各个部件都模拟得非常详细，CPU 模拟模块模拟流水线、超标量、分支预测、猜测执行和寄存器重命名等处理器技术，缓存模拟可以模拟两级缓存，可以模拟出缓存的容量、替换策略、缓存行大小、组相连等参数对性能的影响。这三种模式模拟的精确性依次提高，相应的模拟速度也依次降低。SimOS 的多种运行模式在运行时可动态切换，实现在用户只关心功能的情况下快速模拟，在用户关心细节的情况下详细模拟，达到速度与详尽性的统一。

SimOS 提供了强大、高效的数据收集模块。用户利用它可以方便地定制自己需要收集的数据。SimOS 的数据采集重点解决两个方面的问题，一是低层次的数据如何映射至高层抽象结构，如何区分进程，区分内核行为和应用程序的行为等；另一个是数据处理速度问题，如果太慢将会大大影响模拟时间。SimOS 采用的方案是硬件模拟模块负责事件生成，如指令执行、内存管理单元例外、缓存不命中、设备中断等，而事件处理例程的定义和注册交由用户去做。事件处理过程采用工具命令语言（Tool Command Language， TCL）脚本编写，利用它可以访问及改变目标机器状态，可以看到应用程序的数据结构，因此作为一个桥梁，

能够使低层次的数据更加有意义。数据收集模块还提供了桶选择器，用来将事件归类，如缓存不命中是在内核空间还是在用户空间，属于哪一个函数/过程等。

SimOS 虽然模拟的平台不多，但由于 SimOS 可以不加修改地运行操作系统以及强大的数据收集分析功能，使得其在从体系结构设计和研究，到操作系统开发和性能调整，以及编译器、应用程序性能瓶颈的探究等广泛领域得到应用。但它的缺点是代码量大，难以维护。

5.1.3 SimICS

SimICS[7]最初是由瑞典计算机科学研究所（Swedish Institute of Computer Science）开发的，是一个全系统模拟器，可模拟非常多的平台，无修改运行多种操作系统；

SimICS 可以模拟多种指令集架构，包括 Alpha、x86-64、ARM、EM64T、IA-64、MIPS、PowerPC、POWER、SPARC、x86 等；可模拟众多外设，如键盘鼠标，直接内存存取，软盘、磁盘，3D 图形加速、网络以及自定义设备等；其上可运行多种操作系统，如 DOS、Windows、VxWorks、OSE、Solaris、Tru64、FreeBSD、Linux、QNX、RTEMS 等；可模拟多种系统如嵌入式系统、机顶盒、交换机、对称式多处理器架构、机群（Cluster）以及这些系统构成的复杂网络。

SimICS 采用一种面向对象的描述语言来配置目标系统，对模拟过程的控制通过命令行接口或者脚本来完成，对于多个 SimICS 模拟目标构成的网络，采用一个称为 SimICS Central 的模块来进行虚拟时间同步和消息路由。

SimICS 一个重要的特点是容易扩展，提供了功能丰富的应用程序接口和数据结构，使得用户可以方便地定义自己的设备模块，添加新的控制命令和数据收集分析例程。SimICS 在虚拟设备模拟和调试环境支持上做了相当多的工作，取得了较大的成果。它引入了设备模型语言 DML（Device Modeling Language）和相应的编译器 DMLC（Device Modeling Language Compiler），用户可以利用 DML 快速创建非标准(未来)的虚拟设备，通过 DMLC 与系统其他部分整合；另外，SimICS 实现了可逆执行及调试的功能，使得程序不用重启可以恢复到以前的执行状态，这个对调试而言很有意义。由于逐个周期地保存处理器、存储系统及外设的状态需要非常大的存储空间，同时会使得模拟时间增长以至于不可容忍，所以 SimICS 采用检查点机制，在回滚粒度、存储及时间消耗上作了一个折衷。如果用户需要程序逆向执行，则回滚至目标之前最近的一个检查点，再执行至目标指令，现在技术条件下程序运行速度之快可以完全掩盖这种机制带来的延迟。

SimICS 现在已经商业化，成为 Virtutech 公司的产品。由于 SimICS 支持的指令集架构、外设非常丰富，可以启动多种操作系统，并且提供标准接口，使得加入新的设备非常容易，所以得到研究和开发人员的广泛使用，尤其在嵌入式系统的硬件设计和软件开发调试方面。

5.1.4 Mambo

Mambo[5]是 IBM 开发的一个全系统模拟器，主要用在操作系统开发、系统自举、应用程序性能特征检测、性能调优、能耗、硬件设计方案预研等。

Mambo 主要对 IBM PowerPC 系列处理器进行功能模拟和时序模拟。为了支持应用程序的开发，Mambo 模拟器中还包含了对各种外设的模拟。为了达到灵活性，Mambo 在模块化和可配置化方面做了很多工作。为了提高速度又不失精度，它提供了多种时钟模型，从简单的每条指令一个时钟周期，访存操作立即完成，到详细的每个设备都做时钟级模拟。为了既达到时钟级模拟的效果，又不牺牲速度，Mambo 中使用了一种独特的 cycle approximate（时

钟周期近似）执行模式。在这种模式下，根据用户事先统计得到的缓存不命中率等来根据概率计算访存延迟，而不是去实际查找模拟的缓存来决定本次访存的时间。

除了支持系统级模拟，Mambo 还提供了用户级模拟，在 Mambo 中称之为 standalone 模式。这种模式把操作系统功能集成在模拟器中，使得模拟器上可以直接运行用户程序。对于数据采集，Mambo 中使用一种称之为 emitter（发射器）的方式，即由 Mambo 产生数据到内存区，这段内存区是一段循环缓冲区，再由其他辅助程序从缓冲区中读出数据进行分析。这种数据采集方法避免了对模拟器的污染。

作为一个商用模拟器，Mambo 在产品开发和系统性能分析方面发挥了重要作用，例如：

- 支持开发 K42 操作系统；
- 使得 BlueGene/L[6]在硬件可以使用后，整个系统在一周内运行起来；
- 发现 PowerPC CPU 的一个控制寄存器存在竞争条件，使设计人员得以及时对该控制寄存器的几个控制位的语义做了修改。

近来计算机体系结构方面能耗逐渐受到重视，因此 Mambo 近期发展目标是加强对能耗模拟，主要是验证一些改善能耗的想法，比如对于访存密集型应用降低 CPU 主频来减少能耗等；另一方面，PowerPC 处理器向众核方向发展，模拟负载也越来越重，因此 Mambo 本身也在做并行化方面的工作。

5.1.5 RSIM

指令级并行处理器可以隐藏部分的访存延时，因此访存阻塞带来的系统性能瓶颈可能会发生改变。为了对这种情况进行研究，如果使用顺序发射、读阻塞（read block），没有投机执行的系统模拟器无法很好地体现指令级并行的特征。为了模拟出指令级并行系统特征，经常使用方法是 simpleN，也就是将处理器的 CPU 频率乘以 N，将内存访问时间乘以 N，N 的范围为 1 到处理器的发射宽度。但 simpleN 的方法存在两个缺点：1) 由于 N 值是随应用和系统变化的，因此没有一个很好的方法可以决定 N 的大小；2)无法模拟指令级并行系统中同一个处理器同时多个读写请求的情况。编译器可以对 read miss clustering（读缺失聚合）优化，从而使访问同一个动态随机存储器缓冲区的请求一起发出，这样可以提高内存系统的效率，但如果采用 sinleN 的方法就无法模拟出这种影响。因此，莱斯（Rice ）大学 1997 年发布了模拟指令级并行多处理器系统 RSIM[15]。

RSIM 模拟的目标系统是一个 CC-NUMA[7]系统，支持 MESI[8]或 MSI[9]内存一致性模型，主要模拟的部件包括、处理器、内存单元、缓存、目录和互联网络。RSIM 基于 MIPS R10000 体系结构，内存和网络系统使用 RPPT（Rice parallel program testbed，莱斯大学并行计算程序试验床），指令集是升阳（Sun Microsystems ）的 SPARC V9。RSIM 可以模拟出多发射、动态指令调度和非阻塞内存读写等系统特征，并且还模拟了寄存器重命名、静态和动态分支预测、SMT 等技术。缓存模拟部分模拟了多端口、直写（write through）或回写（write back）策略的缓存。为了加速运行，RSIM 引入了 SimOS 的 Embra 模式中的二进制翻译，在 RSIM 中称为 rabbit（脱兔）模式，使 RSIM 可以快速运行到感兴趣的地方。

RSIM 的统计信息十分详细，其中对指令执行的统计包括总的执行周期和每条指令的执行周期等，并且它进一步将执行周期分为处理器忙、等待算术逻辑单元（Arithmetic Logic

[6] IBM 开发的全球运行速度最高的计算机系统
[7] cache-coherent non uniform memory access，高速缓冲存储器一致性非均匀存储器访问
[8] Modified, Exclusive, Shared, Invalid，高速缓存修改、排除、共享、废弃一致性协议
[9] Modified, Shared, and Invalid

Unit， ALU)、等待浮点运算单元（Float Point Unit, FPU)、读数据、写数据、异常、分支、同步等。对缓存统计可以分为冷启动、容量或冲突等造成的失效。

simpleN 方法和 RSIM 的模拟结果表明，simpleN 的方法可能会把 CPU 的执行时间估计得过短，因为每个周期都会有多个指令同时完成；同时对访存的时间又会估计得过长，因为它忽略了指令级并行系统可以对访存延迟做一定程度的隐藏。尤其是对于编译器对 read miss clustering 优化之后的评估，使用简单的 simpleN 方法可能得出的结论是没有性能提升，而 RSIM 模拟结果表明这种技术可以带来很大的性能提升，而后者在实际系统中得到了验证。

5.1.6 M5

M5 是美国密西根大学（University of Michigan）开发的一套模拟器系统，2003 年 10 月发布了最初的 1.0 beta 1 版本，2007 年 11 月发布了 M5 2.0 beta 4。M5 主要目的是提供一个开源的，多机和多系统的模拟器，侧重体现计算机系统在多机和网络方面的功能和性能特点，因此 M5 在各级总线，I/O 模块和网络方面的模拟比较详细。

早期 M5 的开发，在 CPU 模拟方面参考了 SimpleScalar 模拟器，在针对 Alpha 指令集的全系统模拟方面参考了 SimOS/Alpha 模拟器，而在后来则完全独立于其他模拟器，具有自身的特点。M5 使用 C++和 Python 两种语言，C++用于开发底层的功能单元，Python 用于开发上层的模拟器系统。由于开发语言是面向对象的，所以整个系统的开发有利于应用软件工程的方法。

M5 有三个可以相互替换的 CPU 模拟单元，一个是 SimpleCPU，一个是 O3CPU，还有一个是踪迹驱动的模块，用于进行随机的内存系统测试。其中 SimpleCPU 和 O3CPU 都是执行驱动的模块，SimpleCPU 是顺序执行，无流水线的功能级 CPU 模拟器，可以被配置成单周期执行一条或多条指令，而 O3CPU 是乱序执行、超标量、流水线和并发多线程的时钟级 CPU 模拟器，该模块还包括详细的分支预测单元、指令队列、加载/存储（Load/Store）队列和功能单元的模拟。M5 的内存系统包括两类主要的模块：设备和互连。设备是指缓存，内存和 I/O 设备的模拟，互连是指各级总线和网络的模拟，其中各级总线包括前端总线、系统总线和 PCI 总线。缓存的模拟支持可配置的参数有大小、延迟、相联、替换策略和一致性协议等。总线的模拟支持分离的事务模型，可配置的参数有延迟和带宽。对于 I/O 设备的模拟，M5 选择了一款国家半导体公司的 DP83820 型网卡进行详细的模拟。除此之外，M5 还模拟了串行端口和磁盘控制性、中断控制性等设备。在网络模拟方面，M5 拥有一个无损耗、双通道、可配置带宽和延迟的以太网连接单元，用于连接多个机器。

M5 可在 x86、SPARC、Alpha、PowerPC 等处理器和 Linux、MacOS X、Solaris、OpenBSD、Cygwin 等操作系统运行。M5 模拟的指令集有 Alpha、SPARC 和 MIPS，还在扩展到 PowerPC 和 ARM 指令集。M5 可运行在两种模式下，一种是系统调用仿真（System Emulate），直接运行二进制应用程序；另一种是全系统模拟（Full System），能够启动操作系统。目前 M5 在模拟 Alpha 指令集的情况下，可以启动无修改的 Linux 2.4/2.6，FreeBSD， L4Ka::Pistachio，甚至包括惠普/康柏的 Tru64 5.1 版。而在模拟 SPARC 指令集的情况下，可以启动 Solaris。

近两年随着多核时代的来临，由于 M5 模拟器在多机和网络模拟方面的优势，使得其日益受到学术界的重视，在国际顶级期刊上有多篇论文都使用到了 M5 模拟器来论证它们的实验结果。

5.2 多节点模拟器

5.2.1 BGLsim

BGLsim[27]是 IBM 开发的 BlueGene/L 模拟器，运行在真实的 Linux 机群系统上，其运行平台是 X86/Linux，多节点通信使用 MPI 实现，因此 BGLsim 是一个 MPI 程序。整个模拟系统是一个栈结构，如图五 a)所示。BGLsim 模拟了大规模机群系统 BlueGene/L 的所有硬件，在这层虚拟的硬件之上运行的是 BlueGene/L Linux 系统，其上提供了移植的 BlueGene/L MPI 库。应用程序使用这个库来进行一些操作，应用程序是运行在这个虚拟机群系统上的一个 MPI 程序。

与单节点模拟相比，多节点模拟需要增加对节点之间的通信和互联网络的模拟，在多节点模拟的时候主要有三类进程：bglsim、ethgw、idochip。

- bglsim：单节点模拟器，要模拟多少个节点的机群系统便有多少个这类进程，bglsim 通过修改 Mambo 模拟器实现，其中增加了花托形（torus）和树形（tree）网络的模拟；
- ethgw：提供模拟器环境中虚拟以太网和外部真实以太网之间的互联服务，使得外部网络服务对模拟器节点真实可见，这样模拟器就可以使用外部提供的服务，如 NFS[10]等；
- idochip：通过 JTAG[11]接口，提供节点和外部控制系统之间的交互。它以 IP[12]封包的形式传送控制系统发出的底层控制命令，实现对节点状态的监测和控制等功能。

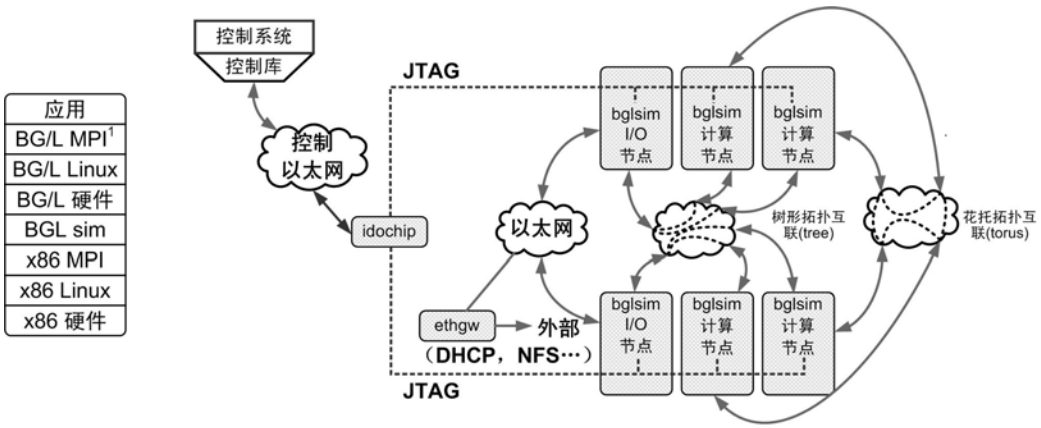

图5. BGLsim 结构

除了上述的部件模拟，BGLsim 使用一个叫做 CommFabric 的共享库提供的抽象应用程序接口来实现进程间通信服务。CommFabric 使用 MPI 作为它的传输层，它需要把不同的网络地址（MAC[13]地址，IP 地址，Torus 网络位置等）映射到相应进程的 MPI_Rank 上。BGLsim 使用了不同的线程来实现通信和模拟，在消息传递的过程中，使用自适应路由算法的真实路由操作，数据包到达每个节点的路由方式由节点中的路由表决定。

5.2.2 MPI-SIM

MPI-SIM[50]是美国加州大学洛杉矶分校（UCLA，the University of California, Los Angeles）开发的一个 MPI 库，主要用来测试、调试、预测并行程序在各种体系结构下的表现，它可

10 Network File System，网络文件系统
11 是一种国际标准测试协，由 Joint Test Action Group——联合测试行动小组开发。
12 Internet Protocol，互联网协议
13 Media Access Control，媒体接入控制

以根据目标系统的参数特征给出不同的模拟结果，主要的参数包括处理器个数和通信延迟。

MPI-SIM 的实现主要包括两个方面：1) 对现有的 MPI 库进行修改，在 MPI 库中实现同步和统计功能，然后对用户程序进行更改，使得对标准 MPI 的调用都转向了 MPI-SIM 调用；2) 将原来程序由串行变为并行，主要是对原程序的全局变量和静态变量进行局部化，具体就是将原来的全局变量变为变量数组，每个处理器访问的时候把自己的标识符（ID）作为参数传进去，使得每个处理器上的线程访问全局变量时相互之间不受影响。

MPI-SIM 是一个并行模拟器，难点在于多进程之间的同步。MPI-SIM 中提出了一种新的保守同步算法，这个算法将现存的 NULL[14]消息机制、条件事件协议组合在一起并使用了一些优化手段减少了同步的开销和频率，这些优化被集成到针对 MPI 程序的模拟库中。MPI-SIM 中将多个进程标记为逻辑进程（logical process, LP），每个逻辑进程结构对应目标程序的进程，其中包含有消息队列、模拟时钟以及等待处理的操作序列。

MPI-SIM 将应用程序要发送的消息放到目标进程对应的逻辑进程的消息队列上，消息的发送时间就是与发送进程对应的逻辑进程的模拟时钟时间，接收的时间就是发送时间加上一个预测发送延迟时间。为了等待一个接收动作完成，逻辑进程将会阻塞，直到有匹配的消息到达。接收进程则按逻辑进程模拟时间的先后顺序而不是按照实际的到达时间从消息队列中提取消息。当一个匹配消息到达时，逻辑进程更新当前时间并返回给发送方一个确认消息，然后开始唤醒执行目标程序。

在 IBM SP2 平台上运行 NPB 测试程序的结果表明 MPI-SIM 可以很好地预测目标系统的性能，并具有很好的加速比。但 MPI-SIM 的主要缺点是需要应用程序源码，并且只能模拟目标指令集架构和宿主机相同的系统。

5.2.3 BigSim

BigSim[51]是伊利诺州大学香槟分校（UIUC, University of Illinois at Urbana-Champaign）开发的并行模拟器，能够通过执行真实应用来预测像 BlueGene/L 这样拥有超大规模并行处理器的高性能计算机的性能。

BigSim 基于 CHARM++并行编程系统，定义了一套底层应用程序接口，并用这些应用程序接口来实现上层的 MPI 接口，将应用程序与模拟器库一同编译为可执行文件。BigSim 基本的应用程序接口包括五个函数：BgNodeInit、registerHandler、addMessage、sendPacket、Utility function。通过这些函数实现对 MPI、CHARM++等编程模型的支持。BigSim 的执行环境也是 CHARM++，所有的应用程序接口都将跳转至模拟器执行，而其他代码则直接在宿主机上运行。

BigSim 对 CHARM++环境依赖过于紧密限制了其广泛使用。

5.3 并行模拟框架

5.3.1 DaSSF

DaSSF[52]是在美国能源部拉斯洛莫斯（Los Alamos）实验室的 SSF（Scalable Simulation Framework，可裁减模拟框架）[53]基础之上建立的模拟框架，用来模拟目前尚没有的更大规

[14] PDES 机制中，为了推进时钟而向对方发送的仅包含自己时钟的消息

模的并行计算机，以及在被模拟的机器上进行 ASCI[15]应用的性能评估。SSF[53]是一个全球网络模拟器，而 DaSSF 是达特茅斯学院(Dartmouth college)在 SSF 基础上开发的，作了不少扩展，比如增强了模拟器本身可扩展性，增加了数据收集功能等。

DaSSF 应用程序接口提供了五个基本类：SSF_Entity、SSF_Process、SSF_OutChannel、SSF_InChannel 和 SSF_Event。SSF_Entity 之间由*Channel 相连，SSF_Event 在*Channel 上传输，SSF_Process 定义了 SSF_Entity 的行为。被模拟系统的配置文件采用 DML 语言（Domain Modeling Language），它是一种适合于网络模拟的描述语言。通过继承前述这几个基本类，辅以 DML，理论上可以模拟任何拓朴，从而拓展了并行机处理器连接网络的拓扑结构，也就扩展了网络，因此基于 DaSSF 进行扩展来模拟并行计算机系统，是比较适合的。

拉斯洛莫斯实验室所建立的并行框架中基本的处理实体包括对称式多处理器架构节点、网卡和交换机。路由算法和电路交换算法都独立出来，没有跟网卡和交换机绑定在一起，以便于在运行时灵活地选择不同的算法。框架本身提供了与对称式多处理器架构节点模拟器交互的接口，采用 PEDS 保守同步机制，因此不存在前述乐观同步带来的内存资源消耗的严重问题。

采用这个框架模拟了较多类型的系统，包括总线连接的或者网卡对等连接的简单系统；一个或者两个交换机连接的八节点系统；大规模的系统，64, 128, …, 一直到 4096 个节点的 ASCI Q machine。其上运行的工作负载不是源于真实的应用，而是从处理器上随机发出的消息。

该模拟框架的主要特点是基于组件的结构，简单的设计，从而使得各个组件可以很容易替换，因此可以按照保真度的需求来合理地选择组件；DML 语言的灵活性和良好的网络模拟基础使得它非常适合于不同拓朴结构的模拟。其缺点是没有对处理器进行较详细地模拟，更没有模拟真实的应用，但是添加这两项功能已经被列入计划。

5.3.2 USSF

辛辛那提大学的 USSF[54]（Ultra-large Scale Simulation Framework）技术发展重点放在宿主机资源特别是内存的合理和充分利用上，因为 USSF 的一个目标是要在普通廉价的平台，如一般规模的机群下完成大规模目标系统的模拟，而这一类平台资源有限，且该框架下的并行离散事件仿真同步内核采用的是乐观同步机制，需要保存大量的状态和事件数据结构，随着规模增大，内存动态需求会迅速增大。

USSF 也是模块化的，而且框架模块与同步内核进行了较完整的分离，使得它提供的两个可同步内核可以无缝地替换。为了降低内存的需求，USSF 在框架的实现上做了很多工作，其核心思想是，改变同步核心的数据结构会较大地影响性能，所以主要应该采用一些技术来减小静态内存需求(可执行代码，描述众多逻辑处理单元的变量等等)，控制动态内存需求（事件和状态的保存）。

为了减小静态内存，在代码生成之前要对目标系统的配置文件和并行框架中描述目标系统逻辑处理单元的数据结构进行分析，找出其中大量重复的描述进行共享，实际上大部分逻辑处理单元只有少量数据和状态不同，将不同部分从结构中分离出来可以在一定程度上减小代码量；另外，这些数据和状态再进一步分离，将有利于其后的动态内存的消耗控制。控制

[15] ASCI: Accelerated Strategic Computing Initiative，现在叫做 Advanced Simulation and Computing Program(ASCP 或者 ASC)，由美国政府倡导。这个项目下的不少超级计算机已被列入 TOP500，而且比较靠前。其中包括 ASCI Q, ASCI White 等。其主要使命是“创造和使用尖端的方针和计算机建模能力”。

动态内存需求的一个主要方式是将状态和事件信息缓存至文件，必要时从文件中调入，而前述的状态和数据分离设计有利于实现信息选择性地调入调出。

USSF 的实验结果表明它更适合于模拟大规模的系统，而且对于那种基本逻辑单元绝大部分特性相同的系统，能表现出更好的性能，比如网络模拟和 VLSI 设计。

5.3.3 HLA

HLA[35]（High Level Architecture，高层体系结构）由美国国防部 1995 年发起，目的是建立一个开放的、通用的建模和仿真框架，提高仿真模块的可重用性，并提供模块间的互操作功能，主要用在军用仿真系统中。HLA 在 2000 年成为 IEEE1516 标准。

HLA 的主要理念是通过 RTI（Run Time Infrastructure，运行时基础设施）提供独立于模拟模块的服务。这样模拟模块和底层服务模块可以独立开发、优化，并提高了各个层次的可重用性。HLA 由规则、接口规范和对象模板三部分组成，对外以应用程序接口的方式提供六种服务：联邦管理(Federation Management)、声明管理（Declaration Management）、对象管理（Object Management）、所有权管理（OwnerShip Management）、时间管理（Time Management）和数据分发管理（Data Distribution Management）。由于同步问题一直是并行模拟的关键问题，所以这六种服务中的时间管理直接决定了 RTI 的性能，因此也成为 HLA 研究的重点。HLA 中的时间管理也采用了并行离散事件仿真同步机制。

5.4 局部模拟器

5.4.1 DiskSim

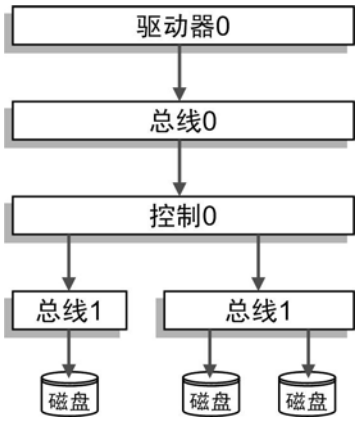

图6. simOS 模拟器结构

DiskSim[12, 55]是卡耐基梅隆大学开发的磁盘模拟器，主要目的是对已有的磁盘性能进行评价，对设计的新的磁盘的模型进行预评价。它是一个高效可以灵活配置的磁盘模拟器，既可以作为一个单独的磁盘模拟器使用，也可以被集成到更高一级的系统模拟器中成为磁盘存储子系统。如图六所示, 它把存储子系统各部分的功能用不同的模块实现。主要的四种模块是：盘组（disk）、控制器（controller）、总线（bus）、驱动程序（driver）。

5.4.2 网络模拟器 ns

ns[13]（network simulator）是美国国防部 1995 年支持开发的网络模拟器，它的前身是 REAL[56]网络模拟器。ns 模拟器在 2002 年推出了第二版，也就目前常见的 ns2。ns2 可以用来研究有线或无线网络。对于有线网络，ns2 可以支持多种路由协议，模拟 TCP/UDP 传输协议，负载流支持 web、 ftp、 telnet、 cbr、 stochastic 等应用，并且支持 drop-tail、 RED、 FQ、 SFQ、 DRR 等排队模型，还提供 IntServ 和 Diffserv 服务质量保证。对于无线网络，ns2 可以提供自组织（ad hoc）路由和移动 IP 管理模拟，并可以进行 MAC 层模拟。

ns2 使用 C++和 OTcl 来开发。C++主要用来处理数据，比如数据包的处理等；OTcl 主要用来对模拟器控制，包括目标系统配置、调用 C++编写的基本模块等，这种数据和控制分离实现的方法既利用了 OTcl 语言的灵活方便特点，又利用了 C++语言的运行高效的特点。ns2

模拟的核心是一个离散时间模拟器，系统维护一个虚拟时钟，所有的模拟都由离散事件来推动。

ns2 除了可运行的模拟器程序外，还提供了预处理器，过程显示工具等，使研究者可以方便地配置系统，并以直观的形式观看模拟结果。

5.4.3 网卡模拟器

目前的互联网需要为不同的应用程序提供不同层次服务。虽然网卡作为网络通信环节中的第一跳，但是它自身通常不提供服务质量（QoS）保障。通常来说一个网站的第二层次（网卡层次之上的一层）不会对性能做特殊考虑。这样一来，大部分的网卡配置很可能会不时地出现过载。网卡模拟器的重点就是研究这种过载以及通信能力的下降对应用程序性能造成的影响。但这种模拟器面临一些困难：受复杂的计算和数据包结构影响，过去那种对CSMA/CD[16]做细致模拟的方式不仅运行慢而且非常繁杂，当网络阻塞或过载的时候这个问题体现得更加明显。快速以太网模拟[57]则绕开了这个问题，同时提供了精确的模拟结构。快速以太网模拟分以下四步完成：

- 设计并验证一个细致的 CSMA/CD 模拟器：这一步可以使用 REAL 网络模拟器实现。使用这个模拟器可以重现各种负载以及工作集配置，并可以用来验证以后开发出的快速模拟器的结果；
- 使用上一步的模拟器收集各种配置下的实验性能指标，并将这些信息简化到一个精简的模型中。这个精简模型应当能够反映出链路吞吐率、包延迟同负载、包大小、缓冲区大小之间的相互关系；
- 开发快速以太网模拟器：主要是利用上一步的简化模型来预测性能；
- 验证这个快速以太网模拟器的结果：就是对细致的模拟器与快速模拟器的运行结果进行比较。

5.4.4 能耗模拟器 Wattch

现代处理器中由于能耗高、温度高而引发的问题越来越突出。因此，不仅仅电路工程师、芯片设计工程师，甚至编译器开发人员也需要认真考虑如何在性能和功耗之间加以权衡。目前大部分能耗分析工具都可以通过计算的方式评估某种设计的能耗，但是这类工具只能在布局布线之后完成，所以有必要开发一套能够从体系结构的层次上分析系统能耗的工具。可是实现这样一个系统非常困难，需要在底层细节、准确性上同上层应用的模拟速度和可移植性之间做折中。Wattch[14]则从体系结构层次上提供一个参数化的模拟器框架并能够准确地量化能耗的优化程度。

Wattch 是 Brooks 在 1998 年基于 simplescalar 开发的一个功耗模拟器，它使用一些在超标量计算机中比较通用的结构模块。通过为这些模块建立参数化的能耗模型来评估整个系统的能耗。这些结构模块可以分为四类：

- 数组单元：缓存、寄存器文件、分支预测部件、指令窗口等等。这类单元的主要参数包括有行列数和读写端口数，因为这两个参数直接影响到解码单元、wordline、bitline 的大小和数量；
- 内容可寻址存储器（Content Addressable Memory，CAM）：变换索引缓冲、加载/存储顺序检测单元等。这类电路的参数与数组单元的类似，只是使用 tagline 和 matchline

[16] Carrier Sense Multiple Access With Collision Detection，载波监听多路访问/冲突检测，是半双工的以太网的工作方式。

代替了 wordline 和 bitline；

- 组合逻辑单元：功能单元、相关性检测逻辑、结果总线等。这类单元的情况比较多，可以根据具体的逻辑结构建立相应的能耗模型；
- 时钟单元：时钟缓冲、时钟线路等。这类单元是高性能处理器上最重要的能耗单元，需要非常细致的思考。

Wattch 的一个很大优点是为 SimpleScalar 模拟器提供了调用接口，这样就能够方便地与 SimpleScalar 结合起来观察各种方案的能耗指数，包括每个 CPU 周期的能耗、程序执行的指令数、全系统的能耗、能耗延迟等。

使用体系结构层次的能耗模拟器还有诸多好处：在考虑能耗的时候人们可以从新的视角重新审视那些过去的技术；编译器和操作系统可以考虑采用各种技术来优化系统的能耗；研究底层系统的能耗解决方案等。

6 模拟器近期发展

随着硬件系统的快速发展，模拟器的模拟负载越来越重。为了解决模拟器的速度问题，人们近期从两个方面做了大量工作。一个是模拟器的并行化，以充分利用现有的并行系统资源；另一个是现场可编程门阵列加速或全系统现场可编程门阵列模拟。

6.1 模拟器并行化

常用的将现有的串行程序并行化的方法主要包括两个部分：假设并行的线程或进程个数为 N，第一部分是将串行程序中的共享变量或静态变量改为 N 维数组；第二部分是函数访问接口也增加一个索引（index）参数，从而使得每个并行线程或进程只访问属于自己的那一部分共享变量。潘瑞（Penry）[58]提出了一种自动将串行程序改为并行的方法，主要包括三部分工作：1）使用 POSIX 线程来让每个线程执行一份应用；2）对于线程局部变量进行线程局部化(TLS：thread local storage)；3）实现时钟级同步，但同步仅发生在访问共享资源的时候。使用这些技术，潘瑞成功地将片上多处理器（CMP，on-chip multiprocessor）模拟器图兰多（Turandot）变成了并行图兰多片上多处理器（Parallel Turandot CMP，PTCMP）模拟器，将 simplescalar 变成了并行。

马修（Matthew）[59]在 2002 年以 SimpleScalar 为基础，尝试使用 MPI 通信的方式实现一个分布式的 CMP 模拟器 SimpleCMP，设计中考虑的主要问题有：

- 由于二级缓存在 CMP 中用于多核间互联，而一级缓存提供 95%的访存，因此可以考虑使用传统的时钟驱动的技术来模拟 CPU 和一级缓存，使用事件驱动的方式来模拟二级缓存；
- 对于处理器和二级缓存的模拟可以分别采用不同的线程来进行；
- 对于二级缓存的模拟，集中模拟和分布式模拟各有自己的问题。如果在集中模拟时，二级缓存采用单独的线程，则当系统规模较小时加速比不会很高。如果系统规模很大，那么二级缓存又会成为模拟的瓶颈。如果采用分布式模拟系统同步开销会比较大，并且可能导致高的二级缓存竞争率；
- 阻塞（blocking）或非阻塞（non blocking）缓存。如果在取指的时候就对缓存做阻塞操作，那么在整个事务处理过程中这个线程就只能等待。如果采用非阻塞的缓存操作，那么在整个事务执行过程中就可以进行其它处理器的模拟。

SimpleCMP 解决了上诉问题后在九个双核的机器上达到了 16 倍的加速比。

为了加速对称式多处理器架构系统上应用特征同构的应用的模拟，卡纳乌加（Kanaujia）[60]提出了 FastMP 的方法，使用一台计算机做详细地模拟，并产生踪迹，这些踪迹也可以代表多节点情况下其他节点的执行。然后将这些踪迹分成不同的片段，其他节点分别执行不同的片段来达到并行执行的目的。

6.2 FAGA 加速

帕特森（Patterson）[28]主持开发了 RAMP（Research Accelerator for Multiple Processors）项目，RAMP 系统全部使用现场可编程门阵列进行全系统模拟。由于速度原因，现有的软件模拟器对 16 核以上系统无法模拟，而 RAMP 目的就在于研究 16 个核心以上的多处理器系统。RAMP 的开发由多个大学和单位完成，是一个综合的软硬件平台，主要用于在多核环境下进行体系结构、编译器、操作系统和应用程序方面的研究。

邱（Chiou）[30]首次设计并实现了一个软硬件结合的模拟器 FAST。FAST 模拟器将功能模拟和时序模拟分离开来，对于功能模拟部分使用软件模拟器 QEMU[8]来实现，而对于时序模拟部分使用现场可编程门阵列来实现。软件模拟器只提供指令集架构，确保程序可以正确运行。现场可编程门阵列模拟器模拟微结构，模拟分支预测、缓存状态等。功能级模拟器和时序模拟器之间通过一个缓冲区来交互。功能级模拟器产生执行的踪迹信息，时序级模拟器从缓冲区中读出踪迹信息进行微结构模拟。由于功能模拟器的运行结果总是正确的，而时序模拟部分则会产生出误预测的指令，因此需要及时地同步，并由时序级模拟器去指导功能级模拟器执行那些被误预测的指令。通过这种方法，可以实现一个运行速度快、模拟精度高的模拟器。

7 总结

人们常用建立模型或开发软硬模拟器对未来的机器性能进行评估、对各种设计方案做比较。由于模拟器对目标系统描述更准确，并且执行驱动的模拟器还可以在硬件开发的同时支持系统软件的开发，可以获取比硬件运行更为丰富的踪迹，因此模拟器，尤其是执行驱动的模拟器得到广泛的应用。

为了使模拟器更准确、更快速地模拟目标系统，并且易于更改配置以模拟新的目标机结构，人们在模拟器开发过程中对于如何提高模拟器精度、灵活性及运行速度都采取了很多方法。虽然可以采用很多的验证技术来确定模拟器精度，但模拟结果误差是无法避免的，因此更有意义地是模拟的相对结果而不是精确值。为了提高模拟器灵活性，一般采用模块化的设计方法，每个目标部件都有模拟精度和速度不同的模块，这样模拟器使用者就可以根据对速度、精度的需求来灵活地定制模拟器的不同模块。为了方便使用者配置，模拟器都采用了简单、直观、灵活的配置语言。制约模拟器使用的关键因素是模拟器的速度，因此设计者对于模拟器如何加速提出了很多种方法，主要包括单处理器模拟加速和并行加速两个方面。单处理加速一般采用穿线码、二进制翻译、直接执行等技术，并行加速常见方法是开发一个基于并行离散事件仿真同步机制的并行框架，该框架提供并行模块间的同步和通信等功能，并以应用程序接口的形式供模块调用。

尽管在提高模拟器速度方面已经做了大量工作，但有时仍不能满足使用者的需要。因此人们使用很多方法来加速模拟器的执行过程，常见的方法有减小输入数据集、对目标程序采样执行、快速推进和检查点重启动等。

参考文献

[1] Todd Austin, E.L., Dan Ernst. *SimpleScalar: An Infrastructure for Computer System Modeling.*

Computer IEEE, 2002. vol. 35, no. 2: p. 59-67

[2] Stephen, A.H. *Using Complete Machine Simulation to Understand Computer System Behavior*. 1998, Stanford University

[3] Binkert, N.L., E.G. Hallnor et al. *Network-Oriented Full-System Simulation using M5.* the Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW), 2003

[4] Vachharajani, M., et al..*Microarchitectural Exploration with Liberty.* Proceedings of the 35th International Symposium on Microarchitecture, 2002

[5] Bohrer, P., et al.. *Mambo: A full system simulator for the PowerPC architecture*. 2004

[6] http://developer.amd.com/simnow.jsp

[7] Magnusson, P.S.C., M.; Eskilson, J.; Forsgren, D. *Simics: A full system simulation platform.* Computer IEEE, 2002. vol.35, no.2: p. 50-58

[8] Bellard, F. *QEMU, a fast and portable dynamic translator.* USENIX Association, 2005: p. 41--41

[9] 沈林峰, 等. *兼容 Linux 应用环境的多粒度全系统模拟平台-SandUPSim.* 计算机工程与应用, 2005. 22

[10] 高翔, 等. *基于龙芯 CPU 的多核全系统模拟器 SimOS-Goodson.* 软件学报, 2007. 4

[11] Florian, S., S. Jens, R. Alexander. *A Cache Simulator for Shared Memory Systems*, in *Proceedings of the International Conference on Computational Science-Part II*. 2001, Springer-Verlag

[12] http://www.pdl.cmu.edu/DiskSim/

[13] http://www.isi.edu/nsnam/ns/

[14] Brooks, D., V. Tiwari, M. Martonosi, *Wattch: a framework for architectural-level power analysis and optimizations.* ISCA, 2000: p. 83-94

[15] Christopher J. Hughes, V.S.P. Parthasarathy Ranganathan, Sarita V. Adve, *RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors.* Computer IEEE, 2002. vol. 35, no. 2: p. 40-49

[16] Emer, J., et al. *Asim: a performance model framework.* Computer, 2002. 35(2): p. 68-76

[17] Sharma, A., et al. *Augmint: a multiprocessor simulation environment for intel x86 architectures.* Technical report, University of Illinois at Urbana-Champaign, 1996

[18] Renau, J., et al. *SESC: cycle accurate architectural simulator.* http://sesc.sourceforge.net, 2005

[19] Bedichek, R.C. *Some Efficient Architecture Simulation Techniques.* Proceedings of the USENIX Winter 1990 Technical Conference, 1990

[20] Bell, J.R.*threaded code.* Commun. ACM, 1973. 16: p. 370-372

[21] Bedichek, R.C. *Talisman: Fast and Accurate Multicomputer Simulation.* Measurement and Modeling of Computer Systems, 1995: p. 14-24"

[22] http://www.vmware.com/

[23] Carl, J.M., D.H. Mark, A.W. David, *Full-system timing-first simulation*, in *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. 2002, ACM Press: Marina Del Rey, California

[24] Larson,E.C.,S.Austin,T..U.o.Michigan, *MASE: a novel infrastructure for detailed microarchitectural modeling.* Performance Analysis of Systems and Software, 2001. ISPASS. 2001 IEEE International Symposium on, 2001

[25] Perumalla, K.S.,*Parallel and distributed simulation: traditional techniques and recent advances.* Winter Simulation Conference 2006, 2006: p. 84--95

[26] Fujimoto, R.M.,*Parallel discrete event simulation.* Commun. ACM, 1990. **33**: p. 30--53

[27] Ceze, L., et al.*Full Circle: Simulating Linux Clusters on Linux Clusters.* Proceedings of the Fourth LCI International Conference on Linux Clusters:The HPC Revolution 2003, 2003

[28] Patterson, D.A., *RAMP: research accelerator for multiple processors - a community vision for a shared experimental parallel HW/SW platform.* Performance Analysis of Systems and Software, 2006 IEEE International Symposium on, 2006: p. 1-

[29] Moo-Kyoung, C., S. Heejun, K. Chong-Min. *Performance Improvement of Multiprocessor Simulation by Optimizing Synchronization a Communication*, in *Proceedings of the 16th IEEE*

International Workshop on Rapid System Prototyping (RSP'05) - Volume 00. 2005, IEEE Computer Society

[30] Chiou, D., et al. *FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators.* MACRO, 2007

[31] Barr, K.C., et al. *Accelerating Multiprocessor Simulation with a Memory Timestamp Record.* Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on, 2005: p. 66-77

[32] Chen, S. *Direct SMARTS: Accelerating microarchitectural simulation through direct execution.* 2004

[33] Milo, M.K.M., et al., *Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset.* SIGARCH Comput. Archit. News, 2005. **33**(4): p. 92-99

[34] Wilsey, D.R.a.P. *An ultra-large scale simulation framework.* Journal of Parallel and Distributed Computing 2002

[35] Defense, U.S.D.o., *High Level Architecture Interface Specification Version 1.3 2.* 1998

[36] Joshua J.Yi,D.J. Lilja. *Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations.* IEEE Transactions on Computers 2006. vol. 55, no. 3: p. 268-280

[37] Lipasti, H.C.a.K.L.a.B.S.a.M. *Precise and Accurate Processor Simulation.* In Proceedings of the Fifth Workshop on Computer, 2002: p. pages 13--22

[38] Reinhardt, S.K., et al. *The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers.* Measurement and Modeling of Computer Systems, 1993: p. 48-60

[39] Mukherjee, S. *Wisconsin Wind Tunnel II: A Fast and Portable Parallel Architecture Simulator.* Workshop on Performance Analysis and Its Impact on Design, 1997

[40] *http://www.spec.org/benchmarks.html*

[41] *http://www.nas.nasa.gov/Resources/Software/npb.html*

[42] Jaswinder Pal, S., W. Wolf-Dietrich, G. Anoop. *SPLASH: Stanford parallel applications for shared-memory*. 1992, ACM. p. 5-44

[43] KleinOsowski, A. D.J. Lilja, *MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research.* Computer Architecture Letters, 2002

[44] Aashish Phansalkar Ajay Joshi Eeckhout, L.J., L.K. *Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites*, in *Performance Analysis of Systems and Software, 2005. ISPASS 2005. IEEE International Symposium on*. 2005: Austin, TX

[45] Joshua, J.Y., J.L. David, M.H. Douglas. *A Statistically Rigorous Approach for Improving Simulation Methodology*, in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*. 2003, IEEE Computer Society

[46] Gurindar, S.S., F. Manoj. *High-bandwidth data memory systems for superscalar processors*, in *Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*. 1991, ACM: Santa Clara, California, United States

[47] Gary, L. *Accelerating Architectural Simulation by Parallel Execution of Trace Samples*. 1993, Sun Microsystems, Inc

[48] Wunderlich, R.E.W., T.F.; Falsafi, B.; Hoe, J.C. *SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling.* Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on, 2003

[49] Sherwood, T., et al. *Automatically characterizing large scale program behavior*. 2002, ACM SIGOPS Operating Systems Review ACM SIGOPS Operating Systems Review

[50] Bagrodia, S.P.a.R. *MPI-SIM: Using Parallel Simulation to Evaluate MPI Programs.* Winter Simulation Conference, 1998: p. 467-474

[51] Gengbin, Z., et al. *Simulation-based performance prediction for large parallel machines.* Int. J. Parallel Program, 2005. **33**(2): p. 183-207

[52] Berkbigler, K.P., B.W. Bush, K. Davis *An Approach to Extreme-Scale Simulation of Novel Architectures.* Proceedings of the Conference on Systemics, Cybernetics, and Informatics (SCI'02),

2002

[53] http://www.ssfnet.org

[54] Kalyan S. Perumalla, P.D. *Ultra-scale Parallel Discrete Event Applications.* BGW Consortium Days Report, 2006

[55] John, S., Bucy, R. Gregory, Ganger. *The DiskSim Simulation Environment Version3.0 Reference Manual.* reports-archive.adm.cs.cmu.edu/anon/2003/CMU-CS-03-102.pdf 2003

[56] http://www.cs.cornell.edu/skeshav/real/overview.html

[57] Jia, W. , K. Srinivasan. *Efficient and Accurate Ethernet Simulation*, in *Proceedings of the 24th Annual IEEE Conference on Local Computer Networks*. 1999, IEEE Computer Society

[58] Penry, D.A., et al. *Exploiting Parallelism and Structure to Accelerate the Simulation of Chip Multi-processors* the Twelfth International Symposium on High-Performance Computer Architecture (HPCA), 2006

[59] Matthew, C., G. Alan *Parallel simulation of chip-multiprocessor architectures*. 2002, ACM. p. 176-200

[60] Kanaujia, S., et al. *FastMP:A Multi-core Simulation Methodology*

[61] R. Plackett ,J. Burman. *The Design of Optimum Multifactorial Experiments*, Biometrika, Vol 33, Issue 4, June 1956, Pages 305-325

作者简介：

许建卫： 中国科学院计算技术研究所博士研究生
杨 伟： 中国科学院计算技术研究所硕士研究生
潘晓雷： 中国科学院计算技术研究所硕士研究生
郑 规： 中国科学院计算技术研究所硕士研究生
赵健博： 中国科学院计算技术研究所硕士研究生
陈明宇： 中国科学院计算技术研究所研究员、硕士生导师

信息技术快报网站介绍

时值元旦佳节，《信息技术快报》的全体成员祝您元旦快乐，在新的一年里身体健康、工作顺利！感谢您在过去的日子里对《信息技术快报》的关注和厚爱，也正是有了您的支持，《快报》才能茁壮地成长。

信息技术快报网站[http://lib.ict.ac.cn/ITL/index.asp]建于 2007 年 5 月，并于 2007 年 12 月进行了第一次网站改版。在计算所领导和快报编委领导的大力关怀下，网站围绕报道计算所科研工作的发展动态，不断完善管理机制，丰富网站内容，增强网站功能，改进服务方式，使网站水平不断提高。我们将 2003 年至 2007 年的《信息技术快报》内容登载在快报网站上，使您既可以单篇检索，又可以全文浏览。快报网站延续了《信息技术快报》的一贯的指导思想：将始终把宣传最新信息技术发展动态放在首要位置，也将始终把宣传计算所科研人员的学术观点和创新思想放在重要位置，力求成为政府的决策者、广大计算机界同行及各个企业了解计算所科研工作的窗口、交流学术研究成果和展望信息技术战略发展的更快捷、更有效的平台。

未来，我们将会进一步统筹规划网站的内容，并分步加以实施。欢迎您登陆我们的网站，给我们提出宝贵意见，同时也关注我们的成长与发展！

《信息技术快报》编辑部
2008 年 1 月 1 日