**高通量众核并行模拟**

**加速技术研究**

**方国庆1,2, 李文明2,3, 余洋1, 张洋2,3,叶笑春2,安虹1**

1(中国科学技术大学大学 计算机科学与技术学院，合肥 230027)

2(中国科学院计算技术研究所计算机 体系结构国家重点实验室 北京 100190)

3(中国科学院大学 北京 100049)

**摘　要**：软件模拟器是体系结构研究的重要工具，对体系结构发展有着不可替代的推动作用，而软件模拟器中最为关键的指标是模拟速度，尤其是在大规模众核结构模拟中，模拟速度更是至关重要。本文基于高通量众核结构模拟平台，提出了一系列模拟加速技术：首先，基于优化单条指令或指令块的思路，本文提出了查找表方法对指令译码进行了加速；其次，基于提高模拟器并发度和模拟平台并行性的思路，本文设计实现了并行离散事件模拟框架，并从消息映射、队列无锁化以及模拟进度控制三个方面进行了优化；最后基于事件模拟中存在大量内存申请和释放操作的特征，本文提出了内存池管理方案，提高了动态内存管理的效率。实验以典型高通量应用wordcount, terasort, kmp等基准测试程序进行评估，结果表明，查找表、并行离散事件模拟和内存池三种加速方案分别将相应阶段的模拟速度提高 26.14X，3.94X和12.47％。

**关键词：**高通量处理器；众核模拟器；查找表；并行离散事件模拟；内存池

Research of Acceleration Techniques for Many-core Parallel Simulation Fitting for High-Throughput Applications

FANG Guoqing1,2, LI Wengming2,3, YU Yang1, ZHANG Yang2,3，Ye Xiaochun2, AN Hong1

1(School of Computer Science and Technology, University of Science and Technology, Hefei 230027, China)

2(State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, 100190, China)

3(University of Chinese Academy of Sciences, Beijing, 100049, China)

**【Abstract】**As a key tool in the research of computer architecture, software simulator plays an irreplaceable role in promoting its development. It is critical to speed up architecture simulators. To accelerate execution of single instruction or instruction block, the lookup table scheme is applied in the decode stage of pipeline. To make use of concurrency in simulator and parallelism in host machine, we introduce the method of Parallel Discrete Event Simulation(PDES), and optimize it in aspects of mapping between messages and working threads, lock-free implementation of message queue and control of simulating process. Meanwhile, based on the discovery of massive memory allocation and free during message processing, the memory pool scheme is introduced to speed up the dynamic memory management scheme. High-throughput applications including wordcount, terasort and kmp are used as benchmarks in evaluation of the system. Experiments show that lookup table, PDES and memory pool scheme speed up as much as 26.14X, 3.94X and 12.47% in simulating their corresponding stages.

**【Key words】** high-throughput processor; many-core simulator; loop-up table; PDES; memory pool

# **概 述**

云计算与大数据处理等新兴应用正在迅猛发展并占据互联网市场的高点。目前已经建立的大型数据中心从规模上已经接近甚至超过了最大的高性能计算系统。传统的科学计算具有任务单一、负载类型稳定、单个任务计算量大且计算局部性好的特点。与此不同，新兴的数据中心应用主要面向互联网和物联网的服务，其特点是任务多样，负载变化频繁，单个任务计算量不大，但任务的数量及数据规模巨大，性能衡量指标主要关注的是系统的整体吞吐量，这类应用我们称之为高通量应用。相应的，区别于高性能计算缩短单个并行计算任务运行时间的研制目标，数据中心类应用系统的目标则是要提高单位时间内处理并发请求的数目[1][2]。但是到目前还没有设计出成熟的面向高通量计算的计算机系统[3]。

作为体系结构研究中的主要工具，高通量计算机系统的研究也往往先基于模拟器进行评估。随着众核体系结构各功能部件数量与实现复杂度的增加，传统串行模拟器因为速度的原因已经严重不适于大规模众核处理器的模拟，成为限制模拟器发展的主要因素。例如，目前常用的串行模拟器GEM5[4]、MARSS[5]，模拟速度大约在200KIPS左右。基于此速度，模拟一个真实物理核的一秒钟大约需要几个小时，模拟一千个核的一秒钟，几乎需要一年的时间。而与串行模拟器相对的并行模拟器能并发模拟功能部件执行状态，显著提高模拟速度，被广泛用于众核体系结构的模拟。然而目前大规模并行模拟器的开发仍然面临诸多挑战。

为提高体系结构模拟器模拟速度，本文从指令译码、并行离散事件模拟和内存操作三个角度对模拟器进行加速研究。本文的主要贡献如下：

* 查找表技术。基于提高单条指令或指令块执行速度的思路，提出静态查找表技术以充分利用模拟中相对充足的内存空间资源，减少花费在计算上的时间，以争取更快的执行速度。
* 并行模拟框架。基于提高模拟器并发度的思路，模拟器使用组件化的设计，不同组件之间如果需要通信，则发送消息到模拟框架，由框架根据所属组件的类型，调用相应组件的处理算法完成对消息的处理或转发。基于利用模拟平台并行性的思路，对组件间海量消息的处理可以利用宿主平台的多核资源完成，每个核承担一部分消息的处理，整体执行时间将显著减少。
* 内存池技术。模拟器运行过程中会有大量旧消息的释放和新消息的产生。实验中我们发现，随之产生的大量内存空间的释放和申请将成为系统性能提升的瓶颈。本文使用内存池的方案优化组件内存空间的管理，从而消除了大量的内存空间的释放和申请。

# **相 关 工 作**

众核体系结构规模日益庞大，大规模并行体系结构的模拟变成巨大的挑战。目前，大多数模拟器都是串行执行。串行模拟器用一个主机线程来模拟整个目标系统，当目标系统的核数增加时，分配给单个核的模拟性能就会下降。现有的模拟器加速技术主要分为三类：

1. **提高单条指令或指令块的执行速度**

执行驱动的模拟器，其执行的最基本对象是一条条指令。直观上看，如果可以提高每一条指令的执行速度，那么整个模拟器的运行时间也将会显著减少。这类加速技术包括各类软件调优及编译优化技术、直接执行[6]、二进制翻译技术以及FPGA硬件加速[7]。例如，QEMU[8]模拟器使用指令二进制翻译和翻译块链接的技术大大提高的模拟器的执行速度；使用FPGA的仿真方案也已经得到的越来越广泛的使用[7]。

1. **减少模拟指令数**

减少模拟指令[9]是指在模拟程序的过程中并不进行测试程序的完全性能模拟，而是将大部分指令使用功能模拟运行，仅挑选程序执行中的一部分来进行性能模拟并用这部分模拟的结果构造整个程序的模拟结果。减少模拟指令的方法有精简输入集、快速推进和采样。如simplescalar[10]模拟器提供功能模拟与时序模拟两种方法，在使用完全的功能模拟、结合SimPoint[11]提供的聚类功能后，可以提供良好的采样模拟方案，在保持高模拟精度的情况下大大缩短模拟时间。

1. **提高模拟器并发度和利用模拟平台并行性**

这种方法的思路是使用并行模拟算法或者利用模拟平台的多核资源加速模拟过程。藤本等提出的并行离散事件仿真[12]技术，通过局部的同步达到整个系统同步的效果，大大提高模拟器的并发度。在串行模拟速度受所在宿主机主频增速放缓而停滞不前的情况下，传统串行模拟方式无法满足对新兴体系结构模拟规模和速度的需求。随着近年单处理器向多核、多处理器方向的迅猛发展，充分利用现有并行化资源加速模拟速度和精度已经被证明是必要且高效的。利用并行技术对模拟器进行加速有很多实际例子，比如GAS[13]模拟器在使用并行离散事件模拟方法对众核模拟进行加速后，模拟精度几乎不变的前提下在16个核上获得了10.9倍的速度提高。

# **优化技术研究**

1. **高通量众核模拟平台**
2. **查找表技术**

在计算机科学中，静态查找表是用简单的查询操作替换运行时计算的数组或者关联数组。由于省去了不必要的函数调用、分支转移及复杂算术运算，使用静态查找表的方案通常能取得显著的加速。

在模拟器实现中，查找表技术应用于以下三个方面；PopCount算法的优化、指令条件域检查以及软件模拟踪迹缓存。

1. PopCount

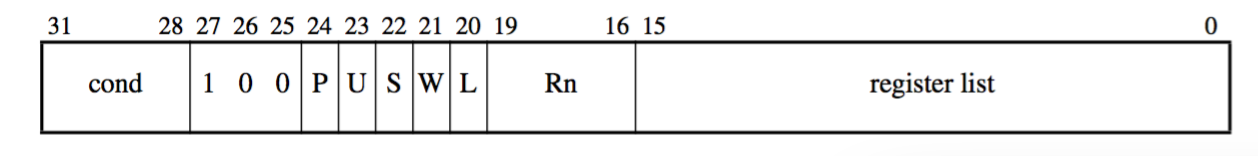
ARMv6指令集中Load/Store Multiple指令[14]的格式如图 1所示。

图 1 Load/Store Multiple指令格式

其中register list域的每一位对应一个通用寄存器，bit0指代R0,bit15指代R15（即PC寄存器）。在计算指令执行结束后内存地址address的过程中需要register list域中“1”的个数。优化前,模拟器使用md\_ones()函数计算给定32位整数的二进制表示中1的个数。如算法1所示：

**算法1 md\_ones()**

|  |
| --- |
| 算法: 统计register list域中1的个数 |
| 输入: 指令register域: val  步骤：  cnt = 0;  while (val != 0)  {  cnt = cnt + val & 0x1;  val = val >> 1;  }  输出：val的二进制标志中1的个数: cnt |

算法1在现代计算机架构上将需要数以百计的时钟周期才能完成。这是因为它造成了许多分支和循环，严重影响了执行效率。

对算法1进行加速的方法是构建一个包含256个条目的表，每个条目分别存储对应8位二进制数中“1”的个数。通过采用循环展开并截取register域的低16位可以进一步加速该算法。采用以上策略优化后，查询register域中“1”个数的算法2为：

**算法2 PopCount()**

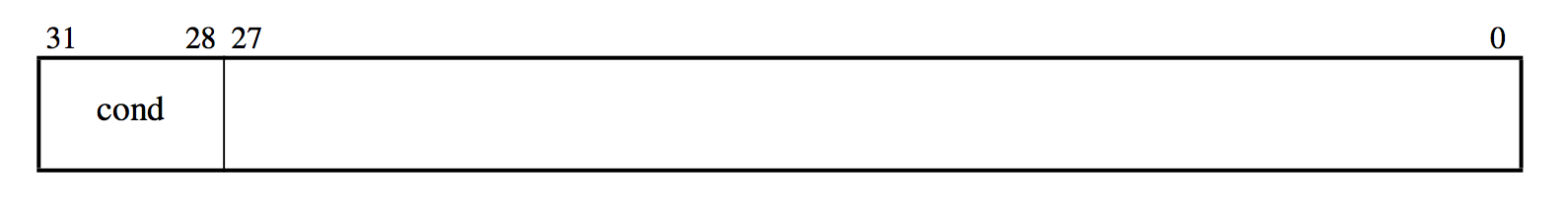
|  |
| --- |
| **算法: 统计**register list域中1的**个数** |
| **输入:** 指令register域: val  包含256个元素的查找表：tab  **步骤：**  cnt = tab [val & 0xff] +  tab [(val>>8) & 0xff];  输出：val的二进制标志中1的个数:cnt |

算法2不包含分支，仅有两次内存访问、几乎没有算术运算，大幅地提升了速度。

1. 指令条件域检查

大部分ARM指令可以被 “条件的” 执行，这就意味着仅在当前状态寄存器CPSR中的NZCV标志满足指令中指定的条件域时，该条指令才能够对编程模型状态、内存以及协处理器产生正常的影响。如果这些标志位不满足这个条件域，那么，该条指令就被当作NOP处理——像正常情况一样跳转到下一条指令执行，包含对中断和放弃预取的检查而不产生任何其他效果[14]。

如图 2所示，每条指令包含一个4位（指令的第28至31位）的条件域：



**图 2指令条件域**

因而在指令译码阶段的最开始都会涉及到对 condition field 的检查，如算法3所示，原有的解决方案是通过对4位条件域对应的16种取值使用switch选择后进行针对性的计算，得到对应的值。

这种方法的繁琐耗时之处在于：

1. switch 语句的匹配
2. 读取 PSR 寄存器的每个状态标记位，并进行一些逻辑运算
3. 函数调用

**算法3 md\_cond\_ok()**

|  |
| --- |
| 算法: 指令条件域检测 |
| 输入: 指令二进制码：inst  程序状态寄存器: psr  步骤：  condition = ((inst >> 28) & 0x0f);  switch (condition)  {  case COND\_EQ:  res = \_PSR\_Z(psr);  break;  case ....  .....  case COND\_GT:  res = ((!\_PSR\_N(psr) &&  !\_PSR\_V(psr) &&  !\_PSR\_Z(psr)) || (\_PSR\_N(psr) &&  \_PSR\_V(psr) && !\_PSR\_Z(psr)));  break;  case COND\_LE:  res = ((\_PSR\_N(psr) && !\_PSR\_V(psr)) ||  (!\_PSR\_N(psr) &&  \_PSR\_V(psr)) || \_PSR\_Z(psr));  break;  case COND\_AL:  res = TRUE;  break;  case COND\_NV:  res = FALSE;  break;  default:  panic ("bogus predicate condition");  }  输出：表征该指令是否被执行的标志res |

采用查找表的加速方案加速指令条件域判断实际上是基于以下的发现：

1. 条件域cond所有取值是有限的（16种）
2. 程序状态寄存器PSR的高四位nzcv的取值也是有限的（16种）

查找表枚举出所有可能的cond域与nzcv域配对时的结果。如 表 1所示。

表中横轴展示了所有条件域的取值，纵轴是状态寄存器中**NZCV**四个标志位的取值, 表的内容是一个布尔型变量，取值为**1**表示通过条件判断，该条指令将会被执行，反之将跳过该指令。

**表 1 COND域与NZCV匹配结果**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **COND** | | | | | | | | | | | | | | | |
| **EQ** | **NE** | **CS** | **CC** | **MI** | **PL** | **VS** | **VC** | **HI** | **LS** | **GE** | **LT** | **GT** | **LE** | **AL** | **NV** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **N**  **Z**  **C**  **V** | **0** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| **1** | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **2** | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **3** | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| **4** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| **5** | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **6** | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| **7** | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **8** | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **9** | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| **10** | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| **11** | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **12** | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **13** | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| **14** | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **15** | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

算法4描述如何借助于查找表，判断一条指令是否将被执行。

**算法4 md\_cond\_ok()**

|  |
| --- |
| 算法: 指令条件域检测 |
| 输入: 指令条件域的十进制表示: cond  程序状态寄存器中NZCV域: nzcv  存储所有256种匹配值的查找表: tab  步骤：  res = tab[nzcv][cond];  输出：表征该指令是否被执行的标志res |

一次内存访问，不含任何函数调用、分支跳转和算术运算。

1. 踪迹缓存

踪迹缓存（Trace Cache）是一个特别的指令缓存，它捕获动态指令序列。由于其每一行存储动态指令流的一个快照或者踪迹，这个结构被称作踪迹缓存。一个踪迹是动态指令流中一个包含最多n条指令并且以任意点起始的最多m个基本块的指令序列。其中n是一个踪迹缓存 行的大小，m是分支预测器的吞吐量。一个踪迹完全是由一个起始地址和最多m-1个描述后续路径的分支结果决定的[15]。

模拟器中，可以添加一个软件模拟的踪迹缓存结构来存储指令块的译码结果。第一次遇到一个踪迹时，会在踪迹缓存中分配一行。当指令被从指令缓存中取出时对应踪迹缓存行将被填充。如果程序执行过程中再次遇到同一个踪迹（起始地址相同且分支预测结果相同），踪迹缓存中相应的行就已经存在，它将被直接喂送到指令译码器中。否则，将按照正常渠道从指令缓存中获取指令。

软件模拟的踪迹缓存结构本质上是一种运行时构建的动态查找表。踪迹缓存机制省去了大量重复的取指译码过程，加速了模拟过程的推进。

1. **并行离散事件模拟框架**

并行离散事件模拟（Parallel Discrete Event Simulation, PDES）是指在一个并行计算机上执行一个离散事件模拟程序。历史上，PDES中非规则的、数据依赖的本质使其被认为是一类难以使用超级计算机硬件的向量化技术提供性能收益的应用[12]。并行离散事件模拟通过开发模拟模型中天然存在的并行性，显著提升了模拟器的性能和容量，使得在更少的时间内研究更耗时、更细化的模型成为可能。PDES是一个细粒度的并行应用，其性能和可扩展性受到通信延迟的限制。多线程方式的实现消除了许多消息拷贝并显著的减小了同步延迟[16]。

本文在多线程方式实现的并行离散事件模拟框架[17]基础上，针对使用中存在的性能瓶颈进行如下三个方面的加速。

1. 消息映射：从固定映射到随机映射

框架调度的对象是系统中所有组件产生的消息。在高通量众核结构模拟中，这些组件包括core组件、memory组件、cache组件和mesh组件，共计三千余个。加速前，组件按照其编号分成若干组，每组组件的消息分发给一个工作线程处理，如图 3所示。这种消息与工作线程的固定映射方式实现简单，且兼顾了组件大类之间的均衡性，但是在实际中可能会导致不同工作线程的工作量严重不均衡，由于组件的编号由其组件的类型及在拓扑中位置决定，同一类型的连续多个组件会映射到同一个工作线程上。并且，不同类型组件，其消息数量、延迟等属性特征差别较大。这两个原因很容易导致负载分配不均，多线程优势无法得到充分发挥。



**图 4 随机消息映射**



**图 3 固定消息映射**

优化后，框架采用带有消息分配器的随机映射的方式，如图 4所示。消息分配器是一个独立的线程，统一处理组件接收到的消息，并按照工作线程的负载情况进行分发。不再按照拓扑结构分配消息，每个工作线程可能执行到任何一个组件的消息，每个组件的消息也可能被任何一个工作线程处理。由组件类型差异而导致的工作线程负载不均衡问题便不复存在。

1. 消息队列：从全局互斥锁到无锁队列

各个工作线程与主线程之间通过消息队列进行通信。工作线程从消息队列中取出一组要在本轮循环处理的消息，并在处理完成后，带回新产生的消息。初始的机制是使用一个全局的消息队列，并使用一个互斥锁隔离各个线程的消息存取。这种大锁结构带来的问题是：在消息粒度较小而使得消息存取较频繁的情况下，多个线程将会频繁地申请该锁，存取消息的过程将变得严重串行化，成为消息调度功能的瓶颈。

优化后，框架为每对“主线程－工作线程”的每个方向（主线程转发消息到工作线程以及工作线程转发消息到主线程）提供了一个循环消息队列，基于多读者单一写者模型[18]，可以不使用锁实现每个线程对消息的同步。无锁化的实现降低了消息的存取开销，提高了框架的并行性。

1. 同步算法：从CMB算法到cycle-by-cycle模型

并行离散事件模拟的推进要求在不同逻辑单元之间高效地维护事件发生的先后关系。根据是否严格按时间戳顺序来处理事件，PDES的同步算法分为保守同步和乐观同步。为了在保证精确性的基础上实现并行模拟，本文中框架的实现采用保守的同步算法。

CMB算法是经典保守同步算法，其基本思想是组件之间按非递减的顺序发送消息，并且为每一对连接维护一个消息接收队列，并记录队列中消息时间戳的最小值，通过最小时间戳的不断增长，整个系统得以向前推进。如果队列为空，那么队列时间则是最后一个收到的消息的时间戳[17]。

基于CMB算法更新所有组件的最小时间戳是一个循环迭代的过程。在对象数目较多且上下游关系形成若干长链或者长环的情况下，迭代的次数将快速增加，以至于成为了模拟框架的性能瓶颈。

优化后的框架，使用红黑树按照时间戳递增的顺序管理消息队列。在主线程的统一调度下，工作线程逐个时间戳地执行所有消息。主线程的工作只是简单的从消息队列中分发最小时间戳的消息给工作线程，并将返回的新消息纳入消息队列的管理之中。

1. **内存池**

动态内存管理方案如C中的malloc或C++中new操作受限于变化的块大小导致的碎片，当需要频繁操作时在实际系统中性能表现不佳。一个更加高效的解决方案是预先分配一系列同等大小的内存块，称作内存池[19] 。

模拟器运行过程中会有大量旧消息的释放和新消息的产生。实验中我们发现，随之产生的大量内存空间的释放和申请将成为系统性能提升的瓶颈。本文使用内存池的方案加速消息内存空间的管理，内存的释放和申请的速度显著提升。使用内存池的优点包括：首先，内存池可以在固定的运行时间内分配内存。对于池中上千个对象的内存释放操作只需一个操作即可完成，而动态内存管理则需要为每个对象独立的释放内存。其次，区块尺寸固定的内存池不需要存储每次分配的元数据（用于描述诸如区块大小等特征），这提供了显著的内存节约功能，尤其是在小尺寸内存分配的情况下。最后，内存池可以按照层次化的树状结构[20]进行分组，非常适合某些特殊的编程结构，如循环和递归。

# **实验与结果分析**

1. **实验配置**
2. **实验环境**

支持模拟器运行的软硬件环境如表 2所示。

**表 2 宿主服务器配置**

|  |  |  |
| --- | --- | --- |
| **软件平台** | | |
| 操作系统 | CentOS Linux release 7.0.1406 (Core) | |
| 编译器 | GCC 4.8.2 | |
| **硬件平台** | | |
| CPU | | 4块Quad-Core AMD Opteron™ Processor 8347 HE |
| 每块4核，共16个物理线程 |
| L1 Cache | | 私有，每个核有64KB I-Cache和64KB D-Cache |
| L2 Cache | | 共享，每个CPU 512KB，共2MB |
| LLC  **图 5 组件纵向连接及mesh网络结构** | | 共享，每个CPU 2MB，共8MB |

1. **拓扑配置**

实验采用图 5所示的拓扑配置。图 5左侧展示了不同类型组件之间的连接结构：以router组件为核心，router组件往上依次是cache组件和core组件，下侧是memory组件。Router组件还剩余左右两个端口，用于与其他router组件相连，如图 5右侧所示。Router组件之间连接形成的片上网络的结构是可配置的，实验中使用的是32x32的mesh结构。Core组件运行时采用SMT＝8的配置，共计1024个core组件，8192个模拟线程。

1. **查找表**

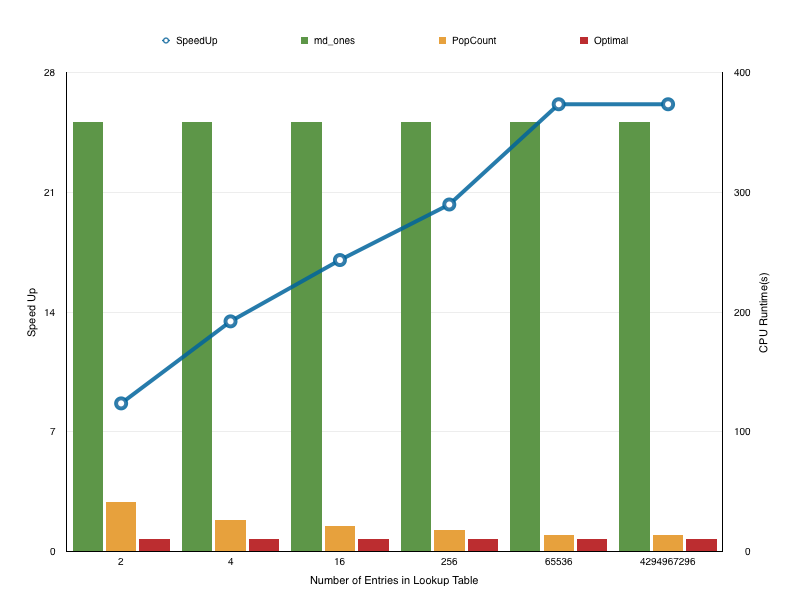
以PopCount问题为例，测试查找表方案的加速效果。统计2的30次方个随机数中二进制“1”的个数，统计不同算法下程序运行时间，并计算加速比，如图 6所示。

可以看到，md\_ones算法运行时间远大于PopCount算法，并且，随着查找表入口数逐渐增加（内存访问的次数逐渐减少），这种差距逐步扩大。最优的情况下PopCount算法可以达到26. 19倍的加速效果。并且已经非常接近于求解该问题所能达到的时间下界。

另一方面，构造查找表所需的工作量及占用的运行时内存空间与查找表入口数成反比，最优加速效果对应的内存空间占用达到4GB。综合考虑加速效果和查找表占用的内存空间，本文以256个入口数作为系统运行时的配置，在该配置下，查找表方案达到20.28倍的加速，仅占用256B的内存空间。

需要注意的是，由于存储墙的存在，内存访问的速度低于处理器运算速度，导致在查找表所替换的计算量很小的情况下，查找表的效果并不明显。一方面，从内存中提取结果需要更多的时间；另一方面，由于增大了所需的内存并且破坏了高速缓存，如果查找表太大，那么几乎每次访问查找表都会导致高速缓存缺失。这两方面原因共同限制了查找表方法加速效果的发挥。

1. **并行离散事件模拟**



**图 6 查找表尺寸与加速效果的关系**

使用7个线程运行并行模拟框架，统计多组测试程序在执行不同数量指令的情况下程序的最大加速比，结果如图 7所示：



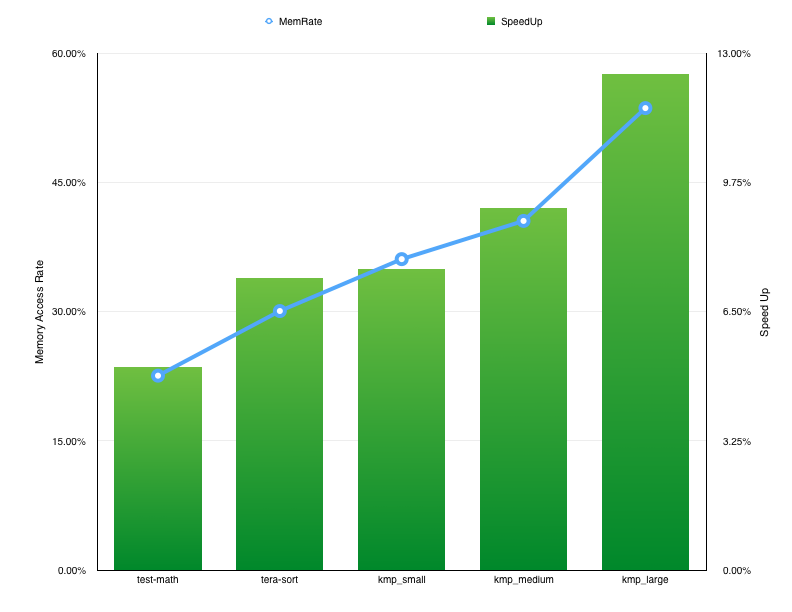
**图 7 PDES 框架加速效果图**

图中每个散点指示一个测试程序在指定的输入集下可以的最大加速比。观察可知，随着程序输入集的增大，同一个测试程序呈现出的加速效果也更明显。综合来看，优化后的并行离散事件模拟框架在7个工作线程的配置下，可以达到最大3.94倍的加速，平均加速比3.17。

1. **内存池**

测试多组应用程序及同一应用程序在不同数据集下运行时间，统计采用内存池方案的加速效果，如图 8所示。

在指令总数一定的情况下，访存指令的比例越高，程序运行时产生的消息数量越多，加速效果越明显。在访存指令的处理过程中，模拟器调用框架提供的消息发送功能向其他组件发送新消息，随之产生旧消息的释放和新消息的生成。因此，内存池的加速效果取决于测试程序中访存指令的比例。测试程序test-math属于计算密集型应用，其加速效果显著低于访存指令比例较高的tera-sort与kmp。并且，kmp程序的输入集越大，访存指令比例越高，加速效果越显著。



**图 8 访存指令比例及加速比**

# **结语**

本文在分析和总结常用体系结构模拟器加速技术的基础上，从查找表、并行模拟框架及内存管理三个不同方向对基于并行离散事件模拟框架的众核并行模拟器进行加速。代表性的高通量应用程序测试实验表明，三个加速方案效果显著。但是，实验中发现，当工作线程数量达到四个后，继续增加工作线程数量无法获得显著的性能提升。因此，下一步工作方向是提高并行模拟框架的可扩展性，或者在可扩展性难以提升的情况下动态确定最优工作线程数。

参考文献

1. 李国杰. 大数据对计算机系统的挑战[R]. CNCC大数据论坛, 长沙, 2013
2. Wang Yuanzhuo, Jin Xiaolong, Cheng Xueqi. Network Big Data: Present and Future[J]. Chinese Journal of Computers, 2013, 36(6):1125-1138 (in Chinese)

(王元卓, 靳小龙, 程学旗. 网络大数据: 现状与展望[J]. 计算机学报, 2013, 36(6):1125-1138)

1. Michael Ferdman, Almutaz Adileh, Onur Kocberber, et al. Clearing the clouds: a study of emerging scale-out workloads on modern hardware[C] //Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS`12). London, UK, 2012:37-48
2. N. Binkert, B. Beckmann, G. Black, et al. The gem5 simulator[C] //ACM SIGARCH Computer Architecture News, 2011, 39 (2):1-7
3. Avadh Patel, Furat Afram, Shunfei Chen, Kanad Ghose. MARSS: a full system simulator for multicore x86 CPUs[C] //Proceedings of the 48th Design Automation Conference (DAC`11). 2011:1050-1055

1. [Miller J.E.](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=p_Authors:.QT.Miller,%20J.E..QT.&searchWithin=p_Author_Ids:37275462900&newsearch=true" \o "), Kasture H., Kurian G., Gruenwald C., Beckmann N., Celio C., Eastep J., Agarwal A.. Graphite: A distributed parallel simulator for multicores[C] //IEEE 16th International Symposium on High Performance Computer Architecture (HPCA`10). Bangalore, India, 2010:1-12
2. Derek Chiou, Dam Sunwoo, Joonsoo Kim, Nikhil A. Patil, William Reinhart, Darrel Eric Johnson, Jebediah Keefe, Hari Angepat. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators[C] //Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO`07). Chicago, USA, 2007: 249-261
3. Bellard, Fabrice. QEMU, a Fast and Portable Dynamic Translator[C] //USENIX Annual Technical Conference, FREENIX Track. 2005.
4. 杨小溪，高晓彤，张为华. 若干体系结构模拟器加速技术的分析与对比[J]. 计算机应用与软件2011, 28(8):5-8
5. AUSTIN T, LARSON E, ERNST D. Simplescalar: An infrastructure for computer system modeling[J]. IEEE Transactions on Computers 2002,35(02).
6. Hamerly G,Perelman E,Lau J. SimPoint 3.0: Faster and more flexible program phase analysis[J]. Journal of Instruction Level Parallelism 2005,7(04).
7. Richard M. Fujimoto. Parallel Discrete Event Simulation[C] //Communications of the ACM, 33(10):30-53, Oct. 1990.
8. 吕慧伟, 程元, 白露, 陈明宇, 范东睿, 孙凝晖. 众核处理器和众核集群的并行模拟[J]. 计算机研究与发展, 2013, 50(5):1110-1117
9. ARM Architecture Reference Manual [EB/OL]. 8 Feb. 2016.http://poincare.matf.bg.ac.rs/~milan/download/micro/arm.pdf
10. Rotenberg, Eric, Steve Bennett, and James E. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching[C] //Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture. IEEE Computer Society, 1996.
11. Wang, Jingjing, Deepak Jagtap, N. Abu-Ghazaleh, et al. Parallel discrete event simulation for multi-core systems: Analysis and optimization[C] //Parallel and Distributed Systems, IEEE Transactions on 25.6 (2014): 1574-1584.
12. 李文明, 叶笑春, 张洋, 宋风龙，等. BDSim: 面向大数据应用的组件化高可配并行模拟框架. 计算机学报. 2015; 10:004.
13. Varma, Pradeep. "Two lock-free, constant-space, multiple-(impure)-reader, single-writer structures." U.S. Patent No. 6,304,924. 16 Oct. 2001.
14. Wikipedia. Memory Pool[EB/OL]. 8 Feb. 2016. https://en.wikipedia.org/wiki/Memory\_pool.
15. Cline, Robert C, and Daniel Garfinkel. "Method of managing memory allocation by association of memory blocks with a tree structure." U.S. Patent No. 5,247,634. 21 Sep. 1993.
16. L. Lamport. Specifying Concurrent Program Modules[C] //ACM Transactions on Programming Languages and Systems, 1983, 5(2): 190-222