PartitionSim：一个面向众核结构的并行模拟器

焦 帅[1),2)] 徐卫志[1),2)] 唐士斌[1),2)] 范东睿[1)] 孙凝晖[1)]

[1)](中国科学院计算技术研究所系统结构重点实验室 北京 100190)

[2)](中国科学院研究生院 北京 100049)

摘 要 该文提出了一个面向众核处理器的并行模拟器：PartitionSim. PartitionSim 采用了一种新颖的方法——时序分割来加速众核结构模拟. 时序分割的提出基于这样的观察：在众核结构中，有些模块之间频繁交互而有的模块之间没有交互. 有鉴于此，该方法将目标结构分割成两部分：交互部分和非交互部分. 当模拟交互部分时，主机线程严格同步，维持时序精确. 当模拟非交互部分时，主机线程通过异步运行，提高模拟速度，并且产生较小的时序损失. 文中所述工作在一个 16 核的 SMP 机器上用 PartitionSim 模拟了千核规模的 Godson-T 众核结构. 实验结果显示，PartitionSim 展示出良好的加速比，达到最高 25MIPS 的模拟速度，时序损失平均值为 0.92%.

关键词 并行模拟；众核；Godson-T；时序分割

中图法分类号 TP302 **DOI 号**：10.3724/SP.J.1016.2011.02084

PartitionSim：A Parallel Simulator for Many-Cores

JIAO Shuai[1),2)] XU Wei-Zhi[1),2)] TANG Shi-Bin[1),2)] FAN Dong-Rui[1)] SUN Ning-Hui[1)]

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

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

Abstract This paper introduces PartitionSim, a parallel simulator for future thousand-core processors with software-managed cache coherence. The purpose of PartitionSim is to improve the simulation performance of many-core architectures at the expense of little accuracy sacrifice. To achieve this goal, we propose a novel technique: timing partition. Timing partition is based on such an observation: in a target system, interacting components communicate with each other and impose simulation synchronization while non-interacting components don't communicate with each other and allow asynchronous simulation. It divides the target timing models into two groups: non-interacting group and interacting group. Non-interacting timing models are simulated by host threads that synchronize little with each other to improve speed and hurt little accuracy, while interacting timing models are simulated by host threads that synchronize strictly with each other to preserve accuracy. Using PartitionSim, We have simulated a target composed of thousands of cores on a 16-core SMP machine. The evaluation results show that PartitionSim scales well with near linear speedup and has considerable performance (up to 25MIPS) at the expense of little accuracy sacrifice (average 0.92%).

Keywords parallel simulation; multi-core; many-core; Godson-T; timing partition

---

收稿日期：2011-08-29；最终修改稿收到日期：2011-09-22. 本课题得到国家自然科学基金重点项目(60736012)、国家“九七三”重点基础研究发展规划项目基金(2011CB302500)、国家“八六三”高技术研究发展计划项目基金(2009AA01Z103)、国家杰出青年科学基金(60925009)、国家自然科学基金创新研究群体科学基金(60921002)、北京市自然科学基金(4092044)、“核高基”国家科技重大专项(2011ZX01028-001-002)资助.**焦 帅**，男，1984 年生，博士研究生，中国计算机学会(CCF)会员，主要研究方向为高性能计算机体系结构和并行算法. E-mail：jiaoshuai@ict.ac.cn. **徐卫志**，男，1982 年生，博士研究生，主要研究方向为高性能计算机体系结构、并行算法等. **唐士斌**，男，1982 年生，博士研究生，研究方向为高性能计算机体系结构、并行算法. **范东睿**，男，1979 年生，博士，副研究员，博士生导师，主要研究领域为众核处理器体系结构、高通量处理器体系结构. **孙凝晖**，男，1968 年生，博士，研究员，博士生导师，主要研究领域为计算机体系结构、高性能计算和分布式操作系统等.

1 引　言

随着“众核时代”的到来，一些众核处理器[1-2]已经出现.在单芯片上集成上千核已经可以实现[3]，并将在不远的未来商用化.模拟这样的并行结构是个巨大的挑战.目前，大多数模拟器都是串行的.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.

目前已经有许多方法被用来加速模拟.这些方法包括并行模拟[4-6]、直接执行[6]、FPGA 加速[7]等等.在这些加速技术中，并行模拟因为能够利用目标结构天然的并行特征，能在低成本 SMP 计算机上实现，成为了最有吸引力的加速技术.且有代表性的并行模拟工作是基于离散事件驱动的模拟器 PDES[8].

PDES 模拟器分为传统 PDES 和现代 PDES.传统的 PDES 模拟器注重精确性.它们或者保守地提供有限的同步放松(quanta[9]与 lookahead[10])，或者激进地放松同步但又引入额外的开销来应对时序错误.现代 PDES 模拟器(例如 SlackSim[5]和 Graphite[6])倾向于提高模拟速度，为此不得不激进地放松同步并牺牲精确性.

注意到传统 PDES 和现代 PDES 都展现出相同的同步行为：模拟目标结构所有模块时，要么都放松同步模拟，要么都严格同步模拟.然而通过仔细观察众核结构的通信特征，我们发现：有些模块间通信频繁，而有的模块则很少通信.这个观察在基于软件管理 Cache 一致性的众核结构(Intel SCC[2]、Rigel[3]和 Godson-T[11-12])中尤为明显.在传统的带有硬件 Cache 一致性的众核结构中，一致性协议要求频繁的 Cache 间通信.准确地模拟这些通信需要在每个 Cache 访问事件上做同步.在软件管理 Cache 一致性的情况下，Cache 只被本地的小核访问.Cache 间没有通信.模拟这样的 Cache 不需要同步.一旦 Cache 可以被异步模拟，那么整个小核(流水线＋Cache)就可以被异步模拟.

有鉴于此，本文的基本思想如下：当模拟一个软件 Cache 一致的众核结构时，放松对小核的同步模拟可以提高模拟速度并且损失很小的准确度.基于这个想法，本文提出了新的模拟方法：时序分割.时序分割是把目标时序模块分为两部分：非交互部分和交互部分.非交互部分被异步模拟，交互部分被严格地同步模拟.在本文中，我们把所提出的众核结构分为 Core(非交互)部分和 NOC(交互)部分.

要保证时序分割方法能够工作，有两个问题需要解决：

(1)虽然 Core 与 Core 之间没有时序通信，但是 Core 与 NOC 之间还有通信，必须去掉这些通信，才能让 Core 完全、自由地被异步模拟.

(2)异步模拟 Core 仍然会导致时序损失，如何减少这些时序损失.

为了解决上述问题，我们提出了两个方法：时序假设和时序重建.时序假设使用建模估计的方法来假设 NOC 到 Core 之间的时序通信.基于这些假设的时序，Core 就可以脱离 NOC 异步模拟.然而，由于 Core 基于假设的 NOC-to-Core 时序运行，那么产生的 Core-to-NOC 时序通信便不是准确的.时序重建用来尽量准确地恢复 Core-to-NOC 的时序通信.图 1 展示了本文的基本思想.

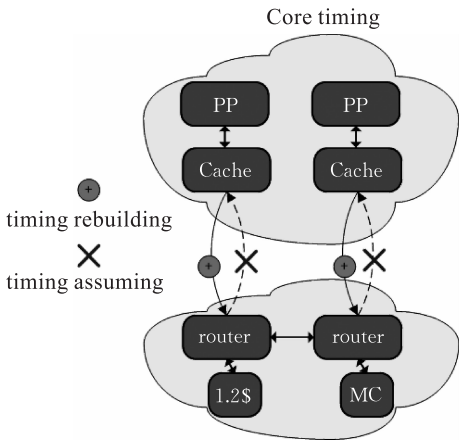


图 1　时序分割示意图(目标时序模块被分为 Core 和 NOC；时序假设(timing assuming)去除 Core 对 NOC 的时序依赖；时序重建(timing rebuilding)恢复 Core-to-NOC 的时序通信)

采用时序分割技术，我们构建了一个并行模拟器 PartitionSim，来模拟 Godson-T 众核架构.PartitionSim 使用 POSIX thread(Pthread)实现，运行在 SMP linux 上.通过运行移植的 SPLASH-2 应用程序，我们评估了 PartitionSim 在模拟千核 Godson-T 架构时的性能.结果显示，PartitionSim 具有良好的加速比，速度可达 25MIPS，平均时序损失为 0.92%.

本文第 2 节描述 PartitionSim 的系统结构；第 3 节讨论时序分割的核心思想；第 4 节讨论实验结果；第 5 节描述相关工作；然后第 6 节给出结论和未来工作.

2 PartitionSim 体系结构

PartitionSim 是一个面向众核结构的应用程序级并行模拟器. 图 2 展示了一个多线程应用程序如何被映射到一个众核处理器中,并由一个四核的 SMP 宿主机模拟. 如图 2 所示,目标众核处理器是典型的 tile 设计,包括两种 tile:Core tile 和 MC(Memory Controller 内存控制器) tile. 这些 tile 被一个片上 mesh 网络连接起来. 每个 Core tile 包含一个小核,一个共享 L2$ 和一个 router. 每个 MC tile 包含一个 router 和一个内存控制器. MC tile 放置在 mesh 网络两侧. 内存控制器的带宽可以配置以便模拟不同的片外带宽. PartitionSim 采用模块化设计,它包含一个功能模块和多个时序模块. 功能模块和小核的时序模块合起来称为 Core model,其它的时序模块:router、L2$ 和 MC 合起来称为 NOC model. 这些模块均来自 Godson-T 众核模拟器 GAS[13],其正确性和准确性都是被 Godson-T 原型芯片[12]验证过的.

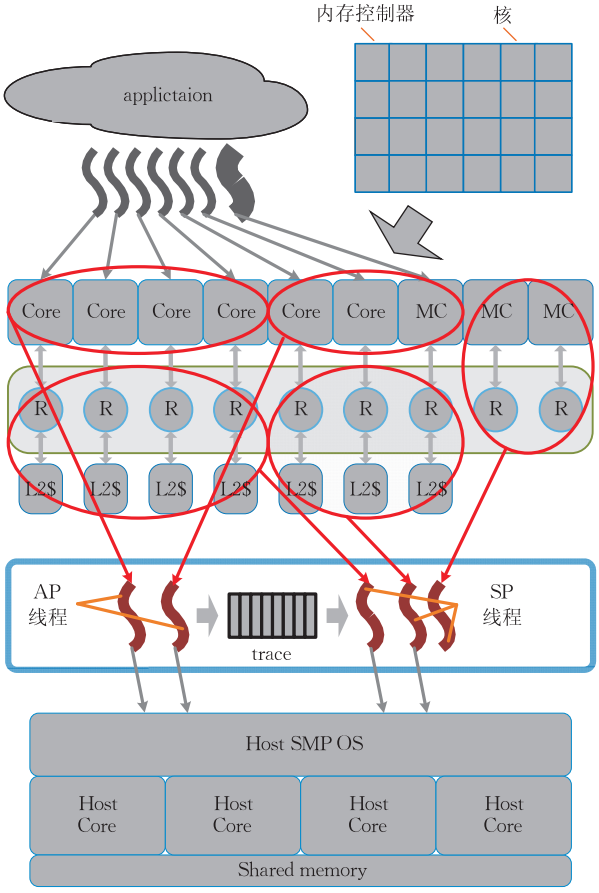


图 2 PartitionSim 的结构图

如图 2 所示,在 PartitionSim 里一共有两种线程:AP(Async-Parallel)线程和 SP(Sync-Parallel)线程. AP 线程模拟 Core model,SP 线程模拟 NOC model. 图 3 展示了 PartitionSim 的详细实现. AP 线程之间异步运行,SP 线程之间每个时钟周期都需要同步. 注意,当目标程序中有同步时,AP 线程也要同步以保证正确性. AP 线程输出 trace 给 SP 线程. SP 线程借助这些 trace 来完成 NOC 部分的时序模拟. 每个 trace 单元包含一个时间戳、事件类型和相关的地址或者数据. 在目前的设计中,AP 线程把 trace 写入文件,然后由 SP 线程读出来. 这样的内存—磁盘—内存的方式简化了模拟器的设计,同时又能方便评估 AP 线程和 SP 线程的时间开销. 另外,以文件形式存在的 trace 可以节省后续重复模拟的时间,这在模拟不同参数的片上网络时尤其有效,因为 Core 部分不需要重复模拟,只需要直接读入 trace 文件.

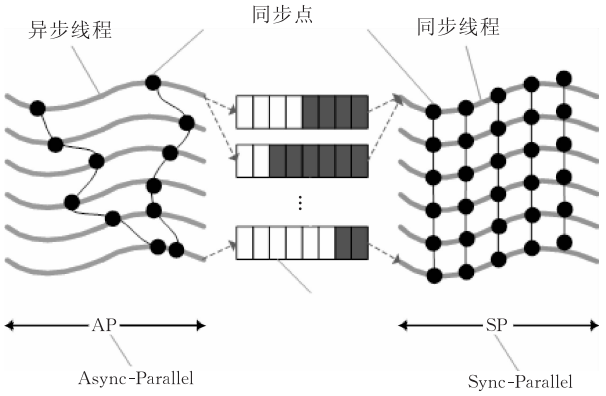


图 3 PartitionSim 的线程细节图

相应地,整个模拟时间也被分为 AP 时间和 SP 时间. AP 时间又被分为 APS(AP Sequential) 时间和 APP(AP Parallel) 时间. APS 时间指模拟器模拟目标程序串行阶段的时间. 目标程序串行阶段是目标程序只有一个活动线程的阶段. APP 时间是指模拟器模拟目标程序并行阶段的时间. SP 时间被划分成 SPW(SP Workload)和 SPB(SP Barrier)时间. SPW 是指 SP 线程的负载开销(Workload)时间,SPB 是指 SP 线程的同步(Barrier)开销时间. APS 部分被单个线程模拟,模拟时间不会因为宿主机的核数变化而变化. APP 部分由多 AP 线程异步并行模拟,是并行加速比的主要贡献者. 对于 SPW 和 SPB, 当主机核数增大时,SPW 时间因为并行模拟而减少,SPB 时间因为同步开销增大而增加.

3　时序分割

这一节讨论时序分割的关键：时序假设和时序重建. 下面，我们将以 Cache-Miss-Refill 事件为例来解释时序假设和时序重建的工作原理. 一个 Cache-Miss-Refill 事件包含一个 Cache-Miss (Cache 失命中)事件和对应的 Cache-Refill(Cache 回填)事件.

当一个 Core 时序模块发出一个 Cache-Miss 事件时，它会粗略计算相应的 Cache-Miss-Refill 延迟，包括片上网络延迟、L2$ 处理时间和可能的片外延迟. 在 PartitionSim 中，我们采用简单的算法来估计这些延迟：片上网络的延迟通过一个片上网络拥塞模型来计算；L2$ 的 miss/hit 信息和可能的片外延迟由简单的估计模型来预测. 在下文中，我们用 AL(Assumed Latency)和 RL(Real Latency)指代假设和真实的 Cache-Miss-Refill 延迟.

Core 计算出 AL 后，会根据 AL 来继续模拟 Core model，直到 Core 完全被阻塞或者 Cache-Refill 事件发生. 在这个过程中所模拟的目标时序被称为重叠时序(Overlapping cycles). 重叠时序可以很容易地从现代流水线处理器架构中观察到. 例如，当一个访存请求在 Cache 中发生 miss 并等待回填时，流水线可以继续执行流水线上的指令. 此时，核外时序和核内时序就会有重叠. 图 4(a)描述了重叠时序的一个简单示意. 接下来，为了方便讨论，我们会用 MO(Maximum Overlapping cycles)来表示一个小核在无限大回填时间情况下发生的重叠时序.

NOC 模块接收 Core 模块发来的 Cache-Miss 事件，重建它的时间戳，然后模拟它的 NOC 时序. 重建时序需要 Core 时序模块和 NOC 时序模块的协作. Core 维持一个本地时钟，本地时钟不包含重叠时序.

NOC 记录所有 NOC 延迟的总和，把它加到事件的时间戳上来重建其事件戳.

图 4(b)描述了一个时序重建的例子：Core 在本地时钟的第 3 拍发出了一个 Cache-Miss 事件；NOC 模拟这个事件的 Cache-Miss-Refill 延迟(时间戳为 10)并且记录下来为下次时序重建准备；于此同时 Core 继续模拟重叠时序，注意此时 Core 的本地时钟被关闭并且不记录重叠时序. 当重叠时序耗尽时，Core 重新打开本地时钟继续模拟. 在本地时钟第 8 拍，Core 发出第 2 个 Cache-Miss 事件(时间戳为 8). NOC 模块重建其时间戳为 18(18＝8＋10)，然后再用它去模拟 NOC 延迟.

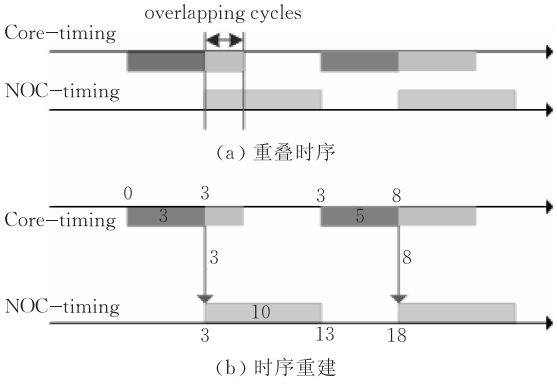


图　4

当 MO ＜AL 并且 MO ＜RL 时，也就是说，不管是在假设的时序下，还是在真实时序下，一个 Core 都会在 Cache-Miss 后，Cache-Refill 前阻塞. 此时，AL 和 RL 的差别并不影响时序重建的结果，因为两种情况下的重叠时序是相同的. 实时上，现代众核结构多采用简单的顺序小核和通用的互联网络. 在这种情况下，核外访存的延迟(NOC 延迟)往往要大于小核可以运行的重叠时序. 在这种情况下，本文提出的时序重建工作就能准确地恢复 Core-to-NOC 的通信时序.

在某些情况下，Core 在阻塞前连发两个 Cache-Miss(例如，指令 Cache-Miss 和数据 Cache-Miss). 在这些情况下，Core 会假设相应的 Cache-Refill 事件是串行发生，并且间隔足够的时间. 虽然这个假设是不准确的，但这种时序损失是有限的，因为这种情况是小概率事件. 在现代处理器中，L1$ 的命中率经常超过 95%，指令和数据 Cache-Miss 的概率更小.

4　实　验

这节介绍相关实验和结果. 4.1 节描述了主机和目标结构配置. 4.2 和 4.3 节讨论 PartitionSim 的模拟速度和精确度. 4.4 节对 PartitionSim 和传统 PDES、Graphite 进行比较.

4.1　实验环境

本实验的宿主机是一个 16 核的 Intel(R) Xeon (R) E7420 CPUs，主频 2.13GHz，内存大小 128GB DRAM. 操作系统是 Red Hat SMP Linux，kernel 版本是 2.6.18. 模拟目标结构是 4 种千核 Godson-T 架构，分别命名为 T1024，T2048，T4096 和 T8192. 4 种结构的配置都列在表 1 中. 以 T1024 为例，1024 个

Core tile 和 64 个 MC tile 连接在一个 34×32 的片上网络上. 关于 Godson-T 和它的软件管理 Cache 一致性的实现可以参照文献[10,14]. 对于目标程序,我们模拟了 SPLASH-2 里 3 个具有代表性的程序(lu,radix 和 fft)和一个矩阵乘算法 matrix. 表 2 列举了这些测试程序的负载参数.

表 1 目标结构参数

目标程序	Private 参数	
	Mesh 大小	Core 数
T1024	(32+2)×32	1024
T2048	(64+2)×32	2048
T4096	(64+2)×64	4096
T8192	(128+2)×64	8192

注:时钟频率:1 GHz; L1 (D/I) Cache: Private, 32 KB (per core tile), 32 Byte line size, 4-way associative, LRU replacement; L2 Cache: shared, 128 KB(per L2$ tile), 64 Byte line size, 8-way associative, LRU replacement; OFF 芯片延迟: 150 cycles.

表 2 目标程序参数

目标函数	问题大小	线程规模
matrix	1024×1024	1K,2K,4K,8K
fft	16M points	1K,2K,4K
radix	64M keys	1K,2K,4K,8K
lu	1024×1024 (8×8 block)	1K,2K,4K,8K

4.2 模拟速度

表 3 列举了实验结果,包括指令数目、并行时钟周期和不同数目主机线程加速下的运行时间. 每个程序都在不同的架构上运行. 例如,fft_1024 即指 fft 创建了 1024 个线程,运行在 T1024 上. 运行结果对应的 MIPS(Million Instructions Per Second)值(指令数目/运行时间)显示在图 5 上. 模拟时间比例分布显示在图 6 上.

表 3 实验结果

目标应用	指令数/10^9	并行时钟周期/10^9	在不同核数情况下的模拟时间/s				
			1	2	4	8	16
radix_1024	20.5	0.055	18806	9525	4770	2873	1821
radix_2048	20.9	0.024	24829	11939	6130	3219	1708
radix_4096	21.7	0.017	35948	18466	8997	4459	2084
radix_8192	23.4	0.020	72947	36192	16349	7719	4347
fft_1024	29.2	0.129	32516	23194	18300	14109	12571
fft_2048	29.6	0.147	44952	29853	21400	16401	14649
fft_4096	30.7	0.131	64766	44020	28543	20434	15634
lu_1024	48.5	0.051	27596	14780	7650	4778	2757
lu_2048	91.1	0.048	54764	26545	13950	7421	4205
lu_4096	176.2	0.046	114893	53441	26175	13742	7090
lu_8192	346.2	0.046	276820	133236	55111	26444	13626
matrix_1024	15.1	0.015	10173	4941	2457	1310	766
matrix_2048	15.1	0.008	12842	5940	2824	1376	828
matrix_4096	15.2	0.005	14971	7809	3408	1468	679
matrix_8192	15.2	0.004	16751	9925	4768	1915	898

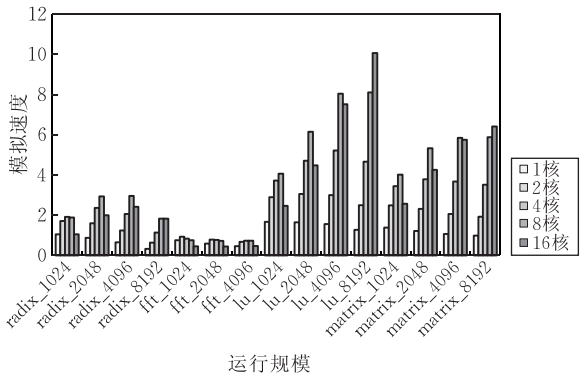


图 5 不同核数情况下的模拟速度

从图 5 可以看出:一个程序 APP 比例越大,获得的加速比越大. 如图 6 显示,目标程序的 APP 比例大小排序是:lu * p01>matrix_* p01>radix_* p01>fft_* p01. 图 5 中显示的加速比也正好符合这个顺序. 这是因为 APP 部分是完全异步模拟的,是并行模拟最受益的部分. 注意,fft_* 展示出最坏的加速比,这是因为它的 APP 比例最小,并且 APS 比例最大,因为 APS 部分是串行模拟的,并不能从并行模拟中获益.

尽管同步代价(SPB 的时间比例)随着主机核数增加而增加,但是加速比却没有显著降低. 这是因为 SPB 的比例相比其它部分小(图 6). 出现这个现象是因为:在千核情况下,每个主机线程要模拟上百个目标小核,庞大的工作量完全淹没了同步开销的效果.

4.3 模拟精度

图 7 展示了 4 个目标应用的模拟精确度. 精确度是通过与串行模拟的时序结果比较得来的(串行

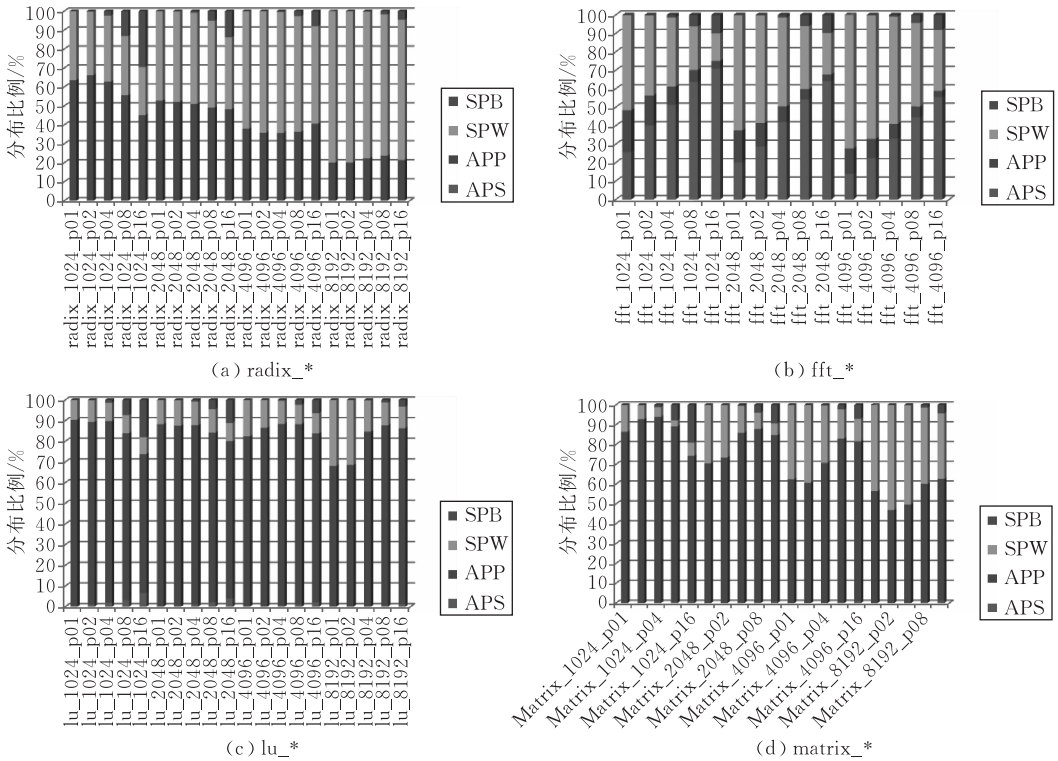


图 6　radix_*(a)，fft_*(b)，lu_*(c)，and matrix_*(d)的模拟时间分布

模拟的时序结果作为我们的“黄金”标准). 如图 6 所示，时序错误(Error)都小于 1.4%. 因为大多数 Cache-Miss-Refill 事件都符合 MO<AL 并且 MO<RL. 为了证明这样的解释，我们从 fft_1024，lu_1024，radix_1024，and matrix_1024 中分别采样了 1M 时钟周期. 在这些时间里，对每个事件记录 3 个参数：AL、RL 和 MO. 把所有目标小核的所有事件的这 3 个参数加起来形成了图 8 的结果. 从图中可以看出，MO<AL<RL. 这符合 MO<AL 并且 MO<RL.

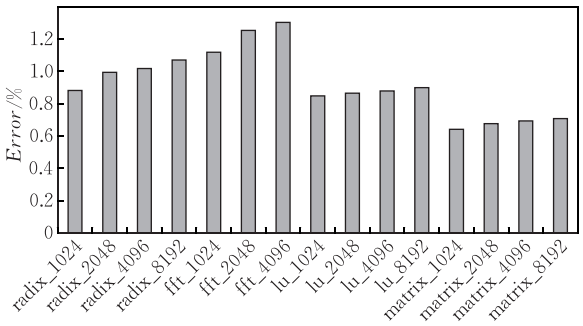


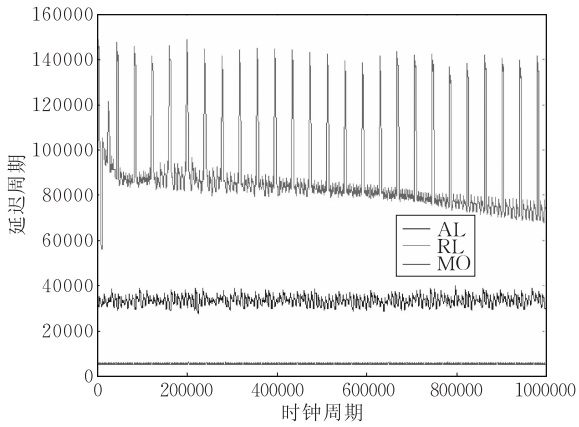
图 7　时序错误

一个有趣的现象是，fft_*虽然有最大的串行部分(APS)，但却表现出最大的时序错误. 仔细分析这个程序，我们发现，fft_*相比于其它程序，有更多时序违例的可能. 这是因为 fft 的程序特征：数据局部性差，片上通信频繁，长延迟指令最多. fft_1024 的第一级 Cache 的命中率为 94.3%，这要低于 radix_1024(98.9%)，lu_1024(99.9%)和 matrix(99.9%). 频繁的 Cache-Miss 事件会增大并发 Cache-Miss 的可能. 另外，在 fft 里，除法指令的比例约为 3%，在 Godson-T 小核中，一个单精度除法指令最多耗费 11 个时钟周期，一个双精度除法指令最多耗费 17 个时钟周期. 这些长延迟指令容易导致超过 AL 和 RL 的重叠时序. 这两种情况导致 fft*的时序重建效果相对较差，从而产生相对大的时序错误.

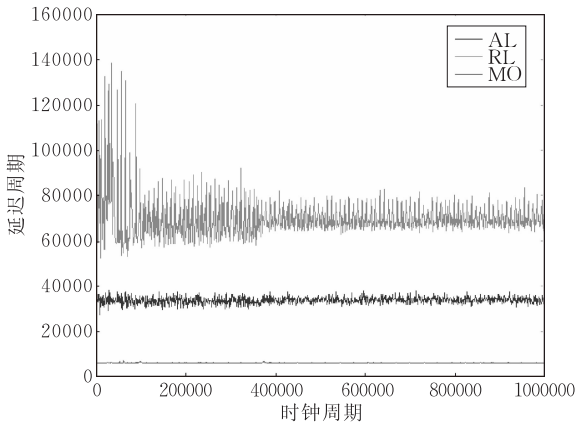
4.4　模拟比较

这一节列出了相关的比较实验. 第 1 个实验是比较 PartitionSim 和传统 PDES 方法，其中传统 PDES 方法以按拍同步的方式并行模拟. 如图 9 所示，PartitionSim 无论是在绝对速度(MIPS)还是在加速比上都比传统 PDES 方案要好. 这是因为传统 PDES 方法需要每个主机线程在每拍都要遍历所有相关目标模块(Core、Router 和 L2$)，导致很差的 Cache 利用率. 这一点可以从比较两种方法的串行模拟看出来. 如图 9 所示，串行模拟时，PartitionSim 要明显快于 PDES：radix_1024 快 76%，lu_1024 快 60%，matrix_1024 快 80%，lu_1024 快 200%.

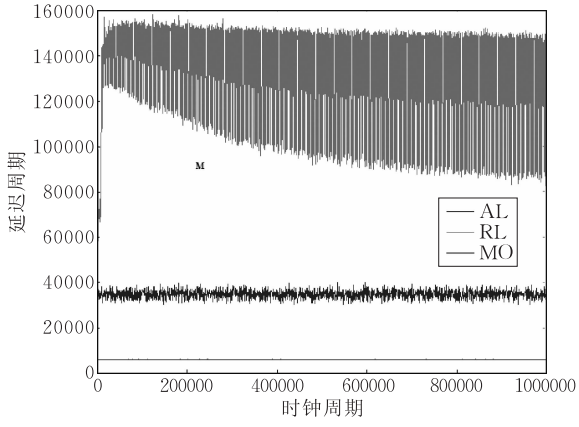
第 2 个实验是比较 PartitionSim 与 Graphite.



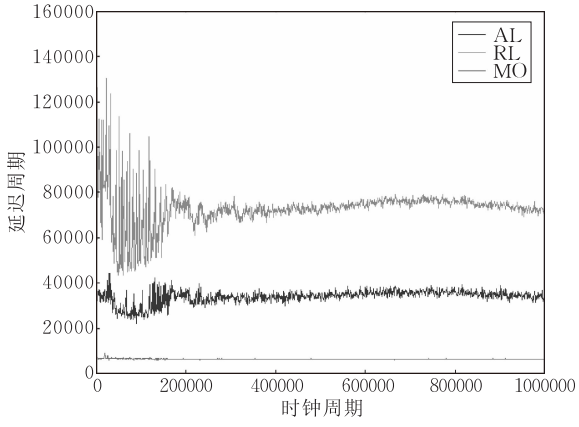
(a) matrix_1024



(b) radix_1024



(c) fft_1024



(d) lu_1024

图 8 4 个程序中采样的 1M 时钟周期中的 AL(上面),RL(中间)和 MO(下面)

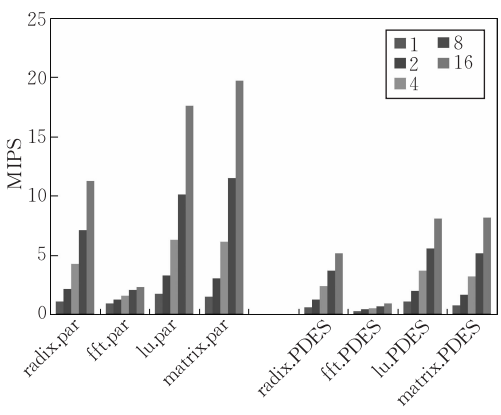


图 9 PartitionSim(.par)和传统 PDES(.PDES)模拟速度比较

图 10 显示了两个模拟器在 fft, radix, 和 lu 上的加速比. PartitionSim 的加速比是通过计算 *_1024, *_2048, *_4096和 *_8192的平均值得来的. Graphite 的加速比来自于文献[9]. 注意 Graphite 的 16 核模拟是在两个机器上实现的. 比较结果显示,PartitionSim 在单机情况下(1 到 8 核)比 Graphite 的加速比稍好. 对于精确度,虽然 PartitionSim 的精确度依赖于目标程序的特征,但其展现的平均时序错误仅为 0.93%,这要比 Graphite 低, Graphite 的时序错误在 Lax 情况下是 7.5%,在 laxP2P 情况下是 1.28%,在 lax-barrier 情况下是 1.31%. PartitionSim 的高精确度来自它的时序重建算法和对交互部分(NOC)的按拍同步模拟. 这种同步方式比 Graphite 的 lax,laxP2P,lax-Barrier 的都要严格.

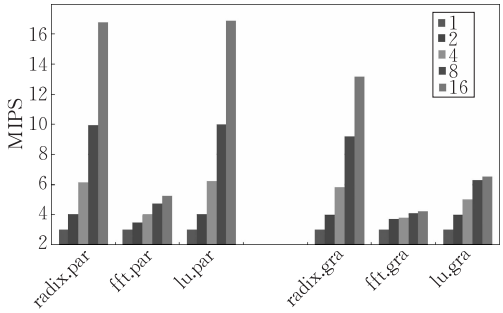


图 10 PartitionSim(.par)和 Graphite (.gra)的加速比比较

5 相关工作

模拟是开发新型计算结构的关键技术. 目前有很多模拟器,但大多数是串行的. 串行模拟器运行在

单个主机线程上，当目标结构上的核数越来越多时，分配给单个核的模拟能力就会下降. 目前的加速模拟技术包括：并行模拟[4-6]、直接执行[6]、FPGA 加速[7]等等. 在这一节，我们对与本文密切相关的工作进行讨论，包括：传统 PDES(并行离散事件驱动的模拟器)模拟器、SlackSim、Graphite 和 COTSon.

在传统的 PDES 模拟器里，主线程要经常同步以维持系统的正确性和准确性. 放松同步是加速 PDES 的主要方式. 基本上有两种放松方式：保守的和激进的. 保守方法非常小心地避免时序违例，并保证事件被完全顺序模拟. 这方面的放松方法包括文献[14]的时间桶同步方式、WWTII[9]的基于 quantum 的同步方式和文献[10]的提前量(lookahead)方式.

时间桶(time bucket)同步方式把模拟过程分成 T 周期的间隔，并行的线程每隔 T 个时钟周期同步一次. T 被定义为目标模块的最小交互延迟. 基于 quantum 的同步方式与时间桶的方式很相似. “quantum”也就是时间桶里的时钟周期间隔 T. 唯一的不同是 quantum 是动态计算而来的. 在提前量方法中，接收事件的 Core 会得到一个提前量 L，然后对这个小核就可以安全地模拟 L 个时钟周期而不需要同步.

激进的放松方法允许时序违例发生，但是一旦它检测到时序违例，就会发生回滚. 代表性模拟器包括 GTW[4]. 最著名的激进方法是 Time Warp[14]操作系统中采用的技术. 这种技术周期性地设置检查点，一旦发现时序违例就回滚到最近的检查点，然后以此为起点重起安全模拟.

虽然激进的 PDES 方案提供更多的放松，但是代价昂贵. 首先，必须有额外的时间和空间开销来保存检查点. 第二，额外的时间和空间来传递和保存额外的消息和事件(如 GTW 中的 anti-messages). 第三，回滚和重新执行都会耗费存储和计算资源. 另外，如果放松的尺度过大，时序违例和回滚就更容易发生，会降低模拟效率.

与传统 PDES 模拟器不同的是，现代 PDES 模拟器更加侧重速度. 典型的模拟器有 SlackSim 和 Graphite. 在 SlackSim 里有两种主机线程：Core 线程和 Manager 线程. Core 线程和 Manager 线程互相通信，构建一个时钟窗口. 当所有 thread 的本地时钟落在这个窗口里时，就可以自由运行. 当时钟差距超过这个窗口时，同步就会发生. 这种方法相比于传统 PDES 能产生更大的放松量. 但是，一个大的时钟窗口又会产生更多的时钟错误和违例. Graphite 采用了多级分布式模拟，从而提供更好的扩展性. 然而，跨机通信的高延迟不适合严格的同步方案. 所以 Graphite 不得不采用近似估计和建模的方法来构建“全局时钟”来提高精确度.

时序重建的概念首先由 COTSon[15]提出，COTSon 是一个支持功能-时序分割的模拟器. 在 COTSon 里，时序重建被用来重建自旋锁(spin-lock)的时序. PartitionSim 是一个支持时序-时序分割的模拟器. 在 PartitionSim 里，时序重建主要用来恢复被假设的时序通信. 另外，COTSon 的功能部分采用 AMD 的 SimNow!. SimNow!是个串行模拟器，它限制了 COTSon 的扩展性.

6　结论和下一步工作

这篇文章介绍了 PartitionSim，一个面向未来众核结构的模拟器，并提出了一个新颖的并行模拟方法来提高模拟速度并产生较小的时序损失. 实验显示，PartitionSim 在模拟上千核的 Godson-T 众核结构时有良好的加速比、可观的速度和较小的时序损失. 与传统 PDES 和现代 PDES(Graphite)的比较显示，PartitionSim 具有明显的速度和精度优势. 本文的工作集中在对 Godson-T 这样的软件管理的 Cache 一致性众核结构上，下一步我们将研究用时序分割的方法去模拟更通用的众核结构，如采用硬件 Cache 一致性的众核结构.

参　考　文　献

[1] Bell S, Edwards B, Amann J et al. TILE64 processor: A 64-core SoC with mesh inter-connect//Proceedings of the International Solid-State Circuits Conference. San Francisco, USA, 2008: 88-598

[2] Howard J, Dighe S, Hoskote Y et al. A 48-core IA-32 message-passing processor with DVFS in 45 nm CMOS//Proceedings of the International Solid-State Circuits Conference. San Francisco, USA, 2010: 108-109

[3] Kelm John H, Johnson Daniel R, Johnson Matthew R et al. Rigel: An architecture and scalable programming//Proceedings of the International Symposium of Computer Architecture. Saint-Malo, France, 2009: 140-151

[4] Das S R, Fujimoto R, Panesar K S, Allison D, Hybinette M. GTW: A time warp system for shared memory multiprocessors//Proceedings of the Winter Simulation Conference. Lake Buena Vista, USA, 1994: 1332-1339

[5] Chen J, Annavaram M, Dubois M. SlackSim: A platform for parallel simulations of CMPs on CMPs. ACM SIGARCH Computer Architecture News, 2009, 37(2): 20-29

[6] Miller J E, Kasture H, Kurian G et al. Graphite: A distrib-

uted parallel simulator for multicores//Proceedings of the 16th IEEE International Symposium on High-Performance Computer Architecture. Bangalore, India, 2010: 1-12

[7] Chiou D, Sunwoo D, Kim J et al. FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle accurate simulators//Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture. Porto Alegre, Brazil, 2007: 249-261

[8] Fujimoto R M. Parallel discrete event simulation. Communications of the ACM, 1990, 33(10): 30-53

[9] Mukherjee S S, Reinhardt S, Falsafi B et al. Wisconsin wind Tunnel II: A fast, portable parallel architecture simulator. IEEE Concurrency, 2000, 8(4): 12-20

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

[11] Fan D R, Yuan N, Zhang J C et al. Godson-t: A many-core processor for efficient multithreaded program executions. Journal of Computer Science and Technology, 2009, 24(6): 1061-1073

[12] Fan Dongrui, Wang Da, Fan Lingjun. High-efficient architecture of Godson-T many-core processor//Proceedings of Hot Chips 23. Palo Alto, USA, 2011(to appear)

[13] Lv Huiwei, Cheng Yuan, Bai Lu, Chen Mingyu, Fan Dongrui, Sun Ninghui. P-GAS: Parallelizing a cycle-accurate event-driven many-core processor simulator using parallel discrete event simulation//Proceedings of the 24th ACM/IEEE/SCS Workshop on Principle of Advanced and Distributed Simulation (PADS 2010). Atlanta, USA, 2010: 1-8

[14] Jefferson D, Beckman B, Wieland F, Blume L, Diloreto M. Time warp operating system//Proceedings of the 11th ACM Symposium on Operating System Principles. Austin, USA, 1987: 77-93

[15] Monchiero M, Ahn J H, Falcón A, Ortega D, Faraboschi P. How to simulate 1000 cores. ACM SIGARCH Computer Architecture News, 2009, 37(2): 10-19

JIAO Shuai, born in 1984, Ph. D. candidate. His research interests include high performance computer architecture and parallel algorithm.

XU Wei-Zhi, born in 1982, Ph. D. candidate. His research interests include high performance computer architecture and parallel algorithm.

TANG Shi-Bin, born in 1982, Ph. D. candidate. His research interests include high performance computer architecture and parallel algorithm.

FAN Dong-Rui, born in 1979, Ph. D., associate professor. His research interests include low-power architecture and processor micro-architecture.

SUN Ning-Hui, born in 1968, Ph. D., professor, Ph. D. supervisor. His main research interests include computer architecture, high performance computing and distributed OS.

Background

As the "many-core era" approaches, some many-core processors have already arrived. Thousand-core processor is no longer infeasible, and it is likely that thousands of cores on a single die will become a commodity. Simulating such parallel systems is a big problem. Currently, the majority of simulators available are sequential. Sequential simulators run the simulation workload on a single host thread. When the number of cores increases in a target system, the simulation performance for each core goes down.

Conventional PDES simulators are loyal to accuracy. They either conservatively provide limited synchronization relaxation (small quanta or look-ahead) or aggressively allow large scale of relaxation at the expense of introducing additional overhead to handle timing violations. Modern PDES simulators (e. g., SlackSim and Graphite), which intend to deliver speed increase, has to relax the synchronization condition between host threads and sacrifice accuracy. This paper introduces PartitionSim, a parallel simulator for future thousand-core processors with software-managed cache coherence. The purpose of PartitionSim is to improve the simulation performance of many-core architectures at the expense of little accuracy sacrifice. To achieve this goal, we propose a novel technique: timing partition. Timing partition is based on such an observation: in a target system, interacting components communicate with each other and impose simulation synchronization while non-interacting components don't communicate with each other and allow asynchronous simulation. It divides the target timing models into two groups: non-interacting group and interacting group. Non-interacting timing models are simulated by host threads that synchronize little with each other to improve speed and hurt little accuracy, while interacting timing models are simulated by host threads that synchronize strictly with each other to preserve accuracy.

This work is under the support of the State Key Program of National Natural Science Foundation of China (grant No. 60736012), the National Basic Research Program (973 Program) of China (grant No. 2005CB321600), the National High Technology Research and Development Program (863 Program) of China (grant No. 2009AA01Z103), the National Science Foundation for Distinguished Young Scholars of China (grant No. 60925009), the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (grant No. 60921002), the Beijing Natural Science Foundation (grant No. 4092044).