

众核体系结构对 Cilk 语言的硬件支持及评测研究

龙国平 张军超 范东睿

(中国科学院计算技术研究所系统结构重点实验室 北京 100190)

摘 要 如何编程众核体系结构是当前一个亟待解决的问题. 研究可扩展的硬件机制支持 Cilk 编程模型的目的是在良好的编程性和可扩展硬件实现之间达到平衡. Cilk 语言是 C 的精简扩展, 程序员编写 Cilk 程序时和串行编程近似, 且不需关心调度、负载均衡和局部性等系统底层相关的问题. 文中以域一致性存储模型为基础, 主要工作包括两方面: 首先针对域一致性模型编程性不好的缺点提出一种以数据为中心维护高速缓存一致性的方法; 其次提出实现 DAG Consistency 的缓存一致性协议, 并在此基础上支持 Cilk 编程模型. 实验结果表明, 当处理器核数目较少(<16)时所有测试程序都能获得比较好的性能加速, 并且指出了众核情况下(>16)难以获得理想加速效果的两个根本原因: 静态路由导致片上网络带宽利用不均衡以及有限的访存带宽.

关键词 众核体系结构; Cilk; 域一致性; 编程模型; 存储模型

中图法分类号 TP302 **DOI号**: 10.3724/SP.J.1016.2008.01975

Architectural Support and Evaluation of Cilk Language on Many-Core Architectures

LONG Guo-Ping ZHANG Jun-Chao FAN Dong-Rui

(Key Laboratory of Computer Systems and Architecture, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing 100190)

Abstract How to program many-core architectures is a critical issue. This paper studies scalable hardware mechanisms to support Cilk programming model, in order to achieve a good balance between programmability and scalable hardware implementation. The Cilk language is a simple extension to C, and writing Cilk programs is similar with sequential programming. In addition, programmers need not worry about system dependent issues, such as scheduling, load balancing and data locality, etc. This work is based on scope consistency. First a data centric approach is proposed for improving the programmability of scope consistency. Then architectural support for DAG consistency is proposed. Experimental results on a set of scientific benchmark programs show good performance speed up for a small number of cores. Experimental results also reveal two fundamental reasons which limit the performance scalability of Cilk computations on many-core architectures: the unbalanced on chip network bandwidth usage and limited memory bandwidth.

Keywords many-core architecture; Cilk; scope consistency; programming model; memory model

1 引言

集成电路工艺技术的进步使得将来的微处理器能够集成几百个甚至上千个处理核心. 近年来, 众核体系结构因其大规模并行处理能力得到了产业界^[1-3]和学术界^[4-7]的广泛关注. 前人基于众核结构的工作表明在科学计算和一些非规则计算领域, 经过手工程序优化能得到很高的性能. 我们之前的理论分析结果^[8]也表明众核体系结构对局部性好的稠密矩阵运算有着巨大的潜力. 尽管如此, 如何编程众核体系结构仍然是一个没有很好解决的问题.

在传统的共享存储体系结构中, 如硬件 ccNUMA 系统、SMP 系统以及多核系统中, 多线程编程模型具有十分重要的地位, 并且在操作系统内核、数据库管理系统和网络中间件等基础软件领域获得了广泛应用. 多线程编程以顺序一致性(sequential consistency)^[9-10]存储模型为基础, 通常需要基于目录或总线的集中式高速缓存一致性协议^[11]的硬件支持. 然而传统协议扩展性在众核上实现代价高昂: 首先现有目录/总线协议的语义都要求每当一个写操作执行时, 都要主动将多个旧的共享版本置为无效, 当共享的处理节点很多时且互联网络拓扑结构比较复杂时, 这一操作的开销很大; 其次就目录协议而言, 目录存储本身会带来很大开销, 此外协议的状态机维护代价也会随着处理节点增加而增长. 基于此, 我们认为将来的众核体系结构应当采用扩展性更好的硬件机制来解决高速缓存的一致性问题. 一种可扩展地支持多线程编程的方法是采用弱一致性模型, 如域一致性(scope consistency)^[12], 但编程语义也随之改变, 程序员需要手工标示出所有对共享数据的访问以保证正确的值传播^[13], 这在一定程度上失去了顺序一致性模型下编程的直观性, 给程序员带来了额外编程负担.

本文基于域一致性模型, 研究在众核体系结构上对 Cilk 编程模型的硬件支持. 以域一致性模型为基础是为了实现可扩展高速缓存一致性协议(第 2 节). Cilk^[14-16]是 MIT 的 Leiserson 等人提出的一种多线程编程语言, 是 C 的精简扩展. 选择在众核处理器支持 Cilk 语言的原因在于: (1) 任何一个 Cilk 程序都有一个串行语义, 程序员编写 Cilk 程序和串行编程类似, 一些串行程序很容易就能并行化成 Cilk 程序; (2) 程序员在编程时只需要把计算中存在的并行性表达清楚, 无需关心底层系统相关的调

度、复杂均衡和局部性等问题, 所有关于并行性的挖掘及系统相关的工作由 Cilk runtime 完成; (3) Cilk runtime 的基于任务窃取的随机调度(random work stealing scheduling)策略对 Cilk 程序能获得理论上优化的性能保证, 这对于性能分析和预测具有重要意义.

本文工作主要分为两部分. 首先是对 Cilk runtime 的支持. 为此, 我们提出一种以数据为中心维护高速缓存一致性的方法, 目的是改善域一致性模型的可编程性. 以此为基础, 我们顺利实现了基于顺序一致性的多线程 Cilk runtime (Version 5.4.6) 到基于域一致性的众核结构的移植. 其次是对 DAG 存储一致性模型的硬件支持, 目的是提供程序员友好 Cilk 编程环境. 需要说明的是, 基于域一致性的多线程编程模型只对 Cilk runtime 可见, 对应用程序员而言是不可见的, 应用程序员只需要用 Cilk 语言编写并行程序.

本文第 2 节定义本文研究所需要的众核体系结构平台 Godson-T V3; 第 3 节详细讨论域一致性模型编程性的改善问题; 第 4 节讨论对 DAG 存储模型的体系结构支持; 第 5 节介绍本文的评估环境; 第 6 节报告我们的实验结果; 第 7 节介绍相关工作; 最后一节总结全文.

2 Godson-T 众核体系结构

2.1 体系结构模型

本文的工作基于 Godson-T V3 的多线程体系结构. Godson-T 是中国科学院计算技术研究所目前正在研究中的一个面向较大规模并行处理的众核体系结构^[7]. 如图 1 所示, 64 个处理器核组织成为一个 8×8 的二维 MESH 结构. 四周分布着 16 个按照地址散列(interleaved)的二级缓存体(L2 Cache Bank), 4 个为一组, 每一组对应一个片上的内存控制器. 芯片内集成了 4 个 I/O 控制器和外设交互.

处理器核的基本结构如图 1 的右半部分所示, 是一种简单的静态双发射顺序执行的结构. 图中 Fetch&Decode 模块负责取指和译码; RFs 模块负责依赖分析, 寄存器堆访问和指令发射, 最多每拍可以发射两条指令到功能部件中, 功能部件里只有访存部件需要和 Router 和 Cache 交互. 图中 Coherence Vector 逻辑是本文为了改善域一致性编程性增加的逻辑, 在第 3 节详细讨论. 目前每个处理器核支持用户态 MIPS 指令集.

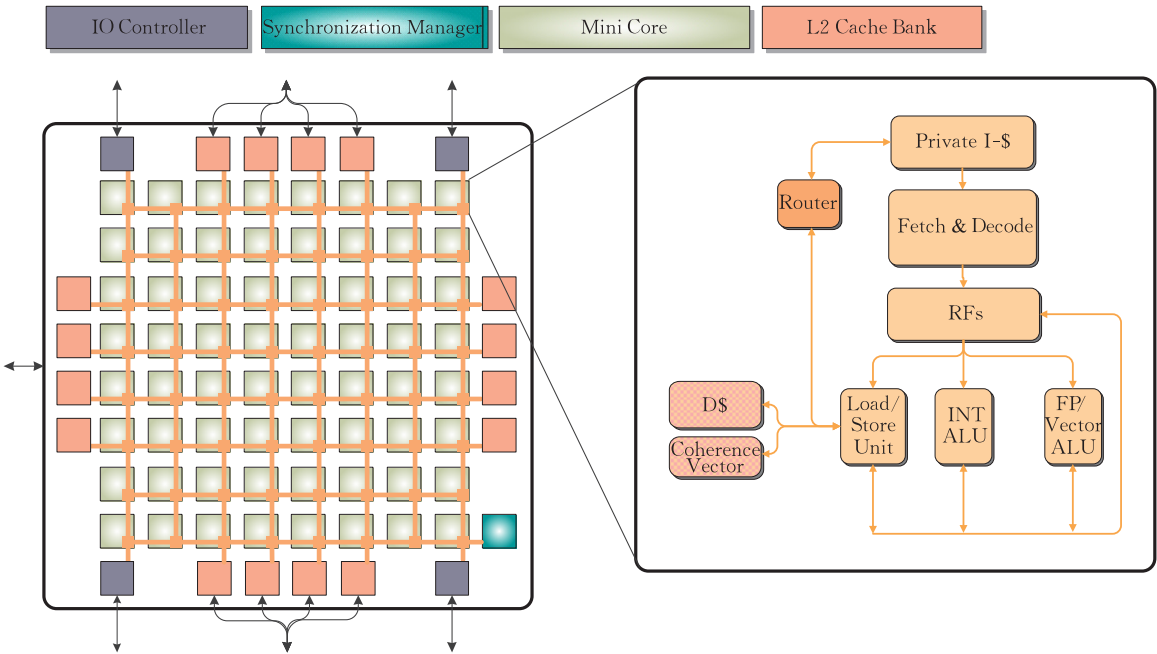


图 1 Godson-T V3 体系结构

Godson-T 片上网络的选路机制是静态 X-Y 路由,其具体工作方式如下:若处理器核 (x,y) 需要发送一个消息到处理器核 (m,n) ,那么该消息首先沿着垂直方向(X 轴)逐跳送至处理器核 (m,y) ,然后再沿水平方向(Y 轴)逐跳送到目的地 (m,n) 。片上消息的路由机制对片上网络的带宽利用率有很大影响(第 6 节)。

Godson-T V3 支持域一致性模型,由于 L2 缓存所有核共享,因此需要维护的是 L1 缓存之间的一致性。L1 缓存对于临界区外的访存操作而言是写回(write back)策略,硬件不维护临界区外数据访问的一致性。对于临界区内的访存操作需要分取数(Load)和存数(Store)操作分别处理。临界区内的所有存数操作必须写穿透(write through)到 L2 缓存。对于取数操作,如果是第一次访问某内存地址,则无条件放弃本地拷贝(本地拷贝有可能已经被其他

线程写过),转而从 L2 缓存取正确数据;如果是后续访问,则和普通缓存操作相同,一旦命中就从缓存中读取数据,只有不命中才去 L2 缓存取数。

2.2 编程模型

表 1 总结了所有和编程模型相关的指令,除 fence, acquire 和 release 操作之外,表中其余部分均为本文支持 Cilk 语言而新增的指令。acquire/release 指令需要一个锁标示符(lock_id)作为参数,这一点和传统对内存地址加锁略有不同,acquire/release 完全在片内实现,无须访问内存,只需要一个标示符指出不同的互斥临界区即可。bcr/ecr 声明一个需要硬件维护数据一致性的代码区,没有互斥语义,目的是让程序员用来编写域一致性模型上的多线程程序,类似于域一致性模型中的 open_scope/close_scope 操作。表 1 中最后 4 条指令将在后续内容详细讨论。

表 1 Godson-T V3 编程模型相关指令支持

指令	描述
fence	实现 fence 语义
acquire(lock_id)	启动一个互斥临界区
release(lock_id)	结束一个互斥临界区
bcr(region_id)	相当于域一致性模型的 open_scope,定义一个需要硬件维护数据一致性的临界区,bcr 操作没有互斥语义。
ecr(region_id)	相当于域一致性模型的 close_scope,结束一个需要硬件维护数据一致性的临界区,ecr 操作没有互斥语义。
remote_reconcile(src, dest)	请求一个远程处理器核将本地写过的(dirty)缓存行写到 L2 缓存中。
reconcile_flush	将所有写过的(dirty)缓存行写到 L2 缓存,然后将所有缓存行置为无效。
reconcile	将所有写过的(dirty)缓存行写到 L2 缓存中。
register_addr	将一个缓存行(cache line)对齐的内存地址注册到 Coherence Vector 中。

3 改善域一致性模型的编程性

3.1 动机

存储模型的弱化使得程序员需要承受额外的编程负担来维护数据一致性. 图 2(a)是一个生产者线程,其产生一组数据,然后设置标志位 flag 表示数据就绪;图 2(b)是消费者线程,其不断查询标志位 flag,直到被设置然后开始处理生产者生成的数据. 在这个例子中,首先两个线程可能同时访问 flag 变量,需要 acquire/release 来保证互斥访问;其次,无论是生产者线程还是消费者线程,在访问数据的时候

都需要 bcr/ecr 操作来保证数据的一致性,而事实上根据程序员的直觉,消费者处理数据(Consume Data)的时候表明生产者已经完成了数据生成(Produce Data). 让程序员手工维护数据的一致性不可避免地加重了编程负担.

域一致性模型还使得基于顺序一致性的多线程程序的移植变得困难. 虽然理论上可以通过编译器或者程序员分析所有程序代码找出所有对共享数据的访问,但是实际上难以实现. 特别是含有指针的行为复杂的程序,数据访问模式和程序输入有关,这就使得分析更为困难. 因此,有必要研究改善域一致性模型编程性的方法.

3.2 硬件机制

本文提出一种以数据为中心维护数据一致性的硬件机制:Coherence Vector(见图 1). 工作原理是:程序员或编译器在对共享数据的访问之前插入一段代码把共享数据的内存地址注册到 Coherence Vector 中,随后一旦有对共享数据的访问都能被硬件检测到从而触发一定的数据一致性维护动作. 由于 Coherence Vector 容量有限,而程序中的共享数据的信息量可能很大,我们将 Coherence Vector 组织成 Bloom Filter 的结构^[17],这样虽然存在误杀可能,但不会影响程序的正确性.

Coherence Vector 和访存部件的关系如图 3 所示. 访存指令发射进访存部件后首先在 addr 流水级计算地址,完成后同时访问数据缓存(data cache)和

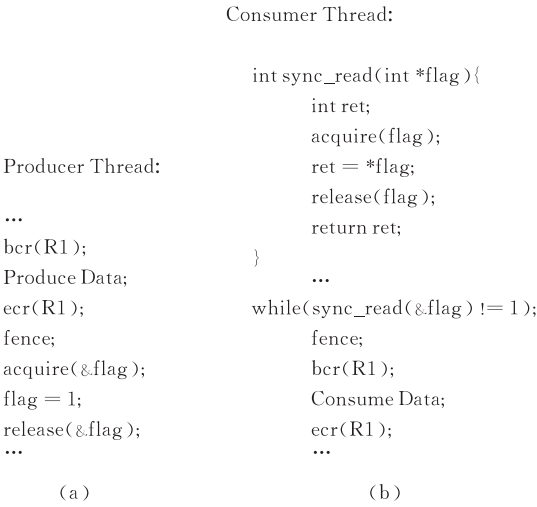


图 2 域一致性模型的编程性

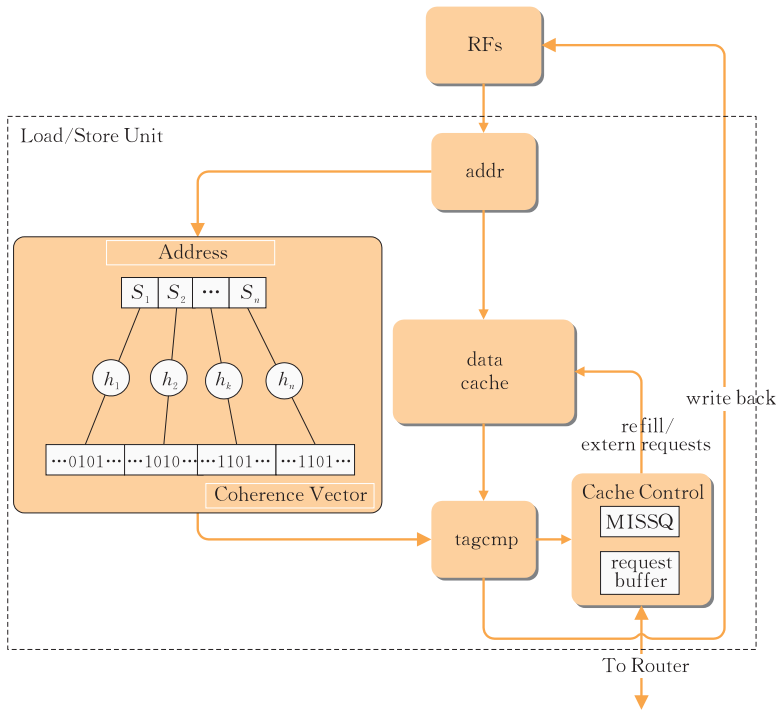


图 3 Coherence Vector 和访存部件的关系

Coherence Vector,访问结果在 tagcmp 流水级进行校验,如果缓存命中且数据正确则写回(write back)到寄存器堆中,否则产生一个访问 L2 缓存的请求并交付缓存控制(cache control)模块进行回填处理.注意 Coherence Vector 和数据缓存在同一流水级,因此对 Coherence Vector 的访问延迟被隐藏在对数据缓存的访问过程中.

表 1 中的 register_addr 指令用于把一个 L1 缓存行(cache line)对齐内存地址注册到 Coherence Vector 中. Coherence Vector 硬件上是一个 Bloom Filter 结构,其硬件实现已有成熟的研究^[18-20]. 基本思想是把地址分成若干位段, S_1, S_2, \dots, S_n , 每个位段通过一个 Hash 函数映射到一个位向量中的相应位置. Bloom Filter 的基本性质是,如果一个内存地址之前注册到了 Coherence Vector 中,那么这个地址就一定存在于 Coherence Vector 所表达的集合中. 因此,只要在共享数据的访问之前注册共享地址,那么所有对共享数据的内存访问都会在 Coherence Vector 命中,并被硬件检测到.

系统启动时 Coherence Vector 被清 0,此后每执行一条 register_addr 指令就将一个内存地址记录到集合中. 当一个取数操作执行时,访问数据缓存同时访问 Coherence Vector,如果在 Coherence Vector 集合中命中,即使数据缓存命中也触发一个 L2 回填请求,即对于共享数据的访问始终放弃本地拷贝,只信任共享 L2 缓存的内容. 当存数操作执行时,如果在 Coherence Vector 集合命中,则直接写回到 L2 缓存.

Coherence Vector 的优点在于将对共享访问的

分析转化为对共享数据的分析,程序员编程相对容易. 其缺点在于共享地址的取数操作无条件放弃本地缓存的数据,对性能不利,存数操作需要穿透到 L2 缓存,这在一定程度上会增加片上网络的压力. 在编写多线程程序时可以把 Coherence Vector 作为 Godson-T 默认编程模型的补充. 对于共享数据访问易于分析的场所,尽可能用 bcr/ecr 指令来提高取数操作的效率,而对于一些行为复杂但对整体性能影响不大的代码,可以用 Coherence Vector 来减低程序分析的难度.

为了在 Godson-T 上支持 Cilk 编程模型,修改 Cilk runtime 的默认实现,在系统初始化时让每个处理器核都把系统中的共享数据地址注册到访存部件中的 Coherence Vector 中,通过硬件支持保证数据访问的一致性. 除了上述关键的修改外,其他针对 Godson-T 的修改包括同步操作、线程创建、线程同步和退出等等系统相关的内容.

4 DAG 存储模型的硬件支持

4.1 动机

DAG Consistency 是根据 Cilk 计算的本质特征抽象出来的存储模型^[16,21]. 其相对于传统存储模型突出特征,是从计算固有的依赖关系出发来定义存储模型,而不是给各个线程的访存序列施加一组约束规则. 对 DAG 存储模型(DAG Consistency)提供硬件支持的目的在于给应用程序员提供一个友好的 Cilk 编程环境,否则程序员仍然需要额外的编程努力来维护数据的一致性,如图 4 所示.

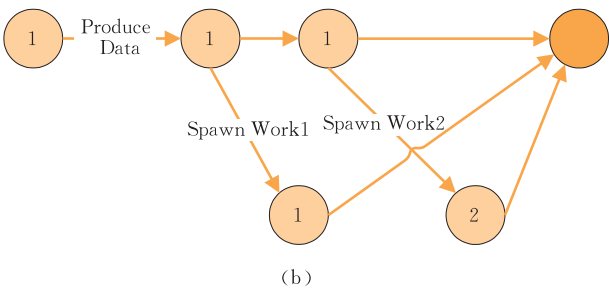
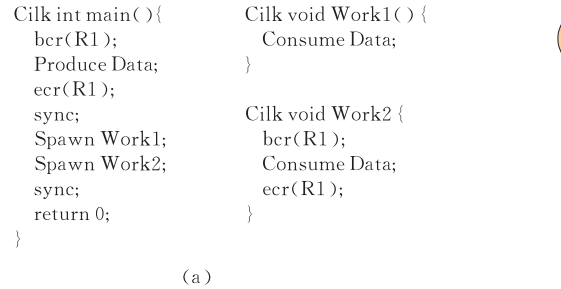


图 4 需要对 DAG Consistency 的硬件支持的原因

图 4(a)是一个简单的 Cilk 程序,假设此时没有对 DAG Consistency 的硬件支持. 主线程首先产生一块数据(Produce Data),然后创建两个可以并行的任务 Work1 和 Work2,两者都需要主线程产生的数据. 图 4(b)是该程序对应的数据依赖关系图. 关于 Cilk 程序以及对应依赖关系图的更多内容请参

考文献[15-16].
现假设有两个处理器来执行此程序,一种可能的任务分配方案如图 4(b)所示,标号为 1 的任务(包括 Produce Data 和 Work1)由处理器 1 执行,标号为 2 的任务(Work2)由处理器 2 执行,没有标号的任务可能被处理器 1 或者处理器 2 执行,取决于

二者的相对执行速度. 根据域一致性编程模型的要求可以得出, 既然 Work2 由处理器 2 执行, 处理器 1 在生成数据(produce Data)以及处理器 2 消费数据(Consume Data)时都需要 bcr/ecr 指令来保证正确的值传播. 注意 Work1 (Consume Data)不需要 bcr/ecr 指令, 是因为根据 Cilk runtime 的调度算法, Work1 和 Produce Data 任务必定由同一处理器执行. 通过对 DAG Consistency 进行结构支持的目的在于进一步消除程序员的这一维护数据一致性的编程负担.

4.2 DAG Consistency 的硬件支持

实现 DAG Consistency 的一种简单方法是 BACKER 算法^[16]. 给定一个 Cilk 程序及相应的 DAG 图, 如果在 DAG 图中存在一条从节点 A 到节点 B 的依赖边(即 B 必须在 A 之后执行), 且两者分别被调度到两个处理器核执行, 则: (1) 在节点 B 被执行前, 处理器核执行 reconcile_flush 指令将其 L1 缓存脏(dirty)的缓存行(cache line)写入 L2 缓存, 并将 L1 缓存清空, 与此同时执行节点 A 的处理器核必须执行 reconcile 指令将其 L1 缓存脏行写入 L2 缓存; (2) 在节点 A 执行完毕, 需执行 reconcile 指令将 L1 缓存中的脏行写入 L2 缓存.

为了实现 BACKER 算法, 首先需要解决的问题是在什么情况下调用表 1 中的缓存操作指令, 这是在 Cilk runtime 中实现的. 当系统初启时, 只有 0 号处理器核有计算任务, 但是随着计算的推进, 其他空闲的处理器核可以随机选择一个忙的处理器核窃取任务执行. 当发生一次任务窃取时, 不妨设处理器核 i 从处理器核 j 窃取任务 W . 在处理器核 i 开始执行 W 之前, 首先执行一条 reconcile_flush 指令把 L1 缓存清空, 这样才能保证执行 W 的过程中不

取到 L1 缓存中的旧数据, 然后还需要执行一条 remote_reconcile 指令通过片上网络通知处理器 j 将其 L1 缓存的所有脏行写透到 L2 缓存, 这样是为了保证处理器 i 在执行 W 的过程中能够取到最新的数据拷贝. 在 Cilk 计算中, 任何一个子线程在结束时都有一条指向父线程的回边, 即所谓 fully strict computation^[15]. 因此, 任何一个处理器核在执行完当前任务(意味着一个子线程的结束), 并在从就绪队列中取后续任务执行之前, 必须执行一条 reconcile 指令, 保证执行对应的父线程的处理器核能取到最新的数据拷贝.

所有对缓存操作指令(reconcile, reconcile_flush, remote_reconcile)的调用都是在 runtime 中进行的, 对应用程序员是不可见的, 而且经过 runtime 的抽象, 硬件的域一致性编程模型对程序员也是不可见的. 因此, 在有了 DAG Consistency 的硬件支持之后, 图 4 中为了维护数据一致而增加的 bcr/ecr 操作就不需要了, runtime 会自动调用缓存操作指令保证执行的正确性.

5 评估环境

本文的试验平台为 Godson-T V3 众核体系结构模拟器, 主要部件的配置参数如表 2 所示. Godson-T 默认情况下提供类似 Pthread 的多线程编程环境, 我们将系统软件环境进行必要修改以支持 Cilk runtime. Godson-T 不支持虚存, 所有线程共享同一个 4GB 的 32 位地址空间, 所有处理器核都是不可抢占的. 试验评估所需要的编译环境是 GCC-3.3.3 交叉编译器(X86-MIPS).

表 2 Godson-T 主要结构参数

部件	说明
处理器核	顺序双发射 8 级流水线, 用户态 MIPS 指令集支持, 一个定点部件, 一个浮点部件, 一个访存部件.
片上网络	8×8 MESH 网络, 静态 X-Y 路由.
Router	4 级流水线, 峰值带宽为 16GB/s, 两个虚通道, 每个虚通道能缓存一个请求.
L1 缓存	每个处理器核有 32KB 的 L1 缓存, 缓存行大小为 32 字节, 访问延迟 1 拍. L1 缓存不支持 outstanding miss.
L2 缓存	16 个按照地址散列的 Cache Bank, 每个 Bank 256KB. L2 缓存行大小为 64 字节, 访问延迟为 4 拍, 每个 L2 Bank 能同时缓存 4 个访存请求.
内存控制器	4 个 DDR2 内存控制器, 峰值带宽为 32GB/s.

本文实验的测试程序及问题规模如表 3 所示, 所有程序均来自 Cilk 软件包(version 5.4.6). 其中 Fib、LU 和 FFT 分别计算 Fibonacci 数列求解、稠密矩阵 LU 分解和快速傅立叶变换; CilkSort 和 Cholesky 分别实现了归并排序和稀疏矩阵的 Cholesky

分解; blockMM 和 strassenMM 则分别用分块和 strassen 算法实现矩阵乘法运算. 我们对 blockMM、FFT 和 CilkSort 分别实验了两种大小的问题规模, 后续实验结果若非特别说明均指默认问题规模下的测试结果. 所有程序均用-O3 优化选项编译.

表 3 测试程序

程序	问题规模
Fib	30
blockMM	1024 (default) and 2048
strassenMM	1024
LU	1024
Cholesky	$-n\ 1000 -z\ 4000$
FFT	1024×1024 and $4\times 1024\times 1024$ (default)
CilkSort	4000000 (default) and 6000000

6 实验结果

6.1 Coherence Vector 硬件代价和误杀率的折中

Coherence Vector 本质上试图通过有限的硬件开销来表达容量可变信息集合,任何一个 Bloom Filter 的硬件结构都存在容量和误杀率之间的权衡问题. 本文将误杀率定义为一个私有内存地址被误判为共享地址的概率. 表 4 列出了启动 64 个线程运行 Fib 程序,在不同配置情况下误杀率和硬件代价之间的关系. 选择 Fib 程序是因为其本身所需要的计算很少,大部分运行时间都在 runtime 中,因此可以通过这个程序很好地了解 runtime 的行为. 需要说明的是,不同配置下测量的误杀率和程序行为有关,不同的程序,甚至同一个程序,启动的线程数目或问题规模不同得到的误杀率数据也不尽相同.

表 4 不同配置情况下 Coherence Vector 的误杀率和硬件代价

配置	误杀率	面积/mm ²	说明
C1	0.546	0.051×64	9/6/6/6
C2	0.448	0.054×64	9/7/6/5
C3	0.414	0.059×64	9/8/5/5
C4	0.257	0.077×64	9/9/5/4
C5	0.232	0.115×64	9/9/9
C6	0.212	0.137×64	10/9/8
C7	0.187	0.203×64	11/9/7
C8	0.143	1.871×64	13/14
C9	0.117	5.208×64	11/16

由于 L1 缓存行大小为 32 字节,且 Coherence Vector 中只需要记录缓存行对齐的地址,因此只需要考察 32 位地址的高 27 位即可. 表 4 中的每一种配置给出了一种位段划分方案,如 9/6/6/6 就表示把 27 位地址划分为 9、6、6、6 4 个位段,用 4 个 Hash 函数映射到对应的位向量集合. 我们用 Verilog 硬件描述语言实现了 Coherence Vector 的主要逻辑,通过 Synopsys Design Compiler (Version Y-2006.06) 综合得到面积数据. 由于每个处理器都有相同的 Coherence Vector 逻辑,表 4 中 x64 表示 Coherence Vector 在芯片中占用的总面积. 从表 4 可以看出一

个基本的趋势,硬件代价越高(面积越大),误杀率越低. 综合表 4 面积和误杀率的结果,可以发现 C1~C4 误杀率较高,而 C5~C9 面积较大,因此我们的后续试验均按照配置 C4 进行.

为了研究误杀率对性能的影响,图 5 给出了对应于表 4 的各种配置相对于配置 C1 的性能. 图中 X 轴表示表 5 中的各种配置,Y 轴表示相对性能. 可以看出,误杀率越高,性能相应地越低,但总体性能偏差不超过 3%,表明在这里误杀率对性能的影响不大. 原因在于:(1)在测量的程序中,访问共享数据的取数操作本身数量有限;(2)只有在该取数操作被误杀且在一级缓存命中的情况下才会引起性能损失. 如果一个取数操作在一级缓存不命中,不管是否误杀都会产生到二级缓存的回填操作,这种情况下不会带来性能损失.

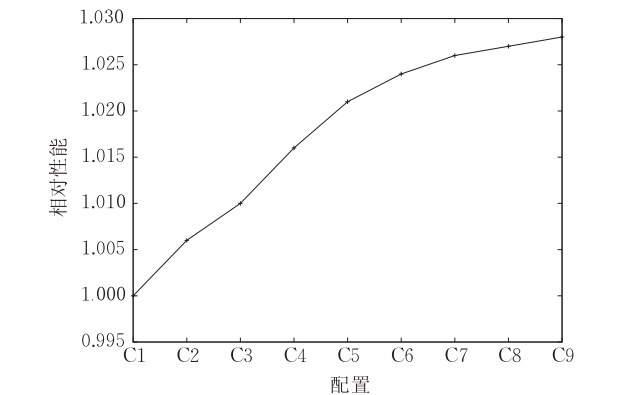


图 5 不同配置情况下的性能比较

6.2 和 SMP 系统的比较

表 5 列出了当启动 8 个处理器核(线程)时所有测试程序得到的性能数据,进行这一分析的主要目的是看能否验证 Frigo 等人在 8 节点 SMP 系统中得到的结论^[14]. 和文献[14]的测量方法不同,我们这里的运行时间以模拟器的时钟周期数为准,而不是标准时钟. $T_1, T_\infty, T_1/T_\infty, T_8, T_s$ 和 T_s/T_8 分别表示总工作量、关键路径上的工作量、并行度(parallel slackness)、启动 8 线程时的性能、串行版本的性能、runtime 开销和相对串行版本的加速比,关于这些参数的精确定义请参考文献[14]. 注意每个 Cilk 程序都有串行语义,这意味着其执行结果和串行版本(C elision)的结果是一样的. 每个程序的串行版本是一个串程序(运行时间为 T_s),和只有一个线程运行的并程序(其运行时间为 T_1)的区别在于,并程序里有 runtime 的额外开销,而串行程序没有,因此 T_1/T_s 能够表示不同程序的 runtime 开销.

表 5 启动 8 个线程时的性能数据

程序	T_1	T_∞	T_1/T_∞	T_8	T_s	T_1/T_s	T_s/T_8
Fib	3226423006	15947	202321	402329207	115229393	28	0.28
blockMM	4349945329	49020	88738	695886742	4306876563	1.02	6.2
strassenMM	3770399345	93518439	40	654405491	3759121979	1.003	5.7
LU	4081721410	28257463	144	606303069	4041308326	1.01	6.7
Cholesky	1758220058	25769290	68	308996501	1490016998	1.18	4.8
FFT	4894784837	803984	6088	705689531	4994678405	0.98	7.1
CilkSort	2067177665	1150438	1797	291373505	1722648054	1.2	5.9

可以看出,当我们的试验的问题规模和文献[14]相等甚至更大(FFT)时,在 Godson-T 上测量到的并行度更高,主要原因是因为相对于 SMP 系统,片上系统的核间通信代价要低得多.表 5 中 Cholesky 和 LU 的加速比和文献[14]相比偏低,是因为模拟时间的限制使得我们没能运行和文献[14]一样大的问题规模.对于大多数程序,我们测量的 runtime 开销和性能加速比和 SMP 系统相近文献[14].Fib 程序的结果是个例外,实验发现无论是 runtime 开销还是加速效果都比文献[14]差很多.我们目前尚不清楚文献[14]结果很好的原因,但 Fib 在 Godson-T 上 runtime 开销很大的主要原因在于每次递归调用时过程状态的分配和释放代价相对于 Fib 本身的计算而言要高很多.

6.3 众核加速比

由表 5 可以看出,blockMM、CilkSort、FFT 和

Fib 4 个程序有较高的并行度.在众核体系结构上,我们更关心这些程序能否获得很好性能加速.在每个处理器核计算能力一定的情况下,加速比越好则说明越能充分发挥众核并行处理的潜能.

在众核结构上,制约性能加速比的结构方便的因素主要来自两个方面.首先是有限的访存带宽,这一方面受制于片上内存控制器有限的内存访问带宽;另一方面来自 L2 Cache Bank 的访问竞争,当多个处理器核访问同一个 Bank 的数据时,则会带来额外的排队延迟.其次是片上网络的带宽无法充分利用,现有的软件系统没有针对片上网络的拓扑结构进行特别优化,这可能导致片上网络有些路径请求过多,而另一些路径空闲的情况.为了研究这些因素对性能的影响,图 6 给出了 blockMM、CilkSort、FFT 和 Fib 4 个程序的性能加速曲线.

图 6(a)~(d)中的 linear 曲线表示理想加速趋

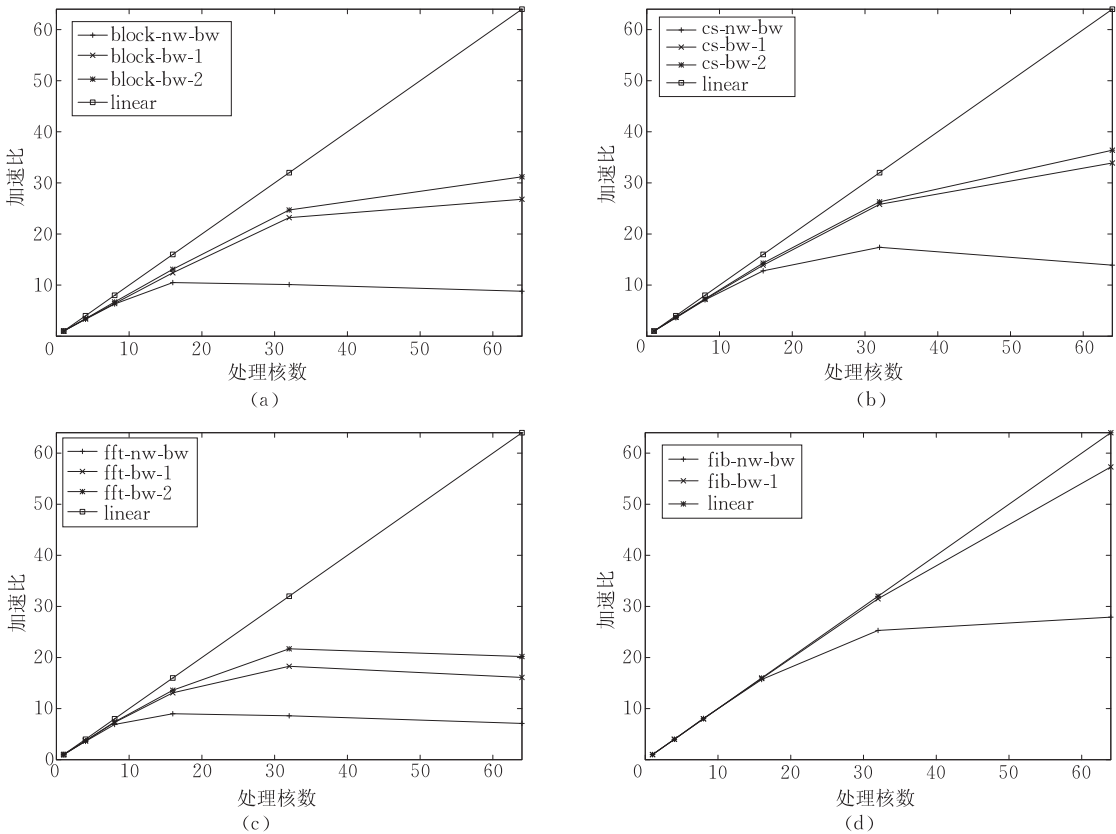


图 6 众核情况下的性能加速比分析

势,其余曲线均为不同模拟器配置和问题规模下测量得到的结果.其中 block-nw+ bw、cs-nw+ bw、fft-nw+ bw 和 fib-nw+ bw 4 条曲线分别表示 blockMM、CilkSort、FFT 和 Fib 4 个程序在片上网络比较保守的配置下测量得到的加速效果.可以看出 blockMM 和 FFT 在处理器核数目超过 16 时就无法得到进一步性能提高,CilkSort 的性能加速趋势能延续到 32 个核,Fib 程序访存压力小,故在 64 核的时候仍有加速,但加速曲线相对理想的线性加速曲线有很大差距.blockMM 虽然访存密集,但是局部性非常好,加速比差的原因在于 Cilk 程序用递归形式编写,编译器能做的优化有限,产生大量对 L2 缓存的中间结果的存取.FFT 加速效果也不理想,原因在于处理器核间需要较多的通信,而这种数据通信都通过共享的 L2 缓存中转.

图 6 中 prog-bw-1 和 prog-bw-2 (prog 在 (a)、(b)、(c) 和 (d) 中分别指 block、cs、fft 和 fib,另外 fib 程序只有 fib-bw-1 曲线)都表示将片上网络带宽配置成无限的情况下测量的性能加速效果,但 prog-bw-2 测量的问题规模比 prog-bw-1 大.我们对 Fib 程序只测量了一个问题规模,因为其加速比已经能接近线性,对 blockMM、CilkSort 和 FFT 分别都测量了两个问题规模,目的是考察网络带宽无限的情况下加速趋势和问题规模的关系.可以看出,所有程序的整体加速趋势有了很大改观,除 FFT 以外,其余 3 个程序都能加速到 64 核,但是即使在片上网络理想的情况下,仍然能看到 L2 的访问竞争和 DRAM 访问带宽对加速比的制约.

6.4 片上带宽的利用率分析

为了进一步认识片上网络的带宽利用情况对性能加速比的影响,图 7 给出了运行不同程序时所有 Router (每个处理器核对应一个 Router,一共 64 个)的负载分布情况.图 7 的所有数据均假设在保守

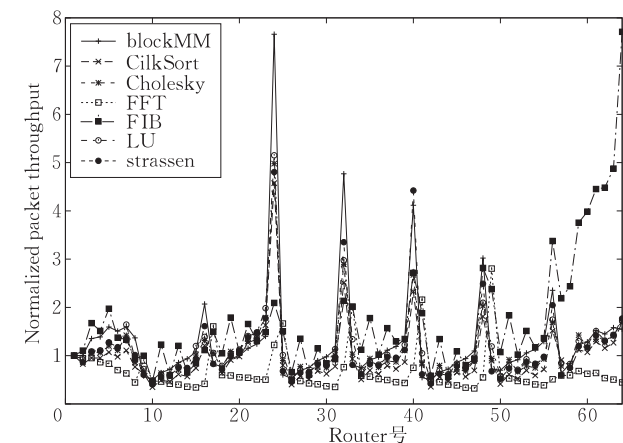


图 7 运行不同程序时片上网络各 Router 的负载分布

的网络配置(如表 3 所示),并且所有处理器核都启动的情况下运行得到.图中 X 轴表示 Router 号(从 1 到 64),Y 轴表示每个 Router 归一化后的消息数目,具体计算方法为:对于一个给定程序,Router R 的归一化消息数目等于 Router R 实际处理的消息数目和 Router 1 实际处理的消息数目的比值.采用归一化消息数目使得我们便于认识不同的程序对片上网络带宽利用的共性规律.

可以看出,所有程序呈现的共同特征是,负载最重的 Router 都分布在 MESH 网络的第 1 列和最后 1 列(即 Router 24,Router 32,Router 40,Router 48 和 Router 56 等等).由于所有处理器核都需要到 L2 缓存取数,且核间通信的数据也通过 L2 缓存中转,一般而言靠近 L2 Bank 的 Router 比网络内部的 Router 负载更重.另外,由于 Godson-T 采取静态 X-Y 路由,所有发往 L2 缓存的请求在回填(refill)的时候都优先进行 X 选路(垂直方向),因此在所有靠近 L2 Bank 的 Router 中,第 1 列和第 8 列的 Router 比第 1 行和第 8 行的 Router 总体负载要重很多.总体来说,由于 L2 Cache Bank 的分布特征以及 X-Y 静态选路,使得所有程序都无法均衡地利用片上网络的带宽.

7 相关工作

Cilk 是一个多线程的编程语言,尤其适合表达一些非规则的并行性.关于 Cilk 语言的实现^[14]、任务调度^[15]以及存储模型^[16, 21]等方面过去已有大量的研究.早期 Cilk 版本不能很好地表达循环级并行性,后来 Yi 等人提出一种编译优化方法^[22]能够自动把循环代码转换成递归代码以充分挖掘其缓存无关(cache oblivious)的特性.近年来,Cilk 的表达能力得到进一步改进,并且增加了对循环级并行的支持.

本文的 Coherence Vector 机制很大程度上受到以数据为中心进行同步^[23-24]的启发.Vaziri 等人基于 Java 语言率先提出对共享数据进行必要约束,然后由编译器自动保证线程之间的正确同步.Ceze 等人则在此基础上提出了对数据为中心进行同步的体系结构支持.本文的工作和前人研究的共同点在于以数据为中心,区别在于本文的方法用来解决数据一致性问题而非同步问题,另外在实现方法上本文基于 Bloom Filter 来设计 Coherence Vector,不需要担心硬件容量不够而溢出的问题.

目前还没有一个公认的编程模型能很好解决众

核体系结构的编程问题,但已有一些研究工作对这个问题进行了有益的尝试. Cascaval 等人在 IBM 的 Cyclops 众核体系结构平台上评估了 OpenMP 编程模型^[25-26]. Zea 等人为众核体系结构平台提出了一种叫 Servo 的基于组件间消息传递的编程模型^[27]. Olivier 等人扩展了组件编程的思想并提出了 Capsule 系统来解决多核平台上的编程问题,其主要优点是能根据硬件资源自动调整并行粒度^[28]. Hwu 等人则认为众核结构上需要隐式编程模型,即应该更多依赖编译器或硬件来自动挖掘更多的并行性^[29],这一点和 Cilk 编程模型有相通之处.

8 结 论

如何对众核处理器编程是一个还没有很好解决的问题. 传统的基于目录或总线的高速缓存一致性协议扩展性有限,而采用弱一致性模型又会使众核结构上的多线程编程变得更加困难. 本文基于域一致性模型,提出 Coherence Vector 机制改善了 Godson-T 的编程性,并顺利支持了 Cilk runtime. 本文还提出了对 DAG Consistency 的硬件支持,最终实现了对 Cilk 语言的完全支持.

实验结果表明,当启动的线程数目较少(<16)时所有测试程序都能获得比较好的性能加速. 但是在众核情况下性能加速却不理想,由于所有处理器核需要的数据都来源于 L2 Cache,核间通信的数据也通过 L2 中转,这使得片上网络和有限的访存带宽成为性能瓶颈. 实验还发现受静态 X-Y 选路的影响,所有程序都无法均衡地利用片上带宽. 我们下一步工作将进一步研究解决这些问题的方法.

致 谢 本文所立足的 Godson-T V3 模拟器由黄河、林伟、周永彬、袁楠、宋凤龙、余磊、焦帅和刘玉萍等同事搭建,在此表示感谢. 另外,特别感谢评审老师的深刻评价和建议!

参 考 文 献

- [1] Cascaval C, Castaños J G, Ceze L, Denneau M, Gupta M, Lieber D, Moreira J E, Strauss K, Warren Jr H S. Evaluation of a multithreaded architecture for cellular computing//Proceedings of the International Symposium on High Performance Computer Architecture. Boston, USA, 2002: 311-322
- [2] Almási G, Cascaval C, Castaños J G, Denneau M, Lieber D, Moreira J E, Warren Jr H S. Dissecting cyclops: A detailed analysis of a multithreaded architecture. SIGARCH Computer Architecture News, 2003, 31(1): 26-38
- [3] Vangal S, Howard J, Ruhl G, Dighe S, Wilson H, Tschanz J, Finan D, Iyer P, Singh A, Jacob T, Jain S, Venkataraman S, Hoskote Y, Borkar N. An 80-Tile 1.28TFLOPS network-on-chip in 65nm CMOS//Proceedings of the IEEE International Solid-State Circuits Conference. San Francisco, California, USA, 2007: 98-589
- [4] Asanovic K, Bodik R, Catanzaro B C, Gebis J J, Husbands P, Keutzer K, Patterson D A, Plishker W L, Shalf J, Williams S W, Yelick K A. The landscape of parallel computing research: A view from berkeley. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html>
- [5] Zhu W R, Sreedhar V C, Hu Z, Gao G R. Synchronization state buffer: Supporting efficient fine-grain synchronization on many-core architectures//Proceedings of the International Symposium on Computer Architecture. San Diego, CA, USA, 2007: 35-45
- [6] Wentzlaff D, Griffin P, Hoffmann H, Bao L, Edwards B, Ramey C, Mattina M, Miao C C, Brown J F, Agarwal A. On-chip interconnection architecture of the Tile processor. IEEE Micro, 2007, 27(5): 15-31
- [7] Tan G, Fan D, Zhang J, Russo A, Gao G R. Experience on optimizing irregular computation for memory hierarchy in manycore architecture//Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. Salt Lake City, Utah, USA, 2008: 279-280
- [8] Long G P, Fan D R, Zhang J C, Song F L, Yuan N, Lin W. A performance model of dense matrix operations on many-core architectures//Proceedings of the European Conference on Parallel and Distributed Computing. 2008: 120-129
- [9] Lamport L. How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Transactions on Computers, 1979, 28(9): 690-691
- [10] Adve S V, Gharachorloo K. Shared memory consistency models: A tutorial. IEEE Computer, 1996, 29(12): 66-76
- [11] Lenoski D, Laudon J, Gharachorloo K, Gupta A, Hennessy J L. The directory-based cache coherence protocol for the DASH multiprocessor//Proceedings of the International Symposium on Computer Architecture. Seattle, WA, USA, 1990: 148-159
- [12] Ifteodi L, Singh J P, Li K. Scope consistency: A bridge between release consistency and entry consistency. Theory Computing Systems, 1998, 31(4): 451-473
- [13] Hu Wei-Wu. Shared Memory Architectures. Beijing: High Education Press, 2001(in Chinese)
(胡伟武. 共享存储体系结构. 北京: 高等教育出版社, 2001)
- [14] Frigo M, Leiserson C E, Randall K H. The implementation of the Cilk-5 multithreaded language//Proceedings of the International Symposium on Programming Languages Design and Implementation. Montreal, Canada, 1998: 212-223

- [15] Blumofe R D, Leiserson C E. Scheduling multithreaded computations by work stealing//Proceedings of the Annual IEEE Symposium on Foundations of Computer Science. Santa Fe, New Mexico, 1994: 256-368
- [16] Blumofe R D, Frigo M, Joerg C F, Leiserson C E, Randall K H. Dag-consistent distributed shared memory//Proceedings of the International Parallel Processing Symposium. Honolulu, Hawaii, 1996: 132-141
- [17] Bloom B. Space/Time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970
- [18] Ceze L, Tuck J, Torrellas J, Cascaval C. Bulk disambiguation of speculative threads in multiprocessors//Proceedings of the International Symposium on Computer Architecture. Boston, USA, 2006: 227-238
- [19] Moore K E, Bobba J, Moravan M J, Hill M D, Wood D A. LogTM: Log-based transactional memory//Proceedings of International Symposium on High Performance Computer Architecture. Austin, Texas, USA, 2006: 254-265
- [20] Sanchez D, Yen L, Hill M D, Sankaralingam K. Implementing signatures for transactional memory//Proceedings of the International Symposium on Microarchitecture. 2007: 123-133
- [21] Frigo M, Luchangco V. Computation-centric memory models//Proceedings of the International Symposium on Parallel Algorithms and Architectures. Puerto Vallarta, Mexico, 1998: 240-249
- [22] Yi Q, Adve V S, Kennedy K. Transforming loops to recursion for multi-level memory hierarchies//Proceedings of the International Symposium on Programming Languages Design and Implementation. Vancouver, Canada, 2000: 169-181
- [23] Ceze L, Montesinos P, Praun C V, Torrellas J. Colorama: Architectural support for data-centric synchronization//Proceedings of the International Symposium on High Performance Computer Architecture. Phoenix, Arizona, USA, 2007: 133-144
- [24] Vaziri M, Tip F, Dolby J. Associating synchronization constraints with data in an object-oriented language//Proceedings of the International Symposium on Principles of Programming Languages. Charleston, South Carolina, 2006: 334-345
- [25] Ródenas D, Martorell X, Ayguadé E, Labarta J, Almási G, Cascaval C, Castaños J G, Moreira J E. Optimizing NANOS OpenMP for the IBM Cyclops multithreaded architecture//Proceedings of the International Parallel and Distributed Processing Symposium. Denver, Colorado, USA, 2005: 110-118
- [26] Almasi G S, Ayguadé E, Cascaval C, Castaños J G, Labarta J, Martínez F, Martorell X, Moreira J E. Evaluation of OpenMP for the Cyclops multithreaded architecture//Proceedings of the International Workshop on OpenMP Applications and Tools. Toronto, Canada, 2003: 69-83
- [27] Zea N, Sartori J, Kumar R. Servo: A programming model for many-core computing. ACM SIGARCH Computer Architecture News, 2007, 36(2): 28-37
- [28] Palatin P, Lhuillier Y, Temam O. CAPSULE: Hardware-assisted parallel execution of component-based programs//Proceedings of the International Symposium on Microarchitecture. 2006: 247-258
- [29] Hwu W M, Ryoo S, Ueng S Z, Kelm J H, Gelado I, Stone S S, Kidd R E, Bagsorkhi S S, Mahesri A, Tsao S C, Navarro N, Lumetta S S, Frank M I, Patel S J. Implicitly parallel programming models for thousand-core microprocessors//Proceedings of the Design Automation Conference. San Diego, CA, USA, 2007: 754-759



LONG Guo-Ping, born in 1982, Ph.D. candidate. His research interests include computer architecture, parallel programming, performance modeling and evaluation.

ZHANG Jun-Chao, born in 1976, Ph.D.. His research interests include computer architecture, compiler design, runtime systems.

FAN Dong-Rui, born in 1979, Ph.D.. His research interests include computer architecture, low power design.