

众核处理器中使用写掩码实现混合写回/写穿透策略

林 伟^{1),2)} 叶笑春^{1),2)} 宋风龙^{1),2)} 张 浩¹⁾

¹⁾(中国科学院计算技术研究所计算机体系结构重点实验室 北京 100190)

²⁾(中国科学院研究生院 北京 100049)

摘 要 高速缓存采用写回策略,能极大地节省对片上网络和访存带宽的消耗,这对于片上众核(大于 16 核)的结构尤为重要.与通常多核系统中基于目录/总线的写无效或写更新协议不同,文中给出了片上实现域一致性存储模型和基于硬件锁的缓存一致性协议的方案并提出了在 L1 高速缓存保存写掩码的方法,用以记录本地更新缓存块的字节位置,解决了写回策略下伪共享带来的缓存一致性问题.文中还进一步提出两种优化掩码存储空间开销的新方法:通过设定程序中较少出现的、长度为 1~3 字节的写指令为写穿透,在 L1 中每 4 字节设置一位写掩码,将写掩码的芯片面积开销压缩到字节粒度的 27.9%;设计项数为 L1 缓存块总数 12.5%的多路写掩码缓存,在不损失性能的情况下,将面积开销压缩到字节粒度的 17.7%.搭建的众核平台 Godson-T 采用域一致性存储模型,使用写掩码实现混合写回/写穿透缓存策略(临界区内写穿透,临界区外写回).实验使用 splash2 的 3 个程序和 2 个生物计算程序进行评估.结果表明,相对于完全写穿透,混合写回策略在 32 和 64 线程的配置下普遍获得 24%以上的性能提升,性能略优于完全写回,并且采用两种优化空间开销的新方法后性能无损失.

关键词 众核;写掩码;写掩码缓存;域一致性;伪共享;写无效;写更新

中图法分类号 TP302 **DOI 号**: 10.3724/SP.J.1016.2008.01918

Using Write Mask to Support Hybrid Write-Back and Write-Through Cache Policy on Many-Core Architectures

LIN Wei^{1),2)} YE Xiao-Chun^{1),2)} SONG Feng-Long^{1),2)} ZHANG Hao¹⁾

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 Write-back cache policy can greatly save bandwidth consumption for write operations. It's particularly beneficial in many-core architecture. Normally CMP uses write-invalid or write-update cache protocol like directory based MESI which is hardly scalable and too complex. Alternatively the authors implemented scope consistency (and lock-based cache coherence protocol) on chip, add write-mask for each cacheline of L1 Dcache to record the written byte's location and solve the false sharing problem. To further optimize the write-mask storage overhead, two methods are proposed. First the authors set store instructions with 1/2/3 bytes write-through property and let every 4-byte data has 1 bit write-mask. This method can compress the chip area of write-mask to 27.9% of origin byte-grain design. Secondly they design write mask buffer whose entry counts 12.5% of total number of Dcache blocks and compress the area overhead to 17.7% of origin without performance lost. On Godson-T 64-core platform which uses scope consistency, they use write-mask to implement hybrid WB/WT cache policy (in the scope range with possible data race we implement write-through, but out of the scope range without data race they choose

write-back). Three splash2 programs and two biological programs are evaluate. The results show that performance improvement is above 24% compared to completely write-through and no performance lost under the two storage optimizations.

Keywords many-core; write mask; write mask buffer; scope consistency; false sharing; write-invalid; write-update

1 引言

随着摩尔定律进一步延续,片上资源的大规模增长,单核处理器遇到功耗/性能瓶颈,体系结构迎来了片上大规模并行(大于 16 核)的时代^[1-2]. 中国科学院计算技术研究所的 Godson-T 是面向计算密集型应用的高性能众核处理器^[1],其注重计算能力和可编程性的平衡,把更多的片上资源用于计算部件,在这样的设计理念下,采用顺序双发射的精简处理器核,抛弃了基于目录的 Cache 一致性协议^[3],并选择弱一致性^[4]中基于锁的域一致性^[4-5]作为片上的存储模型,在可编程性和计算部件占芯片的比例上取得了较好的平衡.

本文要解决的问题是在采用共享存储的编程模型、且硬件不采用基于目录的写无效/写更新协议下,如何克服伪共享(false sharing)产生的一致性问题,高效实现写回的缓存策略,最终解决写穿透造成的带宽额外消耗的问题.

首先,写穿透的缓存策略使多层的存储结构简化为平坦的一层,使得每个写操作的新值都能及时传播到其它处理器核,回避了空间上一致性的问题^[3]. 其缺点是这些写操作都必须经过片上网络和存储结构底层,因此需要消耗大量的网络和访存的带宽以及功耗. 随着程序的进一步并行化,使执行时间线性缩短,从而单位时间内出现更密集的访存和同步操作,加重了网络和访存的负担,使平均访存延迟大幅增长. 而写回策略由于能够吸收大部分高速缓存命中的写操作,使之不需要立即占用片上网络,在 L1 高速缓存中写合并后延迟替换回共享的 L2 高速缓存,因此众核结构下实现写回策略尤为重要.

但实现写回策略将随之产生 3 个问题,即值延迟传播;处理器之间写操作如何排序;缓存块的伪共享. 传统 CMP 采用写无效或写更新的 Cache 一致性协议解决新值的传播问题,例如基于总线或者目录的 MESI 协议. 目录的缺点存储空间开销,由于其大小将随处理器的数目线性增长,所以扩展性较差. 我们假设 L2 共有 M 个储存块,那么在 P 核的处理

器中,目录的空间消耗为 $P \times M$ ^[3]. 虽然业界提出了很多压缩目录的方法(压缩 P 和 M),但除了空间开销的问题外,实现协议本身过于复杂,其正确性不容易验证.

不采用目录,则无法实现写无效/更新协议. 这种情况下实现写回的缓存策略在碰到伪共享将产生的一致性问题:如图 1 所示,当多个处理器核修改了同一地址所在缓存块的不同部分,它们之间不存在数据竞争. 但是随后从 L1 缓存依次替换回 L2 缓存时,L2 将面对多个版本的 L1 缓存块数据,保留任何一项,都会造成另一部分的数据丢失.

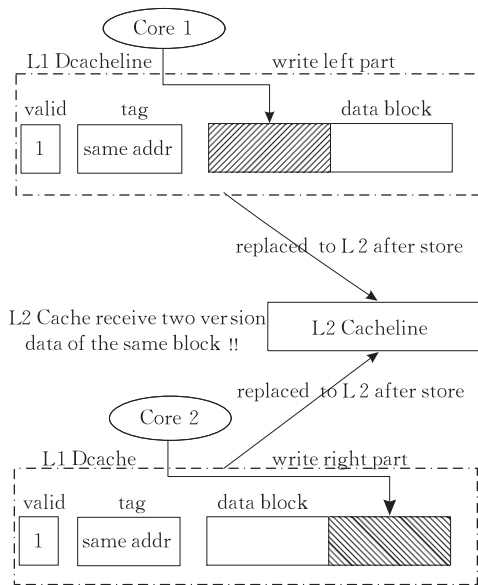


图 1 伪共享带来的一致性问题

经典使用目录的系统是 DASH,它使用写无效的协议,解决了图 1 所示伪共享问题,但是对于伪共享造成“乒乓”效应并没有很好的解决方法^[6-7]. 该问题还可以通过合理的分配地址空间,使各处理器的私有数据地址不属于同一个缓存块来解决,但这将给程序员增加负担,例如,(1) malloc 函数分配堆空间时必须添加 `_align(sizeof(cacheline))`^[6]; (2) 定义结构体 `struct S{int a; int b; int c; int d;}`,其中变量 a, b, c, d 地址连续并属于同一个缓存块长度内. 需要避免不同的处理器分别修改 $S.a, S.b, S.c, S.d$; (3) 常见的分块矩阵运算,不容易确保各块的地址

不属于同一缓存块。

Godson-T 不使用目录,在片上实现域一致性存储模型,利用锁限制多个处理器对共享变量访问的顺序.我们对临界区内采取写穿透策略,解决了共享数据值传播和对同一个地址写操作满足全局序的内存一致性(memory consistency)要求.但是由于不能实现写无效或写更新,因此最初无法解决伪共享问题,临界区外也只能设置为写穿透,影响了性能.

本文有以下两点贡献:

(1)提出给每个 L1 缓存硬件记录写掩码 WM (Write Mask)的方法,解决域一致性模型下写回策略伪共享带来的一致性问题的,实现混合写穿透/写回策略,提高了 Godson-T 整体性能.

(2)提出了两种压缩写掩码存储空间开销的新方法:①通过设定存数长度为 1~3 字节的写指令为写穿透 WT(Write Through),在 L1 缓存块中每 4 字节分配一位写掩码,将写掩码的芯片面积开销压缩到字节粒度的 27.9%;②设计了 4 路的写掩码缓存 WMB(Write Mask Buffer),进一步将面积开销压缩为原来的 17.7%.并且这两种优化方法不会造成性能损失.

本文第 2 节分析相关工作;第 3 节阐述 Godson-T 片上域一致性存储模型;第 4 节提出在 L1 缓存块中记录写掩码实现 WB/WT 混合策略的 2 种设计方案,即 WM32(每个缓存块 32bit WM)和 WM8(每个缓存块 8bit WM);第 5 节阐述写掩码缓存 WMB 的详细设计;第 6 节给出实验结果并进行分析;最后是总结和未来工作.

2 相关工作

2.1 存储模型的弱化

Lamport 提出 SC(顺序一致性)模型^[8],他把各处理器的访存操作交叉,构成一个串行的序列,这个序列要求满足 2 点:(1)单处理器内的程序序;(2)原子性,即发出下一条操作前必须保证之前的访存操作全局可见.虽然 SC 性能低效,但它已经成为评价程序正确性的标准.

Sarita 和 Kourosh 在 1990 年分别提出 DRF0^[9]和弱一致性模型^[10]:把访存相关的指令区分为普通 load/store 指令和同步指令,同步指令再细化为 acquire/release 操作.只对同步指令维持 SC,而对占大部分的普通访存并不限制处理器之间的相互顺序,但需要维持处理器同步指令之间的相互顺序,即

acq 和 rel 带有 memory fence 的语义,以保证同步指令之前的访存结果全局可见.他们还指出弱一致性环境下写无数据竞争的程序可以满足 SC.对 acq/rel 操作 fence 语义作用范围的进一步弱化,产生域一致性模型^[5],它并不要求 acq/rel 之外的访存操作全局可见,而只需要维持 acq/rel 之间的访存操作在 rel 时全局可见. Godson-T 遵循该规范,与软件 DSM 下域一致性不同的是,Godson-T 是基于硬件锁实现.

2.2 众核结构存储管理

IBM Cyclops 64 采用全局和私有的 SPM (Scratch Pad Memory)设计,由程序员控制 SPM 空间,访存操作通过 crossbar,延迟为固定的相同拍数,直接实现 SC^[11]. IBM cell 为异构多核处理器,子核 SPE 只能访问各自独立的 256KB 局部存储空间(LS),通过 DMA 同全局地址空间进行数据交互^[12]. Tile64 延续 MIT Raw^[13]的设计,提供 5 套专用物理网络(不同于虚通道),其中有专门用于 Cache 同内存控制器交互的 MTU,和不同处理器 Cache 之间交互的 TDN,Tile64 提供 neighborhood caching 机制,即分布式共享 Cache,对于非只读的数据只能保存在一个 home 节点中,其它的节点通过 TDN 访问数据,数据保护通过 TLB 进行.由于多道网络可能产生乱序,增加 memory fence 的同步指令^[14].

2.3 写掩码机制

Karp 和 Sarkar Vivek 提出软件方法记录位掩码(bit mask)以实现数据合并机制(Data Merging),用以解决伪共享问题^[6].本文与之不同之处在于:(1)本文通过 store 指令的起始地址和长度直接设置写掩码并保存于私有的 L1 中;而该文的 bit mask 记录共享的存储(例如二级高速缓存(L2)存储)块中更新过的字节位置,并需要低效的数据对比确定 bit mask,例如从一级高速缓存(L1)替换出的脏数据块需要同 L2 中的数据逐字节比对,如果不同,才设置 bit mask;(2)本文采用简洁的硬件保存 WM,而该文使用软件的记录方法,它对需要记录位掩码的存储块分配内存空间,由存储块中硬件保存的指针指向该空间.

2.4 硬件锁机制

QOLB(Queue of Lock Bit)是一个基于硬件等待队列的非阻塞锁^[15]. QOLB 中申请锁的处理器核会轮询本地 L1 中的一个特殊的缓存块(shadow cacheline).当硬件锁被释放,会对下一个候选者发回一个确认,将这个缓存块重置,以结束候选者的本

地自旋. Godson-T 不同于 QOLB, 我们采用基于等待队列的阻塞式锁, 即对于申请不到锁的处理器核, 将阻塞流水线.

3 Godson-T 片上域一致性存储模型

Godson-T 的结构如图 2 所示. 64 个顺序双发射的微核(mini-core)组织为 2 维的 MESH 结构; L2

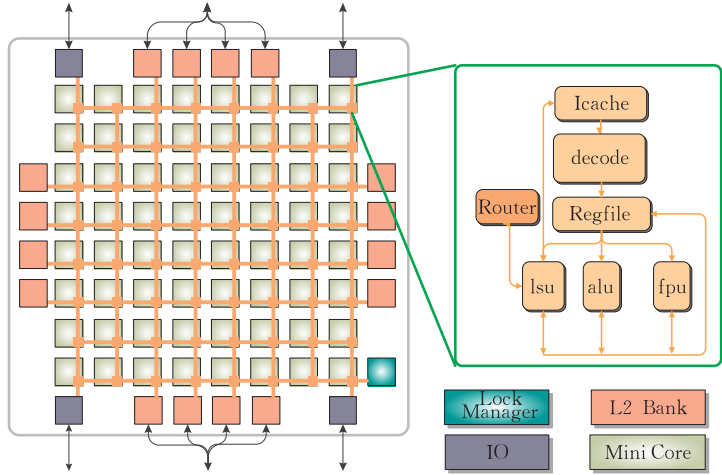


图 2 Godson-T 结构示意图

3.1 硬件锁管理器

域一致性是基于锁的弱一致性, 在软件分布共享存储实现中采用原子指令实现的锁机制, 而在 Godson-T 中采用基于等待队列阻塞方式的硬件锁机制. 我们增加 lock(lock_id) 和 unlock(lock_id) 2 条指令用于实现锁, 使用的锁标记 lock_id 区别于普通访存的地址.

微核执行 lock 指令, 将通过片上网络向硬件锁管理器申请特定的锁. 如果该锁已经被其它微核占用并未释放, 该申请将被保存在一个先进先出的硬件等待队列中. 锁释放后将给下一个请求者发送回一个 ACK, 恢复其被阻塞的流水线. 顺序发射的微核, 能保证 lock 之后的任何指令无法超前执行. 根据域一致性存储模型的规范, lock 不需要具备 memory fence 的语义, 即不保证 lock 之前的 store 指令全局可见.

执行 unlock 指令也发送请求到锁管理器释放相应的锁. 与 lock 指令不同, unlock 需要具备 memory fence 的语义, 以保证 unlock 之前的 store 写穿透到 L2. 由于 L2 多体并分布在芯片的四周, 最复杂的情况是 unlock 之前有 16 个 store 分别前往不同位置的 L2 体, 为保证这些 store 都能被 unlock 执行完之前写回各 L2 体, 我们使用以下的策略:

分为 16 个 Bank 并按地址低位散列在芯片四周, 每侧的 4 个 L2 Bank 共享一个内存控制器; 右下方的 router 连接一个支持锁嵌套的硬件锁(同步)管理器(lock manager), 支持 lock、unlock、barrier、fence 在片上的高效实现; 共 4 个 IO 控制器同片外交互. 本文给出的片上域一致性存储模型, 为保证程序的正确性, 要求程序员编写 data race free(DRF0)的程序.

(1) mesh 网络采用静态 xy 选路策略, 保证发往同一目的地的各个请求按先后顺序到达.

(2) unlock 指令将产生最多 16 个网络包, 先分别到达各个 L2 体所在的 router 后, 再折向目的地锁管理器.

(3) 等待所有的请求包汇集到锁管理器后, 才真正释放锁.

(4) 每个微核配置一个长度为 16bit 的位向量, 对应 16 个 L2 体, 如果 unlock 之前有 store 地址发往该体, 则对应位设置为 1. 当执行 unlock 指令时, 若该向量中某位为 0, 表明无 store 前往该 L2 分体, 则不必发送途经该 L2 体的请求包.

通过以上 4 点, 我们保证了 release 的 fence 语义.

3.2 域一致性模型下基于锁的 Cache 一致性实现

我们要求程序员将有数据竞争的变量或者代码段使用 lock/unlock 或 barrier 这些同步指令显式标识出范围, 产生无数据竞争的程序(DRF0)^[9]. 以下我们称 lock/unlock 之间标出的程序段为临界区. 对于临界区外, 由于无共享数据, 所以不需要维护 Cache 一致性. 对于临界区内的 load 指令, 如果首次访问命中, 则不信任相应的 L1 缓存块而将其设置为无效, 必须到共享的 L2 取新的数据, 这是因为临

界区内大部分为共享数据,它们可能已经被其它线程改写.但回填后,该临界区此缓存块再次被访问命中,是可被信任使用的.对于临界区内的 store 指令,必须保证在退出临界区前写到 L2,如果 L1 命中,则同时修改 L1;如果 L1 miss 则采用写缺失不分配的写策略,不需要回填 L1,直接写穿透到 L2.对于临界区外的 store,在本文提出写掩码机制之前,也采用写穿透 WT 策略.由此 L2 相当于传递共享数据的场所,解决了值传播问题.

4 混合写回/写穿透高速缓存策略

一般高效的并行程序,临界区粒度都很小,因此设置域一致性模型下临界区内 store 写穿透,对性能影响较小;对于临界区外的非竞争数据,则应当采用 WB,以节省带宽消耗.以下 2 小节阐述 WM32 和 WM8 的设计,克服了临界区外写回策略产生的伪共享问题,从而实现 WB/WT 混合策略.

4.1 字节粒度写掩码 WM32

Godson-T 中 L1 缓存块的长度为 32Byte.我们对每个缓存块添加 32bit 的 WM,以记录本地 L1 更新过的缓存块的字节位置.当 L1 缓存块被替换并写回到 L2 时,L2 会根据相应的 WM,只真正写回其中为 1 对应字节的数据,从而解决图 1 伪共享产生的一致性问题的.L1 的结构如图 3 所示.

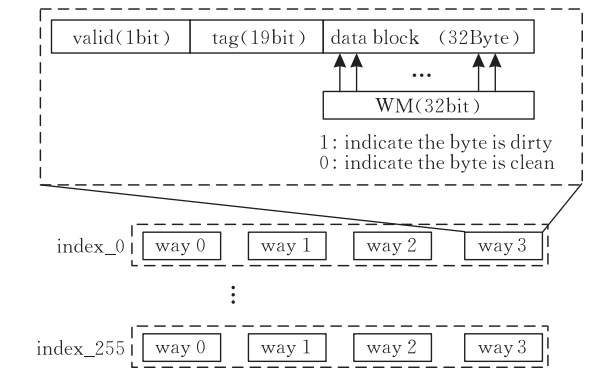


图 3 WM32 L1 Cache

由于实现写回策略,我们增加 L1 替换(replace)写回 L2 的操作.如果被替换出 L1 的缓存块 WM 不为 0,表明该块被写过(dirty),需要将其写回 L2.进入临界区后首次 load 操作,若 L1 命中则设置该缓存块无效,如果该缓存块的 WM 不为 0,设置无效前也需要写回 L2.当 L2 接受到 L1 的写回请求后,L2 只修改 WM 为 1 对应位置的字节.

对于临界区内的 store 保留 2 个属性:(1)写穿

透属性,写穿透 L2 同时如果 L1 命中则必须修改本地的 L1 缓存块,但不修改对应的 WM;(2)写缓存缺失不分配,不需要从 L2 回填,直接写穿透到 L2.对于临界区外的 store 如果 L1 缺失,则需要回填并分配缓存块,回填后根据起始地址和存数的长度设置对应的 WM.

4.2 字粒度写掩码 WM8

WM32 虽然能够解决伪共享问题,但是缺点是硬件开销相对较大,WM 存储空间达到 L1 容量的 12.5%.Godson-T 片上网络的数据宽度为 128bit,因此每次写回 L2 需要增加传送 16bit 的 WM.其中增加的比例(12.5%)并不依赖网络的宽度.

通过分析程序,我们发现实际只写 1~3 字节 store 是极少出现的(表 2).这是因为现代的处理器的 64 或 32 位系统,程序员也较少定义长为 8bit/16bit 的变量.Godson-T ISA 为 MIPS,其中有 store 1 字节和 2 字节的 SB 和 SH;1~3 字节的 SWR/SWL 非对齐指令;以及 4 字节和 8 字节的 SW 和 SDC1.基于以上发现,我们设定 SB/SH/SWR/SWL 指令写穿透,从而不需要对每个字节设置 1 位 WM,而只需每隔 4 字节设置 1 位 WM.如图 4 所示.

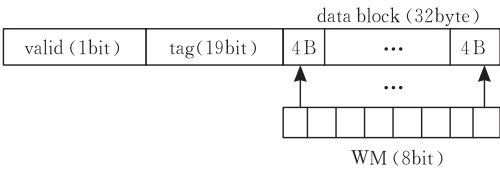


图 4 WM8 L1 缓存块结构

对于 SB/SH/SWR/SWL 指令,无论位于临界区外或者临界区内,均采用写缺失不分配并且写穿透的 Cache 策略,具体为:若 L1 命中,只改写缓存块的数据但不改动 WM,同时写穿透到 L2;若 L1 缺失,直接写穿透到 L2,不对 L1 和 WM 进行任何操作.临界区外的 SDC1 指令由于写 8 个字节的数据,所以需要相应设置 2 位的 WM.当 L1 的某个缓存块写回 L2 时,也会根据相应的 WM 决定数据是否需要写回 L2,但不同于 WM32,此时 WM 的每 1bit 决定了对应 4 字节的数据的去留.通过这样的方法,我们将存储 WM 的硬件开销减少到 WM32 的 25%,32KB 的 L1 只需要 1KB 的空间存储 WM.并且网络每次传送 16Byte 的数据,只需要传送 4bit 的 WM.

临界区内的 load 操作首次访问 L1 命中,根据基于锁的一致性,将设置 L1 缓存块无效,并将该块

WM 为 1 的部分写回 L2. 进一步优化的方法可以是,对于首次 L1 命中的 load 操作,如果地址范围对应的 WM 全部为 1,则不需要设置该缓存块无效并写回 L2. 这是因为这样的数据是本地微核在临界区外写过的,在其它 L1 中肯定不存在对应地址范围 WM 也为 1 的缓存块,否则为 data race 程序,不满足域一致性的要求.

5 写掩码缓存 WMB

除了上节描述的 2 种支持混合 WB 的方法外,我们再提出一种 WMB 的新方法. WMB 是位于 L1 中一个小的缓存,用于缓存 WM. 它只有一级结构. WMB 项数远小于 L1 缓存块的数目,从而进一步节

省了 WM 的存储空间开销.

5.1 WMB 的结构

如表 2 所示,store 通常为 load 数量的 1/8 到 1/2,数量较少. 通过程序运行期间采样,我们还发现脏缓存块占全部 L1 的比例通常不超过 12.5%. 由此设计出项数仅为 L1 缓存块数 1/8 的 WMB. 在 Godson-T 的 32KB L1 配置下,每个缓存块大小 32Byte,共 $(4\text{-way} \times 256)$ 个缓存块,所以 WMB 的总项数设定为 128. 考虑到 4 次 $128 \times 8\text{bit}$ CAM (Content Address Memor) 查找逻辑耗费的芯片面积过大(约为 14 万平方微米),所以 WMB 组织成 4 路,每路 32-entry 只保存该路 L1 缓存块的 WM,如图 5 所示.

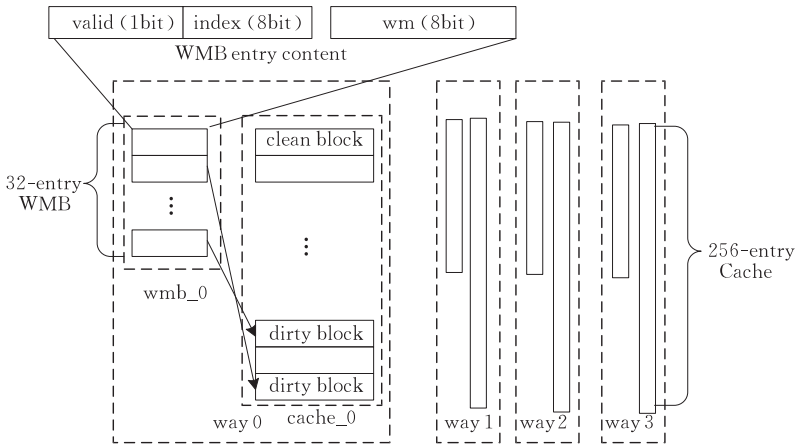


图 5 4 路 WMB 结构

5.2 WMB 单项结构

WMB 每项的结构如上图 5 所示,包括 3 个部分:

- (1) 有效位;
- (2) 8bit index, 指向该路中脏缓存块的位置, index 对应地址的 5~12 位;
- (3) WM 为 4 字节粒度的写掩码, 8bit.

5.3 配置 WMB 的访存流水线

图 6 的虚线划分出 4 级的访存流水线,其中第 1 级 memaddr 用于计算访存地址; WMB 与 L1 Dcache 同属于第 2 级流水,每个访存操作在读取 L1 的同时,分别做 4-way WMB 的查找,逐项比较访存地址的[12:5]位与保存在 WMB 中的 index,得到所访问 L1 的 4 路缓存块对应的 4 个 WM. 若某路 WM 缺失,说明该路的缓存块 clean 或者无效. 32-entry CAM 查找的延迟小于 0.6ns,可以隐藏在 L1 Dcache 的访问延时中. 第 3 级 tagcmp 进行 TAG

比对,判断 L1 是否命中. Godson-T 不使用虚存,所以不存在 TLB, TAG 为地址的高 19 位. 第 4 级为 LSU,其中 empty 缓存是为了协同乘法部件的 4 级执行流水, L1 命中的访存操作可以在 tagcmp 级旁路(forward)结果,但必须在第 4 级多停留一拍后,才能提交指令和改写目标寄存器. 统一设定功能部件 4 级流水的目的是使各个功能部件之间按顺序发射并提交指令,满足精确例外需求;该流水级还包括处理访存缺失请求的 DMQ 以及处理由于替换或写穿透而写回 L2 请求的 WBQ/STB.

WMB 和 L1 之间为包含关系(inclusive),即不会出现 L1 缺失,但是 WMB 命中的情形,在替换出 L1 时候,也必须把相应命中的 WMB 项设置无效. 不同情形下的 Cache 策略具体如下:

对 load 处理:

- (1) 无论临界区内外的 L1 load 缺失,都需要从 L2 回填 L1. 回填前可能需要替换 L1 缓存块,根据

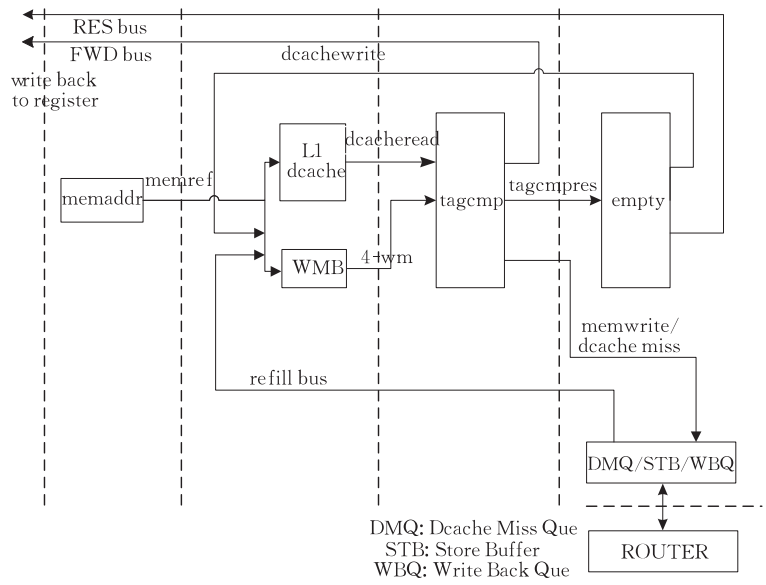


图 6 Godson-T WMB 的 4 级访存流水线

被选替换路的 WM 判断被替换块是否 clean 和是否写回 L2,完成替换后,才真正回填 L1.

(2)临界区内首次 load 访问,L1 命中时需要根据该命中缓存块的 WM 判断是否需要将该缓存块写回 L2,然后设置该命中的缓存块无效.

对 store 处理:临界区内的所有 store 操作和临界区外的 SB/SH/SWR/SWL 指令都为写穿透,写缺失不分配,并且不修改 WMB;只有临界区外的 store,需要设置 WMB 项,共分 4 种情况:

(1)L1 命中,同时该路 WMB 命中或者有空闲项,到达流水线最后一级,提交 store,并设置 L1 和 WMB.

(2)L1 命中,同时该路 WMB 缺失且无空闲项,说明命中缓存块 clean,只需要对 WMB 进行替换,同时将被替换 WMB 项对应的 L1 缓存块(一定 dirty)写回 L2,但不需要无效该缓存块. WMB 的替换策略选用 Round Robin 的方法,store 指令需读出各路 WMB 待替换项的内容,送入 STB(store buffer),完成 WMB 替换后,才能提交 store.

(3)L1 缺失,则优先替换 WMB 有空闲项所在路的 L1,保证 4-way WMB 的均衡性.若某路

WMB 有空闲项,需要根据该路读出的 WM 将 L1 替换写回 L2.完成 L1 回填后,store 修改 L1 缓存块和 WMB 的空闲项,完成提交.

(4)L1 缺失,并且 4 路 WMB 都无空闲项,则按普通的 Cache 替换策略选择某路 Cache 做替换,此时被替换的缓存块可能是 clean 的,所以替换后该路 WMB 依然保持 full 的状态,所以还必须做一次 WMB 替换操作.整体过程相当于(3)到(2).

5.4 WMB 的空间开销

由于 WMB 的项数是 L1 总缓存块的 1/8,每项 WMB 需要保存 8bit 的 WM 和 8bit 的 index,因此空间理论上接近于 WM8 的 25%.本文 L1 Cache 大小为 32KB,理论上仅需要 2048bit 的空间用于保存 WM.

6 实验及评估

6.1 实验平台 Godson-T

本文试验平台为 cycle 精度的事件驱动 C 模拟器 Godson-T,模拟器结构如图 2 所示,结构的参数见表 1.该模拟器实现 mips2 的指令集,我们选

表 1 Godson-T 结构参数

模块	描述
mini-core	8 级流水,顺序双发射,2-ALU, 2-FPU, 1-MEM,但仅有 1 个 DIV/MUL 部件
L1 Dcache	私有,4 路组相联,共 32KB, 32Byte/cache block, 访问命中延迟 1 拍,当 L1 缺失会阻塞流水线
L1 Icache	私有,2 路组相联,共 16KB,缓存块 32Byte,访问命中延迟 1 拍
L2 Cache	总容量 2MB, 16-Bank, 128KB/Bank; 8 路组相联,64Byte/cache block; 12~40 拍的命中延迟;每个 L2 体中有 4-entry 的请求队列,允许 out-standing miss
network-on-chip	8×8 2-D MESH;静态 X-Y routing
router	2 级全流水(routing 和 switch),每个端口有 2 个虚通道,各有 2 个缓存;数据宽度为 128bit,地址域 32bit,少数其它的控制位
memory controller	共 4 个,512bit 宽,1 项的请求队列,读延迟 52 拍,写延迟 32 拍

择 gcc-3.3.3 O32-ABI 的交叉工具链编译应用. Godson-T 的 runtime 系统提供类似 pthread 的编程接口,包括 `thread_create()`, `thread_exit()`, `thread_join()`以及 `barrier()`等;线程管理采用非抢占式.

6.2 应用程序

表 2 列出各个程序的属性. 所有的应用都添加 -O3 的编译优化选项. 我们选取的问题规模接近 L2 的 2M 容量. 其中 lu, fft, radix 选自 splash2 中的

kernel 测试集^[16], 并使用 -t 选项通过正确性验证. pfind 是生物计算程序,它将质谱数据与已知蛋白质序列进行比对,得到相似度分值,平均每个系列大小为 15 字符^[17]. iBlastP 程序是一个类 blastp^[18] 的蛋白质序列比对程序,通过允许空位插入的局部比对方法查找序列之间的相似片段,测试系列均取自于 Swiss-Prot 蛋白质数据库. 这两个典型的生物计算程序,它们在同类算法中较优,是 Godson-T 面向的重要应用领域之一.

表 2 应用程序属性,包括单核(-P1)执行后并行区的指令数和 IPC

程序	问题规模	Total Inst/ 10 ⁶	Total Flops/ 10 ⁶	Total read/ 10 ⁶	Total write/ 10 ⁶	sdcl+sw/ 10 ⁶	IPC
pfind	16384 个系列	293.9	0	58.2	9.2	9.16	0.75
iBlastP	600 个序列	5017.7	0	954.7	62.4	21.2	0.64
lu	512×512 matirx, 16×16 blocks	618.2	98.2	113.9	51.6	50.8	0.64
radix	1M integers, Radix 1024	284.3	133.2	60.9	7.4	7.36	0.55
fft	64k double points	101.8	10.2	24.3	12.3	11.9	0.76

6.3 性能分析

6.3.1 混合 WT/WB 策略同完全 WT 性能的对比

从图 7 中看到,完全 WT 的策略下,除

iBlastP 外加速比均在 16 核后出现瓶颈,甚至负增长. 采用混合 WT/WB 和 WB_ALL 后各程序的性能和加速比均有明显提升,并且各 WB 策略下加速比接近.

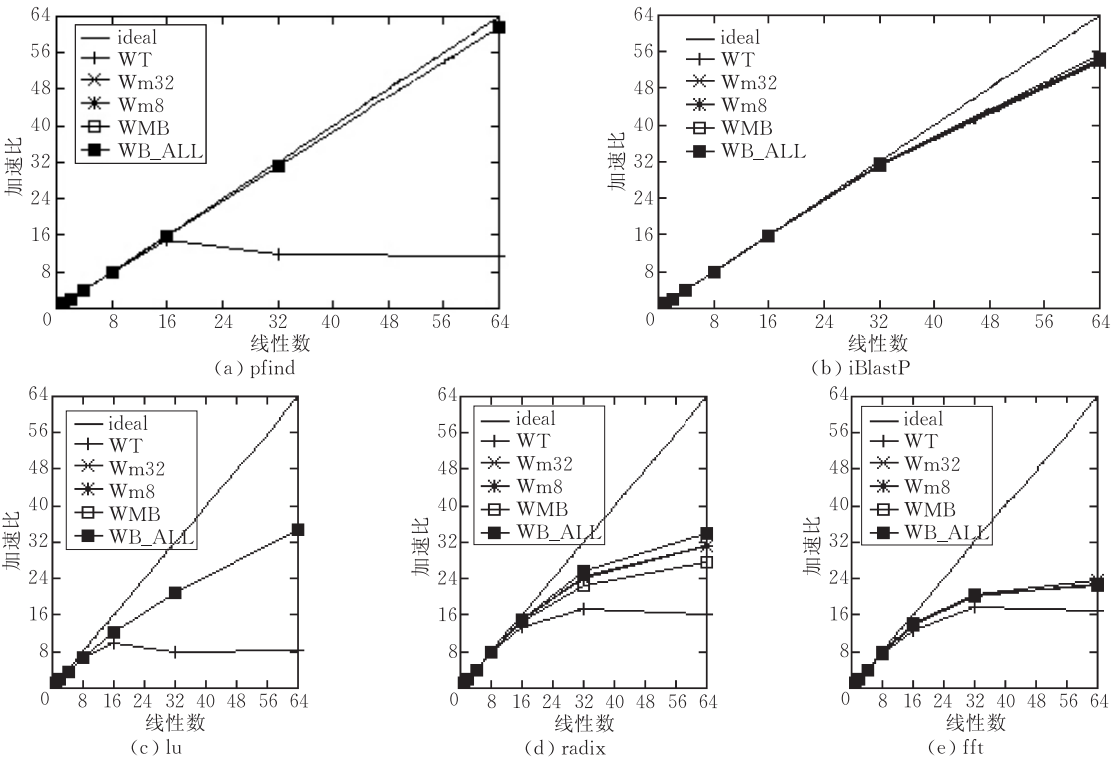


图 7 不同线程数下的加速比

如图 8 所示,其中 pfind 程序的性能在 32 线程和 64 线程下比 WT 分别提高 167%和 428%,加速比在 WM32 下绝对线性. WT 策略下 pfind 程序在 32/64 线程下性能恶化原因在于过多的 store 操作

造成 L2 缓存块的冲突缺失(conflict miss). pfind 程序的 store 指令中大于 95%操作本地的栈空间,而 Godson-T 平台给每个线程预留彼此连续的 8 兆栈空间,所以按照 SIMP 的执行方式和 L2 Bank 按照

地址低位的散列方式,同时可有 32/64 个 store 竞争同一个 Bank,同一个 index 的 8 路 L2 缓存块,彼此替换,造成某些 L2 Bank store 操作的缺失率能超过 50%,间接造成网络拥塞.而混合 WB/WT 策

略下由于程序的数据局部性好、重用率高,所以大部分的 store 由于 L1 命中而直接存储本地 L1,无须到达网络,避免了多线程数下网络和 L2 的冗余负担.

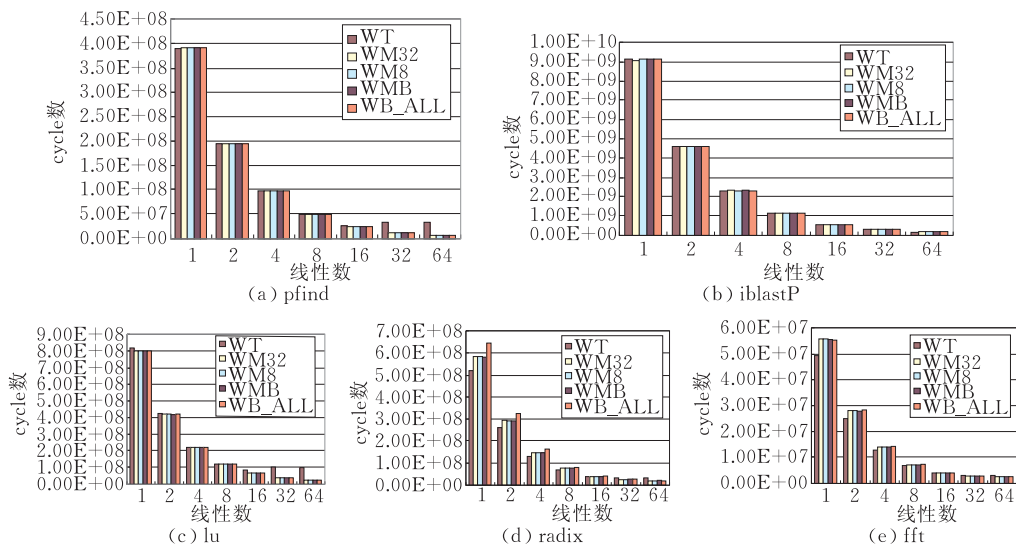


图 8 并行部分执行 cycle 数

iBlastP 程序由于只有 9.9% 的访存指令是 store(见表 2),而且 80% 的指令是用于计算,由于 store 指令只占极少数,计算访存比高,所以各种 Cache 策略下性能差距不大.

fft 和 radix 在 64 线程下,WM32 比 WT 分别提升 24% 和 74%.图 7 中 fft、radix 加速比的提升受限于程序中过多出现 barrier 同步,其中 radix 10 次、fft 6 次.目前 Godson-T 域一致性下 barrier 同步,需要将 L1 内非堆栈空间的数据写回 L2 并将其设置无效,用以保证 barrier 的 fence 语义. L1 集中写回操作,会使网络上产生突发(burst)的大量写回包,造成网络短时间内拥塞.

图 8 中 radix 和 fft 在小于 16 的低线程数下, WB 的性能不如 WT,除 barrier 外,另一个的原因是 Godson-T 的 L1 不支持 out-standing miss,当 store 发生 L1 缺失的时候,需要阻塞流水线,等来较长时间的 L1 回填.如果直接 WT,反而不阻塞流水线. radix 和 fft 程序 store 后的数据重用率较低,不但造成额外的流水线阻塞,还污染 L1.由于有 WM,我们可以进一步优化,设定 store miss 不用等待 L1 回填,而直接修改分配的缓存块.相关的改进想法在第 7 节未来工作(1)中列出.

lu 在 32 线程和 64 线程下分别比 WT 提升 165% 和 326%,并且加速比在 64 线程下能达到 35,原因同 pfind 相似.

6.3.2 两种 WM 存储空间优化方案(WM8/WMB)对性能的影响

通过图 8 的 WM8/WMB 同 WM32 的 cycle 对比,我们可以看出两种空间的优化方案下的性能与 WM32 几乎相同,只有 radix 在 32 和 64 线程下采用 WMB 损失 5%,也说明 WMB 项数采用 L1 缓存块总数 1/8(本文是 4-way \times 32-entry)的设计是合理的.

6.3.3 混合 WB/WT 策略同完全写回(WB_ALL)的性能比较

本文还比较了完全写回策略 WB_ALL 的性能,使用 WM32 的方法记录 WM,在 release 指令执行时,才将临界区内 store 后的脏缓存块一次性集中写回 L2.从图 8 可以看出除 radix 在小于 16 线程数下性能较低外,其它仅略低于混合策略($<0.5\%$),原因在于 release 需要一次性写回大量的 L1 缓存块,形成堵塞,推迟 release 释放锁的时间.我们认为临界区一般较短,不同的 Cache 策略对性能影响很小;但是从芯片面积的消耗上看,WM32 是不合理的,所以不区分临界区内外 WB/WT,至少要保证 SB 和 SH 指令 WT 属性.

6.4 实际芯片面积分析

本文采用 TSMC130 纳米工艺库进行 ASIC 流程实现设计,使用 Synopsys 公司的 Design Compiler 进行物理综合,得到初步的芯片面积和时延数据,对

于大的缓存块,我们均使用 Artisan 公司的 130nm Memory Compiler 生成的标准 RAM 单元,以获得最小的面积.其中,WM32 和 WM8 方案中用于保存 4-way \times 256 个 32/8bit 宽 WM 的寄存器堆,都使用 128 \times 32bit 的单端口 RAM 单元实现,不同的是 WM32 使用 8 个不带掩码的 RAM,而 WM8 中使用 2 个带 4bit 掩码的 RAM,但 RAM 单元面积相同,都为 67860.3 μm^2 ;WMB 实现中由于没有订制的 CAM 单元,所以本文直接用 verilog 写 for 循环比对普通寄存器的方式实现 4 次每次 32 项的 CAM 查找(8bit 宽),这部分的组合逻辑需要 28386.8 μm^2 ,尽管如此,如图 10 所示 WMB 比 WM8 有 42%的面积节省.WM8 和 WMB 下存储和组合逻辑面积总和分别为 WM32 的 27.9%和 17.7%.从图 9 可以看出 WM32 下写掩码的总面积逻辑占 min-core 的总面积(WM32: 4425560 μm^2)的 13%,因此两种方法确实有效节省了整个芯片的面积.

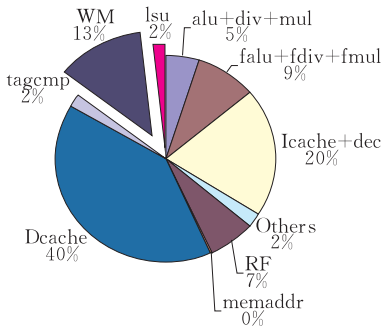


图 9 Godson-T mini-core 各部分面积比例(WM32)

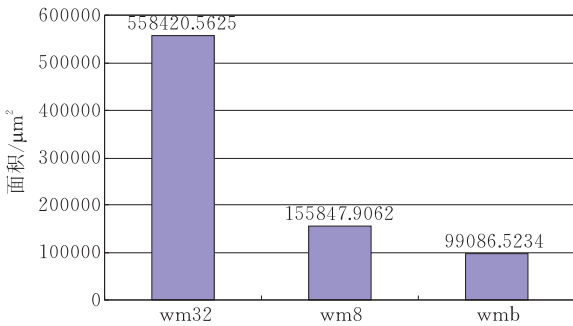


图 10 WM 存储逻辑+组合逻辑面积

7 结论和未来工作

本文提出 L1 保存写掩码的方法,成功解决了域一致性存储模型下写回策略在伪共享时产生的正确性问题,在我们搭建的 Godson-T 众核平台上实现了混合写回策略.实验表明,写回策略对于缓解网络和 L2 的压力是至关重要的,它能够维持 32 及 64

线程配置下的加速比.相对于写穿透策略,性能普遍提升 24%以上.对于写掩码的空间开销,本文还相应提出了两种优化方案:WM8 和 WMB,在性能同 WM32 和 WB_ALL 几乎相同的前提下,将 WM 的芯片面积分别缩小到 WM32 的 27.9%和 17.7%,减少了 40 多万平方微米的芯片面积.

我们下一步工作是:(1)研究写掩码方法对位置一致性(LC)^[19]或者 CRF 一致性^[20]等弱一致模型普遍适用性.(2)L1 添加 WM 后,可以实现 no-fetch-on-write 的写策略^[21];当 store L1 缺失,不必等待 L2 回填,而直接修改分配的空闲 L1 缓存块的相应字节,并使用 WM 记录位置.预期能更好的支持流式应用.(3)如果有定制的 CAM,可以进一步优化 WMB 的面积开销.

参 考 文 献

- [1] Huang He, Yuan Nan, Lin Wei et al. Architecture supported synchronization-based cache coherence protocol for many-core Processors//Proceedings of the ISCA Workshop on Chip Multiprocessor Memory Systems and Interconnects. Beijing, China, 2008: 51-53
- [2] Asanovic Krste, Bodik Ras et al. The landscape of parallel computing research: A view from Berkeley. University of California, California, USA; Technical Report UCB/EECS-2006-183, 2006
- [3] Culler David E, Singh Jaswinder Pal, Gupta Anoop. Parallel Computer Architecture: A Hardware/Software Approach. San Fransisco, USA; Morgan Kaufmann, 1998
- [4] Hu Wei-Wu, Shi Wei-Song, Tang Zhi-Min. A software dsm system based on a new cache coherence protocol. Chinese Journal of Computers, 1999, 20(5): 467-474(in Chinese)
(胡伟武,施巍松,唐志敏.基于新型 cache 一致性协议的共享虚拟存储系统.计算机学报, 1999, 20(5): 467-474)
- [5] Iftode Liviu, Singh Jaswinder Pal, Li Kai. Scope consistency: A bridge between release consistency and entry consistency//Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures. Padua, Italy, 1996: 277-287
- [6] Karp A H, Sarkar Vivek. Data merging for shared-memory multiprocessors//Proceedings of the 26th Hawaii International Conference on System Sciences. Hawaii, USA, 1993: 244-256
- [7] Lenoski Daniel, Laudon James, Joe Truman et al. The dash prototype: Implementation and performance//Proceedings of the 19th International Symposium on Computer Architecture. Queensland, Australia, 1992: 92-103
- [8] Lamport Leslie. How to make a multiprocessor computer that correctly executes multiprocess program. IEEE Transactions on Computers, 1979, 28(9): 690-691

- [9] Adve Sarita V, Hill Mark D. Weak ordering—A new definition//Proceedings of the 17th International Symposium on Computer Architecture. Seattle, USA, 1990: 2-14
- [10] Gharachorloo Kourosh, Lenoski Daniel et al. Memory consistency and event ordering in scalable shared-memory multiprocessors//Proceedings of the 17th International Symposium on Computer Architecture. Seattle, USA, 1990: 15-26
- [11] del Cuvillo Juan, Zhu Wei-Rong et al. Toward a software infrastructure for the cyclops-64 cellular architecture//Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment. St. John's, Canada, 2006: 9-20
- [12] Gschwind Michael, Hofstee H Peter et al. Synergistic processing in cell's multicore architecture. IEEE Micro, 2006, 26(2): 10-24
- [13] Taylor Michael Bedford, Lee Walter et al. Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ilp and streams//Proceedings of the 31st International Symposium on Computer Architecture. München, Germany, 2004: 2-13
- [14] Wentzlaff David, Griffin Patrick et al. On-chip interconnection architecture of the tile processor. IEEE Micro, 2007, 27(5): 15-31
- [15] Kāgi Alain, Burger Doug, Goodman James R. Efficient synchronization: Let them eat qolb//Proceedings of the 24th International Symposium on Computer Architecture. Denver, USA, 1997: 170-180
- [16] Woo Steven Cameron, Ohara Moriyoshi, Torrie Evan. The splash-2 programs: Characterization and methodological considerations//Proceedings of the 22nd International Symposium on Computer architecture. S. Margherita Ligure, Italy, 1995: 24-36
- [17] Fu Yan, Yang Qiang, Sun Rui-Xiang, Li De-Quan et al. Exploiting the kernel trick to correlate fragment ions for peptide identification via tandem mass spectrometry. Bioinformatics, 2004, 20(12): 1948-1954
- [18] Altschul S, Madden T, Schaffer A et al. Gapped blast and psi-blast: A new generation of protein database search programs. Nucleic Acids Research, 1997, 25(17): 3389-3402
- [19] Gao Guang R, Sarkar Vivek. Location consistency—A new memory model and cache consistency protocol. IEEE Transactions on Computers, 2000, 49(8): 798-813
- [20] Shen Xiao-Wei, Arvind, Rudolph Larry et al. Commit-reconcile & fences (crf): A new memory model for architects and compiler writers//Proceedings of the 26th International Symposium on Computer Architecture. Atlanta, USA, 1999: 150-161
- [21] Jouppi Norman P. Cache write policies and performance//Proceedings of the 20th International Symposium on Computer architecture. San Diego, USA, 1993: 191-201



LIN Wei, born in 1979, Ph. D. candidate. His research interests focus on building next-generation hardware and software compute infrastructure using many-core, massively parallel and speculating execution approach.

YE Xiao-Chun, born in 1981, Ph. D. candidate. His re-

search interests include parallel algorithm and high performance computer architecture.

SONG Feng-Long, born in 1980, Ph. D. candidate. His research interests focus on memory hierarchy and performance evaluation.

ZHANG Hao, born in 1980, Ph. D. His research interests include virtualization and multi-core architecture.