计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20150154
52(6):1254-1265,2015

MACT:高通量众核处理器离散访存请求批量处理机制

李文明[1,2] 叶笑春[1] 王 达[1] 郑 方[4] 李宏亮[4] 林 晗[3] 范东睿[1] 孙凝晖[1]
[1](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
[2](中国科学院大学计算机与控制学院 北京 100049)
[3](中国科学技术大学计算机科学与技术学院 合肥 230022)
[4](数学工程与先进计算国家重点实验室 江苏无锡 214125)
(liwenming@ict.ac.cn)

MACT: Discrete Memory Access Requests Batch Processing Mechanism for High-Throughput Many-Core Processor

Li Wenming[1,2], Ye Xiaochun[1], Wang Da[1], Zheng Fang[4], Li Hongliang[4], Lin Han[3], Fan Dongrui[1], and Sun Ninghui[1]
[1](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*School of Computer and Control Engineering*, *University of Chinese Academy of Sciences*, *Beijing* 100049)
[3](*School of Computer Science and Technology*, *University of Science and Technology of China*, *Hefei* 230022)
[4](*State Key Laboratory of Mathematical Engineering and Advanced Computing*, *Wuxi*, *Jiangsu* 214125)

Abstract The rapid development of new high-throughput applications, such as Web services, brings huge challenges to traditional processors which target at high-performance applications. High-throughput many-core processors, as new processors, become hotspot for high-throughput applications. However, with the dramatic increase in the number of on chip cores, combined with the property of memory intensive of high throughput applications, the "memory wall" problems have intensified. After analyzing the memory access behavior of high throughput applications, it is found out that there are a large proportion of fine-grained granularity memory accesses which degrade the efficiency of bandwidth utilization and cause unnecessary energy consumption. Based on this observation, in high-throughput many-core processors design, memory access collection table (MACT) is implemented to collect discrete memory access requests and to handle them in batch under deadline constraint. Using MACT hardware mechanism, both bandwidth utilization and execution efficiency have been improved. QoS is also guaranteed by employing time-window mechanism, which insures that all the requests can be sent before the deadline. WordCount, TeraSort and Search are typical high-throughput application benchmarks which are used in experiments. The experimental results show that MACT reduces the number of memory accesses requests by 49% and improves bandwidth efficiency by 24%, and the average execution speed is improved by 89%.

Key words high throughput processor; memory access collection table (MACT); time window mechanism; cache; scratchpad memory(SPM)

摘 要 网络服务等新型高通量应用的迅速兴起给传统处理器设计带来了巨大的挑战.高通量众核处理器作为面向此类应用的新型处理器结构成为研究热点.然而,随着片上处理核数量的剧增,加之高通量

收稿日期:2015-02-15;**修回日期**:2015-04-21
基金项目:国家"九七三"重点基础研究发展计划基金项目(2011CB302501);国家"八六三"高技术研究发展计划基金项目(2012AA010901,2015AA011204);"核高基"国家科技重大专项基金项目(2013ZX0102-8001-001-001);国家自然科学基金项目(61173007,61332009,61204047)

应用的数据密集型特点,“存储墙”问题进一步加剧.通过分析高通量应用访存行为,发现此类应用存在着大量的细粒度访存,降低了访存带宽的有效利用率.基于此分析,在高通量处理器设计中通过添加访存请求收集表(memory access collection table, MACT)硬件机制,结合消息式内存机制,用于收集离散的访存请求并进行批量处理.MACT硬件机制的实现,提高了访存带宽的有效利用率,同时也提高了执行效率;并通过时间窗口机制,确保访存请求在最晚期限之前发送出去,保证任务的实时性.实验以典型高通量应用WordCount,TeraSort,Search为基准测试程序.添加MACT硬件机制后,访存数量减少约49%,访存带宽提高约24%,平均执行速度提高约89%.

关键词 高通量处理器;访存请求收集表;时间窗口机制;高速缓冲存储器;便签式存储器

中图法分类号 TP302

随着互联网技术的迅猛发展,新型的高通量应用已经成为主流.以社交网络、云计算和网络多媒体等为代表的新型高通量应用,其程序特征已经从传统的浮点计算变为了处理大量高并发的用户服务请求.这类应用的目标已不再是传统的LINPACK[1]速度而是高通量,即提高单位时间内处理的并发任务数目[2-3].这使得面向传统高性能应用的处理器体系结构在针对网络服务这种高并发、高实时、低延迟的高通量应用时表现出了多方面的不足[4].高通量处理器研究逐渐成为体系结构发展的热门方向之一[5-6].

无论是传统的高性能处理器还是新型的高通量处理器,它们都属于经典的冯.诺依曼体系结构.因此,“存储墙”仍是高通量处理器需要面对的巨大挑战[7].更糟糕的是,为了适应高并发、高吞吐等应用特征,片上处理器逐渐发展到百核乃至千核级[8-9].图1所示为国际半导体技术蓝图(International Technology Roadmap for Semiconductors, ITRS)对未来单个芯片上的处理器数目的预测.从图1可以看出,到2026年单芯片上的主处理器数目将接近200个,而数据处理单元数将达到1 000个以上[10].如此大规模的片上处理单元,再加之高通量应用属于数据密集型应用[11-12],进一步加剧了高通量处理器的“存储墙”问题.另一方面,对于存在大量细粒度访存的高通量应用程序来说,以高速缓存行(cache line)为粒度的访存方式,既浪费了有效带宽的利用率[13-14],也影响了应用程序的整体执行效率.

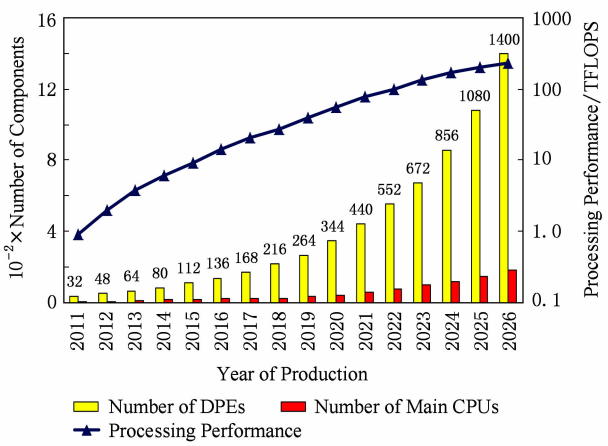


Fig. 1 Prediction of CPU number and performance on a chip from ITRS.

图1 ITRS对芯片上处理器数目和性能的预测

本文通过分析高通量应用访存行为发现,高通量应用属于访存密集型,绝大部分的访存粒度小于64 B,并且数据重用距离长.这会导致大量Cache Line的数据空间局部性在发挥作用之前就被替换出Cache;另一方面,高通量应用追求的不再是尽量缩短单个任务的处理时间,而是在允许的时间范围内处理尽可能多的任务[2].因此高通量应用在保证实时性的情况下,可以容忍一定程度的延迟.

基于以上高通量应用的特性,受城市交通中“定制公交”的启发,本文首次提出了访存请求收集表(memory access collection table, MACT)硬件机制,通过对处理器核发出的离散访存请求的收集,提升高通量处理器访存带宽的有效利用率,缓解“存储墙”问题.MACT的主要功能及特点在于:1)能够按照处理逻辑实际使用的数据大小为粒度进行访存;2)收集离散的访存请求,进行批量处理,高效利用访存带宽;3)通过硬件窗口机制,保证任务的实时性需求.本文实验基于128核片上众核模拟器,对MACT硬件机制进行评估.结果表明,使用MACT硬件机制后,虽然单个消息由于收集等待的原因导致延迟有所增加,但是访存请求数目减少约49%,访存带宽提高约24%,平均执行速度提高约89%.

1 相关工作

近几年,针对众核处理器中的大量细粒度存储访问带来的性能降低以及Cache粗粒度划分造成的

资源浪费的问题,研究人员提出了不同的解决方案,下面介绍3种具有代表性的研究成果.

Liu等人在文献[15]中,通过添加Vector DMA Cache(VDC)对向量DSP中的访存数据通路效率进行了提高.其主要做法是在DSP中DMA总线和向量存储器(VM)之间添加VDC.通过VDC的添加,可以大量命中DMA请求,且与VM的数据交换只有在VDC进行数据替换时发生.因此,可以显著降低通路中的访存请求数目,减少请求冲突,缩短请求延迟,从而提高执行效率.此实现机制与本文的不同在于,本文是在使用便签存储器(scratchpad memory, SPM)片上存储的情况下,对访存通路中存在的大量细粒度的访存请求进行收集和批量处理,以达到减低访存请求数量,提高带宽利用率,进而提高执行效率的目的.

Yuan等人在文献[16]中,通过观察GPU发出的访存请求后发现,访存请求具有一定行访问局部性,内存控制器通过乱序调度进行优化可以达到更好的局部性;但是相对于顺序调度来说,乱序调度资源消耗太大.作者发现访存请求的局部性是被数据通路的传输仲裁策略所破坏,于是提出了"Hold Grant"和"Row-Matching Hold Grant"调度策略,在传输的过程中保护了访存请求的局部性,通过简单的顺序调度接近了算法复杂且开销大的乱序调度的性能.此机制是通过修改访存通路上的仲裁和调度策略来保护和优化访存行为的局部性特点,提高访存效率.与本文提出的MACT相比,虽然目的都是提高访存效率,但是思想和实现方法并不一样.MACT是通过收集离散细粒度访存批量处理来提高访存效率,且应用的环境也存在明显的不同.

Cache细粒度划分和访存策略是另一类解决因Cache粗粒度管理造成的资源浪费和低效的方法,如Vantage[17], PIPP[18], Decay-based Replacement[19]等是Cache细粒度划分和管理的代表性工作.其主要思路是打破传统Cache中以Cache Line或是Cache Way为粒度的划分和替换策略.因为多核或单核同时多线程(simultaneous multithreading, SMT)技术的出现导致Cache被多核多线程所共享,粗粒度的管理会导致Cache性能的降低,因此,按照线程、任务或固定细粒度为单位对Cache进行细粒度的划分和访存操作会带来性能上的提升.Cache细粒度管理与本文提出的MACT存在本质上的区别,MACT是在数据通路上收集细粒度访存请求进行批量处理,是一种对细粒度访存请求的管理机制;而Cache的细粒度划分和访存策略是在通路前端实现的优化策略,并不影响访存请求在通路中的传输过程.

2 高通量应用访存特点分析

高通量应用具有并发度高、访存比大、访存粒度小等新特点[11].为了引导后续对高通量处理器访存通路的设计,本文首先对传统多核片上存储结构在新型高通量应用下的性能表现进行评估和分析.

高通量应用包括离线大数据分析、在线实时大数据分析和实时交互等类型的负载.为了尽可能覆盖高通量应用各方面的特征,达到充分测试和评估的目的,实验选用WordCount, TeraSort, Search三个典型的基于MapReduce[20]的高通量应用作为分析和实验的基准测试程序.MapReduce是面向集群系统的一种并行编程框架,已经被广泛应用于不同应用的分布式并行处理系统中.WordCount和TeraSort来自Phoenix++[21].Phoenix首次把MapReduce编程框架移植到了共享存储多核系统上,Phoenix++又从算法、实现细节和系统调用3个层面对Phoenix进行了深度优化.Search来自Xapian工程[22](开源搜索引擎项目),分为Search Index(索引建立)和Search Quest(实时交互)2部分.

本实验硬件平台选用Intel Xeon E5645多核处理器对上述高通量应用的访存行为进行相关测试.实验平台配置及实验基准测试程序介绍分别如表1和表2所示:

Table 1 Configuration of Experiment Platform

表1 实验平台配置

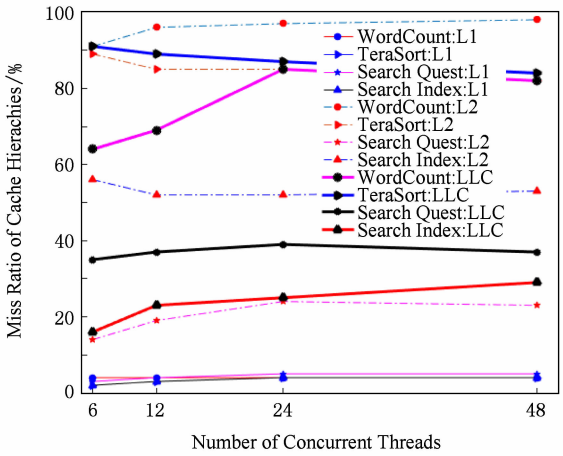
Component	Configuration
CPU	Intel Xeon E5645 2.4 GHz, 2 CPU, 6 Physical Cores/CPU, 12 Cores in Total 2-Way HT, 24 Logic Threads in Total
L1 Cache	Private L1 Cache, 32 KB I-Cache, 32 KB D-Cache for Each Core
L2 Cache	Private L2 Cache, 256 KB
LLC	12 MB, CPU Shared

Table 2 Introduction of Experiment Benchmarks

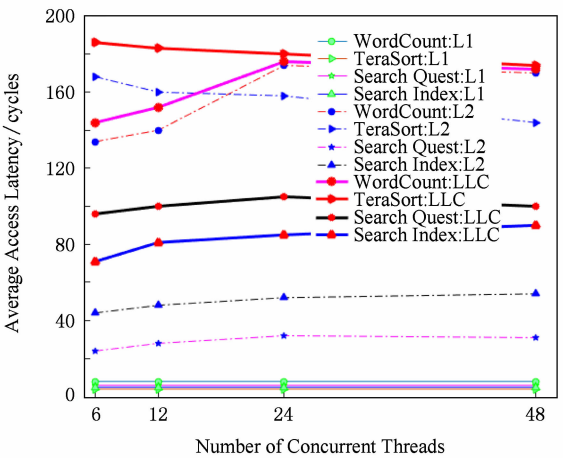
表2 实验基准测试程序介绍

Application	Data Set
WordCount	4 GB
TeraSort	1 GB
Search Index	1.6 GB HTML Files, 1 GB Index Items
Search Quest	1 GB Index Files, 2×10^5 Requests

从硬件数据通路对高通量应用的支持角度分析,实验结果如图 2 所示.从图 2 可以看出,3 类高通量应用在访存行为上存在一个共同现象:L1 Data Cache 的命中率非常高,L2 Cache 的缺失率很高,LLC 的缺失率介于 L1 Cache 和 L2 Cache 之间.这说明这些应用在访存行为上的一个共同特征:数据有很好的时间局部性,但是数据在 L2 Cache 中的重用距离超过了 L2 Cache 的相联度,导致 L2 Cache 的访问几乎全部是缺失;而 LLC 的相联度大并且容量也大,反而比 L2 Cache 有更多命中,但是绝对命中率也并不理想.从面积和功耗的角度来说,L2 Cache 和 LLC 的命中率并不高,但却占用了大量的片上面积,同时也带来了较高的功耗比.因此,在高通量应用环境下,传统多级 Cache 带来的性能提高相对于其所带来的面积和功耗来说,收益并不明显.



(a) Miss ratio of caches under different configurations



(b) Access latency of caches under different configurations

Fig. 2 Cache performance in traditional multi-core processor under four applications.

图 2 传统多核处理器 Cache 结构在 4 种测试程序下的性能统计

从高通量应用访存特性对硬件的需求角度分析,本文对高通量应用访存特性进行了测试和分析,图 3 为 3 类应用访存粒度分布统计结果,实验结果是在 64 位主机上使用 Pin tools[23] 统计得出.从图 3 可以看出,绝大多数的访存粒度不超过 8 B.若在 32 位主机上测试,大部分粒度不超过 4 B.

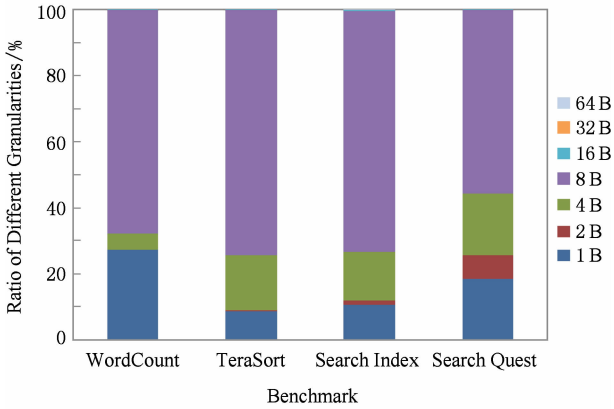


Fig. 3 Memory access granularity distribution.

图 3 访存粒度分布统计图

图 4 统计结果为当一个 Cache Line 被替换出 L1 Data Cache 时,整个 Cache Line 中未被使用的字节数所占比例.从图 4 可以看出,对于典型高通量应用来说,30%以上的 Cache Line 字节数未被使用就被替换出 Cache,TeraSort 的比例甚至超过了 50%;大量的无用数据在 Cache 和 Memory 之间传输,浪费了带宽的有效利用率同时也降低了程序的执行效率.

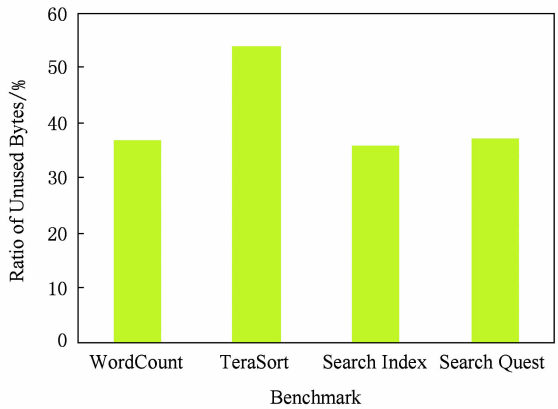


Fig. 4 Proportion of unused bytes within a Cache Line.

图 4 Cache Line 被替换时未被使用字节比例

本文通过上述实验及分析发现:1)一级 Cache 几乎消耗完了所有的访存局部性,多级 Cache 的作用并不大而且还占用较大的片上面积和功耗,此结果与文献[24-26]得出的结论相符.2)在高通量应用

中,绝大部分的访存粒度不会超过 8 B,而在主流处理器设计中 Cache Line 粒度通常为 64 B,这会导致很多数据在未使用之前就被替换出 Cache,浪费片上存储资源及传输带宽.

因此,在高通量应用的细粒度访存环境中,Cache Line 挖掘程序的空间局部性效果并不好,而且还会导致大量无用数据的搬运,降低带宽的有效利用率,影响应用程序的整体执行效率.因此,需要使用其他片上存储器来替代 Cache 的缓存作用,还能以真实粒度进行访存.

3 支持 MACT 机制的片上访存结构设计

通过第 2 节的分析可知,多级 Cache 在高通量应用负载下表现的并不理想.而且,在传统的处理器设计中,Cache 通常占片上整体面积的比例较大,也会带来较大的功耗[27]. SPM[28] 是一种和 Cache 类似的片上可编程存储器,相对于 Cache 来说,SPM 在同等容量下具有能耗低、存储面积利用率高等优点.研究表明,SPM 与同等容量的 Cache 相比,功耗平均节省 40%,面积节省约为 56%[28].而且,两者在访问时间量级上相同,都可以在数个时钟周期内完成.因此,高通量处理器使用 SPM 作为片上数据存储结构,充分利用 SPM 在真实访存粒度上的优势,进一步改善高通量处理器的性能.

在高通量处理器中,片上处理器核以线程核组(thread core group, TCG)为单位,如图 5 所示.每

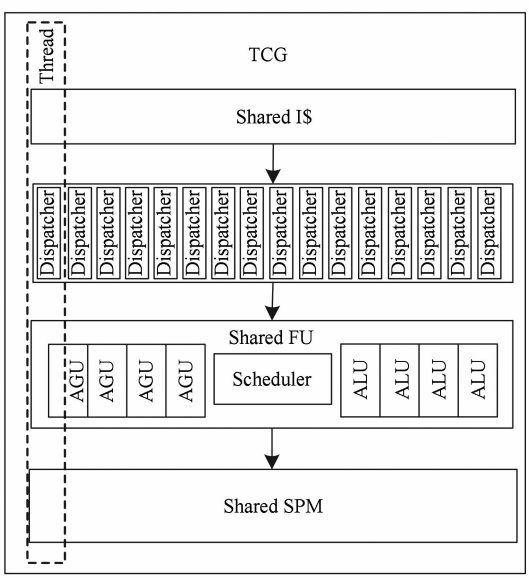


Fig. 5 TCG logic architecture.

图 5 TCG 逻辑结构图

个 TCG 包括 16 个逻辑核,属 SMT 流水线设计,逻辑核之间共享功能部件. TCG 中线程之间共享 SPM,用于与消息式内存控制器[13]交换可变粒度的访问. SPM 为全局共享空间,即虽然每个 TCG 拥有一份 SPM 存储空间,但是所有 TCG 之间可以共享这些 SPM 存储空间. SPM 与消息式内存直接进行数据交换,且 TCG 之间无需维护数据一致性.通过对高通量应用分析及参考目前主流商用处理器片上存储的设计,给每个 TCG 配置 32 KB 的 SPM 空间.

在高通量处理器访存通路设计中,SPM 与内存之间的数据交换是基于类似渗透(percolation)[29]的局部性优化思想实现的.在 TCG 中,选取一个独立线程作为 Helper 线程,负责 SPM 与内存之间的数据迁移,Helper 线程可以根据程序控制预先对 SPM 中的数据进行替换.因此只有当某个线程需要访问数据时,该数据才出现在片内存储中.该技术的合理性是基于 IBM Cyclops64 的多线程执行模型.其中,SPM 需要存储和替换的数据由程序员在程序中指定和控制.因此,在设计中添加了新的访存指令支持此类操作,语义为:load mem_addr, spm_addr 以及 store spm_addr, mem_addr.功能是将内存中的数据下载(load)至 SPM 中或将 SPM 中的数据存储(store)至内存中.

在高通量处理器任务管理方面,实现了基于 Pthread 的多线程管理操作系统 Runtime.以典型的高通量应用 MapReduce 框架为例,其利用多线程的加速方法是将要处理的文件(document)分割成为相互不同的片段(split);然后每个线程处理自己的文本段并记录结果;最后再进行一个将结果合并的过程.其中,Map 过程中生成的多线程会逐一映射到各 TCG 中的硬件多线程上.每个线程独立完成处理任务之后在 Reduce 阶段将结果统一写回至主线程中.因为 SPM 是所有 TCG 之间可共享的,所以可以实现上述 Map 和 Reduce 过程中的各数据迁移和控制.

高通量应用中存在大量的细粒度访存,通过使用 SPM 片上存储实现按照处理逻辑实际使用的数据大小粒度进行访存,以此减少带宽和存储空间的浪费,提高执行效率.但是大量的细粒度离散访存会给片上网络带来巨大的传输压力.为此,如图 6 所示,本文在 TCG 访存请求端口上实现 MACT 硬件机制,用于收集 TCG 中 SPM 发出的离散访存请求,并按照访存截止时间(deadline)的需求批量处理收集的访存请求,最后将收集到的访存请求打包发

送到消息式内存控制器[13],以达到改善访存带宽的有效利用率,提高执行效率的目的.

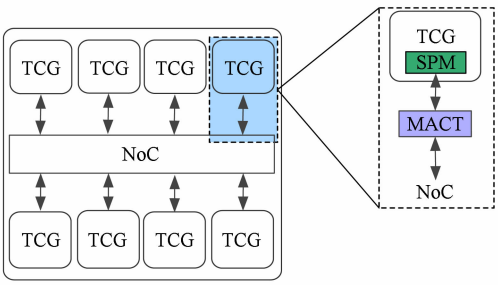


Fig. 6 Partial architecture of high-throughput many-core and data path.

图 6 高通量众核处理器局部结构及数据通路示意图

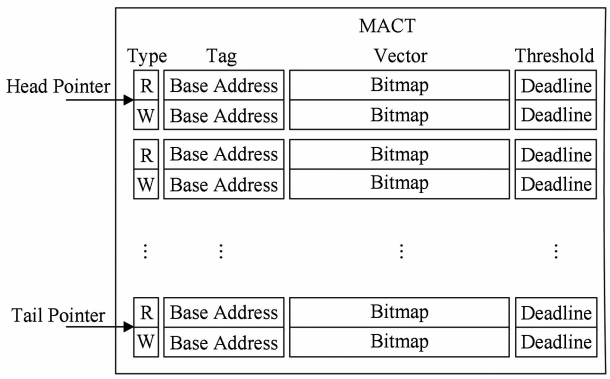


Fig. 7 Structure of MACT.

图 7 MACT 内部结构示意图

4 MACT 硬件机制

MACT 内部实现结构如图 7 所示,MACT 中每一行有 4 个域:Type,Tag,Vector,Threshold,分别代表操作类型、基地址、位向量、时间阈.操作类型(Type)表示本行请求是读操作(R)还是写操作(W);基地址(Tag)表示本行中收集到的访存请求的地址范围落在此基地址加上 Vector 所能表示的地址范围之内;位向量(Vector)使用 0/1 编码用于表示此地址空间是否有访存请求,用于收集以基地址为起始域的连续地址的访存请求;时间阈(Threshold)记录本行收集到的访存请求的最晚发送时间阈值,用于保证访存请求的实时性.高通量处理器对发出的访存操作设置裕度时间,在确保实时性的同时,用于控制合并后的访存请求发给内存控制器的时机.

在 MACT 中,Vector 域每 1 b 映射内存范围 8 B,每行 128 b,因此每行映射的内存范围为 1 KB.每行设置 Tag 域,Tag 即为 64 b 内存地址除去 1 KB 的偏移量(offset)后剩余的地址位,用于收集该起始地址开始的 1 KB 地址范围内的访存请求.每个 MACT 共有 32 个逻辑行,因此一个 MACT 只映射内存的 32 KB 内存地址空间.在 MACT 中,为了便于识别和批量管理,读写操作分开处理.一个基地址对应 2 行,分别为读操作行和写操作行;但是,两者视为一个逻辑行,只是当 MACT 接收到访存请求时,对访存请求的类型进行判断,写入不同的读行或写行.

MACT 工作原理及步骤如图 8 所示.TCG 中 SPM 发出的不同粒度的访存请求按照 Type 和 Threshold 要求被 MACT 所收集,如图 8 步骤①所示;然后按照访存请求的类型、地址和时间要求分别插入不同的收集行中等待发送,当 MACT 的某一行满或者达到最晚发送要求时(根据访存裕度时间确定),本行访存请求会被打包以一个访存消息包的形式发送到片上网络,如图 8 步骤②所示;MACT 发出的访存消息包经过片上网络传递给消息式内存控制器,内存控制器对消息包进行解包和处理,最终发送至消息式内存,完成读写操作,如图 8 中步骤③所示.

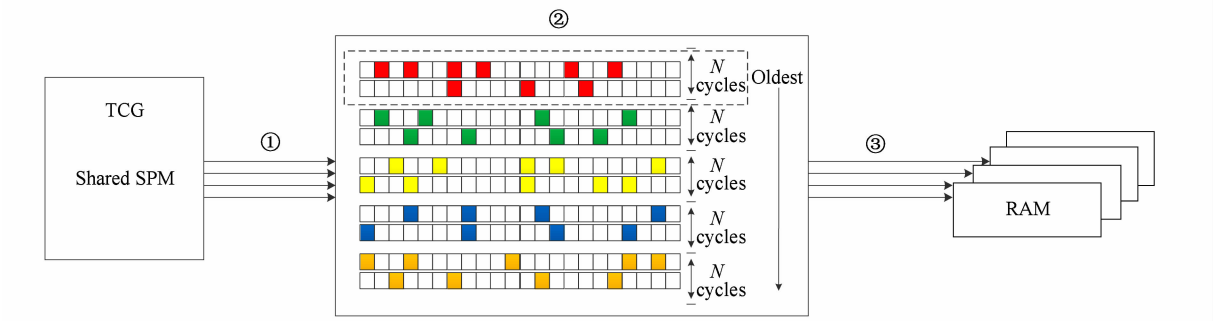


Fig. 8 Three Steps of MACT operations.

图 8 MACT 操作步骤

MACT 将请求消息发送到 Memory 以后,不再缓存请求消息.

当 Memory 回填 SPM 时,将对应的基地址和位向量返回,MACT 根据 Memory 返回的基地址与位向量,解析数据,发送至 SPM 中.

4.1 MACT 窗口机制

在 MACT 中,一个固定的时间段被称为一个窗口. 在窗口内向 MACT 提交访存请求被称为窗口填充,如图 9(a)所示. 窗口在满足一定条件后关闭(时间到或空间满),将窗口中的访存请求交给分检模块处理,将可以合并的访存请求按照消息类型进行分类,同时打开下一个窗口继续收集访存请求,如图 9(b)所示.

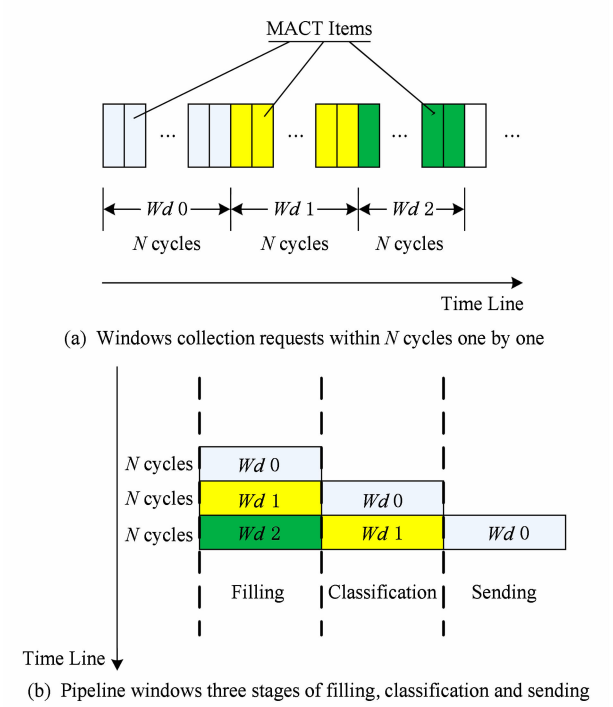


Fig. 9 Pipeline of MACT operations.

图 9 MACT 操作流水

分检模块将收集到的访存请求,按照请求类型、请求地址的不同,分别进行处理和打包,交给发送模块. 发送模块一旦检测到有数据包需要发送且网络接口允许注入数据包,则将需要发送的打包后的访存集数据包交由片上网络传输至内存控制器.

4.1.1 窗口关闭的条件

影响窗口操作的限制条件主要有 2 个:访存请求最晚发送时间和 MACT 收集容量:

1) 窗口开放时间满足时间上限 N cycles. 如图 9(b)所示,若一批访存请求发送的最晚时间相比收集时刻延迟 T cycles,则 $N=T/3$. 为了确保服务质量,必须能够保证将收集到的访存请求在发送 Deadline 之前发出.

2) 窗口内的访存请求达到上限(M 个访存请求,在此为 32 个逻辑行). 因为 MACT 容量有一定的限制,当容量满时窗口提前关闭.

4.1.2 窗口参数设定

窗口机制类似处理器内部流水线,是保证 MACT 正确运行的基本准则. 窗口参数的设置必须能够满足访存请求在要求最晚之前被处理和发送出去. 具体的参数设定如下:

1) 应保证分检模块在 N cycles 内能完成窗口内访存请求的分类处理,即 N cycles 大于等于分类处理时间;

2) 应保证发送模块在 N cycles 内能将分类处理过的访存请求发送完毕,即 N cycles 大于等于发送时间;

3) M 值应保证窗口内的访存请求能够在 N cycles 内被分类处理模块处理完毕,即 N cycles 大于等于处理 M 条访存请求所需的时间;

4) M 和 N 的值应能保证不延长访存请求所要求的最晚完成时间.

4.2 MACT 多类型访存支持

为了支持处理器发出的各种形式的访存请求,MACT 也必须提供多种形式访存功能,包括单地址请求、向量请求、批量请求、命令组合式请求.

1) 单地址请求. 若对时间要求不高,可以将 MACT 中的阈值标记为低优先级,从而将 SPM 的单个请求以向量访存或者批量访存的方式发送到消息式内存控制器;若对时间要求高,会在 MACT 的阈值域标记为高优先级,SPM 的请求不经过时间窗口直接发送.

2) 向量请求. 由 MACT 收集,某一行的阈值达到后,MACT 将该行的请求打包发送到消息式内存控制器.

3) 批量请求. 将 MACT 收集表中跨行的请求打包发送到内存控制器中. MACT 中消息发送是以时间窗口的方式管理. MACT 将物理时间分成多个窗口,没有特殊相应时间标记的请求消息会标记为上一时间窗口的信息,当某时间窗口的等待时间完成,则将该窗口内的消息集体发送:①该行请求(1 KB 的地址范围)若有向量特性,按向量方式访存;②对于没有向量特性的行,则多行打包统一发送.

4) 命令组合式请求. 在线程核组中新增全局地

址空间编址的 MACT 控制寄存器:*type*,*src_addr*,*dst_addr*,*stride*,*len*,*num_op*. 其中,*type* 表示功能类型编码,*src_addr* 表示源内存空间地址,*dst_addr* 表示目的内存空间地址,*stride* 表示操作跨度(用于 gather/scatter 操作),*len* 表示操作数长度,*num_op* 表示操作数个数. 通过访存操作对该地址进行读写同时 *type* 寄存器复用为发射寄存器,当收到对 *type* 寄存器的写操作时,控制逻辑会对上述寄存器的内容打包发送到消息式内存控制器.

4.3 添加 MACT 机制后的数据通路

MACT 中不存储访存的真实数据,只存储相关的请求信号.

如果是 SPM Write 操作,表示 SPM 将数据写回到 Memory,需要先到 SPM 中将数据取出,整合成访存的信息格式后发送到 MACT 中,根据请求类型分别归入不同的类别,在合适的时间发送到内存控制器. 数据顺利写到 Memory 之后,发送一个确认消息 ACK 给 MACT,将其中对应项置为无效. 因为 MACT 中没有数据域,SPM 的 Write 操作分请求与写回 2 个阶段,如图 10 所示. 首先沿请求网络向 MACT 中发 Write 的请求消息,当在 MACT 中排队成功以后,MACT 回复 ACK 给 SPM,SPM 沿数据网络发送数据到内存.

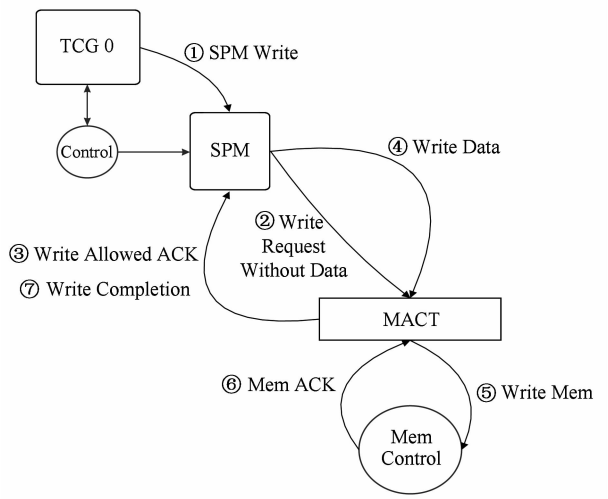


Fig. 10 Data path of SPM Write operation.

图 10 SPM 写操作数据通路示意图

如果是 SPM Read 操作,表示从内存中将数据读取至 SPM 中. 内存地址和读取的数据大小等信息都在寄存器堆中有保存. 读请求首先发送到 MACT 中,请求经过处理整合之后传送到内存控制器,内存控制器负责取出要读取的数据回填至 SPM 中. 数据通路示意图类似写操作,在此不重复.

5 实 验

5.1 实验平台

实验平台是由中国科学院计算技术研究所开发的大规模并行模拟框架 SimICT[30]. 在此框架上实现了 3 个功能组件:XBAR,VCore,MCU. XBAR 为基于 AMBA AXI 协议的 Crossbar 总线,实验中使用 XBAR 组件搭建三级总线互连结构. VCore 代表虚拟 Trace 核,因本实验主要研究访存性能,所以无需模拟完整的核内结构,可以降低模拟的复杂度. VCore 组件需要 2 类实例化:带 MACT 结构和不带 MACT 结构,用于对比评估. MCU 为存储控制单元,接收 MACT 收集发送来的访存数据包.

5.2 实验模型拓扑结构

本实验以 128 核的高通量众核结构作为实验模型,目标系统拓扑结构如图 11 所示. 每 8 个 TCG 连接到一个三级总线,每个二级总线挂载 4 个三级总线,4 个二级总线连接到一级总线,形成 128 核的多级总线互连结构. 4 个 MCU 分别挂载到 4 个二级总线之上.

5.3 MACT 模拟方案

当一个访存事件从 SPM 发送到 MACT 时:

1) 若访存类型为读数据事件,则清除 RMACT 中相关表项,并将表中 Bitmap 为 1 的地址的数据发送给总线.

2) 若访存类型为读地址事件或写地址事件,则需要将该地址插入到 RMACT 或 WMACT 中. 如果在现有表项中存在该地址的基址记录,则在此表项的 Bitmap 中将对应位置设置为 1 并记录 Msg_in,如果有重复置 1 的情况则同样设置记录多个 Msg_in,最后返回 ACK. 如果表项中不存在该地址的基址记录,若表项已满则向总线发送 NACK 消息,若不满则增加一项基址与该地址基址一致的一项,用当前周期加上固定延时(本实验设置为 16 cycles)作为此新项的 Deadline 并发送 ACK 消息.

3) 若访存类型为写数据事件,直接返回 ACK. MACT 在模拟中采用自驱动模型,每个周期都会给自身发送一个驱动事件,检查当前周期内 RMACT 和 WMACT 中 Deadline 满足的项,将其收集的请求一并发送到总线,并清空该项.

5.4 实验结果及分析

实验从 4 个方面对 MACT 的效果进行评估和分析,包括整体执行时间、单个内存消息平均延迟、

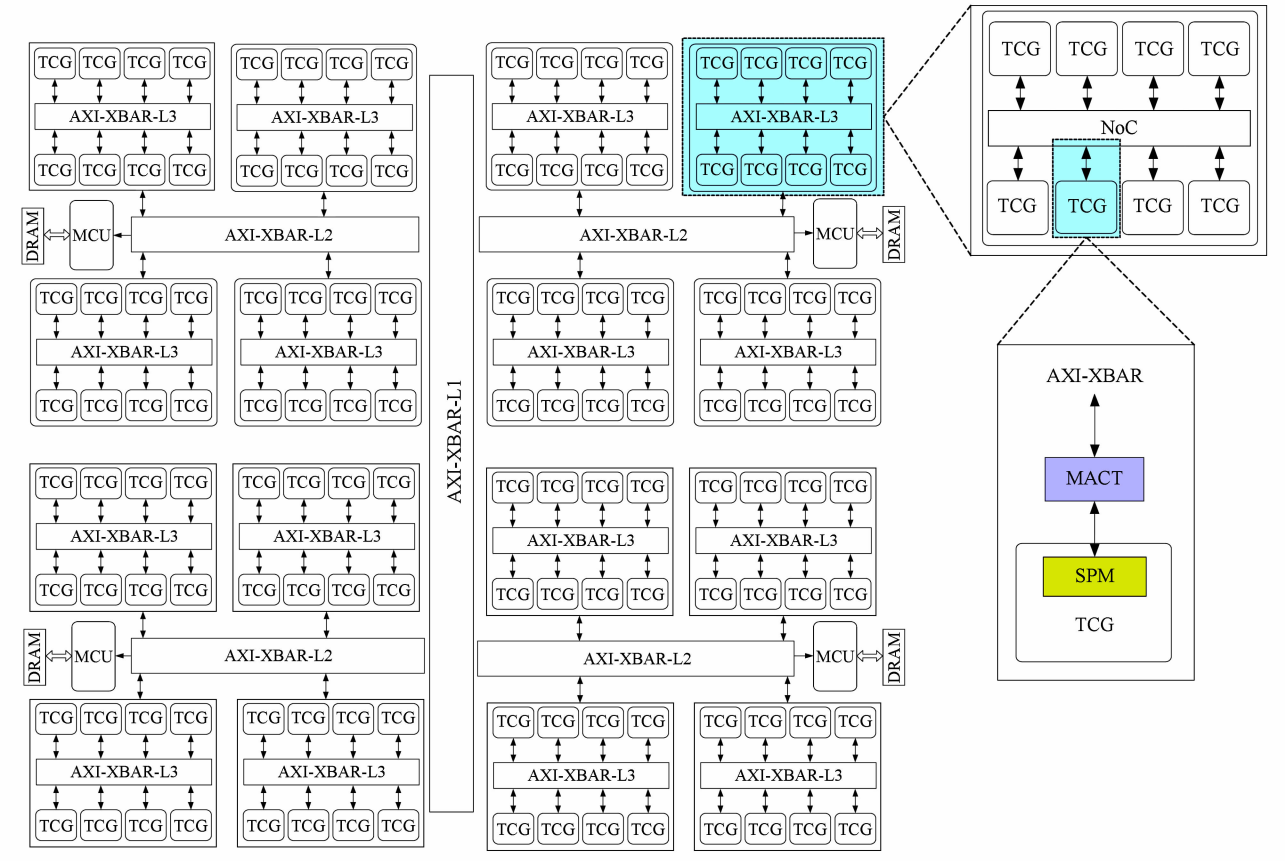


Fig. 11 128-core topology.

图 11 128 核拓扑结构图

访存带宽以及访存请求数量. 实验结果总体对比如图 12 所示,使用 MACT 结构的访存请求消息包数目减少约 49%,有效访存带宽提高约 24%,平均执行速度提高约 89%.

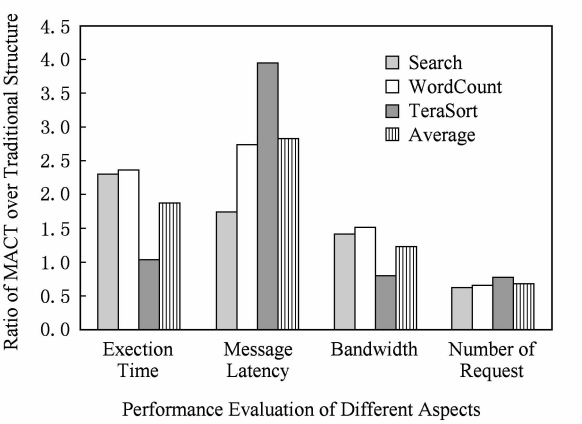


Fig. 12 Performance comparison between MACT and traditional structures.

图 12 MACT 结构相对于传统结构性能对比统计图

实验结果详细分析如下:

1) 应用程序整体执行时间下降(如图 13 所示). 增加 MACT 硬件机制后,Search,Wordcount,TeraSort 三个应用程序分别获得了 2.29,2.35,1.02 倍的加速比. 加速比的来源主要有 2 个方面:①传统内存控制器带宽利用率较低,通过 MACT 对访存请求的收集可以有效提高带宽的利用率,提高执行效率;②通过收集和批量处理,减少了发往网络中的访存消息包数量,降低了网络拥塞度. 其中,TeraSort 应用程序的加速效果不明显,加速比只有 1.02 左右,主要原因是其访存操作不够密集、吞吐量较低,导致 MACT 对访存请求的收集效果不明显,从而对执行时间的影响也非常有限.

2) 消息从访存部件到内存控制器的平均延迟上升(如图 14 所示). 由于 SPM 发出的访存消息统一在 MACT 中收集并等待一定 cycles 才被发送至片上网络,所以延迟会相应增大;但是由于处理器在发出访存请求时设置访存裕度值,并且 MACT 中采用时间窗口机制,所以仍能保证访存请求质量. 因为高通量应用虽然具一定的实时性但也并非越快越好,而是可以容忍一定程度的延迟. MACT 硬件机制的实现正是利用了高通量应用的这一特点来实现离散访存消息的收集和批量处理.

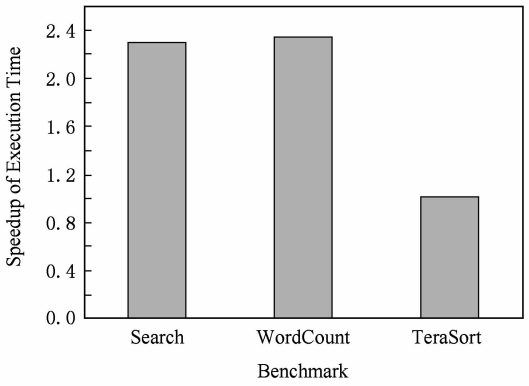


Fig. 13 Execution time comparison between traditional structure and MACT structure.

图 13 传统结构和 MACT 结构的执行时间比

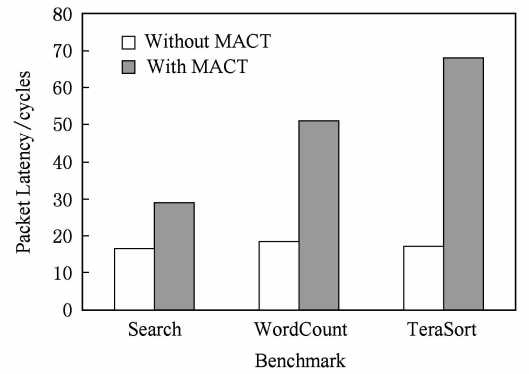


Fig. 14 Single message average delay comparison.

图 14 单个内存消息的平均延迟比较

3) 访存带宽比提高(如图 15 所示). 对于 Search 和 WordCount 应用来说,有效带宽均有不同程度的提高,分别为 1.42 和 1.51 倍;但对于 TeraSort 应用程序来说,虽然执行时间和访存请求消息包数目都有所降低,但是它们之间的比值即访存带宽比(个/cycle)反而降低了 21%,这主要是因为 TeraSort 应用程序访存密度低、访存压力小.

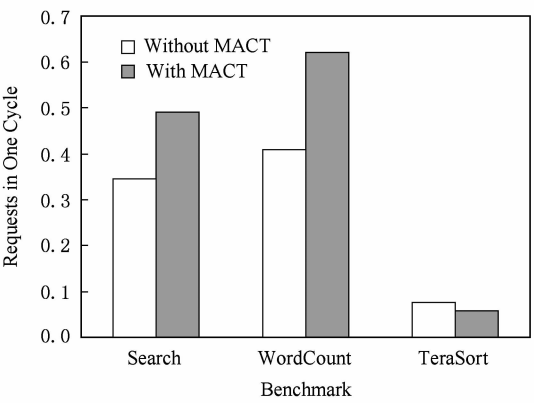


Fig. 15 Memory bandwidth comparison.

图 15 访存带宽比比较

4) 访存请求数量降低(如图 16 所示). 使用 MACT 硬件机制后,Search,Wordcount,TeraSort 三个应用程序分别获得了 1.62,1.58,1.25 倍的访存数量减少比. 这是因为很大一部分的离散访存请求被收集和批量处理,所以在数据通路上看到的效果是访存请求消息包数量有了明显的降低.

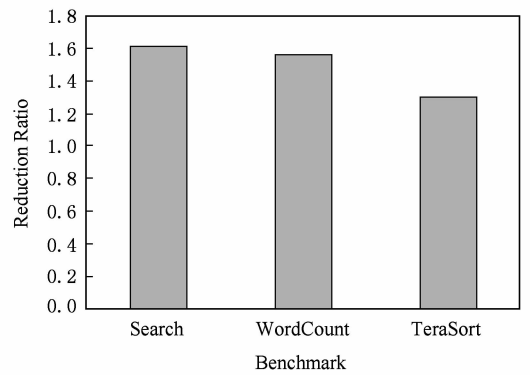


Fig. 16 Memory request number reduction ratio of traditional structure over MACT structure.

图 16 MACT 结构相对于传统结构的访存请求数量减少比

由于实验结果可知,在高通量众核处理器中,添加 MACT 硬件机制的访存通路可以通过对 SPM 发出的真实粒度的离散访存请求进行收集和批量处理,从而提高了有效带宽的利用率,减少了通路中访存请求消息包数量,降低了应用程序整体执行时间. 因此,在高通量众核处理器中,添加 MACT 硬件机制的访存通路相对于传统结构表现出了更强的性能.

5.5 硬件开销评估

在 MACT 硬件机制实现中,每个 TCG 需要添加一个 MACT 硬件结构来收集从 SPM 发出的访存请求. 由图 7 可知,每个 MACT 内部有 32×2 行存储结构,用来收集 32 KB 的 SPM 发出的访存请求. 其中每行大小为 Type(1 b),Tag(54 b),Vector(128 b)以及 Threshold(本文设置为 16 cycles,4 b)四者之和,即一行共 187 b,可用 24 B 存储结构实现,整个 MACT 即为 1536 B. 分检打包后的数据包大小需要存储在缓冲区中,其大小不会超过 MACT 中一行存储结构的大小,所以整个 MACT 的存储开销约为 1560 B. 因此,在每个 TCG 中 MACT 的硬件开销仅约为32 KB 的SPM 大小的 4.8%.

6 结论及未来工作

高通量众核处理器作为针对高吞吐、高并发、强

实时、低延迟等高通量应用特点的新型片上众核处理器逐渐成为研究热点. 片上核数的猛增以及高通量应用的访存密集型特点都进一步加重了"存储墙"问题. 针对此,本文提出了 MACT 硬件机制,用于收集从处理单元发出的离散访存请求进行批量处理,同时利用时间窗口机制保证任务的实时性. 实验结果表明,通过 MACT 对离散访存请求的收集和批量处理,访存请求消息包数目减少了约 49%,平均执行速度提高约 89%,访存带宽提高约 24%. 由于访存消息统一在 MACT 中收集并等待一定延迟才继续发送,所以延迟会相应增大,但总体执行时间得到了有效的缩减. 可见,MACT 硬件机制在新型高通量应用场景下对高通量众核处理器性能具有良好的提升作用,且硬件开销仅为片上 SPM 大小的 4.8%.

本研究的下一步工作主要集中在对 MACT 在更大规模片上众核处理器上进行结构优化和性能评估. 首先,对 MACT 硬件设计的面积和功耗进行评估和优化;其次,针对不同的网络拓扑结构,MACT 在拓扑中的位置和大小等参数也是影响 MACT 性能的重要因素,需要进一步优化和评估,使其在高通量众核处理器中发挥更大的作用.

参 考 文 献

[1] Netlib. LINPACK [EB/OL]. [2015-01-15]. http://www.netlib.org/linpack

[2] Li Guojie. Big data challenge for computer systems [J]. Communications of the CCF, 2013, 9(12): 33-35 (in Chinese)
(李国杰. 大数据对计算机系统的挑战[J]. 中国计算机学会通讯, 2013, 9(12): 33-35)

[3] 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)

[4] Ferdman Michael, Adileh Almutaz, Kocberber Onur, et al. Clearing the clouds: A study of emerging scale-out workloads on modern hardware [C] //Proc of the 17th Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS'12). New York: ACM, 2012: 37-48

[5] Zhan Jianfeng, Zhang Lixin, Sun Ninghui, et al. High volume throughput computing: Identifying and characterizing throughput oriented workloads in data centers [C] //Proc of Int Parallel and Distributed Processing Symp Workshops & PhD Forum (IPDPSW'12). New York: ACM, 2012: 1712-1721

[6] Han Yiding, Ancajas D M, Chakraborty K, et al. Exploring high-throughput computing paradigm for global routing [J]. Very Large Scale Integration Systems, 2013, 22(1): 155-167

[7] Murphy R. Memory: The center of the universe [C] //Proc of IEEE Workshop on Microelectronics and Electron Devices (WMED'13). Piscataway, NJ: IEEE, 2013: 1947-3834

[8] Intel. Product brief: The Intel Xeon Phi product family [EB/OL]. [2015-01-15]. http://www/intel.com/content/www/us/en/high-performance-computing/high-performance-xeon-phi-coprocessor-brief.html

[9] EZchip Semiconductor Ltd. 72-Core processor for intelligent networking and video processing [EB/OL]. [2015-01-15]. http://www.tilera.com/products/?ezchip=585&spage=618

[10] ITRS. 2013 ITRS summary files [EB/OL]. [2015-01-15]. http://www.itrs.net

[11] Zhan Jianfeng, Wang Lei, Sun Ninghui. Performance evaluation of high throughput computer [J]. Communications of the CCF, 2011, 7(7): 40-43 (in Chinese)
(詹剑锋, 王磊, 孙凝晖. 高通量计算机的性能评价[J]. 中国计算机学会通讯, 2011, 7(7): 40-43)

[12] Wang Lei, Zhan Jianfeng, Luo Chunjie, et al. BigDataBench: A big data benchmark suite from Internet services [C] //Proc of the 19th Int Symp on High Performance Computer Architecture (HPCA'14). Piscataway, NJ: IEEE, 2014: 488-499

[13] Chen Mingyu, Ruan Yuan, Huang Yongbing, et al. Studies of key technology on message-based memory system [J]. Journal of Network New Media, 2013, 2(1): 29-37 (in Chinese)
(陈明宇, 阮元, 黄永兵, 等. 基于消息的内存系统关键技术研究[J]. 网络新媒体技术, 2013, 2(1): 29-37)

[14] Chen Licheng, Lu Tianyue, Wang Yanan, et al. MIMS: Towards a message interface based memory system [J]. Journal of Computer Science and Technology, 2014, 29(2): 255-272

[15] Liu Sheng, Chen Shuming, Chen Haiyan, et al. Supporting efficient memory conflicts reduction using the DMA cache technique in vector DSPs [C] //Proc of the 6th IEEE Int Conf on Networking, Architecture and Storage (NAS'11). Piscataway, NJ: IEEE, 2011: 302-308

[16] Yuan George L, Bakhoda Ali, Aamodt Tor M. Complexity effective memory access scheduling for many-core accelerator architectures [C] //Proc of the 42nd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO'42). New York: IEEE/ACM, 2009: 12-16

[17] Sanchez D, Kozyrakis C. Vantage: Scalable and efficient fine-grain cache partitioning [C] //Proc of the 38th Annual Int Symp on Computer Architecture (ISCA'11). Piscataway, NJ: IEEE, 2011: 57-68

[18] Xie Yuejian, Loh G H. PIPP: Promotion/insertion pseudo-partitioning of multi-core shared caches [C] //Proc of the 36th Annual Int Symp on Computer Architecture (ISCA'09). New York: ACM, 2009: 174-183

[19] Stefanos Kaxiras, Hu Zhigang, Margaret Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power [C] //Proc of the 28th Int Symp on Computer Architecture (ISCA'01). Piscataway, NJ: IEEE, 2001: 240-251

[20] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1):107-113

[21] Talbot Justin, Yoo Richard M, Kozyrakis Christos. Phoenix++: Modular MapReduce for shared-memory systems [C] //Proc of the 2nd Int Workshop on MapReduce and Its Applications (MAPREDUCE'11). New York: ACM, 2011: 9-11

[22] Xapian project. An open source search engine library [EB/OL]. [2015-01-15]. http://xapian.org

[23] Luk C K, Cohn R, Muthetal R. Pin: Building customized program analysis tools with dynamic instrumentation [J]. ACM SIGPLAN Notices, 2005, 40(6): 190-200

[24] Montanaro J, Witek R T, Anne K, et al. A 160 MHz, 32 b, 0.5 W CMOS RISC microprocessor [J]. Solid-State Circuits, 1996, 31(11): 1703-1714

[25] Intel. Intel XScale Microarchitecture [R]. Santa Clara, CA: Intel, 2001

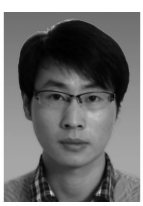
[26] Zhang Chuanjun, Vahid F, Najjar W. A highly-configurable cache architecture for embedded systems [C] //Proc of the 30th Annual Int Symp on Computer Architecture (ISCA'03). New York: ACM, 2003: 136-146

[27] Avissar O, Barua R, Stewart D. An optimal memory allocation scheme for scratch-pod based embedded systems [J]. ACM Trans on Embedded Systems, 2002, 1(1): 6-26

[28] Banakar R, Steinke S, Lee B S, et al. Scratchpad memory: A design alternative for cache on-chip memory in embedded system [C] //Proc of the 10th Int Symp on Hardware/Software Codesign (CODES'02). New York: ACM, 2002: 73-78

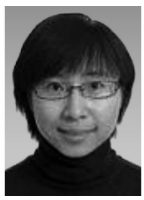
[29] Tan Guangming, Sreedhar C Vugranam, Gao Guang R. Just-In-Time locality and percolation for optimizing irregular applications on a manycore architecture [C] //Proc of the 21st Int Workshop Languages and Compilers for Parallel Computing (LCPC'08). Berlin: Springer, 2008: 331-342

[30] Ye Xiaochun, Fan Dongrui, Sun Ninghui, et al. SimICT: A fast and flexible framework for performance and power evaluation of large-scale architecture [C] //Proc of Int Symp on Low Power Electronics and Design (ISLPED'13). Piscataway, NJ: IEEE, 2013: 273-278

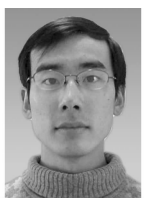


Li Wenming, born in 1988. PhD candidate. Student member of China Computer Federation. His main research interests include high throughput computing architecture and software simulation.

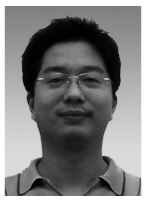
Ye Xiaochun, born in 1981. PhD, associate professor. Member of China Computer Federation. His main research interests include algorithm paralleling and optimizing, software simulation, and architecture for high-performance computer.



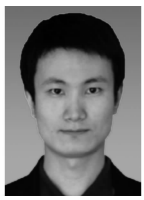
Wang Da, born in 1981. PhD, associate professor. Member of China Computer Federation. Her main research interests include many-core processor micro-architecture, IC testing and analysis, and very large scale integration(VLSI) design and testing.



Zheng Fang, born in 1984. PhD candidate, engineer. Member of China Computer Federation. His main research interests include high performance computing and processors architecture.



Li Hongliang, born in 1975. PhD, associate professor. Member of China Computer Federation. His main research interests include computer architecture and processors architecture.



Lin Han, born in 1990. PhD candidate at University of Science and Technology of China. His main research interests include many-core architecture and software simulation(linhan@ncic.ac.cn).

Fan Dongrui, born in 1979. PhD, professor, PhD supervisor. Senior member of China Computer Federation. His main research interests include many-core processor design, high throughput processor design and low power micro-architecture.

Sun Ninghui, born in 1968. PhD, professor and PhD supervisor. Fellow member of China Computer Federation. His main research interests include computer architecture, high performance computing and high throughput computing(snh@ict.ac.cn).