基于横向局部性的多核计算模型

袁 良[1,3] 张云泉[1,2]
(中国科学院软件研究所并行软件与计算科学实验室 北京 100190)[1]
(中国科学院计算机科学国家重点实验室 北京 100190)[2] (中国科学院研究生院 北京 100049)[3]

摘 要 片内多核已成为延长摩尔定律的方式,并行算法设计、编程模型、编译器和运行时系统都需要利用计算模型进行分析。现有多核模型对线程间共享缓存等资源的竞争已有较精确的模型,但是对于线程间数据共享考虑较少。提出线程间共享缓存的横向局部性和任务共享率概念,基于此扩展串行存储层次模型 RAM(h),提出考虑任务共享率的多核并行计算模型 MRAM(h)。

关键词 多核,并行计算模型,共享缓存,横向局部性
中图法分类号 TP317 **文献标识码** A

Multi-core Parallel Computational Model Based on Horizontal Locality

YUAN Liang[1,3] ZHANG Yun-quan[1,2]
(Lab. of Parallel Software and Computational Science, Institute of Software, CAS, Beijing 100190, China)[1]
(State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[2]
(Graduate University, Chinese Academy of Sciences, Beijing 100049, China)[3]

Abstract Almost all modern CPUs are multi-cores with shared cache on chip. A number of models have been proposed for predicting the shared cache contention, but few of them consider the influence of shared cache sharing. This paper proposed horizontal locality and task shared ratio, and then proposed a parallel computational model for multi-core architecture, which can be used by parallel algorithm, compiler, parallel programming model and runtime system.

Keywords Multi-core, Parallel computational model, Shared cache, Horizontal locality

1 引言

CPU运行速度和存储器带宽以及延迟之间的巨大差异一直是影响算法性能的关键因素。利用算法局部性原理,在CPU与存储器之间加入Cache进行数据缓冲,降低数据访问延迟,提高访问带宽,是一种改善存储墙问题的有效技术。同时,人们将算法局部性分为时间局部性和空间局部性,以指导算法改进,获得更好的局部性,并提出了大量的优化局部性的算法和不同的计算模型。

此外,由于功耗墙等原因,过去几十年依靠时钟频率提升CPU运行速度的模式不再可行,多核是维持摩尔定律的唯一方式。多核出现之前,通过SMP、MPP和集群方式并行计算的计算单元即CPU之间无论是使用MPI编写SPMD程序,还是通过PGAS进行统一地址空间编程,都只是通过内存或者网络单一层进行通信和数据共享,只在网络或者存储器一层考虑数据共享和通信的优化,而多核引入的片内多级共享Cache,使得计算单元即核之间存在多级共享存储,可以在不同存储层次进行通信和数据共享。

并行计算模型提供了并行算法的设计基础,是对不同并行体系结构模型的抽象。现有并行计算模型可以分为3类,以PRAM为代表的第一代模型主要面向共享存储结构,以BSP、LogP为代表的第二代模型主要面向分布式存储结构,随着存储层次、通信层次的增减,第三代计算模型主要对已有模型进行层次化扩展,包括基于BSP的Multi-BSP,基于LogP的扩展模型Memory LogP和Log_nP等,基于RAM和PRAM的扩展模型HMM、BT、UMM和RAM(h)等。由于多核的多层共享存储结构,$MLog_nP$等进行了横向扩展。

本文考虑了多核引入的复杂的存储层次以及多层次共享存储,扩展了面向纵向存储层次的时空局部性,提出多核上横向局部性的概念,用来描述多核之间在多存储层次共享上的局部性,在多核上扩展了经典的局部性概念,指出在3C模型中加入横向局部性的分析,可以计算该算法在多核上的Cache缺失率,以改进多核上并行算法的局部性,提高算法性能。基于横向局部性和任务共享率,对RAM(h)模型进行了扩展,提出了多核上的并行计算模型MRAM(h)。

基于MRAM(h)模型,提出了任务间共享率概念,用来指导算法任务到核的映射。针对不同的映射方法,在MRAM(h)模型下对经典分块矩阵乘进行了分析。测试结果表明了

到稿日期:2011-10-20 返修日期:2011-12-25 本文受863计划(2006AA01A125,2009AA01A129,2009AA01A134),核高基(2009ZX01036-001-002),中国科学院知识创新工程重大项目课题(KGCX1-YW-13),财政部国家重大科研装备研制项目(ZDYZ2008-2)资助。
袁 良(1984—),男,博士生,主要研究领域为并行算法、并行编程模型,E-mail:ilpiny@gmail.com;张云泉(1973—),男,博士,研究员,博士生导师,主要研究领域为高性能计算及并行数值软件、并行计算模型、并行数据库、海量数据并行处理,E-mail:zyq@mail.rdcps.ac.cn(通信作者)。

MRAM(h)模型的有效性，并在 MRAM(h)模型下对经典分块矩阵乘算法进行了改进。

2 相关工作

并行计算模型提供了并行算法的设计基础，是对不同并行体系结构模型的抽象。Zhang 等[16]进行了综述，将计算模型分为面向共享存储、面向通信以及面向存储层次 3 类。现有并行计算模型可以分为 3 类，以 PRAM[1]为代表的第一代模型主要面向共享存储结构，包括若干基于 PRAM 的扩展模型，例如引入分块的 BPRAM[19]、加入异步的 APRAM 和 Phase PRAM[20]、融合流处理的 Stream PRAM[23]等。

第二代模型主要面向分布式存储结构，主要包括 BSP[2]、LogP[6,9]以及扩展模型。LogP 模型包括延迟、带宽、处理器开销以及处理器个数 4 个参数对网络通信建模，LogGP[5]在此基础上增减了长消息带宽，LogGPS[8]考虑了不同通信协议的同步开销，LoPC[7]和 LogPC[10]加入了网络竞争模型，LogGPO[11]引入了计算与通信重叠因子。

第三代计算模型主要通过对已有模型的层次化扩展，对存储层次、通信层次进行建模。非一致访存开销的 HMM 模型，对访问地址为 a 的存储单元，假定其开销为 $f(a)$，其中的 $f(a)$是地址 a 的单调上升函数，以此来计算算法的访存开销。而 BT(Block Transfer)模型又增加了对块传递的改进。这两个模型都根据所访问的数据地址来决定相应的开销，不适用于随机访问存储系统，即访问开销主要由其所在的存储层次数而不是访存地址决定，每一层都有其相对固定的访问开销。UMH 模型假定其存储访问开销为存储层次数 k 的函数 $f(k)$，但 UMH 模型忽略了存储访问中的时间和空间局部性。基于 LogP 的扩展模型 Memory LogP[4]和 Log_nP[3]等，RAM(h)[14]将 RAM 模型中的单层访存扩展为多层访存，并引入分块描述空间局部性。

此外，针对多核处理器，近年来也设计了新的模型。Savage[22]设计了 UMM，分析了矩阵乘等问题下界。Valiant[24]基于 BSP 设计了多核上模型 Multi-BSP，由于多核的多层共享存储结构，Tu 等[17]基于 $\text{Log}_n\text{P}/\text{Log}_3\text{P}$ 模型，提出了考虑横向通信层次性的 $\text{MLog}_n\text{P}/\text{MLog}_3\text{P}$ 模型，然而该模型只对通信参数进行层次化分析，提高了在多核处理器集群上通信性能预测的准确性，并没有考虑任务间共享。Wu 等[18]将 Cache 抖动分为层间抖动和核间抖动，核间抖动主要指多核处理器私有 Cache 写回失效一致性协议下由于数据共享造成的 Cache 在多个核之间的抖动，提出了能有效降低抖动的软硬件合作管理的 Cache。

针对串行程序局部性，Ding 和 Kennedy[27]提出了数组重组启发式方法，其将总是同时被访问的数组合并存储。Zhong 等从访存 trace 中获得数据访问亲缘性对数组重组或者切分。针对并行程序共享缓存的优化，Jiang[26]等证明在大于双核共享缓存系统上，根据程序间冲突图进行程序级调度是 NPC 问题，并提出了一种启发式层次调度方法。Zhang 等通过实验证明了考虑共享缓存的算法能获得较高的加速比。Tam 等提出了基于 PMU 的动态线程调度方法，即利用 PMU 获得 CPU 间通信量，并记录每个出现的访存模式，利用启发式方法重新调度线程。

3 基于横向局部性的计算模型

本节首先介绍对 RAM 模型进行纵向局部性扩展的 RAM(h)模型(见图 1)，然后提出基于多核共享 Cache 的横向局部性概念，用来描述多核 CPU 上的线程间数据共享，并考虑 NUMA 体系结构带来的线程访问远程数据局部性，针对算法提出了任务共享率的概念，利用横向局部性扩展 RAM(h)模型，提出多核上并行计算模型 MRAM(h)(见图 2)，用来指导任务在核间分配，最后针对不同的分配方法，分析了分块矩阵乘算法在 MRAM(h)模型下的结果。

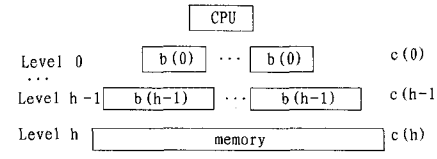

图 1 RAM(h)模型

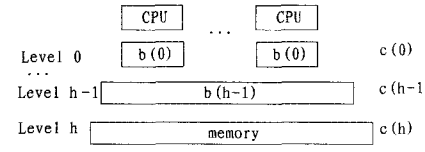

图 2 MRAM(h)模型

3.1 RAM(h)模型

我们用矩阵乘和简单存储层次说明了数据局部性原理，矩阵乘 $X=Y\times Z$ 的时间复杂性为 $O(n^3)$，存储矩阵的空间复杂性为 $O(n^2)$，每个 XYZ 矩阵中的元素需要访问 $O(n)$次，对基于三重循环的基本矩阵乘算法，6 种具有相同时间和空间复杂度的算法实现在存储访问上有巨大差距[14]，这是由于访问数据所在的存储层次不同，这也是影响算法实现性能的主要因素之一，亦即文献[15]中引出的存储复杂性概念也是建立 RAM(h)模型的基本出发点。

RAM(h)模型建立在 RAM 模型之上，在存储开销方面进行了扩展，将 RAM 模型中只有一层单位时间开销的存储扩展为多层，离处理器越远其容量越大，存储访问速度越慢；反之离处理器越近其容量越小，存储访问速度越快。每次存储访问的开销由数据所驻留的离处理器最近的层次数亦即算法数据重用的能力决定。数据所在的层次越低，重用越好，存储访问越连续，开销就越小。数据在 RAM(h)模型中以块为单位进行传递，对某一分块中任一数据单元的访问将导致该数据所在的整个数据块被从最远端取到离处理器最近的层次。提出该模型的目的之一是鼓励用户对数据在不同存储层次间的移动进行优化，并尽量减少数据移动开销在整个计算中所占的比重。

为了对不同的算法或算法的不同实现进行定量度量，文献[14,15]中提出了存储访问复杂性的概念并具体给出了存储访问复杂性的度量指标：存储访问复杂度。算法的复杂性包括计算复杂性和存储访问复杂性，其中计算复杂性包含传统的时间复杂性和空间复杂性，是一个算法的静态属性，不会随着算法的不同实现形式有太大变化；而存储访问复杂性是一个算法的动态属性，会随着算法实现形式和实现平台的不同而出现较大变化。定义算法的存储访问复杂性是并行或串行算法的数据分布和数据访问模式与程序所运行的计算平台

存储层次之间交互所形成的算法所访问的数据的分布状态以及由此决定的算法访存开销。

存储访问复杂性分析是对算法中占主导地位的存储访问模式特征的提取。算法的存储访问复杂度 $m(n)$ 定义为算法实现在不同存储访问次数的加权和，权重即为存储层次的访问开销。数据在不同存储层次之间的总移动次数与总浮点操作次数的比 $q(n)$ 为总访存次数与计算复杂度之比，因此 $m(n)$ 和 $q(n)$ 的乘积 T_{fp} 表示算法的每次浮点操作中花在数据移动上的时间，即 $T_{fp}=m(n)*q(n)$（也叫做规范化存储访问复杂度）。

3.2 横向局部性

本文将局部性分为纵向局部性和横向局部性，经典的串行程序局部性称为纵向局部性，包括时间局部性和空间局部性；由多核上复杂的共享存储层次带来的线程间共享 Cache，甚至是多级共享 Cache，带来了线程间在 Cache 一层上的读共享和读写冲突问题，由此引出线程间局部性（横向局部性）。读共享是指不同核上的线程在同一存储层次访问相同数据。读写冲突是指同一个 Cache 上读写不同数据带来的冲突缺失和容量缺失。

横向局部性是描述读共享和读写冲突的一种局部性。读共享摊销了访问高一层存储的延迟和带宽，提高了算法性能。也可分为时间局部性和空间局部性。时间局部性指一个线程访问某数据并将该数据缓存到共享 Cache，在此数据被置换出共享 Cache 前，另一个线程在该层共享 Cache 中访问该数据。空间局部性是指一个线程访问某个数据，在此数据被置换出共享 Cache 前，另一个线程在该层共享 Cache 上访问该数据所在缓存行中的其他数据。

在纵向局部性概念中，3C（Compulsory，Capacity，Conflict）模型是一种常用的分析方法，可以用来分析算法的 Cache 缺失率，以指导设计具有更好纵向局部性的算法。针对横向局部性，也可以利用 3C 模型在多核上进行分析，但线程间的数据共享，会降低 Compulsory 和 Capacity 缺失率，线程间的读写冲突可能会增加 Capacity 和 Conflict 缺失率。基于类似方法，本文利用 MRAM(h) 分析算法的访存复杂度。

目前 CPU 已经集成了内存控制器和相应的 CPU 间通信连接（例如 AMD 的 HyperTransfer 技术和 Intel 的 QPI 技术），多路 CPU 系统是一种 NUMA 体系结构，例如，本文实验平台之一为两路四核 Intel Xeon X5550，在一定程度上缓解了 UMA 系统内存带宽限制，提高了系统的可扩展性。但是不同的 CPU 访问同一段内存物理地址空间的延迟和带宽不同，针对这种情况，需要考虑跨 CPU 的横向局部性，第 4 节实验结果表明，跨 CPU 两线程共享数据时，也能提高远程线程访问的性能。

3.3 任务共享率

基于分布式存储的并行系统的各结点任务之间通信是通过显式消息传递来完成的，这样可以通过任务之间的通信率，结合实际的并行系统的互联模式，进行任务分配，使得通信代价达到最低，MPIPP 是这类工作的代表。而多核处理器或者 SMP 由于是基于共享内存，没有显式消息通信机制，也没有显式的通信率的概念，并且通信率只是单一层次的概念，而共享率可以在不同的存储层次进行共享，因此不能类似分布式存储的系统，进行任务间的有效分配和映射。由于复杂的存储层次所带来的性能的影响，一个算法又必须要考虑任务间、数据划分间的通信率，然后结合存储层次进行任务分配，故引入多核上任务共享率的概念，将通信理解为共享，通信率作为共享率，为多核上的任务分配提供依据。

通过任务与数据访问的关系，可以计算任务间共享率，本文只对规则程序以简单的分块矩阵乘为例加以说明，但是对于非规则计算，首先通过任务访问数据的位置，得到任务与数据之间的矩阵（Task-Data Access，TD），类似方法在文献[28]中用到，但是其工作只是关注如何合并任务，提高程序纵向局部性，而我们利用 TD 矩阵计算任务间数据共享率矩阵（Task-Task Share，TT），关注如何提高任务间横向局部性，通过 TD 矩阵乘其转置，可以得到任务间共享率 $TT=TD\times TD'$。考虑矩阵乘法 $C=A\times B$，采取分块矩阵算法，如图 4 所示，对 C 进行划分，计算任务分为 4 个小矩阵：C_{11}，C_{12}，C_{21}，C_{22}。$C_{11}=A_{11}\times B_{11}+A_{12}\times B_{21}$，以此类推。可以得到算法任务数据访问矩阵，如图 3 所示，1 表示访问，0 表示不访问，每个子任务都需要两个矩阵 A 和 B 中的子块，并且两个任务共享 A_{11} 和 A_{12} 两个块，两个的共享率为 2，图 4 列出了 4 个子任务之间的共享率。如果对共享率归一化，则 C_{11} 和 C_{12} 要取 8 个块，有两个重用，故归一化之后的共享率是 1/4。多核或者 SMP 上的任务共享率就是任务之间共享的数据量的大小。从任务共享率的角度进行任务分配，即处理器映射。分配到核，因为不同的核之间的通信带宽和延迟是不一样的，不同的映射方式也就带来不同的性能。

	A_{11}	A_{12}	A_{21}	A_{22}	B_{11}	B_{12}	B_{21}	B_{22}
C_{11}	1	1	0	0	1	0	1	0
C_{12}	1	1	0	0	0	1	0	1
C_{21}	0	0	1	1	1	0	1	0
C_{22}	0	0	1	1	0	1	0	1

图 3 分块矩阵乘算法任务数据 TD 矩阵

	C_{12}	C_{21}	C_{22}
C_{11}	2	2	0
C_{12}	—	0	2
C_{21}	—	—	2

图 4 分块矩阵乘法任务间共享率 TT 矩阵

3.4 MRAM(h) 模型

RAM(h) 模型没有共享存储层次概念，是单核上基于纵向局部性的计算模型。而多核的出现使得算法在不同存储层次上可以共享数据，产生了横向局部性。通过利用数据共享，获得较好的横向局部性，算法可以进一步降低存储复杂度，MRAM(h) 就是在多核上考虑共享存储层次的 RAM(h) 的扩展，融和了纵向局部性和横向局部性的概念。

RAM(h) 模型的关键思想是算法实现与算法分离，算法应用 RAM(h) 模型计算访存时间，第一步确定数据在哪层复用，第二步计算数据在该层复用的次数，根据这两步，结合复用的基本方式即连续的或非连续的，得到一个模型计算结果，即为 RAM(h) 下的访存复杂度。RAM(h) 模型有两种基本复用方式：连续和非连续。读长度为 l 的连续数据的开销：$[1/b(h-1)]c(h)+[l/b(h-2)]c(h-1)+\cdots+[1/b(0)]c(1)+l\times c(0)$，连续读数据的开销可以通过非连续数据的开销来计算。为了准确起见，用实验测定连续的存储访问参数，而不是从非连续的访问参数计算得到，一般的连续数据访问，其平均开销为：$t_{1m}=t_m/l_2+t_{L2}(l_2-l_1)/l_1l_2+t_{L1}(l_1-1)/l_1$，$t_{1L2}=$

$t_{L2}/l_1+t_{L1}(l_1-1)/l_1$(这两个式子是读长度为 l 的连续数据的开销计算公式的细化,剔除了重叠计算部分)。

除了上述 RAM(h)的两个关键思想,在多核上的扩展模型 MRAM(h)还具有两个要求,即横向局部性的要求,这也是建立 MRAM(h)的出发点和关键思想:数据共享较多的任务要分配在具有较高共享存储层次的核之间,以利用任务间的数据局部性;任务的数据访问时间相近,防止共享数据所在的 Cache Line 被替换。

对于共享的数据的访问时间,MRAM(h)的计算方法是进行均摊,因为即使数据都是由一个核的访问而将数据读入 Cache,也可以采取优化方法,使得数据访问时间可以均摊计算。例如矩阵在 L2 Cache 一层进行数据共享,则访问内存的复杂度就可以进行均摊,下节中对分块矩阵乘算法进行分析就用到了该原则。所以,通过在 MRAM(h)模型下分析算法,改进算法,摊销了较低存储层次的访存延迟和带宽。不同核通过对共享数据进行不同部分的访问,可以降低数据在存储器 bank 访问的冲突,还可以摊销访问延迟,降低存储器带宽需求。例如,在第 5 节中,两个核分别从数组的两端对数组进行访问,可以有效地摊销延迟。线程间在更高存储层次共享数据,减低多核需求的空间大小,使得可以利用更大分块。

例如单核矩阵乘算法中 B 的一个子块的存储复杂度为 $B^2(t_{bm}+(N-1)t_{L2})$,其中 $(N-1)t_{L2}$ 为 RAM(h)模型下的数据重用,而如果子任务在 L2 Cache 进行共享,则 t_{bm} 就为 MRAM(h)模型下的数据共享,使用前面所提出的 MRAM(h)模型优化方法,就可以均摊数据共享部分的存储复杂度。

4 实验结果

本节使用类似 STREAM 中的 SUM 函数来测试横向局部性,并用分块矩阵乘来测试。本节使用两个测试平台,表 1 列出两个测试平台参数,平台 1 为两路四核 Xeon X5472 处理器,通过 FSB 共享 16GB 内存,测试平台 2 为两路四核 Xeon X5550 处理器,两个 CPU 分表连接 8GB 内存,两个平台 Linux 版本均为 2.6.35-generic,编译器均为 gcc4.4.5。对两个平台使用 Calibrator 测试延迟,平台 1 共享缓存延迟为 19.52ns,98.74ns,平台 2 为 22.75ns,47.83ns。用 STREAM 测试带宽,平台 1 单核访存最大带宽为 3.5GB/s,系统最大内存带宽为7.6GB/s,平台 2 分别为 10.6GB/s 和 22.8GB/s。

表 1 测试平台

	Xeon X5472 Harpertown	Xeon X5550 Nehalem
Number of Cores	8 cores(2 sockets,FSB)	8 cores (2 sockets, NUMA)
Clock Frequency	3.0G	2.67G
L1	32kB, 8-way, 64B/line, 4cycle	32kB, 8-way, 64B/line, 4cycle
L2	6MB, 24-way, 64B/line, 15cycle	256kB, 8-way, 64B/line, 10cycle
L3	—	8MB, 16-way, 64B/line, 38cycle
Memory	16G,319cycle	16G,87cycle

4.1 横向局部性

文献[25]用类似本文横向局部性概念优化 PARSEC 程序,获得了较高性能加速,本文不只面向循环级优化,也适用于任务级数据共享,本节对横向局部性进行测试,以表明最优情况下充分利用横向局部性的加速比,并就 NUMA 内存结构对横向局部性的影响也进行测试和分析。

本文重点关注线程间共享数据带来的横向局部性效果,所以不采用测试延迟和带宽的传统方法,例如利用指针链接方法测试延迟,用汇编代码编写只加载数据的方法测试带宽,本文只通过已有测试程序获得单线程带宽和延迟,而采用更为直接和简单的循环遍历来测试横向局部性在最好情况下的性能提升效果。在一次实验中,测试程序需要遍历大小为 n 的数组 r 次,进行求和操作,将任务映射到 p 个核 $(c_1,\cdots,c_p)$,根据这 p 个核对数组是进行分块遍历(即切分数组 n 为 p 块,每个核遍历 r 次)还是共享遍历(每个核分别遍历整个数组 r/p 次),分别记为 $p(c_1,\cdots,c_p)$ 和 $s(c_1,\cdots,c_p)$。

图 5 和图 6 是在两个平台上的实验结果,横轴为不同核数以及线程到核的映射方法,纵轴加速比以单线程访问本地内存(对平台 1 均为本地内存)的实测时间为基准。平台 1 实验结果如图 5 所示,左边 3 组为两核并行执行,采用 3 种映射方法,$p/s(0,1)$为两个共享 L2 cache 的核,此时算法采用共享与分块的加速比分别为 1.94 和 1.12,说明共享缓存在带宽受限的情况下能提高近一倍性能,证明了 MRAM(h)均摊开销的合理性;$p/s(0,2)$为两个核在一个 CPU 内,但是不共享缓存,所有共享访问没有明显优势;$p/s(0,4)$为不同 CPU 中有两个核,此时分块算法由于能更为充分地利用带宽,减少竞争,因此性能反而更优。中间 3 组为 4 核情况,也有 3 种映射方法,$s/p(0,1,2,3)$为同一 CPU 内有 4 核,分块访问由于带宽受限,4 核并行类似双核并行,无加速比,而共享访问由于存在 2 个共享 L2 Cache,达到 2.56 加速比;$s/p(0,1,4,5)$为两个 CPU 内各取共享缓存的两个核,增加了内存带宽,达到 2.17 加速比,而共享访问情况为 3.45。右边所有 8 个核并行执行,依然由于带宽受限,分块访存加速比为 2.36,而共享情况可达 5.05 加速比。

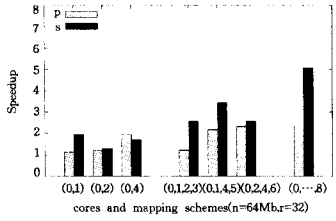

图 5 平台 1 上共享 LLC 横向局部性

图 6 为平台 2 上测试结果,由于平台 2 为 NUMA 结构,所有测试结果均在 core0 分配数组的情况下进行测试,即 core0 到 core3 与数组具有亲缘性,而 core4 到 core7 通过 QPI 访问数组。由于单 CPU 内只有一个共享缓存,4 个核对称,因此在图中用 CPU 内核数来表示映射方法,而忽略具体核编号,(a,b)表示有 a 个线程与数组有亲缘性,b 个线程没有亲缘性。图中左边 3 组为两个核情况,此时不映射的核是否共享缓存以及是否采用共享遍历方法,都能获得近两倍的加速比,说明在两核顺序遍历的情况下,带宽不是瓶颈;对于中间 2 组情况,即 3 个线程和 4 个线程情况,由于带宽依然不是瓶颈,可以看到不同映射情况下共享访问的加速比相近,而不共享情况随着访存亲缘性降低,性能下降。对于最右边的 8 核均用满情况,可以看到由于带宽成为瓶颈,不共享访存加速比不能提高,而共享情况可以达到将近 8 倍加速比。

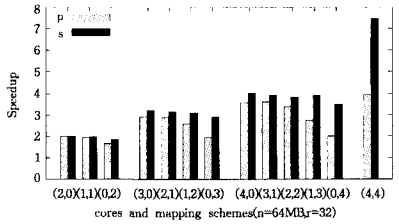

图 6　平台 2 上共享 LLC 横向局部性

4.2　矩阵乘

本节采用具有较好纵向局部性的串行 ikj 形式分块算法，通过设置 LDX，LDY，LDZ 为 $n+1$ 减少冲突缺失，分析考虑容量缺失，研究 MRAM(h)模型的有效性，图 7 为 *ikj* 形式分块矩阵乘算法代码。对该算法在 RAM(h)模型下进行的分析可知，计算每个 Z 矩阵中元素要 N 次乘法，而一次循环只计算 B 次乘法，故要取 X 矩阵整体 N/B 次。Z 矩阵有 N/B 个列块，故 Z 取 N^2/B^2 次列块。X 矩阵与 Z 矩阵的块访问模式相同。X 矩阵每个列块的访存开销：一个列块有 NB 个元素，每个重用 B 次，一次从主存读，$B-1$ 次从寄存器读，不计入时间，即为 NBt_{1m}。Z 矩阵每个列块中每个元素的访存写开销为 Bt_{L1}，读开销为 $t_{1m}+(B-1)t_{L1}$，即 $NB(t_{1m}+(B-1)t_{L1}+Bt_{L1})$。$Y$ 矩阵每个元素使用 N 次，故整体取一次，共有 N^2/B^2 个小块，Y 是非连续访问，B^2 个元素，每个元素重用 N 次，即为 $B^2(t_{1m}+(N-1)t_{1L2})$。表的第一行列出了单核上 ikj 形式分块矩阵乘算法的访存复杂度。

```
for(int kk=0; kk<n; kk+=b)
for(int jj=0; jj<n; jj+=b)
  for(int i=0; i<n; i++)
    for(int k=kk; k<kk+b-1; k++)
      for(int j=jj; j<jj+b-1; j++)
        Zij += Xik * Ykj;
```

图 7　分块矩阵乘 ikj 形式算法代码

由于 Z 矩阵要写入数据，因此应避免共享，对 X 和 Y 矩阵在 Memory 和 L2 Cache 层次进行数据重用。对矩阵 Z 进行划分，图 8－图 10 将整个任务 Z 分为计算 4 个小矩阵，Z_{11}，Z_{12}，Z_{21}，Z_{22}。Z_{11} 和 Z_{12} 都需要 4 个矩阵 X 和 Y 中的小块，并且两个任务共享 X_{11} 和 X_{12} 两个块，两个的共享率为 2。图 4 列出了 4 个子任务之间的共享率。如果对共享率归一化，则 Z_{11} 和 Z_{12} 要取 8 个块，有两个重用，故归一化之后的共享率是 2/8。

将 Z_{11}，Z_{12}，Z_{21}，Z_{22} 看成 4 个矩阵乘法，采用 ikj 形式的分块矩阵乘，共有 3 种任务分配方法，如图 8－图 10 所示，由于 3 种方式中两个 CPU 内的访存模式相同，因此只考虑 Z_{11} 所在的 CPU。表 2 列出了顺序以及 3 种情况在 MRAM(h)下的分析结果。

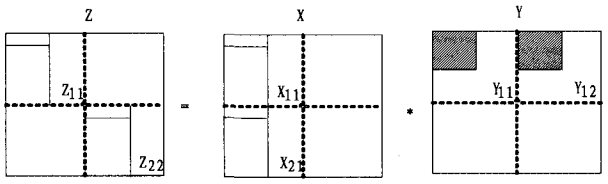

图 8　Z_{11} 和 Z_{22}，无共享数据

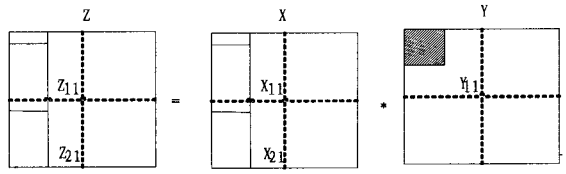

图 9　Z_{11} 和 Z_{21}，共享 Y 矩阵子块。

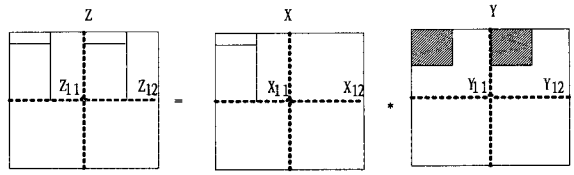

图 10　Z_{11} 和 Z_{12}，共享 X 矩阵子块

表 2　单核及多核上分块矩阵乘算法访存复杂度

Blocked MMM	Z	X	Y
Sequential	$\frac{N^2}{B^2}\times NB\{t_{1m}+(B-1)t_{L1}+Bt_{L1}\}$	$\frac{N^2}{B^2}NBt_{1m}$	$\frac{N^2}{B^2}\times B^2\{t_{1m}+(N-1)t_{1L2}\}$
Parallel, Scheme1	$2\frac{N^2}{4B^2}\times\frac{N}{2}B\{t_{1m}+(B-1)t_{L1}+Bt_{L1}\}$	$2\frac{N^2}{4B^2}\frac{N}{2}Bt_{1m}$	$2\frac{N^2}{4B^2}\times B^2\{t_{1m}+(\frac{N}{2}-1)t_{1L2}\}$
Parallel, Scheme2	$2\frac{N^2}{4B^2}\times\frac{N}{2}B\{t_{1m}+(B-1)t_{L1}+Bt_{L1}\}$	$\frac{1}{2}2\frac{N^2}{4B^2}\frac{N}{2}Bt_{1m}$	$2\frac{N^2}{4B^2}\times B^2\{t_{1m}+(\frac{N}{2}-1)t_{1L2}\}$
Parallel, Scheme3	$2\frac{N^2}{4B^2}\times\frac{N}{2}B\{t_{1m}+(B-1)t_{L1}+Bt_{L1}\}$	$2\frac{N^2}{4B^2}\frac{N}{2}Bt_{1m}$	$2\frac{N^2}{4B^2}\times B^2\{\frac{1}{2}t_{1m}+(\frac{N}{2}-1)t_{1L2}\}$

表 3 是 3 种映射方法在两个平台上的实验结果，数据均为实测运行时间。3 个矩阵大小均为 8192 * 8192。对于 4 个线程情况，采用图 8－图 10 中的映射方法，从结果可以看出共享 Y 子块时在 1024 分块和 2048 分块情况下均有一定加速比。对于 8 线程情况，对 Z 矩阵进行 2 * 4 和 4 * 2 两种情况划分，两种情况均有 3 种对应图 8－图 10 的映射方法，其共享率依次递减。从实验结果可以看到，充分考虑任务共享率的映射方法 1 比映射方法 3 有较高性能提升。

表 3　分块矩阵乘测试结果

		平台 1		平台 2	
	BlockSize	1024	2048	1024	2048
2 * 2	Scheme1	103.1	208.8	130.3	125.2
	Scheme2	334.7	392.5	139.6	131.9
	Scheme3	334.7	392.3	139.5	131.9
2 * 4	Scheme1	52.1	163.4	112.2	95.7
	Scheme2	237.9	426.0	147.5	131.2
	Scheme3	247.4	428.6	143.7	126.0
4 * 2	Scheme1	52.6	141.2	108.6	86.5
	Scheme2	233.4	276.7	147.4	99.0
	Scheme3	233.4	273.8	108.9	98.1

结束语　扩展 RAM 中单层存储层次为多层的 RAM(h)模型，其存储复杂度只描述了存储层次间的数据重用，并没有考虑多核共享缓存所带来的线程间共享存储的数据共享。本文在多核上对 RAM(h)进行横向扩展，同时考虑数据重用和数据共享。RAM(h)和 MRAM(h)模型的主要目的都是在实现算法时要充分考虑数据所在存储层次，尽量考虑数据重用，以提高算法的性能，MRAM(h)还要求考虑并行计算任务之间的数据共享率，尽量使得共享数据较多的任务分配得更近，也即在更高的存储层次共享数据，此外，还要保证共享数据的

访问时间相离较近。

RAM(h)和MRAM(h)都是考虑访存带宽和延迟受限而不是计算能力受限的算法。由于MRAM(h)基于多核，在访问时间 $c(h)$ 上，其采取最大带宽时的延迟，线程间带宽争夺带来延迟变化的问题是下一步的研究工作。本文只定性对几种线程间共享方式进行分析，没有与实测结果进行比较，详细的分析将是下一步的研究工作。

参 考 文 献

[1] Fortune S, Wyllie J. Parallelism in random access machines[C]// Proceedings of the Tenth Annual ACM Symposium on Theory of Computing(STOC'78). ACM, 1978: 114-118

[2] Valiant L. A bridging model for parallel computation[J]. Communication of ACM, 1990, 33: 103-111

[3] Cameron K W, Ge R, et al. lognP and log3P: Accurate Analytical Models of Point-to-Point Communication in Distributed Systems [J]. IEEE Trans. Comput, 2007, 56(3): 314-327

[4] Cameron K W, Sun X-H. Quantifying Locality Effect in Data Access Delay: Memory logP[C]// Proceedings of the 17th International Symposium on Parallel and Distributed Processing(IPDPS '03). IEEE Computer Society, 2003

[5] Alexandrov A, Ionescu M F, et al. LogGP: incorporating long messages into the LogP model—one step closer towards a realistic model for parallel computation[C]// Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures(SPAA'95). ACM, 1995: 95-105

[6] Culler D, Karp R, et al. LogP: towards a realistic model of parallel computation[C]// Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming (PPOPP '93). ACM, 1993: 1-12

[7] Frank M I, Agarwal A, et al. LoPC: modeling contention in parallel algorithms[C]// Proceedings of the sixth ACM SIGPLAN symposium on Principles and practice of parallel programming (PPOPP '97). ACM, 1997: 276-287

[8] Ino F, Fujimoto N, et al. LogGPS: a parallel computational model for synchronization analysis[C]// Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming(PPoPP '01). ACM, 2001: 133-142

[9] Karp R M, Sahay A, et al. Optimal broadcast and summation in the LogP model[C]// Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures (SPAA'93). ACM, 1993: 142-153

[10] Moritz C A, Frank M I. LoGPC: Modeling Network Contention in Message-Passing Programs[J]. IEEE Trans. Parallel Distrib. Syst., 2001, 12(4): 404-415

[11] Chen Wen-guang, Zhai Ji-dong, Zhang Jin, et al. LogGPO: An Accurate Communication Model for Performance Prediction of MPI Programs[J]. Sci. China Ser. F-Inf. Sci. j., 2009, 52(1): 1-6

[12] Zhang Yun-quan, Sun Jia-chang, Tang Zhi-min, et al. Memory Complexity on Numerical Programs [J]. Chinese Journal of Computers, 2000, 23(4): 363-373

[13] Zhang Yun-quan. Performance Optimization on Parallel Numerical Software Package and Study on Memory Complexity [D]. Beijing: Institute of Software, Chinese Academy of Sciences, 2000

[14] Zhang Yun-quan. DRAM(h): A Parallel Computation Model for High Performance Numerical Computing[J]. Chinese Journal of Computers, 2003, 26(12): 1660-1670

[15] Zhang Yun-quan, Sun Jia-chang, Tang Zhi-min, et al. Memory Complexity on Numerical Programs [J]. Chinese Journal of Computers, 2000, 23(4): 363-373

[16] Zhang Yun-quan, Chen Guo-liang, Sun Guang-zhong, et al. Models of parallel computation: a survey and classification [J]. Front. Comput. Sci. China, 2007, 1(2): 156-165

[17] Tu Bi-bo, Zou Ming, Zhan Jian-feng, et al. Research on Parallel Computation Model with Memory Hierarchy on Multi-Core Clusters[J]. Chinese Journal of Computers, 2008, 31(11): 1948-1955

[18] Wu Jun-jie, Yang Xue-jun, Zeng Kun, et al. DOOC: A Software/Hardware Co-managed Cache Architecture for Reducing Cache Thrashing[J]. Journal of Computer Research and Development, 2008, 45(12): 2020-2032

[19] Aggarwal A, Chandra A K, Snir M. On communication latency in pram computations[C]// Proceedings of the first annual ACM symposium on Parallel algorithms and architectures (SPAA'89). ACM. New York, NY, USA, 1989: 11-21

[20] Gibbons P B. A more practical pram model[C]// Proceedings of the first annual ACM symposium on Parallel algorithms and architectures(SPAA '89). ACM, New York, NY, USA, 1989: 158-168

[21] Molka D, Hackenberg D, Schone R. Memory performance and cache co-herency effects on an intel nehalem multiprocessor system[C]// Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques(PACT'09). IEEE Computer Society, Washington, DC, USA, 2009: 261-270

[22] Savage J E, Zubair M. A unified model for multicore architectures[C]// Proceedings of the 1st International Forum on Next-generation Multicore/Manycore Technologies (IFMT'08). ACM, New York, NY, USA, 2008: 1-12

[23] Ulm D R, Scherger M. Stream pram [C]// Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium(IPDPS'05). Workshop 14. IEEE Computer Society, Washington, DC, USA, 2005: 265

[24] Valiant L G. A bridging model for multi-core computing[J]. J. Comput. Syst. Sci., 2011, 77(1): 154-166

[25] Zhang E Z, Jiang Y, Shen X. Does cache sharing on modern cmp matter to the performance of contemporary multithreaded programs? [C]// Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP'10). ACM, New York, NY, USA, 2010: 203-212

[26] Jiang Y, Shen X, Chen J, et al. Analysis and approximation of optimal co-scheduling on chip multiprocessors[C]// Proceedings of the 17th international conference on Parallel architectures and compilation techniques(PACT '08). ACM, 2008: 220-229

[27] Ding C, Kennedy K. Improving cache performance in dynamic applications through data and computation reorganization at run time[C]// Proceedings of the ACM SIGPLAN 1999 conference on programming language design and implementation (PLDI '99). ACM, 1999: 229-241

[28] Yan Y, Zhang X, Zhang Z. Cacheminer: A runtime approach to exploit cache locality on smp[J]. IEEE Trans. Parallel Distrib. Syst., 2000, 11(4): 357-374