面向NUMA集群的代数多重网格算法优化

顾 坚[1,2] 刘 伟[3]
(国家高性能计算机工程技术研究中心 北京 100193)[1]
(公安部网络安全保卫局 北京 100741)[2] (北京邮电大学计算机学院 北京 100876)[3]

摘 要 代数多重网格(AMG)是众多数值模拟应用的核心算法,在基于多核的NUMA架构的机群系统上,AMG的并行扩展性暴露了新的问题。通过设计感知NUMA架构的内存分配器,将划分给多个线程的数据分割并绑定到运行对应线程的CPU所属的NUMA存储节点上,从而改善了OpenMP多线程并行的数据局部性,使BoomerAMG程序在大规模多核计算平台上具有更好的并行扩展性。在单节点和小规模机群的测试中,使用NAAlloc分配器分别获得了最高16%和60%的性能提升。

关键词 代数多重网格,NUMA,多核,局部性
中图法分类号 O246 **文献标识码** A

Optimizing Algebraic Multigrid on NUMA-based Cluster System

GU Jian[1,2] LIU Wei[3]
(The National High Performance Computer Engineering Technology Research Center, Beijing 100193, China)[1]
(Network Security Service of the Ministry of Public Security, Beijing 100741, China)[2]
(School of Computer Science & Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)[3]

Abstract Algebraic Multigrid (AMG) as a key method of many numerical simulation programs, a serious scalability problem is encountered when it scales on mulit-core NUMA based cluster. There is a problem that data tend to be allocated nearby the CPU core on which the main thread are running, and the data lack locality. By applying our NUMA-aware memory allocator, each partition of a data block can be binded to the corresponding NUMA memory nodes which the data owner thread is laid on, so as to successfully maintain more data locality when using OpenMP parallelizing AMG, and make BoomerAMG scaling up more easily and efficiently. In the experiment on a single node and a small 16 nodes cluster, using NAAlloc allocator gets 16% and 60% peak performance improvement respectively.

Keywords Algebraic mulitgrid, NUMA, Multic-core, Locality

1 介绍

大规模的数值模拟的普遍应用对计算机系统的计算能力提出了很大的挑战,AMG作为众多数值模拟应用的核心算法,消耗了大部分的计算时间,使得优化AMG的计算效率具有重要的实用价值。同时随着集成电路工艺和超级计算机体系结构的发展,大规模多核系统成为主流,在这种新的趋势下,AMG的并行扩展性暴露了新的问题。优化大规模多核机群上的AMG并行扩展性成为一个有挑战性和创新性的研究课题[1]。

多重网格方法可分为几何多重网格(GMG)和代数多重网格(AMG)两种。GMG方法是最先被发现和应用的多重网格算法,它利用待求解问题的几何信息定义规则的网格层次结构,构造各多重网格组成部件。一般情况下,定义在结构化网格和半结构化网格上的应用问题,适用GMG方法,通常可以高效地求解。同时GMG方法由于使用规则的网格结构,因此非常适合在大规模并行系统上扩展。经过30多年的研究和发展,GMG算法和理论都基本成熟,它在单机上和各类并行机上,都得到广泛应用并取得了很大成功,目前已经成为数值计算中一种重要的加速迭代收敛的技术。AMG方法晚于GMG方法出现,它同GMG基于相同的思想,然而在构造多重网格的层次结构和算子的方法上有所不同。AMG的"网格"由所有未知数或其子集构成,构造多重网格各层的算子只利用代数信息,如系数矩阵(待求解方程 $Au=f$ 中的矩阵 A)的元素,无需问题的几何网格信息,相对于GMG方法,其健壮性大大增强。这样AMG方法成为一种适应性很好的黑盒子解法器(black-box solver),同时还保持了GMG方法良好的算法可扩展性。特别是AMG方法求解不需要问题的几何网格信息,对于很多具有非结构网格、复杂区域或者非光滑系数的问题,可以方便有效地进行处理,从而在复杂系统的数值模拟中得到广泛使用。

AMG方法具有低的算法复杂度、良好的并行扩展性和

到稿日期:2013-08-01 返修日期:2014-03-13
顾 坚(1956—),男,硕士,高级工程师,主要研究方向为计算机网络安全、并行计算,E-mail:gahsh@sina.cn;刘 伟(1979—),男,博士生,主要研究方向为计算机网络安全、数据挖掘。

方便易用的特性，并且能够处理复杂的非结构化网格问题以及实际应用中对很多具有重要价值的数值模拟问题，因此已成为最受欢迎的迭代方法之一。AMG 的应用领域包括但不限于流体力学、微观力学、半导体、高能物理、石油勘测等需要复杂系统数值模拟的学科。在 2005 年国际超级计算机会议上，美国劳伦斯伯克利实验室(LBNL)的著名计算科学专家 Horst Simon 列出了他认为的近 20 年超级计算领域的十大重要成就[2]，以多重网格为代表的多水平算法是唯一被列出的数值算法。2009 年 9 月，美国能源部科学办公室评出计算科学领域在过去 18 个月中取得的十大突破性成就[3]，其中就包括劳伦斯利弗莫尔实验室(LLNL)开发的针对电磁场计算的 AMG 解法器(AMS)。

2 AMG 算法

2.1 算法原理

考虑上一节中的稀疏线性方程组 $AU=f$ 的求解，A 为系数矩阵，*AMG* 中将未知量下标的集合定义为网格，记为 $\Omega=\{0,1,\cdots,n-1\}$，初始网格层 $\Omega^0=\Omega$，初始网格算子 $A^0=A$。AMG 算法包括启动阶段(Setup Phase) 和求解阶段(Solve Phase)，启动阶段主要定义多重网格的各个组件(Component)，包括嵌套网格层、插值/限制算子及光滑算子(Jacobi、Gauss-Seidel 等迭代)等；求解阶段重复执行多重网格循环，其中有 V-Cycle、W-Cycle 等不同模式，这里以 V-Cycle 为例。

—启动阶段

(1)构造嵌套网格层，从最细网格层 $k=0$ 开始：

a)将 Ω^k 划分为粗网格和细网格集合，$\Omega^k=C^k\cup F^k$，$C^k\cap F^k=\emptyset$；

b)设 $\Omega^{k+1}=C^k$。

(2)构造插值算子 P^k，限制算子 $R^k=(P^k)^T$，使用 Galerkin 方法构造网格算子 $A^k=R^kA^{k-1}P^k$。

(3)如果 Ω^{k+1} 足够小，记下此最大层数 *max_level*，停止；否则 k 自增，回到上面(1)的过程。

—求解阶段

执行 V-Cycle，初始时为最细层网格 $k=0$，标记 V-Cycle 的方向，初始为向下，即向粗网格方向：

光滑：对方程 $A^ku^k=f^k$ 做数次松弛迭代；

(1)如果 $k=max\_level$，直接在粗网格上求解方程，标记 V-Cycle 方向向上；

(2)如果 V-Cycle 方向向下，进行粗网格限制：初始化粗网格未知量 $u^{k+1}=0$；

a)计算细网格残差 $r^k=f^k-A^ku^k$；

b)残差限制到粗网格，并作为粗网格方程的边界值 $f^{k+1}=R^{k+1}r^k$；

c)$k=k+1$，返回(1)。

(3)如果 V-Cycle 方向向上，校正细网格：

a)使用插值方法校正近似解 $u^k=u^k+P^ku^{k+1}$。

b)如果 $k=0$，则 V-Cycle 结束退出；否则 $k=k-1$，回到(1)。

校验残差范数 $r=f^0-A^0u^0$ 是否满足 $\|r\|<$tolerance，如果不满足则重复 V-Cycle，直至得到足够精度的解。

具体到 AMG 所使用的离散模板(stencil)、粗化算法、松弛算子等要素具有很多种选择，包括各种算法组件的参数，需要应用程序编写者根据实际需求和计算资源等作出合理配置。本课题的研究受时间所限，将重点实现部分常用的算法组件，并使用典型的参数评测优化后的程序性能。

2.2 BoomerAMG 并行实现

本文基于 BoomerAMG[4-6] 的并行实现展开并行算法的优化研究工作，如下分别是 BoomerAMG 的建立阶段和求解阶段的流程图。

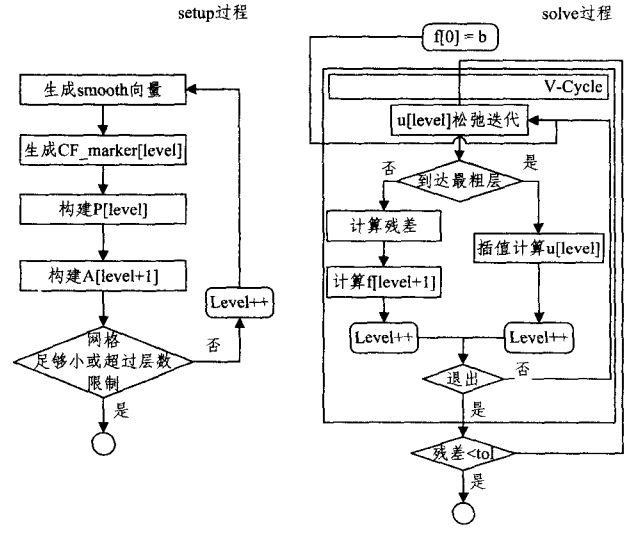

图 1 BoomerAMG 程序流程

步骤说明，建立阶段：

—cf_marker 数组记录了网格中的行是否为粗点所在的行，后面求解阶段的松弛运算中将使用它来分别完成粗点和细点的两轮迭代；

—矩阵 P 为插值矩阵，其转置矩阵 R 为限制矩阵；

—矩阵 A 为网格算子，构建方法为 $A[k+1]=R[k]A[k]P[k]$。

求解阶段：

—残差计算完成 $res[k]=f[k]-A[k]u[k]$；

—残差限制到粗网格，计算 $f[k+1]=R[k]res[k]$；

—插值计算 $u[k]=u[k+1]P[k]$；

—如果一个 V-Cycle 结束后计算残差范数满足精度要求，则求解结束；否则需继续重复新一轮的 V-Cycle。

AMG 中的建立阶段和求解阶段并不一定是一一对应的关系，鉴于在很多应用中把 AMG 当作一种局部区域求解的方法使用，外围的迭代可能进行很多次，而同一网格结构不同边界条件的求解无需重复 AMG 建立步骤，所以本文研究的重点放在 AMG 求解阶段时间的优化上。

BoomerAMG 包含节点间和节点内两层并行。节点间的并行依照区域分割原理，将待求解问题的系数矩阵、迭代矩阵、未知量向量、中间结果等数据按节点划分，并分布存放在所有的计算节点上，在计算区域重叠的时候节点间集中交换需要的数据。节点间并行使用 MPI 编程模型，数据分割和交互都是显式进行的。为了实现节点间的并行，BoomerAMG 设计了一套分布式的数据存储结构，应用程序的编写者需要了解这套框架，才能在多节点环境下正确使用 AMG 求解问题。BoomerAMG 在顶层使用分布式的数据结构适应多节点的 MPI 编程模型，同时在节点内本地矩阵 diag/offd 的矩阵-向量运算实现中使用 OpenMP 的细粒度并行。节点内的细

粒度并行层次对应用编写者是透明的，使用 BoomerAMG 时不需要关心多线程并行实现的细节。同样地，如果我们试图优化 BoomerAMG 在节点内的并行，首先需要了解这一层次并行的实现方法。由于本地矩阵 diag/offd 均按照通用的 CSR 格式存储，自然地多线程并行使用按行划分数据的方式，目前的 BoomerAMG 仅仅简单按行平均分配矩阵元素给多个线程同时运算。这样做一方面是由于按行划分之后，细粒度并行的实现相对容易，另一方面的好处是大多数情况下每个线程只需要计算和更新自己这部分的结果向量，线程之间互相不干扰，而只用于读的矩阵是所有线程共享的，使得这种数据划分的方案效率较高。不过，BoomerAMG 细粒度并行层次的性能仍然有提升的空间。

以经典的二维拉普拉斯问题进行实验，实验平台为单节点 2 路 Intel Xeon X5650 CPU，共 12 个 CPU 核心，问题规模为 4k×4k 网格，待求解方程矩阵包含 80M 非零元。分别统计当使用 1/2/4/8/12 个 CPU 核心参与运算时消耗的求解时间，实验结果如图 2 所示。若单个节点的 CPU 核数增加到 8 核以上，可以明显看到，不论使用 MPI 或者是 OpenMP 并行的方案，同样问题的求解时间并没有改善，甚至在某些情况下求解时间反而增加了。

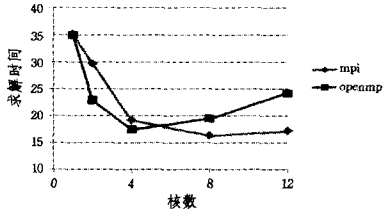

图 2　结点内 AMG 并行算法的扩展性

考虑到未来的大规模计算平台单个节点的 CPU 核数迅速增加的趋势，AMG 在多核系统上的强可扩展性问题促使我们必须仔细分析这种现象产生的原因，并且有针对地采取措施优化程序的多核并行性能，提高多核平台的计算效率，从而使 AMG 在新的大规模多核计算平台上仍然保持良好的实用性。

3　AMG 算法的多层次并行优化

3.1　NUMA 体系结构和并行

在多核体系结构下实现多层次并行，有许多对性能产生影响的因素需要加以考虑，其中尤其突出的是 NUMA(Non-Uniform Memory Access)存储结构的影响。现代大规模计算平台的高性能节点普遍使用 ccNUMA(cache coherent NUMA)体系结构，依据单一节点中 CPU 插槽的个数可以分为两路、四路甚至更多处理器的架构。多路服务器具体的硬件架构则包含更加复杂的细节，并且随着计算机结构的研究和产业的发展，变化非常迅速。比如近几年 CPU 都普遍采用内部集成存储控制器的方式连接内存，抛弃了以往使用北桥芯片的方案。这样的硬件架构的变化使得应用软件需要不断进行优化，以适应计算平台的新特性，并取得好的并行性能。

单个节点上多个物理 CPU 通过 QPI(Intel QuickPath Interconnect)点对点连接，组成典型的 ccNUMA 结构。每个 CPU 具有独立的内存，称为一个 NUMA 存储节点，依据 CPU 数量和连接方式的不同，访问非本地的存储节点需要跨越一个甚至更多 CPU 才能完成。基于 AMD CPU 的多路服务器具有相似的架构，除了 CPU 之间的连接使用的是 HT (Hypre Transport)的替代方案。在这样的 ccNUMA 节点上数据远程访问的开销对程序性能的影响不容忽视。图 3 是在自己的两路服务器节点上通过 numademo 测试得到的远程内存访问的带宽损失，视不同的内存操作模式这个数值在 25%～40%。对于当前主流的多核服务器，定性的结果应是类似的，即因为 NUMA 结构的特性，并行应用程序的性能优化的一个重要措施是优化数据局部性，尽量避免远程内存访问。

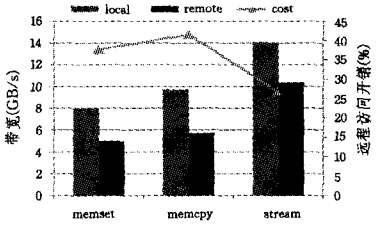

图 3　NUMA 节点的远程访问开销

3.2　层次化并行

BoomerAMG 支持使用 MPI+OpenMP 多层次并行的方式，以提高大规模多核机群系统上的并行扩展性。在以 ccNUMA 节点组成的常见机群系统上，使用 MPI 或 OpenMP 并行编程的实现大致有 3 种组合模式，分别为纯 MPI、多 MPI 进程混合 OpenMP、单 MPI 进程加 OpenMP，图 4 中显示了节点内的进程和线程在 3 种模式下的分布。

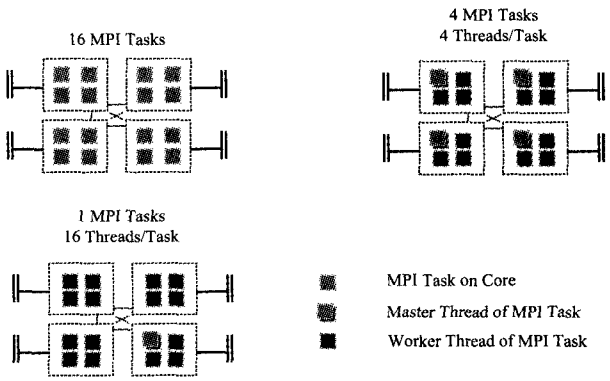

图 4　3 种 MPI+OpenMP 组合模式的示意图

纯 MPI 的模式在每个处理器核上运行一个 MPI 进程，即使考虑使用的 MPI 库对节点内进程通信做了优化，相对 OpenMP 一类的共享内存的并行方案，其数据通信开销仍然较大；同时整个系统 MPI 进程的数量与处理器核数相当，在当前计算平台单节点集成度迅速提高的趋势下难以大规模扩展。在单节点多 MPI 进程的模式下，一般在每个物理 CPU 上运行一个 MPI 进程，即 MPI 进程的数量与节点 CPU 插槽(socket)数相当，一个物理 CPU 内使用 OpenMP 并行，OpenMP 线程数与 CPU 核心数目相当。在单节点只启动一个 MPI 进程的模式下，节点内部所有 CPU 核使用 OpenMP 做并行扩展。比较后面两种常用的多层次并行的进程/线程组合模式：

一多 MPI 进程模式将每个物理 CPU 作为顶层计算单元，每个计算单元管理各自独立的数据存储空间，避免了 NUMA 系统上内存远程访问的操作，这是它的优势；相对的单节点单 MPI 进程的分配模式利用了 ccNUMA 系统节点内所有处理器核共享内存的特性，最大限度地减少了 MPI 进程

间显式数据交换的开销。

—在多 MPI 进程模式下 MPI 进程的数目与系统物理 CPU 的数量相当，而单 MPI 进程模式的 MPI 进程数目只与系统计算节点数目相当，相比系统物理 CPU 的数量，大规模机群的计算节点数目增长是很缓慢的，考虑到第 1 节引言中指出的大规模 MPI 程序的可靠性随着 MPI 进程数目增加而下降的趋势，单节点单 MPI 进程的模式更适宜在大规模机群上扩展，并且适应未来高性能计算的发展趋势。

基于以上两个原因，我们倾向于使用单节点单 MPI 进程的模式实现多层次并行。不过这种方案的问题也很明显，在 NUMA 结构的节点上所有处理器核心使用 OpenMP 做共享存储的并行，容易产生远程内存访问，对访存带宽敏感的应用程序，这种额外的访存开销将降低程序的并行性能。因此在并行程序的设计实现上需要注意保证数据局部性(data locality)，克服 NUMA 系统特性的不利影响。

3.3 数据局部性优化

BoomerAMG 程序中线程共享的数据包括本地矩阵和本地向量数据两种，其中本地矩阵涉及多线程间的数据划分，是优化 NUMA 系统上数据局部性要考虑的重点对象。AMG 中矩阵数据空间的分配是在建立阶段完成的，因为建立阶段特别是建立粗化矩阵的阶段的算法具有串行本性，使矩阵空间的分配和数据的写操作只能由 MPI 主线程完成。如果使用 3.2 节所述的总 MPI 进程数最少的单 MPI/OpenMP 组合方式，使用原有的 malloc 族内存分配函数而不采取针对 NUMA 结构的措施，则矩阵数据将倾向于集中在主线程所在的存储节点上，导致求解阶段中其他的工作线程读取矩阵数据成为远程访问，数据局部性被破坏。

在现有的 BoomerAMG 程序框架下，以求解阶段的数据划分为目标，保持多线程数据局部性的合理设计应该是这样：矩阵数据在逻辑上是连续整体，地址空间线性连续；同时为了保证每个 CPU 核上的工作线程访问自己拥有的数据都是本地访问，以物理页面为基本单位的内存空间应该尽量分配在所属线程所在的存储节点上，如图 5 并行数据分割绑定的模式所示。

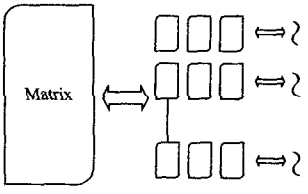

图 5 并行数据分割绑定的模式

考虑具体的实现方法，由于上一节中已经说明的原因，不适宜使用现有的内存堆管理器，比较合理的方法是使用 Linux 系统提供的基础 MUMA API，设计实现符合上面需求的内存分配方案，做到兼顾功能的完备和接口的抽象，之后可以考虑对一般的多层次并行程序的实用性。实现方案的原理示意如图 6 所示。多线程共享的矩阵数据由 NUMA-Aware 内存分配器分割和绑定到对应的 NUMA 节点上，数据划分的区间可以通过分配器提供的接口由用户程序指定，结合辅助的线程 CPU 关联操作，可以保证矩阵数据的局部性。

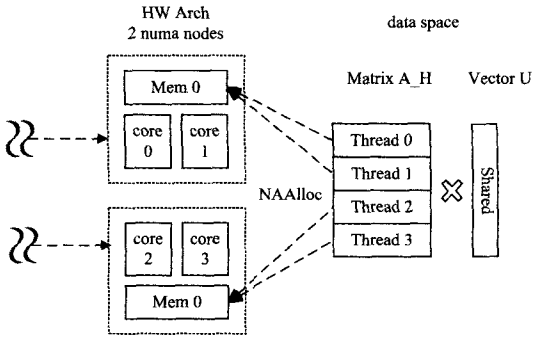

图 6 并行数据分割绑定的实现方案

3.4 NUMA-Aware 内存分配器

将上一节中提出的 BoomerAMG 数据局部性的解决方案具体化，同时考虑实际应用中的实用性，本节综合出 NUMA-Aware 内存分配器 NA-Allocator 应该具有的几个方面功能：

—数据分割绑定：最主要的特性是应用程序通过暴露的接口指定期望的数据划分区间，内存分配器依此完成内存空间的申请和将分割子空间绑定到对应的存储节点上。

—线程与 NUMA 节点的自动映射：使 NA-Allocator 具有实用性所需的重要特性，即应用程序无需关心线程运行在哪个 NUMA 节点上，而是由分配器在初始化时探知计算机内的 CPU 连接拓扑结构和线程绑定的策略，维护线程与 NUMA 节点的映射关系，供内存绑定时使用。

—维护内存分配状态：涉及到存储空间的释放，维护所有已分配内存块的信息是内存分配器的基本功能。在此基础上，可以实现诸如内存分配日志等附加的实用功能。

实现上面功能的 NUMA-Aware 内存分配器的主要函数和运作流程如图 7 所示。其中有几点细节值得做一些更详细的解释：

—NAInit 需要的系统结构相关参数在 BoomerAMG 代码库的编译阶段提取，以静态数组形式存储在一个 C 语言头文件中，供 AMG 应用程序引用，以实现绑定内存时无需应用程序指定存储节点的功能。最主要的参数是系统分配的逻辑 CPU 核心序号 core_id、主板分配的唯一 CPU 核心序号 acipid 和 NUMA 存储节点序号三者的映射关系，由于这种映射在 Linux 系统启动时已经完成过分析和记录，因此我们只需要读取相关的系统文件就可以获得。其中，core_id/acipid 映射通过分析/proc/cpuinfo 文件获得，core_id/numa_node 映射通过分析/sys /devices /system /node[N]的目录结构获得。另外需要在此过程之前先检查系统版本，以确定系统支持 NUMA 特性，否则不使用这个 NUMA-Aware 内存分配器。

—NAInit 调用时获取线程绑定的信息，方法是在 OpenMP 并行代码中调用一段嵌入的汇编代码读取 CPU 属性寄存器，得到运行线所在 CPU 核的 acipid，对照上面另外两个映射表，获得线程-NUMA 存储节点映射。目前的实现仍然需要程序运行时外部手工指定线程绑定，使用 OMP_CPU_AFFINITY 环境变量。

—NAAlloc 保存已分配内存块信息、使用 STL 的 map 结构，方便添加和查找。

—BoomerAMG 默认的本地矩阵存储格式是 CSR，每行数据长度不等，数据分割绑定时通过一个长度等于本地线程

数目的数组传入数据划分方案，数组每个数值标志对应线程的结束数据的位置。这种方式隐含要求矩阵数据的划分按照线程号顺序进行，不过这点在 OpenMP 并行实现上容易保证。

一保存的内存分配日志可以用于分析应用程序的内存使用情况，也可以检查可能的内存分配失去局部性的问题。

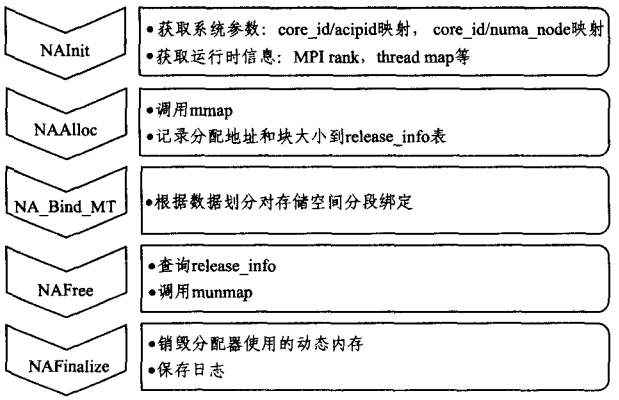

图 7　NUMA-Aware 内存分配器的运作流程

4　实验结果

实验使用的两种硬件平台配置如表 1 所列。

表 1　实验平台参数

实验平台		I	II
处理器		Intel Xeon X5650	Intel Xeon X7550
架构		Westmere EP	Nehalem-EX
主频		2.66GHz	2.00GHz
sockets		2	8
内存	类型	DDR3	DDR3
	容量	24GB	128GB
	核心频率	1.333GHz	1.066GHz
	带宽/socket	32GB/s	17GB/s
PCIe2.0		x16	x16

系统环境为 CentOS5.3，Linux 内核版本 2.6.18，编译环境使用 gcc 套件。代码优化基于 HYPRE 软件包 2.7beta 版。

首先展示实验平台 I 单节点上原始 BoomerAMG 程序在几种不同的 MPI+OpenMP 两层并行模式下的求解性能（见图 8），每种模式均使用了合适的进程/线程绑定策略：其中 mpi-only 线代表纯 MPI 并行模式，openmp-only 线代表一个节点只启动一个 MPI 进程，节点内使用 OpenMP 并行扩展，hybrid_2xThreads 指一个节点上启动两个进程，分别绑定到两个 socket 上的混合模式，而 hybrid_4xThreads 对应一个节点使用 4 个 MPI 进程。在实验平台 II 上重复同样的实验，得到的结果如图 9 所示，其中 hybrid-mpi 线代表每个 CPU socket 上启动一个 MPI 进程、socket 内部使用 OpenMP 扩展的混合并行方式。

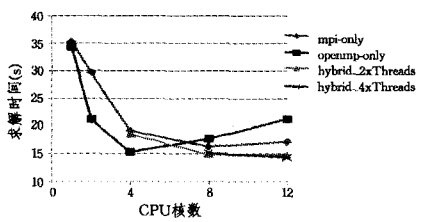

图 8　原始 BoomerAMG 程序各种并行组合模式的性能（平台 I）

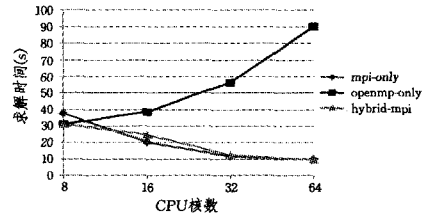

图 9　原始 BoomerAMG 程序各种并行组合模式的性能（平台 II）

从结果中可以看到，BoomerAMG 在节点内作并行扩展的几种方案中，OpenMP 的并行扩展性有很大的改进空间，纯 MPI 并行有时不能发挥最大的运算能力，而根据硬件架构选择多 MPI 进程加 OpenMP 线程的并行方案能够获得最佳的性能。

考虑到希望能尽量使用较少的 MPI 进程数量完成多节点的并行扩展，迫切需要优化单 MPI 进程+OpenMP 并行的组合模式，即上图中 openmp-only 线代表的方案，使这种单节点启动单个 MPI 进程的模式具有更大的实用性。

下面是使用 NUMA_Aware 内存分配器在实验平台 I 上测得的单节点上的性能改进，如图 10 所示。更具体地说，我们使用 NAAlloc 代替默认的 malloc 族函数，在 AMG 建立阶段完成各层网格算子 A 的对角线矩阵块和插值算子 P 的空间分配，使得这部分数据在求解阶段使用 OpenMP 并行时保有局部性，即分割给线程的数据块位于线程对应 CPU 核的 NUMA 存储节点上。在单节点使用 1～12CPU 核时，AMG 求解阶段性能得到了 3%～16%的提高。在实验平台 II 上重复同样的实验，测得的单节点 OpenMP 的扩展性如图 11 所示。

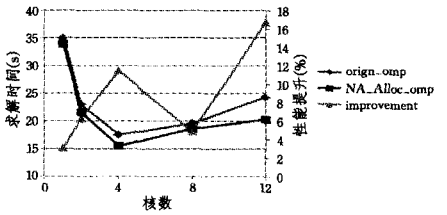

图 10　NAAlloc 的单节点 OpenMP 性能提升（平台 I）

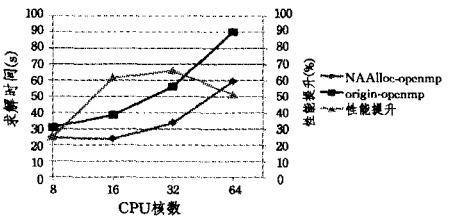

图 11　NAAlloc 的单节点 OpenMP 性能提升（平台 II）

其中横坐标代表的 CPU 核数的扩展方式是分别使用 1/2/4/8 个物理 CPU 的测试结果。使用 NAAlloc 之后，单节点 OpenMP 的性能相比原始 BoomerAMG 的 OpenMP 性能有很大的改进，求解阶段的性能提高了 25%～60%。

进一步评测在小规模机群上使用 NAAlloc 优化 BoomerAMG 强扩展性的效果，每个计算节点的配置同实验平台 I。具体的方法是保持总的问题规模不变，固定使用 16 个计算节点，改变每个节点使用的 CPU 核数量，分别使用原始 BoomerAMG 的 OpenMP 并行扩展和 NAAlloc 改进存储分配，统计多线程下的求解阶段时间，计算相对每个节点单个线程的归一化加速比，得到图 12 中的结果，每个节点分别使用 2/4/8/12 个 CPU 核时的归一化加速比提高了 50%～60%，

表明 NAAlloc 优化 OpenMP 并行数据局部性的方案对改善机群上 AMG 强可扩展性有较好的效果。

值得注意的是，尽管使用 NAAlloc 代替 malloc 完成部分大矩阵的存储分配，一定程度改善了 AMG 求解阶段 OpenMP 并行的数据局部性，但是使用 OpenMP 完成跨 socket 的节点内并行仍然不是一种好的方案，实验平台 II 上的测试结果是一个典型的例子。分析跨 socket 的 OpenMP 并行扩展性下降的原因，一方面是跨 socket 的 OpenMP 并行段开销增大，如同步等操作开销，另一方面更重要的原因是，BoomerAMG 的计算核心代码对 OpenMP 并行没有做充分的优化，更细致的时间分析表明跨 socket 后转置 SpMV 的性能下降非常快，其中包含大量跨 socket 的数据运算。因此可以认为 BoomerAMG 的节点内 OpenMP 并行的性能还有很大的改进空间，综合算法、核心代码实现和 NUMA-aware 内存分配等各方面的优化手段是必要的。

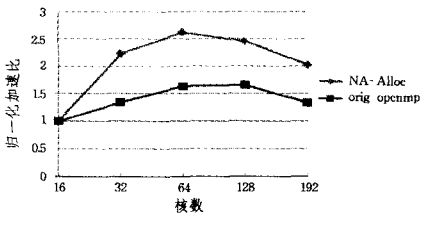

图 12　NAAlloc 在小规模机群上的 BoomerAMG 性能提升

5　相关工作

AMG 算法具有收敛速度与问题规模呈线性相关的优良特性，而且具有很好的并行潜力，因此从 AMG 算法提出开始，人们就致力于将它移植到各种计算平台上，发掘其并行潜力。从早期的向量机、SIMD 并行机，到较新的超立方体、MPP 和大规模集群系统，有大量应用和优化 AMG 并行运算的工作[7-9]。较早的比较系统的实践和讨论多重网格类算法的工作来自德国的 GMD，他们在包括 CrayY-MP、Intel ipsc/860、Meiko、SUPRENUM、Neube、CM-2、CM-5 等工作站机群上实现了经典的多重网格算法的并行化，并且在大量实验的基础上探讨了此类算法的并行效率[10]。目前 AMG 算法的并行优化工作仍然集中在如何在尽量不影响算法收敛速度的前提下，利用计算平台的性能特性，对一定规模的问题优化 AMG 的并行效率。

当前大规模多核系统成为主流的并行计算平台，AMG 在多核系统上的并行扩展性受到新的挑战：强可扩展性不理想，单个计算节点的 CPU 核数增加，运算时间没有达到预期的改善，系统的计算能力存在很大浪费。近两年这些问题已经逐渐引起人们的重视，特别是作为 AMG 软件的开发和应用的重要机构的美国 LLNL 实验室已经有数篇论文和报告对 AMG 在新的多核平台上的并行扩展遇到的困难和优化手段进行了实验和探讨[11,12]。LLNL 的研究结果指出在新的多核平台上有必要对 AMG 的并行扩展性做重新评估，影响 AMG 在多核系统上扩展性的因素可能包括单节点多个 sock 和单 CPU 多核心的复杂层次结构、多层 cache 共享、多内存控制器的 NUMA 特性和平均每个 CPU 的存储带宽减小等，这些平台相关的因素对 AMG 的并行性能的影响需要进一步更细致的工作来评估，然而在大规模多核平台上的 AMG 并行优化目前才逐步展开。

在大规模计算系统的节点核心数迅速增加、内部层次结构越来越复杂的趋势下，鉴于传统的纯 MPI 的 AMG 计算模型已经显示出难以获得高效的并行扩展性，LLNL 的研究指出应该重视结合共享内存编程模型如 OpenMP 的混合并行模式，这方面已有了初步的探索性工作。实验显示 MPI/OpenMP 混合并行在多核平台上受到数据局部性等因素影响，难以取得很好的并行效率，并且在实现上需要相当的技巧[13]。目前的工作只使用了简单的原型问题作为验证，且未能形成在大规模多核平台上扩展 AMG 的成熟的解决方案。

结束语　本文基于对多核体系结构下 AMG 程序的多层次并行的原理和性能分析，认为从优化 OpenMP 多线程并行的数据局部性入手是解决 AMG 在新的多核计算平台上的并行扩展性的有效方向，并且设计和实现了一个在 BoomerAMG 中使用的较为通用的内存分配器，其适应多核平台的 NUMA 架构特性，对改善 OpenMP 并行的数据局部性具有较好的效果，总体上提高了 BoomerAMG 在多核机群上的并行扩展性。NUMA_Aware 内存分配器的工作对其他访存带宽敏感的并行程序优化同样具有借鉴意义。

参 考 文 献

[1] 裴文兵，徐小文. 国家 863 课题“激光聚变领域的高性能计算应用研究”进展报告[R]. 北京应用物理与计算数学研究所，2011

[2] Simon H. Progress in Supercomputing: The Top Three Breakthroughs of the Last 20 Years and the Top Three Challenges of the next 20 Years[M]. Pennyhill Press, 2014

[3] Office of Science, DOE. Top Breakthroughs in Computational Science[OL]. http://www.scidacreview.org/0901/html/bt.html

[4] Holter W. A vectorized multigrid solver for the three dimensional poisson equation[J]. Appl. Math. and Comput., 1986, 19(1-4): 127-144

[5] Falgout R D. An Introduction to Algebraic Multigrid, Computing in Science and Engineering[J]. Special Issue on Multigrid Computing, 2006(8): 24-33

[6] Falgout R, Brannick J, Brezina M, et al. New Multigrid Solver Advances in TOPS[C]// Proceeding of SciDAC 2005, Journal of Physics: Conference Series, Institute of Physics. San Francisco, CA, 2005

[7] McByan O, Frederickson P, Linden J, et al. Multigrid methods on parallel computers-A survey of recent developments[J]. Impact of computing in science and engineering, 1991, 13: 1-75

[8] Chan T, Saad Y. Multigrid algorithms on the hypercube multigrid processors[J]. IEEE Trans. Comput., 1986, 35(11): 969-977

[9] McByan O. The SUPRENUM and GENESIS projects[J]. Parallel Computing, 1994, 20: 1389-1396

[10] Gahvari H, Baker A H, Schulz M, et al. Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms[C]// Proc. of the 25th International Conference on Supercomputing (ICS 2011). Tucson, AZ, 2011

[11] Baker A H, Falgout R D, Gamblin T, et al. Scaling Algebraic Multigrid Solvers: On the Road to Exascale[C]// Proc. of Competence in High Performance Computing, CiHPC 2010. Schwetzingen Germany, 2010

[12] Baker A H, Falgout R D, Kolev T V, et al. Scaling hypre's Multigrid Solvers to 100000 Cores[C]// High Performance Scientific Computing: Algorithms and Applications. 2012: 261-279

[13] Baker A H, Gamblin T, Schulz M, et al. Challenges of Scaling Algebraic Multigrid Across Modern Multicore Architectures[C]// Proc. of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2011). Anchorage, AK, 2011