第 36 卷 第 8 期 计 算 机 工 程 2010 年 4 月
Vol.36 No.8 Computer Engineering April 2010
·软件技术与数据库· 文章编号：1000—3428(2010)08—0049—03 文献标识码：A 中图分类号：TP311.5

HPMR 在并行矩阵计算中的应用

郑启龙[1,2]，吴晓伟[1,2]，房 明[1,2]，王 昊[1,2]，汪 胜[1,2]，王向前[1,2]
(1. 中国科学技术大学计算机科学技术学院，合肥 230027；2. 安徽省高性能计算与应用重点实验室，合肥 230026)

摘 要：为了解决传统并行编程难度大、效率低的问题，提出一种基于 MapReduce 模型的并行编程方法，在高性能 MapReduce 平台上实现矩阵并行 LU 分解。实验结果表明，相比传统并行编程模型，MapReduce 模型并行程序可较好满足高性能数值计算需求，其编程简洁性和可读性能有效提升并行编程效率。

关键词：高性能 MapReduce；并行编程；数值计算；LU 分解

Application of HPMR in Parallel Matrix Computation

ZHENG Qi-long[1,2], WU Xiao-wei[1,2], FANG Ming[1,2], WANG Hao[1,2], WANG Sheng[1,2], WANG Xiang-qian[1,2]
(1. School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027;
2. Anhui Province Key Laboratory of High Performance Computing and Application, Hefei 230026)

【Abstract】 In order to solve the problems of difficulty and low efficiency in traditional parallel programming, this paper presents a parallel programming method based on MapReduce model, realizes matrix parallel LU decomposition under High Performance MapReduce(HPMR) platform. Experimental result shows that the parallel programs implemented via the MapReduce model can meet the need of high-performance numerical computing, and its programming simplicity and readability to enhance the efficiency of parallel programming compared with traditional parallel programming models.

【Key words】 High Performance MapReduce(HPMR); parallel programming; numerical computation; LU decomposition

MapReduce[1-2]是由 Google 提出的一种并行分布式编程模型。在 MapRedcue 模型中用户只须指定一个 map 函数来处理一个输入的 key/value 对，产生中间结果 key/value 对，再通过一个由用户指定的 reduce 函数来处理中间结果中具有相同 key 值的 value。借鉴 Hadoop[3]，以 Boost MPI 为基础实现了 MapReduce 系统——高性能 MapReduce(High Performance MapReduce, HPMR)[4]，它更适应于数值计算。本文分析矩阵的 LU 分解算法在 HPMR 上的应用，阐述运用 MapReduce 模型解决并行问题的方法，该方法清晰简洁。

1 MapReduce 与传统并行编程模型的比较

在传统并行编程过程中，程序员必须花大量的精力去处理进程间通信。WordCount[1]被用来统计输入数据中各单词出现的次数。以 WordCount 为例，MPI 的一种实现方式为：在各个 MPI 进程完成各自的统计任务后，将结果汇总给某一个进程，然后由该进程进行最终统计。该实现方式很容易在完成最终统计任务的进程处形成通信瓶颈，而最终统计任务是以顺序方式完成，会导致统计效率低下。采用并行分类统计能提升效率，即收集各个进程获得的关于某个相同单词的局部统计信息，并将这些信息汇总给某个进程处进行分类统计。然而由于这种实现方式仅确定能产生某个单词统计信息的进程范围，因此会增加编程复杂度。

在 MapReduce 模型下，完成单词统计的具体步骤为：

(1)用户编写 Map 程序对出现的单词 word 产生中间结果 key/value 偶对，如<word,1>。

(2)这些分布产生的中间结果将按 key 值的不同进行汇总处理，产生 key(word)相同的 value 列表，如<word,1,1,1,…>，并作为 Reduce 阶段的输入，这个阶段的工作将由 MapReduce 运行系统自动完成。

(3)用户提供的 Reduce 程序只须将获得的 value 累加即可得到结果。

Map 算法和 Reduce 算法如下[1]：

```
Map(String key, String value):
        // key: document name
        // value: document content
        for each word w in value:
                EmitIntermediate(w, “1”);
Reduce(String key, Iterator values):
        // key: a word
        // value: a list of counts
        int result=0;
        for each v in values:
                Result +=ParseInt(v);
        Emit(AsString(result));
```

图 1 是在 MapReduce 模型下的 WordCount 运行示意图。由上文可知，与传统并行编程模型相比，MapReduce 模型具有较高的并行表述抽象性。用户仅须提供 Map 和 Reduce 操作函数即可。与 MPI 类似，MapReduce 是一种显式数据划分的并行分布式编程模型。在 Map 阶段局部数据处理通常产生局部结果。用户可以借助 MapReduce 模型提供的 key/value

基金项目：核高基重大专项基金资助项目(2009ZX01034-001-001-002)；国家自然科学基金资助重点项目(60533020)；安徽省自然科学基金资助项目(090412068)
作者简介：郑启龙(1969－)，男，副教授，主研方向：并行与分布式计算；吴晓伟、房 明、王 昊、汪 胜、王向前，硕士研究生
收稿日期：2009-11-28 **E-mail：**qlzheng@ustc.edu.cn

策略，把对全局(或阶段性)计算结果有影响且有关联的 value 采用相同 key 标志。由于这种策略的简单抽象，因此用户可以较容易地把握局部和全局关系，从而确保问题求解并行实现的正确性，并进一步降低并行分布式编程的难度。

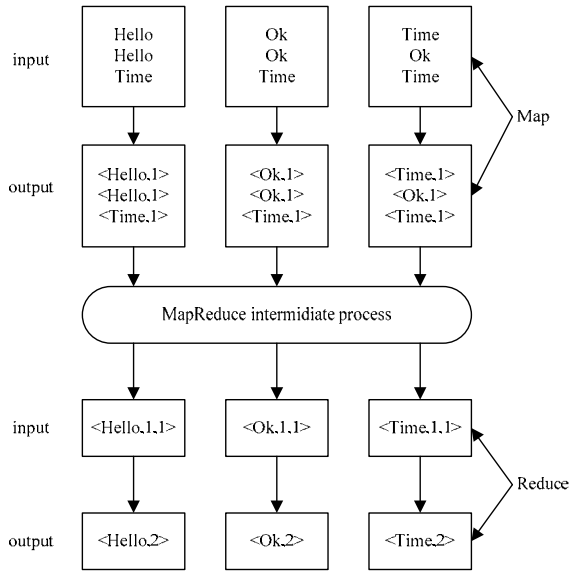

图 1　MapReduce 模型下的 WordCount 运行示意图

2　HPMR 在矩阵计算中的应用

2.1　HPMR 中矩阵类的设计

在矩阵并行计算前须对矩阵进行划分，即把一个大矩阵划分成若干子矩阵，每个子矩阵分配给不同进程或线程处理。在 HPMR 中，每个矩阵子块称为 sub_matrix 类，包含 2 个部分的内容：子矩阵描述和子矩阵数据。子矩阵描述包含划分相关的信息，如划分方式、总块数、本块块号、子块起始/终止行(或列)号等，以及和矩阵计算特征有关的信息，如 LU 分解中的主行行号。而子矩阵数据为实际存储子块数据地址的指针，如图 2 所示。

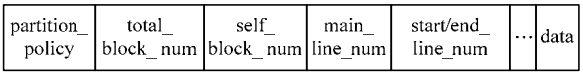

图 2　sub_marix 类

2.2　矩阵并行 LU 分解算法

2.2.1　算法分析与设计

LU 分解算法是把一个给定的 n 阶方阵 $\boldsymbol{A}$ 分解成一个下三角阵。在分解的过程中，主要计算是利用主行 k(k: 1~n)依次对其余各行 i(i>k)做初等变换，各行计算之间没有数据相关性，因此，可以对矩阵 $\boldsymbol{A}$ 按行划分实现并行计算[5]。LU 分解主要串行代码如下：

```
Begin
  for k=1 to n do      //主行选取
        for i=k+1 to n do          //消元计算
              a[i, k]=a[i, k]/a[k, k]
        end for
        for i=k+1 to n do
              for j=k+1 to n do
                    a[i, j]= a[i, j]-a[i, k]×a[k, j]
              end for
        end for
  end for
End
```

由于 MapReduce 依然属于显式数据划分与并行计算模型，因此须按照特定数据划分策略将待分解的矩阵划分为若干个子数据块，所用的划分方法可以是按行连续划分或按行交错划分等方式。然后根据 LU 分解串行计算的语义及划分方式，可以较容易地写出 Map 和 Reduce 处理过程，描述如下：

(1)Map：检查当前数据子块是否要使用当前主行进行变换。1)如果 Map 持有的数据子块不包含主行号 i 的数据行，且主行号 i 没有超越本地数据子块所持有行的行号范围，则产生<key=本地数据子块号#current_block，Val=本地数据子块>；2)如果 Map 持有的数据子块包含了主行号 i 的数据行，则产生<key=本地数据子块号#current_block，Val=本地数据子块>，然后针对所有其他需要变换的划分子块分别产生<key=其他数据子块号#other_block，Val=主行数据>。

(2)Reduce：对要变换的数据子块实施相应主行变换。1)对经过 MapReduce 运行时汇聚后的<key=数据子块号#block，Val_1=子块号为#block 的数据子块，Val_2=主行数据>中的数据子块进行相应主行变换；2)产生<key=数据子块号#block，Val＝变换后的子块号为#block 的数据子块(主行号增1)>作为下一轮 MapReduce 的输入。

图 3 显示了某轮 MapReduce 过程中 Map 产生的 key/val，其中，左边给出进行 LU 分解的矩阵划分，如子块 b_0, b_1 和 b_2；右边给出了 Map 产生的 key/val 偶对序列，b_0 因含有主行 i 而产生 4 个 key/val 偶对，而其余划分分别只产生 1 个 key/val 偶对。

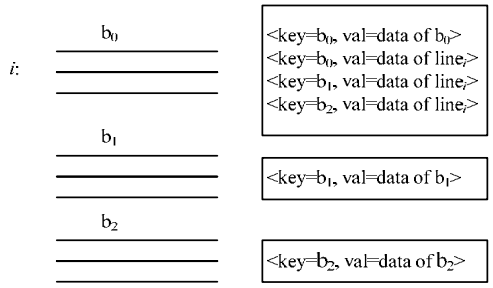

图 3　某轮 MapReduce 中的 Map 产生的 key/val 对

经过一轮 MapReduce 过程，完成了以某一行为主行的 LU 变换，对于 n 阶的方阵来说，完成整个 LU 变换的过程仅要进行 n-1 轮的变换。HPMR 为用户提供了控制迭代次数的接口，通过这些接口可以设置 MapReduce 循环的轮数和循环停止的条件[4]。

2.2.2　性能对比及分析

2 000 阶、4 000 阶、6 000 阶、8 000 阶矩阵 LU 分解分别如图 4~图 7 所示。

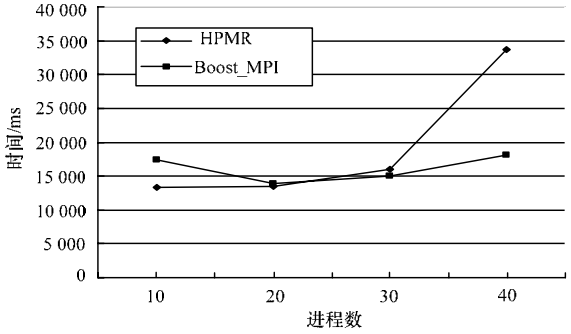

图 4　2 000 阶矩阵 LU 分解

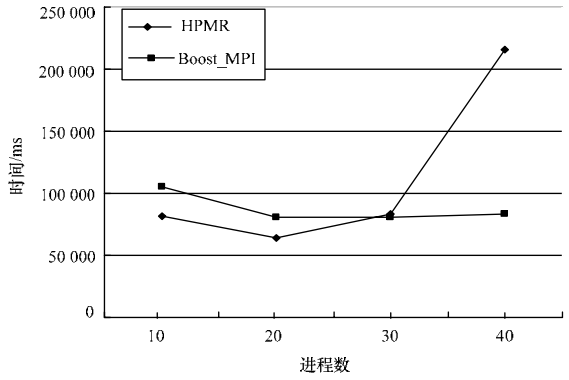

图 5　4 000 阶矩阵 LU 分解

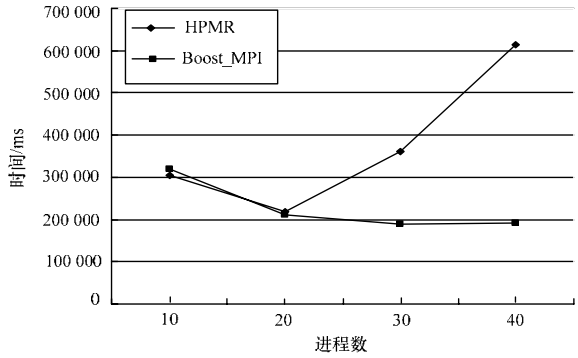

图 6　6 000 阶矩阵 LU 分解

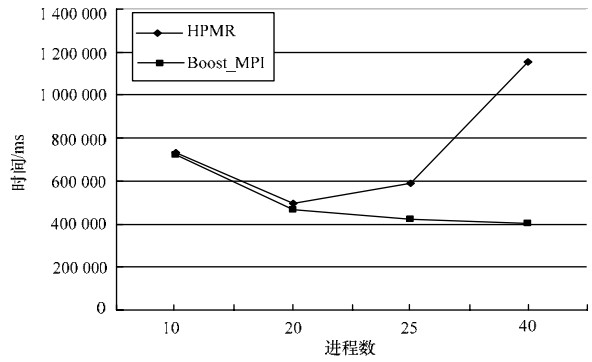

图 7　8 000 阶矩阵 LU 分解

由此可知，当进程数较少时，HPMR 的性能与 Boost_MPI 相当，当进程数为 10 时，HPMR 的性能略高。在 LU 分解中总有一个进程将主行发送给其他需要变换的进程，这种一到多的通信方式，在 Boost_MPI 实现的 LU 分解程序中是通过广播实现的，在 HPMR 中采用点到点的通信方式。根据本文测试，在进程数目较少的情况下，采用点对点通信方式实现的 LU 分解比采用广播方式实现的性能好。但当进程数目增多的时候广播方式的性能更好。当进程数目大于 30 时采用广播通信的 Boost_MPI 程序的性能比 HPMR 好得多，但是在进程数小于 25 时，HPMR 有很好的加速性能。

20 个进程的性能比较如图 8 所示，由此可知，在进程数为 20 时，随着矩阵规模的增大，HPMR 的性能接近于 Boost_MPI，但在进程数增多时，用点对点方式实现影响了一到多通信的性能。

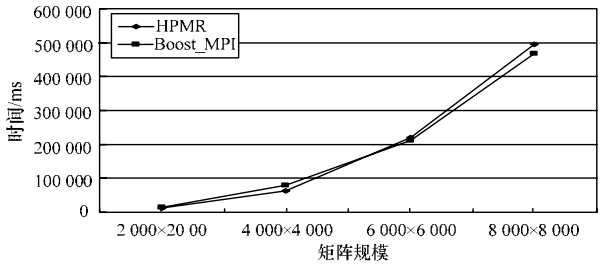

图 8　20 个进程的性能比较

2.2.3　MapReduce 时间复杂度分析

MapReduce 程序在每一轮的运行中都有 3 个过程：(1)Map 过程，时间用 t_m 表示；(2)中间结果的处理过程，时间用 t_p 表示；(3)Reduce 过程，时间用 t_r 表示。数据分发过程所需时间用 t_d 表示，最后结果收集所需时间用 t_q 表示，整个 MapReduce 过程的迭代次数用 n 表示，整个过程所用的时间用 T 表示。

使用大同步并行(Bulk Synchronous Parallel, BSP)模型[6]对 MapRedcue 进行分析。MapRedcue 一个超级计算步的成本为

$$T_i = \max\{t_{mi}\} + \max\{t_{ri}\} + t_p + L$$

其中，L 为每轮之间的路障同步时间。整个 MapRedcue 程序所需时间为

$$T = t_d + \sum_{i=1}^{n} t_{mi} + \sum_{i=1}^{n} t_{ri} + \sum_{i=1}^{n} (t_p + L) + t_q$$

由于 t_m 和 t_r 是数据处理过程所需要的时间，与具体应用的算法有关，在系统中很难进行优化。t_p 除了同步外，key 的统计和 key/value 对的通信占了很大比例。因此，设计合理的中间结果处理过程是减少 t_p，提高整个系统运行效率最有效的途径。

3　结束语

作为一种新型的并行分布式编程模型，HPMR 具有较高的并行表述抽象性，可有效地降低并行编程的难度，提升并行编程生产率。本文下一步工作是将该模型引入到更多的数值/非数值高性能计算领域，并重点优化 HPMR 运行时系统，使其具有更好的运行性能。

参考文献

[1] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[C]//Proc. of OSDI'04. San Francisco, USA: [s. n.], 2004.

[2] Lammel R. Google's Mapreduce Programming Model-revisited[J]. Science of Computer Programming, 2008, 7(1): 208-237.

[3] Kurdyumov A. HadoopMapReduce[EB/OL]. (2008-06-01). http://wiki.apache.org/hadoop/HadoopMapReduce.

[4] 郑启龙, 王　昊, 吴晓伟, 等. HPMR: 多核集群上的高性能计算支撑平台[J]. 微电子学与计算机, 2008, 25(9): 21-23, 27.

[5] 陈国良, 安　虹, 陈　崚, 等. 并行算法实践[M]. 北京: 高等教育出版社, 2004.

[6] 陈国良. 并行算法的设计与分析[M]. 北京: 高等教育出版社, 1994.

编辑　陆燕菲