HPMR 内存管理模块优化设计①

郑启龙，汪 睿，王向前

(中国科学技术大学 计算机科学与技术学院，合肥 230027)
(安徽省高性能计算重点实验室，合肥 230026)

摘 要：HPMR 系统是一个采用 MapReduce 模型的高性能计算软件支撑平台，它改进了 MapReduce 模型以适应高性能计算的需求。高效的 HPMR 系统内存管理模块是保证其效率的重要模块。HPMR 系统中有两个角色，Master 和 Worker。Master 负责从输入数据文件中读入数据块并分配给 Workers。Worker 负责接收 master 分配的数据块、管理 map 函数的输入输出模块的输入输出数据、管理 reduce 函数输入输出数据。目前的内存管理模块存在管理通信冗余、管理低效、数据处理并行不足等缺点。本文根据成熟的内存优化理论，重新设计了 HPMR 底层的数据管理机制，提出了基于内存池的内存管理。实验表明，新的内存管理模块是保证 HPMR 系统高效的必要条件。

关键词：MapReduce; HPMR; 内存池；数据管理

Optimized Design for Memory Management Module in HPMR

ZHENG Qi-Long, WANG Rui, WANG Xiang-Qian

(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
(Anhui Provincial Key Laboratory of High Performance Computing, Hefei 230026, China)

Abstract: HPMR is a high performance computing platform based on MapReduce model. It has improved the MapReduce model to meet the need of high performance computing. Efficient memory management module ensures the efficiency of HPMR. There are two roles in HPMR, Master and worker. Master reads data blocks from the input data file and assign them to workers. Worker receives the data blocks from master, manages input and output module of the map and reduce function. The current memory management module in HPMR has some shortcomings: redundancy, inefficiency and lack of parallelism. This paper redesigned the underlying data management mechanism of HPMR based on mature memory optimization theory, proposed new memory management way based on memory pool. Experiments show that the new memory management module is necessary for efficient HPMR system.

Key words: MapReduce; HPMR; memory pool; data management

1 引言

1.1 MapReduce 并行编程模型

MapReduce[1-3]是由 Google 于 2004 年提出的并行编程模型，它被用在处理和产生海量数据集的实现中。MapReduce 模型(简称 MR 模型)的优点有：简单易学、编写串行程序自动并行运行、高度抽象性等。

MR 模型的高度抽象性体现在三个核心概念：map 函数、reduce 函数和键值对<key,value>(简称 KV)。下面阐述了这三个核心概念各自的功能：

(1) map 函数的功能：执行 Map 任务，处理输入的<key,value>，处理结果以新的<key,value>形式表现，处理过程称为“分组”，即要使在 Reduce 任务中具有操作相关性的 value 对应相同的 key。

(2) reduce 函数的功能：执行 Reduce 任务，读入<key,list(value)>，每个 reduce 函数处理的对象是对应同一 key 的 value 集合，处理结果以新的<key,value>形式表现，处理过程称为“合并”。

(3) 键值对<key, value>的功能：<key,value>有两

① 基金项目:工信部核高基重大专项(2009X01028-002-003);安徽省自然科学基金(090412068)
收稿时间:2010-12-08;收到修改稿时间:2011-01-15

部分组成，value 代表与任务有关的数据，key 代表 value 的“分组编号”，也就是指明 value 需要被放在哪个“分组”中，参与接下去的运算过程。

MR 模型具有非常广泛的适用性。本文中把一次 Map-Reduce 过程简称为一次 MR 过程，也称一轮 MR 过程,简称一轮。MR 过程既包含了任务并行，又包含了数据并行。MR 过程代表了“数据分组—数据收集—数据处理”一整套过程，任何并行程序都可以分解成一个或多个 MR 过程。MR 模型可以解决绝大多数并行编程问题[4]，支持多种并行计算机体系结构和并行算法结构。

1.2 HPMR 系统

HPMR(High-Performance MapReduce)[5]是本实验室为了推广 MR 模型而开发的面向高性能计算的 Map Reduce 系统。它为用户编辑、编译、运行和调试 MapReduce 程序提供多方面帮助，支持大规模计算的任务分配和自动并行。

HPMR 使用两级 key(key1 和 key2)代替 Google 的 MR 模型中的单 key。在原 MR 模型中，key 的意义是 value“组”的组号，在 HPMR 的 MR 模型中，key1 是“大组”的组号，key2 是“小组”的组号，“大组”中包含着若干“小组”。

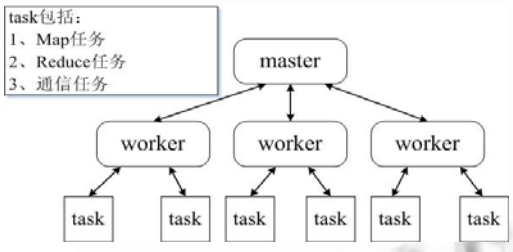

图 1 master 和 worker 的关系

HPMR 平台上程序运行时需要两种角色，master 和 worker，两者关系如图 1 所示，master 负责管理所有 worker，worker 负责执行子任务，包括 Map 任务、Reduce 任务和通信任务。

目前 HPMR 系统数据管理模块存在数据拷贝冗余、管理低效、数据处理并行不足等缺点。故重新了设计了基于内存池的数据管理模块。

本文其余部分组织方式如下:首先介绍了目前 HPMR 系统的数据管理模块及缺点（第 2 节），然后具体讨论引入内存池管理机制的数据管理模块的设计（第 3 节）以及在新的数据管理模块下 HPMR 系统的涉及到一些细节问题（第 4 节）。接着以实例测试验证新的内存管理机制的优化效果（第 5 节）。最后是本文的结论（第 6 节）。

2 HPMR的数据管理模块

2.1 HPMR 的数据管理模块的功能

数据管理模块主要负责管理 workers 执行计算工作时所需的数据，包括输入/输出数据和中间结果数据。数据管理是连接任务管理与通信管理的媒介，保证数据在两个模块之间有序、有组织地流转。

master 数据管理模块负责：

(1) 从输入数据文件中读入数据块。

(2) 调用数据块分配算法，把数据块分配给各个 workers。

(3) 把 MapReduce 程序运行的输出结果写入输出文件中。

worker 数据管理模块负责：

(1) 接收 master 分配的数据块。

(2) 管理 map 函数的输入数据和输出数据。

(3) 管理 reduce 函数的输入数据和输出数据。

(4) 为通信模块提供待发送的数据(送给其他 workers)；接收并存储来自通信管理模块的数据(来自其他 workers 或 master)。

因为输入数据的划分方法与具体的 MapReduce 程序密切相关，所以 HPMR 不会对 MapReduce 程序的初始输入数据进行自动分割。用户需要根据程序要求，手动把初始输入数据分割成若干数据块(Data Block)，存储在输入文件中。

worker 的数据管理模块管理着两个数据“容器”：一个是存储数据块的 Block Container，另一个是存储<key1,key2,value>的 KV Container。map 函数的输入数据和 reduce 函数的输出数据(将作为下一轮 map 函数的输入数据)保存在 Block Container 中。map 函数的输出数据和 reduce 函数的输入数据(也就是经过通信阶段之后的<key1,key2,value>数据)保存在 KV Container 中。

2.2 HPMR 数据管理的缺点

HPMR 的数据管理模块的数据流动如图 2 所示，Master 从输入文件(程序员已经按照编程逻辑把数据块分好)依次读取数据块，然后插入到 block 列表中，

Master 调用数据分块算法，把数据分配给 Workers，最后把属于每一个 Worker 的 block 子表按照 block 依次发送出去。

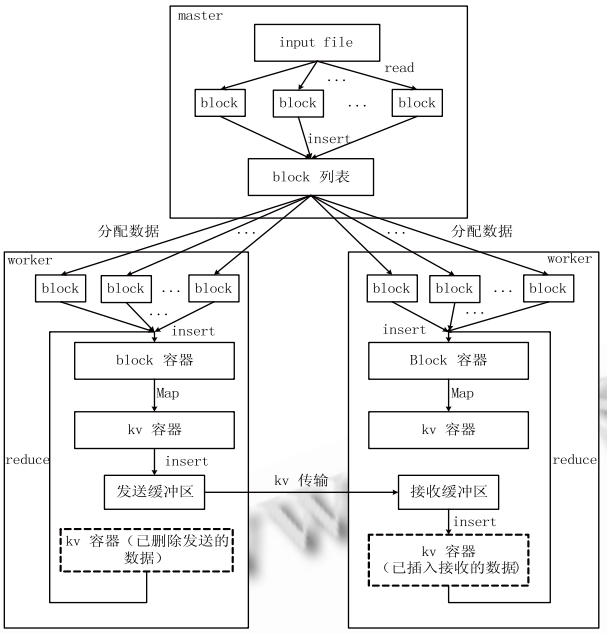

图 2　数据管理模块的数据流动示意图

Worker 把依次接收到的 block 块插入到 block 容器里，然后把 Block 容器里的数据作为 map 过程的输入，map 过程产生了新的 kv 数据插入到 kv 容器中。经过 kv 路由过程(未在本图中画出)，worker 在路由表的指导下进行 kv 传输。本图中左边的 worker 是发送方，右边的 worker 是接收方。左边的 worker 把需要发送的数据拷贝到发送缓冲区，然后从 kv 容器删除，右边的 worker 作为接收方先接收数据到数据缓冲区，然后把缓冲区里的数据插入到 kv 容器中。接着所有的 workers 以 kv 容器的数据为输入，进行 reduce 过程，reduce 过程产生新的 block 块，插入到 block 容器中。进入下一轮的 Map-Reduce 过程。

目前的数据管理模块的数据拷贝过程频繁，依次为 master 上的(1)读取 block 数据，(2)插入 block 数据块到 block 列表，(3)并发送分配给 Worker 的 block 数据, 和 Worker 上的(4)接收 block 数据块,(5)插入 block 数据块到 block 容器中，(6)插入 map 过程新产生的 kv 数据到 kv 容器中，(7) 插入需要发送出去的 kv 数据到发送缓冲区 (8) 接收 kv 数据到接收缓冲区(9)把接收缓冲区的数据插入到 kv 容器里(10)把新产生的 block 块插入到 block 容器中。进入到下一轮的 Map-reduce 过程的数据拷贝过程。可以看到，(1)、(3)、(4)、(7)这几步的数据拷贝是必需的，而(2)、(5)、(6)、(8)、(9)、(10)这么多步的数据拷贝不是必要的。如果进行很多轮 Map-Reduce 过程，则不必要的数据拷贝次数更多，数据拷贝开销更大。

3　基于内存池的内存管理模块

3.1　内存池基本数据结构

为了高效的管理 MapReduce 内部数据，我们采取在 Mapreduce 进行初始化时就对主要数据分配成一个内存块，该内存块主要有内存单元组成，一个内存单元可以容纳该数据类型的一个变量。在程序运行过程需要生成该数据类型的变量时就从这个内存块取一内存单元使用，如果在使用完毕后，选择合适的时机还给内存池。如果一个内存块不能满足程序的需要，则再统一申请同样大小的内存块。多个内存块就构成了内存池。如图 3 所示，Blocklist 是内存块链表的头指针，是内存池的代表。Head 是目前可用内存单元链表的头指针，代表当前内存池可用的内存单元[6]。

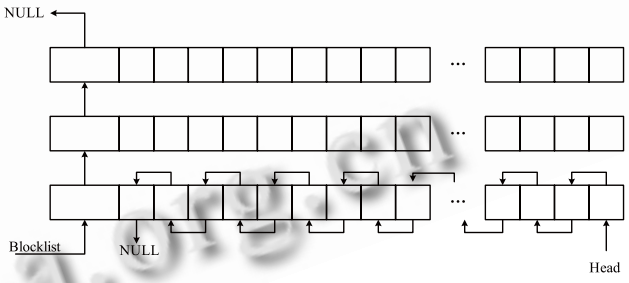

图 3　内存池示意图

3.2　基于内存池的数据容器

由 2.2 节的分析，可知 HPMR 的内存数据管理非常低效。有必要把高性能 Mapreduce 的数据管理模块建立在内存池的基础上。HPMR 的两个容器是 Block Container 和 KV Container。在基于内存池的设计中，可以使用链表和有序链表来取代 Block Container 和 KV Container。如图 4 所示。这样，HPMR 的两个容器都是基于内存池的统一的链表，区别在于 reduce_block_sortedlist 是基于 key1 和 key2 的有序链表，而 map_block_list 则不需如此。

● 基于内存池的 block 块的构建时机：

(1) master 从输入数据文件中读入数据块时。

(2) 当 worker 从 master 接收分配的 block 块时。

(3) 当 worker map 过程需要产生新的 kv 数据时。

(4) 当 worker 从其它 worker 接收 kv 数据时。

(5) 当 reduce 需要产生新的 block 块时。

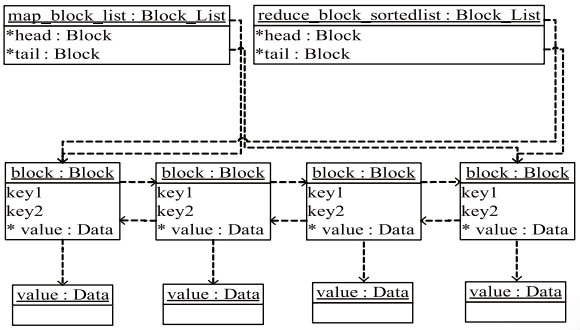

图 4 基于内存池的数据容器

● 基于内存池的 block 块的构建步骤：

(1) 从 Block 内存池的 Head 空闲链表取下一个内存单元，初始化，记作 block。

(2) 从Data内存池的Head空闲链表取下一个内存单元，记作 value,初始化。

(3) block 的 value 指针指向 value。

● 基于内存池的 block 的发送接收过程：

(1) 发送方：先用一个临时指针指向当前 block 的 value，然后置 block 的 value 为 NULL。先发送 block，再发送当前 value。然后 block 的 value 指针指向当前 value。

(2) 接收方：按照 block 块的构建步骤依次接收 value 指针为空的 block 块和 value，构建 block 块，然后按照 key1、key2 的层次按序插入 reduce_block_sortedlist 链表。

● 基于内存池的 block 释放时机

(1) Master 为 worker 分配完 block 时，统一释放 block 链表。

(2) Worker 在 map 阶段后，统一释放 map_block_list。

(3) Worker 在 kv 路由传输阶段向其它 worker 发送完发送 KV 数据，先从 map_block_list 删除，从把该数据释放到内存池中。

(4) Worker 在 reduce 阶段后，统一删除 reduce_block_sortedlist。

● 基于内存池的 block 链表的释放步骤：

(1) 从 block 链表中取下一个 block 块。

(2) 先释放该 block 块的 Value，即把 Value 所占的内存单元还给 Data 内存池的 Head 空闲链表。

(3) 再释放 block 本身，即把 Block 所占的内存单元还给 Block 内存池的 Head 空闲链表。

从 Data 内存池取下一个内存单元，记作 value,初始化。block 的 value 指针指向 value。

4 新的数据管理模块下HPMR的构建

4.1 多种 Data 类型的 block 块的构建

在 Mapreduce 过程中，会经常涉及到占用不同内存大小的 Value，如果只用一种 Data 类型，则会极大的浪费内存空间。可以对图 3 的内存池容器进行扩展，如图 3 所示。该图表示了是表示在 Mapreduce 过程会产生两种 Data 类型的 Value。map_block_list 和 reduce_block_sortedlist 的 block 类型基于此经进行扩展，即可满足 Mapreduce 的需要[7-8]。

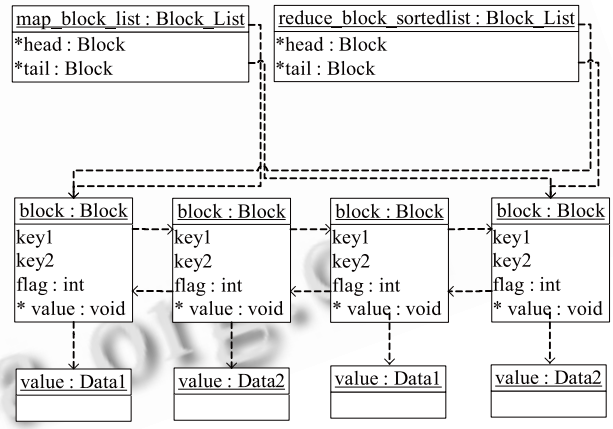

图 5 基于内存池的多种 Data 类型 Value 数据容器的扩展

4.2 Map 和 Reduce 的实现模式

● Map 过程

输入：map_block_list

输出：reduce_block_sortedlist

处理过程：依次遍历 map_block_list，针对每一个 block，产生 KV，有三种情况：

(a) 所要产生的 KV 的 Value 类型与当前 block 一致，则修改 block 的 key1,key2 ，将当前 block 块按照 key1、key2 的层次按序插入到 reduce_block_sortedlist，并从 map_block_list 中删除该 block。

(b) 所要产生的 KV 的 Value 类型与当前 block 不一致，则构建新的 Value 和 block 块，修改 block 的 key1,key2 ，将构建的 block 块按照 key1、key2 的层次按序插入到 reduce_block_sortedlilist。

(c) 对不需要产生 KV 的情况的当前 block，不做处理。

执行 map 后统一释放 map_block_list 到内存池。在进行 KV 路由之后进行 KV 传输，即按照基于内存池的 block 的发送接收过程执行。之后 reduce_block_sortedlist 基于 key1 和 key2 的有序链表。

● Reduce 过程

输入：reduce_block_sortedlist

输出：map_block_list

处理过程：针对 reduce_block_sortedlist，在 key1 相同的情况遍历 key2 对应 block，进行相应的运算，有三种情况：

(a) 所要产生 block 可以使用的当前 KV block 块，则进行运算之后 ，将当前 KV block 块按照插入到 map_block-list，并从 reduce_block_sorted list 中删除该 block。

(b) 所要产生 block 不可以使用的 block，则从内存池中构建新的 Value 和 block 块，进行运算之后 ，将当前 block 块按照插入到 map_block-list。

(c) 对当前 KV 数据不需要做处理。

然后统一释放 reduce_block_sortedlist 到内存池中。进入到下一轮 Mapredce 过程。

4.3 基于内存池数据模块的 HPMR 的数据拷贝分析

基于内存池的数据流动如图 6 所示。与图 2 对比，可知，只有图 2 中必需的(1)、(3)、(4)、(7)这三步的数据拷贝存在基于内存池管理模块的 HPMR 中。而(2)、(5)、(6)、(8)、(9)、(10)这么多步的数据拷贝不必要的拷贝则完全省去。可以预测基于内存池数据模块的 HPMR 的高效性。

4.4 基于内存池的数据模块拷贝处理时间复杂度分析

假定一个程序需要 N 轮 MR 过程才能完成，且单位拷贝时间为 T(单位拷贝时间即为拷贝总数据一次所占用的时间)，总共有 W 个 worker 和 1 个 master,并忽略 KV 传输的数据拷贝量。在原来的数据管理模块的时间复杂度 Copy_Time 估算为：

$$Copy\_time=3*T+4*(T/W)*1+2*(T/W)*(N-1)$$

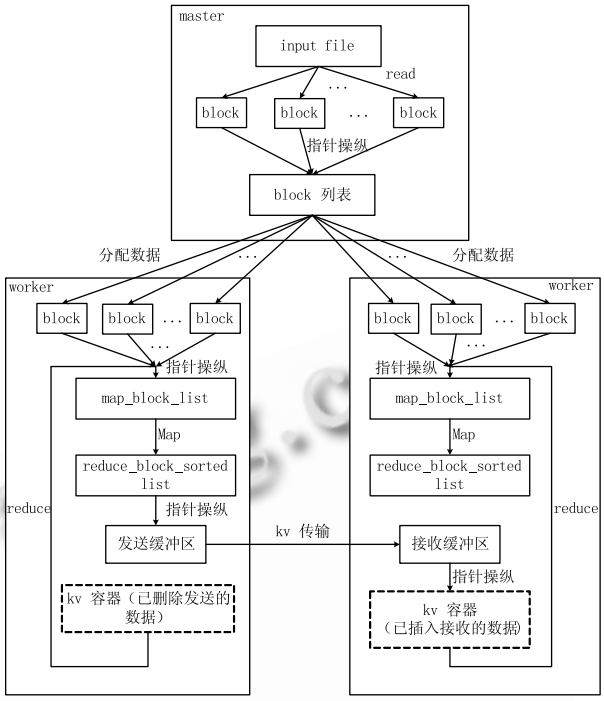

图 6　新的数据管理模块的数据流动

基于内存池的数据管理模块的时间复杂度为:

$$Memory\_Pool\_Copy\_time=2*T+2*(T/W)*1+1*(T/W)*(N-1)$$

基于内存池的数据管理模块的引入对数据管理模块的数据拷贝处理所达到的百分比记作 E，则：

E=(Copy_Time-Memory_Pool_Copy_time)/Copy_Time=(3*T+4*(T/W)*1+2*(T/W)*(N-1))-(2*T+2*(T/W)*1+1*(T/W)*(N-1))/ (3*T+4*(T/W)*1+2*(T/W)*(N-1))
=(1*T+2*(T/W)*1+1*(T/W)*(N-1))/(3*T+4*(T/W)*1+2*(T/W)*(N-1)

当 1<<N 时，

E ≈ 50%

可见，引入基于内存池的数据管理模块，和旧的数据管理模块，提高效率趋于 50%。

5　推测执行技术下的通信性能

5.1 测试平台

HPMR 使用国家高性能计算中心（合肥）的曙光 TC4000A 做为硬件测试平台，这台高性能计算机由 42 个计算节点、1 个登录节点和 1 个控制节点组成，它的原则峰值达到 26752 亿 flops。每个计算节点的配置

如下：

CPU：双 AMD Opteron 2347（4 核，主频 1.9GHz）

内存：4GB

硬盘：160GB SATA

计算网络：1000M 快速以太网

操作系统：Red Hat AS 4.6 for Linux

MPI 系统：MPICH2-1.0.8

5.2 内存池管理数据模块的优化效果

方腔中温度传播的数值模拟[9]是用程序模拟温度在一个封闭方腔中的传播过程，方腔被分成 1000010000 的网格，然后平均分成 400 块 1000025 的网格，交给 40 个进程（40 个 workers）。图 7 的实验条件：方腔被分成 5000×5000 的网格，然后平均交给 40 个进程（40 个 workers）。图 8 的实验条件：方腔是被分为 5000×5000 的网格，MR 过程次数为 500。

从图 7 可以看出，使用内存池的数据管理模块(记为 memory pool)和没有使用内存池优化技术的数据管理模块(记为 no memory pool)相比，随着 MR 过程次数的增加，引入内存池的数据管理模块所达到的效率 E 越来越趋近于 50%。这和理论分析是一致的。从图 8 可以看出，随着参加进程数目增多，引入内存池所达到的效率 E 也在 50%左右，这也是符合理论分析结果的。可见，基于内存池的数据管理模块所达到的效果是比较理想的。

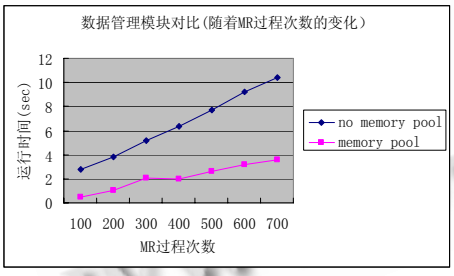

图 7 数据管理模块时间复杂度随着 MR 过程次数的变化

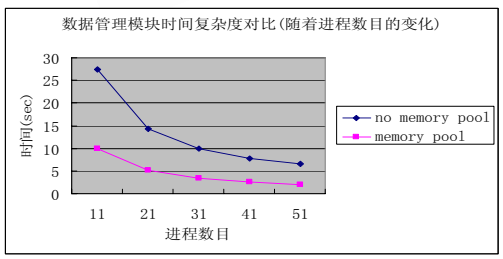

图 8 数据管理模块时间复杂度随着进程数目的变化

6 结语

本文提出基于内存池的新数据管理模块，并在此基础上详细论述了在新数据模块的基础上构建 HPMR 的细节，最后以温度传播为例验证了它的新的数据管理模块通的效率。实验表明，基于内存池的数据管理模块是构建高效 HPMR 的必要条件[10]。

参考文献

1 Dean J, Ghemawat S. MapReduce:simplified data proc. on large clusters. 6th Symposium on Operating Systems Design and Implemention. San Francisco, California, USA, 2004: 1–13.

2 Lämmel R. Google's MapReduce programming model – revisited. Science of Computer Programming Journal, 2006. 1–42.

3 Ranger C, et al. Evaluating MapReduce for Multi-core and Multiprocessor Systems. Proc. of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, 2007. 13–24.

4 王昊.多核集群上的高性能 MapReduce 平台的研究与实现[硕士学位论文].合肥:中国科学技术大学,2009.9–14, 39–40.

5 郑启龙,王昊,吴晓伟,房明.HPMR:多核集群上的高性能计算支撑平台.微电子学与计算机,2008,25(9):21-23.

6 Renau J, Strauss K, Ceze L.Thread-Level Speculation on a CMP can be energy efficient. Proc. of the 19th Annual International Conference on Supercomputing, 2005. 219–228.

7 Josuttis NM. The C++ Standard Libaray: A Tutorial and Reference. Addison Wesley Longman, Inc, 1999. 171–190.

8 Cormen TH, et al. Introduction to Algorithms. 2nd Ed. The United States: The MIT, 2001.chapter 13.

9 Rolando A, Chávez.C. Natural-convection Heat Transfer in Supercritical Fluids [MS Thesis]. Mechanical Engineering University of Puerto Rico Mayagüez Campus, 2004.

10 陈国良,等.并行算法实践.北京:高等教育出版社,2004. 455–498.