关于多核系统并行程序效率的编程因素及其研究

王文义 冉晓龙
(中原工学院并行处理技术研究所 郑州 450007)

摘 要 着重分析了多核架构系统中内存对齐技术与cache利用率等因素对并行程序性能的影响。用共享存储环境OpenMP分析了并行计算量与处理器核心数目之间的关系,通过用MPI编程实现的矩阵相乘的行划分和CANNON算法等实例分析,指出了只有综合考虑了多核系统的结构特征、系统软件、多核编程语言环境以及正确运用算法等,才能设计出高效且能耗又小的并行应用程序。

关键词 绿色计算,内存对齐,OpenMP,CANNON算法,多核处理器
中图法分类号 TP302 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.8.005

Programming Factors about Efficiency of Parallel Program in Multi-core System and its Research

WANG Wen-yi RAN Xiao-long
(Institute of Parallel Processing Technology, Zhongyuan Institute of Technology, Zhengzhou 450007, China)

Abstract This paper emphatically analyzed the effects of the memory alignment technology, cache using ratio and such as these factors on parallel program performance in the multi-core architecture system. This paper used shared memory environment OpenMP to analyze the relationship between parallel amount of calculation and the number of the processor core. The results of experiment, which were programmed by MPI to implement the row-divided algorithm and CANNON algorithm of matrix multiplication, point out that only in the comprehensive consideration of the architecture feature of multi-core system, system software, multi-core programming language environment and the correct use of algorithm can we design a better parallel application—being capable of high efficiency and small energy consumption.

Keywords Green computing, Memory alignment, OpenMP, CANNON algorithm, Multi-core processor

1 引言

随着全球气候变化的严峻发展形势,计算机的能耗问题已经紧迫地提上了议事日程。由于计算机几乎已经普及到了世界的每个角落——从各国超算组织成千上万台的巨型并行系统、高性能集群系统,到各行各业的桌面电脑,一直到遍布每个家庭甚至每个人的微芯片产品,都不得不对它们自身的功耗指标(Mflops/W)提出近乎苛刻的要求。从硬件上看,计算机处理器由原来的单核高主频逐渐发展到现在的多核低主频,从而大幅度地降低了计算机功耗,而在应用型并行软件设计上,也由原来的单核处理器编程,演变到现在的多核处理器编程[1]、多机编程,以及分布式处理编程等等,所有这些都是围绕着一个主题展开,即如何能有效利用计算机芯片的核结构和分布式资源的特点,来达到管控和提高设备的运行效率即节能减排的目的。

本文从应用软件角度出发,侧重考虑在并行应用程序设计中如何提高数据存取速度以及如何提高计算性能(注:文中各处出现的如并行程序性能、系统有效速度、计算性能和运算效率等,尽管字面表述或略有差异,但核心都只有一个,即应用端一方究竟怎样才能获得多核系统理想的解决问题速度)等,并做了一些实践工作,在此写出来,以飨读者。

2 数据对齐存取

2.1 数据内存对齐存取

数据在内存中的存放位置并不是想当然的可以随意存储,而是要受到一定限制并遵守一定规则的。比如说可以按照双字节、四字节或八字节等形式存储,通常称这些存取单位为内存存取粒度。如果数据不对齐,可能会造成运行速度变慢,甚至毫无征兆的出错而不易检测等现象。下面将以实例说明这个问题对程序性能造成的影响。

内存对齐主要是针对结构体[2]的,而结构体中有不同的数据类型,比如整型、字符型、双精度型和字符串数组等。把这些不同的数据类型进行内存边界对齐,可以减少cache与内存的数据交换次数。

alignment16函数代表一次处理2个字节(文中只给出这种形式,其它两种形式略),然后对其运行时间进行分析。工作是对一个10MB的缓冲区进行读取、取反并写回数据,实验结果如图1所示。

到稿日期:2014-08-09 返修日期:2014-10-26 本文受国家自然科学基金项目(61379079),河南省基础与前沿技术研究项目(082300410300)资助。

王文义(1947—),男,教授,硕士生导师,主要研究方向为并行处理技术与高性能计算,E-mail:wwy@zzu.edu.cn;冉晓龙(1987—),男,硕士生,主要研究方向为多核程序设计技术。

由图1可以看出，单字节存取速度近乎是一条直线，是3个函数中最慢的。对齐的双字节比非对齐的双字节速度快了近40%，且非对齐的双字节与单字节存取速度相近。对齐的四字节速度是最快的，非对齐的四字节速度比对齐的双字节速度稍慢些，但比非对齐的双字节要快。

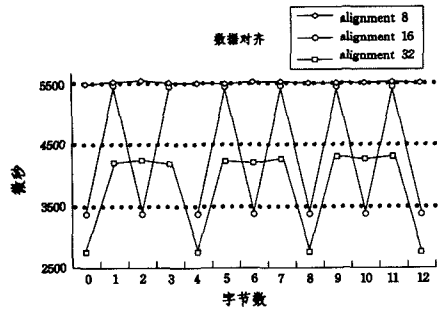

图1 单字节-双字节-四字节的存取速度

按两字节对齐的主要代码如下：

```
void Munge16(void * data,uint32_t size)
{
  uint16_t * data16 = (uint16_t * ) data;
  uint16_t * data16End = data16 + (size >> 1);/ * Divide size
  by 2. * /
  uint8_t * data8 = (uint8_t * ) data16End;
  uint8_t * data8End=data8 + (size & 0x00000001);/ * Strip up-
  per 31 bits. * /
  while(data16 != data16End)
    {
      * data16++ = - * data16;
    }
  while(data8 != data8End)
    {
    * data8++ = - * data8;
    }
}
```

对规模较小的程序，该问题意义还不太明显，但尚若面对的是一个服务器级程序，要运行大量的线程和面对海量用户的使用，这时对服务器的存取将会达到极高数量级，于是内存不对齐的情况就会对程序性能带来较大的影响。这种情况应足以引起重视。

2.2 数据 cache 对齐存取

实践证明，对大多数高性能科学计算课题来说，计算机的运算性能大部分都被消耗在对大容量矩阵运算的处理上。因为动辄是成百上千兆字节的矩阵规模，它们是无法一下子被整个放进高速存储器中以完成计算的。在理想情况下，当然希望在运算进行时数据均已到达 cache 而无须等待。所以可以说，cache 利用程度的高低能够近似反映出计算机有效速度的大小。

2.2.1 cache 利用率 CUR

可测量（measurable）CMR（Cache Missing Ratio）代表 cache 缺失率且：

CMR=（cache 缺失时间/CPU 时间）* 100%

而且在 cache 缺失延迟情况下，计算机的有效速度可以近似地表示为：

$$V_e \approx V_P * (1-CMR) = V_P * CUR \quad (1)$$

其中，V_e、V_P 分别代表处理器的有效速度和峰值速度；基于当处理器的运算单元开始工作时，运算数应该存在于 cache 之中的常识，可以把 CUR（Cache Using Ratio）理解为 cache 中平均每个数据参加运算的次数，即

$$CUR = \text{指令执行总次数/访问数据的总个数} \quad (2)$$

即在 cache 中，数据参与运算的总次数与 CUR 成正比，运算次数越高，计算机的有效速度也就越高。

2.2.2 矩阵运算模型的 CUR

假定矩阵 A、B、C、E 规模为(n,n)，向量为 $D(n)$。

（1）矩阵与向量乘：$E=A*D$

由式（2）知

$CUR=2n^2/(n^2+2n)$，当 $n\to\infty$ 时，$CUR\approx 2$

（2）矩阵与矩阵乘：$C=A*B$

$CUR=2n^3/(3n^2)=2/3*n$

可见上述两种模型都具有较高的 cache 利用率。所以只要在先期的并行程序设计中能够把巨型数组分解成合适与 cache 对齐的尺寸，就有利于让编译器把数据地址定位到 cache[3]，这无疑会明显提高计算机的有效速度。

2.2.3 矩阵计算实例

本文计算平台为 InfiniBand 集群，用 2 个 Intel core i7 870，2.93GHz，四核八线程，8GB 内存的处理器。每个处理器的理论峰值速度为：

V_P =2.93(GHz) * 4(核) * 4(每周期指令数)

=46.88Gflops

处理器结构如图2所示。缓存采用三级结构，其 L3 采用独特的全包含式（Full Inclusive）设计，可被片上所有核心共享，容量为 8MB。这样的缓存容量，显然在并行程序设计中不能再对它熟视无睹了。

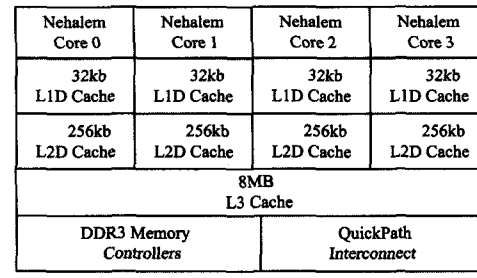

图2 Nehalem Core i7 高速缓存层次架构

使用线性代数通用程序库 HP VECLIB 代码进行测试（篇幅所限，细节略），对矩阵规模 3000 * 3000，向量 3000 测得的结果如表1所列。

表1 两种计算模型的实验结果

计算模型	含义	CMR	有效速度/处理器
E=A*D	矩阵向量乘	45.3%	25.64Gflops
C=A*B	矩阵矩阵乘	23.6%	35.82Gflops

从表1可以看出，所获得的有效速度较之一般并行系统，通常只有30%左右峰值速度的效率已经是很好了。

除了上述因素外，在多核架构系统的硬件选择上，为提高系统的并行运算效率，还应注意硬件是否设置有整数处理器以增加予取（Prefetching）功能，从而保证在总体上增加通信带宽。

3 并行编程环境选择

3.1 基于 OpenMP 的并行计算量与核心数间的关系

OpenMP[4-6] 是被广泛接受的共享存储编程工具，其特点

是它可对应用程序进行分段并行化，从而实现程序的高效运行。比较典型的情况是 for 循环[7]的并行化，由于 for 循环要耗费大量的程序运行时间，若能实现并行化，将可以节省大量的运行时间。下面分 3 种情况进行讲论。

(1)对于小型 for 循环，一般体现不出并行化的优势。通过运行下面的实验代码，可以得到的串行与并行结果的比较，如图 3 所示。

```
void test ( )
{
  for(int i=0;i<10000;i++)
  {
    //功能计算部分;
  }
}
# pragma omp parallel for
  For (int i = 0;i<100;i++)
    {
      test ( );
    }
```

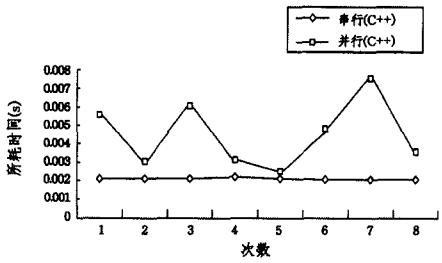

图 3　串行与并行运行的时间对比

由图 3 可以看出：串行的执行时间近乎保持一条直线，似乎与执行次数无关。而并行运行时间则是在串行所示直线之上上下浮动，呈现出一种跳跃性的忽高忽低的结果，并且时间消耗较之串行情况反而增加，几乎不曾出现小于串行的情况。这是因为计算规模太小(for 循环的次数)，并行化后的耗时会大于串行是因为对于 OpenMP 环境，并行化后计算机尚需要额外对处理器进行分配和调度以及创建并行线程等，而所有这些操作都需要时间，所以当这种时间代价大于处理器计算的任务量时，并行化的效果反而不理想。

(2)当计算(任务)量逐渐增加时，即 N 从 1 逐渐递增，从而控制并分配到每个核心的负载量为一个 compute02，两个 compute02，……等(由 OpenMP 的 for 循环并行机制决定)，主要代码和实验结果(见图 4)如下所示：

```
void compute02()
{
  for(int i=0;i<5000000;i++)
  {
    for(int j=0;j<1000;j++);
  //功能计算部分;
  }
}
#pragma omp parallel for
  for(int i = 0;i < N;i++)
  {
    compute02();
  }
```

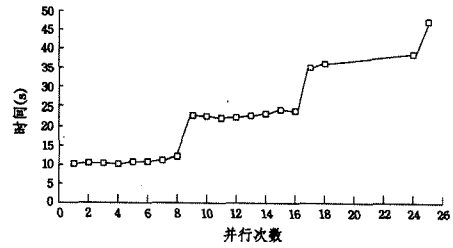

图 4　计算量增加时的处理器耗时曲线

由图 4 可知，当 N 为 1－8，小于处理器支持的线程数[8]时，任务量能够被均匀分配到部分核中，耗时基本相同；而当计算量达到处理器核心数的 1.1～2 倍之间时，耗时还是基本稳定的。这符合多核处理器的设计初衷，因为只要有一个核的工作量达到了两倍，那么时间消耗也就为原来的两倍，尽管其中有些核的时间消耗可能就等于原值。该图主要表明了多核处理器系统中负载均衡的重要性[5]，这种情况与对并行系统的“wall clock time”的考量非常相似。

(3)当计算量固定时，以 *total* 设定总计算量，N 控制并行度(为输入参数)从 1 开始递增，*task_per* 控制每个核的任务负载量，实验结果如图 5 所示。主要代码如下所示：

```
int total=CONSTANT;//一个较大的常数;
int task_per=total/N;
void compute03(int task_per)
{
  for(int i=0;i<task_per;i++)
  {
    for(int j=0;j<1000;j++);
      //功能计算部分;
  }
}
#pragma omp parallel for
  for(int i = 0;i < N;i++)
  {
    compute03(task_per);
  }
```

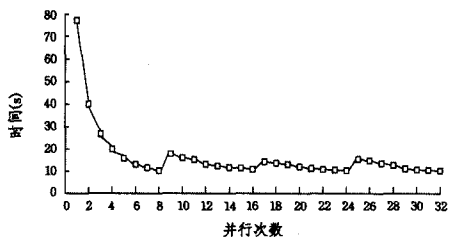

图 5　计算量一定时的处理器耗时曲线

由图 5 看出，在计算量一定，而 N 从 1 变为 2 时，任务分配是由一个核心增至两个，耗时约为原来二分之一，当 N 为 3 时，核心分配增加为 3 个，耗时约为原来的三分之一。其余依此类推，直至增至 8 个核心，此时处理器核才算得到了充分利用[6,8]。当 N 为 9 时，8 个核心充分利用，且其中一个核为其余核的 2 倍计算量，N 为 16 时，任务进一步得到细化，各个核得到更加均衡地负载。可以看出，当任务分解数是处理器核心数的倍数如 8、16、24 时，耗时将呈阶梯状下降趋势，此时的性能达到最好，各个处理器核充分得到利用。所以，对任务的分解是非常重要的，一个好的分解可以使任务被均匀地分配到各个 CPU 核上，从而可以在效率和能耗上均取得比较满意的结果。

3.2 基于 MPI 的弦振荡串/并行模型分析

对弦振荡模型进行并行化，如图 6 所示。其实现过程如下：

(1)for 循环部分可以进行分解，让多个进程同时进行。

(2)在分解的计算部分间存在数据依赖关系。因为每进行一次计算，就需要给相邻进程传递一次数据，才能进行下一轮循环。用 MPI[9,10] 完成这些消息的传递。

(3)判断 $i<n$，若是，转步骤(2)，否则，结束——MPI_FINALIZE。

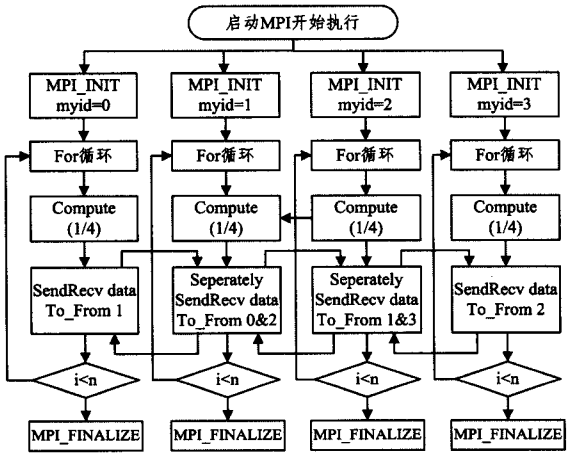

图 6 弦振荡的并行模型

对弦振荡进行了 MPI 算法实现，并模拟出弦振荡随时间运动的轨迹(见图 7)。对该模型分别进行串行和并行运行后的时间对比，可以发现并行化的效果并不明显。这是因为每进行一次循环，就需要向邻近进程发送和接收数据一次，而真正在循环中进行的计算部分，其耗时却远小于消息的传递时间，以致最后花在 for 循环中 MPI 消息传递的时间就变得非常可观了。

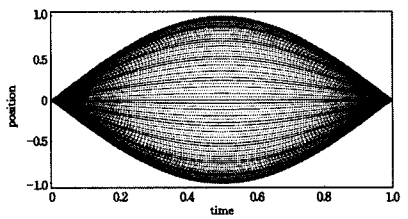

图 7 弦振荡结果图

上述实例表明，对到底应该选用什么并行设计环境更好，不能一概而论[11]，因为在有些情况下根据计算平台的结构特点选择 MPI+OpenMP 混合环境或许效果会更好些；当然，也不一定非要牵强地进行并行化不可，只有在计算量非常庞大并对时间有特殊要求而用一般计算工具难以实现时，才值得去研究和实现并行化。

4 算法选择

对矩阵乘[8,12]进行按行分解和按 CANNON 算法的 MPI 实现。实验选择的矩阵是 1000 * 1000，int 类型数据占 4B，此规模矩阵已经占到 4GB 的内存，对于内存 8GB 的计算机来说比较合适。

由于一个处理器最多可以支持 8 个线程进行并行计算，且 CANNON 算法的局限性——进程数必须是平方数，为了便于与按行分解对比，这里选择 4 个进程。实验结果如表 2 所列。

表 2 矩阵算法时间效率分析(时间单位：s)

算法	串行算法	分行算法	CANNON 算法
运行时间	7.4060	1.5695	2.6112
进程数目	1	4	4
加速比	1	4.71	2.83
效率(加速比/进程数)	1	1.17	0.71

由表 2 可以看出，并行后两种算法都获得了较好的加速比，但分行算法的加速比更优，这与图 5 所描述的曲线规律是吻合的。在效率上，分行后的效率大于 1，称之为超线性加速，同样也优于 CANNON 算法。这是因为在并行度相同的情况下，CANNON 算法的笛卡尔处理器是阵列分配并进行了预处理，结果使矩阵的分发和结果回收机制占用了较多的时间。

因此，在实现相同的并行计算时虽然使用不同的算法都可以完成任务，但计算机的耗时、加速比[13]与效率却不尽相同，可见在实现并行计算时算法选择的重要性。

结束语 在完成并行计算任务的过程中，只有在充分利用计算机的多核资源，综合权衡系统的体系结构特征、系统与应用软件、并行编程语言环境以及正确选用算法等多种因素的前提下，才能真正达到提高系统速度、降低能耗的目的。

参 考 文 献

[1] http://wenku.baidu.com/view/e67cbf630b1c59eef8c7b457.html

[2] Kai Hwang. Advanced Computer Architecture:Parallelism Scalability Programmability[M]. New York: McGraw-Hill Inc., 1993

[3] 王文义，董绍静. 大规模并行处理系统及其程序设计方法研究——Cache 缺失延迟、层次算法和可定域性[J]. 计算机研究与发展，1999，36(5)：78-82
Wang Wen-yi, Dong Shao-jing. Large scale parallel processing system and its program design——Cache deletion delay, hierarchial algorithm and localizability[J]. Journal of Computer Research and Development, 1996, 36(5): 78-82

[4] 罗秋明，明仲，刘刚，等. OpenMP 编译原理及实现技术[M]. 北京：清华大学出版社，2012：20-49

[5] 刘胜飞，张云泉，孙相征. 一种改进的 OpenMP 指导调度策略研究[J]. 计算机研究与发展，2010，47(4)：687-694
Liu Sheng-fei, Zhang Yun-quan, Sun Xiang-zheng. An Improved Guided Loop Scheduling Algorithm for OpenMP[J]. Journal of Computer Research and Development, 2010, 47(4): 687-694

[6] 徐磊，徐莹，张丹丹. 多核构架下 OpenMP 多线程应用运行性能的研究[J]. 计算机工程与科学，2009，31(11)：50-53
Xu Lei, Xu Ying, Zhang Dan-dan. A Study of the Open MP Multithread Application Execution Performance on Multicore Architecures[J]. Computer Engineering and Science, 2009, 31(11): 50-53

[7] Thoman P, Jordan H, Pellegrini S, et al. Automatic OpenMP loop scheduling: a combined compiler and runtime approach[C]// Proceedings of 8th International Workshop on OpenMP. Rome, 2012: 88-101

[8] Shameem A, Jason R. 多核程序设计技术-通过软件多线程提升性能[M]. 李宝峰，富弘毅，李韬，译. 北京：电子工业出版社，2007：145-283
Shameem A, Jason R. Multi-core Programming: Increasing performance Through Software Multi-threading[M]. Intel Corporation, 2006

[9] 都志辉. 高性能计算之并行编程技术—MPI 并行程序设计[M]. 北京：清华大学出版社，2001：52-68

(下转第 59 页)

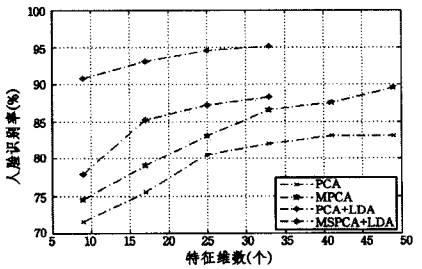

图 3 ORL 人脸库上 4 种识别算法比较

本文算法实际上是 PCA+LDA 与局部排序 PCA 的融合，其本质是在对数据图像进行 PCA+LDA 方法之前用局部排序 PCA 方法将数据图像压缩，使得其主能量最大化，然后将得到的新的训练样本进行 PCA+LDA。通过图 3 可以看到，MSPCA+LDA 方法在识别率上比 PCA 和 PCA+LDA 都有明显提高。从表 1 的数据中可以看出，PCA 方法的最大识别率为 83%，MPCA 方法为 89.5%，PCA+LDA 方法为 88.25%，MSPCA+LDA 方法为 95.12%。

表 1 ORL 人脸库实验结果(%)

	PCA	MPCA	PCA+LDA	MSPCA+LDA
9	72%	74.5%	77.88%	90.75%
17	76%	79%	85.25%	93%
25	81%	83%	87.13%	94.5%
33	82%	86.5%	88.25%	95.12%
41	83%	87.5%	—	—
49	83%	89.5%	—	—

结束语 本文提出了一种新型的分块排序 PCA 方法，然后将经过分块排序 PCA 算法后得出的特征矩阵进行 LDA，得到最后结果。与传统的线性鉴别方法 PCA、PCA+LDA 方法相比，本文算法减弱了经典算法对于全局特征的依赖，增强了局部特征在识别中的作用，而局部特征能够更好地反映图像之间的差异，由于光照和人脸表情变化时，经典人脸识别算法抽取的整体特征受这些条件的影响较大，因此识别效果会受较大影响。而实际上当人脸表情和光照条件变化时，只有部分的人脸区域变化明显而其他区域并无变化，所以分块后抽取的人脸局部特征减弱了整体变化所带来的影响而加强了局部变化的影响。这样就减弱了在识别过程中光照和人脸表情变化给识别所带来的不利影响，从而提高了识别率。分块降低了用于特征抽取的分块图像的图像向量的维数，同时增加了分块图像训练样本的数目，小样本问题也随之消失，大大降低了识别的复杂度。同时随着特征维数的降低，识别时间也将减少。本文只给出了 4×2 分块方式的实验结果，并没有给出多种分块方式。如何找出最佳分块方式以及是否存在使得识别率最高的最优分块方式将成为以后研究的重点。

参 考 文 献

[1] Belhumeur P N, Hespanha J P, Kriengman D J. Eigenfaces vs Fisherfaces: Recognition using class specific linearprojection [J]. IEEE Trans on Pattern Anal Machine Intell, 1997, 19(7): 711-720

[2] 边肇祺，张学工. 模式识别(第二版)[M]. 北京：清华大学出版社，1999：176-177
Bian Zhao-qi, Zhang Xue-gong. Pattern Recognition [M]. Beijing: Tsinghua University Press, 2000

[3] Kirby M, Sirovich L. Application of the KL procedure for the characterization of human faces [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1990, 12(1): 103-108

[4] Turk M, Pentland A. Face processing: Models for recognition [C]// Proceedings of Intelligent Robots and Computer Vision VIII. 1989, 1: 22-32

[5] Turk M, Pentland A. Eigenfaces for recognition [J]. Journal of Cognitive Neuroscience, 1991, 3(1): 71-86

[6] Turk M, Pentland A. Face recognition using Eigenfaces [C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Hawaii, USA, 1991: 586-591

[7] 金忠. 人脸图像特征抽取与维数研究 [D]. 南京：南京理工大学，1999
Jin Zhong. Feature extraction and dimension research of human faces[D]. Nanjing: Nanjing University of science and technology, 1999

[8] Yang Jian, Yu Yang-jing. Why can LDA be performed in PCA transformed space[J]. Pattern Recognition, 2003, 36: 563-566

[9] 杨键，杨静宇，叶辉，等. Fisher 线性鉴别分析的理论研究及其应用[J]. 自动化学报，2003，29(4)：482-493
Yang jian, Yang jing-yu, Ye Hui, et al. Theory of fisher linear discriminant analysis and its apllication[J]. Acta Automatica Sinica, 2003, 29(4): 482-493

[10] Jin Zhong, Yang Jing, et al. Face Recognition based on uncorrelated discriminant transformation [J]. Pattern Recognition, 2001, 34(7): 1405-1416

[11] Chen Li-fen, Liao Hong-yuan, et al. A new LDA-Based Face Recognition System Which Can Solve the Small Sample Size Problem [J]. Pattern Recognition, 2000, 33: 1713-1726

[12] Wu Xiao-jun, Josef K, Yu Yang-jing. A New Direct LDA Algorithm for Feature Extraction in Face Recognition [C]// Proceedings of the 17th International Conference on Pattern Recognition. Cambridge, UK, 2004: 545-548

[13] 陈伏兵，高秀梅，张生亮，等. 基于分块 PCA 的人脸识别方法[J]. 小型微型计算机系统，2006，27(10)：1943-1947
Chen Fu-bing, Gao Xiu-mei, Zhang Sheng-liang, et al. Face Recognition based on modular PCA approach[J]. Mini-Micro Systems, 2006, 27(10): 1943-1947

[14] 陈伏兵，杨静宇. 分块 PCA 及其在人脸识别中的应用[J]. 计算机工程与设计，2007，28(8)：1889-1892
Chen Fu-bing, Yang Jing-yu. Modular PCA and its application in face recognition[J]. Computer Engineering and Design, 2007, 28(8): 1889-1892

(上接第 31 页)

[10] Group W, Luck E, Skjellum A. Using MPI: Portable Parallel Programming with the Message Passing Interface [M]. Cambridge, MA: MIT Press, 1999

[11] Brown R. Performance and Productivity Comparison Between OpenMP and MPI[J]. Int Parallel Prog, 2007, 35: 441-458

[12] 刻公孝，申卫昌，刘骊，等. 一种基于 MPICH 的高效矩阵相乘并行算法[J]. 计算机工程与应用，2009，45(26)：72-73
Yan Gong-xiao, Shen wei-chang, Liu Li, et al. Effective matrix multiplication parallel algorithm based on MPICH[J]. Computer Engineering and Applications, 2009, 45(26): 72-73

[13] 王之元，胡庆丰，陈娟. 能耗并行加速比：高性能计算系统综合性能的有效度量[J]. 计算机工程与科学，2009，31(11)：113-116
Wang Zhi-yuan, Hu Qing-feng, Chen Juan. Power Parallel Speedup: An Effective Metric for Evaluating the Comprehensive Performance of High-Performance Computing Systems [J]. Computer Engineering and Science, 2009, 31(11): 113-116