

Windows 环境下 FFT 多核并行算法的设计实现

张燕燕, 洪 龙

(南京邮电大学 计算机学院, 江苏 南京 210003)

摘 要:多核技术的问世,使得人们在桌面计算机环境下研究并行算法,运行并行程序成为可能。与此同时,如何充分利用多核技术进行并行程序设计却是所面临的巨大挑战。在叙述了多核技术,并将其与超线程技术比较后,介绍了 Windows 环境下的常用的多核编程工具 OpenMP,并重点描述了并行语句 Fork/Join;在简述了信号处理中常用的 FFT 后,重点分析了 FFT 的按时间基 2 抽取形式,并据此利用 OpenMP 设计了一个 n 核环境下的 FFT 并行算法,通过对相应程序的运行,结果表明,该算法加速比接近 n 。

关键词:多核技术;超线程技术;并行程序设计;Windows;OpenMP;FFT 并行算法

中图分类号:TP301.6

文献标识码:A

文章编号:1673-629X(2010)09-0074-04

Design and Implementation of FFT Parallel Algorithm with Multi-core Techniques in Windows Environment

ZHANG Yan-yan, HONG Long

(School of Computer, Nanjing University of Posts & Telecommunications, Nanjing 210003, China)

Abstract: The advent of multi-core techniques makes it possible that to study the parallel algorithms or to execute the parallel programs under the desktop computer environment. However, how to fully exploit parallel programming with multi-core techniques is the huge challenge confronting us. In this paper, OpenMP, a multi-core programming tool commonly used on Windows platform, was introduced after introducing the multi-core techniques and comparing with the hyper-threading techniques, and Fork/Join was discussed in detail, which was a kind of the parallel statements. Fast Fourier Transformation (FFT), one of elementary operations commonly used in signal processing, was introduced briefly, and then the Decimation In Time (DIT) based on 2 FFT was intensively analyzed, which was a frequently used form of FFT; it was proposed that an FFT parallel algorithm on the basis of the DIT-2FFT with OpenMP, under n cores' environment. Through running the applications, the result showed that the speed-up ratio of the algorithm was very close to n .

Key words: multi-core techniques; hyper-threading techniques; parallel programming; Windows; OpenMP; FFT parallel algorithm

0 引 言

2009 年国际数据公司的统计数据表明,当今的计算机系统几乎都采用多核微处理器设计,在多核环境下进行程序设计已成为不可逆转的潮流。多核是指在一块芯片上集成多个计算核。在多核环境下,程序可以同时运行在多个物理核中。如图 1(a)所示的双核环境下,线程 1 和线程 2 独自占用执行单元 1 和执行单元 2,在物理意义上实现了并行。

而在超线程环境下,多个线程只是并发地在一个物理核上运行。如图 1(b)所示,线程 1 和线程 2 都是在一个执行核上运行,只是超线程技术采用逻辑核模

拟物理核的策略^[1],所以会产生分别对应线程 1 和线程 2 的两个 CPU 状态。

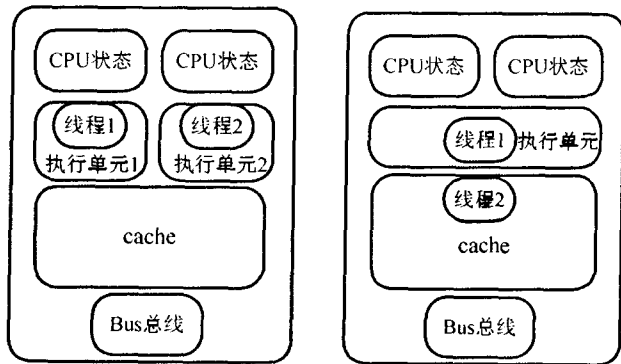


图 1 超线程技术与多核技术的比较

多核技术的普遍使用,使得在一般环境下研究和实现高性能计算成为可能,与此同时,如何充分利用多核进行并行程序设计却是所面临的巨大挑战^[2~4]。文

收稿日期:2010-01-20;修回日期:2010-04-22

作者简介:张燕燕(1984-),女,山东泰安人,硕士研究生,研究方向为并行计算及其体系结构;洪 龙,研究员级高级工程师,教授,研究方向为计算机系统结构,非经典逻辑及应用。

中在介绍了 Windows 环境下多核编程工具后,设计了多核环境下的 FFT 算法,并根据程序运行结果,对算法的性能进行了分析。

1 基于 Windows 环境的 OpenMP

目前 Windows 环境下支持多核程序开发的方法主要有三种: Windows 多线程^[5,6]、OpenMP^[7]和 MPI^[8]。Windows 多线程是利用 Win32 API 或 MFC 以及 .NET Framework 提供的接口,通过 Windows 的线程库实现并行计算。实现形式的多样化给 Windows 编程带来了很大的灵活性,但也使得多线程编程变得复杂。MPI 是一种基于消息传递的并行编程技术。其优点是灵活,容易实现多个节点间的并行处理。但由于进程独立性和显式消息传递的特点, MPI 标准较为烦琐,基于其开发并行程序也更加复杂。

OpenMP 是一种面向多处理器多线程的并行编程模型。它在相对更细粒度的并行性开发中具有明显优势,可扩展性较好;同时 OpenMP 以指导语句的形式指导循环的并行计算,易于使用;此外, OpenMP 中不需要显式通信,没有 MPI 程序中那样的消息传递开销^[9]。

OpenMP 在并行执行程序时,采用的是“Fork/Join”方式, Fork 语句用于创建新线程或者唤醒已有线程,执行 Fork(*s*)语句时,派生出标号为 *s* 的新线程; Join 语句与 Fork 语句相配合,用于实现已派生多线程的汇和, Join 的形式为 Join(*t*),其中 *t* 为已派生出的并发线程个数^[10]。例如在一个四核处理器上,有标号为 *s*, *t*, *k*, *l* 的四个独立线程,采用 Fork/Join 实现并行的算法如下:

```
Begin
Fork(s); // 依次派生出 3 个标号分别为 s, t, k 的线程
Fork(t);
Fork(k);
线程 l
Join(3); // 等待已派生出的 3 个并发线程都执行完并行代码后
汇合成主线程 l
后续程序
End
```

如图 2 所示,开始的时候,在核 1 上运行一个叫做“主线程”的程序,采用 Fork(*s*)、Fork(*t*)、Fork(*k*) 语句,依次派生出标号为 *s*、*t*、*k* 的三个线程后,主线程运行 *l*, 此时四个线程同时执行。当 *l* 运行结束,遇到 Join(3) 时,主线程一直等到三个子线程运行结束才开始执行后续程序。

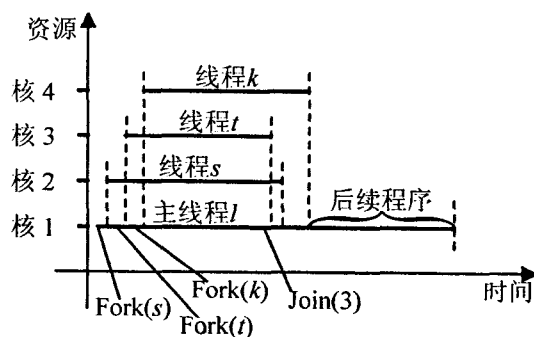


图 2 Fork/Join 并行模式示意图

除了采用 Join 语句实现同步外,栅障(barrier)也是 OpenMP 用于线程同步的一种方法。barrier 使程序等到多线程完成当前的循环、结构化块或并行区,再继续执行后面的工作。

以并行程序中并行部分的子线程的相似性区分,可把它们分为同构子线程和异构子线程。若多个子线程在处理不同的数据时执行的是相同的代码,则称这些线程为同构子线程,也称作单程序多数据(Single Program Multiple Data);若多个子线程可执行不同的代码,则称这些子线程为异构子线程,也称为多程序多数据(Multiple Program Multiple Data)。OpenMP 中的 for 就是用于同构子线程并行的指令。for 指令是用来将一个 for 循环分配到多个线程中执行。for 命令一般与 parallel 命令合起来形成 parallel for 命令使用。对于 for 指令,使用 barrier 来实现同步的程序如下:

```
#pragma omp parallel for
for( i = 1; i <= m; i++) {
Pi
#pragma omp barrier
Qi
}
```

barrier 语句实现等待效应。当 P_i 结束遇到该语句时,它必须等待,直到所有并发线程全部跟上之后才能继续执行 Q_i 程序块。其中 P_i 就对应图 2 中线程 *s*, *t*, *k*, *l* 所执行的代码, Q_i 即为图 2 中的后续程序。

2 FFT 并行算法的设计

2.1 FFT 简介

离散 Fourier 变换(DFT)是数字信号处理中的基本运算,在信号处理、图像处理、微分方程求解、医学电子学、雷达或无线电天文学等许多领域都有着广泛的应用。按时间抽取(DIT)基-2 快速 Fourier 变换(FFT)是 FFT 的主要形式之一,它是在时域上将有限长序列(在科学技术工作中人们通过采样所得到的离散时间信号大多是有限长的 N 点序列。若连续时间信

号为 $x_a(t)$, 采样周期为 T , 则采样得到的离散时间信号(序列)为: $x(m) = x_a(t)|_{t=mT} = x_a(mT) = x(mT)$, 即为有限长序列。逐次分解为长度减半的奇序号子序列和偶序号子序列, 用子序列的 DFT 来实现整个序列 DFT^[11]。设 N 点有限长序列 $x(m)$ ($0 \leq m \leq N-1$), $N = 2^s$ (s 为正整数)。首先按奇、偶序号将 $x(m)$ 分解为两个长度为 $N/2$ 的子序列, 即:

$$\begin{cases} x_1(r) = x(2r) \\ x_2(r) = x(2r+1) \end{cases} \quad r = 0, 1, \dots, \frac{N}{2} - 1 \quad (1)$$

又已知 $x(m)$ 的 DFT 为,

$$X(k) = \text{DFT}[x(m)] = \sum_{m=0}^{N-1} x(m) W_N^{km}, \quad k = 0, 1, \dots, N-1 \quad \text{且 } m = 0, 1, \dots, N-1 \quad (2)$$

则由式(1)和式(2)得 N 点有限长序列 $x(m)$ 的 DFT 可由式(3)全部确定出来:

$$\begin{cases} X(k) = X_1(k) + W_N^k X_2(k) \\ X(k + N/2) = X_1(k) - W_N^k X_2(k) \end{cases} \quad k = 0, 1, \dots, \frac{N}{2} - 1 \quad (3)$$

其中 $X(k)$ 和 $X(k + N/2)$ 构成 N 点 DFT, $X_1(k)$ 、 $X_2(k)$ 分别为子序列 $x_1(r)$ 和 $x_2(r)$ 的 $N/2$ 点 DFT, $W_N^k = e^{-j\frac{2\pi}{N}k}$ 称为旋转因子。式(3)运算可用蝶形运算信号流程图符号来表示, 如图3所示, 它对应一次复数乘法和两次复数加法运算。

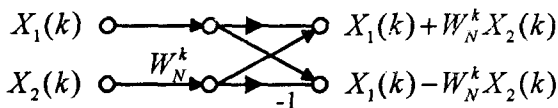


图3 蝶形运算信号流程图符号

时域序列的分解过程为: 先将 N 点 DFT 分解为两个 $N/2$ 点 DFT, 再分解为 4 个 $N/4$ 点 DFT, 进而是 8 个 $N/8$ 点 DFT, 直至 $N/2$ 个 2 点 DFT。每分解一次称为一级运算。对于 $N = 2^s$ 点 DFT, 共有 $s = \log_2 N$ 级运算。例如一个 $N = 2^3 = 8$ 的 DFT 运算, 经过 DIT 基-2FFT 抽取后得到如图4所示的结构。

在图4中从左到右依次为第1级、第2级和第3级。按时间抽取时, 首先分解得到的是最后一级运算, 即第3级, 然后依次向前一级分解。每个矩形框内是一级运算, 它包含 $N/2$ 个蝶形运算, 其中任意两个蝶形运算之间是独立的, 没有数据交互, 符合并行执行的条件。为了保证每级运算是按自然顺序进行, 需要在每级运算之前先调整序列顺序。这部分操作采用互连函数中的均匀洗牌、碟式置换和位序颠倒置换^[12], 如图4, 在第一级运算前要进行位序颠倒置换。

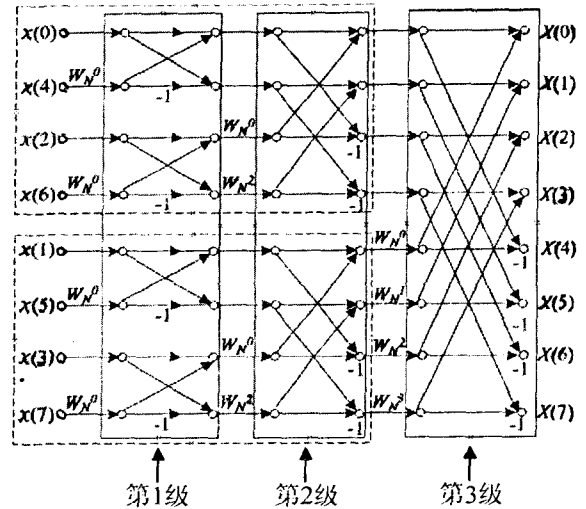


图4 按时间抽取基-2FFT

$$\rho(x_{m-1}x_{m-2}\cdots x_1x_0) = x_0x_1\cdots x_{m-2}x_{m-1}$$

其中 $x_{m-1}x_{m-2}\cdots x_1x_0$ 是序号的二进制表示; 然后在第二级运算前要进行均匀洗牌置换。

$$S(x_{m-1}x_{m-2}\cdots x_1x_0) = x_{m-2}x_{m-3}\cdots x_1x_0x_{m-1}$$

接着在第三级运算前进行超蝶置换。

$$\beta_{(k)}(x_{m-1}x_{m-2}\cdots x_{m-k}x_{m-k-1}x_{m-k-2}\cdots x_1x_0) = x_{m-k-1}x_{m-2}\cdots x_{m-k}x_{m-1}x_{m-k-2}\cdots x_1x_0$$

第三级运算全部结束后再进行一次超蝶置换, 就可以将经过 DFT 变换后的序列按输入顺序存放。用“ \oplus ”表示蝶形运算, 则 $n = 8$ 的 FFT 运算可以表示为:

$$\beta^{(1)} \oplus \beta^{(1)} \oplus S \oplus \rho$$

2.2 基于多核的 FFT 算法

利用 OpenMP 实现相应的 FFT 并行算法如下:

利用 barrier 命令显式同步的方式:

```

Begin
输入当前 CPU 核数 n // 可通过 OpenMP 的库函数 omp_get_num_procs 来获取 n
输入点数 N // 若 N 不足 2 的整数次幂, 可采用后补零值点的方法来补齐数据
输入有限长序列 x[m] // m = N
计算级数 s // s = log2N
#pragma omp parallel for private(k) num_threads(n) // 分配 n 个线程依次到 n 个核上
for(k = 0, 1, ..., N/2 - 1) // 计算旋转因子
{
    W_N^k = e^{-j\frac{2\pi}{N}k};
}
#pragma omp barrier // 显式同步, 等待 for 循环全部执行完毕
for(r = 1, 2, ..., s) // 逐级运算
{
    调整序列 x[m] 的顺序
    置 c = 2^{s-r-1} // 每级中相同旋转因子的个数

```

表 1 变换前的数据

变换前的数据			
$x(0) = (342, 124)$	$x(4) = (35, 56)$	$x(8) = (45, 0)$	$x(12) = (245, 678)$
$x(1) = (546, 565)$	$x(5) = (12987, 47247)$	$x(9) = (34, 76)$	$x(13) = (3543, 986)$
$x(2) = (7834, 239)$	$x(6) = (235, 134)$	$x(10) = (23254, 456)$	$x(14) = (3434, 5657)$
$x(3) = (127, 4755)$	$x(7) = (1987, 14)$	$x(11) = (24456, 456)$	$x(15) = (5432, 9765)$

```
#pragma omp parallel for private(i,k) num_threads(n) // 分配
n 个线程依次到 n 个核上
for(i = 0,1,..., N/2)
{
    置 k = i% (2^r) * c; // 旋转因子的序号
     $X_r(i) = X_{r-1}(i) + W_N^k X_{r-1}(i+1);$ 
     $X_r(i+1) = X_{r-1}(i) - W_N^k X_{r-1}(i+1);$ 
}
#pragma omp barrier // 显式同步, 等待 for 循环全部执行完毕
}
调整变换后序列的存储顺序为输入顺序
输出变换后的序列  $X[m]$ 
End
```

其中 private 子句是将括号内的变量声明成线程私有的变量,即指定每个线程都有它自己的变量私有副本,其他线程无法访问私有副本。

去掉上述算法中的两个“#pragma omp barrier”语句即变为隐式同步方式。这是因为在 parallel、for、sections 和 single 结构的最后,会有一个隐式的 barrier,该 barrier 由编译器生成或从运行库中调用。

2.3 运行结果分析

在 Microsoft Visual Studio 2008 中进行程序设计。输入 $N = 16$ 点变换数据进行测试,如表 1 所示。

串行执行情况下得到如表 2 所示的结果:串行执行时间为 31ms。

表 2 串行执行结果

执行时间	经过 FFT 变换后的数据
31ms	$X(0) = (1934, 1530)$
	$X(1) = (33620, 97934)$
	$X(2) = (57095, -23107)$
	$X(3) = (76423, 66059)$
	$X(4) = (-4701.8, 34620.3)$
	$X(5) = (85070.2, 16418.3)$
	$X(6) = (-8835.87, 883.682)$
	$X(7) = (-75390.8, 19204.4)$
	$X(8) = (16371, -10755)$
	$X(9) = (-49257, -85277)$
	$X(10) = (-1657.23, -17776.3)$
	$X(11) = (7167.23, 768.328)$
	$X(12) = (-257.276, -52466.5)$
	$X(13) = (-79763.1, -564.086)$
	$X(14) = (4278.98, -776.9)$
	$X(15) = (5471.68, -42271.2)$

双核环境下,利用 OpenMP 进行并行设计,得到如表 3 所示的结果:并行执行时间为 16ms。

表 3 并行执行结果

执行时间	经过 FFT 变换后的数据
16ms	$X(0) = (1934, 1530)$
	$X(1) = (33620, 97934)$
	$X(2) = (57095, -23107)$
	$X(3) = (76423, 66059)$
	$X(4) = (-4701.8, 34620.3)$
	$X(5) = (85070.2, 16418.3)$
	$X(6) = (-8835.87, 883.682)$
	$X(7) = (-75390.8, 19204.4)$
	$X(8) = (16371, -10755)$
	$X(9) = (-49257, -85277)$
	$X(10) = (-1657.23, -17776.3)$
	$X(11) = (7167.23, 768.328)$
	$X(12) = (-257.276, -52466.5)$
	$X(13) = (-79763.1, -564.086)$
	$X(14) = (4278.98, -776.9)$
	$X(15) = (5471.68, -42271.2)$

加速比 $S(n) = \text{串行时间} / \text{并行时间} = 31\text{ms} / 16\text{ms} = 1.94$, 效率 = 加速比 / CPU 核数 = $S(n) / 2 = 0.97$ 。在实际应用中,算法设计要考虑在特定的多核平台上,如何将一个具体应用问题进行任务分割,选择什么样的任务粒度,有效的使并行计算和通信等额外开销达到平衡,以实现高效利用系统资源的目的^[13]。例如在上述情况下,观察图 4 可发现,两个虚线框之间是没有数据交互的。因此,也可采用 Windows 多线程的方法利用一个线程处理图 4 中上虚线框中的计算,另一个线程处理下虚线框中的计算。这样同步只出现在最后一级运算之前,而不用每级运算前都要保持同步,减少了因为等待同步而造成的额外开销。

3 结束语

文中在简单比较了多核技术和超线程技术之后,详细介绍了 Windows 环境下的常用并行方法 OpenMP,及 Fork/Join 并行语句,并利用 OpenMP 设计出了 n 核环境下的 FFT 并行算法,通过在双核处理器上实际运行得出,该算法的加速比为 1.94,因此有效地实现了多核资源的充分利用。

EndIf

EndFor

Return BMS

由于从候选服务集合 SResults 匹配得到的符合用户需求的 QoS 可能不止一个,而用户追求的都是高性价比^[9]的服务,所以用 λ 来刻画 QoS 的性价比, λ 与可靠性、安全性成正比,与响应时间和服务费用成反比,则 λ 值越大,性价比越高。但是这样得出的仍有可能有多个具有性价比的服务,可以把 BMS 中的服务列出来,给用户灵活的选择权,同时加上该服务的 QoS 描述,让用户在可以比较的前提下选出更符合自己需求的服务。同时也可以让用户列出对 QoS 的 4 个参量的优先级,系统给以最优选择。

6 结束语

Web 服务发现就是从 Web 服务库中搜索并发现能满足用户服务请求的 Web 服务的过程。文中的基于语义模糊匹配策略提高了 Web 服务发现的准确率,而 QoS 匹配约束支持解决了从已发现的能满足用户服务请求的原子服务集中找到用户满意的服务,而且还支持用户参与选择的状态。

参考文献:

- [1] Web Service Description Language[EB/OL]. 2007-06-26.
http://www.w3.org/TR/2007/REC-wsdl20-20070626.

(上接第 77 页)

参考文献:

- [1] Akhter S, Roberts J. Multi-Core Programming: Increasing Performance Through Software Multi-Threading [M]. translated by Li Bao-feng, Fu Hong-yi, Li Tao. Beijing: Electronics Industry Press, 2007.
- [2] Li Jianhua, Guo Weibin, Zheng Hong. An Undergraduate Parallel and Distributed Computing Course in Multi-Core Era [C]. IEEE, The 9th International Conference for Young Computer Scientists. Zhangjiazhai: [s. n.], 2008: 2412 - 2416.
- [3] Wang Lei, Fang Jia-yong, Gao Cheng-jin. Parallel test scheduling on multi-core platform [C]//IEEE AUTOTESTCON 2008. Salt Lake City, UT: [s. n.], 2008: 504 - 507.
- [4] Yang Shu-Sian, Wang Sung-Wen, Chen Hong-Ming, et al. A Parallelism Encoding Framework for the Temporal Scalability of H. 264 AVC Scalable Extension [C]//Ninth IEEE International Symposium on Multimedia 2007 - Workshops. Taichung: [s. n.], 2007: 397 - 400.

- [2] Universal Description, Discovery and Integration (UDDI) [EB/OL]. 2001. http://www.uddi.org/specification.html.
- [3] Friesen A, Altenhofen M. Matching composed semantic web services at publishing time [C]//Commerce Technology Workshops, Seventh IEEE International Conference. Munich, Germany: [s. n.], 2005: 64 - 70.
- [4] Paolucci M, Kawamura T, Payne T R, et al. Semantic Matching of Web Services Capabilities [J]. Lecture Notes in Computer Science, 2002, 2342: 333 - 347.
- [5] 杨清平, 邱玉辉, 蒲国林. 基于 QoS 和 OWL-S 的 Web 服务发现研究 [J]. 计算机科学, 2009(3): 146 - 166.
- [6] 张孝国, 黄广君, 郭洪涛, 等. 基于语义的 Web 服务发现技术研究 [J]. 计算机应用, 2008(4): 881 - 883.
- [7] William S, Austin T. Ontologies [J]. IEEE Intelligent Systems, 1999(1/2): 18 - 19.
- [8] 吴健, 吴朝晖, 李莹, 等. 基于本体论和词汇语义相似度的 Web 服务发现 [J]. 计算机学报, 2005(4): 595 - 602.
- [9] Ganesan P, Garcia - Molina H, Widom J. Exploiting Hierarchical Domain Structure to Compute Similarity [J]. ACM Transactions on Information Systems, 2003, 21(1): 64 - 93.
- [10] 张献, 李舟军, 李梦君. 一种关于语义 Web 服务匹配的策略和实现 [J]. 计算机科学, 2007, 34(15): 99 - 103.
- [11] 彭晖, 史忠植, 邱莉榕, 等. 基于本体概念相似度的语义 Web 服务匹配算法 [J]. 计算机工程, 2008(8): 51 - 53.
- [12] 吴海鹏, 饶若楠. 一种基于服务本体及其词汇语义的 Web 服务匹配算法 [J]. 计算机应用与软件, 2008(5): 131 - 133.

- [5] 苏成顺, 李贞培. 基于多线程的分段图像轮廓跟踪算法 [J]. 计算机技术与发展, 2009, 19(10): 99 - 101.
- [6] 李中年, 张朋, 谢杨华. 基于 DHT 快速插值并行算模的研究 [J]. 计算机技术与发展, 2007, 17(5): 242 - 244.
- [7] 林萌. Hausdorff 距离多核并行技术及其研究应用 [D]. 厦门: 厦门大学, 2008.
- [8] MPI: A Message - Passing Interface Standard [S/OL]. 1995 - 07. http://www.mpi-forum.org/docs/mpi-11.html/mpi-report.html.
- [9] 吴建平, 王正华, 李晓梅. 利用混合编程改善 SMP 机群上并行矩阵乘法的性能 [J]. 国防科技大学学报, 2003, 28(4): 68 - 72.
- [10] 多核系列教材编写组. 多核程序设计 [M]. 北京: 清华大学出版社, 2007.
- [11] 王凤文, 舒冬梅, 赵宏才. 数字信号处理 [M]. 第 2 版. 北京: 北京邮电大学出版社, 2006.
- [12] 洪龙, 李爱群, 周宁宁, 等. 计算机系统设计 - 概念与技术 [M]. 西安: 西安电子科技大学出版社, 2004.
- [13] 李学干. 计算机系统结构 [M]. 第 3 版. 西安: 西安电子科技大学出版社, 2004.