典型并行算法的实现性能分析

雷英杰[1], 霍红卫[2]
(1. 空军工程大学 导弹学院, 陕西 三原 713800;2. 西安电子科技大学 计算机学院, 陕西 西安 710071)

摘 要:讨论和分析了几种典型的并行算法及其各种处理方法在基于Windows XP环境、消息传递接口MPI并行编程环境支持和C++语言描述的编程实现问题,给出了相应并行程序详尽的计算结果,对比分析了它们的计算性能,以及它们对计算精度产生的影响。分析结论以相应并行算法的实际编程实现和试验计算数据为基础,可信度高。设计实例表明,分析方法是有效的。

关键词:并行计算;消息传递接口;并行算法;高性能计算
中图分类号:TP393 **文献标识码**:A **文章编号**:1009-3516(2003)05-0067-04

并行算法计算性能问题是高端、高性能、大规模并行计算领域非常重要的研究内容[1]。本文以计算 π 值并行算法为例,通过对若干典型并行算法基于消息传递接口MPI(Message Passing Interface)编程[2]和C语言描述的Hostless程序实现及其运行结果的分析,给出一些新的对比分析结论。

1 MPI并行编程环境

在基于MPI的编程模型中,计算是由一个或多个彼此通过调用函数库函数进行消息收、发通信的进程所组成。在绝大部分MPI实现中,一组固定的进程在程序初始化时生成。这些进程可以执行相同或不同的程序。进程间的通信可以是点到点的,也可以是群体的(Collective)。MPI最重要的特性是使用了称之为通信体的机构,允许程序员定义一种封装内部通信结构的模块。所谓通信体就是一个进程组加上进程活动环境,其中进程组就是一组有限或有序的进程集合。所谓有限意即组内包含有限数目的 n 个进程依次按0,1,…,$n-1$ 整数定序(Ranked)。MPI中的进程活动环境是指系统指定的超级标记(Supertag),它能安全地将彼此相互冲突的通信区分开来。每个通信体都有一个不同的、系统指定的进程活动环境,在这一个进程活动环境中发送的消息不能在另一个进程活动环境中被接收。

MPI不是一个独立的、自包含的软件系统,MPI进程是重量级、单线程的进程[2]。MPI标准并不指明如何启动并行计算,它可通过命令行参数指定应被生成的进程数,然后按SPMD或MPMD方式执行程序[3]。

MPI并行程序中经常需要一些进程组间的群体通信,包括:①路障(Barrier)——同步所有进程;②广播(Broadcast)——从一个进程发送一条数据给所有进程;③收集(Gather)——从所有进程收集数据到一个进程;④散射(Scatter)——从一个进程散发多条数据给所有进程;⑤归约(Reduction)——包括求和、求积等。MPI包含的函数多达200个,它们的功能及参数描述参见文献[4]、[5]等。

2 问题与算法描述

设计求 π 值并行算法的关键是构造一个合适的函数 $f(x)$,使得它计算起来既简便,误差又小。即使

收稿日期:2003-05-12
基金项目:国家教育部骨干教师资助计划项目(GG-810-90039-1003)资助
作者简介:雷英杰(1956-),男,陕西渭南人,教授,博士生导师;主要从事智能信息处理与模式识别研究;
霍红卫(1963-),女,陕西西安人,主要从事算法设计与分析,并行与分布计算研究.

$$\pi = \int_a^b f(x)\,dx \approx \sum_{i=0}^{N-1} h \cdot f(x_i)$$

这里取 $a=0, b=1, N$ 为子区间分割数,于是 $h = 1/N$。

我们选取二种函数,一种是 $f(x)=4\sqrt{1-x^2}$,其物理意义是单位圆面积 $S=\pi \cdot r^2$,我们称之为算法 1。另一种函数是 $f(x)=4/(1+x^2)$,可以直接进行积分,我们称之为算法 2。

算式中的 x_i 在计算各个子区间的小矩形面积时,按左边界计算取 $x_i = h*i$,按右边界计算为 $x_i = h*(i+1)$,按中间值计算为 $x_i = h*(i+0.5)$,按梯形计算时则要同时用到左、右两个边界。

并行算法实现时,分配每一个进程或 CPU 计算其中的若干个子区间,然后通过归约,得到最终结果[6]。计算过程中,精度越高,要求 N 越大。对于足够大的 N,要求在 p 个进程或处理机上尽量均衡分配计算任务,通过归约完成总的计算。

3 编程实现

3.1 编程与运行环境

所实现的并行计算 π 值的程序环境如下:MPI 支持为 MPICH 1.2.4;编程语言为 Microsoft Visual C++ 6.0;开发环境为 Microsoft Windows XP & Visual Studio 6.0;硬件环境为 P III 900 CPU, 128M RAM, 256k Cache;运行环境为 Windows XP。

3.2 基于 MPI 的编程

算法实现中用到的 MPI 函数主要有:MPI_INIT——启动 MPI 计算;MPI_Comm_size——确定进程数;MPI_Comm_rank——确定自己的进程标识数;MPI_Get_processor_name——获取处理机名字;MPI_Wtime ——获取系统时钟时间,以便记录算法始末运行的时刻和计算算法总的运行时间;MPI_Bcast——广播进程信息给该组的所有其它进程;MPI_Reduce——归约,使各个进程所计算的值归约为一个值;MPI_Finalize——结束 MPI 进程。此外,比较重要的函数还有 MPI_Send——发送一条消息;MPI_Recv——接受一条消息等。

对于足够大的子区间划分数 N 和进程数 p,每个进程均衡完成的计算量为总计算量的 N/p。每一个计算进程通过 MPI_Bcast 函数将自己计算的结果传递给其他进程,最后由某一进程(标识数 MyId 为 0)利用 MPI_Reduce 函数进行归约,从而得到总的计算结果。

3.3 实现的算法功能

对于算法 1,程序实现了如下功能:① 利用小矩形左边界分解计算函数子区间面积的数值积分法,记为 RLA1;②利用小矩形中间界分解计算函数子区间面积的数值积分法,记为 RMA1;③利用小矩形右边界分解计算函数子区间面积的数值积分法,记为 RRA1;④利用小梯形分解计算函数子区间面积的数值积分法,记为 RTA1。

对于算法 2,程序实现了如下功能:①利用小梯形分解计算函数子区间面积的数值积分法,记为 ATA2;②利用小矩形中间界分解计算函数子区间面积的数值积分法,记为 AMA2

为分析方便起见,表 1 列出了程序实现的功能号、算法及处理方法之间的对应关系。

使用时通过交互式菜单指定选择相应的算法及其实现功能,然后指定相应的子区间分割数目,程序将给出相应的计算结果。此外,也可以通过编辑一个文本文件,来设定需要选择的功能号和需要指定的子区间划分数目,然后在启动运行该程序时通过系统所提供的重定向功能,把该程序的输入重新定向到编辑好的文本文件。"功能号"取值可参见表 1,"子区间分割数"取值应为任何足够大的正整数,如 $10^2 \sim 10^8$。

表 1 程序功能、算法、处理方法对应关系

功能号	简记符	所用的算法	子区间	使用边界
0	RLA1		矩形	左边界
1	RMA1		矩形	中间界
2	RRA1	算法 1	矩形	右边界
3	RTA1		梯形	左、右边界
6	ATA2	算法 2	梯形	左、右边界
7	AMA2		矩形	中间界

4　结果及分析

针对各种算法功能,分别指定不同的子区间分割数,它们各自的计算结果如表2～表7所示。在设定计算参数时,子区间分割数从10^2～10^8共分7个数量级。表中分别列出子区间分割数N、新计算出来的π值与25位π值比较后的误差值,以及本次计算所用的时间(单位为s)。

精确π值的前25位为3.141 592 653 589 793 238 462 643。

表2　RLA1计算结果

N	误差值	所用时间/s
10^2	0.018 824 378 189 251 9	0.000 037
10^3	0.001 962 813 321 229 7	0.000 108
10^4	0.000 198 824 021 528 0	0.000 908
10^5	0.000 019 962 812 214 4	0.009 161
10^6	0.000 001 998 824 183 1	0.094 868
10^7	0.000 000 199 962 918 7	0.978 500
10^8	0.000 000 019 997 082 2	10.239 773

表3　RMA1计算结果

N	误差值	所用时间/s
10^2	0.000 344 204 310 215 1	0.000 027
10^3	0.000 010 891 323 113 2	0.000 107
10^4	0.000 000 344 434 838 0	0.000 909
10^5	0.000 000 010 892 036 6	0.029 086
10^6	0.000 000 000 344 535 1	0.109 987
10^7	0.000 000 000 010 552 9	1.196 961
10^8	0.000 000 000 001 047 6	12.075 547

表4　RRA1计算结果

N	误差值	所用时间/s
10^2	0.021 175 621 810 748 2	0.000 029
10^3	0.002 037 186 678 770 3	0.000 101
10^4	0.000 201 175 978 471 9	0.000 841
10^5	0.000 020 037 187 785 4	0.008 257
10^6	0.000 002 001 175 817 1	0.122 723
10^7	0.000 000 200 037 080 7	1.129 378
10^8	0.000 000 020 002 918 4	11.09 7461

表5　RTA1计算结果

N	误差值	所用时间/s
10^2	0.001 175 621 810 747 7	0.000 037
10^3	0.000 037 186 678 760 7	0.000 208
10^4	0.000 001 175 978 480 4	0.001 921
10^5	0.000 000 037 187 841 3	0.039 095
10^6	0.000 000 001 175 921 1	0.261 560
10^7	0.000 000 000 037 529 1	2.568 481
10^8	0.000 000 000 000 465 0	36.019 209

表6　ATA1计算结果

N	误差值	所用时间/s
10^2	0.000 016 666 666 665 4	0.000 053
10^3	0.000 000 166 666 666 7	0.000 233
10^4	0.000 000 001 666 661 9	0.002 008
10^5	0.000 000 000 016 663 1	0.040 195
10^6	0.000 000 000 000 273 1	0.290 561
10^7	0.000 000 000 000 063 9	2.699 639
10^8	0.000 000 000 000 633 3	26.838 695

表7　AMA1计算结果

N	误差值	所用时间/s
10^2	0.000 008 333 333 332 3	0.000 035
10^3	0.000 000 083 333 329 6	0.000 118
10^4	0.000 000 000 833 341 0	0.000 894
10^5	0.000 000 000 008 368 4	0.008 680
10^6	0.000 000 000 000 028 9	0.128 094
10^7	0.000 000 000 000 062 2	1.177 771
10^8	0.000 000 000 000 633 3	10.505 207

通过对计算结果的仔细观察、分析和比较,可以得出如下几点结论:

1)子区间分割数与计算时间的关系。无论对于哪一种计算功能,子区间的划分数目每增加一个数量级,计算π值所需要花费的时间基本上也随之增加一个数量级。

2)算法性能。在分割子区间数目相同的情况下,无论是从计算时间,还是从计算精度方面来看,算法2都要优于算法1。算法2在计算精度方面的优势十分明显。

究其原因,主要是因为算法2使用了反正切函数的导数函数作为母函数,计算简单,只包含一次乘法和一次除法。而算法1所使用的母函数除包含2次乘法之外,还需要多进行一次求平方根开方运算,而正是这个开方运算,既花费较多的时间,又使计算累计误差随机传播,从而降低了计算的速度和精度。

单从计算时间来看,无论是算法1还是算法2,各种矩形法的计算时间基本都在同一个量级上,而梯形

法的计算时间则要长得多。

3) 梯形法与矩形法。两者相比,由于梯形法要计算左、右边界两次,所以计算时间的相应增加远不止一倍。通过比较表2、表3、表4 与表5,以及比较表6 与表7,这一点十分明显。以子区间划分 10^8 为例,对于算法1,矩形法的计算时间大致为 10 ~ 12 s,而梯形法的计算时间则高达 36 s;对于算法2,矩形法的计算时间大致为 10 ~ 11 s,而梯形法的计算时间则高达 26.8 s。

在计算精度方面,当子区间分割数相对较小时,譬如 10^2、10^3 等,梯形法的计算精度明显高于矩形法,特别是对于算法1。当子区间分割数相对较大时,譬如 10^7、10^8 等,梯形法与矩形法的计算精度已经没有明显差别。从理论上讲,梯形法的精度要高于矩形法,但从实践方面看,结果并不尽然,有时候梯形法的精度甚至要低于矩形法。这一情况产生的主要原因是理论上总是假定计算机每一步计算都是较为精确的,也很少假定所用算法在计算过程中产生的误差的分布和传播。实际上,对计算精度的影响还远不止于此。例如,求平方根的开方运算,可能导致计算结果有效数位显著减少,从而影响最终的计算结果。这一点可以从以上使用两种算法的计算结果表看出。

4) 子区间分割数与计算误差。宏观上看,随着子区间分割数的增加,计算误差总体上呈现出下降的趋势,但这种下降不一定是单调的。对于算法1 而言,基本是单调下降的;而对于算法2 来说,则呈现波动下降的态势。

子区间分割数每增加一个数量级,算法1 总能使计算误差下降 1 至 2 个数量级。当子区间分割数大到一定程度后,如 10^8 数量级,算法2 并不能总使计算误差再继续单调下降或保持稳定,而是出现了波动。这一点如表6 和表7 所示。其主要原因可能与计算误差的随机传播有关[7~8,10]。

5 结论

本文集中讨论和分析了几种典型的计算 π 值的并行算法及其各种处理方法在基于 Windows XP 环境、MPI 并行编程环境支持和 VC + + 6.0 下 C 语言描述的编程实现问题,给出了计算程序详尽的计算结果,对比分析了各种算法及其处理方法的性能,以及它们对计算精度产生的影响。本文得出的若干分析结论是以相应算法的实际编程实现和试验计算数据为基础的,因而可信度高。

参考文献:

[1] 陈国良. 并行计算——结构·算法·编程[M]. 北京:高等教育出版社,1999.

[2] 黄 凯,徐志伟. 可扩展并行计算———技术、结构与编程[M]. 北京:机械工业出版社,2000.

[3] Takeda K, Allsopp N K, Hardwick J C, et al. An Assessment of MPI Environments for Windows NT[J]. The Journal of Supercomputing, 2001, (19): 315 - 323.

[4] Khalid Al - Tawil, Csaba Andras Moritz. Performance Modeling and Evaluation of MPI[J]. Journal of Parallel and Distributed Computing, 2001, 61(2): 202 - 223.

[5] Dale Shires, Ram Mohan. An Evaluation of HPF and MPI Approaches and Performance in Unstructured Finite Element Simulations[J]. Journal of Mathematical Modelling and Algorithms, 2001, 1(3): 153 - 167.

[6] Siegfried Benkner, Viera Sipkova. Exploiting Distributed - Memory and Shared - Memory Parallelism on Clusters of SMPs with Data Parallel Programs[J]. International Journal of Parallel Programming, 2001, 31(1): 3 - 19.

[7] Davison Germain, Alan Morris, Steven Parker, et al. Performance Analysis Integration in the Uintah Software Development Cycle[J]. International Journal of Parallel Programming, 2003, 31(1): 35 - 53.

[8] Wenheng Liu, Cho - Li Wang, Prasanna Viktor K. Portable and Scalable Algorithm for Irregular All - to - All Communication [J]. Journal of Parallel and Distributed Computing, 2002, 62(10): 1493 - 1526.

[9] Hongzhang Shan, Jaswinder Singh, Leonid Oliker, et al. Message passing and shared address space parallelism on an SMP cluster[J]. Parallel Computing, 2003, 29(2): 167 - 186.

[10] 雷英杰,郑全弟. 图像工程的基本概念和理论基础[J]. 空军工程大学学报(自然科学版), 2003, 4(4): 60 - 64.

(编辑:田新华)

(下转第 74 页)