ISSN 1009-3044
Computer Knowledge and Technology 电脑知识与技术
Vol.9, No.14, May 2013.
E-mail: kfyj@dnzs.net.cn
http://www.dnzs.net.cn
Tel:+86-551-65690963 65690964

基于四种并行计算模式的自然对数底并行计算方法

刘荣,朱建伟,李富合,李磊

(中国石油大学(华东),山东 青岛 266580)

摘要:通过讨论自然对数底e计算的并行结构,分别实现了Window 多线程、OpenMP、MPI 和OpenCL 四种语言计算e值。其中前三种是基于CPU 的并行模式,openCL基于GPU 的并行模式。根据数值实验的结果,分析了各种并行计算模式的优缺点。

关键词:并行计算;OpenMP ;OpenCL;GPU;MPI;自然对数

中图分类号:TP312 文献标识码:A 文章编号:1009-3044(2013)14-3415-05

Natural Logarithm of Parallel Computing Based Four Parallel Computing Model

LIU Rong, ZHU Jian-wei, LI Fu-he, LI Lei

(China University of Petroleum (East China), Qingdao 266580, China)

Abstract: By discussing the parallel structure of the natural logarithm base e, The calculation of the natural logarithm is achived with Window multi-thread, OpenMP, MPI and OpenCL four languages . The first three are CPU-based parallel mode, OpenCL is based GPU parallel mode. According to the results of numerical experiments, the advantages and disadvantages of the various parallel computing model is discussed.

Key words: Parallel computing; OpenMP; OpenCL; GPU; MPI; natural logarithm

1 概述

e作为数学常数,是自然对数函数的底数。它就像圆周率π和虚数单位i,是数学中最重要的常数之一,也是第一个被获证为超越数的非故意构造的数。自然对数底*e*是数学分析中使用非常广泛的无理数之一,它在金融、数学等领域[1-2]有着重要的应用。

对于自然对数底的计算,尚晓明[3]介绍了三种在教学中的常见的计算方法,而张新仁,徐化忠[4]介绍了计算机串行计算自然对数底的方法和实现。虽然自然对数底的值已经内置在很多的计算软件中,但是对于自然对数底的并行计算方法缺少相关的介绍。对自然对数底进行并行结构分析,不仅可以了解这一重要的数学常数,还可以分析不同的并行计算模式的优缺点。

2 *e*值计算算法分析

对于自然对数底e的计算公式有很多,在文中只讨论一种计算公式(1)。

2.1 串行算法分析

自然对数的公式如下:

$$e \approx 1+\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{(n-1)!}+\frac{1}{n!} \tag{1}$$

对于(1)式分析可得,计算量主要由阶乘决定。但是在串行下,我们在计算阶乘时要充分利用已计算的数值,来减少计算量。基于这种考虑,实现了串行下的计算E的算法:

对于以上算法,当N=10000000时,运行时间为78ms。但是对于多CPU的电脑,计算e时存在着资源浪费,没有利用多CPU的优势。

2.2 并行算法分析

考虑到式(1)的结构特点,它是由有限个多项式构成,在计算上有一定的独立性,结合多CPU 的架构,在这里讨论两种计算e值的并行算法。

2.2.1 多项式分段并行算法

根据算法2.1 的思想,利用计算阶乘的顺序性,可以将多项式隔项相加。对多项式(1)进行如下分割(假设n为偶数):

收稿日期:2013-04-23
基金项目:国家大学生创新性实验项目“儿童上网管理及姿态矫正”(111042557)
作者简介:刘荣(1990-),男,山东肥城县人,学生,担任项目负责人,软件框架、算法设计主要负责人。

$$e \approx 1+\frac{1}{1!}+\frac{1}{3!}+\cdots+\frac{1}{(n-1)!}+\frac{1}{n!}+\frac{1}{2!}+\frac{1}{4!}+\cdots+\frac{1}{(n-4)!}+\frac{1}{(n)!} \tag{2}$$

(2) 式是基于奇偶分成两段,即e值得计算结果是两者之和,两者的计算过程是独立的。在这里可以根据CPU的个数分成合适的段数来提高计算效率。具体的算法实现如下:

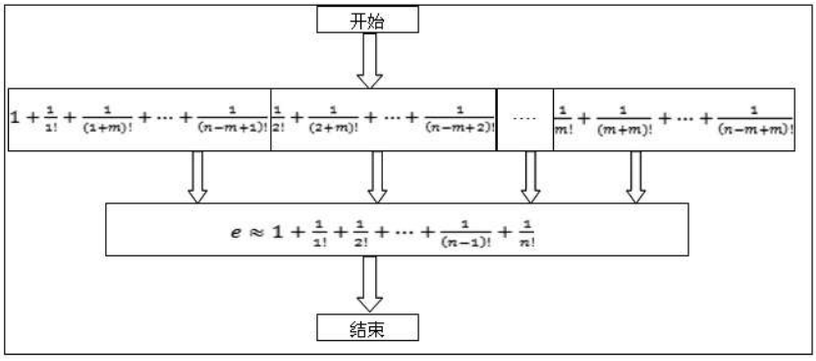

图1　并行算法结构1的流程图(假设n能被m整除)

2.2.2 基于提取公因式的并行算法

考虑到(1)式计算阶乘的特点,可以通过提取公因式方法来把多项式分割,分割成相互独立的计算过程,分配到不同的CPU上计算。

$$e \approx 1+\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{(m)!}+\frac{1}{m!}\left(\frac{1}{(m+1)}+\frac{1}{(m+1)(m+2)}+\cdots+\frac{1}{(m+1)\ldots n}\right) \tag{3}$$

对于(3)式,可看出将(1)式分成了两段,两段多项式独立计算完成后进行整理合并。在具体的计算中可以根据CPU的个数将多项式分成合适的个数。以下是算法实现。

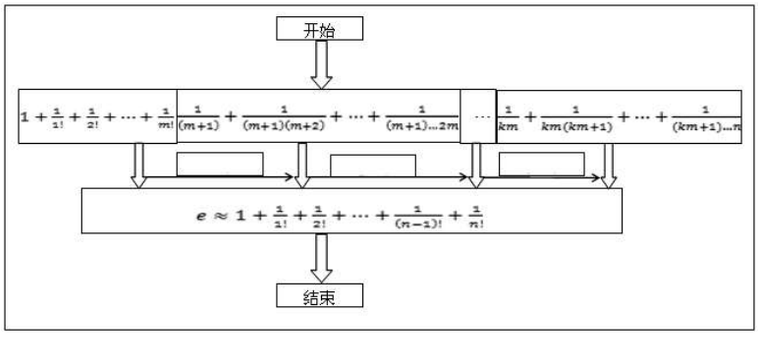

图2　并行结构2的流程图

3 并行实现

在上一节中,讨论了自然对数底e的两种并行计算结构,分析了并行计算的可行性。下面分别通过四种并行语言实现自然对数底e的计算,其中前三种语言多线程、openMP和MPI基于CPU设备,openCL是基于GPU设备。

3.1 多线程实现

最开始,线程只是用于分配单个处理器的处理时间的一种工具。程序在逻辑意义上被分割为数个线程;假如机器本身安装了多个处理器,那么程序会运行得更快,毋需作出任何特殊的调校。在多线程设计中需要注意资源共享和同步,对于自然对数底的计算,(2)和(3)式中避免了线程之间的数据竞争。对于(2)式,数据计算结构具有独立性和相同性,因此在每个线程中需要给定初值,并传回计算结果。在这里,初值是线程号,具体线程函数实现见(线程函数3.1.1),同时,每个线程计算结束后将结果传回。对于(3)式,与(2)式相比,除要传回每一个线程的计算结果外,还需要传回每一个线程所计算的阶乘,最后进行数据整理。为了方便起见,在这里将计算所得数据存放在全局数组中,具体的线程函数见(线程函数3.1.2)。相比上式,这里还要对数据进行整理,这里不再给出。

线程函数3.1.1

```
for(int i = threadID ; i <= N;i+=CpuNum)
{for(int j=0; j<CpuNum && j<i;j++)
fact *= (i-j);
e += (1.0/fact);
}
*((double *)arg) = e;
```

线程函数3.1.2

```
double fact = 1;
for(int i = start; i < end; i++)
{  fact *= i;
result += (1.0/fact);
}
ptr[0] = result;  ptr[1] = fact;
```

3.2 OpenMP实现

OpenMP[5]提供了对并行算法的高层的抽象描述,程序员通过在源代码中加入专用pragma来指明自己的意图,由此编译器可以自动将程序进行并行化,并在必要之处加入同步互斥以及通信。相比于多线程,openMP更加简单和灵活。利用openMP的编译指导语句,实现了自然对数底e的并行计算,这两种方式都是拷贝复制执行,所以要在程序中合理的利用线程号,对多项式进行分段。如下,算法3.2.1和算法3.2.2分别对应计算公式(2)和(3)。

算法3.2.1

```
#pragma omp parallel private(fact_temp,e_temp) reduction(+:e)
{   int  threadID = omp_get_thread_num();
fact_temp = 1;e_temp=0;
for(int i = threadID ; i <= StepNum;i+=CpuNum)
{
for(int j=0; j<CpuNum && j<i;j++)
{  fact_temp *= (i-j);
}
e_temp += (1.0/fact_temp);
}
e += e_temp;
}
```

算法3.2.2

```
#pragma omp parallel
{   int id = omp_get_thread_num();
int offest = StepNum/CpuNum;
int start_temp = id*offest+1;
int end_temp = (id+1)*offest+1;
double res = 0;
double fact_temp = 1;
for(int i = start_temp; i < end_temp; i++)
{
fact_temp *= i;
res += (1.0/fact_temp);
}
result[id*2] = res;
result[id*2+1] = fact_temp;
}
```

循环并行化是使用OpemMP来并行化程序的重要部分,它是并行区域编程的一个特例。使用parallel for编译指导语句能将for循环中的工作分配到一个线程组中,而每个线程组中的每一个线程将完成循环中的一部分。对于算法3.2.2,还可以利用openMP的编译指导语parallel for进行简化。这样可以简化程序,方便串行程序到并行程序的转换。

3.3 MPI实现

MPI是基于消息传递的并行计算模式,建立消息传递标准的主要优点是可移植性和易于使用。以低级消息传递程序为基础的较高级和(或)抽象程序所构成的分布存储通信环境中,标准化的效益特别明显。随着高性能计算技术的普及,MPI标准如今已经成为事实意义上的消息传递并行编程标准,也是最为流行的并行编程接口。

算法3.3.1

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
fact = 1;e = 0;
for(i = myid+1; i <= StepNum;i+=numprocs)
{
for( j=0; j<numprocs && j<i;j++)
fact *= (i-j);
e += (1.0/fact);
```

```
}
MPI_Reduce(&e, &myE, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

在这里,我们只实现自然对数底e的第一种计算模式(即式(2))。通过分析式(2)的计算结构,利用MPI的广播机制和规约机制,可以实现自然对数底e值的计算。广播机制是一对多的传送消息,它的作用是从一个root进程向组内所有其它进程发送一条消息。而规约机制则可以将子线程计算的数据存入root进程数据中,即在计算e值得公式中,对多段多项式的结果进行规约。具体的代码见(算法3.3.1)。

3.4 OpenCL实现

随着GPGPU在超级计算环境中的日益普及,它在应用领域表现出优秀的性能功效比和性价比优势。OpenCL作为GPGPU的并行语言的一种,OpenCL [6]是一个为异构平台编写程序的框架,此异构平台可由CPU,GPU或其他类型的处理器组成。OpenCL基于kerels函数运行,执行时也是拷贝复制执行模式。参考上面三种语言的并行实现计算自然对数底e,可以很快的得到所对应Kernel函数,在kernels中要注意合理利用线程ID号来分配多项式的起始位置。

对于OpenCL而言,在运行Kernel函数时,要完成一系列的初始化工作[2],同时需要选取合适的维数和大小,在这里我们采用一维的线程映射。以下是对应的kernel函数。

算法3.4.1

```
__kernel
void CaluE(__global float* result, int StepNum,int MaxItem)
{
int id = get_global_id(0);
float fact = 1;float e = 0;
for(int i = id+1; i <= StepNum;i+=MaxItem)
{
for(int j=0; j<MaxItem && j<i;j++)
{
fact *= (i-j);
}
e += (1.0/fact);
}
result[id] = e;
};
```

算法3.4.2

```
__kernel
void CaluE_2(__global float* result, int StepNum,int MaxItem)
{
int id=get_global_id(0);float start,end,res;
int offest = StepNum/MaxItem;
start = id*offest+1;end = (id+1)*offest+1;
res = 0;float fact = 1;
for(int i = start; i < end; i++)
{
fact *= i;
res += (1.0/fact);
}
result[id*2] = res;
result[id*2+1] = fact;
barrier(CLK_LOCAL_MEM_FENCE);
};
```

4 分析与讨论

上文中,我们分析了自然对数底e的并行计算结构和具体语言实现。为了分析四种并行计算语言的优缺点,我们在特定的设备(AMD 2*1)上进行数值试验。由于自然对数底E的计算精度不是关注的重点,在这里不考虑计算机的精度截断(即计算机精度不够,导致无用计算)。

4.1 两种并行算法比较

分析自然对数底e的两种并行计算结构,可以发现式(2)的并行结构相互独立,没用利用阶乘计算的依赖性。式(3)虽然考虑了阶乘的依赖性,但是增加了线程之间的数据传递。为了分析两种并行计算结构的效率,采用了前两种并行语言进行数值试验,结果如表1(步数1000000)。

表1 两种并行算法比较

	第一种并行结构	第二种并行结构
多线程	19.236	13.609
OpenMP	20.236	12.609

由表4.1.1可以看出,第二种并行结构在并行模式下具有更高的计算效率,数据传输所用的时间相对于重复阶乘的计算所用的时间很小。因此,在并行算法设计时,要对比减少通讯和增加通讯所用的时间多少,合理设计算法达到良好的并行计算效率。

4.2 并行语言分析

在文中,采用四种并行语言实现了自然对数底e值的计算,四种并行语言在实现方式上各有不同。为了对比它们的各自优势,对两种并行并行计算模式进行数值试验。为了分析的方便,以下的数值试验所采取的计算步数为1000000000,数值试验结果如表2。

表2 四种并行语言的比较

	串行	多线程	OpenMP	MPI	OpenCL(work_item=20)
第一种并行结构	24.185	19.236	20.236	18.23	17.680
第二种并行结构	24.185	13.609	12.609		4.376
加速比1	1	1.25	1.19		1.36
加速比2	1	1.77	1.92		5.5

由上表的实验数据可知,多线程和OpenMP结果大体差不多,这与实际是相符合的,OpenMP是对多线程的封装,两者在实现原理上是一样的。在CPU上,区别于前两者的并行实现,MPI是基于通信的并行实现,对于大数据的传输要考虑数据传输的时间,合理调整程序。不同于前者三种并行语言,OpenCL在GPU上的实现是基于显卡的,GPU的计算能力是低于GPU的,但是GPU的众核计算效率是大于单一GPU的。

5 结论

本文通过自然对数底e值并行计算的四种语言实现,分析了四种并行语言的实现特点和计算优势。由实验结果可以看出,CPU和GPU的并行计算还是有一定差异的,需要根据各自的特点进行协调配合,可达到更好的计算效率。

参考文献:

[1] 陈仁政.不可思议的e[M].北京:科学出版社,2005:1-88.
[2] BRIAN J M.e:数中之大师[J].余敏安,译.数学译林.2007(3):213-226.
[3] 尚晓明.超越数π与e的计算方法及其应用[J].焦作大学学报,2012(1):73-74.
[4] 张新仁,徐化忠.自然对数的近似计算[J].山东电大学报,2012(3):64.
[5] 多核系列编写组.多核程序设计[M].北京:清华大学出版社,2007.
[6] Benedict R.Gaster Lee Howes David R.Kaeli. OpenCL 异构计算[M].北京:清华大学出版社,2012.