并行计算 实验一

PB19071501 李平治

I. 问题描述

计算出自然常数的前100位有效数字。自行参考合适的数值方法（数值积分、无穷级数、蒙特卡洛等）实现。选取OpenMP、MPI、CUDA中的一种实现。可以调用第三方高精度计算库，数值方法必须自行实现。

前200位有效数字参考结果：

2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742746639193200305992181741359662904357290033

II. 算法设计

i.问题分析

计算自然常数的方法包括数值积分、无穷级数和蒙特卡洛模拟等。蒙特卡洛模拟要想收敛到要求的精度，需要多得多的迭代时间；数值积分方法和无穷级数在计算复杂度上大致相同，但无穷级数更容易编码。因此，本人选择无穷级数方法来计算，所使用级数如下:

$$e = \Sigma_{k=0}^{+inf} \frac{1}{k!}$$

（参考: 麦克劳林级数[Colin Maclaurin, 18th century]）

ii.算法描述

- 该算法的串行设计如下:

```
total = 0
for k=0 to N-1:
  item = 1 / factorial(k)
  total += item
```

- 整个循环的每一步都互不依赖，因此可以做并行化设计，以下为PCAM设计方法学的描述

1. 划分：将级数计算的每一项作为一个单独划分，以及最终求和计算作为一个划分.
2. 通信：结构化通信。级数的“每一项计算”和最终”求和计算“组成以“求和计算”为根、“每一项计算”为叶子的树.
3. 组合：按照处理器个数N将“每一项计算”的任务平均分为N部分，在这些处理器上进行部分求和后再传到最终求和的处理器；相比每次计算出每一项后都进行通信，这样可以极大减少通信时间，优化性能.
4. 映射：由于该算法具有结构化的局部通信，映射设计十分显然.

总得来说，本次算法采用集中式顺序求和方法，伪代码如下:

```
total = 0
for i=0 to proc_number-1 par-do:
  sub_total[i] = 0
  for k=0 to N/proc_number-1:
    item = 1 / factorial(proc_number*k+i)
    sub_total[i] += item

for i=0 to proc_number-1:
  total += sub_total[i]
```

III. 实验评测

i.实验配置

本次实验在服务器上运行，相关配置如下:

- **24v**CPUs (Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz)
- MemTotal: 188GB
- Linux version 3.10.0-1160.el7.x86_64 (gcc version 4.8.5, Red Hat 4.8.5-44)
- **Python** 3.9.11, 使用并行库 **MPI: openmpi 4.0.2**

ii.实验结果

a. 正确性验证

*e*的前100位应当为:

```
2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642 7
```

用级数前20000项计算的实际结果为

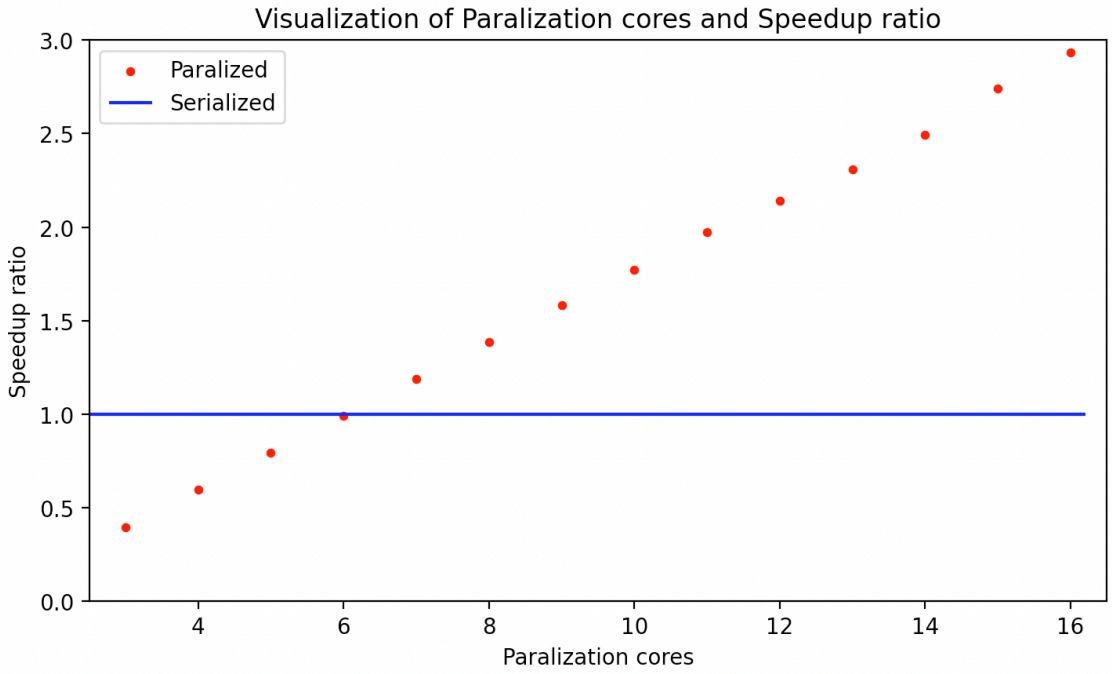
```
e = 2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642 7
```

表明结果正确，符合要求.

b.加速比分析

选择处理器核数3-16的并行程序和串行程序进行分析:

并行处理器核数	程序运行时间/s	加速比
串行	746.980	1.0000
3	1891.768	0.3949
4	1254.946	0.5952
5	942.857	0.7923
6	754.498	0.9900
7	628.950	1.1877
8	538.581	1.3869
9	472.387	1.5813
10	421.443	1.7724
11	378.473	1.9737
12	348.644	2.1425
13	323.303	2.3105
14	299.748	2.4920
15	272.551	2.7407
16	254.671	2.9331



可以看出，基本符合线性加速比，但远不及理想情况下的加速比。

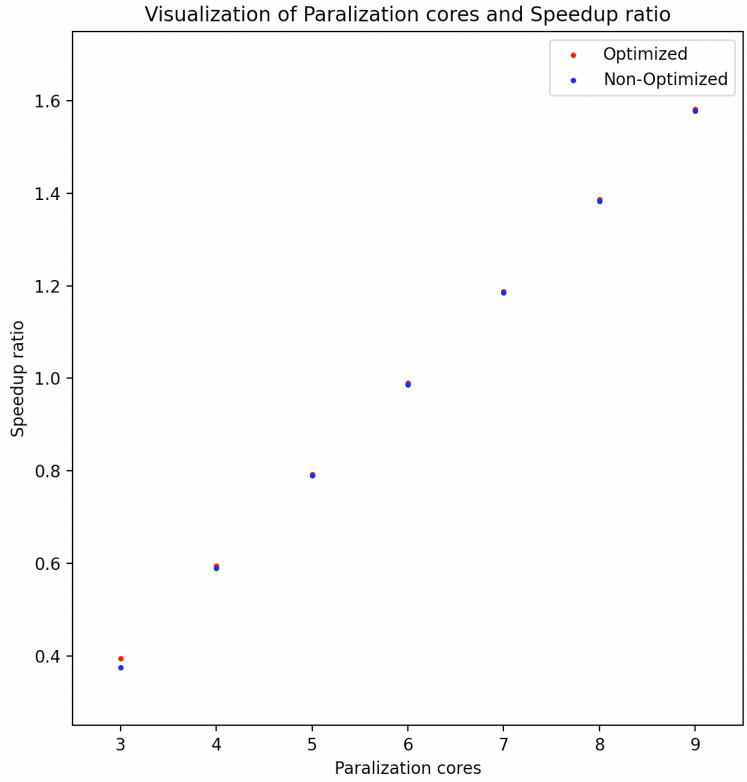
分析并行核数较小的情况，发现并行进程间通信消耗了非常多的时间，以至于在$N+1\leq 6$（"+1"是因为实际上有一个核仅承担了汇总求和的工作，计算量相对于其他核来说可以忽略不计）的情况下并行效果甚至差于串行.

c.消融实验

本次实验采用了"前N-1个进程计算完该核上的项之和后再传递给第N个进程"的通信优化策略，为了验证该优化的效果，本人进行了以下测试：

选取$N=3,4,5,6,7,8,9$并且没有该优化的情况进行时间统计：

并行处理器核数	程序运行时间/s	加速比
3	1991.987	0.3750
4	1267.008	0.5896
5	945.445	0.7901
6	756.898	0.9869
7	630.055	1.1856
8	539.890	1.3836
9	473.281	1.5783



可以看出，减少通信次数确实有不同程度的优化效果。

然而，当并行核心数N较小时，优化效果明显；当并行核心数N较大时，优化效果较弱。猜测可能是由于算法的树形通信结构，通信次数随着N的增大而增大，因而优化效果变弱。

IV. 结论

本次实验考量了不同并行数下的并行算法效果，同时对比测量了通信优化的实际效果。结果表面随着并行数的增加，运行时间通常有所减少，加速比增加；随着通信次数的减小，加速比也会随之提高。

参考

[1] Maclaurin's series: https://mathworld.wolfram.com/MaclaurinSeries.html

[2] MPI for Python: https://mpi4py.readthedocs.io/en/stable/#

附录

本次实验的代码都与本文档一同打包提交，其中`base.py`为串行算法作对比用，`parallel.py`为并行算法。