并行计算实验一

PB19030888 张舒恒

问题描述

计算自然常数e的前100位有效数字，采用合适的数值方法(如数值积分，无穷级数，蒙特卡洛模拟)实现。选取OpenMP，MPI，CUDA其中一种并行策略。

算法设计

问题分析

首先我们要考虑用哪种数值方法，若采用**无穷级数**[1]，易知e的泰勒展开为$\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots$，由于阶乘运算是迭代运算所以是不可并行的，即使是第三方高精度库也是对小数字(约100以内)的阶乘缓存和预取，对大数字的阶乘仍然是迭代运算，所以此部分不可并行。但求和操作可以并行，且在利用第三方高精度库内置了部分正整数阶乘结果的前提下可以减少乘法运算时间占总时间的比重，提高可并行的加法运算时间比重，从而有效提高加速比。为了并行加法运算需要预先将阶乘运算结果导入$mpfr_t\ \ item[n]$，其中n用于精度控制。对于负载均衡，如果是利用数组自己设计高精度运算则负载不完全均衡，因为有无限小数和有限小数，由于精度问题运算并不总是时间相同的；如果是调用高精度库，则会有统一的精度管理，在此条件下的小数加法运算可以认为是负载均衡的。

若采用**蒙特卡洛模拟**[2]，可以利用$\int_1^2\frac{1}{x}dx=ln2$求出$ln2$再利用指数运算求出$e$，虽然每次在区域内随机撒点并记录是否在曲线$xy=1$下方这一操作可以很好地进行并行化，但是由于是泛随机化模拟，这种方法收敛速度很慢精度十分不足，且最后一步的浮点数指数运算还会受到机器硬件的精度限制，并且使用高精度库也不能改变机器硬件对于这种运算精度的限制。还可以利用**概率论与数理统计**的知识，模拟n次将m个信封随机分发给m个人，统计全错位排列数r，极限情况下$\frac{r}{n}$即是e，这种方法和前面积分模拟法类似，可以很好地进行并行化，但是收敛速度很慢，也存在着精度困难的问题。蒙特卡洛模拟是负载完全均衡且无需额外的并行工作量的良好可并行算法。

算法描述

对于**无穷级数**方法，算法设计是符合PCAM设计理念的，例如如果需要计算$\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\ldots+\frac{1}{100!}$，则预先调用高精度库串行计算阶乘结果，再进行划分操作，划分方法是每25项交给一个处理器进行加法运算，最后的合并即为四个处理器结果相加。伪代码如下：

```
for i = 1 to 100:
    mpfr_set_d (u, double(i), MPFR_RNDD)
    mpfr_div (t[i+1], t[i], u, MPFR_RNDD)
#pragma omp parallel for
for i = 1 to 100:
    mpfr_add (s, s, t[i], MPFR_RNDD)
```

对于**蒙特卡洛模拟**方法，算法设计也是符合PCAM设计理念的，划分方法是每个处理器同时对区域进行随机取点，并记录在曲线上方还是下方，最后没有合并操作，由曲线下方点数除以总点数即可求出$ln2$，再利用指数运算求出e，伪代码如下：

```
#pragma omp parallel for
x = random.uniform(1,2)
y = random.uniform(0,1)
if y < 1/x:
    counts += 1
else:
    do nothing
```

实验评测

实验配置

软硬件配置

CPU参数：

Processor (CPU)	
CPU Name	11th Gen Intel® Core™ i7-11600H @ 2.90GHz
Threading	1 CPU - 6 Core - 12 Threads
Frequency	4192.09 MHz (42 * 99.81 MHz) - Uncore: 3493.4 MHz
Multiplier	Current: 42 / Min: 8 / Max: 46
Architecture	Tiger Lake / Stepping: R0 / Technology: 10 nm
CPUID / Ext.	6.D.1 / 6.8D
IA Extensions	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, AES, AVX, AVX2, AVX512F, FMA3, SHA
Caches	L1D : 48 KB / L2 : 1280 KB / L3 : 18432 KB
Caches Assoc.	L1D : 12-way / L2 : 20-way / L3 : 12-way
Microcode	Rev. 0x34
TDP / Vcore	45 Watts / 1.117 Volts
Temperature	94 °C / 201 °F
Type	Retail
Cores Frequencies	#00: 4192.09 MHz #01: 4192.09 MHz #02: 4192.09 MHz #03: 4192.09 MHz #04: 4192.09 MHz #05: 4192.09 MHz

RAM参数：

Memory (RAM)	
Total Size	32768 MB
Type	Dual Channel (128 bit) DDR4-SDRAM
Frequency	1595.9 MHz (DDR4-3192) - Ratio 3:48
Timings	22-22-22-52-1 (tCAS-tRCD-tRP-tRAS-tCR)
Slot #1 Module	16384 MB (DDR4-3200) - P/N: MG8A08WQV00-S3200A
Slot #2 Module	16384 MB (DDR4-3200) - P/N: MG8A08WQV00-S3200A

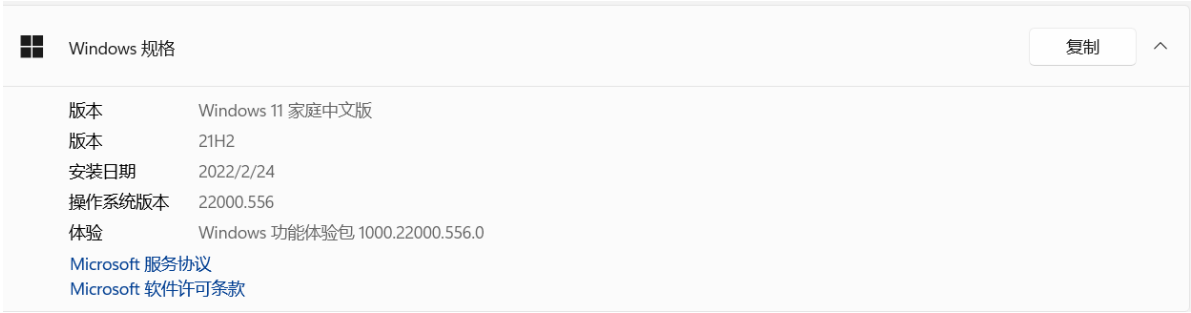
GPU参数：

Graphic Card (GPU)	
GPU #1 Type	NVIDIA GeForce RTX 3050 Laptop GPU (GA107) @ 1500 MHz
GPU #1 Brand	ASUSTeK Computer Inc.
GPU #1 VRAM	4096 MB @ 6001 MHz
GPU #2 Type	Intel(R) UHD Graphics
GPU #2 Brand	ASUSTeK Computer Inc.

编译器参数：

```
C:\Users\凝雨>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=c:/mingw/bin/../libexec/gcc/mingw32/9.2.0/lto-wrapper.exe
Target: mingw32
Configured with: ../src/gcc-9.2.0/configure --build=x86_64-pc-linux-gnu --host=mingw32 --target=mingw32 --disable-win32-regi
stry --with-arch=i586 --with-tune=generic --enable-static --enable-shared --enable-threads --enable-languages=c,c++,objc,obj
-c++,fortran,ada --with-dwarf2 --disable-sjlj-exceptions --enable-version-specific-runtime-libs --enable-libgomp --disable-l
ibvtv --with-libiconv-prefix=/mingw --with-libintl-prefix=/mingw --enable-libstdcxx-debug --disable-build-format-warnings --
prefix=/mingw --with-gmp=/mingw --with-mpfr=/mingw --with-mpc=/mingw --with-isl=/mingw --enable-nls --with-pkgversion='MinGW
.org GCC Build-2'
Thread model: win32
gcc version 9.2.0 (MinGW.org GCC Build-2)
```

操作系统参数：



第三方高精度库MPFR版本参数：

```c
/* Define MPFR version number */
#define MPFR_VERSION_MAJOR 4
#define MPFR_VERSION_MINOR 0
#define MPFR_VERSION_PATCHLEVEL 2
#define MPFR_VERSION_STRING "4.0.2"
```

数据集配置

本实验中**无穷级数**方法所用到的阶乘结果均是调用第三方高精度库迭代运算的结果，**蒙特卡洛模拟**方法所用的随机数是由$math.random()$方法生成，这样做的原因是C++的第三方高精度库可以实现任意硬件条件能满足的精度需求，python的随机数生成不需要随机数种子，使用更加方便。

实验结果

正确性验证

使用**无穷级数**方法计算得出e = 2.71828 18284 59045 23536 02874 71352 66249 77572 47093 69995 95749 66967 62772 40766 30353 54759 45713 82178 52516 64274 27466 39193 20030 59921 81741 35966 29043 57290 03342 95260 59563 07381 32328 62794 34907 63233 82988 07531 95251 01901 15738 34187 93070 21540 89149 93488 41675 09244 76146 06680 82264 80016 84774 11853 74234 54424 37107 53907 77449 92069 55170 27618 38606 26133 13845

83000 75204 49338 26560 29760 67371 13200 70932 87091，这个结果是完全准确的。使用**蒙特卡洛模拟**方法只能得出e = 2.71828 18，精度十分不足。

加速比分析

使用**无穷级数**方法固定问题规模为100位精度，进行100次测试取平均结果，加速比和线程数关系如下表：

加速比	线程数
1.13435	2
1.43436	4
1.60901	6
2.09013	8
2.62345	10
2.83402	12

可以看出在问题规模为100位精度时加速效果并不明显，加速比随线程数增大增长缓慢，主要原因是精度较小时可并行的加法运算时间占总时间比重较小。

使用**蒙特卡洛模拟**方法固定问题规模为7位精度，进行100次测试取平均结果，加速比和线程数关系如下表：

加速比	线程数
1.60892	2
3.40782	4
5.10973	6
7.39210	8
8.67012	10
11.09218	12

可以看出**蒙特卡洛模拟**方法可以较好地进行并行化处理，加速比随线程数增大而线性增大，在一些情况下比较接近于线程数。

可扩放性分析

由于**蒙特卡洛模拟**方法本身的精度问题，模拟近10亿次才可得到准确的7位精度，很难调整问题规模进行研究，所以在此仅进行**无穷级数**的可扩放性分析。由于我的CPU-i7-11600H是**12线程**的，所以这里以默认线程数进行实验，将问题规模逐步增大至10 0000位精度，进行100次测试取平均结果，加速比和问题规模关系如下表：

加速比	精度要求
2.84739	100
4.83021	1000
7.09072	1 0000
7.69201	10 0000

可以看出当精度足够大时加速效果得以体现，这和上述问题分析中关于加速比和精度要求的关系的预测是相吻合的。

结论

求自然常数e的**无穷级数**方法收敛速度快，精度高，但在问题规模较小时加速效果不理想，问题规模较大时有一定的加速效果；**蒙特卡洛模拟**方法收敛速度慢，精度低，但自身具有良好的可加速性。

致谢

在进行该实验的过程中遇到了配置高精度环境和OpenMP环境的小困难，在此感谢热心同学的帮助以及助教的解答。

参考文献

[1]无穷级数求和初探-知乎专栏无穷级数求和（1） - 知乎 (zhihu.com)

[2]蒙特卡洛模拟浅析蒙特卡洛模拟(Monte Carlo Simulation)浅析 - 知乎 (zhihu.com)