第二次作业

黄道吉-1600017857

先期工作

近似函数的benchmark

基于这里的代码进行速度和误差的测试

课程机器上的结果(cpu)

近似函数	时间	误差的和
1 / (1 + exp(-x))	95.8ns	0
atan(pi * x / 2) * 2 / pi	27.9ns	7.1804149573
x / sqrt(1 + x^2)	17.3ns	0.1520901924
erf(sqrt(pi) * x / 2)	6.9ns	0.2417127401
tanh(x)	5.7ns	0.1840642978
x / (1 + x)	8.8ns	17.5337329089

采用 tanh(x) 进行当x较大时的估计, 发现当x > 1k时, 计算的误差的和就已经小于1e-7了

并行化处理

将1e5个数拆成100 * 100 * 10的形式, 即10 * 10个block, 每一个block中10 * 10个线程, 每一个线程对10个数求和

汇总每一个block中的结果到全局变量中, 最终在cpu中求和

时间优化

IO时间

采用fread()函数读取数据, 根据这篇博客的数据, fread()和mmap()是比较快的方法

代码实现如下

```
void read_input(double* input, int size) {
    FILE* fp = fopen(FILE_NAME, "r");
    if (fp) {
        for (int i = 0; i < size; i += 1) {
            fscanf(fp, "%lf\n", &input[i]);
        }
    }
    else {
        printf("error: reading input, in read_input()\n");
    }
}
```

汇总数据

对每一个block中的数据, 采用类似树状数组求和的方法, 先将[64, 100)之内的数加到[0, 32)中, 再对[0, 64)中的数据类似二叉树形式的求和, 每一步将idx = i的数据存储到idx = i/2中.

代码实现如下

```
if (tid > 63) {
    tmp[tid - 36] = tmp[tid] + tmp[tid - 36];
}
__syncthreads();
int i = 32;
while (i != 0) {
    if (tid < i) {
        tmp[tid] = tmp[tid + i] + tmp[tid];
    }
    __syncthreads();
    i /= 2;
}
if (tid == 0) {
    gpuans[bid] = tmp[0];
}
```

实验结果

从warmup之后开始计时, 到free结束之截止, 时间共计50ms(课程机器上). 但这是用cpu计时得到结果, 可能会有误差

用nvprof详细分析所用时间, 发现(包括warmup在内的)cudaMalloc耗时最长, 而除去warm丿后, 实际的cudaMalloc只用时1ms; cudaMemcpy用时0.3ms; 实际计算函数(calc)用时27μs, 这还包含在每一个block内进行reduce的过程, 按照课程群中的讨论的计时方法(不计算warmup, 不计算memcpy, 不计算reduce), 可以认为程序的运行时间为27μs, 即0.027ms.

已编译程序和代码在课程机器上的位置为

```
src=/usr/Huang.Daoji/2.cu
bin=/usr/Huang.Daoji/a.out
```

```
==11633== Profiling application: ./a.out
==11633== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 99.22%  3.9313ms         3  1.3104ms  1.6960us  3.7981ms  [CUDA memcpy HtoD]
  0.71%  27.968us         1  27.968us  27.968us  27.968us  calc(double*, double*
)
  0.07%  2.7840us         1  2.7840us  2.7840us  2.7840us  [CUDA memcpy DtoH]

==11633== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 98.18%  367.38ms         3  122.46ms  387.24us  366.59ms  cudaMalloc
  1.31%  4.8986ms         4  1.2246ms  23.669us  4.5321ms  cudaMemcpy
  0.17%  639.95us         2  319.97us  313.95us  326.00us  cudaGetDeviceProperti
es
  0.15%  567.45us         3  189.15us  125.22us  300.19us  cudaFree
  0.09%  352.56us        91  3.8740us     260ns  120.07us  cuDeviceGetAttribute
  0.06%  222.84us         1  222.84us  222.84us  222.84us  cuDeviceTotalMem
  0.02%  69.074us         1  69.074us  69.074us  69.074us  cudaLaunch
  0.01%  31.382us         1  31.382us  31.382us  31.382us  cuDeviceGetName
  0.00%  16.068us         1  16.068us  16.068us  16.068us  cudaSetDevice
  0.00%  5.3750us         3  1.7910us     357ns  4.2460us  cuDeviceGetCount
  0.00%  4.3810us         1  4.3810us  4.3810us  4.3810us  cudaGetDeviceCount
  0.00%  3.2300us         1  3.2300us  3.2300us  3.2300us  cudaConfigureCall
  0.00%  1.9730us         3     657ns     395ns  1.1110us  cuDeviceGet
  0.00%  1.4120us         2     706ns     337ns  1.0750us  cudaSetupArgument
[manycore@master Huang.Daoji]$
```

2018-05-08