**π Report by Songtao**

1. 算法选择

使用BBP算法迭代求解：

 \pi = \sum_{k = 0}^{\infty}\left[ \frac{1}{16^k} \left( \frac{4}{8k + 1} - \frac{2}{8k + 4} - \frac{1}{8k + 5} - \frac{1}{8k + 6} \right) \right].

迭代公式如上，只需要实现高精度的小数加减法，以及高精度数除低精度数。对于高精度数采用进制进行存储。 对于结果输出，采用每次乘以10，最终的溢出进位就是小数点后的数位（十进制），使用这种方法输出结果。

1. 主要函数功能

|  |  |
| --- | --- |
| 函数名 | 功能 |
| NumberNew | 分配空间并初始化 |
| NumberFree | 释放空间 |
| NumberAdd | 高精度加法 |
| NumberMinus | 高精度减法 |
| NumberDiv | 高精度除以低精度 |
| NumberMul10 | 高精度乘10 |
| NumberDump | 输出小数部分 |
| BBP\_K | 计算每一次迭代的结果 |

1. 并行化

使用OpenMP实现线程级并行。

#pragma omp parallel for schedule(dynamic,64)

for (i = 1; i < K\_Obj; i++)

{

BBP\_K(i);

}

BBP\_K(K\_Obj); // sum

每一个线程都有一些自己的私有空间，计算时将结果存在每个线程自己的空间中，最终将各个线程的结果相加得到最终的结果。每个线程计算哪些是动态决定的，便于各个线程间负载平衡的实现，以及减少线程间等待。

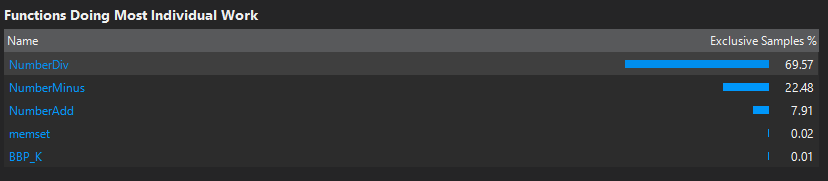
1. 评测
2. 八线程情况下，运行时间随规模变化：

可以看出时间并非现行增加，因为随着计算位数增加，高精度位数和迭代次数都随着增加，大概是个O()的量级。

1. 规模固定（N=100,000），加速比随线程数变化：

在8线程处可以明显的看到拐点，其实在4线程处也应该有一个小的拐点，主要是由于超线程造成的。

1. VS2013 性能分析工具



用VS的工具简单看了下，发现除法函数占用的时间最多。可以作为优化的入手点。

1. 可以改进的地方

由于现在的所有for循环都有较强的迭代间依赖，因此编译器很难自动向量化。为此可以手动使用一些内建函数进行优化，可以快很多。但可能需要较多的内存开销，不过整体的内存开销就不大，再多点也是可以接受的。所以这个地方可以改进。

1. 总结

学习使用了下之前从来没有用过的单元检测，感觉挺有意思的！