Optlab 实验报告

郭丹琪 2018202067

一. 优化思路

1. 考虑到每次对 lineorder_table_info 里元素的访问都可能带来内存引用，而在循环里又对 lineorder_table_info 里元素进行了多次访问，所以考虑在每次循环的一开始先把 lineorder_table_info 里需要的元素存到临时变量里。随后发现，在循环里不是直接使用 lineorder_table_info 里的元素，而是每次都将其进行一定的运算，所以考虑在循环的一开始把 lineorder_table_info 里的元素存到临时变量的时候直接对其进行运算。所以在每次循环的一开始，将 lineorder_table_info.table -> lo_quantity[i]、lineorder_table_info.table -> lo_extendedprice[i] * (1 - lineorder_table_info.table -> lo_discount[i])、lineorder_table_info.table -> lo_extendedprice[i] * (1 - lineorder_table_info.table -> lo_discount[i]) * (1 + lineorder_table_info.table -> lo_tax[i])都计算好，并存在临时变量中，以消除不必要的内存引用和重复的计算。

2. 考虑到循环次数很多，所以可以用循环展开的方式减少循环的迭代次数。同时，为了提高并行性，使用多个累计变量，这样可以利用多个功能单元和它们的流水线能力。因为考虑到循环中需要的累积变量较多，可能会有寄存器溢出的问题，所以只尝试了 2*2 循环展开和 4*4 循环展开。2*2 循环展开是在每次循环中处理索引值相邻的两组元素，并在循环前对 quantity_sum、discount_total_price、tax_discount_total_price、quantity_sum_with_condition、discount_total_price_with_condition、tax_discount_total_price_with_condition 都设置了一个另外的累积变量，在循环中，索引值为偶数的存在原累积变量中，索引值为奇数的存在新的累积变量中。将原本循环的边界改成 for (i = 0; i < length ; i+=2)，其中 length=lineorder_table_info.rows-1，并且在该循环结束，对剩下的没有遍历到的向量元素再进行处理。当所有遍历结束后，再将原累积变量和新累积变量的值加起来。

二. 优化效果

尝试了 2*2 循环展开和 4*4 循环展开后，发现 4*4 循环展开的效果不如 2*2 循环展开，分析可能是寄存器溢出的原因，所以最后使用 2*2 循环展开。

因为本机上运行不了，所以直接在服务器上进行了测试。由于服务器性能较不稳定，每次运行的时间变化有所波动，但是基本都比原程序运行的时间要短。其中用服务器上 lineorder.tbl(s=0.1)的数据集时，原程序运行时间为 100492906，而优化后程序的运行效果最好的几次结果为：

```
2018202067@VM-0-46-ubuntu:~$ g++ query.cpp -o query -O1 -mavx512dq
2018202067@VM-0-46-ubuntu:~$ ./query
153078795
11482644073920.253906
11942201064624.316406
81150336
6087293950606.017578
6330745210686.654297
running time is 51650680
```

```
2018202067@VM-0-46-ubuntu:~$ ./query
153078795
11482644073920.253906
11942201064624.316406
81150336
6087293950606.017578
6330745210686.654297
running time is 51077426
```

相当于改进了约 2 倍。其中浮点数的计算因为运算顺序的问题，结果跟原程序略有差异，但是差异不超过 1，在可接受范围内。