并行程序设计实验:CPU 架构编程

Github 仓库中包含算法的 C++ 代码，数据处理和图像绘制的 Python 代码，实验报告的 Latex 源文件，实验时的部分截图。地址：https://github.com/Yifan-Guan/Parallel-lab-1-CPU-programming.git

一、N 阶方阵与向量内积

1、算法设计

1.1、平凡算法

逐列访问矩阵元素，一步外循环得到一个内积。

```
void inner_product(int n, int* a, int** b, int* s)
{
        for (int i = 0; i < n; i++) {
                s[i] = 0;
                for (int j = 0; j < n; j++) {
                        s[i] += a[i] * b[i][j];
                }
        }
}
```

1.2、cache 优化算法

逐行访问矩阵元素，一步外循环只会得到内积的一个累加项。

```
void inner_product(int n, int* a, int** b, int* s)
{
    for (int i = 0; i < n; i++) {
        s[i] = 0;
    }
```

Perf report of inner product			
Program	L1-dcache-loads	L1-dcache-load-misses	L1-dcache-prefetches
Trivial	697614965	7722901	5268536
Optimized	630400939	7577799	4972996

表 1: Perf 分析结果

```
6       for (int j = 0; j < n; j++) {
7           for (int i = 0; i < n; i++) {
8               s[i] += a[i] * b[j][i];
9           }
10      }
11  }
```

2、编程实现

Arm 平台：WSL（2.4.13.0）Ubuntu（22.04.5 LTS）系统下使用 aarch64-linux-gnu-g++ ((Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0)交叉编译,在 qemu-aarch64（6.2.0 (Debian 1:6.2+dfsg-2ubuntu6.25)）上执行。

x64 平台：WSL（2.4.13.0）Ubuntu（22.04.5 LTS）系统下使用 g++ (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0 编译执行。

3、性能测试

测试数据中，向量 a:$a_i = i^2$，矩阵 b:$b_{ij} = i + j$，问题规模迭代次数取 100（每迭代一次，n 增加 10），运行时间使用 gettimeofday() 函数获取。在 arm 平台上的测试结果与在 x64 平台上的测试结果分别绘制为折线图 1和折线图 2。

4、profilling

x64 平台，WSL（2.4.13.0）Ubuntu（22.04.5 LTS）系统下使用 perf（5.15.178）分别对程序运行时的 Cache load, Cache load miss, Cache prefetches 指标进行分析，参数设置与上一节相同。得到结果绘制为表格 1。针对 perf 的分析报告，使用 hotspot（1.3.0）绘制火焰图，分别为图 3和图 4。

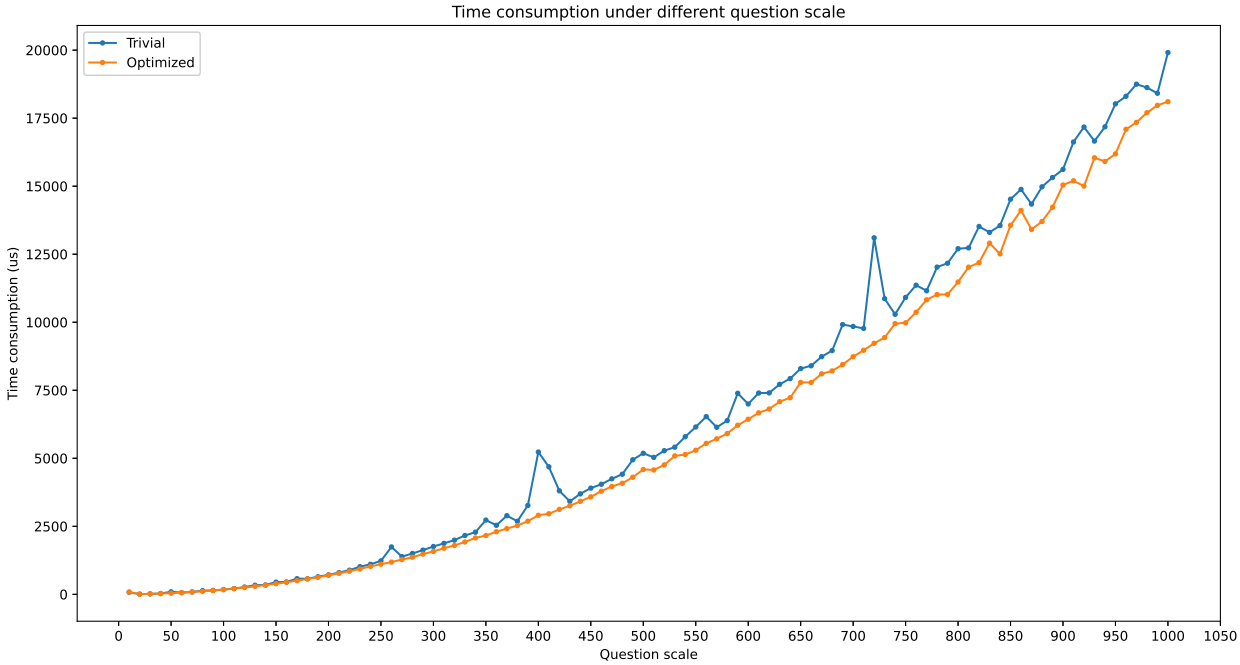

图 1: Product time consumption of arm

5、结果分析

从折线图 1和折线图 2不难看出，使用了 Cache 优化算法运行时间显著降低，且问题规模越大降低作用越明显。由于本实验中的 arm 环境是由 WSL+qemu 获得，性能上也显著弱于 x64 平台。由 perf 分析结果表格 1可知，Cache 优化算法的 Cache loads, Cache load misses, Cache prefetches 三项指标均低于平凡算法，Cache load missed 火焰图也很好地佐证了这一点。

二、N 个数求和

1、算法设计

1.1、平凡算法

链式算法，将所有元素累加得到最终结果。

```
1 int sum(int n, int* a)
2 {
3     int result = 0;
4     for (int i = 0; i < n; i++) {
```

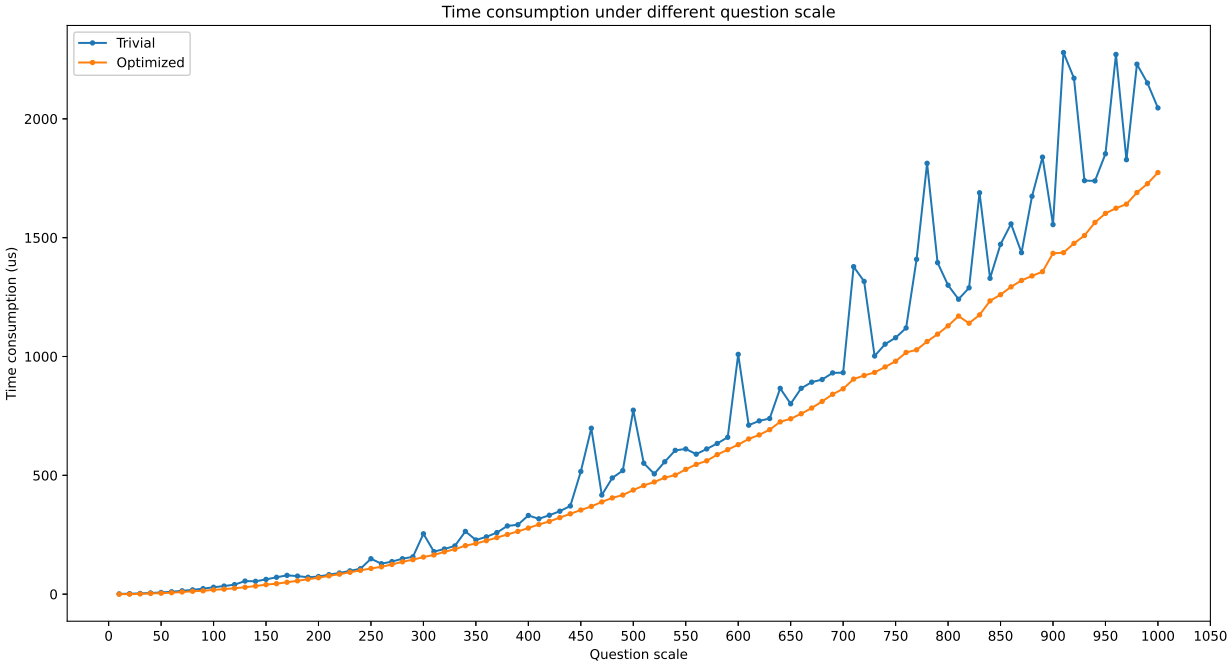

图 2: Product time consumption of x64

```
        result += a[i];
    }
    return result;
}
```

1.2、超标量优化算法

1、多链路算法，通过两个中间和变量，将循环次数减少一半。

```
int sum(int n, int* a)
{
    int sum_1 = 0;
    int sum_2 = 0;
    for (int i = 0; i < n; i += 2) {
        sum_1 += a[i];
        sum_2 += a[i + 1];
    }
    return sum_1 + sum_2;
}
```

图 3: Trivial 算法的 cache miss 火焰图

图 4: Optimized 算法的 cache miss 火焰图

2、递归算法，加数从两端开始两两相加，得到中间结果后重复，直到得到最终结果。

```
int sum(int n, int* a)
{
    if (n == 1) {
        return a[0];
    }
    else {
        for (int i = 0; i < n / 2; i++) {
            a[i] += a[n - 1 - i];
        }
        sum(n / 2, a);
    }
}
```

3、双循环算法，对于数列的前 n/2 项，有：$a_i = a_{i\times 2} + a_{i\times 2+2}$。这样每循环一次，数列的长度减小一半，同时避免了递归算法对栈空间的大量需求。

```
int sum(int n, int* a)
{
    for (int m = n; m > 1; m /= 2) {
        for (int i = 0; i < m / 2; i++) {
            a[i] = a[i * 2] + a[i * 2 + 1];
        }
    }
    return a[0];
}
```

三、编程实现

Arm 平台：WSL（2.4.13.0）Ubuntu（22.04.5 LTS）系统下使用 aarch64-linux-gnu-g++（(Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0）交叉编译，在 qemu-aarch64（6.2.0 (Debian 1:6.2+dfsg-2ubuntu6.25)）上执行。

x64 平台：WSL（2.4.13.0）Ubuntu（22.04.5 LTS）系统下使用 g++ (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0 编译执行。

1、性能测试

测试数据中，数列采用 Fibonacci 数列，即 $a_i = a_{i-1} + a_{i-2}, i > 2, a_1 = a_2 = 1$。问题规模迭代次数在 arm 平台上取 20，在 x64 平台上取 22（这是由于 arm 平台由 wsl+qemu 得到，性能低于 x64 平台），每迭代一次问题规模增加一倍，确保值为 2 的幂。由于单次求和运算在问题规模较小时执行速度很快，所以得到的运行数据均是执行 100 次的数据。性能测试时间使用 gettimeofday() 函数获取。在 arm 平台上的测试结果与在 x64 平台上的测试结果分别绘制为折线图 7和折线图 10。

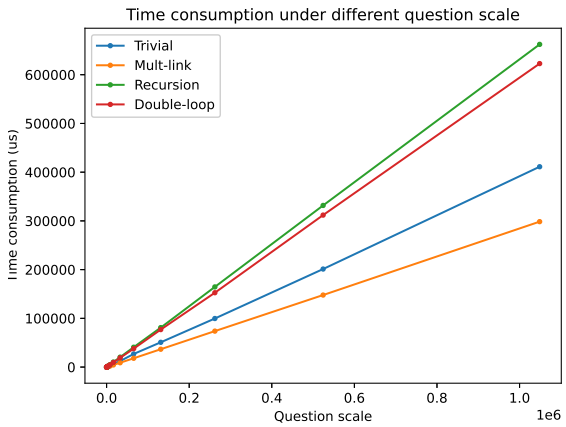

图 5: $x = n$

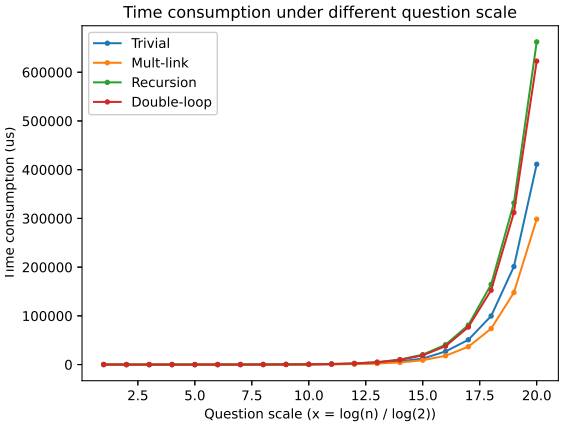

图 6: $x = \log_2(n)$

图 7: Sum time consumption of arm

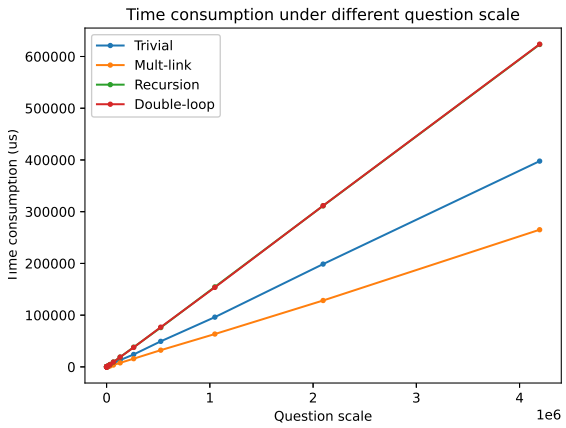

图 8: $x = n$

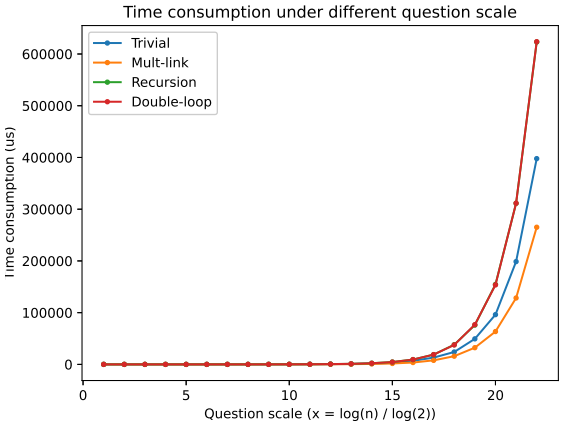

图 9: $x = \log_2(n)$

图 10: Sum time consumption of x64

Perf report of N-number summing				
Program	Trivial	Mult-link	Resursion	Double-loop
Instructions	9449982541	8191702016	24550353073	25389039452
Cycles	3401266727	2308548839	5238332656	5193907468

表 2: Pref 分析结果

四、profilling

x64 平台，WSL（2.4.13.0）Ubuntu（22.04.5 LTS）系统下使用 perf（5.15.178）对平凡算法和三种优化算法运行时的 Instructions 和 Cycles 进行分析，结果如表格 2所示。

五、结果分析

分析折线图 7与折线图 10，优化算法中只有多链路算法的性能优于平凡算法，从 perf 分析结果表 2可以看出，递归算法和双循环算法实际上对 iterations 和 cycles 的作用不减反增，带来了更大的性能开销。