南　开　大　学

计　算　机　学　院

并行程序设计实验报告

体系结构相关编程

赵元鸣 2211757

年级：2022 级

专业：计算机科学与技术

指导教师：王刚

2024 年 3 月 27 日

摘要

关键字：matrix,sum,cache

目录

一、 概述

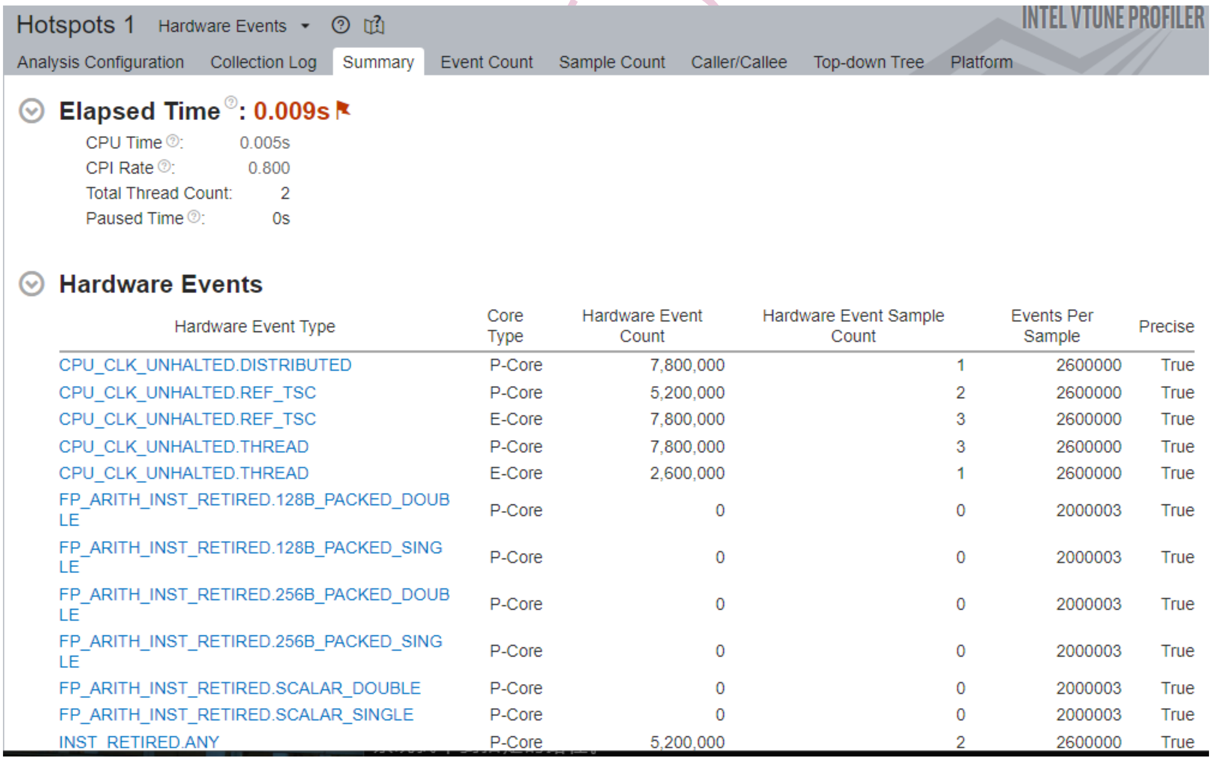
(一) 第一节：矩阵乘法 cache 优化性能对比

平凡算法代码

逐列访问平凡算法

```
void mul(int n, float a[][maxN], float b[][maxN], float c[][maxN]) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            c[i][j] = 0.0;
            for (int k = 0; k < n; ++k) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
}
```

VTune 测试结果 (本次实验 VTune 均采用 HotSpot 算法进行测试)



Hardware Event Type	Core Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample	Precise
CPU_CLK_UNHALTED.DISTRIBUTED	P-Core	7,800,000	1	2600000	True
CPU_CLK_UNHALTED.REF_TSC	P-Core	5,200,000	2	2600000	True
CPU_CLK_UNHALTED.REF_TSC	E-Core	7,800,000	3	2600000	True
CPU_CLK_UNHALTED.THREAD	P-Core	7,800,000	3	2600000	True
CPU_CLK_UNHALTED.THREAD	E-Core	2,600,000	1	2600000	True
FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_SINGLE	P-Core	0	0	2000003	True
INST_RETIRED.ANY	P-Core	5,200,000	2	2600000	True

图 1: VTune 测试

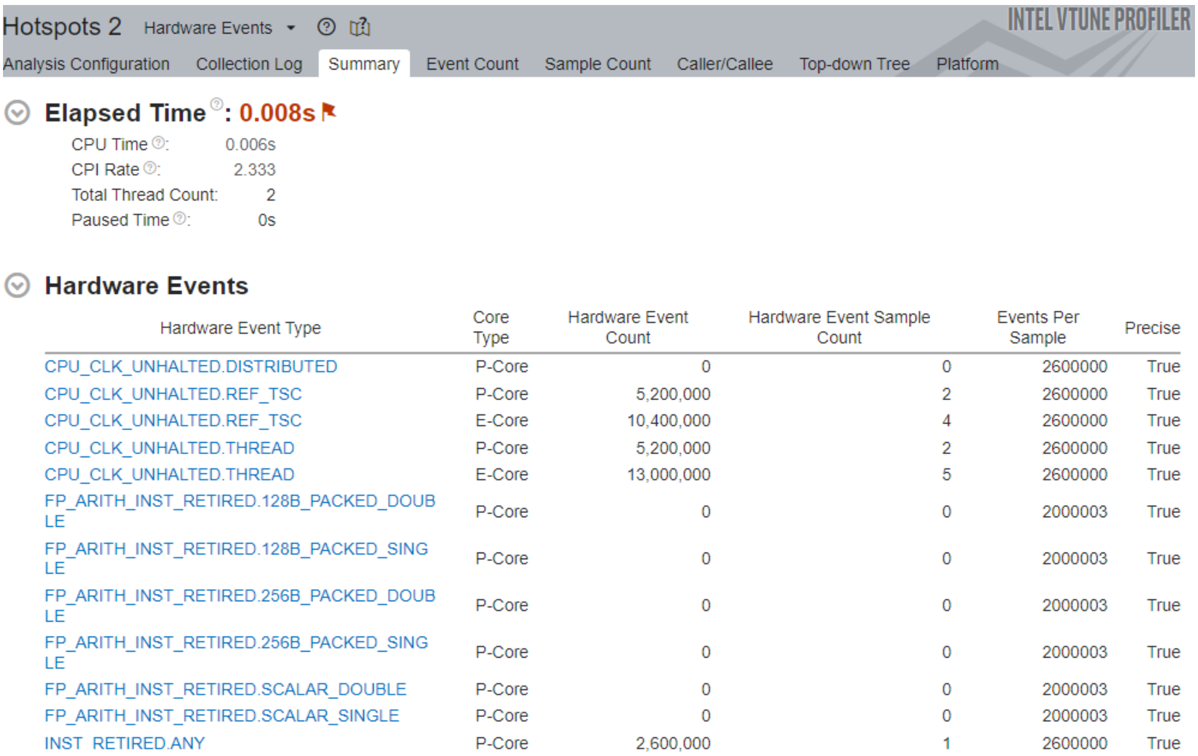
优化算法代码

优化算法算法

```
void trans_mul(int n, float a[][maxN], float b[][maxN], float c[][maxN])
    {
```

```
    for (int i = 0; i < n; ++i) for (int j = 0; j < i; ++j) swap(b[i][j], b[j
        ][i]);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            c[i][j] = 0.0;
            for (int k = 0; k < n; ++k) {
                c[i][j] += a[i][k] * b[j][k];
            }
        }
    }

    for (int i = 0; i < n; ++i) for (int j = 0; j < i; ++j) swap(b[i][j], b[j
        ][i]);
}
```

VTune 测试结果



Hotspots 2 Hardware Events

Analysis Configuration | Collection Log | Summary | Event Count | Sample Count | Caller/Callee | Top-down Tree | Platform

INTEL VTUNE PROFILER

Elapsed Time: 0.008s

CPU Time: 0.006s
CPI Rate: 2.333
Total Thread Count: 2
Paused Time: 0s

Hardware Events

Hardware Event Type	Core Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample	Precise
CPU_CLK_UNHALTED.DISTRIBUTED	P-Core	0	0	2600000	True
CPU_CLK_UNHALTED.REF_TSC	P-Core	5,200,000	2	2600000	True
CPU_CLK_UNHALTED.REF_TSC	E-Core	10,400,000	4	2600000	True
CPU_CLK_UNHALTED.THREAD	P-Core	5,200,000	2	2600000	True
CPU_CLK_UNHALTED.THREAD	E-Core	13,000,000	5	2600000	True
FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_SINGLE	P-Core	0	0	2000003	True
INST_RETIRED.ANY	P-Core	2,600,000	1	2600000	True

图 2: VTune 测试

主函数设计过程

主函数

```
 const int maxN = 1024; // 矩阵的最大值
const int T = 64; // tile size
int GapSize = 128;//设置间隙大小, 每一个矩阵规模自增GapSize
int SizeCounts = 10;//设置区间个数, 可以自由调整
int Counts = 50;//设置每次循环的次数
float a[maxN][maxN];
```

```
float b[maxN][maxN];
float c[maxN][maxN];

//用于矩阵改变数值,为防止数据溢出,随机数的区间为1~9
void change(int n, float a[][maxN], float b[][maxN]) {
    srand(time(0));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            a[i][j] = rand() % 10;
            b[i][j] = rand() % 10;
        }
    }
}

int main(int arg, char *argv[]) {
        //设置初始时间和结束时间
    struct timeval startTime, stopTime;
    //各个矩阵规模的时间数组,为了方便画echarts表格
    double mul_times[SizeCounts], trans_mul_times[SizeCounts], sse_mul_times[
        SizeCounts], sse_tile_times[SizeCounts];
    for (int temp = maxN - GapSize * (SizeCounts - 1); temp <= maxN; temp +=
        GapSize) {
        //设置每一个矩阵规模的总时间,每一个循环都加入到改变量中
        double mul_time = 0, trans_mul_time = 0, sse_mul_time = 0,
            sse_tile_time = 0;
        //循环Counts次
        for (int i = 0; i < Counts; ++i) {
                // 每一个均改变矩阵数值,控制唯一变量
            change(temp, a, b);
            gettimeofday(&startTime, NULL);
            mul(temp, a, b, c);
            gettimeofday(&stopTime, NULL);
            // 计算一个矩阵相乘的时间,并且加入的总时间中
            mul_time +=
                    (stopTime.tv_sec - startTime.tv_sec) * 1000 +
                    (double) (stopTime.tv_usec - startTime.tv_usec) * 0.001;

             gettimeofday(&startTime, NULL);
             trans_mul(temp, a, b, c);
             gettimeofday(&stopTime, NULL);
             trans_mul_time +=
                     (stopTime.tv_sec - startTime.tv_sec) * 1000 + (stopTime.
                         tv_usec - startTime.tv_usec) * 0.001;




        }
```

```
        // 将每一个总和都加入到相对应的数组中
        mul_times[GapSize - 1 - (maxN - temp) / GapSize] = mul_time / Counts;
        trans_mul_times[GapSize - 1 - (maxN - temp) / GapSize] =
            trans_mul_time / Counts;

    }
}
```

计时函数采用 sys/time.h 库中的 gettimeofday 函数，该函数的精确度可以打到微妙级别，所以对函数的运行时间可以很好的估量我还设置了多组循环实验，每一次循环将产生不同的矩阵，每一组矩阵乘法算法重复运行 Counts 次，然后得到平均估值，这样可以很好地减少一定量的误差 为了更好地对比四种算法的运行效率，我设计了一个 GapSize 值，用于矩阵规模每一次都自增 GapSize，直到达到最大值。另外，这里的区间个数自由度非常高，我们可以任意设置区间个数 SizeCounts，以便帮助我们更好地比较算法之间的差异

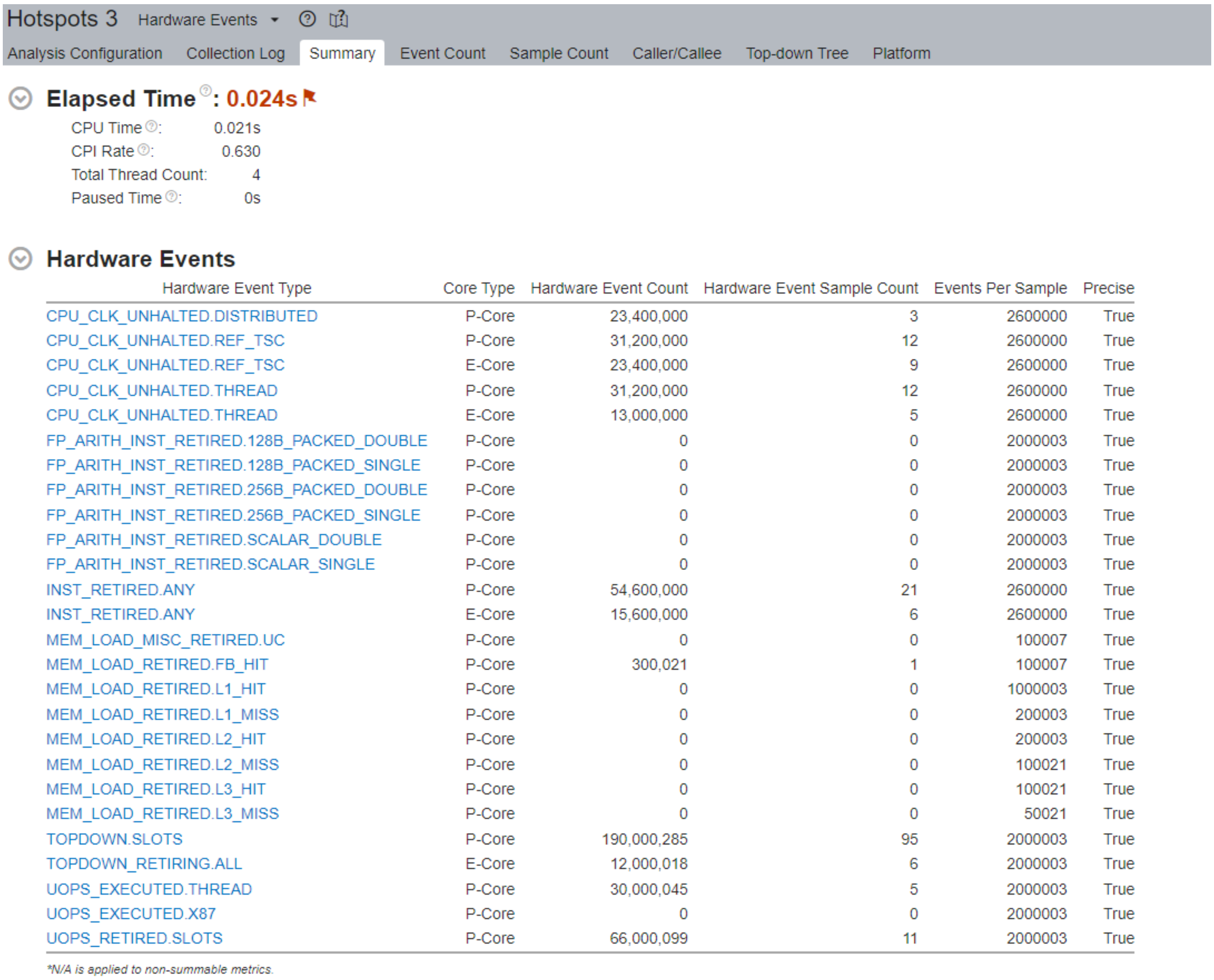
（二） 第二节：n 个数求和优化性能对比

平凡算法代码

加法平凡算法

```
 int main()
{
    int a[100],sum=0;
    for(int i=0;i<100;i++){
        a[i]=i+20;
    }
    for(int i=0;i<100000;i++){
            sum=0;
        for(int j=0;j<100;j++){
            sum+=a[j];
        }
    }
}
```

VTune 测试结果



Hotspots 3 Hardware Events

Analysis Configuration | Collection Log | Summary | Event Count | Sample Count | Caller/Callee | Top-down Tree | Platform

Elapsed Time: 0.024s

CPU Time: 0.021s
CPI Rate: 0.630
Total Thread Count: 4
Paused Time: 0s

Hardware Events

Hardware Event Type	Core Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample	Precise
CPU_CLK_UNHALTED.DISTRIBUTED	P-Core	23,400,000	3	2600000	True
CPU_CLK_UNHALTED.REF_TSC	P-Core	31,200,000	12	2600000	True
CPU_CLK_UNHALTED.REF_TSC	E-Core	23,400,000	9	2600000	True
CPU_CLK_UNHALTED.THREAD	P-Core	31,200,000	12	2600000	True
CPU_CLK_UNHALTED.THREAD	E-Core	13,000,000	5	2600000	True
FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_SINGLE	P-Core	0	0	2000003	True
INST_RETIRED.ANY	P-Core	54,600,000	21	2600000	True
INST_RETIRED.ANY	E-Core	15,600,000	6	2600000	True
MEM_LOAD_MISC_RETIRED.UC	P-Core	0	0	100007	True
MEM_LOAD_RETIRED.FB_HIT	P-Core	300,021	1	100007	True
MEM_LOAD_RETIRED.L1_HIT	P-Core	0	0	1000003	True
MEM_LOAD_RETIRED.L1_MISS	P-Core	0	0	200003	True
MEM_LOAD_RETIRED.L2_HIT	P-Core	0	0	200003	True
MEM_LOAD_RETIRED.L2_MISS	P-Core	0	0	100021	True
MEM_LOAD_RETIRED.L3_HIT	P-Core	0	0	100021	True
MEM_LOAD_RETIRED.L3_MISS	P-Core	0	0	50021	True
TOPDOWN.SLOTS	P-Core	190,000,285	95	2000003	True
TOPDOWN_RETIRING.ALL	E-Core	12,000,018	6	2000003	True
UOPS_EXECUTED.THREAD	P-Core	30,000,045	5	2000003	True
UOPS_EXECUTED.X87	P-Core	0	0	2000003	True
UOPS_RETIRED.SLOTS	P-Core	66,000,099	11	2000003	True

*N/A is applied to non-summable metrics.

图 3: VTune 测试

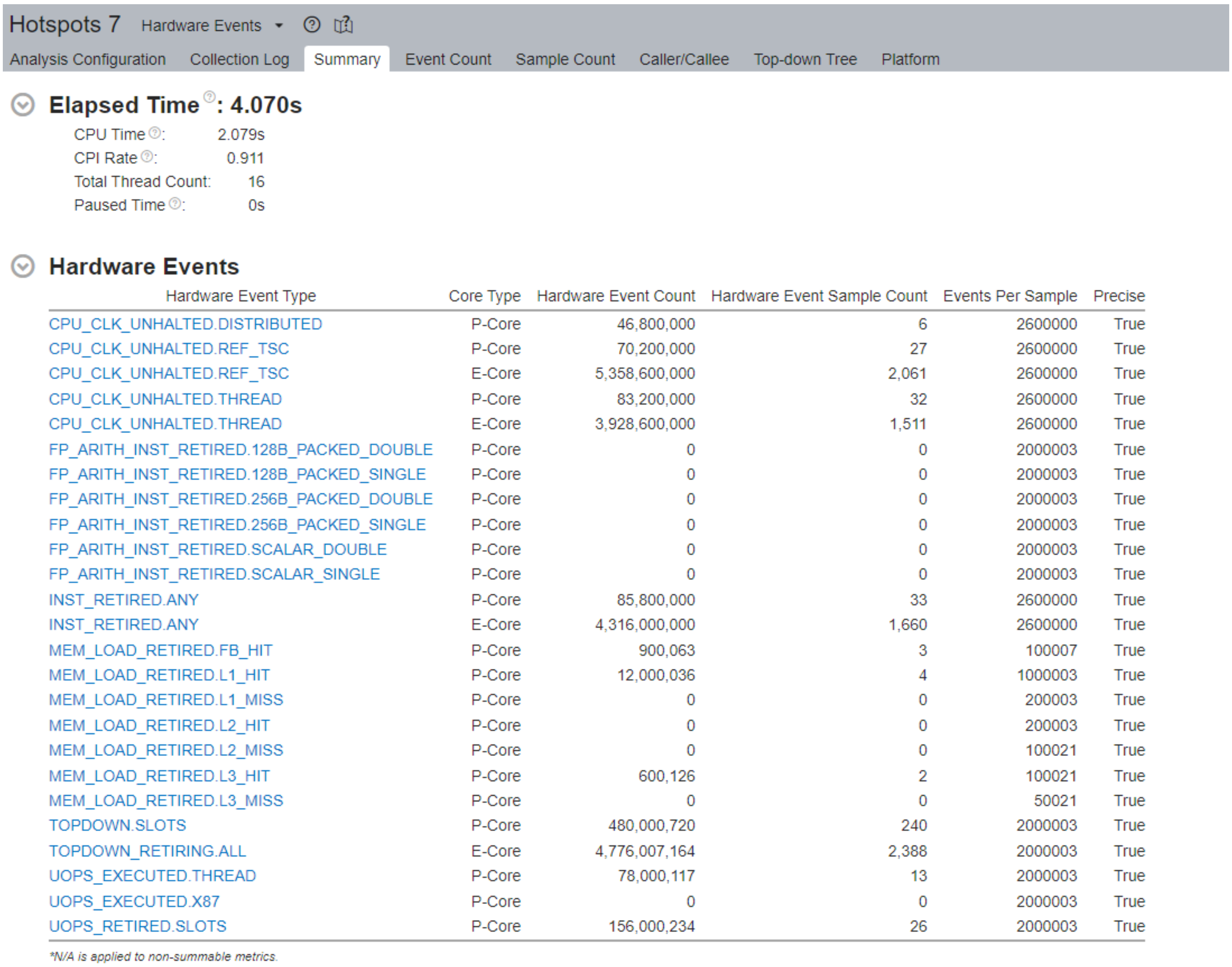
优化算法代码

加法优化算法

```


// 递归:
 1. 将给定元素两两相加, 得到n/2个中间结果;
 2. 将上一步得到的中间结果两两相加, 得到n/4个中间结果;
 3. 依此类推, log(n)个步骤后得到一个值即为最终结果。

 // 实现方式: 递归函数

int  recursion(int n,int*a)
 {
     if (n == 1)
        return a[0] ;
    else
```

```
            {
                  for  (int  i  =  0;  i  <  n  /  2;  i++){
                              a[i]+=a[n-i-1];
                        }
                   n  =  n  /  2;
                  recursion(n,a);
      }
 }

int main()
{
      int a[100],sum=0;
      for(int i=0;i<100;i++){
            a[i]=i+20;
      }

      for(int i=0;i<100000;i++)
            recursion(100,a);
}
```

VTune 测试结果



Hotspots 7 Hardware Events

Analysis Configuration Collection Log Summary Event Count Sample Count Caller/Callee Top-down Tree Platform

Elapsed Time: 4.070s

CPU Time: 2.079s
CPI Rate: 0.911
Total Thread Count: 16
Paused Time: 0s

Hardware Events

Hardware Event Type	Core Type	Hardware Event Count	Hardware Event Sample Count	Events Per Sample	Precise
CPU_CLK_UNHALTED.DISTRIBUTED	P-Core	46,800,000	6	2600000	True
CPU_CLK_UNHALTED.REF_TSC	P-Core	70,200,000	27	2600000	True
CPU_CLK_UNHALTED.REF_TSC	E-Core	5,358,600,000	2,061	2600000	True
CPU_CLK_UNHALTED.THREAD	P-Core	83,200,000	32	2600000	True
CPU_CLK_UNHALTED.THREAD	E-Core	3,928,600,000	1,511	2600000	True
FP_ARITH_INST_RETIRED.128B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.128B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.256B_PACKED_SINGLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_DOUBLE	P-Core	0	0	2000003	True
FP_ARITH_INST_RETIRED.SCALAR_SINGLE	P-Core	0	0	2000003	True
INST_RETIRED.ANY	P-Core	85,800,000	33	2600000	True
INST_RETIRED.ANY	E-Core	4,316,000,000	1,660	2600000	True
MEM_LOAD_RETIRED.FB_HIT	P-Core	900,063	3	100007	True
MEM_LOAD_RETIRED.L1_HIT	P-Core	12,000,036	4	1000003	True
MEM_LOAD_RETIRED.L1_MISS	P-Core	0	0	200003	True
MEM_LOAD_RETIRED.L2_HIT	P-Core	0	0	200003	True
MEM_LOAD_RETIRED.L2_MISS	P-Core	0	0	100021	True
MEM_LOAD_RETIRED.L3_HIT	P-Core	600,126	2	100021	True
MEM_LOAD_RETIRED.L3_MISS	P-Core	0	0	50021	True
TOPDOWN.SLOTS	P-Core	480,000,720	240	2000003	True
TOPDOWN_RETIRING.ALL	E-Core	4,776,007,164	2,388	2000003	True
UOPS_EXECUTED.THREAD	P-Core	78,000,117	13	2000003	True
UOPS_EXECUTED.X87	P-Core	0	0	2000003	True
UOPS_RETIRED.SLOTS	P-Core	156,000,234	26	2000003	True

*N/A is applied to non-summable metrics.

图 4: VTune 测试

从测试结果分析，递归式的优化算法反而会比平凡算法缓慢，经过更进一步测试，当问题规模足够大时，递归优化确实会比平凡算法性能更加优异，而在问题规模较小时，递归优化的结果在一些情况下可能不如平凡算法

本实验所有代码已经上传至 https://github.com/zhaoyuanmingzhendeshuai/ParalellLearning，请读者查阅