

# 《嵌入式软件优化设计》实验手册

（2020年5月版）

**郑重声明：**

**1、实验手册中的所有实验均有本人独立编码、调试和测试。**

**2、实验手册中给出的实验数据和结果完全由本人所完成的程序给出。**

**3、本人了解：不按照前两条要求所完成的实验报告已经构成了抄袭或造假行为，本人将承担相应的不良后果。**

姓名： 成子谦 （签名） 学号： 201730681303

实验得分表

|  |  |  |  |
| --- | --- | --- | --- |
| 实验 | 实验学时数 | 满分 | 得分 |
| 实验一 | 1 | 5 |  |
| 实验二 | 2 | 10 |  |
| 实验三 | 2 | 12 |  |
| 实验四 | 2 | 15 |  |
| 实验五 | 2 | 15 |  |
| 实验六 | 2 | 13 |  |
| 实验七 | 2 | 10 |  |
| 实验八 | 3 | 20 |  |
| 总分 | 16 | 100 |  |

提交日期： 总成绩：

指导教师： （签名）

## 实验一 熟悉TX1平台和开发环境

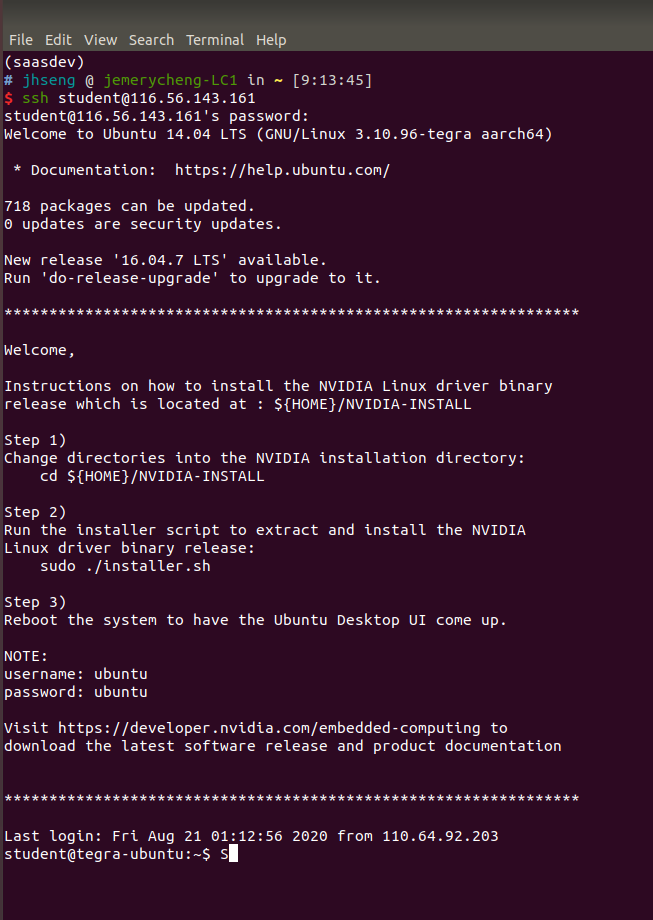
实验学时数：2 实验时间： 2020年8月21日 实验地点： B7-331 实验平台：TX1 得分：

【实验目的】1）熟悉TX1开发平台；2）确定TX1软硬件参数；3）搭建基本开发环境。

【实验内容】

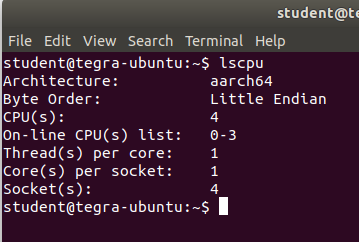
1. （1分）请简述接入TX1平台的方法，并给出登录后的界面截图。

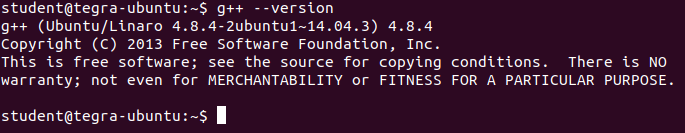
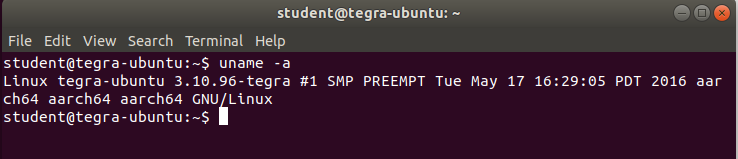
连接校园网后打开终端，执行`ssh [student@116.56.143.161](mailto:student@116.56.143.161)`并输入密码



2、（1分）请简述获取TX1平台硬件参数和软件参数的方法，并给出输出截图。

打开终端，执行`lscpu`， `uname -a`, `g++ --version`三条命令。





3、（2分）填写下述硬件/软件参数表

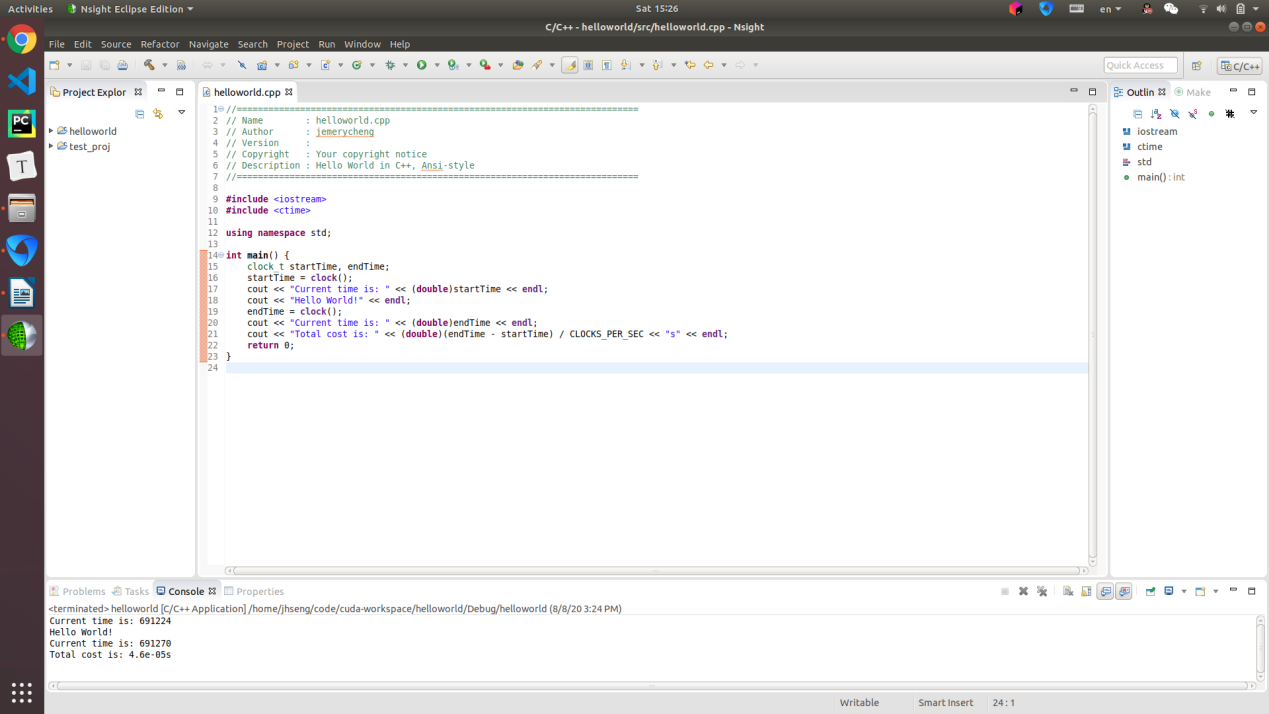
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CPU | | | Cache容量 | | | 内存容量 | 操作系统版本 | 编译器  版本 |
| 型号 | 核数 | 主频 | L1 | L2 | L3 |  |
| ARM Cortex-A57 | 4 | 1734MHz | 80KB | 512 KB to 2 MB | 无 | 4GB | 3.10.96 | 4.8.4 |

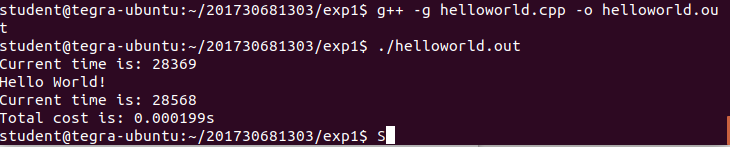
4、（1分）编写一个程序，可以打印“hello world”，并测量程序的运行时间（单位：ms）。

a)请给出编译命令

g++ -o “helloworld” ./src/helloworld.o (本地开发环境)

g++ -g helloworld.cpp -o helloworld.out (TX1)

b)请给出运行结果截图（显示“hello world”和运行时间）

c)该程序的运行时间为 0.199 ms。

## 实验二 消除分支指令优化实验

实验学时数：2 实验时间： 8月21日 实验地点： B7-331 实验平台：TX1 得分：

【实验目的】1）掌握分支指令的优化技巧；2）提升Huffman树编码和解码的实现效率。

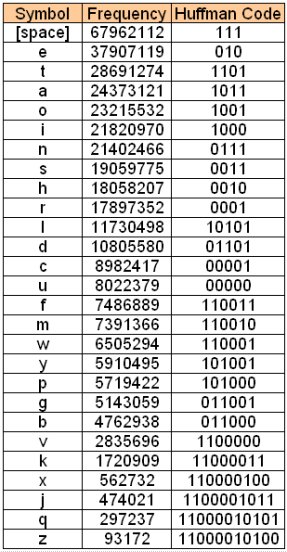
【实验要求】ASCII小写字母和空格的频度和Huffman编码如下表所示。设内存中的文本*M*仅仅包含空格和ASCII码小写字母。例如*M*中的内容为”fly”，则编码总计17位，从最低位到最高位的内容依次为：1101110101101001。请完成以下程序：

表2-1 字符频度和Huffman编码

1）Huffman编码程序，其中输入文本为*M*，长度为N个字节，输出的编码内容存放在事先分配好的内存code中，编码的总位数存储于code\_bits中。

void huffman\_encode(char \*M, int N, unsigned char \*code, int \*code\_bits);

2）Huffman解码程序，其中输入编码内容存储在code中，编码的总位数存储于code\_bits中，输出文本存放在事先分配好的内存M中，解码后的字符串长度存储在N中。

void huffman\_decode(unsigned char \*code, int code\_bits, char \*M, int \*N);

3）输入文本长度约500KB（文件名：Guns of the Gods.txt），并将第一步的编码结果作为第二步解码结果的输入。

【提示】编码过程中可以使用32位字作为基本操作单位，通过移位操作拼接多个字母的编码内容；也可以考虑将编码按照字节次序存储，先事先字节拼接，然后再按照32个字节一组合成一个字。在解码过程中，可以考虑使用2.3.3节中所述的消除分支方法。

【实验内容】

1、（2分）预处理输入文本，满足以下要求：a)所有大写字母转换为小写字母；b)所有非字母的内容转换为空格；c)统计空格和小写字母的出现频率。

处理后文本包括 511609 字节，请给出空格和小写字母的出现频率（按照从小到大排序）。

z 202 q 298 x 473 j 671 v 3233 k 3411 b 5910 p 6431 y 7256 f 8086

g 8724 c 8767 w 8940 m 10192 u 11759 l 15274 d 17179 r 22306 h 24727

s 24777 i 25764 n 26830 o 29214 a 31760 t 36277 e 47776 ` ` 125372

2、（4分）使用表2-1中的Huffman编码对处理后的文本进行编码。编码后的内容占用了 258822字节，压缩比是 49.41% ，编码所需的时间是 7.8835 ms。请简述Huffman编码的优化实现方法。

与传统的二叉哈弗曼树类似，不同的仅仅是每次取三个权值最小的 结点构造一个三叉树，把这三棵树的左中右结点的权值累加值作为新生成树的根结点的权值，并放入节结点序列中，重复操作，直到只含有一棵树为止。

3、（4分）使用表2-1中的Huffman编码对第2步中编码的结果进行解码。使用和不使用消除分支技术的解码时间分别为 441.3 ms和 282.6 ms。使用消除分支方法的加速比为 1.56 。请简述Huffman解码的优化实现方法。

一次性对码流的二位码元进行操作，将标准Huffman码表重建成四叉树结构。解码时，每经过四叉Huffman树的一个节点，相当于从码流中读取两个bit进行huffman码元匹配。理论上，对每个huffman码元解码的搜索深度和搜索时间最多可以减少为原来二叉树的1/2。

该方法来自于《基于四叉树的嵌入式平台Huffman解码优化》。

## 实验三 内存访问优化实验

实验学时数：2 实验时间：8月21日 实验地点：B7-331 实验平台：TX1 得分：

【实验目的】1）熟悉矩阵乘法分块的原理；2）掌握根据Cache容量估计分块的规模。

【实验要求】

*n*阶单精度浮点矩阵乘法：，并求矩阵C的迹Trace。

**程序输入输出要求**

Matrix\_mul *N seed*

参数：

*N:* 矩阵大小

*seed*：输入的种子浮点数

输出格式：

Trace

Trace: 结果矩阵C的迹（），用于验证矩阵乘法的正确性。

矩阵产生函数

/\*

Input: a, b are the N\*N float matrix, 0<seed<1, float

This function should initialize two matrixs with rand\_float()

\*/

float rand\_float(float s){

return 4\*s\*(1-s);

}

void matrix\_gen(float \*a,float \*b,int N,float seed){

float s=seed;

for(int i=0;i<N\*N;i++){

s=rand\_float(s);

a[i]=s;

s=rand\_float(s);

b[i]=s;

}

}

【实验内容】

1、根据前述程序产生*n×n*的单精度浮点随机矩阵A和B，并完成基准矩阵乘法程序。

1.1（1分）TX1的内存容量 3852 MB，可以容纳单精度矩阵乘法的最大阶数*N*= 8192 （*N* 为2的整数次幂）

1.2 （1分）调整*n*，统计不同矩阵规模下串行程序执行时间的变化。

表2-1 不同规模矩阵的串行计算时间——（3分）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 矩阵阶数*n* | 512 | 1024 | 2048 | *N* |
| seed=0.3时的trace值 | 63133.9 | 252536 | 1010301 |  |
| 计算时间(ms) | 1000 | 42000 | 666000 |  |

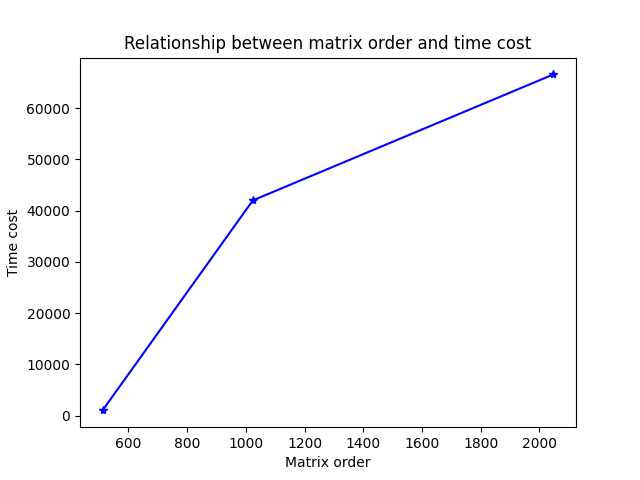
1.2（1分）根据表2-1绘制图2-1，其中X轴为*n*，Y轴为矩阵阶为*n*时的计算时间。

图2-1 不同矩阵规模的计算时间

（1分）*n*阶矩阵乘法的计算复杂度为 O(n3) 。

2、将矩阵分解为*m*×*m*的矩阵小块，每次完成一对小块的计算，以提高Cache的命中率。如下式所示。

，其中*n=N/m*

2.1（2分）TX1的L1和L2 Cache容量分别是 80 KB和 512 KB。*m*设置为 256 比较合适（记为*m*0），原因是 可以放到L2Cache中 。

2.2（4分）调整*m*的大小，得到不同分块大小下的计算时间和加速比。

表2-3 不同分块大小的计算时间和加速比（矩阵阶数为*Ｎ*）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 分块大小(*m*) | 不分块 | *m*0/4 | *m*0/2 | *m*0 | 2*m*0 | 4*m*0 |
| 计算时间(s) | 0.4058 | 0.5850 | 0.6006 | 0.6065 | 0.5989 | 0.5988 |
| 加速比 | 1.00 | 69.36% | 67.56% | 66.90% | 67.75% | 67.76% |

2.3（2分）实验表明，分块大小为 m0/4 时性能最好。这个结果和你的预期一致吗？ 一致 。

## 实验四 SIMD指令优化实验一

实验学时数：2 实验时间： 8月21日 实验地点： B7-331 实验平台：TX1

【实验目的】1）熟悉NEON指令系统；2）使用NEON指令实现*m×m*单精度浮点矩阵乘法（其中*m*是实验三的最优分块大小）。

【实验要求】使用NEON指令可以一次完成4个浮点乘法操作和加法操作。下图给出了4×4的矩阵乘法操作示意图。



1、（5分）请使用实验三中得到的最优*m*值（= 256 ）。请简述使用NEON指令实现*m×m*单精度浮点矩阵乘法的方法。

枚举i、j，初始化4个float32x4\_t类型变量为0。

2、（5分）原有串行程序实现*m×m*单精度浮点矩阵乘法的时间为 3479 ms，使用NEON指令后的执行时间为 3284 ms，加速比为 1.059 。

3、（3分）将NEON实现的*m×m*单精度浮点矩阵乘法模块并入实验二的程序中。请填写在不同矩阵规模的执行时间和加速比。

表2-1 不同规模矩阵的计算时间——（3分）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *N* | 512 | 1024 | … | *N* |
| 未使用NEON指令的计算时间(ms) | 3479 | 25888 |  |  |
| 使用NEON指令的计算时间(ms) | 3284 | 31347 |  |  |
| 加速比 | 1.059 | 0.825 |  |  |

4、（2分）如果需要将该程序移植到AVX、SSE等其他SIMD指令系统，你在软件设计上有何考虑。

需要注意指令格式。和MMX/SSE的不同点在于增强的AVX指令，从指令的格式上就发生了很大的变化。x86(IA-32/Intel 64)架构的基础上增加了prefix(Prefix)，所以实现了新的命令，也使更加复杂的指令得以实现，从而提升了x86 CPU的性能。

## 实验五 SIMD指令优化实验二

实验学时数：2 实验时间： 8月24日 实验地点： B7-331 实验平台：TX1 得分：

【实验目的】1）熟悉图像二维卷积计算原理；2）使用NEON指令实现图像的二维卷积。

【实验要求】

程序输入输出要求

2Dconv  *inputimage outputimage kernelfilename*

参数：

*inputimage:* 输入图像文件路径（BMP格式）

*outputimage*: 输出图像文件路径（BMP格式）

*kernelfilename*：卷积核文件 输入两幅512\*512 24位图像

卷积核

//模糊卷积

int Blurfilter[25] = { 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, };

// 运动模糊

int MotionBlurfilter[81] =

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 };

//边缘探测1

int edges1filter[25] = { -1, 0, 0, 0, 0, 0, -2, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0, -2, 0, 0, 0, 0, 0, -1, };

//边缘探测2

int edges2filter[9] = { -1, -1, -1, -1, 8, -1, -1, -1, -1 };

//锐化1

int sharpen1filter[9] = { -1, -1, -1, -1, 9, -1, -1, -1, -1 };

//锐化2

int sharpen2filter[25] = { -1, -1, -1, -1, -1, -1, 2, 2, 2, -1, -1, 2, 8, 2, -1, -1, 2, 2, 2, -1, -1, -1, -1, -1, -1, };

1、（2分）请查阅资料，并说明BMP文件格式。

BMP是windows操作系统中的标准图像文件格式，它采用位映射存储格式，除了图像深度可选之外，不采用其他任何压缩，所以BMP文件占用的空间很大。BMP格式的文件从头到尾依次是：BMP信息头、位图信息头、调色板以及位图数据。BMP信息头包含bfType、bfSize、bfReserved1、bfReserved2、bfOffBits等信息。位图信息头包括biSize、biWidth、biHeight、biPlanes、biBitCount、biCompression、biSizeImages、biXPelsPerMeter、biYPelsPerMeter、 biClrUsed、biClrImportant等信息。

2、完成程序读取BMP文件格式，并使用下述链接的程序完成图像二维卷积，

https://www.cppentry.com/bencandy.php?fid=49&id=145597

2.1（2分）统计卷积的计算时间。

不同卷积核的计算时间

|  |  |
| --- | --- |
| 核大小 | 计算时间(ms) |
| 9 | 42.682 |
| 25 | 103.107 |
| 81 | 310.522 |

2.2（1分）二维卷积的计算复杂度是 O(n4) 。

2.3（1分）请给出两幅图像卷积的效果图（卷积核=锐化2）。



3、（5分）请说明使用NEON指令系统实现图像灰度化、二维卷积的操作的实现方法。

通过vaddq\_s32、vmulq\_s32等运算指令实现rgb值计算，vst1q\_s32实现数据存入寄存器中。

4、（2分）使用NEON指令系统后性能对比。

|  |  |  |  |
| --- | --- | --- | --- |
| 核大小 | 未使用NEON的计算时间(ms) | 使用NEON的计算时间(ms) | 加速比 |
| 9 | 42.682 | 25.799 | 1.654 |
| 25 | 103.107 | 57.773 | 1.784 |
| 81 | 310.522 | 183.556 | 1.691 |

5、（2分）请给出NEON指令系统后两幅图像卷积的效果图（卷积核=锐化2）。



## 实验六 多线程优化实验一

实验学时数：2 实验时间： 8月24日 实验地点： B7-331 实验平台：TX1

【实验目的】1）多线程程序设计原理；2）使用多线程方法实现矩阵乘法。

【实验要求】在实验四程序的基础上，完成多线程的性能优化。

【实验内容】

1、（1分）TX1上有 4 个处理核，你认为使用 4 个线程的性能会达到最好（此线程数记为*T*）。

2、（3分）请简述多线程完成矩阵乘法的并行算法。使用pthread还是OpenMP进行并行化？并请说明理由。

Openmp。库函数相对完善，利用#pragma即可实现多线程编程。

3、（2分）在实验四程序的基础上，编写可以设定不同线程数*t*的并行矩阵乘法程序，并测量不同矩阵阶数*n*和*t*（*t=*1. *T/*2,*T*,2*T*）组合时的矩阵乘法时间。

不同规模矩阵和并行线程数的计算时间（单位: ms）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 并行线程数*t* | 矩阵阶数*n* | | | |
| 512 | 1024 | … | *N* |
| 1 | 3.568 | 29.5757 |  |  |
| *T/*2 | 3.587 | 28.6116 |  |  |
| *T* | 3.692 | 28.7463 |  |  |
| 2*T* | 3.586 | 28.7092 |  |  |

不同规模矩阵和并行线程数的加速比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 并行线程数*t* | 矩阵阶数*n* | | | |
| 512 | 1024 | … | *N* |
| 1 | 1.0 | 1.0 | 1.0 | 1.0 |
| *T/*2 | 0.995 | 1.033 |  |  |
| *T* | 0.967 | 1.028 |  |  |
| 2*T* | 0.995 | 1.030 |  |  |

4、（4分）请分析上述性能数据。性能最好的线程数是多少？是否和你估计的*T*相同？如果不相同，请说明原因。sharpen2filter

性能最好的线程数是T/2，不相同。实验平台表现很不稳定，难以测出精确值。

1. （3分）*N*阶矩阵乘法中总共需要 2n3 -n2 次浮点乘法和加法操作。请对比矩阵阶为*N*时三种优化方法下矩阵乘法的计算时间和单位时间内完成的单精度浮点性能。哪种方法对提升性能效果最明显？

多线程对提升性能效果最明显。

不同实现方法的提比较（矩阵阶为*N*= 512 ）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实现方法 | 串行实现 | +矩阵分片 | +使用SIMD | +多线程 |
| 计算时间(ms) | 1031 | 1104 | 982 | 907 |
| 加速比 | 1.0 | 0.933 | 1.049 | 1.136 |
| 单位时间内完成的浮点计算（GLOPS） | 260109.9 | 242910.6 | 273088.9 | 295670.6 |

## 实验七 多线程优化实验二

实验学时数：2 实验时间： 8月24日 实验地点： B7-331 实验平台：TX1

【实验目的】1）多线程程序设计原理；2）使用多线程方法实现图像二维卷积。

【实验要求】在实验五程序的基础上，完成多线程的性能优化。

【实验内容】

1、（1分）TX1上有 4 个处理核，你认为使用 4 个线程的性能会达到最好（此线程数记为*T*）。

2、（2分）请简述多线程完成二维图像卷积乘法的并行算法。使用pthread还是OpenMP进行并行化？并请说明理由。

Openmp。库函数相对完善，利用#pragma即可实现多线程编程。

3、（2分）在实验五程序的基础上，编写可以设定不同线程数*t*的二维图像卷积程序，并测量不同卷积核*k*和*t*（*t=*1. *T/*2,*T*,2*T*）组合时的二维图像卷积计算时间和加速比。

不同规模卷积核和并行线程数的计算时间（单位: ms）

|  |  |  |  |
| --- | --- | --- | --- |
| 并行线程数*t* | 卷积核大小*k* | | |
| 9 | 25 | 81 |
| 1 | 276.441 | 659.243 | 1999.33 |
| *T/*2 | 276.44 | 661.172 | 2008.56 |
| *T* | 276.406 | 660.673 | 1995.81 |
| 2*T* | 276.282 | 659.404 | 2010.42 |

不同规模卷积核和并行线程数的加速比

|  |  |  |  |
| --- | --- | --- | --- |
| 并行线程数*t* | 卷积核大小*k* | | |
| 9 | 25 | 81 |
| 1 | 1.0 | 1.0 | 1.0 |
| *T/*2 | 1.00 | 0.997 | 0.995 |
| *T* | 1.00 | 0.997 | 1.001 |
| 2*T* | 1.0005 | 0.999 | 0.994 |

4、（4分）请分析上述性能数据。性能最好的线程数是多少？是否和你估计的*T*相同？如果不相同，请说明原因。

相同。4线程时虽然表现不是很明显，但依然效果最好。

5、（1分）请给出多线程加速后两幅图像卷积的效果图（卷积核=锐化2）。



## 实验八 综合实验

实验学时数：3 实验时间： 8月24日 实验地点： B7-331 实验平台： TX1

【实验目的】1）学习Java的JNI使用方法；2）形成完整的应用软件。

【实验要求】在实验七程序的基础上，使用Java语言设计一个图形界面，实现一个较为完整的二维图像卷积应用软件。该软件的图形界面如下图所示。



二维卷积图像应用软件的主图形界面

该软件至少有五个按钮：

Open：打开原始图像文件（至少支持BMP格式），并在“原始图像显示区域”显示原始图像文件；

Kernel：选择对应的卷积核（至少包含实验五中的所有卷积核）；

Run：开始二维卷积计算，计算结束后在“卷积结果图像显示”区域显示结果图像；

Save：将当前“卷积结果图像显示”区域的图像存储为指定图像文件（至少支持BMP格式）；

Exit：退出本程序。

【实验内容】

1、（3分）使用Java语言实现二维图像卷积计算，并测量不同规模卷积核的二维图像卷积时间，并与实验七中的计算时间对比。

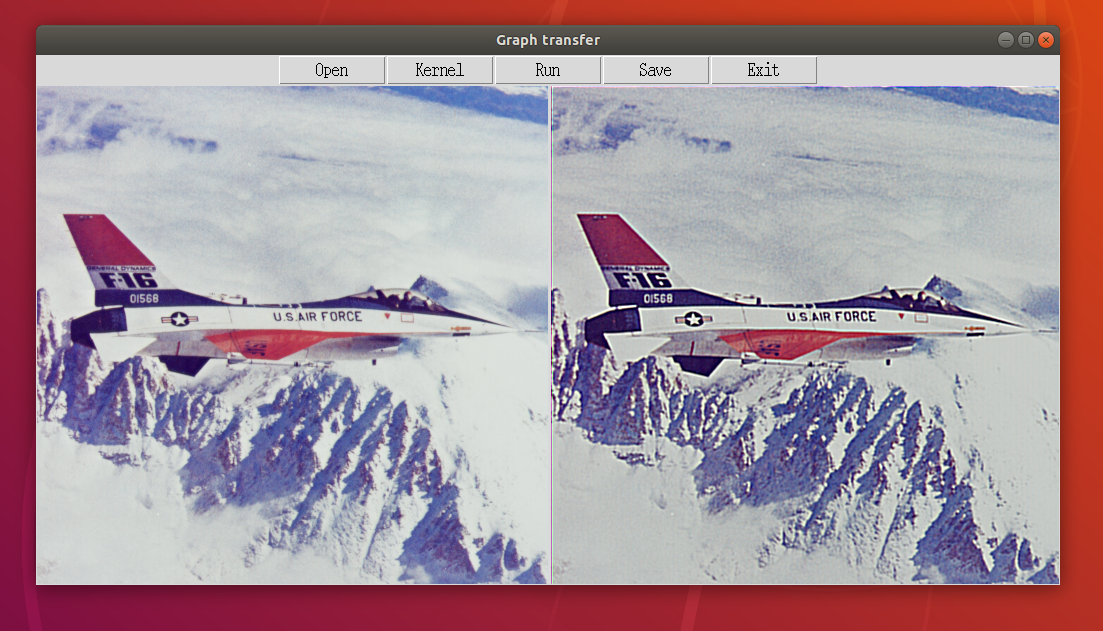
不同卷积核的计算时间

|  |  |  |
| --- | --- | --- |
| 核大小 | Java程序的计算时间(ms) | 实验七中的计算时间(ms) |
| 9 | 407.11 | 276.441 |
| 25 | 1539.2 | 659.243 |
| 81 | 5047.71 | 1999.33 |

2、通过JNI接口集成实验七程序，在TX1/Linux或x86/Linux平台上形成完整的应用。

2.1（2分）说明使用的硬件和软件平台。

Ubuntu 18.04, python3.6

2.2（2分）请展示主图形界面

2.3（3分）你完成软件的功能较【实验要求】中的基本功能有何提升？（例如增加新的图像文件格式，新增一个功能点得0.5分，最多不超过3分）

优化了多线程的运行速度。优化了图片渲染速度。

3、（4分）说明JNI接口的实现方法。

4、请将软件移植到x86/Windows平台。

4.1 （2分）提供程序运行截图

4.2 （2分）说明从NEON指令系统移植到AVX（SSE）指令系统的方法

Redefines ARM NEON 64 and 128-bit vectors as the corresponding x86 SIMD data.

Redefines some functions from ARM NEON to Intel SSE if 1:1 correspondence exists (~50% of 128 bit functions)

Implements some ARM NEON functions using Intel SIMD if the performance effective implementation is possible (~45% of functions)

Implements the remaining NEON functions (for <5% of functions not widely used in applications) using the serial solution and issuing the corresponding "low performance" compiler warning

来源于intel官方文档，<https://software.intel.com/content/www/us/en/develop/articles/from-arm-neon-to-intel-sse-the-automatic-porting-solution-tips-and-tricks.html>

4.3 （2分）说明多线程移植的方法。