分布式系统作业

“多线程并行编程”

第1次作业

姓名：唐晨轩

班级：人工智能与大数据

学号：19335182

1. 问题描述

在第一次课程中已经讲到，早期单节点计算系统并行的粒度分为:Bit级并 行，指 令级并行和线程级并行。现代处理器如Intel、ARM、AMD、Power以及国 产CPU如 华为鲲鹏等，均包含了并行指令集合。1. 请调查这些处理器中的并行（向量）指令集， 并选择其中一种如AVX, SSE等进行编程练习。 此外，现代操作系统为了发挥多核的优 势，支持多线程并行编程模型，请利用多线程的方式实现N个整数的求和，编程语言 不限，可以是Java，也可以是 C/C++

1. 解决方案
2. 请调查这些处理器中的并行（向量）指令集， 并选择其中一种如AVX, SSE等进行编程练习。

Intel支持的各个指令集：

Intel的[x86](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=474226&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)，EM64T，MMX，SSE，SSE2，SSE3，SSSE3 (Super SSE3)，[SSE4A](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=10996845&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)，[SSE4.1](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=6151986&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)，[SSE4.2](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=64100169&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)，AVX，AVX2，AVX-512，VMX等指令集

ARM的指令列表：

|  |  |
| --- | --- |
| ADC | 带进位的32位数加法 |
| ADD | 32位数相加 |
| AND | 32位数的逻辑与 |
| B | 在32M空间内的相对跳转指令 |
| BEQ | 相等则跳转（Branch if EQual） |
| BNE | 不相等则跳转（Branch if Not Equal） |
| BGE | 大于或等于跳转（Branch if Greater than or Equa） |
| BGT | 大于跳转（Branch if Greater Than） |
| BIC | 32位数的逻辑位清零 |
| BKPT | 断点指令 |
| BL | 带链接的相对跳转指令 |
| BLE | 小于或等于跳转（Branch if Less than or Equal） |
| BLEQ | 带链接等于跳转（Branch with Link if EQual） |
| BLLT | 带链接小于跳转（Branch with Link if Less Than） |
| BLT | 小于跳转（Branch if Less Than） |
| BLX | 带链接的切换跳转 |
| BX | 切换跳转 |
| CDP CDP2 | 协处理器数据处理操作 |
| CLZ | 零计数 |
| CMN | 比较两个数的[相反数](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=7495377&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank) |
| CMP | 32位数比较 |
| EOR | 32位逻辑异或 |
| LDC LDC2 | 从协处理器取一个或多个32位值 |
| LDM | 从内存送多个32位字到ARM寄存器 |
| LDR | 从虚拟地址取一个单个的32位值 |
| MCR MCR2 MCRR | 从寄存器送数据到协处理器 |
| MLA | 32位乘累加 |
| MOV | 传送一个32位数到寄存器 |
| MRC MRC2 MRRC | 从协处理器传送数据到寄存器 |
| MRS | 把状态寄存器的值送到[通用寄存器](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=3683713&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank) |
| MSR | 把通用寄存器的值传送到状态寄存器 |
| MUL | 32位乘 |
| MVN | 把一个32位数的逻辑“非”送到寄存器 |
| ORR | 32位逻辑或 |
| PLD | 预装载提示指令 |
| QADD | 有符号32位饱和加 |
| QDADD | 有符号双32位饱和加 |
| QSUB | 有符号32位饱和减 |
| QDSUB | 有符号双32位饱和减 |
| RSB | 逆向32位减法 |
| RSC | 带进位的逆向32法减法 |
| SBC | 带进位的32位减法 |
| SMLAxy | 有符号乘累加(16位\*16位)+32位=32位 |
| SMLAL | 64位有符号乘累加((32位\*32位)+64位=64位) |
| SMALxy | 64位有符号乘累加((32位\*32位)+64位=64位) |
| SMLAWy | 号乘累加((32位\*16位)>>16位)+32位=32位 |
| SMULL | 64位有符号乘累加(32位\*32位)=64位 |
| SMULxy | 有符号乘(16位\*16位=32位) |
| SMULWy | 有符号乘(32位\*16位>>16位=32位) |
| STC STC2 | 从协处理器中把一个或多个32位值存到内存 |
| STM | 把多个32位的寄存器值存放到内存 |
| STR | 把寄存器的值存到一个内存的虚地址内间 |
| SUB | 32位减法 |
| SWI | 软中断 |
| SWP | 把一个字或者一个字节和一个寄存器值交换 |
| TEQ | 等值测试 |
| TST | 位测试 |
| UMLAL | 64位无符号乘累加((32位\*32位)+64位=64位) |
| UMULL | 64位无符号乘累加(32位\*32位)=64位 |

AMD的指令集：

AMD的x86，[x86-64](https://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=617198&ss_c=ssc.citiao.link" \t "https://baike.sogou.com/_blank)，3D-Now!指令集。

Power的指令集：

x86-64。

华为鲲鹏的指令集：

兼容ARM指令集。

这里我选用了AVX指令集来编程练习。

AVX 指令集提供了一系列的 floating-point（浮点）处理指令和 integer（整数）处理指令：

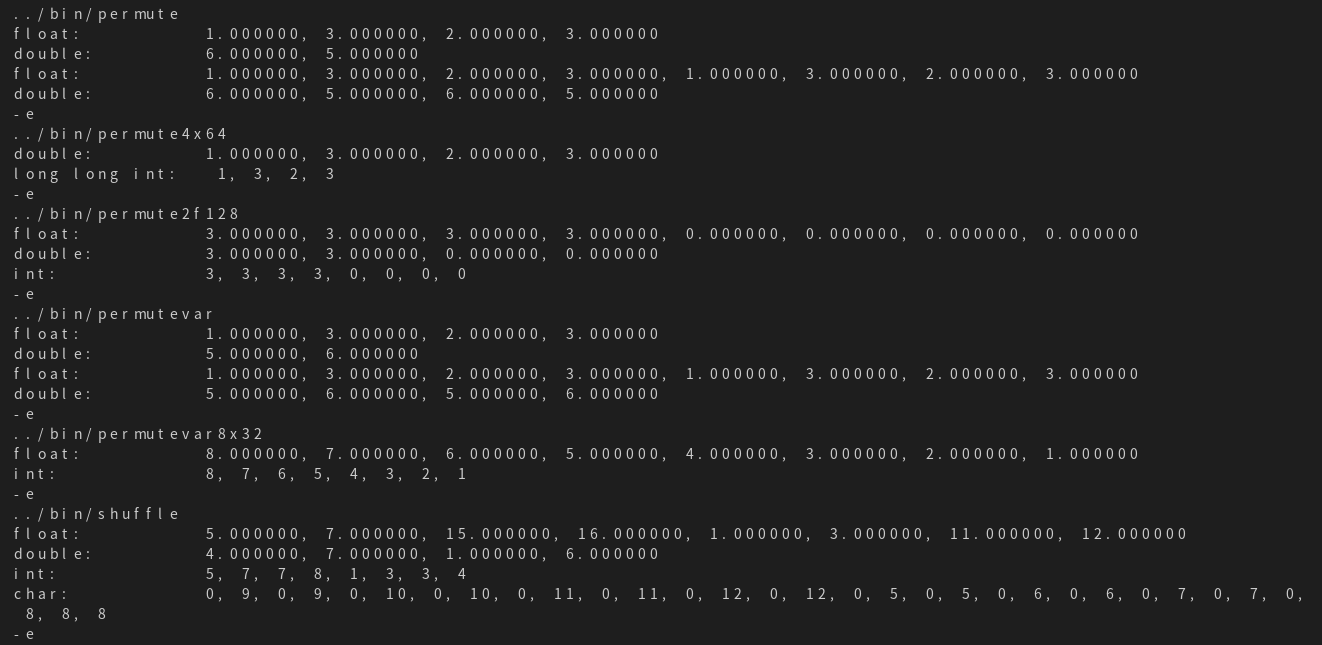
1. 提供的浮点处理指令，包括：
   * 两种宽度的 ****arithmetic****（运算）类指令：
     + 256 位宽度的 vector（矢量）运算
     + 128 位宽度的 vector 与 scalar（标量）运算
   * 两种宽度的 ****non-arithmetic****（非运算）类指令：
     + 256 位 non-arithmetic 类 vector 操作指令
     + 128 位 non-arithmetic 类 vector 与 scalar 操作指令
   * AVX 新增的 non-arithmetic 类指令：这些指令是 AVX 独有的新引进的指令，不存在对应的 SSE 版本
2. 整数处理指令包括：
   * 128 位的 packed integer 运算指令
   * 128 位的 packed integer 处理指令

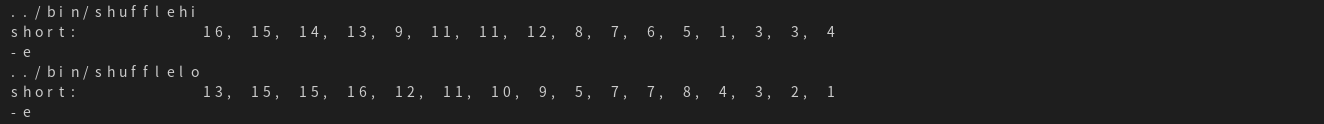
除了新增的 AVX 指令外，其余的指令都是从原有的 SSE 指令移值或扩展而来，这些 AVX 指令有对应的 SSE 版本。

2. 请利用多线程的方式实现N个整数的求和，编程语言 不限，可以是Java，也可以是 C/C++。

1. 实验结果

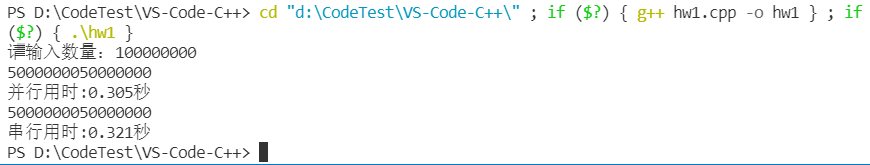
1.VAX



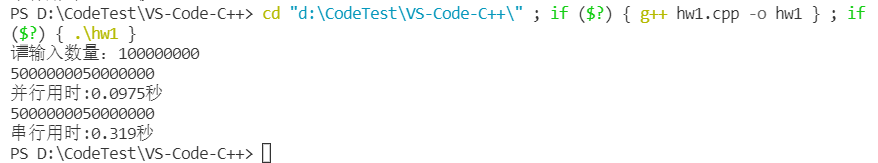


2.N个整数求和

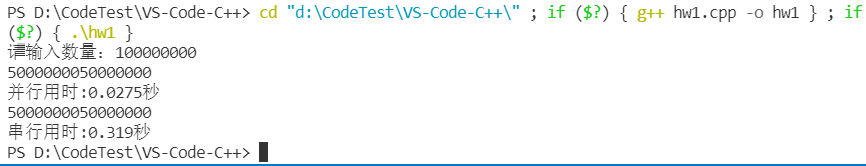
当CPU核心数量为1时：



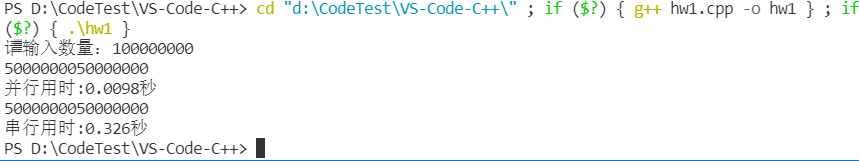
当CPU核心数量为2时：



当CPU核心数量为4时：



当CPU核心数量为10时：



可以看到当CPU的核心数量为1时，串行计算与并行计算的耗时基本一致。

当时当CPU的核心数量增多，并行计算的耗时会越来越短。

1. Write an essay describing a research problem in your major that would benefit from the use of parallel computing. Provide a rough outline of how parallelism would be used. Would you use task- or data-parallelism?

以本次作业为例子：

本次作业中N整数相加这个任务就很好的体现了并行式计算的优点，即计算快速，可以充分运用CPU的多个核心。在我之后的使用中，如果能用上并行编程可以让我的程序更加快速，更好的利用计算机资源。我更倾向于使用任务并行，因为任务并行可以把一个任务模块化，细分成几个部分，每个部分完成一个部分，可以更有条理。

1. 遇到的问题及解决方法

由于对并行式编程的实际操作并不熟悉，所以上课学到的理论知识有些难以实现，比如最开始不知道如何实现并行式编程，后来通过复习PPT看到了基于<windows.h>头文件的并行式编程实例，然后加以改变，才完成任务。根据查询资料，还知道了实现并行式编程有多种办法，比如之前操作系统用过的互斥锁，还有基于<pthread.h>头文件的并行式编程等等。