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

一、 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++

二、 解决方案

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。

Intel 支持的各个指令集：

Intel 的 x86，EM64T，MMX，SSE，SSE2，SSE3，SSSE3 (Super SSE3)，SSE4A，SSE4.1，SSE4.2，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
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
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，x86-64，3D-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++。

三、 实验结果

1.VAX

```
../bin/permute
float:          1.000000, 3.000000, 2.000000, 3.000000
double:         6.000000, 5.000000
float:          1.000000, 3.000000, 2.000000, 3.000000, 1.000000, 3.000000, 2.000000, 3.000000
double:         6.000000, 5.000000, 6.000000, 5.000000
-e
../bin/permute4x64
double:         1.000000, 3.000000, 2.000000, 3.000000
long long int:   1, 3, 2, 3
-e
../bin/permute2f128
float:          3.000000, 3.000000, 3.000000, 3.000000, 0.000000, 0.000000, 0.000000, 0.000000
double:         3.000000, 3.000000, 0.000000, 0.000000
int:            3, 3, 3, 3, 0, 0, 0, 0
-e
../bin/permutevar
float:          1.000000, 3.000000, 2.000000, 3.000000
double:         5.000000, 6.000000
float:          1.000000, 3.000000, 2.000000, 3.000000, 1.000000, 3.000000, 2.000000, 3.000000
double:         5.000000, 6.000000, 5.000000, 6.000000
-e
../bin/permutevar8x32
float:          8.000000, 7.000000, 6.000000, 5.000000, 4.000000, 3.000000, 2.000000, 1.000000
int:            8, 7, 6, 5, 4, 3, 2, 1
-e
../bin/shuffle
float:          5.000000, 7.000000, 15.000000, 16.000000, 1.000000, 3.000000, 11.000000, 12.000000
double:         4.000000, 7.000000, 1.000000, 6.000000
int:            5, 7, 7, 8, 1, 3, 3, 4
char:           0, 9, 0, 9, 0, 10, 0, 10, 0, 11, 0, 11, 0, 12, 0, 12, 0, 5, 0, 5, 0, 6, 0, 6, 0, 7, 0, 7, 0,
 8, 8, 8
-e
```

```
../bin/shufflehi
short:          16, 15, 14, 13, 9, 11, 11, 12, 8, 7, 6, 5, 1, 3, 3, 4
-e
../bin/shufflelo
short:          13, 15, 15, 16, 12, 11, 10, 9, 5, 7, 7, 8, 4, 3, 2, 1
-e
```

2.N 个整数求和

当 CPU 核心数量为 1 时：

```
PS D:\CodeTest\VS-Code-C++> cd "d:\CodeTest\VS-Code-C++\" ; if ($?) { g++ hw1.cpp -o hw1 } ; if
($?) { .\hw1 }
请输入数量：100000000
5000000050000000
并行用时:0.305秒
5000000050000000
串行用时:0.321秒
PS D:\CodeTest\VS-Code-C++>
```

当 CPU 核心数量为 2 时：

```
PS D:\CodeTest\VS-Code-C++> cd "d:\CodeTest\VS-Code-C++\" ; if ($?) { g++ hw1.cpp -o hw1 } ; if
($?) { .\hw1 }
请输入数量：100000000
5000000050000000
并行用时:0.0975秒
5000000050000000
串行用时:0.319秒
PS D:\CodeTest\VS-Code-C++>
```

当 CPU 核心数量为 4 时：

```
PS D:\CodeTest\VS-Code-C++> cd "d:\CodeTest\VS-Code-C++\" ; if ($?) { g++ hw1.cpp -o hw1 } ; if
($?) { .\hw1 }
请输入数量：100000000
5000000050000000
并行用时:0.0275秒
5000000050000000
串行用时:0.319秒
PS D:\CodeTest\VS-Code-C++>
```

当 CPU 核心数量为 10 时：

```
PS D:\CodeTest\VS-Code-C++> cd "d:\CodeTest\VS-Code-C++\" ; if ($?) { g++ hw1.cpp -o hw1 } ; if
($?) { .\hw1 }
请输入数量：100000000
5000000050000000
并行用时:0.0098秒
5000000050000000
串行用时:0.326秒
PS D:\CodeTest\VS-Code-C++>
```

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。

3. 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。

四、 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。