|  |  |  |
| --- | --- | --- |
| Кафедра ЭВМ и С  Лабораторная работа №5  **Изучение организации современных процессоров**  Вариант 5 | Студент | Сидский Н. А. |
| Группа | ИВТ-261 |
| Дата |  |
| Преподаватель |  |
| Дата отчета |  |
| Балл |  |
| Подпись преподавателя |  |

Цели работы:

1. Получить представление об организации и производительности современных вычислительных систем;

2. Познакомиться с программированием систем с общей памятью на примере ПЭВМ с многоядерными процессорами;

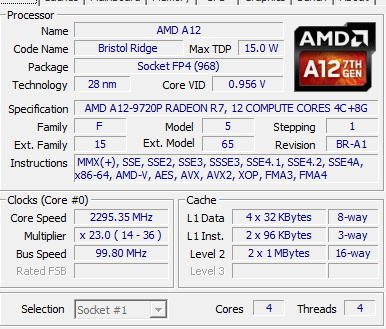
3. Познакомиться с программированием векторных процессоров на примере векторных команд SSE;

4. Познакомиться с некоторыми возможностями современных оптимизирующих компиляторов;

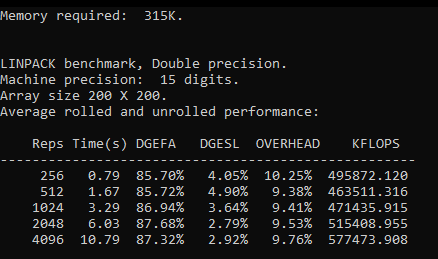
5. Получить навыки разработки программного обеспечения, эффективно выполняющегося на вычислительных системах современной архитектуры.

1) Проверка производительности процессора в linpack.

Характеристики моего процессора:



Результат теста:



Максимальная производительность процессора в тесте на 4096 элементов и составила 577,5 GFlops.

2) Определение времени работы и производительность программы при разных подходах.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N | TSER, сек | , сек |  | TSSE, сек | PSSE | , сек |  |
| 1024 | 0.009 | 0.009 | 111,111 | 0.005 | 800 | 0,002 | 2000 |
| 4096 | 0.174 | 0.128 | 7,813 | 0.06 | 66,667 | 0,011 | 363,637 |
| 10000 | 0.802 | 0.798 | 1,253 | 0.312 | 12,821 | 0,103 | 38,835 |
| 14992 | 1.809 | 1.708 | 0,585 | 0.743 | 5,384 | 0,208 | 19,231 |
| 20000 | 3.173 | 3.06 | 0,327 | 1,285 | 3,113 | 0,59 | 6,78 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | , сек | , сек |  | , сек |  |
| 1024 | 0.009 | 0,005 | 204 800 | 0,001 | 4 096 000 |
| 4096 | 0.128 | 0,097 | 42 226,804 | 0,012 | 1 365 333,333 |
| 10000 | 0.798 | 0,54 | 18 518,519 | 0,07 | 571 428,571 |
| 14992 | 1.708 | 1,235 | 12 139,271 | 0.191 | 313 968,586 |
| 20000 | 3.06 | 2.328 | 8 591,065 | 0.299 | 267 558,528 |

3) Написать программу для вычисления по варианту

Вариант 5: Сложение двух матриц

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N | TSER, сек | , сек |  | TSSE, сек | PSSE | , сек |  | , сек |  | , сек |  |
| 1024 | 0.008 | 0,009 | 111,111 | 0,005 | 800 | 0,004 | 1000 | 0,008 | 128000 | 0,005 | 819200 |
| 4096 | 0.239 | 0,149 | 6,711 | 0,09 | 44,44444 | 0,071 | 56,33803 | 0,118 | 34711,86 | 0,069 | 237449,3 |
| 10000 | 1,385 | 0,874 | 1,1441648 | 0,489 | 8,179959 | 0,384 | 10,41667 | 0,632 | 15822,78 | 0,365 | 109589 |
| 14992 | 2,35 | 2,193 | 0,4559964 | 1,096 | 3,649635 | 0,904 | 4,424779 | 1,798 | 8338,154 | 0,831 | 72163,66 |
| 20000 | 27.683 | 7,673 | 0,1303271 | 6,522 | 0,613309 | 3,085 | 1,296596 | 4,714 | 4242,681 | 1,681 | 47590,72 |

Графики производительности от времени:

Оптимизированное последовательное выполнение

Выполнение с инструкцией SSE

Оптимизированное выполнение с инструкцией SSE

Оптимизированное параллельное выполнение

Оптимизированное параллельное выполнение с использованием SSE

Вывод: Производительность программы повышается, если использовать параллельные вычисления или/и инструкции процессора или / и оптимизацию компилятора. Если использовать все три подхода, то производительность вырастает в разы.

Код программы:

#include "pch.h"

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <emmintrin.h>

#include <math.h>

#include <xmmintrin.h>

#include <mmintrin.h>

#include <chrono>

#include <omp.h>

// Function for parallel matrix-vector multiplication

void SerialResultCalculation(float\* pMatrix, float\* pVector, float\* pResult, int Size) {

int i, j; // Loop variables

for (i = 0; i < Size; i++) {

for (j = 0; j < Size; j++)

pResult[i \* Size + j] += pMatrix[i \* Size + j] + pVector[i \* Size + j];

}

}

// Function for simple definition of matrix and vector elements

void DummyDataInitialization(float\* pMatrix, float\* pVector, int Size) {

int i, j; // Loop variables

for (i = 0; i < Size; i++) {

for (j = 0; j < Size; j++) {

pMatrix[i \* Size + j] = i;

pVector[i \* Size + j] = j;

}

}

}

// Function for memory allocation and definition of object�s elements

void ProcessInitialization(float\*& pMatrix, float\*& pVector, float\*& pResult, int& Size) {

// Size of initial matrix and vector definition

do {

printf("\nEnter size of the initial objects: ");

scanf("%d", &Size);

printf("\nChosen objects size = %d\n", Size);

if (Size <= 0)

printf("\nSize of objects must be greater than 0!\n");

} while (Size <= 0);

// Memory allocation

pMatrix = new float[Size \* Size];

pVector = new float[Size \* Size];

pResult = new float[Size \* Size];

// Definition of matrix and vector elements

DummyDataInitialization(pMatrix, pVector, Size);

}

// Function for parallel matrix-vector multiplication

void ParallelResultCalculation(float\* pMatrix, float\* pVector, float\* pResult, int Size) {

int i, j; // Loop variables

#pragma omp parallel for private (j)

for (i = 0; i < Size; i++) {

for (j = 0; j < Size; j++)

pResult[i \* Size + j] = pMatrix[i \* Size + j] + pVector[i \* Size + j];

}

}

void ParallelResultCalculationSSE(float\* pMatrix, float\* pVector, float\* pResult, int Size) {

\_\_m128 row0;

\_\_m128 row1;

\_\_m128 row2;

\_\_m128 res4;

float\* res\_4;

float res = 0;

res\_4 = (float\*)\_aligned\_malloc(Size \* sizeof(float), 16);

int i, j, k; // Loop variables

#pragma omp parallel for private (j)

for (i = 0; i < Size; i++)

{

res4 = \_mm\_setzero\_ps();

for (j = 0; j < Size; j = j + 4)

{

row1 = \_mm\_load\_ps(&pVector[Size \* (i)+j]);//загрузка вектора

row0 = \_mm\_load\_ps(&pMatrix[Size \* (i)+j]);///загрузка строки матрицы

res4 = \_mm\_add\_ps(row0, row1);//сложение строк матриц

\_mm\_store\_ps(&pResult[Size \* (i)+j], res4);

}

}

}

void SerialResultCalculationSSE(float\* pMatrix, float\* pVector, float\* pResult, int Size)

{

\_\_m128 row0;

\_\_m128 row1;

\_\_m128 row2;

\_\_m128 res4;

float\* res\_4;

float res = 0;

res\_4 = (float\*)\_aligned\_malloc(Size \* sizeof(float), 16);

int i, j, k; // Loop variables

for (i = 0; i < Size; i++)

{

res4 = \_mm\_setzero\_ps();

for (j = 0; j < Size; j = j + 4)

{

row1 = \_mm\_load\_ps(&pVector[Size \* (i) + j]);//загрузка вектора

row0 = \_mm\_load\_ps(&pMatrix[Size \* (i) + j]);///загрузка строки матрицы

res4 = \_mm\_add\_ps(row0, row1);//сложение строк матриц

\_mm\_store\_ps(&pResult[Size \* (i)+j], res4);

}

}

}

// Function for computational process termination

void ProcessTermination(float\* pMatrix, float\* pVector, float\* pResult) {

delete[] pMatrix;

delete[] pVector;

delete[] pResult;

}

void main() {

float\* pMatrix; // The first argument - initial matrix

float\* pVector; // The second argument - initial vector

float\* pResult; // Result vector for matrix-vector multiplication

int Size; // Sizes of initial matrix and vector

double dtime;

struct timespec time1, time2;

printf("Parallel OpenMP matrix-vector multiplication \n");

ProcessInitialization(pMatrix, pVector, pResult, Size);

// serial

auto start = std::chrono::system\_clock::now();

SerialResultCalculation(pMatrix, pVector, pResult, Size);

auto end = std::chrono::system\_clock::now();

dtime = (double)std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count() / 1000;

ProcessTermination(pMatrix, pVector, pResult);

printf("\n Time of execution serial: %f\n", dtime);

// sse

ProcessInitialization(pMatrix, pVector, pResult, Size);

start = std::chrono::system\_clock::now();

SerialResultCalculationSSE(pMatrix, pVector, pResult, Size);

end = std::chrono::system\_clock::now();

dtime = (double)std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count() / 1000;

ProcessTermination(pMatrix, pVector, pResult);

printf("\n Time of execution sse: %f\n", dtime);

// parrallel

ProcessInitialization(pMatrix, pVector, pResult, Size);

start = std::chrono::system\_clock::now();

ParallelResultCalculation(pMatrix, pVector, pResult, Size);

end = std::chrono::system\_clock::now();

dtime = (double)std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count() / 1000;

ProcessTermination(pMatrix, pVector, pResult);

printf("\n Time of execution parallel: %f\n", dtime);

// parallel sse

ProcessInitialization(pMatrix, pVector, pResult, Size);

start = std::chrono::system\_clock::now();

ParallelResultCalculationSSE(pMatrix, pVector, pResult, Size);

end = std::chrono::system\_clock::now();

dtime = (double)std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count() / 1000;

ProcessTermination(pMatrix, pVector, pResult);

printf("\n Time of execution parallel sse: %f\n", dtime);

}