БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра информатики

Факультет НиДО

Специальность ИиТП

Контрольная работа № 1

по дисциплине «Архитектура вычислительных систем»

Выполнил студент: Дегтярев А.А.

группа 393551

Зачетная книжка № 902021-26

Минск 2017

**Программирование многоядерных архитектур**

**Цель работы**. Использование интерфейса OpenMP для программирования простых многопоточных приложений.

**Вариант 2: Умножение матрицы на вектор**

Работа выполнялась под macOs. Компилятор gcc-6, с поддержкой openmp;

Для компиляции необходимо указать флаг: **–fopenmp**

Программа читает пользовательский ввод( задает размеры матрицы и вектора ), заполняет ячейки случайными значениями и выполняет параллельное умножение ячеек.

Пример компиляции и исполнения:



Также можно задать дополнительный аргумент **–debug,** чтобы программа выводила информацию о созданных потоках, как представлено ниже.



Я запустил выполнение умножения матрицы 10000х10000 на вектор 10000.

Ниже представлен скриншот монитора cpu. Который отображает загрузку.  




Исходный код доступен в папке kr1/kr1openmp.c

|  |
| --- |
| //  // kr1openmp.c  // AVS  //  // Created by Alexander Degtyarev on 10/31/16.  // Copyright © 2016 Alexander Degtyarev. All rights reserved.  //  #include <stdio.h>  #include <time.h>  #include <stdlib.h>  #include <omp.h>  #include <stdbool.h>  int main(int argc, const char \* argv[]) {  bool debug\_info = argc > 1;  const float TOP\_CELL\_VAL = 10.0;  unsigned int rows = 0;  unsigned int columns = 0;  //synchronized data input  printf("Set row count\n");  scanf("%u",&rows);  printf("Set collumn count\n");  scanf("%u",&columns);  //allocating resources  float \*vector = malloc (rows \* sizeof(float));  float \*result = malloc (columns \* sizeof(float));  float \*\*matrix;  matrix = (float \*\*) malloc(rows\*sizeof(float \*));  for(int i = 0; i < rows; i++){  matrix[i]=(float \*) malloc(columns \* sizeof(float));  }  //parallel randomize matrix  #pragma omp parallel shared(rows,columns,matrix,vector)  {  srand((unsigned int)time(NULL) ^ omp\_get\_thread\_num());  #pragma omp for  for(int i = 0; i < rows; ++i)  {  result[i] = 0.0;  for (int j = 0; j < columns; ++j)  {  if(i==0){  vector[j] = ((float)rand()/(float)(RAND\_MAX)) \* TOP\_CELL\_VAL;  }  matrix[i][j] = ((float)rand()/(float)(RAND\_MAX)) \* TOP\_CELL\_VAL;  if(debug\_info) printf(" %d.%d Thread: %d:%d\n",i,j,omp\_get\_thread\_num(), omp\_get\_num\_threads());  }  }  }  //synchronized print  printf("\n -MATRIX-\n");  for(int i = 0; i < rows; ++i)  {  for (int j = 0; j < columns; ++j)  {  printf(" %3.2f",matrix[i][j]);  }  printf("\n");  }  printf("\n -VECTOR-\n");  for (int j = 0; j < columns; ++j)  {  printf(" %3.2f\n",vector[j]);  }  printf("\n -----------------\n");  //parallel multiplication  #pragma omp parallel for shared(rows,columns,matrix,vector,result)  for(int i = 0; i < rows; ++i)  {  for (int j = 0; j < columns; ++j)  {  result[i]+=matrix[i][j]\*vector[j];  if(debug\_info) printf("Thread: %d:%d\n",omp\_get\_thread\_num(), omp\_get\_num\_threads());  }  }  //synchronized output  printf("\n -RESULT-\n");  for(int i = 0; i < rows; ++i)  {  printf(" %2.1f \n",result[i]);  }  return 0;  } |