[Лабораторная работа №](file:///C:\\Users\\v.shimansky\\Downloads\\AK_html2%20%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D1%85%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC\\%D0%9F%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0\\content\\ak2\\lab1_h.htm) 3

**ПРОГРАММИРОВАНИЕ МНОГОЯДЕРНЫХ АРХИТЕКТУР**

***3.1 ЦЕЛЬ РАБОТЫ***

Использование интерфейса OpenMP для программирования простых многопоточных приложений.

**3.2 ИНТЕРФЕЙС OPENMP**

            OpenMP – интерфейс прикладного программирования (API) для масштабируемых SMP-систем (симметричные мультипроцессорные системы) в модели общей памяти.

Исполняемый процесс в памяти может состоять из множественных нитей, которые имеют общее адресное пространство, но разные потоки команд и раздельные стэки. В простейшем случае, процесс состоит из одной нити, выполняющую функцию main. Нити иногда называют также потоками, легковесными процессами, LWP (light-weight processes). OpenMP основан на существовании множественных потоков в общедоступной памяти [3]. Схема процесса представлена на рисунке 1.

Рис. 1

Все программы OpenMP начинаются как единственный процесс с главным   потоком. Главный поток выполняется последовательно, пока не сталкиваются с первой областью параллельной конструкции. Создание нескольких потоков (FORK) и объединение (JOIN) проиллюстрировано на рисунке 2.

Рис. 2

**3.3 ПРИМЕРЫ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ OPENMP**

***3.3.1 Определение и печать номера потока***

#include <omp.h>

#include <stdio.h>

void main ()

{

       int nthreads, tid;

       /\* Fork a team of threads giving them their own copies of variables \*/

       #pragma omp parallel private(tid)

       {

             /\* Obtain and print thread id \*/

             tid = omp\_get\_thread\_num();

             printf("Hello World from thread = %d\n", tid);

             /\* Only master thread does this \*/

             if (tid == 0)

             {

                    nthreads = omp\_get\_num\_threads();

                    printf("Number of threads = %d\n", nthreads);

             }

       }  /\* All threads join master thread and terminate \*/

}

***3.3.2 Распределение работы***

#include <stdio.h>

#include <omp.h>

#define CHUNKSIZE 100

#define N     1000

void main ()

{

       int i, chunk;

       float a[N], b[N], c[N];

       /\* Some initializations \*/

       for (i=0; i < N; i++)

             a[i] = b[i] = i \* 1.0;

       chunk = CHUNKSIZE;

       #pragma omp parallel shared(a,b,c,chunk) private(i)

       {

              #pragma omp for schedule(dynamic,chunk) nowait

             for (i=0; i < N; i++)

                    c[i] = a[i] + b[i];

       }  /\* end of parallel section \*/

}

***3.3.3 Использование секций***

#include <stdio.h>

#include <omp.h>

#define N 1000

void main ()

{

       int i;

       float a[N], b[N], c[N], d[N];

       /\* Some initializations \*/

       for (i=0; i < N; i++)

       {

             a[i] = i \* 1.5;

             b[i] = i + 22.35;

       }

       #pragma omp parallel shared(a,b,c,d) private(i)

       {

             #pragma omp sections nowait

             {

                    #pragma omp section

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

                           c[i] = a[i] + b[i];

                    #pragma omp section

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

                           d[i] = a[i] \* b[i];

             }  /\* end of sections \*/

       }  /\* end of parallel section \*/

}

***3.3.4. Параллельная реализация одиночных циклов***

#include <stdio.h>

#include <omp.h>

#define N   1000

#define CHUNKSIZE   100

void main ()

{

       int i, chunk;

       float a[N], b[N], c[N];

       /\* Some initializations \*/

       for (i=0; i < N; i++)

             a[i] = b[i] = i \* 1.0;

       chunk = CHUNKSIZE;

       #pragma omp parallel for shared(a,b,c,chunk) private(i) schedule(static,chunk)

       for (i=0; i < n; i++)

             c[i] = a[i] + b[i];

}

***3.3.5 Критические секции***

#include <omp.h>

void main()

{

       int x;

       x = 0;

       #pragma omp parallel shared(x)

       {

             #pragma omp critical

             x = x + 1;

       }  /\* end of parallel section \*/

}

***3.3.6. Редуцируемые операции***

#include <omp.h>

#include <stdio.h>

void main ()

{

       int   i, n, chunk;

       float a[100], b[100], result;

       /\* Some initializations \*/

       n = 100;

       chunk = 10;

       result = 0.0;

       for (i=0; i < n; i++)

       {

             a[i] = i \* 1.0;

             b[i] = i \* 2.0;

       }

#pragma omp parallel for default(shared) private(i) schedule(static,chunk)  \ reduction(+:result)

       for (i=0; i < n; i++)

             result = result + (a[i] \* b[i]);

       printf("Final result= %f\n",result);

}

***3.4 Лабораторное задание***

1.     В соответствии с вариантом задания реализовать алгоритм с использованием интерфейса OpenMP.

2.     Защита лабораторной работы.

***Варианты***

1.     Скалярное произведение двух векторов.

2.     Умножение матрицы на вектор.

3.     Умножение матрицы на матрицу.

4.     Решение системы линейных алгебраических уравнений методом Гаусса.

***3.5 Литература***

1.     Спецификация инструкции cpuid для процессоров Intel <http://www.intel.com/Assets/PDF/appnote/241618.pdf>

2.     Спецификация инструкции cpuid для процессоров AMD <http://support.amd.com/us/Embedded_TechDocs/25481.pdf>

3.     Корнеев В.Д. Параллельное программирование кластеров // Новосибирск. НГТУ. 2008. – 312 с.