Введение в OpenMP (практика)

История и обзор

- OpenMP – одно из наиболее популярных средств программирования для компьютеров с общей памятью, базирующихся на традиционных языках программирования и использовании специальных комментариев.
- OpenMP – один вариант программы для параллельного и последовательного выполнения.
- Разработчиком стандарта является некоммерческая организация OpenMP ARB (Architecture Review Board), в которую входят представители крупных компаний-разработчиков суперкомпьютеров и программного обеспечения.
- OpenMP поддерживает работу с языками Фортран и Си/Си++.

Концепция прагм

- Прагма (из документации Microsoft) – «Директивы #pragma предоставляют каждому компилятору способ обеспечения специальных компьютерных функций и функций операционной системы при сохранении общей совместимости с языками Си и Си++».
- Основной особенностью прагм (директив) является то, что если компилятор не распознает данную прагму, то он должен ее игнорировать (в соответствие со стандартами ANSI Си и Си++).
- В языке Си директивы OpenMP оформляются указаниями препроцессору, начинающимиися с #pragma omp. Ключевое слов omp используется для того, чтобы исключить случайные совпадения имент директив OpenMP с другими именами в программе.

Концепция прагм (продолжение)

- Формат директивы на Си/Си++:

  #pragma omp directive-name [опция, …]

- Все директивы OpenMP можно разделить на 3 категории: определение параллельной области, распределение работы, синхронизация.

Параллельные и последовательные области

- В начале работы программы существует одна «основная» нить. Последовательные участки программы выполняет только основная нить и никакая другая. При входе в параллельную часть программы создаются новые «рабочие» нити, которые уничтожаются при выходе из параллельной части программы.
- Параллельная часть программы начинается с директивы

  #pragma omp parallel [опция, …]
- В OpenMP переменные в параллельных частях программы разделяются на два вида:

- shared (общие)
- private (локальные)

Компиляция и запуск

Компиляция с ключом «-fopenmp»:

gcc file_name.c -fopenmp (по умолчанию исполняемы файл имеет имя «a.out»)

Запуск обычный:

./a.out

В платформах UNIX версия с открытым кодом доступна в проекте компилятора Omni OpenMP (http://www.hpcs.cs.tsukuba.ac.jp/omni-compiler/)

Первая программа, часть 1

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>        //   заголовочный файл OpenMP

int main(int argc, char* argv[]){
#ifdef   _OPENMP
     printf("OpenMP is supported! %d \n", _OPENMP);
#endif
      int a[10];
      int i = 0;
      int myid, num_procs, num_threads;
```

Первая программа, часть 2

```
num_procs = omp_get_num_procs();     //   получение количества доступных вычислительных ядер
printf("Num of processors = %d \n", num_procs);

omp_set_num_threads(2);     //  явное задание количества нитей

num_threads = omp_get_num_threads();     //   получение количества заданных нитей
printf("Number of threads  = %d \n", num_threads);

for (i = 0; i < 10; i++){
 a[i] = i;
}
```

Первая программа, часть 3

```
    myid = omp_get_thread_num();   //   получение номера нити
    printf("Consecutive part, myid = %d\n", myid);

#pragma omp parallel shared(a)  private(myid)     //   начало параллельной части программы
      {
      myid = omp_get_thread_num();
      printf("Parallel part, myid = %d\n", myid);
      //   !!! здесь место для “#pragma omp for“
       }    //   конец параллельной части программы
      myid = omp_get_thread_num();
      printf("Consecutive part, myid = %d\n", myid);
} //   конец функции main
```

Первая программа, часть 4

Если в параллельной программе встретится цикл, то все его итерации выполнятся всеми нитями. Для распределения итераций цикла между нитями можно использовать директиву “for”. Эта директива относится к следующему непосредственно за ней оператору “for”.

```
#pragma omp for
    for (i = 0; i < 10; i++){
        a[i] = a[i] * 2;
        printf("Thread %d has multiply element %d\n",      myid,
i);
    }
```

Первая программа, часть 5

reduction(оператор:список переменных) – опция как директивы “parallel”, так и директивы “for”. Задает оператор и список общих переменных. Для каждой из общих переменных создаются локальные копии в каждой из нитей. Над локальными копиями после исполнения всех действий внутри параллельной части или оператора цикла производится операция, указанная как “оператор”. Оператором могут быть следующие действия: +, -, *, ^, &, |, &&, ||, max, min.

Пример:

```
int b = 0;
#pragma omp parallel reduction (+:b)
        {
       b = b + 1;
       printf(“Current value, b= %d\n", b);
        } //   при выходе из параллельной части произойдет суммирование всех локальных копий из всех нитей
        printf(“Number of threads = %d\n", b);
```

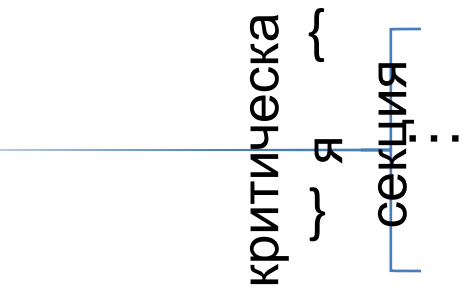
Синхронизация и критическая секция

Оформление барьера:

```
#pragma  omp  barrier
```

Оформление критической секции:

```
#pragma  omp  critical
```



Определение времени работы параллельной программы

double omp_get_wtime(void) – возвращает астрономическое время в секундах (вещественное число), произошедшее с некоторого момента в прошлом. Разность возвращаемых значений покажет время работы данного участка. (Таймеры разных нитей могут быть несинхронизированы и выдавать разные значения).

Пример:

```
double begin, end, total;
begin = omp_get_wtime();
….
end = omp_get_wtime();
total = end – begin;
```