Контрольные вопросы

1. В чем состоят основы технологии OpenMP?

Основой технологии OpenMP является такой подход, при котором указания программиста по организации параллельных вычислений добавляются в программу при помощи тех или иных внеязыковых средств языка программирования - в виде, которые обрабатываются специальным препроцессором до начала компиляции программы. При этом исходный текст программы остается неизменным, и по нему, в случае отсутствия препроцессора, компилятор построит исходный последовательный программный код. Препроцессор же, будучи примененным, заменяет директивы параллелизма на некоторый дополнительный программный код

2. В чем состоят основные преимущества и недостатки технологии OpenMP?

A. Технология OpenMP позволяет эффективно использовать ресурсы многопроцессорных ЭВМ

B. Сложность разработки параллельной программы с использованием технологии OpenMP в значительной степени согласуется со сложностью решаемой задачи

C. Технология OpenMP обеспечивает возможность поэтапной (инкрементной) разработки параллельных программы – директивы OpenMP могут добавляться в последовательную программу постепенно (поэтапно), позволяя уже на ранних этапах разработки получать параллельные программы, готовые к применению;

D. OpenMP позволяет в значительной степени снизить остроту проблемы переносимости параллельных программ между разными компьютерными системами

E. OpenMP рассчитан на мультипроцессоры и DSM-системы (системы с распределенной памятью, на которых смоделирована общая память) и изначально не ориентирован на кластеры;

F. на DSM-системах большого размера (64 и выше),например SGI Altix, эффективность OpenMP-программ невысока, что заставляет программистов использовать MPI;

G. организация взаимодействия потоков через общие переменные, а не через передачу сообщений, часто приводит к трудно обнаруживаемым ошибкам (race condition – условия гонок), а необходимые для поиска таких ошибок средства отладки – либо отсутствуют вообще, либо мало доступны.

3. Что понимается под параллельной программой в рамках технологии OpenMP?

Под параллельной программой в рамках OpenMP понимается программа, для которой в специально указываемых при помощи директив местах - параллельных фрагментах - исполняемый программный код может быть разделен на несколько раздельных командных потоков ( threads ). В общем виде программа представляется в виде набора последовательных ( однопотоковых ) и параллельных ( многопотоковых ) участков программного кода

4. Какие проблемы возникают при использовании общих данных в параллельно выполняемых потоках?

Пусть два потока исполняют один и тот же программный код: n=n+1; для общей переменной n. Тогда эта операция может быть выполнена поочередно (что приведет к получению правильного результата), либо оба потока могут одновременно прочитать значение переменной n, увеличить и записать в эту переменную новое значение (будет получено неправильное значение).

5. Какой формат записи директив OpenMP?

#pragma omp <имя дерективы> [<параметр> [[,] <параметр>]...]

Для иллюстрации приведем пример директивы:

#pragma omp parallel default(shared)

\ private(beta,pi)

6. В чем состоит назначение директивы parallel?

Для выделения параллельных фрагментов программы следует использовать директиву parallel: #pragma omp parallel [ ...]

7. В чем состоят понятия фрагмента, области и секции параллельной программы?

Для обозначения динамически возникающих параллельно выполняемых участков программного кода в OpenMP используется понятие параллельных областей.

- Параллельный фрагмент (parallel construct) – блок программы, управляемый директивой parallel.

- Параллельная область (parallel region) – параллельно выполняемые участки программного кода, динамическивозникающие в результате вызова функций из параллельных фрагментов.

- Параллельная секция (parallel section)– часть параллельного фрагмента, выделяемая для параллельного выполнения при помощи директивы section.

8. Как осуществляется распараллеливание циклов в OpenMP? Какие условия должны выполняться, чтобы циклы могли быть распараллелены?

Для распараллеливания циклов в OpenMP применяется директива for:

#pragma omp for [ <параметр> ...] <цикл>

Важно отметить, что для распараллеливания цикл for должен иметь «вид» цикла со счетчиком): for (index = first; index < end; increment\_expr)

9. Какие возможности имеются в OpenMP для управления распределением итераций циклов между потоками?

Возможность управления распределением итераций цикла между потоками в OpenMP обеспечивается при помощи параметра schedule директивы for. Поле type параметра schedule может принимать следующие значения:

static – статический способ распределения итераций до начала выполнения цикла. Если поле chunk не указано, то итерации делятся поровну между потоками. При заданном значении chunk итерации цикла делятся на блоки размера chunk и эти блоки распределяются между потоками до начала выполнения цикла.

- dynamic – динамический способ распределения итераций. До начала выполнения цикла потокам выделяются блоки итераций размера chunk (если поле chunk не указано, то полагается значение chunk = 1). Дальнейшее выделение итераций (также блоками размера chunk) осуществляется в момент завершения потоками своих ранее назначенных итераций.

- guided – управляемый способ распределения итераций. Данный способ близок к предшествующему варианту, отличие в том, что начальный размер блоков итераций определяется в соответствии с некоторым параметром среды реализации OpenMP, а затем уменьшается экспоненциально, при каждом новом выделении блока итераций.

При этом получаемый размер блока итераций не должен быть меньше значения chunk

- runtime – способ распределения итераций, при котором выбор конкретной схемы (из ранее перечисленных) осуществляется в момент начала выполнения программы в соответствии со значением переменной окружения OMP\_SCHEDULE. Так, например, для задания динамического способа при размере блока итераций 3, следует определить: setenv OMP\_SCHEDULE "dynamic,3" здесь способ распределения итераций между потоками можно менять, не корректируя при этом код программы (т. е. без повторной компиляции и сборки программы).

10. Как определяется порядок выполнения итераций в распараллеливаемых циклах?

По умолчанию, все потоки, прежде чем перейти к выполнению дальнейших вычислений, ожидают окончания выполнения итераций цикла даже, если некоторые из них уже завершили свои вычисления – конец цикла представляет собой некоторый барьер, который потоки могут преодолеть только все вместе. Можно отменить указанную синхронизацию, указав параметр nowait в директиве for – тогда потоки могут продолжить вычисления за переделами цикла, если для них нет итераций цикла для выполнения.

**Код:**

#include <omp.h> // заголовочный файл «omp.h»

#include <iostream>

#include <time.h>

using namespace std;

const int NMAX = 500;

const int MMAX = 500;

void main()

{

int i, j;

int sum;

float a[NMAX][MMAX];

float counter;

srand(time(0));

setlocale(0, "");

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

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

a[i][j] = rand() % 10 + 1;

}

}

// инициализация данных

clock\_t start = clock();

#pragma omp parallel shared(a) private(i)

{

#pragma omp for private(j,sum)

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

{

counter = 0;

for (j = 0; j < NMAX - 1; j++) {

sum = 0;

sum = a[i][j] + a[i][j + 1];

while (sum > 0) {

if ((sum % 10) == 7) {

counter++;

}

sum /= 10;

}

}

cout << "\nNumber of 7 in row " << i << " : " << counter << endl;

}

} /\* Завершение параллельного фрагмента \*/

clock\_t end = clock();

printf("\_\_\_\_\_\_\_\_\_\_Time with MP\_\_\_\_\_\_\_\_\_: %f", difftime(end, start));

start = clock();

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

{

counter = 0;

for (j = 0; j < NMAX - 1; j++) {

sum = 0;

sum = a[i][j] + a[i][j + 1];

while (sum > 0) {

if ((sum % 10) == 7) {

counter++;

}

sum /= 10;

}

}

cout << "\nNumber of 7 in row " << i << " : " << counter << endl;

}

/\* Завершение параллельного фрагмента \*/

end = clock();

printf("\_\_\_\_\_\_\_\_\_\_Time without MP\_\_\_\_\_\_\_\_\_: %f", difftime(end, start));

}

**Результат выполнения:**

\_\_\_\_\_\_\_\_\_\_Time with MP\_\_\_\_\_\_\_\_\_: 1537,000000

\_\_\_\_\_\_\_\_\_\_Time without MP\_\_\_\_\_\_\_\_\_: 1337,000000

Параллельное программирование повышает эффективность только тогда, когда каждая итерация достаточно затратна с точки зрения процессорного времени.