Параллельные алгоритмы: OpenMP. Основные прагмы

Н. И. Хохлов

МФТИ, Долгопрудный

11 мая 2017 г.

Директива for

for

#pragma omp for [clause[[,]clause] ...]

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- schedule(type[, chunk])
- collapse(n)
- ordered
- nowait

Директива for

lastprivate(list)

Переменным, перечисленным в списке, присваивается результат с последнего витка цикла.

schedule(type[, chunk])

Опция задаёт, каким образом итерации цикла распределяются между нитями;

Директива for

collapse(n)

Опция указывает, что n последовательных тесновложенных циклов ассоциируется с данной директивой; для циклов образуется общее пространство итераций, которое делится между нитями; если опция collapse не задана, то директива относится только к одному непосредственно следующему за ней циклу.

ordered

Опция, говорящая о том, что в цикле могут встречаться директивы ordered; в этом случае определяется блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле.

Директива for

На вид параллельных циклов накладываются достаточно жёсткие ограничения. В частности, предполагается, что корректная программа не должна зависеть от того, какая именно нить какую итерацию параллельного цикла выполнит. Нельзя использовать побочный выход из параллельного цикла. Размер блока итераций, указанный в опции schedule, не должен изменяться в рамках цикла.

Формат параллельного цикла

for([целочисленный тип] i = инвариант цикла; i <,>,=,<=,>= инвариант цикла; i +,-= инвариант цикла)

Директива for

Итеративная переменная распределяемого цикла по смыслу должна быть локальной, поэтому в случае, если она специфицирована общей, то она неявно делается локальной при входе в цикл. После завершения цикла значение итеративной переменной цикла не определено, если она не указана в опции lastprivate.

Пример 1

Пример 1

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
    int A[10], B[10], C[10], i, n;
    for (i=0; i<10; i++)A[i]=i; B[i]=2*i; C[i]=0;
#pragma omp parallel shared(A,B,C) private(i,n)
{
    n=omp_get_thread_num();
#pragma omp for
    for (i=0; i<10; i++){
        C[i]=A[i]+B[i];
        printf("Task %d element %d\n  n, i);
    }
}
}
```

Директива for, опция schedule

В опции schedule параметр type задаёт следующий тип распределения итераций.

- static
- dynamic
- guided
- auto
- runtime

Директива for, опция schedule

static

Блочно-циклическое распределение итераций цикла; размер блока – chunk. Первый блок из chunk итераций выполняет нулевая нить, второй блок — следующая и т.д. до последней нити, затем распределение снова начинается с нулевой нити. Если значение chunk не указано, то всё множество итераций делится на непрерывные куски примерно одинакового размера (конкретный способ зависит от реализации), и полученные порции итераций распределяются между нитями.

Директива for, опция schedule

dynamic

Динамическое распределение итераций с фиксированным размером блока: сначала каждая нить получает chunk итераций (по умолчанию chunk=1), та нить, которая заканчивает выполнение своей порции итераций, получает первую свободную порцию из chunk итераций. Освободившиеся нити получают новые порции итераций до тех пор, пока все порции не будут исчерпаны. Последняя порция может содержать меньше итераций, чем все остальные.

Директива for, опция schedule

guided

Динамическое распределение итераций, при котором размер порции уменьшается с некоторого начального значения до величины chunk (по умолчанию chunk=1) пропорционально количеству ещё не распределённых итераций, делённому на количество нитей, выполняющих цикл. Размер первоначально выделяемого блока зависит от реализации. В ряде случаев такое распределение позволяет аккуратнее разделить работу и сбалансировать загрузку нитей. Количество итераций в последней порции может оказаться меньше значения chunk.

Директива for, опция schedule

auto

Способ распределения итераций выбирается компилятором и/или системой выполнения. Параметр chunk при этом не задаётся.

runtime

Способ распределения итераций выбирается во время работы программы по значению переменной среды OMP_SCHEDULE. Параметр chunk при этом не задаётся.

Пример 2

Пример 2

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
    int i;
#pragma omp parallel private(i)
{
#pragma omp for schedule (static, 2)
    for (i=0; i<10; i++){
        printf("Thread %d iteration %d\n
                omp_get_thread_num(), i);
        sleep(1);
    }
}
}
```

Директива for, опция schedule

Значение по умолчанию переменной OMP_SCHEDULE зависит от реализации. Если переменная задана неправильно, то поведение программы при задании опции runtime также зависит от реализации.

Задать значение переменной OMP_SCHEDULE в Linux в командной оболочке bash можно при помощи команды следующего вида:

OMP_SCHEDULE

```
export OMP_SCHEDULE="dynamic,1"
```

omp_set_schedule

void omp_set_schedule(omp_sched_t type, int chunk);

Изменить значение переменной OMP_SCHEDULE из программы.

Допустимые константы описаны в файле omp.h, как минимум присутствуют следующие:

```
typedef enum omp_sched_t {
    omp_sched_static,
    omp_sched_dynamic,
    omp_sched_guided,
    omp_sched_auto
} omp_sched_t;
```

omp_get_schedule

void omp_get_schedule(omp_sched_t* type, int* chunk);

Узнать текущее значение переменной OMP_SCHEDULE.

Директива for, зависимость по данным

При распараллеливании цикла нужно убедиться в том, что итерации данного цикла не имеют информационных зависимостей. В этом случае его итерации можно выполнять в любом порядке, в том числе параллельно. Компилятор это не проверяет. Если дать указание компилятору распараллелить цикл, содержащий зависимости, результат работы может оказаться некорректным.

Директива sections

sections

#pragma omp sections [clause[[,]clause] ...]

- private(list)
- firstprivate(list)
- lastprivate(list)
- reduction(operator: list)
- nowait

Определяет набор независимых секций кода, каждая из которых выполняется своей нитью.

lastprivate(list)

Переменным присваивается результат из последней секции.

Директива section

section

#pragma omp section

Задаёт участок кода внутри секции sections для выполнения одной нитью.

section

Перед первым участком кода в блоке sections директива section не обязательна. Какие нити будут задействованы для выполнения какой секции, не специфицируется. Если количество нитей больше количества секций, то часть нитей для выполнения данного блока секций не будет задействована. Если количество нитей меньше количества секций, то некоторым (или всем) нитям достанется более одной секции.

Пример 3

Пример 3

```
int n;
#pragma omp parallel private(n)
{
    n=omp_get_thread_num();
#pragma omp sections
{
#pragma omp section
    printf("First section, task %d\n  n);
#pragma omp section
    printf("Second section, task %d\n  n);
}
    printf("Parallel section, task %d\n  n);
}
```

Пример 4

Пример 4

```
    int n=0;
#pragma omp parallel
{
#pragma omp sections lastprivate(n)
{
#pragma omp section
    n=1;
#pragma omp section
    n=2;
}
    printf("Value n at thread %d: %d\n
            omp_get_thread_num(), n);
}
    printf("Value of n: %d\n  n);
```

Директива task

task

#pragma omp task [clause[[,]clause] ...]

- if(expression)
- untied
- default(shared|none);
- private(list)
- firstprivate(list)
- shared(list)

Текущая нить выделяет в качестве задачи ассоциированный с директивой блок операторов. Задача может выполняться немедленно после создания или быть отложенной на неопределённое время и выполняться по частям. Размер таких частей, а также порядок выполнения частей разных отложенных задач определяется реализацией.

Директива task

if(expression)

Порождение новой задачи только при выполнении некоторого условия; если условие не выполняется, то задача будет выполнена текущей нитью и немедленно.

untied

Опция означает, что в случае откладывания задача может быть продолжена любой нитью из числа выполняющих данную параллельную область; если данная опция не указана, то задача может быть продолжена только породившей её нитью.

Директива taskwait

taskwait

```
#pragma omp taskwait
```

Нить, выполнившая данную директиву, приостанавливается до тех пор, пока не будут завершены все ранее запущенные данной нитью независимые задачи.

Директива ordered

ordered

#pragma omp ordered

Директива ordered определяет блок внутри тела цикла, который должен выполняться в том порядке, в котором итерации идут в последовательном цикле.

Относится к самому внутреннему из объемлющих циклов, а в параллельном цикле должна быть задана опция ordered. Нить, выполняющая первую итерацию цикла, выполняет операции данного блока. Нить, выполняющая любую следующую итерацию, должна сначала дождаться выполнения всех операций блока всеми нитями, выполняющими предыдущие итерации. Может использоваться, например, для упорядочения вывода от параллельных нитей.

Пример 5

Пример 5

```
#pragma omp parallel private (i, n)
{
    n=omp_get_thread_num();
#pragma omp for ordered
    for (i=0; i<5; i++)
    {
        printf("Task %d, iteration %d\n  n, i);
#pragma omp ordered
        {
            printf("ordered: Task %d, iteration %d\n  n
        }
    }
}
}
```

Директива critical

critical

#pragma omp critical

С помощью директивы critical оформляется критическая секция программы.

В каждый момент времени в критической секции может находиться не более одной нити. Все другие нити, выполнившие директиву для секции с данным именем, будут заблокированы, пока вошедшая нить не закончит выполнение. Как только работавшая нить выйдет из критической секции, одна из заблокированных нитей войдет в неё. Если на входе стояло несколько нитей, то случайным образом выбирается одна из них, а остальные продолжают ожидание.

Директива atomic

atomic

#pragma omp atomic

Данная директива относится к идущему непосредственно за ней оператору присваивания (на используемые в котором конструкции накладываются достаточно понятные ограничения), гарантируя корректную работу с общей переменной, стоящей в его левой части. На время выполнения оператора блокируется доступ к данной переменной всем запущенным в данный момент нитям, кроме нити, выполняющей операцию. Атомарной является только работа с переменной в левой части оператора присваивания, при этом вычисления в правой части не обязаны быть атомарными.

Пример 6

Пример 6

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
int count = 0;
#pragma omp parallel
    {
#pragma omp atomic
        count++;
    }
    printf("Num of threads: %d\n  count);
}
```

Спасибо за внимание! Вопросы?