Параллельные алгоритмы: OpenMP. Введение

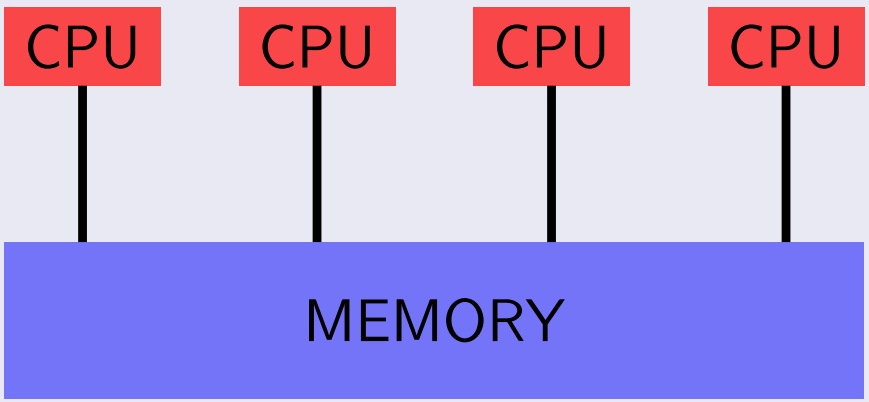
Н. И. Хохлов

МФТИ, Долгопрудный

11 мая 2017 г.

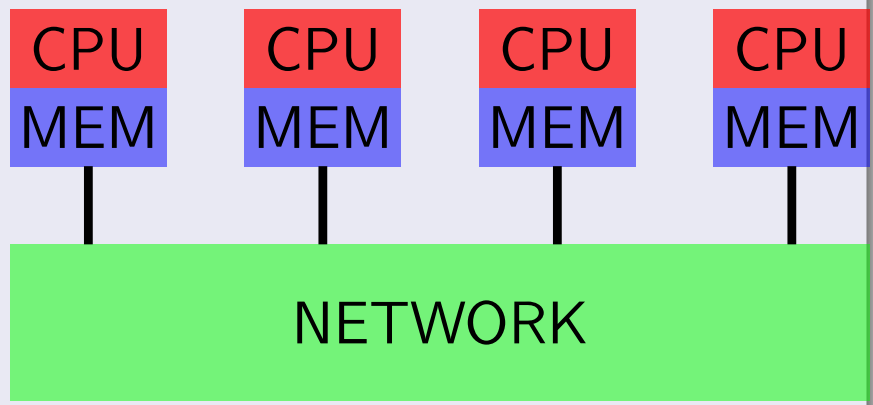
Архитектуры параллельных вычислительных машин

Общая память (shared memory)



Пример – современные много-ядерные/многопроцессорные машины.

Распределенная память (distributed memory)



Пример – несколько одноядерных машин объеденeнных в сеть.

Большинство современных архитектур – гибридные, т. е. многоядерные машины, объеденeнные в одну общую сеть.

Что такое OpenMP?

- OpenMP – стандарт для написания приложений на масштабируемых SMP-системах в модели общей памяти.
- OpenMP – не библиотека, а спецификация (стандарт) для программистов и пользователей.
- Поддержка языков C/C++ и Fortran.

История появления стандарта OpenMP

Особенности

- Один вариант программы для параллельного и последовательного выполнения.
- Любой процесс состоит из нескольких нитей управления, которые имеют общее адресное пространство, но разные потоки команд и раздельные стеки.
- SPMD-модель (Single Program Multiple Data).
- Состоит из трех основных компонент API
  - директив компилятора;
  - библиотечных функций/процедур;
  - переменных окружения.

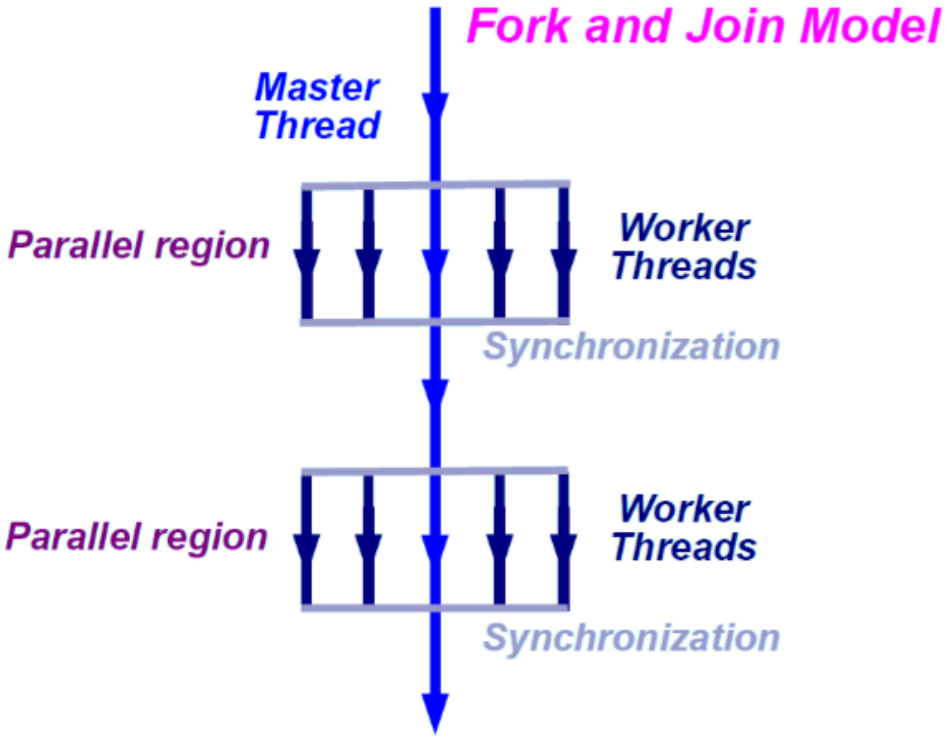
Основные преимущества

- Портируемый:
  - определен для языков C/C++ и Fortran;
  - мульти-платформный, поддерживается на большинстве стандартных ОС.
- Широко распространенный:
  - разрабатывается и поддерживается группой крупных производителей вычислительной техники и программного обеспечения.

Ограничения

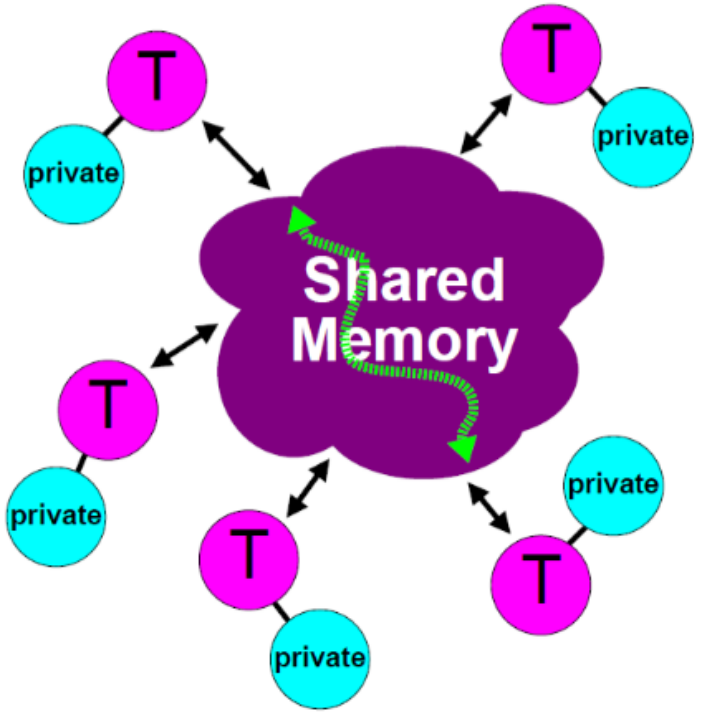
- Не подходит для систем с распределенной памятью.
- Не дает автоматического распараллеливания при компиляции.
- Не гарантирует защиту от зависимостей и конфликтов данных, условий гонки и тупиков.
- Не гарантирует синхронного ввода/вывода в файл, синхронизизацию должен обеспечить программист.

Модель программирования OpenMP



- Программа начинается с последовательной секции (один поток, мастер-поток).
- При врохе в параллельную секцию порождаются нити.
- По завершению параллельной области все нити завершаются, кроме мастер-нити.
- Модель программирования fork-join.

Модель данных OpenMP



Переменные двух видов:

- shared (общие; все нити видят одну и ту же переменную);
- private (локальные, приватные; каждая нить видит свой экземпляр данной переменной).

Модель данных OpenMP

- Все потоки имеют доступ к одной, общей для всех, разделяемой памяти.
- Данные могут быть разделяемые/общие (shared) и локальные (private).
- Разделяемые данные доступны всем потокам.
- Локальные — только потоку-владельцу этих данных.
- Транспорт данных прозрачен для программиста.
- Имеет место неявная синхронизация.

По умолчанию, все переменные, порождённые вне параллельной области, при входе в неё остаются общими. Исключение составляют переменные, являющиеся счетчиками итераций в цикле. Переменные, порождённые внутри параллельной области, по умолчанию являются локальными.

Поддержка OpenMP

- Задан макрос _OPENMP.
- Формат yyyymm, где yyyy и mm – цифры года и месяца, когда был принят поддерживаемый стандар.

Пример

```
#include <stdio.h>
int main()
{
#ifdef _OPENMP
    printf("OpenMP is supported, version %d.\n _OPENMP);
#endif
    return 0;
}
```

Директивы OpenMP

Формат директив (C/C++)

#pragma omp directive [clause ...]

- Большинство директив OpenMP поддерживают параметры (clause).
- Данные параметры обеспечивают поддержку дополнительной информации к директивам.
- Объектом действия большинства директив является один оператор или блок, перед которым расположена директива в исходном тексте программы.

Функции OpenMP

Заголовочный файл (C/C++)

```
#include <omp.h>
```

- Функции и типы данных начинаются с префикса omp_.
- Для использования директив подключение заголовочного файла не требуется.
- Большинство функционала реализовано через директивы.

Директива parallel

parallel

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

- if(scalar-expression)
- num_threads(integer-expression)
- default(shared | none)
- private(list)
- firstprivate(list)
- shared(list)
- copyin(list)
- reduction(operator: list)

Директива parallel

if(scalar-expression)

Выполнение параллельной области по условию.

num_threads(integer-expression)

Явное задание количества нитей, которые будут выполнять параллельную область.

default(shared | none)

Всем переменным в параллельной области, которым явно не назначен класс, будет назначен класс shared; none – всем переменным класс назначается явно.

private(list)

Переменные, для которых порождается локальная копия в каждой нити; начальное значение не определено.

Директива parallel

firstprivate(list)

Переменные, для которых порождается локальная копия в каждой нити; локальные копии инициализируются значениями этих переменных в нити-мастере.

shared(list)

Переменные, для которых порождается локальная копия в каждой нити; локальные копии инициализируются значениями этих переменных в нити-мастере.

copyin(list)

Переменные, объявленные как threadprivate, которые при входе в параллельную область инициализируются значениями соответствующих переменных в нити-мастере.

Директива parallel

reduction(operator: list)

Задаёт оператор и список общих переменных; для каждой переменной создаются локальные копии в каждой нити; они инициализируются соответственно типу оператора (для аддитивных – 0 или аналоги, для мультипликативных – 1 или аналоги); после выполнения всех операторов параллельной области выполняется заданный оператор; оператор это: для языка Си – +, *, -, &, |, ^, &&, ||; порядок выполнения операторов не определён, поэтому результат может отличаться от запуска к запуску.

Пример 1

Пример 1

```
int main(int argc, char *argv[])
{
    printf("Master thread 1\n");
#pragma omp parallel
{
    printf("Parallel region\n");
}
    printf("Master thread 2\n");
    return 0;
}
```

Пример 2

Пример 2

```
int main(int argc, char *argv[])
{
    int count = 0;
#pragma omp parallel reduction (+: count)
{
    count++;
    printf("Current value count: %d\n  count);
}
    printf("Number of threads: %d\n  count);
    return 0;
}
```

Директива parallel: число потоков

- По умолчанию порождается число потоков равное числу ядер на системе.
- Можно задать опцией num_threads(integer-expression).
- Можно задать функцией void omp_set_num_threads(int num).
- Можно задать через переменные окружения OMP_NUM_THREADS.

OMP_NUM_THREADS

export OMP_NUM_THREADS = 5

или

OMP_NUM_THREADS = 5 ./my_program

Пример 3

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
#pragma omp parallel
{
    printf("Parallel region 1\n");
}
omp_set_num_threads(2);
#pragma omp parallel num_threads(3)
{
    printf("Parallel region 2\n");
}
#pragma omp parallel
{
    printf("Parallel region 3\n");
}
```

Функции для работы с потоками

int omp_get_max_threads(void);

Возвращает максимально допустимое число нитей для использования в следующей параллельной области.

int omp_get_num_procs(void);

Возвращает количество процессоров, доступных для использования программе пользователя на момент вызова. Нужно учитывать, что количество доступных процессоров может динамически изменяться.

Функции для работы с потоками

int omp_in_parallel(void);

Возвращает 1, если она была вызвана из активной параллельной области программы.

Пример

```
void mode(void) {
    if(omp_in_parallel()) printf("Parallel region\n");
    else printf("Serial region\n");
}

int main(int argc, char *argv[]) {
    mode();
#pragma omp parallel
    mode();
}
```

Директива single

single

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

- private(list);
- firstprivate(list);
- copyprivate(list) – новые значения переменных списка будут доступны всем одноименным частным переменным других потоков (private и firstprivate); опция не может использоваться совместно с опцией nowait; переменные списка не должны быть перечислены в опциях private и firstprivate данной директивы single;
- nowait – не делать неявную синхронизацию (по умолчанию делается синхронизация).

Директива single

Пример

```
#pragma omp parallel
{
    printf("Message 1\n");
#pragma omp single nowait
    {
        printf("Single\n");
    }
    printf("Message 2\n");
}
```

Директива master

master

#pragma omp master

Выделяет участок кода, который будет выполнен только нитью-мастером. Остальные нити пропускают данный участок и продолжают работу с оператора, расположенного следом. Неявной синхронизации не предполагает.

Пример

```
#pragma omp parallel private(n)
{
    n=1;
#pragma omp master
    n=2;
    printf("n: %d\n  n);
}
```

Нумерация потоков

Все нити в параллельной области нумеруются последовательными целыми числами от 0 до N-1, где N — количество нитей, выполняющих данную область.

int omp_get_thread_num(void);

Позволяет нити получить свой уникальный номер в текущей параллельной области.

int omp_get_num_threads(void);

Позволяет нити получить количество нитей в текущей параллельной области.

Пример

Пример

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
    int n=1;
    printf("serial (1): %d\n n);
#pragma omp parallel private(n)
{
        printf("parallel (1): %d\n n);
        n=omp_get_thread_num();
        printf("parallel (2): %d\n n);
}
    printf("serial (2): %d\n n);
}
```

Пример

Пример

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
    int n=1;
    printf("serial (1): %d\n n);
#pragma omp parallel firstprivate(n)
{
        printf("parallel (1): %d\n n);
        n=omp_get_thread_num();
        printf("parallel (2): %d\n n);
}
    printf("serial (2): %d\n n);
}
```

Директива barrier

barrier

#pragma omp barrier

Барьерная синхронизация потоков внутри параллельной секции.

Пример

```
#pragma omp parallel
{
    printf("Msg 1\n");
#pragma omp barrier
    printf("Msg 2\n");
}
```

Компиляция

У компилятора gcc опция -fopenmp.

Компиляция

```
gcc -fopenmp hello.c -o hello
OMP_NUM_THREADS=10 ./hello
```

Спасибо за внимание! Вопросы?