Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс

Теория и практика параллельных вычислений

Лекция 15

Методы разработки параллельных программ для многопроцессорных систем с общей памятью (стандарт OpenMP)

Гергель В.П.

Содержание

Стратегия подхода…

Интерфейс OpenMP задуман как стандарт параллельного программирования для многопроцессорных систем с общей памятью (SMP, ccNUMA, …)

В общем вид системы с общей памятью описываются в виде модели параллельного компьютера с произвольным доступом к памяти (*parallel random-access machine – PRAM*)

Стратегия подхода…

Динамика развития стандарта

- OpenMP Fortran API v1.0 (1997)
- OpenMP C/C++ API v1.0 (1998)
- OpenMP Fortran API v2.0 (2000)
- (OpenMP C/C++ API v2.0 – ожидается в 2001)

Разработкой стандарта занимается организация OpenMP ARB (ARchitecture Board), в которую вошли представители крупнейших компаний - разработчиков SMP-архитектур и программного обеспечения.

Стратегия подхода…

При разработке стандарта в значительной степени учитывались результаты предшествующих исследований

- X3H5 (ANSI стандарт, 1993) - MPP
- POSIX Threads (*Ptheads*), IEEE 1003.1c, 1995 - Unix

Стратегия подхода…

Основания для достижения эффекта – разделяемые для параллельных процессов данные располагаются в общей памяти и для организации взаимодействия не требуется операций передачи сообщений.

Интерфейс OpenMP служит основой для формирования новой технологии параллельного программирования для систем общей памяти (*shared-memory programming*)

Стратегия подхода…

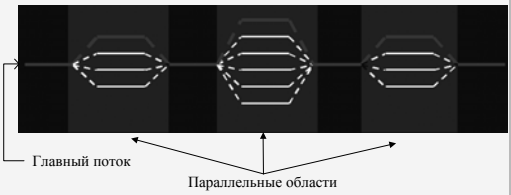
Положительные стороны

- Поэтапное (инкрементальное) распараллеливание
  - Можно распараллеливать последовательные программы поэтапно, не меняя их структуру
- Единственность разрабатываемого кода
  - Нет необходимости поддерживать последовательный и параллельный вариант программы, поскольку директивы игнорируются обычными компиляторами (в общем случае)
- Эффективность
  - Учет и использование возможностей систем с общей памятью
- Стандартизованность (переносимость), поддержка в наиболее распространенных языках (C, Fortran) и платформах (Windows, Unix)

Стратегия подхода…

Принцип организации параллелизма…

- Использование потоков (общее адресное пространство)
- Пульсирующий ("вилочный", *fork-join*) параллелизм



Стратегия подхода

Принцип организации параллелизма…

- При выполнении обычного кода (вне параллельных областей) программа выполняется одним потоком (master thread)
- При появлении директивы #parallel происходит создание "команды" (team) потоков для параллельного выполнения вычислений
- После выхода из области действия директивы #parallel происходит синхронизация, все потоки, кроме master, уничтожаются
- Продолжается последовательное выполнение кода (до очередного появления директивы #parallel)

Стратегия подхода

Структура

- Набор директив компилятора
- Библиотека функций
- Набор переменных окружения

Изложение материала будет проводиться на примере языка C

Директивы OpenMP

Формат записи директив

Формат

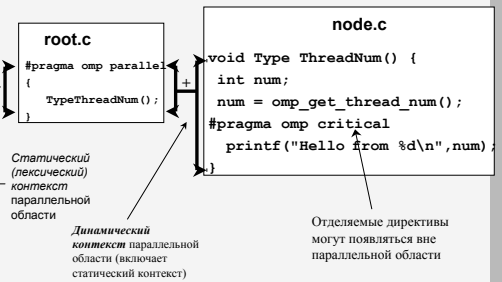
```
#pragma omp имя_директивы [clause,…]
```

Пример

```
#pragma omp parallel default(shared) \
                        private(beta,pi)
```


Директивы OpenMP

Области видимости директив



Типы директив

Директивы OpenMP можно разделить на 3 типа:

- определение параллельной области,
- разделение работы,
- синхронизация.

Определение параллельной области…

Директива **parallel** (основная директива OpenMP)

- Когда поток достигает директиву **parallel**, создается набор (*team*) из N потоков; входной поток является основным потоком этого набора (*master thread*) и имеет номер 0;
- Код области дублируется или разделяется между потоками для параллельного выполнения;
- В конце области обеспечивается синхронизация потоков – выполняется ожидание завершения вычислений всех потоков; далее все потоки завершаются – дальнейшие вычисления продолжает выполнять только основной поток.

Определение параллельной области…

Директива **parallel** (формат)

```
#pragma omp parallel [clause ...] newline
   structured_block
```

clause

```
   if (scalar_expression)
   private (list)
   shared (list)
   default (shared | none)
   firstprivate (list)
   reduction (operator: list)
   copyin (list)
```


Определение параллельной области…

Директива **parallel** (семантика)

- Количество потоков (по убыванию старшинства)
  - **omp_set_num_threads()**
  - **OMP_NUM_THREADS**
  - параметр реализации
- Динамические потоки
  - По умолчанию количество потоков одинаковое – для активизации динамического режима функция **omp_set_dynamic()** или переменная **OMP_DYNAMIC**
- Вложенность параллельных областей (во многих реализациях не обеспечивается)
  - по умолчанию во вложенной области создается один поток;
  - управление - функция **omp_set_nested()** или переменная **OMP_NESTED**
- Параметры (clause) – если условие в if не выполняется, то процессы не создаются

Определение параллельной области

Директива **parallel** (пример)

```
#include <omp.h>
main () {
 int nthreads, tid;
 /* Создание параллельной области */
#pragma omp parallel private(nthreads, tid)
 {
  /* печать номера потока */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);
  /* Печать количества потоков - только master */
  if (tid == 0) {
   nthreads = omp_get_num_threads();
   printf("Number of threads = %d\n", nthreads);
  }
 }/* Завершение параллельной области  */
}
```

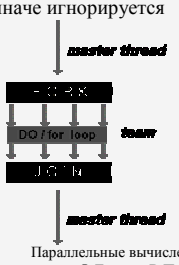

Распределение вычислений между потоками…

Существует 3 директивы для распределения вычислений в параллельной области

- **DO / for** – распараллеливание циклов
- **sections** – распараллеливание раздельных фрагментов кода (функциональное распараллеливание)
- **single** – директива для указания последовательного выполнения кода

! Начало выполнения директив по умолчанию не синхронизируется, завершение директив по умолчанию является синхронным

Директива **DO/for** – распределение итераций цикла между потоками; параллельная область уже должна быть активна, иначе игнорируется



Директива **DO/for** (формат)

```
#pragma omp for [clause ...] newline
  for_loop
clause
 schedule (type [,chunk])
 ordered
 private (list)
 firstprivate (list)
 lastprivate (list)
 shared (list)
 reduction (operator: list)
 nowait
```


Директива **DO/for** - распределение итераций регулируется параметром **schedule**

- **static** – итерации делятся на блоки по chunk итераций и статически разделяются между потоками; если параметр chunk не определен, итерации делятся между потоками равномерно и непрерывно
- **dynamic** – распределение итерационных блоков осуществляется динамически (по умолчанию chunk=1)
- **guided** – размер итерационного блока уменьшает экспоненциально при каждом распределении; chunk определяет минимальный размер блока (по умолчанию chunk=1)
- **runtime** – правило распределения определяется переменной **OMP_SCHEDULE** (при использовании runtime параметр chunk задаваться не должен)

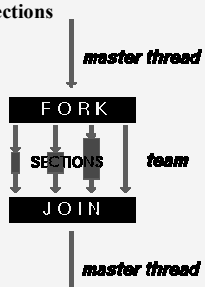
Директива **DO/for** (пример)

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
main () {
 int i, n, chunk;
 float a[NMAX], b[NMAX], c[NMAX];
 /* Some initializations */
 for (i=0; i < NMAX; i++)
  a[i] = b[i] = i * 1.0;
 n = NMAX;
 chunk = CHUNK;
 #pragma omp parallel shared(a,b,c,n,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 */
}
```


Директива **sections** – распределение вычислений для раздельных фрагментов кода

- фрагменты выделяются при помощи директивы **section**
- каждый фрагмент выполняется однократно
- разные фрагменты выполняются разными потоками
- завершение директивы по умолчанию синхронизируется
- директивы section должны использоваться только в статическом контексте

Директива **sections**



Распределение вычислений между потоками…

Директива **sections** (формат)

```
#pragma omp sections [clause ...] newline
{
 #pragma omp section newline
   structured_block
 #pragma omp section newline
   structured_block
}
clause
 private (list)
 firstprivate (list)
 lastprivate (list)
 reduction (operator: list)
 nowait
```

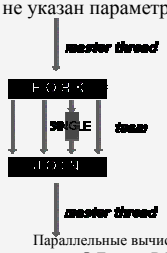

Распределение вычислений между потоками…

Директива **sections** (пример)

```
#include <omp.h>
#define NMAX 1000
main () {
 int i, n;
 float a[NMAX], b[NMAX], c[NMAX];
/* Some initializations */
 for (i=0; i < NMAX; i++)
   a[i] = b[i] = i * 1.0;
 n = NMAX;
 #pragma omp parallel shared(a,b,c,n) private(i)
 {
  #pragma omp sections nowait
  {
    #pragma omp section
      for (i=0; i < n/2; i++)
       c[i] = a[i] + b[i];
    #pragma omp section
      for (i=n/2; i < n; i++)
        c[i] = a[i] + b[i];
  } /* end of sections */
 } /* end of parallel section */
}
```


Распределение вычислений между потоками…

Директива **single** – определение фрагмента кода, который должен быть выполнен только одним потоком; все остальные потоки ожидают завершения выполнения фрагмента (если не указан параметр nowait)



Распределение вычислений между потоками…

Директива **single** (формат)

```
#pragma omp single [clause ...] newline
  structured_block

clause
 private (list)
 firstprivate (list)
 nowait
```


Распределение вычислений между потоками

Объединение директив **parallel** и **for/sections** в случае, если в параллельной области содержится только одну директиву **for** (**sections**)

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
main () {
 int i, n, chunk;
 float a[NMAX], b[NMAX], c[NMAX];
 /* Some initializations */
 for (i=0; i < NMAX; i++)
  a[i] = b[i] = i * 1.0;
 n = NMAX;
 chunk = CHUNK;
 #pragma omp parallel for \
   shared(a,b,c,n) private(i) \
   schedule(static,chunk)
  for (i=0; i < n; i++)
    c[i] = a[i] + b[i];
}
```


Информационные ресурсы

- www.openmp.org
- Что такое OpenMP - http://parallel.ru/tech/tech_dev/openmp.html
- OpenMP C/C++ specification v1.0 http:// www.openmp.org
- Introduction to OpenMP - www.llnl.gov/computing/tutorials/workshops/workshop/openMP/MAIN.html
- Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. (2000). *Parallel Programming in OpenMP*. Morgan Kaufmann Publishers.

Вопросы для обсуждения

- Возможности OpenMP для распараллеливания циклов
- Необходимые средства для синхронизации обработки данных в OpenMP

Задания для самостоятельной работы

- Разработка параллельного метода для вычисления числа π при использовании интерфейса OpenMP

Заключение

- Технология разработки параллельных программ для систем с общей памятью
- Основные положения стандарта OpenMP
- Директивы OpenMP по созданию параллельных областей и распределения вычислений между потоками

Следующая тема

- Методы разработки параллельных программ для многопроцессорных систем с общей памятью (стандарт OpenMP) – 2