

Введение в OpenMP

Алексей А. Романенко
arom@ccfit.nsu.ru

О чем эта лекция?

- Обзор OpenMP
- Сборка OpenMP программы
- Среда исполнения OpenMP
- Способы параллелизации программы с помощью OpenMP
- пр.

Содержание

- Программная модель OpenMP
- История OpenMP
- Обзор OpenMP
 - Условия (Clauses)
 - конструкции
 - Синхронизация потоков
 - Переменные окружения
 - Runtime functions

История OpenMP

- В начале 90-х, производители SMP систем поставляли похожие, на основе директив расширения к Фортрану:
 - Пользователь снабжает последовательные Fortran программы директивами, указывая, какие циклы должны исполняться параллельно
 - Компилятор отвечает за автоматическое распараллеливание циклов по процессорам SMP
 - Одна функциональность, но разная реализация
- Первая попытка стандарта ANSI - проект X3H5 в 1994 году. Он не был принят, в основном благодаря ослабевающему к нему интересу, поскольку популярность набирали системы с распределенной памятью.
- История OpenMP стандарта началась весной 1997 года, когда новые SMP машины стали широко распространены.

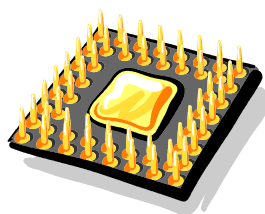
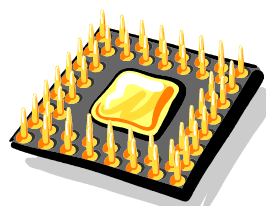
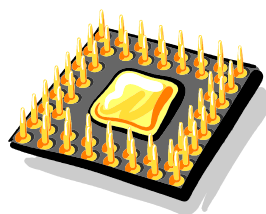
OpenMP сегодня

- Модель OpenMP мощный, но в тоже время компактный стандарт de-facto для программирования систем с общей памятью
- Текущая версия - 3.0
- Спецификация от мая 2008

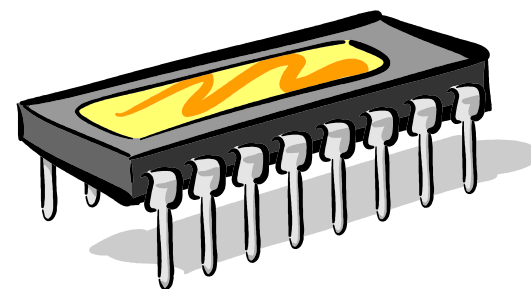
Цели OpenMP

- Быть стандартом для различных архитектур и платформ с распределенной памятью
- Дать простой, но ограниченный набор директив для параллелизации программы.
- Обеспечивать совместимость и возможность инкрементальной параллелизации программы.
- Дать возможность как для мелкозернистого распараллеливания, так и для крупнозернистого.
- Поддержка Fortran (77, 90 и 95), C, и C++

SMP системы

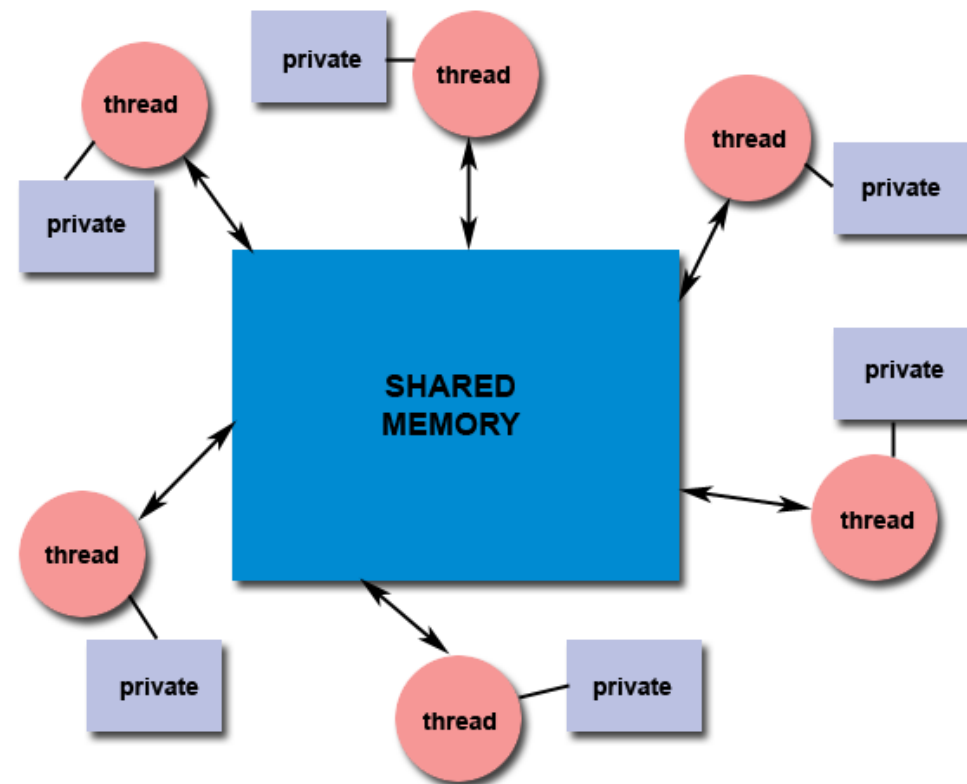


BUS



Модель с разделяемой памятью

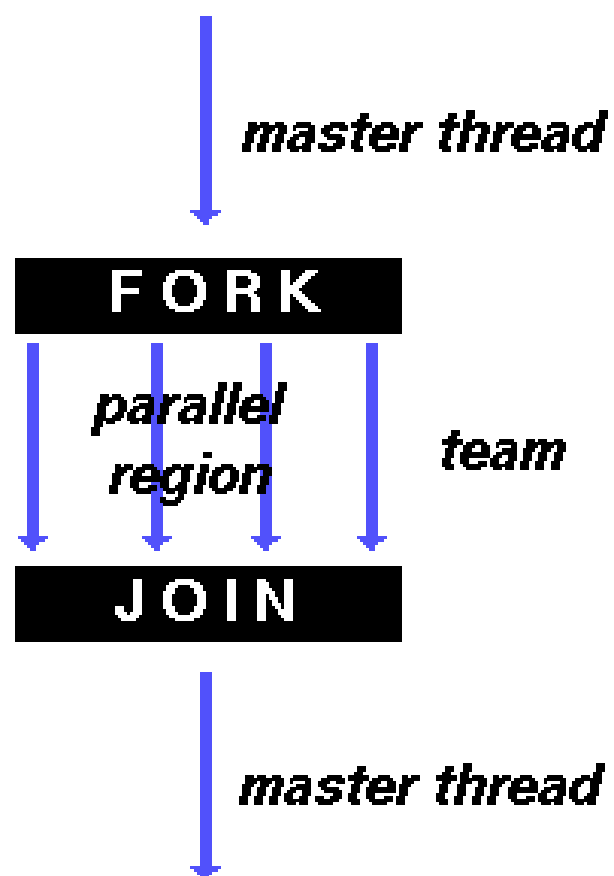
- Все потоки имеют доступ к глобальной разделяемой памяти
- Данные могут быть разделяемые и приватные
- Разделяемые данные доступны всем потокам
- Приватные — только одному
- Синхронизация требуется для доступа к общим данным



О данных

- В параллельных программах все данные имеют "метки":
 - Метка "Private" ⇨ видима только одному потоку
 - Изменения в переменной локальны и не видны другим потокам
 - Пример — локальная переменная в функции, которая выполняется параллельно
 - Метка "Shared" ⇨ видима всем потокам
 - Изменения в переменной видны всем потокам
 - Пример — глобальные данные

Модель выполнения OpenMP



Пример

Sequential code

```
void main() {  
    double x[1000];  
    for(i=0; i<1000; i++){  
        calc_smth(&x[i]);  
    }  
}
```

Parallel code

```
void main() {  
    double x[1000];  
    #pragma omp parallel for ...  
    for(i=0; i<1000; i++){  
        calc_smth(&x[i]);  
    }  
}
```

OpenMP Guided Tour

The logo for OpenMP features the word "OpenMP" in a teal, sans-serif font. The letter "O" is significantly larger than the other letters. A horizontal teal bar is positioned above the text, and another horizontal teal bar is positioned below the text, with the vertical stem of the letter "p" extending through the gap between these two bars. A small "TM" trademark symbol is located to the right of the "P".

OpenMP™

<http://www.openmp.org>

Когда использовать OpenMP?

- Компилятор не может выполнить параллелизацию кода, которую вы хотите:
- Цикл не параллелизуется:
 - Не возможно определить зависимость по данным между итерациями цикла
 - Не достаточная гранулярность
 - Компилятору не достаточно информации

Терминология

- OpenMP Team := Master + Workers
- Параллельный регион — блок кода, который всеми потоками исполняется одновременно
 - Поток мастер имеет ID 0
 - Все потоки синхронизируются при входе в параллельный регион
 - Параллельные регионы могут быть вложены, но поведение зависит от реализации
 - Условие "if" может быть использовано для «ограждения»; Если условие "false", код исполняется последовательно
- Работа в параллельном регионе распределяется между всеми потоками

Параллелизация цикла с помощью OpenMP

```
#pragma omp parallel shared(a,b)
{
#pragma omp for private(i)
  for(i=0; i<10000; i++)
    a[i] = a[i] + b[i];
}
```

Условие

Неявный барьер



Компоненты OpenMP

Формат директив

- C: рекистр имеет значение
 - Синтаксис: `#pragma omp directive [clause [clause] ...]`
- Продление: использовать «\» в прагме
- Условие компиляции: `_OPENMP` макрос определен

Пример

```
#ifdef _OPENMP
printf("Caution: The program was compiled with "
      "OpenMP and can consume all CPU resources "
      " of your PC!\n");
#endif
...
#omp parallel for private(i,j) \
  shared(a,b,c)
{
  for(i=0; i<100; i++)
    for(j=0; j<100; j++)
      a[i] = b[i][j]*c[j];
}
```



Некоторые OpenMP условия

Об OpenMP условиях

- Большинство OpenMP директив поддерживают условия
- Служат для задания дополнительной информации директивам
- Например, **private(a)** для директивы **for**:
 - **#pragma omp for *private(a)***

Условия if/private/shared

- if (скалярное выражение)
 - Выполнить параллельно, если выражение истинно
 - В противном случае - последовательно
- private (list)
 - Переменные не связаны с исходным объектом
 - Все переменные локальны
 - При входе и выходе значение переменных не определено
- shared (list)
 - Данные доступны всем потокам в группе
 - Все потоки имеют доступ к одним и тем же адресам

Пример

```
#omp parallel for private(i,j) \  
    shared(a,b,c) if(M>100)  
{  
    for(i=0; i<M; i++)  
        for(j=0; j<100; j++)  
            a[i] = b[i][j]*c[j];  
}
```

О хранении данных

- Значение частных переменных не определено при входе и выходе из параллельного региона
- Значение исходной переменной (до параллельного региона) не определено после выхода из региона!
- Частная переменная никак не связана с глобальной переменной с тем же именем
- Используйте `first/last private` условия для изменения такого поведения

Условие first/last private

- firstprivate (list)
 - Всем приватным переменным в списке присваивается значение исходных переменных до начала параллельного региона
- lastprivate (list)
 - Переменным присваивается значение того потока, который бы последним исполнялся последовательно.

Пример

```
#pragma omp parallel
{
#pragma omp for private(i) lastprivate(k)
    for(i=0; i<10; i++)
        k = i*i;
}
printf("k = %d\n", k); // k == 81
```

Пример

```
int myid, a;

a = 10;

#pragma omp parallel default(private) \
        firstprivate(a)
{
    myid = omp_get_thread_num();
    printf("Thread%d: a = %d\n", myid, a);
    a = myid;
    printf("Thread%d: a = %d\n", myid, a);
}
```

```
Thread1: a = 10
Thread1: a = 1
Thread2: a = 10
Thread0: a = 10
Thread3: a = 10
Thread3: a = 3
Thread2: a = 2
Thread0: a = 0
```

Условие default

- default (none | shared)
- none
 - Все задается явно
- shared
 - Все переменные разделяемые

Условие reduction - Пример

- Пример:

```
#pragma omp parallel
{
  #pragma for shared(x, sum) private(i)
  for(i=0; i<10000; i++)
    sum = sum + x[i];
}
```

- Нужна осторожность при работе с переменной SUM
- При использовании условия «reduction» компилятор заботится о синхронизации доступа к SUM

Условие reduction

- reduction (operator : list)
 - Редукционные переменные должны быть разделяемыми

```
#pragma omp parallel
{
  #pragma for shared(x) private(i) reduction(+:sum)
  for(i=0; i<10000; i++)
    sum += x[i];
}
```

```
#pragma omp parallel
{
  #pragma for shared(x) private(i) reduction(min:gmin)
  for(i=0; i<10000; i++)
    gmin = min(gmin, x[i]);
}
```

Условие `nowait`

- Используется для минимизации операций синхронизации

```
#pragma omp for nowait  
{  
    ...  
}
```

Параллельный регион

- Параллельный регион, в котором все потоки исполняют блок кода параллельно

```
#pragma omp parallel [clause[[,] clause] ...]
{
    "this will be executed in parallel"
} //implied barrier
```

Параллельный регион - условия

- Поддерживаются следующие условия:
 - if (scalar expression)
 - private (list)
 - shared (list)
 - default (none|shared)
 - reduction (operator: list)
 - copyin (list)
 - firstprivate (list)
 - num_threads (scalar_int_expr)

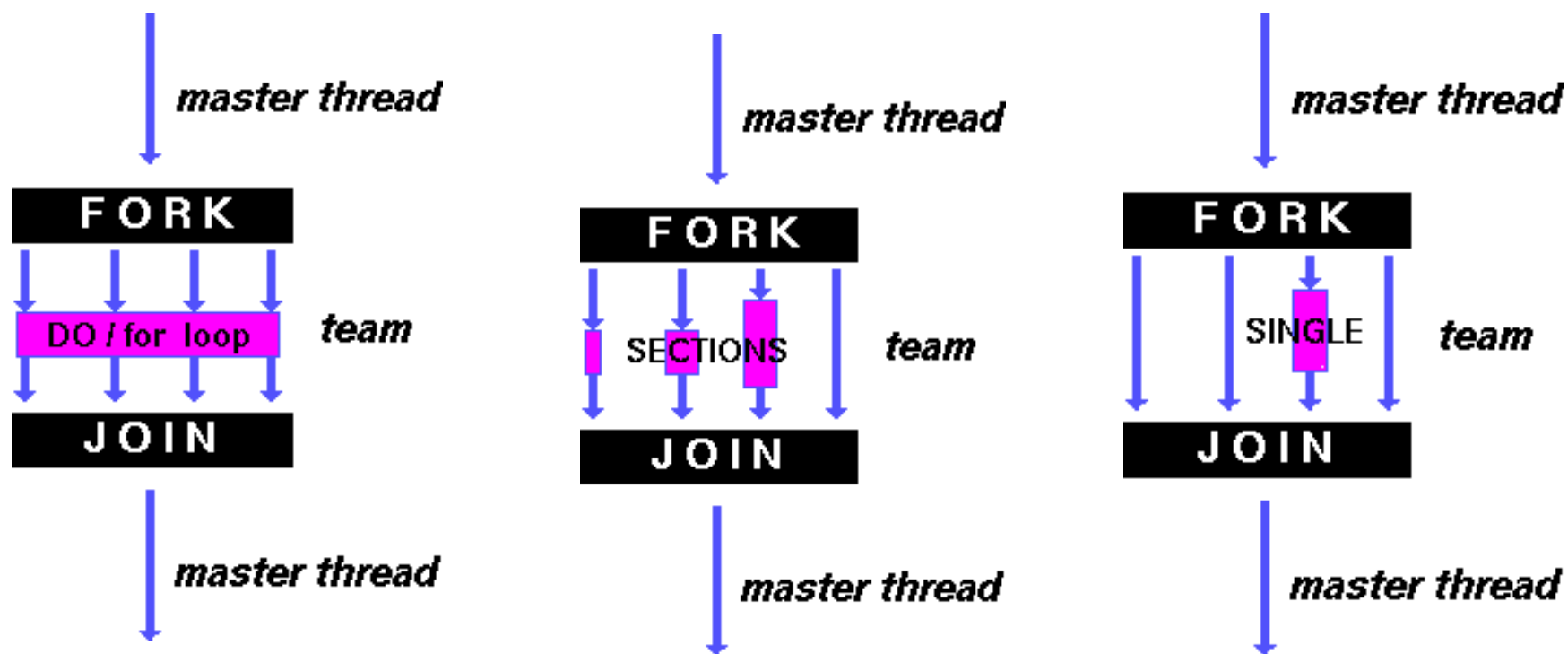


Директивы распределения работы

Конструкции распределения работы

- for, section, single
 - Работа распределяется по потокам
 - Должны быть внутри параллельного региона
 - Нет подразумеваемых барьеров на входе; подразумеваемых барьера на выходе (если NOWAIT указано)
 - Конструкция распределения работы не порождает дополнительного потока

Конструкции распределения работ



Директива `omp for`

- Итерации цикла распределяются по потокам

```
#pragma omp for [clause[[,] clause] ...]  
    <original for-loop>
```
- Поддерживаются следующие условия:
 - `private`
 - `firstprivate`
 - `lastprivate`
 - `reduction`
 - `ordered`
 - `schedule`
 - `nowait`

Балансировка загрузки

- Важные аспекты производительности
- Для обычных операций (например, векторное сложение) балансировка редко нужна
- Для менее регулярных операций требуется балансировка
- Примеры:
 - Транспонирование матриц
 - Умножение треугольных матриц
 - Поиск в списке

Условие schedule

- schedule (static | dynamic | guided [, chunk] | runtime)
- static [, chunk]
 - Итерации распределяются блоками размером "chunk" о потокам в циклической форме
 - При неуказании "chunk", каждый поток выполняет приблизительно N/P итераций, где длина цикла N и P потоков

Условие `schedule`

- `dynamic` [, `chunk`]
 - Фиксированная порция работы
 - Берется следующая свободная порция
- `guided` [, `chunk`]
 - Аналогично предыдущему, но размер порции уменьшается экспоненциально
- `runtime`
 - Определяется переменной окружения `OMP_SCHEDULE`

Директива SECTIONS

- Индивидуальный блок кода для потоков
- Поддерживаемые условия:
 - private
 - firstprivate
 - lastprivate
 - reduction
 - nowait



Синхронизация исполнения

Барьер

- Предположим мы выполняем следующий код:
for (i=0; i < N; i++)
 a[i] = b[i] + c[i];
for (i=0; i < N; i++)
 d[i] = a[i] + b[i];
- Если циклы выполнять параллельно, то может быть неправильный ответ
- Нужна синхронизация по доступу к a[i]

Barrier

- Каждый поток ждет, пока все потоки достигнут определенную точку:
 - `#pragma omp barrier`

Когда использовать барьеры?

- Когда изменение данных происходит асинхронно и целостность данных может быть под вопросом
- Примеры:
 - Между операциями чтения записи одного участка памяти
 - После каждого временного шага в решателе
- К сожалению барьеры могут привести к падению производительности и масштабируемости программы
- Следовательно использовать их надо с осторожностью

Критические секции

- Если `sum` разделяемая переменная, то цикл нельзя исполнять параллельно

```
for (i=0; i < N; i++){  
    .....  
    sum += a[i];  
    .....  
}
```

- Можно использовать критическую секцию:

```
for (i=0; i < N; i++){  
    .....  
    //one at a time can proceed  
    sum += a[i];  
    //next in line, please  
    .....  
}
```

Критическая секция

- Полезны для избавления от ошибок соревнования, чтения записи данных (неопределенный порядок)
- Может привести к тому, что параллельная программа станет последовательной
- Все потоки исполняют код, но не одновременно:
 - `#pragma omp critical [(name)]`
`{<code-block>}`
 - `#pragma omp atomic`
`<statement>`

Конструкции SINGLE и MASTER

- Только один поток из группы исполняет код
`#pragma omp single [clause[,] clause] ...]`
`{ <code-block> }`
- Только основной поток (мастер) исполняет код
`#pragma omp master`
`{<code-block>}`

Переменные окружения OpenMP

- OMP_NUM_THREADS n
- OMP_SCHEDULE “schedule,[chunk]”
- OMP_DYNAMIC { TRUE | FALSE }
- OMP_NESTED { TRUE | FALSE }

Среда выполнения OpenMP

- OpenMP предоставляет различные функции для:
 - Управления средой выполнения
 - Управления семафорами и блокировками
 - Вложенные блокировки возможны, но не рассматриваются
- Функции имеют выше приоритет, чем переменные окружения
- Рекомендуется использовать под управлением макроса `#ifdef for _OPENMP` (C/C++)
- В C/C++ необходимо включать `<omp.h>`

Список OpenMP функций

<code>omp_set_num_threads</code>	Установить количество потоков
<code>omp_get_num_threads</code>	Вернуть количество потоков в группе
<code>omp_get_max_threads</code>	Максимальное количество потоков
<code>omp_get_thread_num</code>	ID потока
<code>omp_get_num_procs</code>	Максимальное количество процессоров
<code>omp_in_parallel</code>	В параллельном регионе?
<code>omp_set_dynamic</code>	<i>Activate dynamic thread adjustment</i>
<code>omp_get_dynamic</code>	Check for dynamic thread adjustment
<code>omp_set_nested</code>	<i>Activate nested parallelism</i>
<code>omp_get_nested</code>	Check for nested parallelism
<code>omp_get_wtime</code>	Вернуть время
<code>omp_get_wtick</code>	Number of seconds between clock ticks

Функции блокировки в OpenMP

- Блокировки — более гибкий способ для управления критическими секциями:
 - Возможно реализовать асинхронное поведение
- Используются специальные переменные:
 - C/C++: тип `omp_lock_t` и `omp_nest_lock_t` для вложенных блокировок
- Можно управлять только через API
- Без инициализации переменных, поведение функций блокировок не определено

Вложенная блокировка

- Простая блокировка: нельзя блокировать дважды
- Вложенная блокировка: один поток может многократно блокировать переменную перед разблокированием
- Список функций аналогичен:

Simple locks

omp_init_lock

omp_destroy_lock

omp_set_lock

omp_unset_lock

omp_test_lock

Nestable locks

omp_init_nest_lock

omp_destroy_nest_lock

omp_set_nest_lock

omp_unset_nest_lock

omp_test_nest_lock

OpenMP и компиляторы

- OpenMP v2.5
 - Visual C++ 2005 (Professional and Team System editions)
 - Intel Parallel Studio
 - Sun Studio
 - Portland Group compilers
 - GCC since version 4.2.
- OpenMP v3.0
 - GCC 4.3.1
 - Nanos compiler
 - Intel Fortran and C/C++ versions 11.0 and 11.1 Compilers, and Intel Parallel Studio.
 - IBM XL C/C++ Compiler
 - Sun Studio 12 update 1

Сборка программы

- `gcc -fopenmp -o test test.c`
- `icc -openmp -o test test.c`

Выводы

- OpenMP - компактная, но мощная модель программирования систем с общей памятью
- OpenMP поддерживает Fortran, C\C++
- OpenMP переносимы на разные SMP системы
- OpenMP программа может исполняться и последовательно.