ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ

Занятие 1. Технология OpenMP

Составитель: Герасимов А.С.

Учебный кластер МФТИ

head.vdi.mipt.ru **remote.vdi.mipt.ru:52960**

ssh login@head.vdi.mipt.ru

- Узлы: 1 головной (head) и 7 вычислительных
- Узлы идентичны: 4 ядра, 15 ГБ ОЗУ
- Система очередей – Torque/PBS

Пример PBS-задачи

job.sh

```
#!/bin/bash

#PBS -l walltime=00:10:00,nodes=7:ppn=1
#PBS -N job_name
#PBS -q batch

uname -n
```

Запуск задачи

```
qsub job.sh
```

Выход задачи:

- <job_name>.o<ID> – выход stdout
- <job_name>.e<ID> – выход stderr

Ограничения:

- 5 заданий / пользователя
- 10 минут выполнения
- 1 ГБ памяти

Просмотр текущих задач в очереди

qstat

```
[kolya@head mpi]$ qstat
Job id          Name             User      Time Use S Queue
--------------- ---------------- --------- -------- - -----
25.localhost    my_job           kolya            0 R batch
26.localhost    my_job           kolya            0 R batch
27.localhost    my_job           kolya            0 R batch
28.localhost    my_job           kolya            0 R batch
29.localhost    my_job           kolya            0 R batch
```

Удаление задачи

qdel <ID>

OpenMP

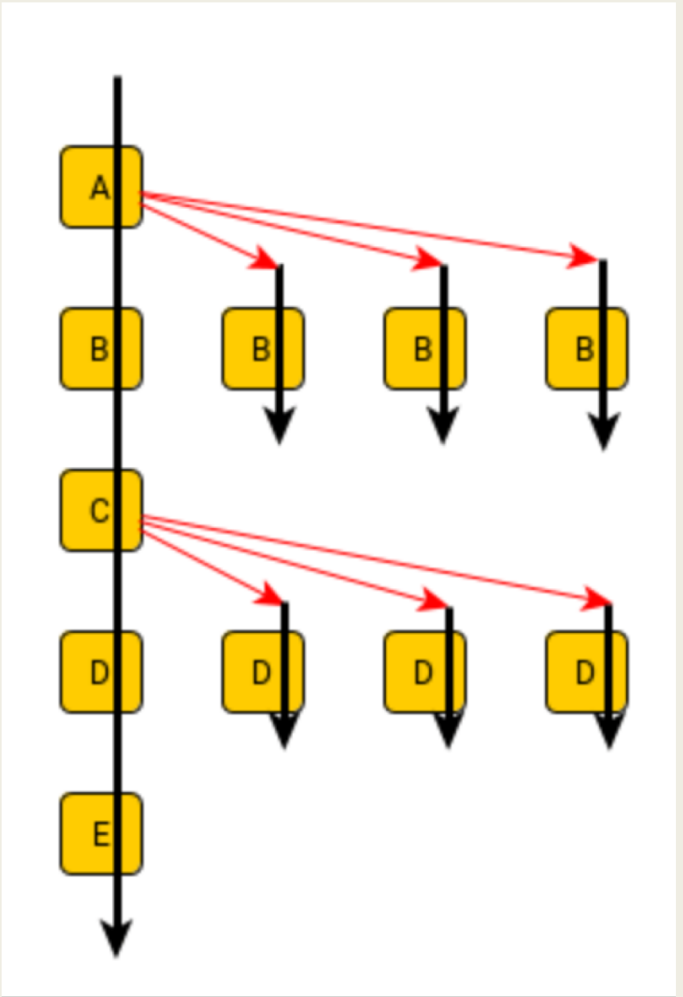
- Библиотека функций и директивы компилятора, предназначенные для написания параллельных программ
- Официально поддерживается *C, C++ и Фортран*, однако можно найти реализации для некоторых других языков, например *Паскаль* и *Java*.
- Поддерживается производителями аппаратуры (*Intel, HP, SGI, Sun, IBM*), разработчиками компиляторов (*Intel, Microsoft, KAI, PGI, PSR, APR, Absoft*)
- Ориентирован на системы с общей памятью
- Основной метод – создание потоков (модель fork-join)

Общие механизмы OpenMP. Инициализация

Основа программы

```
#include "omp.h"

int main(int argc, char** argv) {
// A - single thread
#pragma omp parallel
{
    // B - many threads
}
// C - single thread
}
```



Компиляция программы

```
gcc -fopenmp superhot.c -o hot
```

Запуск программы

```
OMP_NUM_THREADS=N ./hot
```

Общие механизмы OpenMP. Параллельные регионы

- Количество порождаемых потоков для параллельных областей контролируется через переменную окружения **OMP_NUM_THREADS**, а также может задаваться через вызов функции внутри программы.
- Каждый порожденный поток исполняет код в структурном блоке. По умолчанию синхронизация отсутствует и последовательность выполнения не определена.
- После выполнения параллельного участка все потоки, кроме основного, завершаются.
- Каждый поток имеет уникальный номер, который изменяется от 0 до количества потоков –1. Идентификатор потока может быть определен с помощью функции **omp_get_thread_num()**.

Общие механизмы OpenMP. Параллельные регионы

```
#pragma omp parallel
{
myid = omp_get_thread_num();
if(myid == 0)
    do_something();
else
    do_something_else(myid);
} ← sync
```

Общие механизмы OpenMP. Задача 1

- Составить и запустить программу «Hello, world!»
- Вывести размер своего потока
- Потоки распечатывают свои идентификаторы в обратном порядке

Общие механизмы OpenMP. Условия выполнения

```
#pragma omp parallel \
shared(var1, var2, ....) \
private(var1, var2, ...) \
firstprivate(var1, var2, ...) \
reduction(оператор:var1, var2, ...) \
if(выражение) \
default(shared|none)
{
  // parallel block
}
```

Общие механизмы OpenMP.Условия выполнения

- shared(var1, var2, ...)
  - *перечисленные переменные будут разделяться между потоками. Все потоки будут обращаться к одной и той же области памяти.*
- private(var1, var2, ...)
  - *каждый поток должен иметь свою копию переменной на всем протяжении своего исполнения.*
- firstprivate(var1, var2, ...)
  - *инициализируются при входе в параллельный участок кода значением, которое имела переменна до входа в параллельную секцию.*
- lastprivate(var1, var2, ...)
  - *сохраняют свое значение, которое они получили при достижении конца параллельного участка кода.*
- reduction(оператор:var1, var2, ...)
  - *гарантирует безопасное выполнение операций редукции, например, вычисление глобальной суммы.*
- if(выражение)
  - *параллельное выполнение необходимо только если выражение истинно.*
- default(shared|private|none)
  - *область видимости переменных внутри параллельного участка кода по умолчанию.*
- schedule(type[,chank])
  - *контролируется то, как итерации цикла распределяются между потоками.*

Общие механизмы OpenMP. Условия выполнения

```
#pragma omp parallel shared(a) private(myid, x)
{
myid = omp_get_thread_num();
x = work(myid);
if(x < 1.0)
    a[myid] = x;
}
```

```
#pragma omp parallel default(private) shared(a)
{
myid = omp_get_thread_num();
x = work(myid);
if(x < 1.0)
    a[myid] = x;
}
```

Общие механизмы OpenMP. Разделение работы

#pragma omp for [*условие* [,*условие*] ...]

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

Общие механизмы OpenMP. Разделение работы

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

```
#pragma omp parallel sections (nowait?)
{
#pragma omp section
{
    printf("T%d: foo\n", omp_get_thread_num());
}
#pragma omp section
{
    printf("T%d: bar\n", omp_get_thread_num());
}
} // omp sections ← sync
```

Общие механизмы OpenMP. Разделение работы

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

```
#pragma omp single
    printf(”Program finished!\n");
```

Общие механизмы OpenMP. Условия выполнения

If(clause)

```
#pragma omp parallel
{
#pragma omp for if(n>2000)
{
    for(i=0; i<n; i++)
        a[i] = work(i);
}
}
```

lastprivate(var [, var2 ...])

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

Общие механизмы OpenMP. Условия выполнения

reduction(op:var1 [, var2 ...])

```
#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:gsum)
   for(i=0; i<10000; i++)
      gmin = min(gmin, x[i]);
}
```

Общие механизмы OpenMP. Условия выполнения

reduction(op:var1 [, var2 ...])
C/C++ +, - , *, &, ^, |, &&, ||, min, max
Fortran +, -, *, .and., .or., .eqv., .neqv., min, max, iand, ior, ieor

Общие механизмы OpenMP. Условия выполнения

schedule(тип [, размер блока])

Размер блока задает размер каждого пакета на обработку потоком (количество итераций).

Типы расписания:

- static – итерации равномерно распределяются по потокам. Т.е. если в цикле 1000 итераций и 4 потока, то один поток обрабатывает все итерации с 1 по 250, второй – с 251 по 500, третий - с 501 по 750, четвертый с 751 по 1000. Если при этом задан еще и размер блока, то все итерации блоками заданного размера циклически распределяются между потоками.

Статическое распределение работы эффективно, когда время выполнения итераций равно, или приблизительно равно.

- dynamic – работа распределяется пакетами заданного размера (по умолчанию размер равен 1) между потоками.

Как только какой-либо из потоков заканчивает обработку своей порции данных, он захватывает следующую.

В отличии от статического планирования, выполняется многократно (во время выполнения программы).

При этом подходе накладные расходы несколько больше, но можно добиться лучшей балансировки загрузки между потоками.

Общие механизмы OpenMP. Условия выполнения

schedule(тип [, размер блока])

- **guided** – данный тип распределения работы аналогичен предыдущему, однако размер блока изменяется динамически в зависимости от того, сколько необработанных итераций осталось. Размер блока постепенно уменьшается вплоть до указанного значения.

Распределение начинается с некоторого начального размера, зависящего от реализации библиотеки.

При таком подходе можно достичь хорошей балансировки при меньших накладных расходах.

- **runtime** – тип распределения определяется в момент выполнения программы. Удобно в экспериментальных целях для выбора оптимального значения **типа** и **размера блока**.

Тип распределения работ зависит от переменной окружения **OMP_SCHEDULE**.

По умолчанию считается, что установлен статический метод распределения работ.

Общие механизмы OpenMP. Задача 2

- Составить параллельную программу, суммирующую все натуральные числа от 1 до N
- Каждый поток получает свой диапазон чисел для суммирования
- N задается аргументом запуска
- Использовать условия reduction, schedule