Open Multi-Processing

Кононова Анастасия, 6057/2

Высокопроизводительные вычисления, 2011

Содержание

- Open Multi-Processing
- The OpenMP ARB
- Преимущества OpenMP
- Модель «ветвление – объединение»
- Ключевые элементы OpenMP
- Пример программы с использование OpenMP
- OpenMP vs MPI

1. Open Multi-Processing

- открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран
- за основу берется последовательная программа
- совокупность директив компилятора, библиотечных процедур и переменных окружения
- предназначен для программирования многопоточных приложений на многопроцессорных системах с общей памятью

2. The OpenMP Architecture Review Board

- некоммерческая организация (собственник бренда OpenMP)
- спецификация, разработка, выпуск новых версий
- постоянные члены – заинтересованы в создании продуктов для OpenMP
- 1997г. – первая версия, предназначалась для языка Fortran
- 1998г. – первая версия для языков C/C++

2.1. Постоянные члены ARB

- AMD *(Dibyendu Das)*
- Cray *(James Beyer)*
- Fujitsu *(Matthijs van Waveren)*
- HP *(Uriel Schafer)*
- IBM *(Kelvin Li)*
- Intel *(Jay Hoeflinger)*
- NEC *(Kazuhiro Kusano)*
- The Portland Group, Inc. *(Michael Wolfe)*
- Oracle Corporation *(Nawal Copty)*
- Microsoft
- Texas Instruments *(Andy Fritsch)*
- CAPS-Entreprise *(Francois Bodin)*
- NVIDIA *(Yuan Lin)*

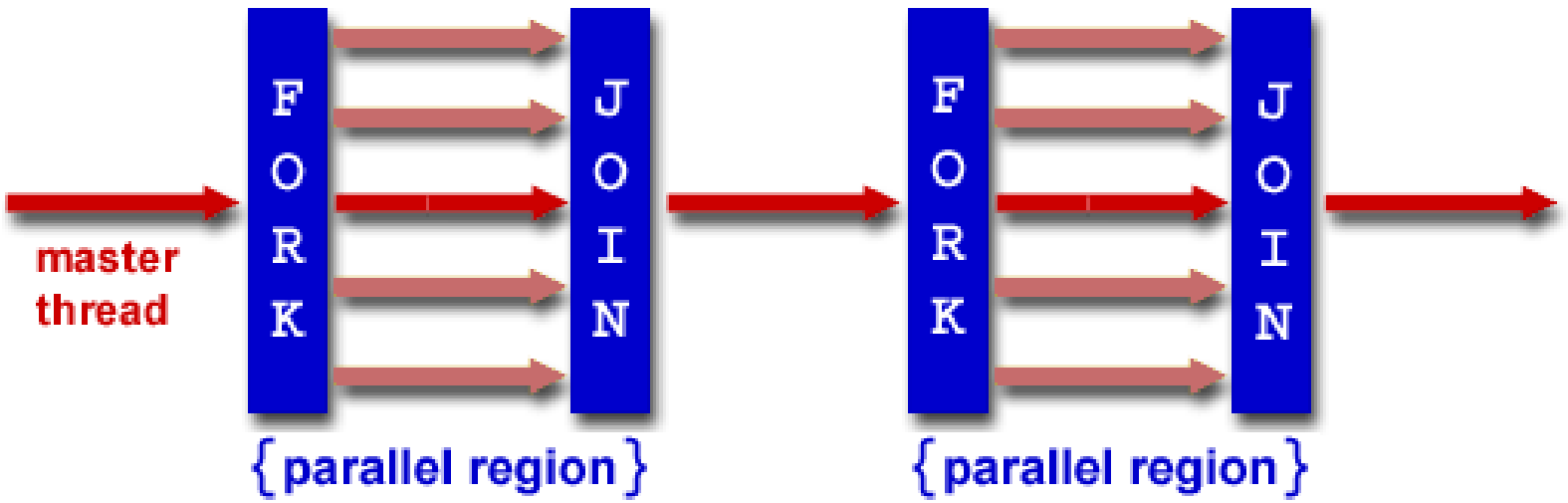
3. Преимущества OpenMP

- «инкрементное распараллеливание» - новая распараллеленная программа – это старая программа
- гибкий механизм – больше возможностей контроля поведения параллельного приложения
- нет необходимости поддерживать последовательную и параллельную версии
- поддержка так называемых оторванных директив «orphan»

4. Модель «ветвление – объединение»

- программа начинается как единственный процесс с главным потоком
- главный поток выполняется последовательно, пока не сталкиваются с областью параллельной конструкции
- fork (ветвление) – главный поток создает группу параллельных потоков.
- join (объединение) – после завершения в области параллельной конструкции, потоки синхронизируются и закрываются, оставляя только главный поток

4.1. Модель «ветвление – объединение»



5. Ключевые элементы OpenMP

- определение параллельной секции
  - порождение нитей
  - клаузы порождения нитей
- разделение работы
  - параллельные секции и циклы
  - клаузы параллельных циклов
  - исполнение одной нитью
- синхронизация
- переменные окружения
- основные функции

5.1. Порождение нитей

```
#pragma omp parallel [clause]
```

- порождаются новые N нитей
- нить с номером 0 становиться master thread
- при выходе из параллельной области master thread ждет завершения работы остальных и продолжает выполнение в одном экземпляре
- по-умолчанию, код внутри параллельной области выполняется всеми нитями одинаково
- распределение работ можно менять с помощью директив DO, SECTIONS и SINGLE

5.2. Клаузы порождения нитей

- **`if`** (условие) – выполнение параллельной секции по условию
- **`private`** (list) – список переменных локальных в каждой нити
- **`shared`** (list) – список переменных, общих для каждой нити
- **`firstprivate`** (list) – список переменных, которые становятся локальными в каждой нити со значениями, ранее присвоенными этим переменным
- **`reduction`** (operator:list) – список переменных, с которыми выполняются операции обобщенно по всем нитям

5.3. Параллельные циклы и секции

```
#pragma omp for [clause …]
#pragma omp sections [clause …]
```

- секции отделяются друг от друга директивой section

5.4. Клаузы параллельных циклов и секций

- **schedule** (type[,chink]) – способ распределения итераций по нитям
- **ordered** – последовательное выполнение витков цикла
- **private** (list)
- **shared** (list)
- **firstprivate** (list)
- **lastprivate** (list) - переменным присваивается результат последнего витка цикла
- **reduction** (operator:list)
- **nowait** – запрет неявной синхронизации

5.5. Исполнение одной нитью

```
#pragma omp single [clause …]
```

- исполняется первой нитью, дошедшей до блока

5.6. Синхронизация

- `#pragma omp master` – блок кода будет выполнен только master thread
- `#pragma omp critical[name]` – критическая секция, выполняется только одним потоком
- `#pragma omp barrier` – точка барьерной синхронизации: каждая нить дожидается остальных

5.7. Переменные окружения

- **shared** – общие переменные
- **private** – приватные переменные (отдельный экземпляр переменной для каждой нити)
- **threadprivate** – применяется к COMMON-блокам, которые необходимо сделать приватными
- **firstprivate** – приватные копии переменной при входе в параллельную область инициализируются значением оригинальной переменной
- **lastprivate** – по окончании параллельно цикла, нить, которая выполнила последнюю итерацию цикла, обновляет значение оригинальной переменной
- **reduction** – переменная, с которой в цикле производится reduction-операция
- **copyin** – применяется к COMMON-блокам. При входе в параллельную область приватные копии этих данных инициализируются оригинальными значениями

5.8. Функции

- `omp_set_num_threads(int num)` – максимальное число нитей для использования в следующей параллельной области
- `omp_get_max_threads()` – возвращает максимальное число нитей
- `omp_get_num_threads()` – возвращает фактическое число нитей в параллельной области программы
- `omp_get_num_procs()` – возвращает число процессов, доступных приложению
- `omp_in_parallel()` – возвращает true если вызвана из параллельной области программы
- `omp_set_dunamic(int dynamic_threads)` – устанавливает значение флага, позволяющего динамически менять число нитей
- `omp_get_dinamic()` – возвращает значение этого флага

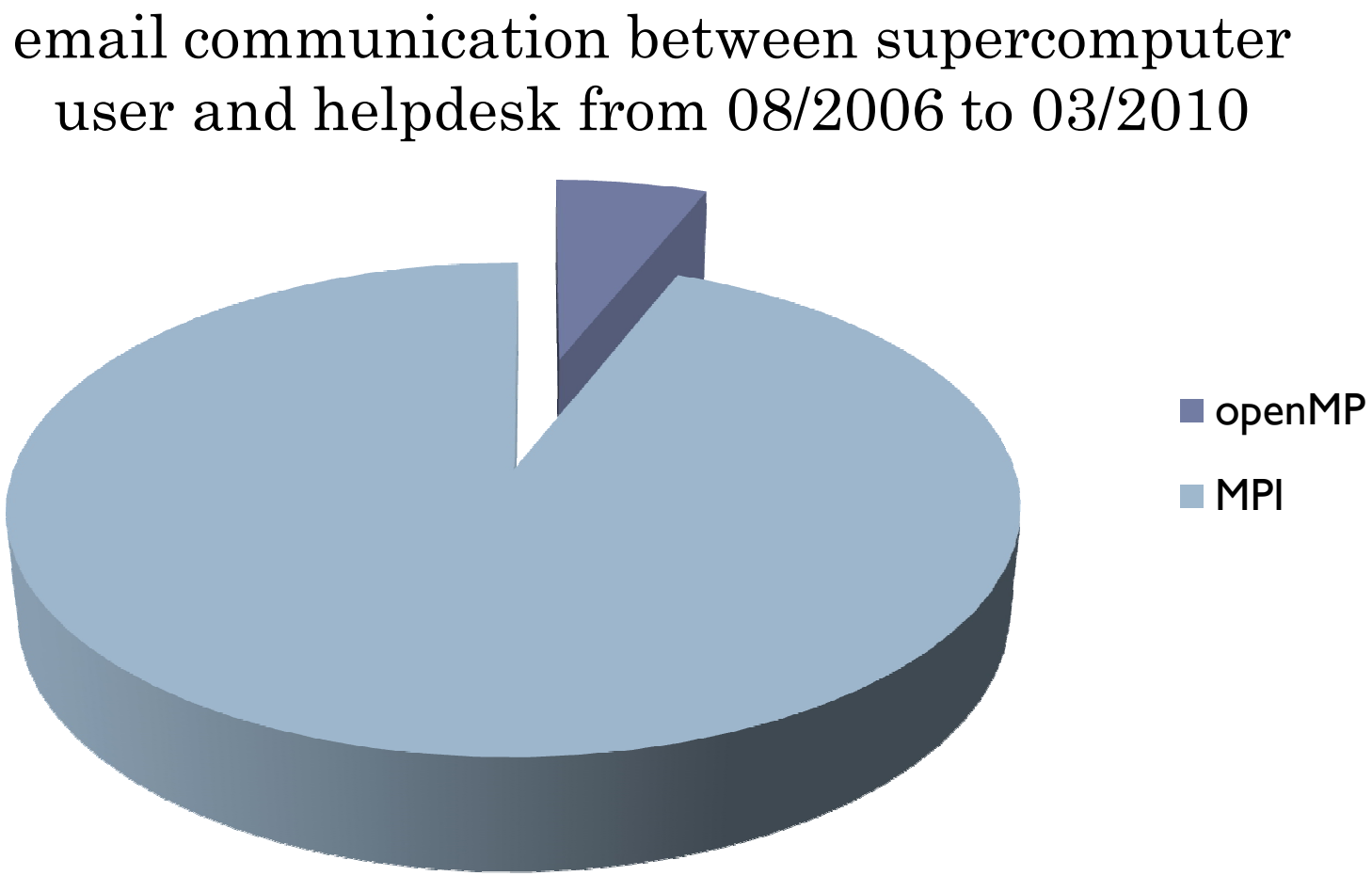
6. Вычисление точного значения числа пи с помощью интеграла

```
# pragma omp parallel {
      # pragma omp single {
            num_threads = omp_get_num_threads();
      }
      # pragma omp for nowait reduction(+:result)
            for(i = 1; i < n+1; i++)
                  result += 1/(1+pow(((i - 0.5)/n),2));
      }
result *= 4/(double)n;
```

6.1. Сравнение результатов

n	100		1000		10000	
кол-во потоков	1	2	1	2	1	2
результат	3.141601	3.141601	3.141593	3.141593	3.141593	3.141593
разница	8.333333e-06	8.333333e-06	8.333333e-08	8.333333e-08	8.333410e-10	8.333410e-10
время (сек)	4e-06	7.8e-05	3.5e-05	9e-05	3.4e-04	1.8e-04

7. OpenMP vs MPI



7.1. OpenMP [за : против]

за

- проще программировать и отлаживать, чем MPI
- директивы могут добавляться постепенно
- программа может работать как последовательная
- не нужно изменять исходный код программы
- код проще для понимания и поддержки

против

- может работать только на компьютерах с общей памятью
- требует компилятор, поддерживающий OpenMP
- обычно используется для распараллеливания циклов

7.2. MPI [за : против]

за

- работает на компьютерах как с общей, так и с разделенной памятью
- может использоваться для решения более широкого круга проблем, чем OpenMP
- каждый процесс имеет свои собственные локальные переменные

против

- требуется много изменений последовательного кода
- сложно отлаживать программы
- производительность ограничена связывающей сетью между узлами

Список литературы и источников

- http://openmp.org/wp/ - официальный сайт
- http://www.intuit.ru/department/se/openmp/
- http://parallel.ru/tech/tech_dev/openmp.html
- http://www.dartmouth.edu/~rc/classes/intro_mpi/parallel_prog_compare.html - Pros and Cons of OpenMP/MPI
- Баденко И. Л. Высокопроизводительные вычисления: учебное пособие. - Санкт-Петербург, Издательство Политехнического университета, 2010г