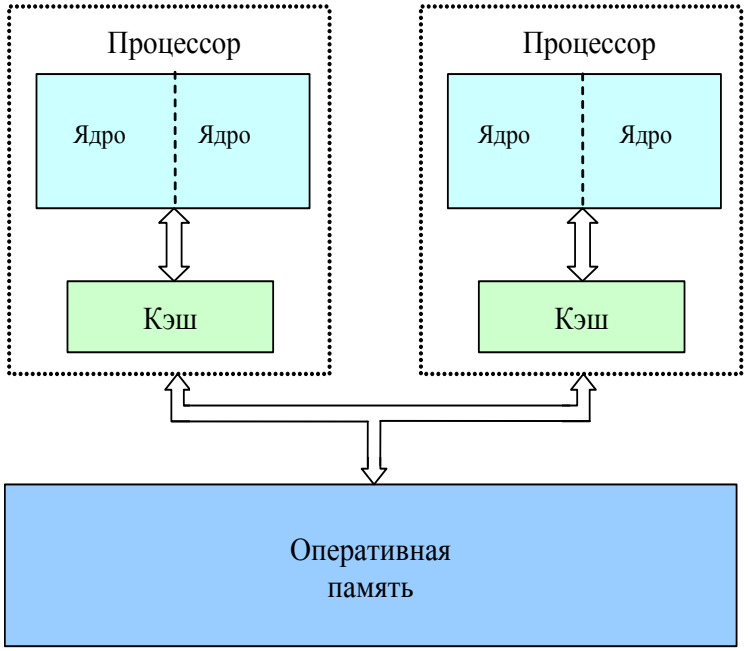
Содержание

- Обзор технологии OpenMP
- Директивы OpenMP
  - Формат, области видимости, типы
  - Определение параллельной области
  - Управление областью видимости данных
  - Распределение вычислений между потоками
  - Операция редукции
  - Синхронизация
  - Совместимость директив и их параметров
- Библиотека функций OpenMP
- Переменные окружения

Обзор технологии OpenMP

- 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*)

Обзор технологии OpenMP

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 (2002)
- OpenMP C/C++, Fortran API v2.5 (2005)
- OpenMP C/C++, Fortran API v3.0 (2008)

- Разработкой стандарта занимается организация OpenMP Architecture Review Board, в которую вошли представители крупнейших компаний - разработчиков SMP-архитектур и программного обеспечения.

Обзор технологии OpenMP

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

Обзор технологии OpenMP

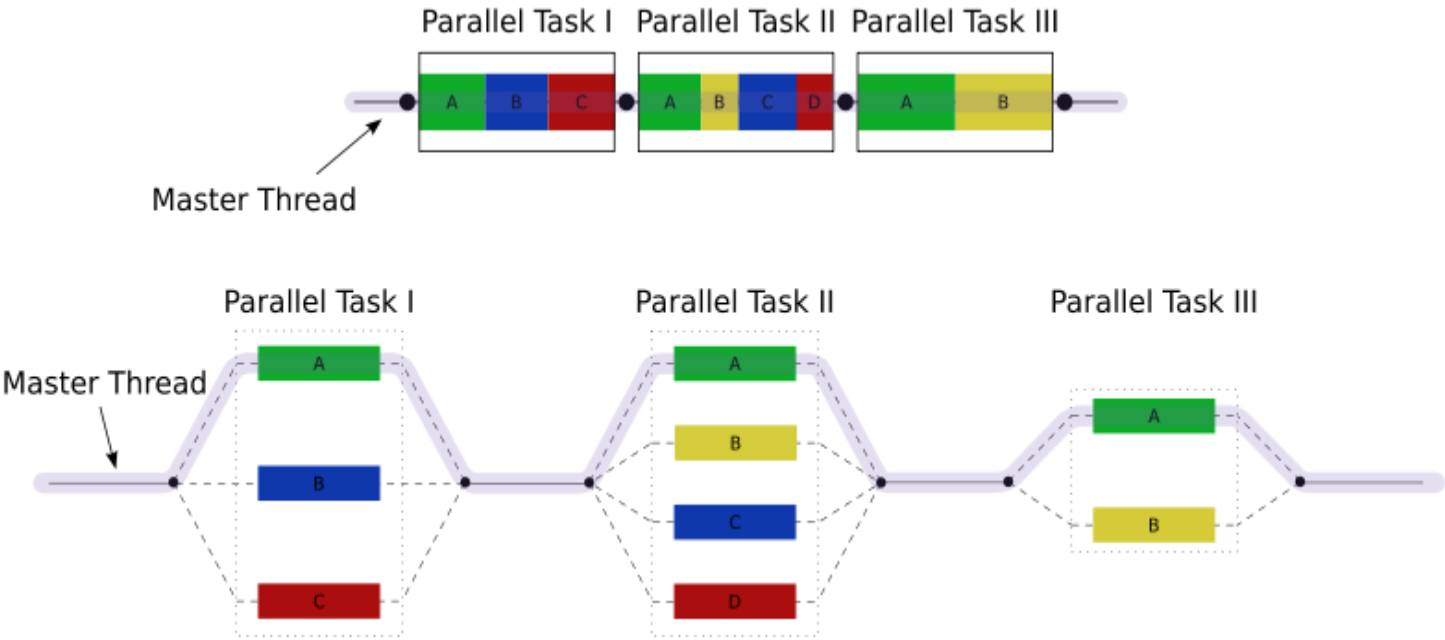
Положительные стороны

- Поэтапное (инкрементальное) распараллеливание
  - Можно распараллеливать последовательные программы поэтапно, не меняя их структуру
- Единственность разрабатываемого кода
  - Нет необходимости поддерживать одновременно последовательный и параллельный вариант программы, поскольку директивы игнорируются обычными компиляторами (в общем случае)
- Эффективность
  - Учет и использование возможностей систем с общей памятью
- Переносимость
  - Поддержка большим числом компиляторов под разные платформы и ОС, стандарт для распространенных языков C/C++, Fortran

Обзор технологии OpenMP

Принципы организации параллелизма

- Использование потоков (общее адресное пространство)
- Пульсирующий (fork-join) параллелизм



* Источник: http://en.wikipedia.org/wiki/OpenMP

Обзор технологии OpenMP

Принципы организации параллелизма

- При выполнении обычного кода (вне параллельных областей) программа исполняется 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)

Обзор технологии OpenMP

Компиляторы

- Список на http://openmp.org/wp/openmp-compilers/

- Версию 3.0 поддерживают:
  - gcc с версии 4.4
  - IBM XL C/C++ V10.1, IBM XL Fortran V12.1
  - Sun Studio Express 7.08 Compilers
- Версию 2.5 поддерживают:
  - Intel C/C++, Visual Fortran Compilers 10.1
  - PathScale Compiler Suite
- Версию 2.0 поддерживают:
  - MS VS 2005, 2008

Обзор технологии OpenMP

Структура

- Набор директив компилятора
- Библиотека функций
- Набор переменных окружения

- Изложение материала будет проводиться на примере C/C++

Директивы OpenMP

Формат записи директив

- Формат

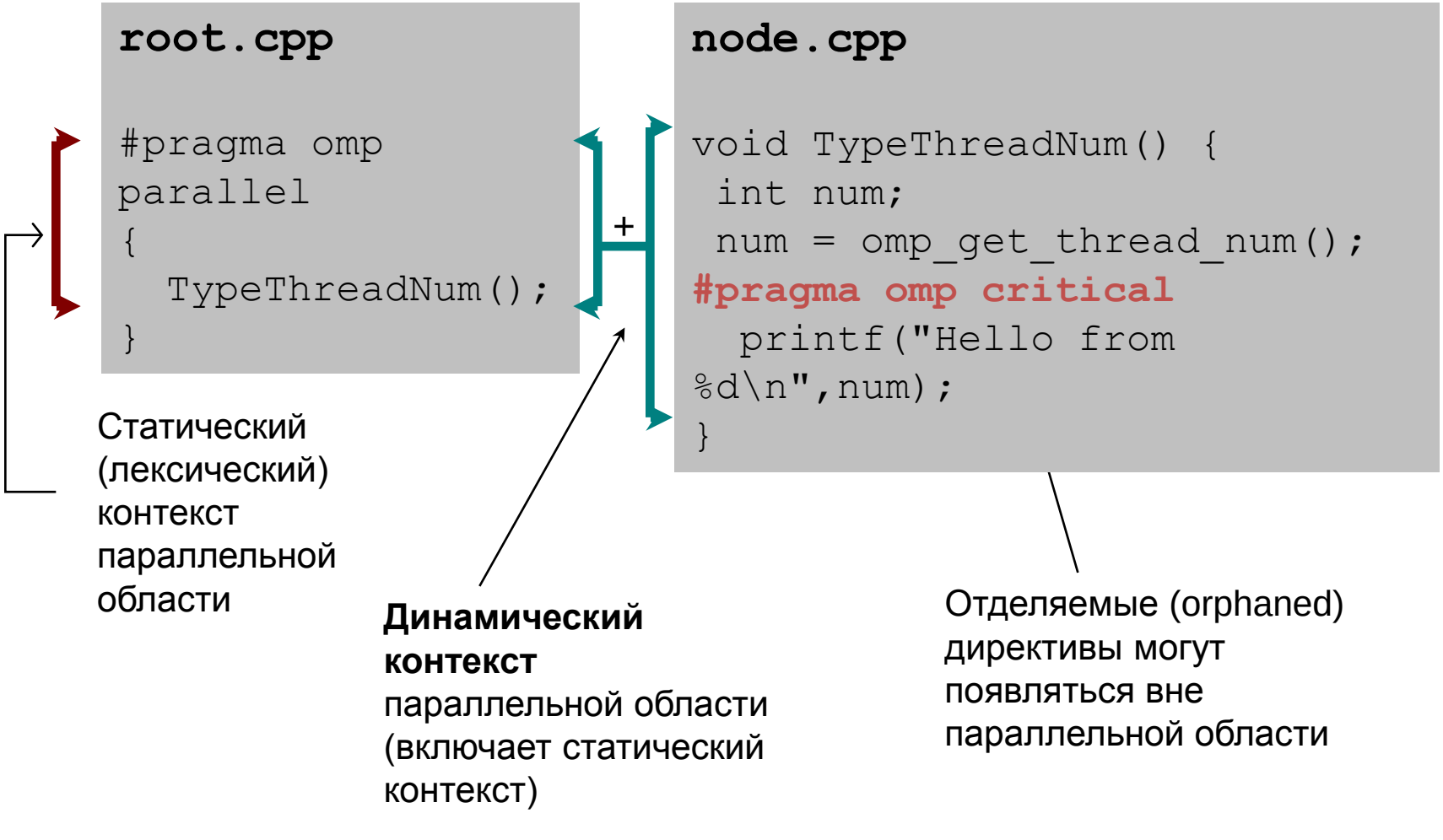
```
#pragma omp имя_директивы [clause,…]
```

- Пример

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

Директивы OpenMP

Формат записи директив



Директивы OpenMP

Типы директив

- Определение параллельной области
- Разделение работы
- Синхронизация

Директивы OpenMP

Определение параллельной области

- Директива **parallel** (основная директива OpenMP)
- Когда основной поток выполнения достигает директиву parallel, создается набор (team) 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

Директивы OpenMP

Определение параллельной области

- Формат директивы parallel

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

- Возможные параметры (clause)

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

Директивы OpenMP

Определение параллельной области

- Количество потоков (по убыванию старшинства)
  - num_threads(N)
  - omp_set_num_threads()
  - OMP_NUM_THREADS
  - Число, равное количеству процессоров, которое “видит” операционная система
- Параметр (clause) if – если условие в if не выполняется, то процессы не создаются

Директивы OpenMP

Определение параллельной области

- Пример использования директивы **parallel**

```
#include <omp.h>
main () {
  int nthreads, tid;
  // Создание параллельной области
  #pragma omp parallel private(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
}
```

Директивы OpenMP

Управление областью видимости данных

- Управление областью видимости обеспечивается при помощи параметров (clause) директив

  private, firstprivate, lastprivate, shared, default,

  reduction, copyin

  которые определяют, какие соотношения существуют между переменными последовательных и параллельных фрагментов выполняемой программы

Директивы OpenMP

Управление областью видимости данных

- Параметр **shared** определяет список переменных, которые будут общими для всех потоков параллельной области; правильность использования таких переменных должна обеспечиваться программистом

```
#pragma omp parallel shared(list)
```

- Параметр **private** определяет список переменных, которые будут локальными для каждого потока; переменные создаются в момент формирования потоков параллельной области; начальное значение переменных является неопределенным

```
#pragma omp parallel private(list)
```

Директивы OpenMP

Определение параллельной области

- Пример использования директивы **parallel**

```
#include <omp.h>
main () {
  int nthreads, tid;
  // Создание параллельной области
  #pragma omp parallel private(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
}
```

Директивы OpenMP

Управление областью видимости данных

- Параметр **firstprivate** позволяет создать локальные переменные потоков, которые перед использованием инициализируются значениями исходных переменных

```
#pragma omp parallel firstprivate(list)
```

- Параметр **lastprivate** позволяет создать локальные переменные потоков, значения которых запоминаются в исходных переменных после завершения параллельной области (используются значения потока, выполнившего последнюю итерацию цикла или последнюю секцию)

```
#pragma omp parallel lastprivate(list)
```

Директивы OpenMP

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

Директивы OpenMP

Распределение вычислений между потоками

- Формат директивы **for**

```
#pragma omp for [clause ...] newline
for loop
```

- Возможные параметры (clause)

```
private(list)
firstprivate(list)
lastprivate(list)
reduction(operator: list)
ordered
schedule(kind[, chunk_size])
nowait
```

Директивы OpenMP

Распределение вычислений между потоками

- Распределение итераций в директиве **for** регулируется параметром (clause) **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 уменьшается 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)

Директивы OpenMP

Распределение вычислений между потоками

- Пример использования директивы **for**

```
#include <omp.h>
#define CHUNK 100
#define NMAX 1000
main () {
  int i, n, chunk;
  float a[NMAX], b[NMAX], c[NMAX];
  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
}
```

Директивы OpenMP

Распределение вычислений между потоками

- Формат директивы **sections**

```
#pragma omp sections [clause ...] newline
{
  #pragma omp section newline
    structured_block
  #pragma omp section newline
    structured_block
}
```

- Возможные параметры (clause)

```
private(list)
firstprivate(list)
lastprivate(list)
reduction(operator: list)
nowait
```

Директивы OpenMP

Распределение вычислений между потоками

- 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

Директивы OpenMP

Распределение вычислений между потоками

- Пример использования директивы **sections**

```
#include <omp.h>
#define NMAX 1000
main () {
  int i, n;
  float a[NMAX], b[NMAX], c[NMAX];
  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
}
```

Директивы OpenMP

Распределение вычислений между потоками

- Объединение директив **parallel** и **for/sections**

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

Директивы OpenMP

Операция редукции

- Параметр **reduction** определяет список переменных, для которых выполняется операция редукции
  - перед выполнением параллельной области для каждого потока создаются копии этих переменных,
  - потоки формируют значения в своих локальных переменных
  - при завершении параллельной области на всеми локальными значениями выполняются необходимые операции редукции, результаты которых запоминаются в исходных (глобальных) переменных

```
reduction (operator: list)
```

Директивы OpenMP

Операция редукции

- Пример использования параметра **reduction**

```
#include <omp.h>
main () { // vector dot product
  int i, n, chunk;
  float a[100], b[100], result;
  n = 100; chunk = 10;
  result = 0.0;
  for (i=0; i < n; i++) {
    a[i] = i * 1.0; b[i] = i * 2.0;
  }
  #pragma omp parallel for default(shared) \
    schedule(static,chunk) reduction(+:result)
  for (i=0; i < n; i++)
    result = result + (a[i] * b[i]);
  printf("Final result= %f\n",result);
}
```

Директивы OpenMP

Операция редукции

- Правила записи параметра **reduction**
  - Возможный формат записи выражения

    x = x op expr

    x = expr op x

    x binop = expr

    x++, ++x, x--, --x
  - x должна быть скалярной переменной
  - expr не должно ссылаться на x
  - op (operator) должна быть неперегруженной операцией вида

    +, -, *, /, &, ^, |, &&, ||
  - binop должна быть неперегруженной операцией вида

    +, -, *, /, &, ^, |

Директивы OpenMP

Синхронизация

- Директива **master** определяет фрагмент кода, который должен быть выполнен только основным потоком; все остальные потоки пропускают данный фрагмент кода (завершение директивы по умолчанию не синхронизируется)

```
#pragma omp master newline
  structured_block
```

- Директива **single** определяет фрагмент кода, который должен быть выполнен только одним потоком (любым)

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

Директивы OpenMP

Синхронизация

- Формат директивы **single**

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

- Возможные параметры (clause)

```
private(list)
firstprivate(list)
copyprivate(list)
nowait
```

- Один поток исполняет блок в **single**, остальные потоки приостанавливаются до завершения выполнения блока

Директивы OpenMP

Синхронизация

- Директива **critical** определяет фрагмент кода, который должен выполняться только одним потоком в каждый текущий момент времени (критическая секция)

```
#pragma omp critical [name] newline
  structured_block
```

Директивы OpenMP

Синхронизация

- Пример использования директивы **critical**

```
#include <omp.h>
main() {
  int x;
  x = 0;
  #pragma omp parallel shared(x)
  {
    #pragma omp critical
      x = x + 1;
  } // end of parallel section
}
```

Директивы OpenMP

Синхронизация

- Директива **barrier** – определяет точку синхронизации, которую должны достигнуть все процессы для продолжения вычислений (директива должны быть вложена в блок)

```
#pragma omp barrier newline
```

Директивы OpenMP

Синхронизация

- Директива **atomic** – определяет переменную, доступ к которой (чтение/запись) должна быть выполнена как неделимая операция

```
#pragma omp atomic newline
  statement_expression
```

- Возможный формат записи выражения

  x binop = expr , x++, ++x, x--, --x

  - x должна быть скалярной переменной

  - expr не должно ссылаться на x

  - binop должна быть неперегруженной операцией вида

  +, -, *, /, &, ^, |, >>, <<

Директивы OpenMP

Синхронизация

- Директива **flush** – определяет точку синхронизации, в которой системой должно быть обеспечено единое для всех процессов состояние памяти (т.е. если потоком какое-либо значение извлекалось из памяти для модификации, измененное значение обязательно должно быть записано в общую память)

```
#pragma omp flush (list) newline
```

- Если указан список list, то восстанавливаются только указанные переменные
- Директива **flush** неявным образом присутствует в директивах **barrier**, **critical**, **ordered**, **parallel**, **for**, **sections**, **single**

Директивы OpenMP

Совместимость директив и их параметров

Clause	Directive					
	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS
IF	●				●	●
PRIVATE	●	●	●	●	●	●
SHARED	●	●			●	●
DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

Библиотека функций OpenMP

```
void omp_set_num_threads(int num_threads)
```

- Позволяет назначить максимальное число потоков для использования в следующей параллельной области (если это число разрешено менять динамически). Вызывается из последовательной области программы

```
int omp_get_max_threads(void)
```

- Возвращает максимальное число потоков

```
int omp_get_num_threads(void)
```

- Возвращает фактическое число потоков в параллельной области программы

Библиотека функций OpenMP

```
int omp_get_thread_num(void)
```

- Возвращает номер потока

```
int omp_get_num_procs(void)
```

- Возвращает число процессоров, доступных приложению

```
int omp_in_parallel(void)
```

- Возвращает true, если вызвана из параллельной области программы

Библиотека функций OpenMP

Функции синхронизации

- В качестве замков используются общие переменные типа omp_lock_t. Данные переменные должны использоваться только как параметры примитивов синхронизации.

```
void omp_init_lock(omp_lock_t *lock)
```

- Инициализирует замок, связанный с переменной lock

```
void omp_destroy_lock(omp_lock_t *lock)
```

- Удаляет замок, связанный с переменной lock

Библиотека функций OpenMP

Функции синхронизации

```
void omp_set_lock(omp_lock_t *lock)
```

- Заставляет вызвавший поток дождаться освобождения замка, а затем захватывает его

```
void omp_unset_lock(omp_lock_t *lock)
```

- Освобождает замок, если он был захвачен потоком ранее

```
int omp_test_lock(omp_lock_t *lock)
```

- Пробует захватить указанный замок. Если это невозможно, возвращает false

Переменные окружения

- **OMP_SCHEDULE** – определяет способ распределения итераций в цикле, если в директиве for использована клауза **schedule**(runtime)
- **OMP_NUM_THREADS** – определяет число нитей для исполнения параллельных областей приложения
- **OMP_DYNAMIC** – разрешает или запрещает динамическое изменение числа нитей
- **OMP_NESTED** – разрешает или запрещает вложенный параллелизм
- Компилятор с поддержкой OpenMP определяет макрос "**_OPENMP**", который может использоваться для условной компиляции отдельных блоков, характерных для параллельной версии программы

Заключение

- Данная лекция посвящен рассмотрению методов параллельного программирования для вычислительных систем с общей памятью с использованием технологии OpenMP.
- В лекции проводится обзор технологии OpenMP
  - Рассматриваются директивы OpenMP позволяющие
    - Определять параллельные области
    - Управлять областью видимости данных
    - Распределять вычислений между потоками
    - Выполнять операции редукции
    - Осуществлять операции синхронизации
  - Дается обзор библиотечных функций OpenMP и используемых переменные окружения

Вопросы для обсуждения …

- Какие компьютерные платформы относятся к числу вычислительных систем с общей памятью?
- Какие подходы используются для разработки параллельных программ?
- В чем состоят основы технологии OpenMP?
- В чем состоит важность стандартизации средств разработки параллельных программ?
- В чем состоят основные преимущества технологии OpenMP?
- Что понимается под параллельной программой в рамках технологии OpenMP?
- Что понимается под понятием потока (thread)?
- Какие возникают проблемы при использовании общих данных в параллельно выполняемых потоках?
- Какой формат записи директив OpenMP?
- В чем состоит назначение директивы **parallel**?
- В чем состоит понятие фрагмента, области и секции параллельной программы?
- Какой минимальный набор директив OpenMP позволяет начать разработку параллельных программ?

Вопросы для обсуждения …

- Как определить время выполнения OpenMP программы?
- Как осуществляется распараллеливание циклов в OpenMP? Какие условия должны выполняться, чтобы циклы могли быть распараллелены?
- Какие возможности имеются в OpenMP для управления распределением итераций циклов между потоками?
- Как определяется порядок выполнения итераций в распараллеливаемых циклах в OpenMP?
- Какие правила синхронизации вычислений в распараллеливаемых циклах в OpenMP?
- Как можно ограничить распараллеливание фрагментов программного кода с невысокой вычислительной сложностью?
- Как определяются общие и локальные переменные потоков?
- Что понимается под операцией редукции?
- Какие способы организации взаимоисключения могут быть использованы в OpenMP?
- Что понимается под атомарной (неделимой) операцией?
- Как определяется критическая секция?

Вопросы для обсуждения

- Какие операции имеются в OpenMP для переменных семафорного типа (замков)?
- Как осуществляется в OpenMP распараллеливание по задачам (директива **sections**)?
- Как определяются однопотоковые участки параллельных фрагментов (директивы **single** и **master**)?
- Как осуществляется синхронизация состояния памяти (директива **flush**)?
- Как используются постоянные локальные переменные потоков (директивы **threadprivate** и **copyin**)?
- Какие средства имеются в OpenMP для управления количеством создаваемых потоков?
- Что понимается под динамическим режимом создания потоков?
- Как осуществляется управление вложенностью параллельных фрагментов?
- В чем состоят особенности разработки параллельных программ с использованием OpenMP на алгоритмическом языке Fortran?
- Как обеспечивается единственность программного кода для последовательного и параллельного вариантов программы?
- Какие компиляторы обеспечивают поддержку технологии OpenMP?

Обзор литературы…

- В наиболее полном виде информация по параллельному программированию для вычислительных систем с общей памятью с использованием OpenMP содержится в.
  - Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Melon, R.. Parallel Programming in OpenMP. San-Francisco, CA: Morgan Kaufmann Publishers., 2000
- Краткое описание OpenMP приводится в
  - Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.
  - Немнюгин С., Стесик О. Параллельное программирование для многопроцессорных вычислительных систем – СПб.: БХВ-Петербург, 2002.
- Полезная информация представлена также в
  - Антонов А.С. "Параллельное программирование с использованием технологии OpenMP: Учебное пособие".- М.: Изд-во МГУ, 2009
  - Kumar V., Grama A., Gupta A., Karypis G. Introduction to Parallel Computing , Inc. 1994 (2th edition, 2003)
  - Quinn M.J. Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill, 2004

Темы заданий для самостоятельной работы …

- Разработайте программу для нахождения минимального (максимального) значения среди элементов вектора.
- Разработайте программу для вычисления скалярного произведения двух векторов.
- Разработайте программу для задачи вычисления определенного интеграла с использованием метода прямоугольников

$$y = \int_a^b f(x)dx \approx h\sum_{i=0}^{N-1} f_i,\ f_i = f(x_i),\ x_i = i\,h,\ h = (b-a)/N$$

- Разработайте программу решения задачи поиска максимального значения среди минимальных элементов строк матрицы (такая задача имеет место для решения матричных игр)

$$y = \max_{1 \le i \le N} \min_{1 \le j \le N} a_{ij}$$

Темы заданий для самостоятельной работы…

- Разработайте программу для задачи 4 при использовании матриц специального типа (ленточных, треугольных и т.п.). Определите время выполнения программы и оцените получаемое ускорение. Выполните вычислительные эксперименты при разных правилах распределения итераций между потоками и сравните эффективность параллельных вычислений.
- Реализуйте операцию редукции с использованием разных способов организации взаимоисключения (атомарные операции, критические секции, синхронизацию при помощи замков). Оцените эффективность разных подходов. Сравните полученные результаты с быстродействием операции редукции, выполняемой посредством параметра *reduction* директивы **for**.
- Разработайте программу для вычисления скалярного произведения для последовательного набора векторов (исходные данные можно подготовить заранее в отдельном файле). Ввод векторов и вычисление их произведения следует организовать как две раздельные задачи, для распараллеливания которых используйте директиву **sections**.

Темы заданий для самостоятельной работы

- Выполните вычислительные экспериментами с ранее разработанными программами при различном количестве потоков (меньше, равно или больше числа имеющихся вычислительных элементов). Определите время выполнения программ и оцените получаемое ускорение.
- Уточните, поддерживает ли используемый Вами компилятор вложенные параллельные фрагменты. При наличии такой поддержки разработайте программы с использованием и без использования вложенного параллелизма. Выполните вычислительные эксперименты и оцените эффективность разных подходов.
- Разработке программу для задачи 4 с использованием распараллеливания циклов разного уровня вложенности. Выполните вычислительные эксперименты и сравните полученные результаты. Оцените величину накладных расходов на создание и завершение потоков.

Обзор литературы

- Достаточно много информации о технологии OpenMP содержится в сети Интернет.
  - Так, могут быть рекомендованы информационно-аналитический портал www.parallel.ru и, конечно же, ресурс www.openmp.org .
- Дополнительная информация по разработке многопоточных программ содержится в
  - Вильямс А. Системное программирование в Windows 2000 для профессионалов. - СПб.: БХВ-Петербург, 2001. (для ОС Windows)
  - Butenhof, D. R. (1007) Programming with POSIX Threads. Boston, MA: Addison-Wesley Professional.,1997 (стандарт POSIX Threads).
- Для рассмотрения общих вопросов параллельного программирования для вычислительных систем с общей памятью может быть рекомендована работа
  - Andrews, G. R. Foundations of Multithreaded, Parallel, and Distributed Programming.. – Reading, MA: Addison-Wesley, 2000 (русский перевод Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. – М.: Издательский дом "Вильямс", 2003)

Следующая тема

- Параллельные методы умножения матрицы на вектор

О проекте

Целью проекта является создание национальной системы подготовки высококвалифицированных кадров в области суперкомпьютерных технологий и специализированного программного обеспечения.

Задачами по проекту являются:

Задача 1. Создание сети научно-образовательных центров суперкомпьютерных технологий (НОЦ СКТ).

Задача 2. Разработка учебно-методического обеспечения системы подготовки, переподготовки и повышения квалификации кадров в области суперкомпьютерных технологий.

Задача 3. Реализация образовательных программ подготовки, переподготовки и повышения квалификации кадров в области суперкомпьютерных технологий.

Задача 4. Развитие интеграции фундаментальных и прикладных исследований и образования в области суперкомпьютерных технологий. Обеспечение взаимодействия с РАН, промышленностью, бизнесом.

Задача 5. Расширение международного сотрудничества в создании системы суперкомпьютерного образования.

Задача 6. Разработка и реализации системы информационного обеспечения общества о достижениях в области суперкомпьютерных технологий.

См. http://www.hpc-education.ru