Введение в параллельные вычисления

Лекция 3. OpenMP (часть 1)

КС-40, КС-44
РХТУ

Преподаватель
Митричев Иван Игоревич, к.т.н.

2017

Выбор модели программирования

Системы с общей памятью
- C++11 threads
- **OpenMP**

Системы с распределенной памятью
- MPI = message passing interface

Гетерогенные системы (графические процессоры)
- CUDA

Некоторые материалы по теме

- **OpenMP. Blaise Barney, Lawrence Livermore National Laboratory**
  https://computing.llnl.gov/tutorials/open_MP/

- The OpenMP® API specification for parallel programming
  http://openmp.org

- Примеры из учебника «Технологии параллельного программирования MPI и OpenMP»
  http://parallel.ru/tech/tech_dev/MPI%26_OpenMP/examples

- **Онлайн-компилятор**

http://coliru.stacked-crooked.com/

Open MultiProcessing

OpenMP (Open Multi-Processing) – открытый стандарт для распараллеливания программ на языках Си, Си++ и Фортран. Дает описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточны приложений на многопроцессорных системах с общей памятью.

Интерфейс OpenMP задуман как стандарт для программирования на масштабируемых SMP-системах
(SSMP, ccNUMA, etc.) в модели общей памяти (shared memory model).

В стандарт OpenMP входят спецификации набора директив компилятора, процедур и переменных среды.

- Открытый стандарт для распараллеливания программ на языках C, C++ и Фортран
- Очень важен в области High Performance Computing (HPC).
- Требует поддержки со стороны компилятора.
- Одна программа для последовательного компьютера (отладка) и параллельного.
- Инкрементальное распараллеливание (циклы).

Современный SMP

HP 9000 (Exemplar)

Производитель	Hewlett-Packard, подразделение высокопроизводительных систем.
Класс	Многопроцессорные сервера с общей памятью (SMP).
Предшественники	SMP/NUMA-системы Convex SPP-1200, SPP-1600, SPP-2000.
Модификации	В настоящее время доступны несколько "классов" систем семейства HP 9000: сервера начального уровня (D,K-class), среднего уровня (N-class) и наиболее мощные системы (V-class).
Процессоры	64-битные процессоры с архитектурой PA-RISC 2.0 (PA-8200, PA-8500).
Число процессоров	N-class - до 8 процессоров. V-class - до 32 процессоров. В дальнейшем ожидается увеличение числа процессоров до 64, а затем до 128.
Масштабируемость	SCA-конфигурации (Scalable Computing Architecture) - до 4 узлов V-class, т.е. до 128 процессоров.
Системное ПО	Устанавливается операционная система HP-UX (совместима на уровне двоичного кода с ОС SPP-UX компьютеров Convex SPP).
Средства программирования	HP MPI - реализация MPI 1.2, оптимизированная к архитектуре Exemplar. Распараллеливающие компиляторы Fortran/C, математическая библиотека HP MLIB. CXperf - с редство анализа производительности программ.
Обзор	Обзор архитектуры серверов HP 9000 класса V корпорации Hewlett-Packard

- parallel.ru/computers/computers.html#exemplar

Преимущества OpenMP

1. За счет идеи **"инкрементального распараллеливания"** OpenMP идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами. Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы OpenMP-директивы.
2. При этом, OpenMP – достаточно **гибкий механизм**, предоставляющий разработчику большие возможности контроля над поведением параллельного приложения.
3. Предполагается, что OpenMP-программа на однопроцессорной платформе может быть использована **в качестве последовательной** программы, т.е. нет необходимости поддерживать последовательную и параллельную версии. Директивы OpenMP просто игнорируются последовательным компилятором, а для вызова процедур OpenMP могут быть подставлены заглушки (stubs), текст которых приведен в спецификациях.
4. Одним из достоинств OpenMP его разработчики считают поддержку так называемых **"orphan" (оторванных) директив**, то есть директивы синхронизации и распределения работы могут не входить непосредственно в лексический контекст параллельной области.

Цели OpenMP

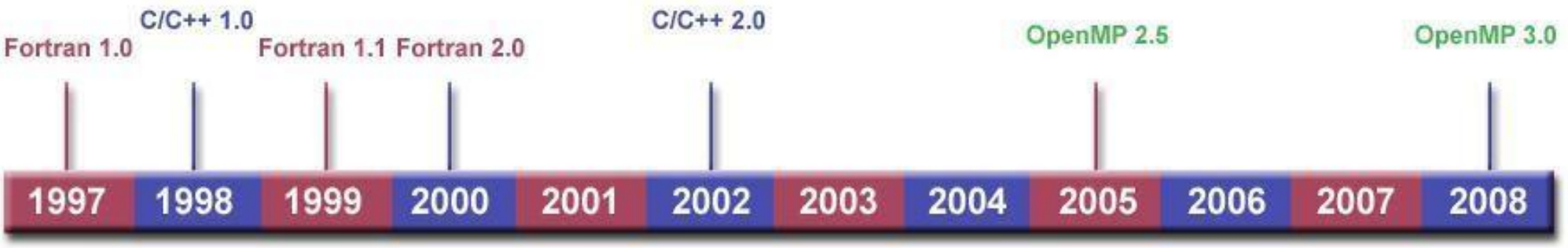
- Стандартизация:
  - Единый стандарт для различных систем/архитектур/платформ с общей памятью.
  - Определен совместными усилиями ведущих поставщиков программного и аппаратного обеспечения.

- Краткость и выразительность:
  - Небольшой набор директив и дополнительных элементов (clause).

- Простота использования:
  - Упрощенный механизм создания параллельных программ, практически не влияющий на работу программиста.

- Переносимость:
  - Поддерживается в C/C++ и Фортран.
  - Поддерживается но множестве платформ (Unix/Linux, MacOS, Windows).

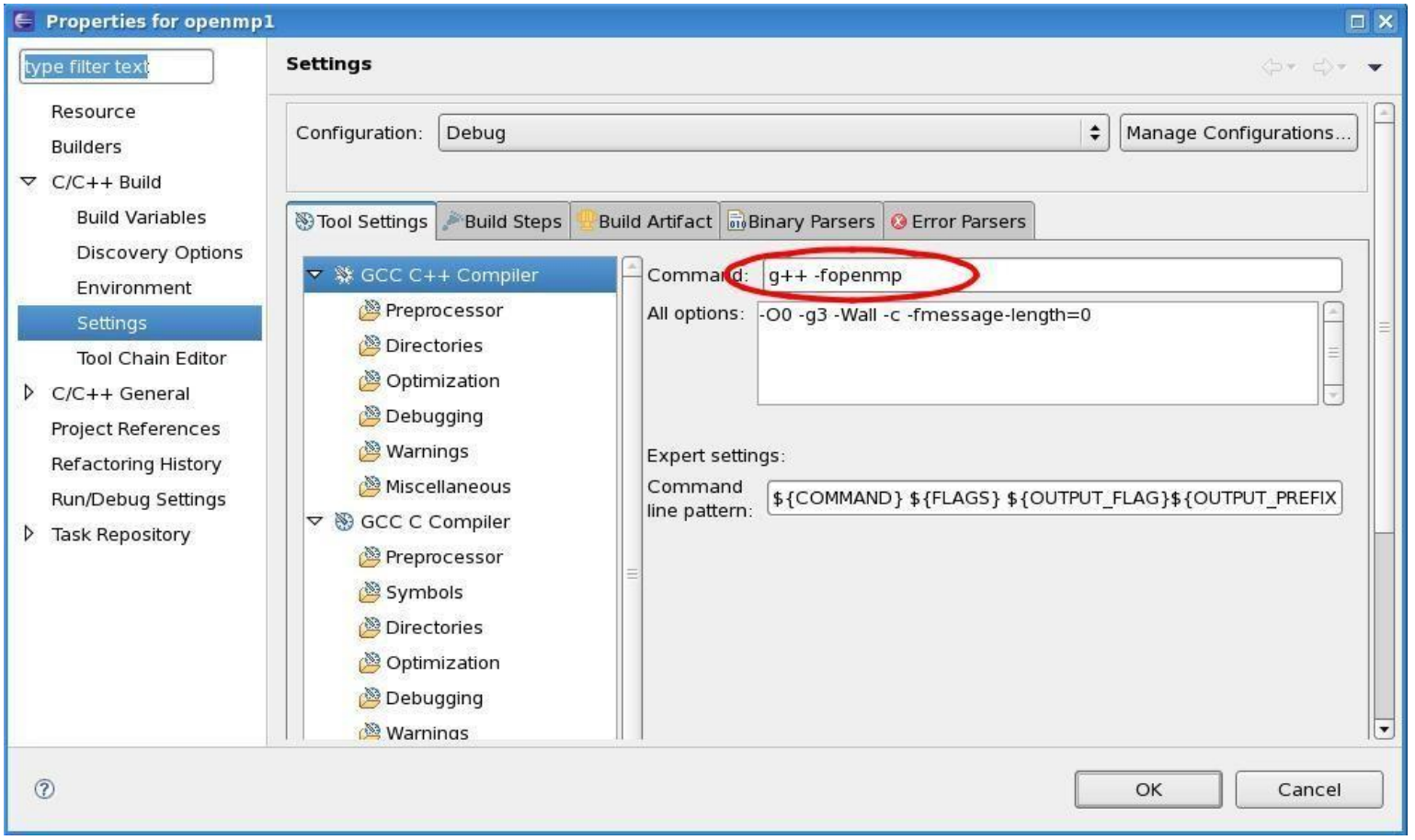
История OpenMP



- Расширения к Фортрану
  - Начало 90-х
- Проект X3H5 – первая попытка стандарта ANSI
  - 1994 год
- Начало разработки OpenMP
  - 1997 год
- Поддерживается OpenMP Architecture Review Board (ARB)
  - Intel, AMD*,ARM*, Cray*, IBM*, HP*, Micron*, NEC*, Nvidia*, Oracle* etc.
- Версии 1.0-2.5 (Октябрь 1997 – Май 2005)
  - Первые версии, внедрение и развитие потокового распараллеливания циклов
- Версии 3.0, 3.1 (Май 2008 – Июль 2011)
  - Добавление и развитие поддержки независимых задач
- Версия 4.0 … (Июль 2013)
  - Поддержка векторизации циклов (SIMD), поддержка ускорителей (target), зависимые задачи, встроенные механизмы обработки ошибок (cancel), пользовательские редукции, расширение атомарных конструкций

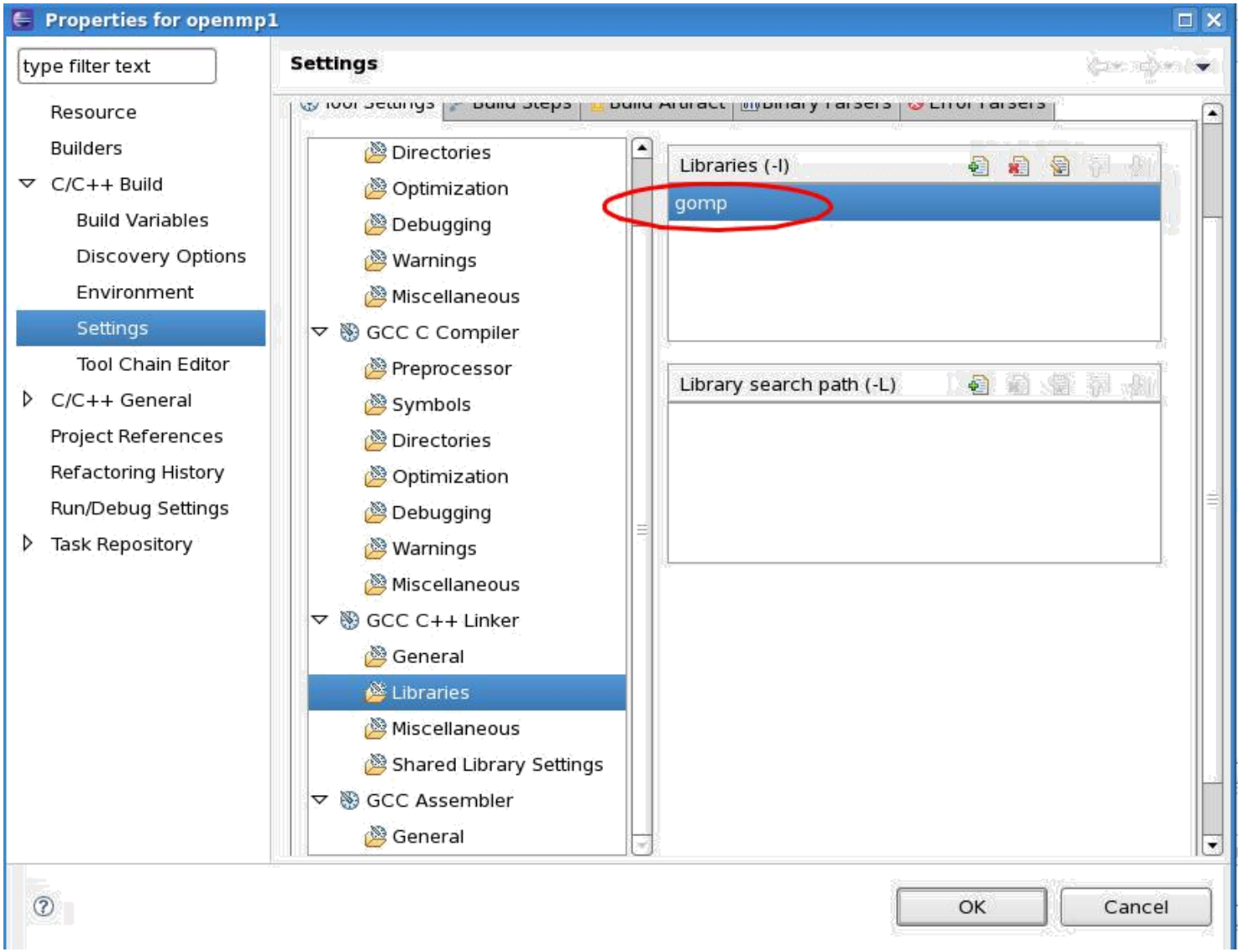
Компиляция в Eclipse

-fopenmp



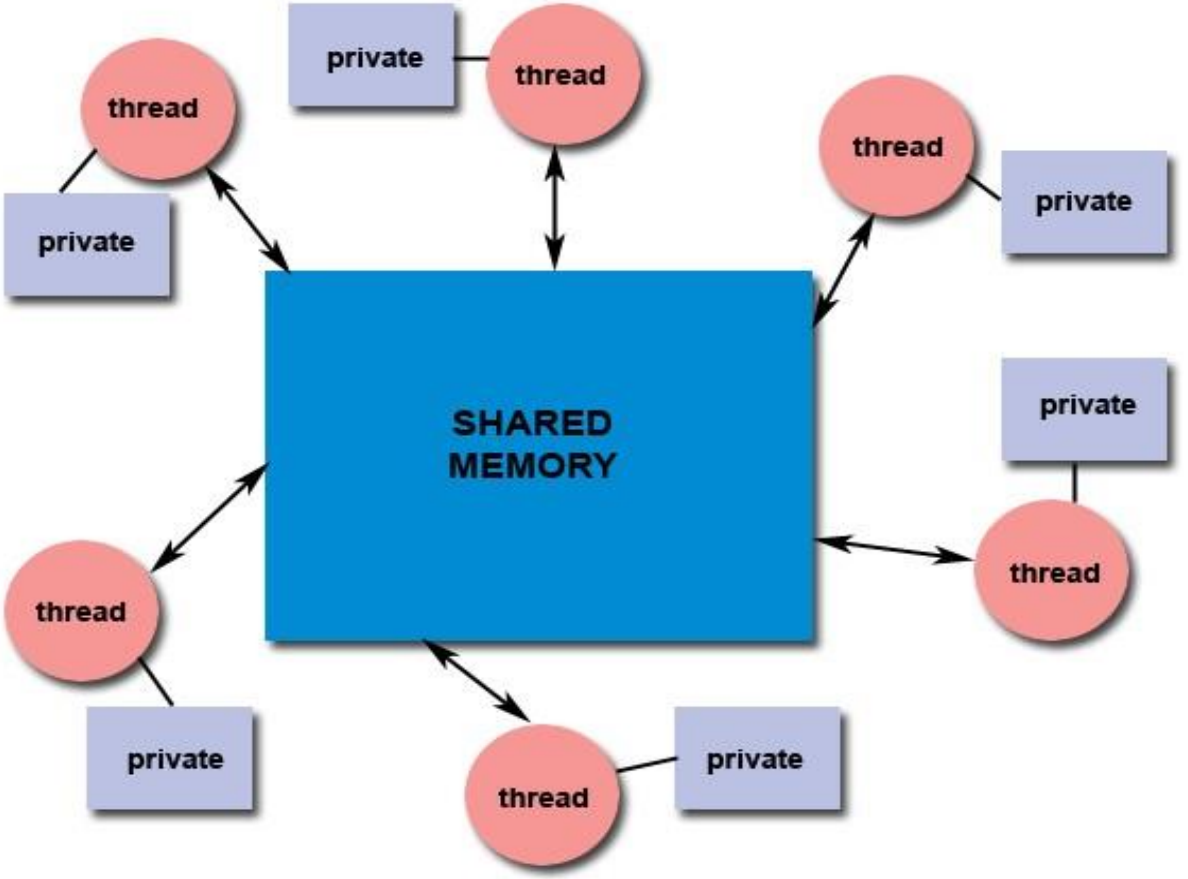
Компиляция в Eclipse

-lgomp



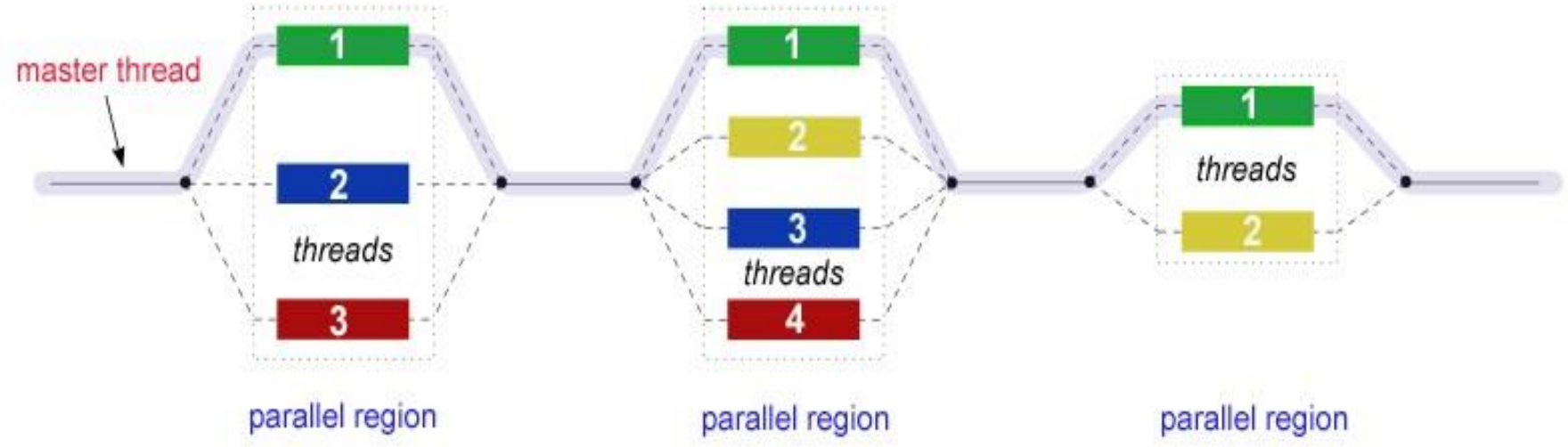
Модель с разделяемой памятью

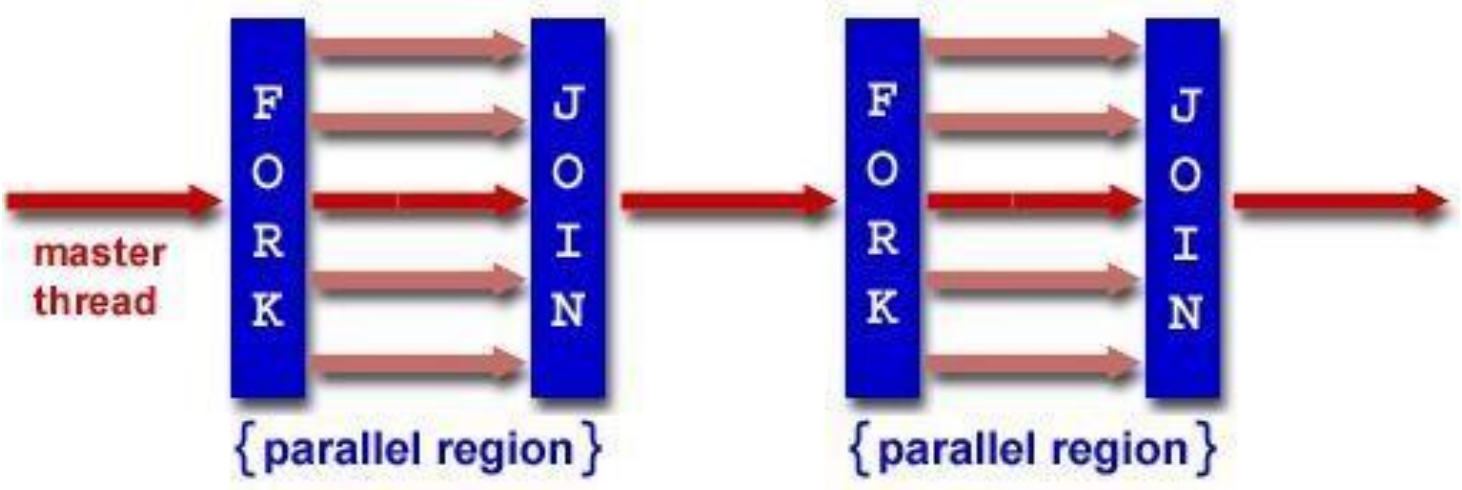
- Все потоки имеют доступ к глобальной общей памяти.
- Данные могут быть общие и приватные.
- Общие данные доступны всем потокам.
- Приватные данные доступны только одному потоку-владельцу.
- Требуется синхронизация для доступа к общим данным.



Модель программирования в OpenMP

https://computing.llnl.gov/tutorials/openMP/#Abstract





Структура OpenMP-программы

Программа представляется в виде последовательных участков кода (serial code) и параллельных регионов (parallel region).

Каждый поток имеет номер Thread Id (числовой идентификатор, или, номер).

Мастер поток (master) имеет номер 0.

Память процесса (heap) является общей для всех потоков. OpenMP реализует динамическое управление потоками (task parallelism).

Когда использовать OpenMP

Знать, когда использовать технологию OpenMP, не менее важно, чем уметь с ней работать.

Целевая платформа является многопроцессорной или многоядерной. Если приложение полностью использует ресурсы одного ядра или процессора, то, сделав его многопоточным при помощи OpenMP, вы почти наверняка повысите его быстродействие.

Приложение должно быть кроссплатформенным. OpenMP – кроссплатформенный и широко поддерживаемый API. А так как он реализован на основе директив pragma, приложение можно скомпилировать даже при помощи компилятора, не поддерживающего стандарт OpenMP.

Выполнение циклов нужно распараллелить. Весь свой потенциал OpenMP демонстрирует при организации параллельного выполнения циклов. Если в приложении есть длительные циклы без зависимостей, OpenMP – идеальное решение.

Перед выпуском приложения нужно повысить его быстродействие. Так как технология OpenMP не требует переработки архитектуры приложения, она прекрасно подходит для внесения в код небольших изменений, позволяющих повысить его быстродействие.

Однако OpenMP – **не панацея от всех бед**. Технология ориентирована в первую очередь на разработчиков высокопроизводительных вычислительных систем и наиболее эффективна, если код включает много циклов и работает с разделяемыми массивами данных.

Как использовать OpenMP в коде

Директивы

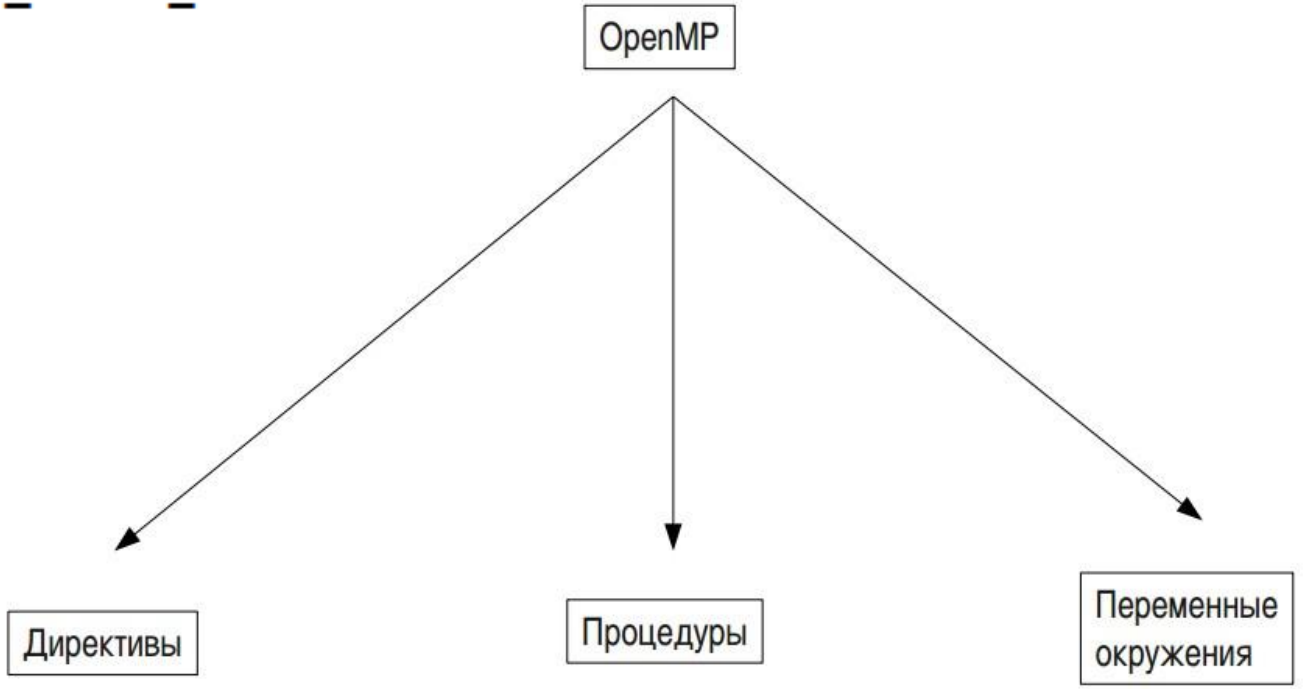
```
atomic
barrier
critical
do
end critical
end do
end master
end parallel
end sections
end single
flush
for
master
ordered
parallel
section
sections
single
task
taskwait
threadprivate
```

Функции

```
omp_destroy_lock
omp_destroy_nest_lock
omp_get_active_level
omp_get_max_threads
omp_get_nested
omp_get_num_procs
omp_get_num_threads
omp_get_schedule
omp_get_team_size
omp_get_thread_limit
omp_get_thread_num
```

Переменные окружения

```
OMP_DYNAMIC
OMP_MAX_ACTIVE_LEVELS
OMP_NESTED
OMP_NUM_THREADS
OMP_SCHEDULE
OMP_STACKSIZE
OMP_THREAD_LIMIT
OMP_WAIT_POLICY
```



Задать/узнать число потоков

export OMP_NUM_THREADS=5 ← задать количество потоков в командной оболочке (bash)

void omp_set_num_threads(int num_threads) ← задать количество потоков внутри кода программы

int omp_get_num_threads() – общее количество потоков (нитей). По умолчанию количество потоков равно количеству логических процессоров в системе (*cat /proc/cpuinfo*)

int omp_get_thread_num() – определение номера потока в текущей параллельной секции кода

int omp_get_max_threads(); – сколько можно запустить потоков по умолчанию (без опции num_threads)

int omp_in_parallel(); – находимся ли в параллельной секции

Директива parallel

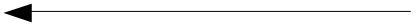
```
int main() {

    // ...
    // ...

#pragma omp parallel
    {
        // ...
    }

    // ...
    // ...

    return 0;
}
```



последовательная часть программы

параллельная часть программы

последовательная часть программы

Директива parallel

```
#pragma omp parallel опции
{
    //...
}
```

Подводные камни OpenMP при программировании на C++

Многие ошибки не диагностируются!

Ошибки приводят к некорректному поведению параллельных программ, созданных на основе технологии OpenMP.

Читайте статью здесь:

http://www.viva64.com/ru/a/0054/#ID0EOG

Ошибка «Отсутствие parallel»

```
 #pragma omp for
  …
```

Код работает в последовательном режиме

Правильный код:

```
#pragma omp parallel
{
  #pragma omp for
  …
}
```

или

```
#pragma omp parallel for
…
```

Ошибка «Двойное распараллеливание»

```
#pragma omp parallel num_threads(2)
{
    ... // N строк кода
    #pragma omp parallel for
    for (int i = 0; i < 10; i++)
      myFunc();
}
```

Код выполнится 20 раз

Также ошибки производительности

1. Ненужная директива flush.
2. Использование критических секций или блокировок вместо atomic.
3. Ненужная защита памяти от одновременной записи.
4. Неоправданно большое количество кода в критической секции.
5. Слишком частое применение критических секций.

Пример - найти ошибку

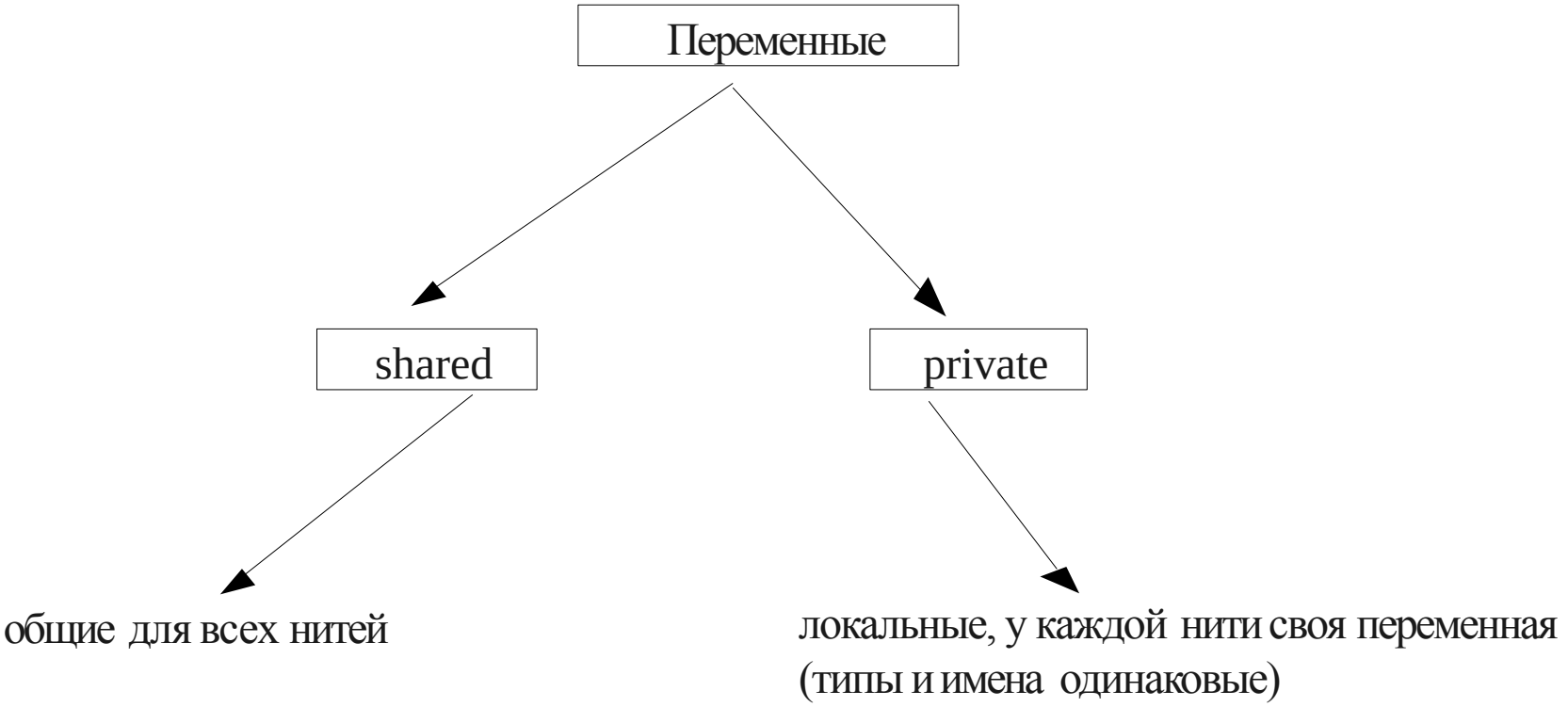
Найти ошибку, чтобы вывод мог быть в виде:

```
Thread:0Thread:
Master:4
Thread:3
Thread:2
1
```

```
#include <iostream>
#include <omp.h>
using namespace std;
int main()
{
  int nthreads, tid;
#pragma omp parallel
  {
    tid = omp_get_thread_num();
    cout << "Thread:" << tid << endl;
    if (tid == 0) {
      nthreads = omp_get_num_threads();
      cout << "Master:" << nthreads << endl;
    }
  }
  return 0;
  }
```

l03_01.cpp

Модель данных в OpenMP



По умолчанию,

- переменные, объявленные вне параллельной области – shared
- переменные, объявленные в параллельной области – private

Опции директивы parallel

Опция	**Пояснение**
if (условие)	Порождать параллельные потоки, или выполнять только главный поток
num_threads (целое число)	Задает количество параллельных потоков
default (shared | none)	Класс переменных по умолчанию; none – класс всем переменным должен быть назначен явно
private (список)	Переменные, для которых порождается локальная копия в каждом потоке
firstprivate	То же, что и private + переменные инициализируются значениями, которые имеют аналогичные перемен ные в главном потоке
shared	Переменные, общие для всех потоков
copyin	Копировать значения переменных threadprivate из главного потока в переменные threadprivate дочерних
reduction (оператор: список)	По окончании параллельного блока применить оператор к переменным из списка

Пример работы опций if, num_threads

```
int is_parallel=0;
#pragma omp parallel if(is_parallel == 1) num_threads(8)
    {
        cout<<"I am thread №
        "<<omp_get_thread_num()<<endl;
    }
```

Сколько потоков в параллельной секции?

Существует следующий приоритет при определении числа потоков
(от старшего правила к младшему)

- Опция if
- **Опция num_threads**
- функция omp_set_num_threads
- переменная окружения OMP_NUM_THREADS
- по умолчанию – количество процессоров/ядер на узле

Пример переменной shared

```
int iShared=0;
#pragma omp parallel num_threads(8)
    {
        iShared=omp_get_thread_num();  cout<<iShared<<endl;
    }
    cout<<"Last value of iShared "<<iShared<<endl;
```

Для улучшения кода - указывать явным образом опцию: shared(iShared)

Пример переменной private

```
#pragma omp parallel num_threads(8)
    {
        int iPrivate=omp_get_thread_num();  cout<<iPrivate<<endl;
    }
//iPrivate здесь не объявлено
```

Пример опции private

```
int iPrivate=0;
#pragma omp parallel num_threads(8) private(iPrivate)
    {
        iPrivate=omp_get_thread_num();  cout<<iPrivate<<endl;
    }
```