Введение в параллельные вычисления

д.т.н. Мокрова Наталия Владиславовна

асп. Морунов Егор, Сырко Денис

пятница	ауд. 118, 119	
12:50 – 14:20	Лекция	
	1 неделя	2 неделя
14:35 - 17:55	Лаб – КС44	Лаб - КС40

Выбор модели программирования

Системы с общей памятью

- C++11 threads (pthreads)
- OpenMP

Системы с распределенной памятью

- MPI = message passing interface

Гетерогенные системы (графические процессоры)

- CUDA

Лекция 5. Возможности OpenMP

- The OpenMP® API specification for parallel programming http://openmp.org.
- Оригинальная документация OpenMP API v.3.0.
- Антонов А.С. Параллельное программирование с использованием технологии OpenMP. М.: Издво МГУ, 2009.
- Примеры из учебника «Технологии параллельного программирования MPI и OpenMP» http://parallel.ru/tech/tech_dev/MPI%26OpenMP/examples
- Примеры http://pro-prof.com/archives/1150
- Демонстрация примеров http://coliru.stacked-crooked.com/
- Параллельное программирование с использованием OpenMP (рус.). Учебный курс на сайте Intuit.ru
- OpenMP. Blaise Barney, Lawrence Livermore National Laboratory https://computing.llnl.gov/tutorials/openMP/

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 суперкомпьютер

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

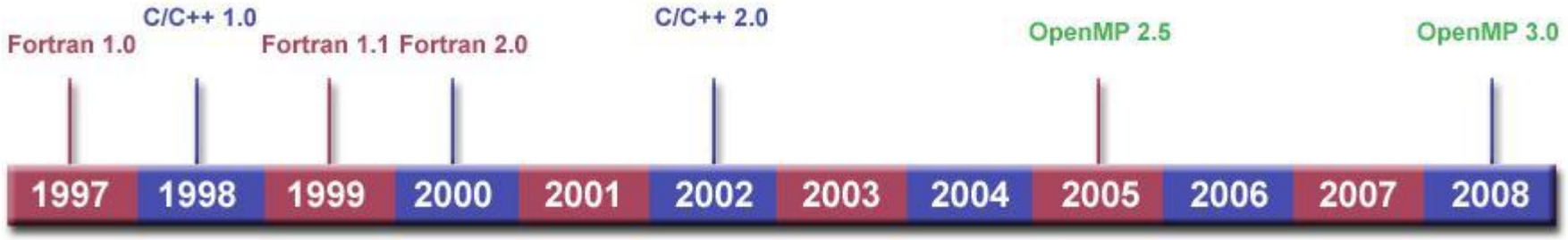
Преимущества OpenMP

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

OpenMP Specifications

Первая версия появилась в 1997 году, предназначалась для языка Fortran. Для C/C++ версия разработана в 1998 году. В 2008 году вышла версия OpenMP 3.0.

OpenMP Specifications

OpenMP 4.5 Specifications

- OpenMP 4.5 Complete Specifications (November 2015) PDF
- OpenMP 4.5 Discussion Forum
- OpenMP 4.5 Summary Card - C/C++ (November 2015) PDF
- OpenMP 4.5 Summary Card - Fortran (November 2015) PDF

OpenMP 4.0 Specifications

- OpenMP 4.0 Complete Specifications (July 2013) (PDF)
- OpenMP 4.0 Discussion Forum
- OpenMP 4.0 Summary Card - C/C++ (October 2013 PDF)
- OpenMP 4.0 Summary Card - Fortran (October 2013 PDF)
- OpenMP Examples 4.0.2 (March 2015 PDF)
- OpenMP 4.0.1 Examples (February 2014 PDF)

Active Technical Report Drafts and Proposals

TR3: **Initial comment draft for the OpenMP 4.1 specification.**

- **Differences between 4.0 and TR3**

…..November 2014

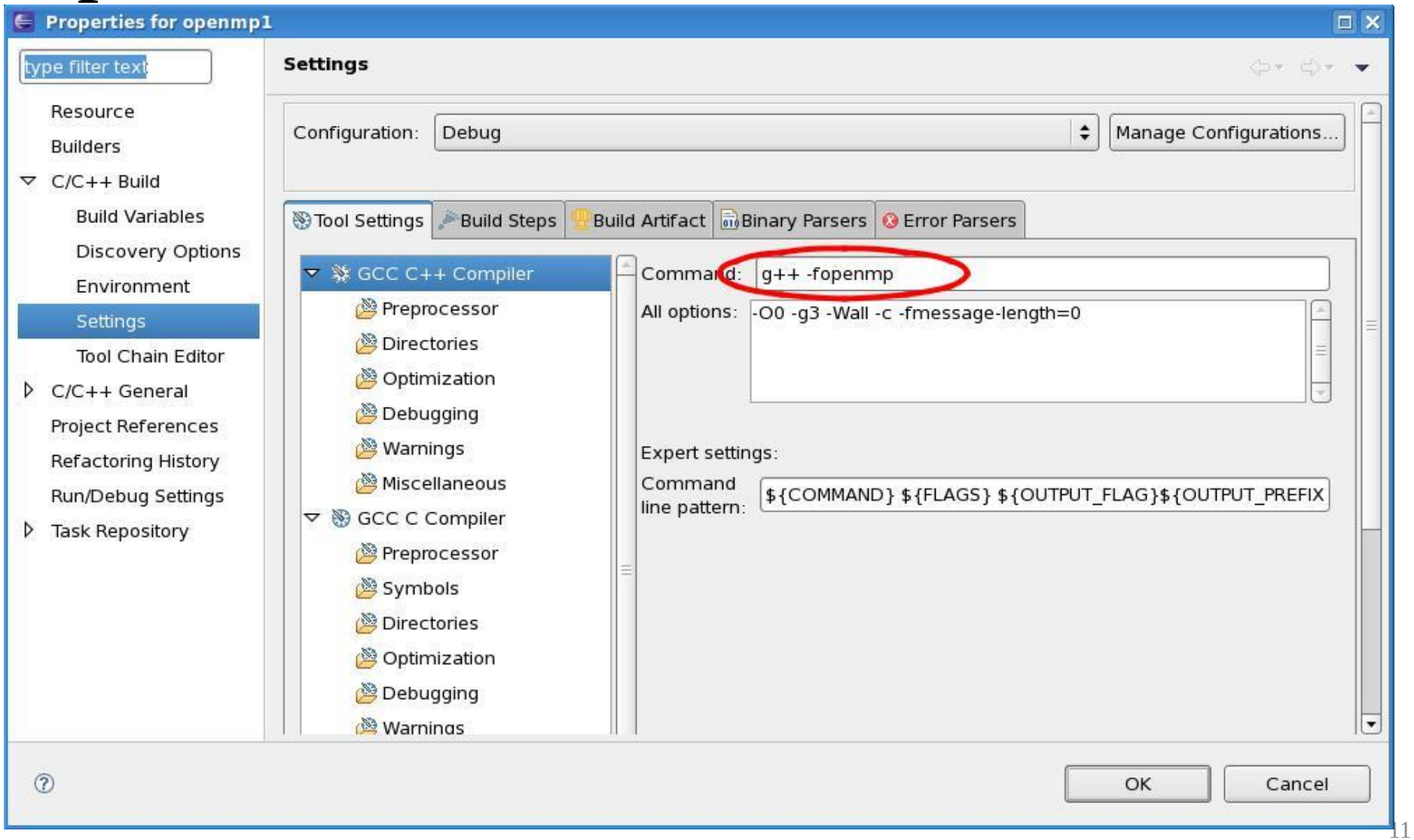
Поддержка OpenMP компиляторами

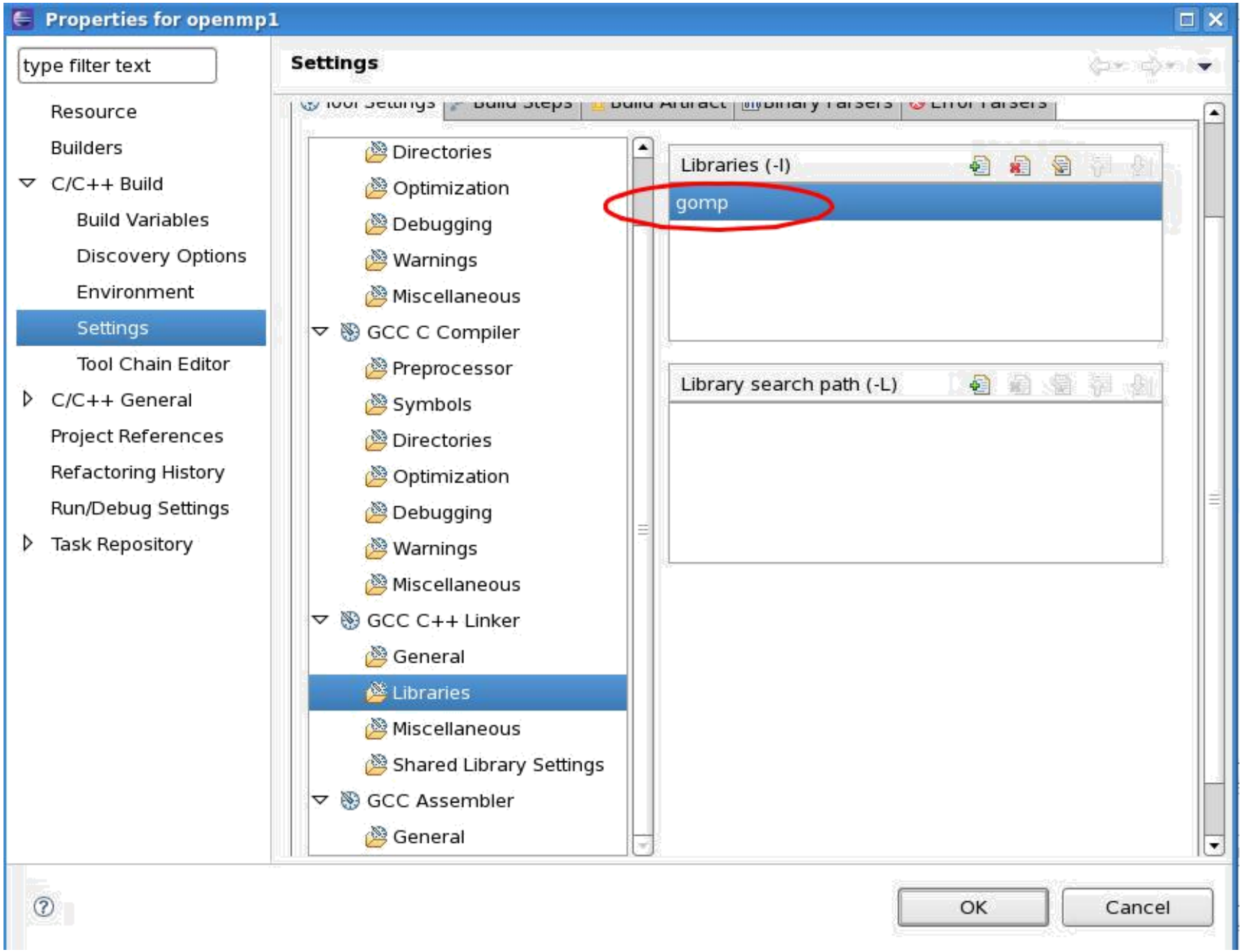
Производитель	Компилятор	Примечание
GNU	gcc (4.3.2)	Linux, Solaris, AIX, MacOSX, Windows
IBM	XL C/C++ Fortran	Windws, AIX, Linux
SUN	C/C++/Fortran	Solaris, Linux
Intel	C/C++/Fortran (10.1)	Windows, Linux, MacOSX
MS	VS 2008 C++	OpenMP 2.0

OpenMP и GCC

GCC	OpenMP
4.1 (Fedora Core 5)	2.5
4.2	2.5
4.4	3.0

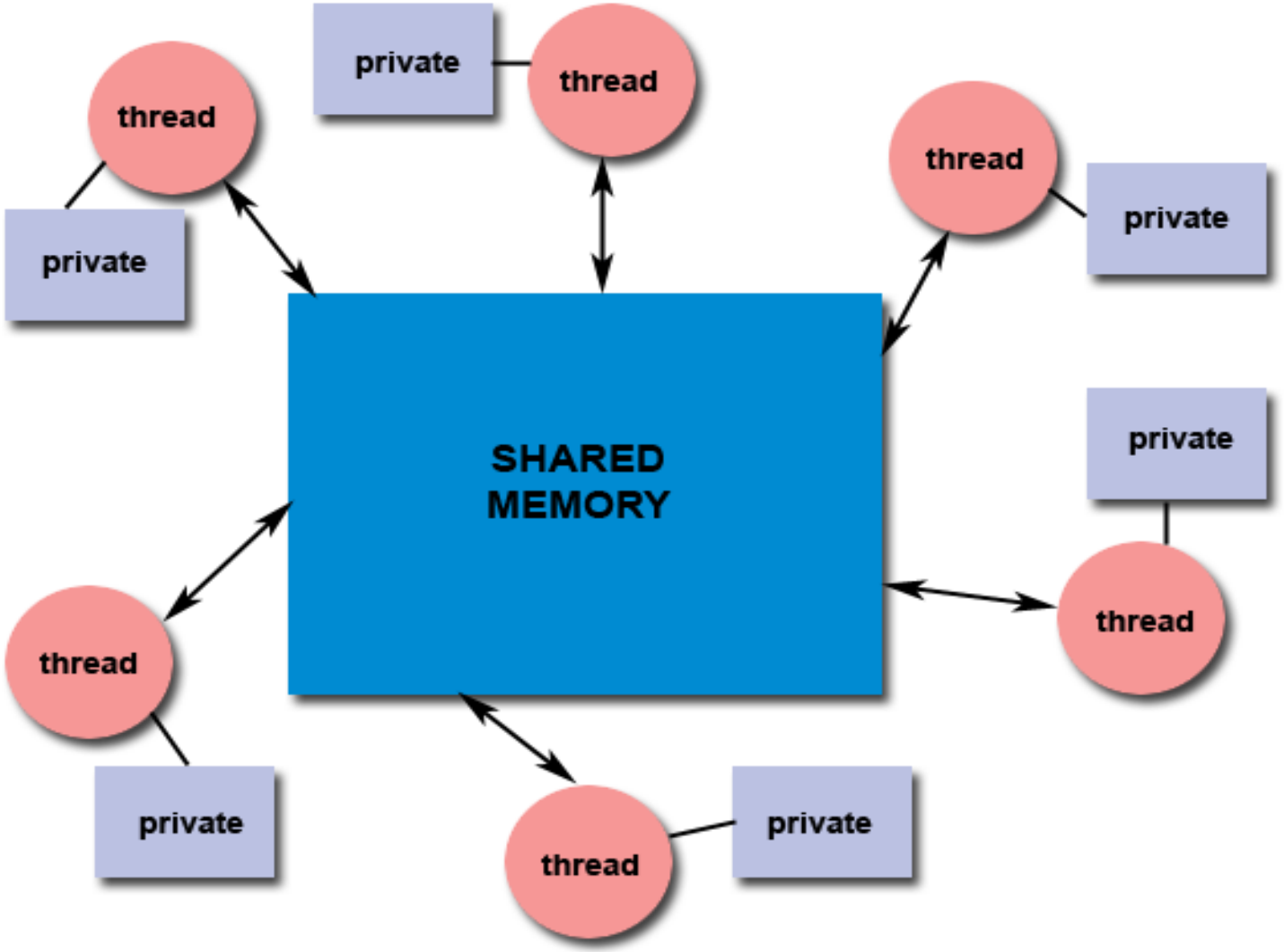
-fopenmp


Properties for openmp1
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Project References
Refactoring History
Run/Debug Settings
Task Repository
Settings
Configuration: Debug
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
GCC C++ Compiler
Preprocessor
Directories
Optimization
Debugging
Warnings
Miscellaneous
GCC C Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Command: g++ -fopenmp
All options: -O0 -g3 -Wall -c -fmessage-length=0
Expert settings:
Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG}${OUTPUT_PREFIX
OK
Cancel


Properties for openmp1
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Project References
Refactoring History
Run/Debug Settings
Task Repository
Settings
Directories
Optimization
Debugging
Warnings
Miscellaneous
GCC C Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
GCC C++ Linker
General
Libraries
Miscellaneous
Shared Library Settings
GCC Assembler
General
Libraries (-l)
gomp
Library search path (-L)
OK
Cancel

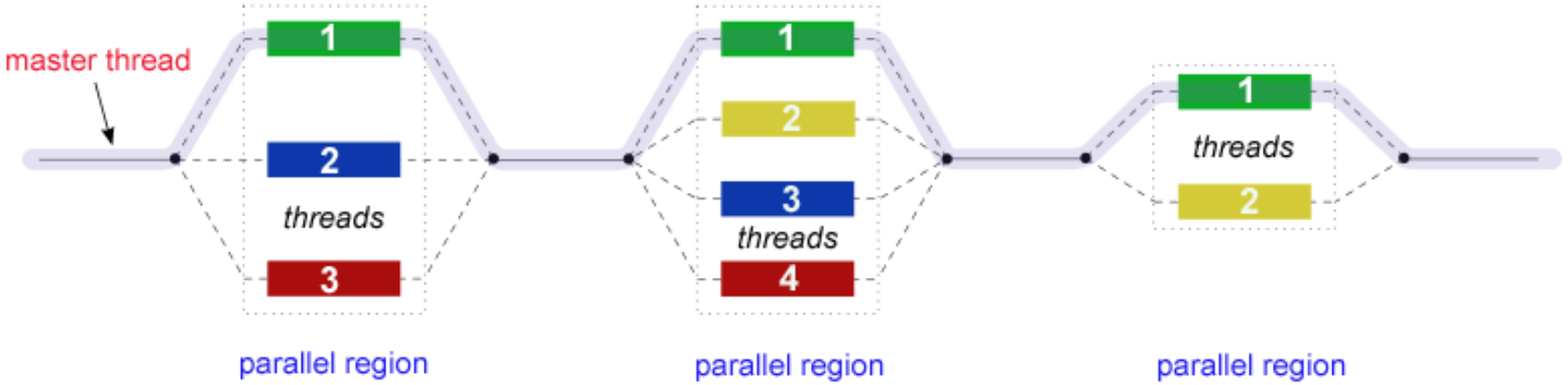
Модель с разделяемой памятью

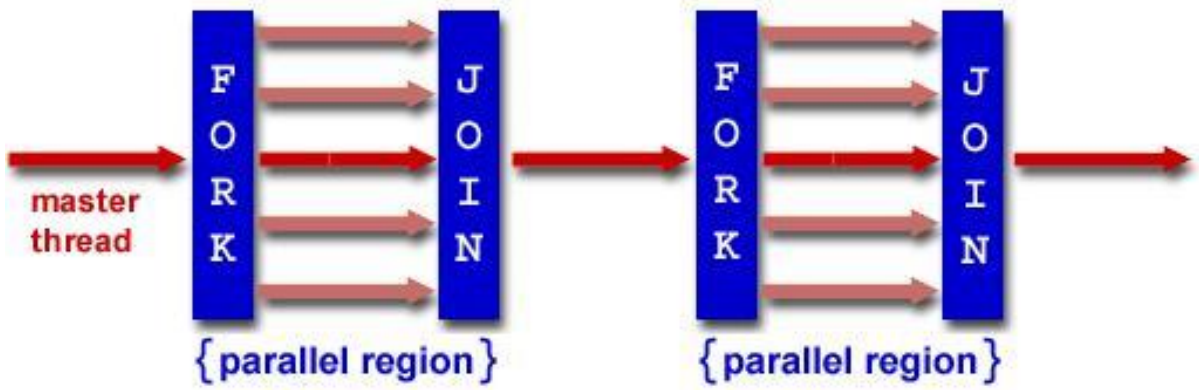
- Все потоки имеют доступ к глобальной общей памяти.
- Данные могут быть общие и приватные.
- Общие данные доступны всем потокам.
- Приватные данные доступны только одному потоку-владельцу.
- Требуется синхронизация для доступа к общим данным.



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.

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.

Cluster OpenMP

Cluster OpenMP – простое средство, требующее незначительного изменения кода для расширения параллелизма OpenMP на кластерные системы под ОС Linux, базирующиеся на 64-разрядной архитектуре Intel.

OpenMP – известная парадигма параллельного программирования для систем с общей памятью. Высокая стоимость больших систем с общей памятью была главным ограничением для использования OpenMP. Для совместного использования нескольких SMP-систем зачастую прибегали к помощи MPI для обмена сообщениями между этими системами. Разумеется, это неизбежно приводило к усложнению не только написания самой программы, но и к усложнению разработки приложения в целом, координации рабочих групп и т.п. Cluster OpenMP позволяет как избавиться от смешивания этих парадигм, так и сократить усилия на написание дополнительного кода. В Cluster OpenMP поддерживается система распределенной общей памяти. Cluster OpenMP позволяет синхронизировать общие переменные только в случае необходимости.

На текущий момент Cluster OpenMP работает на системах под ОС Linux с поддержкой сокетов и uDAPL API на базе Intel Itanium и системах, поддерживающих Intel EM64T. Работоспособность проверена на Ethernet и Infiniband, однако, в будущем будет поддержка и других интерконнектов. Также планируется увеличение производительности за счет расширения использования новых возможностей таких средств, как RDMA.

Расширение Cluster OpenMP доступно для компиляторов Intel под Linux, начиная с версий 9.1. Использование Cluster OpenMP требует помимо собственной лицензию на использование компилятора Intel C++ или Fortran для Linux. Сама же лицензия на Cluster OpenMP может быть получена как отдельно, так и вместе с лицензией на компилятор.

Преимущества Cluster OpenMP

Переносимость и гибкость упрощает и снижает стоимость разработки приложений:

- Упрощает распределение последовательного или OpenMP кода по узлам.
- Позволяет использовать один и тот же код приложений для последовательных, многоядерных и кластерных систем.
- Требует совсем незначительного изменения кода, что упрощает отладку.
- Позволяет слегка измененному коду OpenMP выполняться на большем числе процессоров без вложений в аппаратную составляющую SMP.
- Представляет собой альтернативу MPI, которая может быть быстрее освоена и применена.

Стоимость вычислений на кластерных системах:

Тип кластера	**Аппаратная составляющая расходов**	**Программная составляющая расходов**
Большие SMP системы с общей памятью и OpenMP		
Кластерные системы с распределенной памятью и MPI		
Кластерные системы с распределенной памятью и Cluster OpenMP		

Модель параллельной программы

Последовательная область – один процесс (нить), вход в параллельную область – порождение некоторого числа процессов, их завершение, нить-мастер. Параллельные области могут быть вложенными друг в друга.

В отличие от полноценных процессов, порождение нитей является относительно быстрой операцией, поэтому частые порождения и завершения нитей не так сильно влияют на время выполнения программы.

Для написания эффективной параллельной программы необходимо, чтобы все нити были равномерно загружены, что достигается тщательной балансировкой загрузки при помощи механизмов OpenMP.

Необходимость синхронизации доступа к общим данным – существенна. Само наличие данных, общих для нескольких нитей, приводит к конфликтам при одновременном несогласованном доступе.

Значительная часть функциональности OpenMP предназначена для осуществления различного рода синхронизаций работающих нитей.

OpenMP не выполняет синхронизацию доступа различных нитей к одним и тем же файлам. Пользователь должен явно использовать директивы синхронизации или соответствующие библиотечные функции. При доступе каждой нити к своему файлу никакая синхронизация не требуется.

Ключевые элементы OpenMP

Распараллеливание в OpenMP – вставка специальных директив и вызов вспомогательных функций.

Ключевые элементы:

- конструкции для создания потоков (директива parallel),
- конструкции распределения работы между потоками (директивы DO/for и section),
- конструкции для управления работой с данными (выражения shared и private для определения класса памяти переменных),
- конструкции для синхронизации потоков (директивы critical, atomic и barrier),
- процедуры библиотеки поддержки времени выполнения (например, omp_get_thread_num),
- переменные окружения (например, OMP_NUM_THREADS).

Директивы

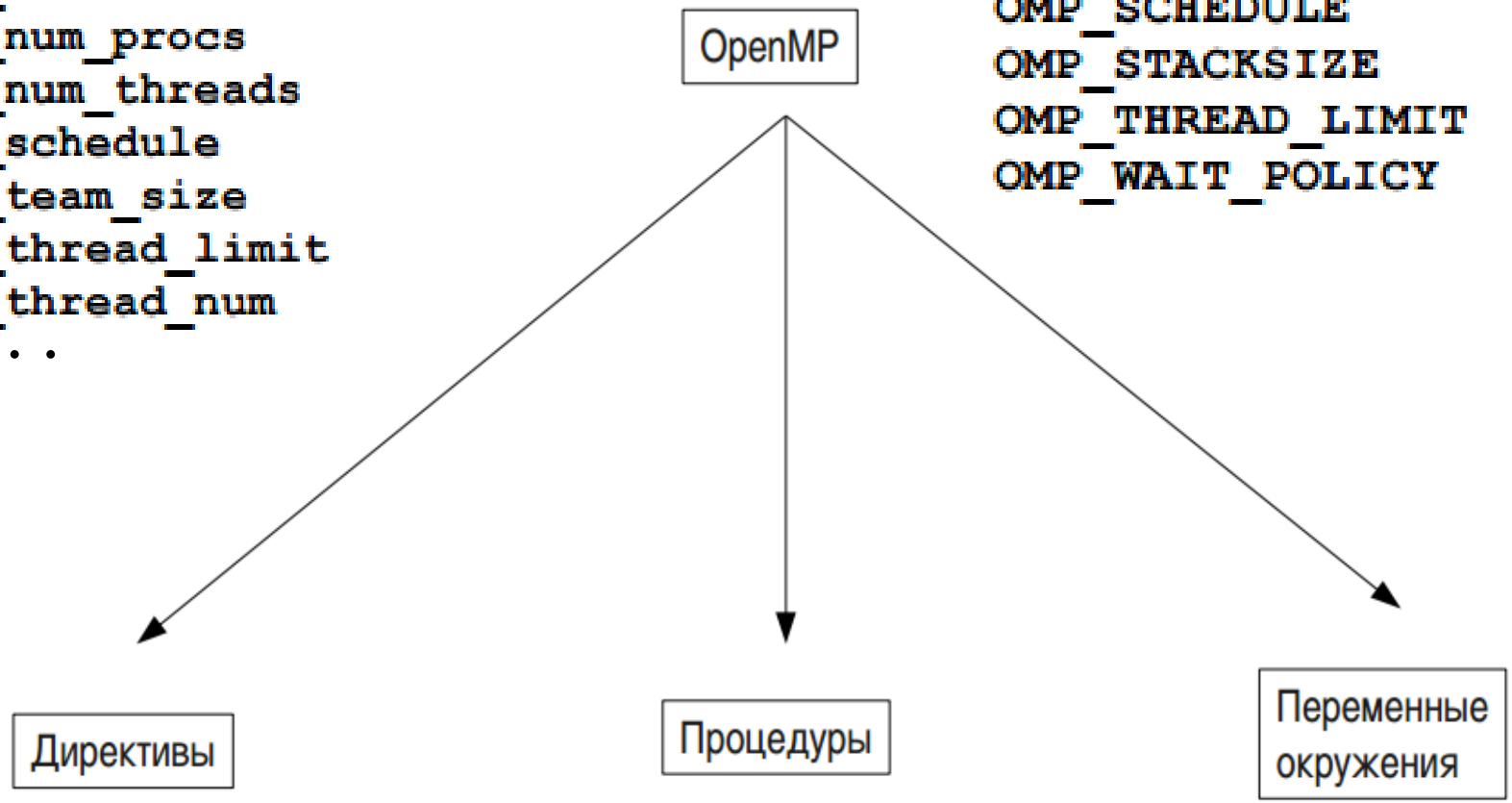
```
atomic
barrier
critical
do
end critical
end do
end master
end parallel
end sections
end single
flush
for
master
ordered
parallel
section
sections
single
task
taskwait
threadprivate
```

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
```



32 подводных камня OpenMP при программировании на Си++

Многие ошибки никак не диагностируются!

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

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

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.

Логические ошибки

1. Отсутствие /openmp.
2. Отсутствие parallel.
3. Отсутствие omp.
4. Отсутствие for.
5. Ненужное распараллеливание.
6. Неправильное применение ordered.
7. Переопределение количества потоков внутри параллельной секции.
8. Попытка использовать блокировку без инициализации переменной.
9. Попытка снять блокировку не из того потока, который ее установил.
10. Попытка использования блокировки как барьера.
11. Зависимость поведения от количества потоков.
12. Некорректное использование динамического создания потоков.
13. Одновременное использование общего ресурса.
14. Незащищенный доступ к общей памяти.
15. Использование директивы flush с указателем.
16. Отсутствие директивы flush.
17. Отсутствие синхронизации.
18. Внешняя переменная задана как threadprivate не во всех модулях.
19. Неинициализированные локальные переменные.
20. Забытая директива threadprivate.
21. Забытое выражение private.
22. Некорректное распараллеливание работы с локальными переменными.
23. Неосторожное применение lastprivate.
24. Непредсказуемые значения threadprivate-переменных в начале параллельных секций.
25. Некоторые ограничения локальных переменных.
26. Локальные переменные не помечены как таковые.
27. Параллельная работа с массивом без упорядочивания итераций.

Логические ошибки

1. Отсутствие /openmp.

Если не включена в настройках компилятора поддержка OpenMP, директивы OpenMP игнорируются. Компилятор не выдаст ни ошибки, ни даже предупреждения, код просто будет выполняться не так, как этого ожидает программист.

2. Отсутствие parallel.

Неправильное написании самих директив кода.

Некорректно:

```
#pragma omp for
... //код
```

Фрагмент кода успешно скомпилируется, но директива #pragma omp for будет проигнорирована компилятором. Таким образом, цикл будет выполняться только одним потоком, и обнаружить это достаточно затруднительно.

Помимо директивы #pragma omp parallel for, эта ошибка может возникнуть и с директивой #pragma omp parallel sections.

Корректно:

```
#pragma omp parallel for
... // код
#pragma omp parallel
{
  #pragma omp for
  ... //код
}
```

Логические ошибки

3. Отсутствие omp.

Некорректно:

```
#pragma omp parallel num_threads(2)
{
 #pragma single
   {
     printf("me\n");
   }
}
```

При выполнении кода сообщение "me" будет выведено два раза вместо ожидаемого одного.

Корректно:

```
#pragma omp parallel num_threads(2)
{
 #pragma omp single
   {
     printf("me\n");
   }
}
```

4. Отсутствие for.

Некорректно:

```
#pragma omp parallel num_threads(2)
for (int i = 0; i < 10; i++)
    myFunc();
```

Директива #pragma omp parallel может относиться как к блоку кода, так и к одной команде.

Корректно:

```
#pragma omp parallel for num_threads(2)
for (int i = 0; i < 10; i++)
    myFunc();
```

Для распараллеливания цикла на два потока следует использовать директиву #pragma omp parallel for (цикл выполнится 10 раз). В первом примере при запуске кода цикл будет выполняться по одному разу в каждом потоке, и функция myFunc будет вызвана 20 раз.

Логические ошибки

5. Ненужное распараллеливание

Применение директивы #pragma omp parallel к большому участку кода при невнимательности программиста может вызывать неожиданное поведение, аналогичное предыдущему случаю:

Некорректно:

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

Корректно:

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

Слово parallel написано внутри секции кода, уже распределенного на два потока.
В результате выполнения этого кода функция myFunc, как и в предыдущем примере, будет выполнена 20 раз, вместо ожидаемых 10 раз.

Анализатор кода?

В большинстве случаев ошибки можно диагностировать автоматически, средствами статического анализатора. На данный момент диагностику некоторых (лишь очень немногих) из них выполняет Intel Thread Checker. Некоторые ошибки диагностируются компиляторами. Специализированного инструмента обнаружения ошибок не существует.

Полезной для разработчиков могла бы оказаться программа, визуально отображающая распараллеливание кода и режимы доступа к переменным в соответствующих параллельных секциях, которой пока не существует.

Разработка статического анализатора кода, который будет диагностировать все перечисленные (пл. 22, 23) ошибки и, возможно, некоторые другие, сможет существенно упростить поиск ошибок, почти все из которых воспроизводятся нестабильно, при разработке параллельных приложений и их устранение.

Пример

Найти ошибку, чтобы вывод мог быть в виде:

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

omp_get_thread_num() – определение номера нити в текущей параллельной секции;
omp_get_num_threads() – количество нитей.
По умолчанию количество потоков равно количеству логических процессоров в системе.

```
#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;
}
```