Введение в OpenMP

Комментарии к слайдам.

Автор: Романенко Алексей

arom@ccfit.nsu.ru

Слайд 1
Данная лекция посвящена OpenMP. Презентация основана на материалах SUN Microsystems.

Слайд 2
Из данной лекции вы узнаете как собирать OpenMP программы, как они исполняются, что надо написать, чтобы программа стала параллельной и многое другое.

Слайд 3
План лекции представлен на слайде.

Слайд 4
История OpenMP началась в начале 90-х, когда стали появляться машины с общей памятью и различные производители пытаясь облегчить жизнь программистам разрабатывали свои системы их программирования. Было много вариантов, но все они имели одну и ту же функциональность. В 1994 были предприняты попытки выделить во всех подходах общее и представить это в качестве стандарта. 1997 год можно считать годом рождения стандарта OpenMP.

Слайд 5
Сейчас OpenMP является стандартом де-факто для программирования систем с общей памятью и в 2008 году был выпущена уже 3-я версия стандарта.

Слайд 6
При разработке стандарта преследовались цели создать простой механизм для распараллеливания программы, сделать его универсальным для всех SMP систем. Кроме того, он должен был легко интегрироваться в различные версии фортрана и встраиваться в Си.

Слайд 7
Напомним, что системы с общей памятью, или SMP системы — это системы в которых все процессоры (ядра одного процессора) имеют доступ к оперативной памяти, которая расположена в том же системном блоке.

Слайд 8
Эта оперативная память может распределятся между всеми потоками параллельной программы. Есть, конечно, и приватные данные, которые доступны только конкретному потоку. Очевидно, что для доступа к общей памяти требуется синхронизация.

Слайд 9
Все переменные имеют метку и в зависимости от этой метки или один и ли группа потоков имеют доступ к этой переменной.

Слайд 10
Модель исполнения OpenMP программы — это модель fork-join. Перед участком кода, который следует выполнять параллельно, создается группа потоков и после параллельного участка все созданные потоки завершаются.

Слайд 11
Правило 10-90 гласит, что 90% времени программа проводит в 10 % кода. Это 10% - циклы. Пример распараллеливания цикла представлен на слайде. Видно, что чтобы сделать цикл параллельным достаточно добавить одну строку. Простота — одна из целей приследуема разработчиками стандарта.

Слайд 12
Полную версию стандарта и пример программ можно найти на сайте, представленном на слайде.

Слайд 13
С каждым годом компиляторы становятся умнее и некоторые из них умеют автоматически распараллеливать код (Intel compiler). Однако не всегда они могут определить зависимость по данным или выявить участки кода нуждающиеся в параллелизации. Вот тогда следует применять OpenMP. Его также следует применять, когда пишется параллельная программа и при этом компилятор не умеет выполнять автоматическое распараллеливание.

Слайд 14
Основной поток и порожденные образуют OpenMP команду, в которой каждый поток пронумерован. Параллельный блок выполняется всеми потоками включая основной — мастер. Перед входом в параллельную область все потоки синхронизируются и начинают свое выполнение. Есть возможность указать в зависимости от каких-либо условий выполнять блок последовательно или параллельно — условие «if». Можно делать вложенные параллельные блоки и указывать как следует делить работу между потоками.

Слайд 15
Еще несколько терминов. На слайде параллелизаций цикла «for». Здесь в прагме «for» - директива, “shared” и “private” - клаузы. Конец параллельной секции — барьер.

Слайд 16

Слайд 17
Стандарт определяет для Си, что директивы и клаузы должны быть в нижнем регистре, что прагма может располагаться на нескольких строках и при компиляции с опциями OpenMP должен быть определен макрос.

Слайд 18
Пример переноса прагм и предопреленных макросов на слайде.

Слайд 19

Слайд 20
Отметим, что многие директивы поддерживают клаузы, которые дают компилятору дополнительную информацию. Все директивы имеют свой набор клауз.

Слайд 21
Клауза «if» - условие на то, стоит ли блок выполнть параллельно или последовательно.
Клауза «shared» задает список переменных, доступ к которым имеют все потоки.
Клауза «private» определяет переменные, которые индивидуальны для каждого потока.

Слайд 22
Пример, в котором цикл будет выполняться параллельно только в том случае, если «M» больше 100.

Слайд 23
Поскольку приватные переменные для каждого потока свои, то значение из при входе в параллельную секцию не определено, так же как не определено значение исходной переменной при выходе из параллельной секции. Чтобы определить значение этих переменных используются специальные клаузы, описание которых представлено на следующем слайде.

Слайд 24
Клаузой «firstprivate» все приватные переменные инициализируются значением, которое переменная имела до входа в параллельную секцию.
Клауза «lastprivate» присваивает внешней переменной значение, которое было на последней итерации.

Слайд 25, 26
Примеры использования клауз «firstprivate» и «lastprivate».

Слайд 27

Слайд 28
Рассматривая пример представленный на слайде, можно заметить, что значение переменной «sum» может варьироваться от запуска к запуску программы. Следует предпринимать дополнительные шаги, чтобы разграничить одновременную запись разных потоков в эту переменную.

Поскольку такие участки кода не редкость, была введена специальна клауза.

Слайд 29
Клауза редукции. Для ее использования следует указать операцию и над какой переменной она выполняется. Примеры демонстрируют применение этой клаузы.

Слайд 30
Редко, но используется клауза «nowait», которая говорит, что не надо на выходе из параллельной секции дожидаться синхронизации со всеми потоками.

Слайд 31
Базовой директивой в OpenMP является директива “parallel”. Она определяет участок кода, который будет выполняться параллельно всеми потоками.

Слайд 32
Эта директива поддерживает клаузы представленные на слайде. Большинство из них были рассмотрены выше. Стоит сказать только, что клауза «num_threads» задает количество потоков в OpenMP команде.

Слайд 33
Следующие директивы задают правило распределение работ между потоками.

Слайд 34
Таких директив три: for, single, section

Слайд 35

На слайде графически представлено то, что скрывается за этими директивами.

Слайд 36
Для директивы “for” итерации цикла распределяются между потоками. Используемые при этом клаузы представлены на слайде. Клауза «ordered», как следует из названия, заставляет выполняться итерации цикла строго в той последовательности, как и в последовательной программе. Клауза «schedule» задает то, как итерации цикла будут делиться между потоками.

Слайд 37
Это необходимо для обеспечения балансировки загрузки потоков и как следствие поднятия производительности программы в целом. Необходимость использовать соответствующую клаузу возникает, когда время выполнения итераций не равномерно.

Слайд 38
schedule static - равномерно (по кругу) распределяет итерации по потокам.

Слайд 39
schedule dnamic — процесс берет следующую свободную порцию данных заданного размера
schedule guided — аналогично предыдущему, но размер блока экспоненциально уменьшается
schedule runtime - политика определяется во время выполнения. Полезно для проведения эксперементов без перекомпиляции программы.

Слайд 40
Если параллельно надо выполнять не итерации цикла а участки кода, то для этих целей может служить директива «sections». Используемые в месте с этой директивой клаузы рассматривались ранее.

Слайд 41

Слайд 42
Если для представленного на слайде примере каждый из циклов выполнять параллельно, то результат будет не предсказуем. Требуется синхронизация для доступа к a[i].

Слайд 43
В некоторых случаях может помочь прагма «barrier». Исполнение потоков продолжится после того, как все потоки достигнут барьера.

Слайд 44
Вариантов использования барьеров много, но надо помнить, что их использование — дорогая операция. Помимо вариантов указанных на слайде, барьеры могут использоваться для замера времени, профилирования.

Слайд 45
Критические секции — еще один способ обеспечить целостность данных. Для корректной работы вычисление суммы должно быть или атомарным, или быть завернутым в критическую секцию.

Слайд 46
Синтаксис использования критических регионов и атомарных операций представлен на слайде. Следует помнить, что синхронизация потоков — дорогая операция и в конечном итоге может привести к тому, что в программе будет все больше последовательных участков.

Слайд 47
Если в параллельном блоке операцию должен выполнять только один поток, то это могут обеспечит прагмы «master» и «single». Разница в том, кто будет этим единственным потоком.

Слайд 48
Выполнение OpenMP программы может контролироваться переменными окружения.

Слайд 49
Таких переменных окружения всего 4.
Первая контролирует количество потоков в команде. По умолчанию количество потоков равно количеству доступных ядер процессоров.

Вторая определяет способ распределения работы между потоками

В некоторых случаях система может динамически изменять количество нитей, используемых для выполнения параллельной области, например, для оптимизации использования ресурсов системы. Это разрешено делать, если переменная среды **OMP_DYNAMIC** установлена в true

Последняя переменная окружения контролирует возможность вложенного параллелизма, т.е. Может ли OpenMP команда создавать новую команду процессов.

Слайд 50

Слайд 51
Помимо прагм и макросов в OpenMP определены функции времени выполнения. Они позволяют получать информация, менять поведение потоков в команде и обеспечивать исключительный доступ к ресурсам.

Слайд 52
Список некоторых функций представлен на слайде.

Слайд 53
Для обеспечения блокировок при доступе к ресурсам требуется использовать особый тип. Это по сути мутекс.

Слайд 54
Для использования локов, их надо создать, инициализировать, а по завершению работы разрушить.

Слайд 55
OpenMP поддерживается не всеми компиляторами, но тех, которые его поддерживают с каждым годом все больше.

Слайд 56
Для сборки OpenMP программы компилятору надо указать дополнительную опцию. Без нее программа так и останется последовательной.

Слайд 57
OpenMP — простой способ сделать вашу программу параллельной с минимумом усилий.