Введение. Параллельные программы как расширение последовательных.

Комментарии к слайдам.

Автор: Романенко Алексей

arom@ccfit.nsu.ru

Слайд 1

Этой лекцией мы начинаем курс, который познакомит вас с основами параллельного программирования, даст представления о том, что такое параллельная программа, на каких платформах она может исполняться и какие есть средства для написания и отладки параллельных программ. Первая лекция является введением и познакомит вас с предметной областью. Все последующие лекции будут посвящены конкретным технологиям.

Зовут меня Алексей Романенко. Если в процессе лекций или практических занятий у вас будут возникать вопросы, задавать их можно мне лично, или писать по указанному адресу.

Слайд 2

На этой лекции вы получите представление о истории развития компьютерной техники и тенденциях, узнаете для чего нужны параллельные программы и когда их имеет смысл использовать, а также получите ответы на ряд других вопросов.

Слайд 3-4

Перечисление основных разделов лекции

Слайд 5

На слайде представлены фотографии людей, которые на мой взгляд внесли наибольший вклад в развитие компьютерной техники, методов обработки и передачи информации. Думаю вам не составит труда вспомнить чем знамениты Алан Тюринг, Джордж Буль, Джон Нейман, Клод Шеннон. С остальными, наверное, будет сложнее. Нордберг Винер считается родоначальником кибернетики, Карл Бабедж известен как создатель арифмометра, а Генри Эдвард Робертс как создатель первого компьютера Altair 8800. Кстати для этого компьютера Бил Гейтс со своим одногруппником и написали Бэйсик, с которого и началась история компании Microsoft.

Слайд 6

Под обработкой информации была подведена научная составляющая и можно выделить две науки, которые имеют непосредственное отношение к теме нашего курса: информатика и кибернетика. Первая рассматривает фундаментальные основы получения и методов обработки информации, а вторая занимается изучением взаимодействующих систем.

Слайд 7

Для обработки информации создавалось множество инструментов, представленных на этом слайде. Здесь и счеты, известные еще с античных времен, и логарифмическая линейка, и упомянутый первый компьютер и современные вычислительные системы насчитывающие не

одну тысячу процессоров.

Слайд 8

Программа — это последовательность действий, направленных на вычисление некоторой функции. На слайде приведен известный алгоритм вычисления корней квадратного уравнения. Последовательная программа шаг за шагом по одной операции выполняет вычисления результата. И такая последовательная программа может выполняться на последовательной вычислительной машине, например Машине Тюринга.

Но всякая ли задача может быть выполнена на такой машине?

Слайд 9

Рассмотрим задачу моделирования плазмы.

Для получения приемлемого результата необходимо не менее миллиона частиц у каждой из которых есть заряд, масса, координаты в пространстве, скорости по трем измерениям, что уже на один шаг по времени требует не менее 32 мегабайт оперативной памяти. Но объем памяти не так критичен ведь сейчас даже простенький компьютер имеет не менее полгигабайта оперативной памяти. Сложность моделирования плазмы заключается в сложности алгоритма, которая является квадратом от числа частиц. Т.е. для миллиона частиц, в простейшем случае, надо рассчитать 10 в 12 степени взаимодействий для одного шага по времени, а таких шагов может быть сотни и тысячи. При частоте современны процессоров в гигагерцы на расчет данной задачи могут уходить недели.

Слайд 10

Задача моделирования плазмы далеко не единственная, которая требует больших вычислительных ресурсов, и на этом слайде представлены области, задачи в которых едва ли могут быть эффективно решены на простом однопроцессорном компьютере.

В частности биоинформатика является одной из областей, где требуется обрабатывать большие объемы информации и использование старых методов уже не эффективно. Именно по этому для вас и был организован этот курс.

Слайд 11

Возвращаясь к упомянутому выше квадратному уравнению и представленному на этом слайде дереву его решения можно заметить, что часть операций можно выполнять независимо друг от друга и при том параллельно. Можно одновременно вычислять квадрат числа **b** и выражение **4*a*c**. Можно параллельно вычислять квадратный корень и **a** умножать на **2**. Можно выделить и другие операции, которые могут выполняться параллельно.

Параллельная программа — это программа, которая позволяет среде исполнения часть операций выполнять параллельно.

Слайд 12

Один из способов реализовать параллельное выполнение инструкций — использовать

несколько арифметических логических устройств в процессоре. Часть операций выполняются на одном устройстве, часть на другом. Это параллелизм на уровне инструкций.

Слайд 13

Если вспомнить как происходило развитие процессорной техники и с чем был связан переход от CISC архитектуры к RISC, то можно предложить другой вариант для параллелизма на уровне инструкций. Так в RISC процессорах для выполнения команды ее надо выбрать из памяти, декодировать, выполнить, положить результат в память или в регистр. Все эти операции занимают фиксированное количество тактов и могут быть увязаны в конвейер. Если все складывается удачно и нет операций ветвления, которые приводят к срыву конвейера, то на каждом такте будет выполняться одна команда, что в несколько раз быстрее, если бы конвейер отсутствовал.

Слайд 14

Применение векторных операций — это еще один способ параллельной обработки данных. Так еще совсем недавно можно было слышать радостные возгласы от любителей компьютерных игр, которые приобретали процессоры, в которых была реализована поддержка инструкций 3DNow, MMX, SSE. Суть этих инструкций в том, что одна операция над 4-мя числами с плавающей точкой в одинарной точности может выполняться за такое же время как над одним. Сейчас инструкции SSE стали стандартом для процессоров как Intel так и AMD и различных инструкций насчитывается более сотни.

Слайд 15

Раньше мы все наблюдали рост частоты процессоров сопровождаемый сожалением по поводу того, что память не поспевает за процессором. К чему же это приводит? На слайде видно, что увеличение частоты процессора в два раза (в два раза сократилось время на вычисления) не дает общего прироста производительности в два раза. Все упирается в скорость доступа к памяти.

Слайд 16

Если для загрузки данных организовать конвейер, как делалось для обработки инструкций, то время программы может быть существенно сокращено и скорость работы памяти будет менее критична.

Слайд 17

Все вышеперечисленные (и прочие) подходы применялись и применяются в однопроцессорных машинах. Следующим шагом стало на одном кристалле размещение двух и более ядер процессора. Сейчас существуют процессоры SPARC имеющие по 8 ядер, каждое из которых способно обрабатывать до 8 потоков команд.

На слайде представлена архитектура 4-х ядерного процессора от компании AMD.

Стоит отметить, что большому росту количества ядер препятствует необходимость поддерживать кэши ядер в актуальном состояние ведь прежде чем ядро процессора соберется использовать данные из своего кэша, оно должно убедиться, что эти данные не изменило другое ядро в своем кэше или памяти.

Слайд 18

Вы обращали как люди общаются между собой? Один человек что-то говорит другому (посылает сообщение) и дожидается от него ответа. Второй человек получив это сообщение его обрабатывает и дает на него ответ (посылает ответное сообщение). И так далее. Иногда кто-нибудь может не дождавшись ответа продолжить говорить дальше. Это уже асинхронное взаимодействие, которое, как любят указывать в анекдотах, происходит у женщин, которые и слушают собеседницу и что-то ей говорят одновременно.

Идею, что компьютеры могут также общаться между собой по средством сообщений высказал еще Нордберг Винер.

Так следующим классом параллельных систем можно считать системы имеющие несколько процессоров.

Ахо же определил параллельную программу, как систему взаимодействующих процессов.

Слайд 19

Выделяют четыре типа параллелизма, которые представлены на слайде. Со вторым типом параллелизма (параллелизм на уроне инструкций) мы уже успели разобраться раньше. Разберемся с остальными.

Слайд 20

Параллелизм на уровне бит.

Очевидно, что увеличивая длину слова нам требуется меньше операций, которые процессор должен выполнить для вычисления значения с операндами, длинна которых больше этого слова.

Так, исторически 4-х битные процессоры были вытеснены 8-ми битными процессорами, а те в свою очередь 16-ти битными. До недавнего времени стандартными считаются 32-битные процессоры (и они являлись таковыми на протяжении около 20 лет) и сейчас им на смену приходят 64-х битне процессоры. Дальнейший рост разрядности слова кажется сомнительным, поскольку 32 и 64 бита позволяют описать большинство величин нашего мира.

Слайд 21

Параллелизм по данным использует подход разбиения всего множества данных на две непересекающиеся подобласти (на слайде область D разбивается на D' и D") и передача обработки каждой из подобластей разным процессорам. При этом типе параллелизма над каждым элементом из подобластей исходной области вычисляется одна и та же функция.

Слайд 22

В параллелизме по задачам же наоборот полностью различные задачи выполняются различными процессорами над одним или разными множествами входных данных. При этом данный тип параллелизма слабо масштибируется и ростом количества процессоров.

Слайд 23

Рассматривая различные системы Флинном была предложена их классификация по способу обработки инструкций. Может быть или один или несколько потоков команд, один или несколько потоков данных. Итого 4 класса систем.

Слайд 24

Под классом SIMD — одна инструкция, один поток команд, понимается «обыкновенный» компьютер, который выполняет последовательно инструкции шаг за шагом. Например Машина Тюринга.

Под классом SIMD — одна инструкция, множество данных, понимаются системы имеющие векторные операции, например те же самые SSE.

Под классом MIMD — множество данных, множество команд, понимаются те системы, которые одновременно могут выполнять различные команды.

Класс MISD — множество инструкций, один поток данных, в некотором плане является вырожденным. Кто-то относит к этому типу конвейеры.

Слайд 25

Помимо классификации Флинна, которая описывает однопроцессорные системы, существует классификация и для параллельных систем. Так выделяют системы с общей и распределенной памятью, многоядерные системы, кластеры, массивно параллельные компьютеры и GRID системы.

На слайде представлен многоядерный специализированный процессор IBM CELL BE и два варианта кластерных систем. Нижний — одна из стоек вычислительной машины BlueGene. Сама машина насчитывает более 100 тысяч процессорных ядер.

Слайд 26

Существуют и специализированные вычислительные системы. К таковым можно отнести системы на базе программируемых матриц (FPGA), вычислители на базе видеочипов (графические карты), векторные машины. На слайде фотография легендарной машины CRAY

Слайд 27

Естественно, что программировать параллельные компьютеры сложнее и параллельные программы помимо ошибок, присущих последовательным программам, содержат ошибки, которые связаны с взаимодействием процессов. К таким ошибкам относят мертвые блокировки (deadlock) и ошибки соревнования (Race-conditions). Кроме того параллельные программы обладают недетерминизмом выполнения, а их масштабируемость также может стать головной болью для программиста.

Слайд 28

Под недетерминизмом понимается особенность параллельной программы, когда заранее

невозможно сказать какая из функций в параллельных процессах начнет или закончит свое выполнение первой.

Ниже 4 варианта выполнения по времени одной и той же программы. Однозначно можно сказать, что функция **C** выполняется параллельно функциям **A** и **B**, которые в свою очередь выполняются последовательно.

Если при этом в параллельных процессах неакуратно использовать разделяемые ресурсы, то это может приводить к различным ошибкам.

Слайд 29

Одной из таких ошибок является ошибка соревнования.

На слайде представлен код для двух потоков/процессов, которые для своей области памяти вычисляют контрольную сумму для массива и результат записывают в общую переменную **sum**. Поскольку неизвестно заранее как и кто первым прочитает значение этой переменной и запишет результат, то гарантировать правильность результата невозможно. В данном случае нужна синхронизация на доступ к переменной **sum**.

Слайд 30

Мертвая блокировка, например, возникает тогда, когда один процесс ожидают ресурсы, захваченные вторым процессом, а тот их не отдает поскольку для завершения работы ему требуются ресурсы первого.

Как вариант борьбы с мертвыми блокировками кто-то может предложить при невозможности захватить все ресурсы освобождать уже захваченные и через некоторое время повторить попытку захвата снова. Но в таком случае мертвая блокировка может перерасти в живую, которая еще неприятнее поскольку поскольку работа программой не выполняется, а ресурсы центрального процессора используются по полной.

Одним из примеров мертвой блокировки и ее перерастанием в живую, с которой вам наверняка приходилось сталкиваться, это решение задачи о голодных философах.

Существует три необходимых и достаточных условий для возникновения мертвых блокировок и устранение одного из них исключает возможность появления подобных ошибок.

Слайд 31

Поскольку мы написали параллельную программу, то добавляя вычислительные ядра мы надеемся ожидать, что программа будет работать все быстрее. Возможность программы следовать данному правилу называется масштабируемостью. Однако не всегда добавление вычислительных ядер приводит к росту производительности. Вводится понятие как степень масштабируемости программ — это количество вычислительных узлов при котором добавление еще одного не дает существенного прироста в производительности программы.

Слайд 32

Масштабируемость программы до некоторой степени описывается законом Амдала, который

гласит, что ускорение программы в зависимости от количества используемых процессоров подчиняется формуле представленной на слайде.

Степень масштабируемости программы представлена на графике.

Таким образом зная долю параллелизма в программе (P) можно оценить на каком количестве процессоров и с каким ускорением ваша программа может выполняться.

Стоит отметить, что представленные графики не выходят на плато. Они могут после выхода на плато идти вниз. Например, два солдата выкопают окоп длинной два метра в два раза быстрее чем один, а вот рота, этот окоп будет капать существенно дольше, поскольку все солдаты уже будут мешаться друг другу и попросту простаивать.

Слайд 33

Системы с общей и распределенной памятью, с которыми вам придется в основном иметь дело обладают своими преимуществами и недостатками. Так программу для системы с общей памятью существенно легче написать и отладить чем программу для системы с распределенной памятью. Однако в тоже время система с общей памятью стоит дороже (на тоже количество ядер) чем система с распределенной памятью и хуже масштабируется.

Слайд 34

Для разработки параллельных программ используются различные языки программирования библиотеки и инструменты. Из представленных на слайде вам предстоит познакомится с OpenMP, POSIX threads и MPI. Первые два подхода ориентированы на системы с общей памятью, последний на системы с распределенной памятью — кластера.

Слайд 35

По завершению курса вы будете иметь представления об инструментах и средствах написания параллельных программ и, я надеюсь, будете применять и совершенствовать эти знания. Но прежде чем разрабатывать параллельную программу вам для себя стоит ответить на ряд вопросов:

1. Стоит ли эта программа того, чтобы быть распараллелина?
2. Почему я хочу сделать эту программу параллельной? Нехватка ресурсов одного компьютера? Ограничения по времени? Отвечая на этот вопрос вы можете сказать нужен ли вам кластер или система с общей памятью и какие должны быть ее параметры.
3. Какая часть программы должна быть распараллелина? Это во многом определяется алгоритмом.
4. Какая вычислительная система будет оптимальна для данной параллельной задачи?
5. А какая система у нас есть или появится в будущем?
6. Сколько времени я потрачу на написание параллельного кода? Ведь может статься так, что ваша программа должна исполниться лишь пару раз, а времени на ее параллелизацию вы потратите больше, чем все время выполнения последовательного кода.

Эти вопросы во многом пересекаются, но тем не менее они должны дать вам ответ для какой параллельной системы вы будете писать программу, с использованием каких алгоритмов и инструментов.

Слайд 36

Вопросы для самопроверки.