Параллельное программирование и суперкомпьютерный кодизайн

Смирнов А.В. asmirnov@srcc.msu.ru

Раздел 1. Теоретические основы высокопроизводительных вычислений

Раздел 1. План.

Теоретические основы высокопроизводительных вычислений

- Предпосылки для возникновения и использования параллельного программирования
- Параллельное программирование и высокопроизводительные вычисления
- Границы применимости параллельного программирования
- Способы оценки производительности и эффективности

Раздел 1. План.

Теоретические основы высокопроизводительных вычислений

- **Предпосылки для возникновения и использования параллельного программирования**
- Параллельное программирование и высокопроизводительные вычисления
- Границы применимости параллельного программирования
- Способы оценки производительности и эффективности

Параллельное программирование

Предпосылки для возникновения и использования параллельного программирования:

- Приложения требуют увеличения производительности компьютеров
- Производительность процессора и памяти ограничена физическими характеристиками применяемых материалов
- Многие задачи содержат независимые компоненты, которые могут решаться одновременно (т.е. параллельно)

Предпосылки для возникновения и использования параллельного программирования:

- **Приложения требуют увеличения производительности компьютеров**
- Производительность процессора и памяти ограничена физическими характеристиками применяемых материалов
- Многие задачи содержат независимые компоненты, которые могут решаться одновременно (т.е. параллельно)

Приложения требуют увеличения производительности компьютеров:

- Растет круг задач, в которых применяются вычисления
- Необходимость быстрой реакции вычислительной системы на запрос пользователя (включая системы реального времени)
- Ленивые программисты, которые рассчитывают на увеличивающуюся производительность компьютеров и пишут неэффективные программы

Приложения требуют увеличения производительности компьютеров:

- Растет круг задач, в которых применяются вычисления
- Необходимость быстрой реакции вычислительной системы на запрос пользователя (включая системы реального времени)
- Ленивые программисты, которые рассчитывают на увеличивающуюся производительность компьютеров и пишут неэффективные программы
- Всеми любимые нейросети, требующие...

Приложения требуют увеличения производительности компьютеров:

- Растет круг задач, в которых применяются вычисления
- Необходимость быстрой реакции вычислительной системы на запрос пользователя (включая системы реального времени)
- Ленивые программисты, которые рассчитывают на увеличивающуюся производительность компьютеров и пишут неэффективные программы
- Всеми любимые нейросети, требующие...
- ...постоянного перемножения больших матриц.

Нейросети (слайд сделан ChatGPT)

1. **Нейрон как линейная комбинация входов.**
   Каждый нейрон получает на вход несколько значений (например, пиксели изображения или результаты предыдущего слоя), умножает их на свои веса и складывает. Формула для выхода одного нейрона:

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b,$$

   где x_i — входы, w_i — веса, b — смещение.

Это обычное умножение матриц (вектор на матрицу).

Нейросети (слайд сделан ChatGPT)

1. **Нейрон как линейная комбинация входов.**
2. **Много нейронов сразу.**
   Если таких нейронов в слое много, то вместо того чтобы писать отдельную формулу для каждого, всё удобно записывать в *матричном виде*.
   - Вектор входов: X (размером $1 \times n$).
   - Матрица весов: W (размером $n \times m$, где m — число нейронов в слое).
   - Вектор выходов: Y (размером $1 \times m$).

   Тогда получается:
   $$Y = XW + b.$$

Это обычное умножение матриц (вектор на матрицу).

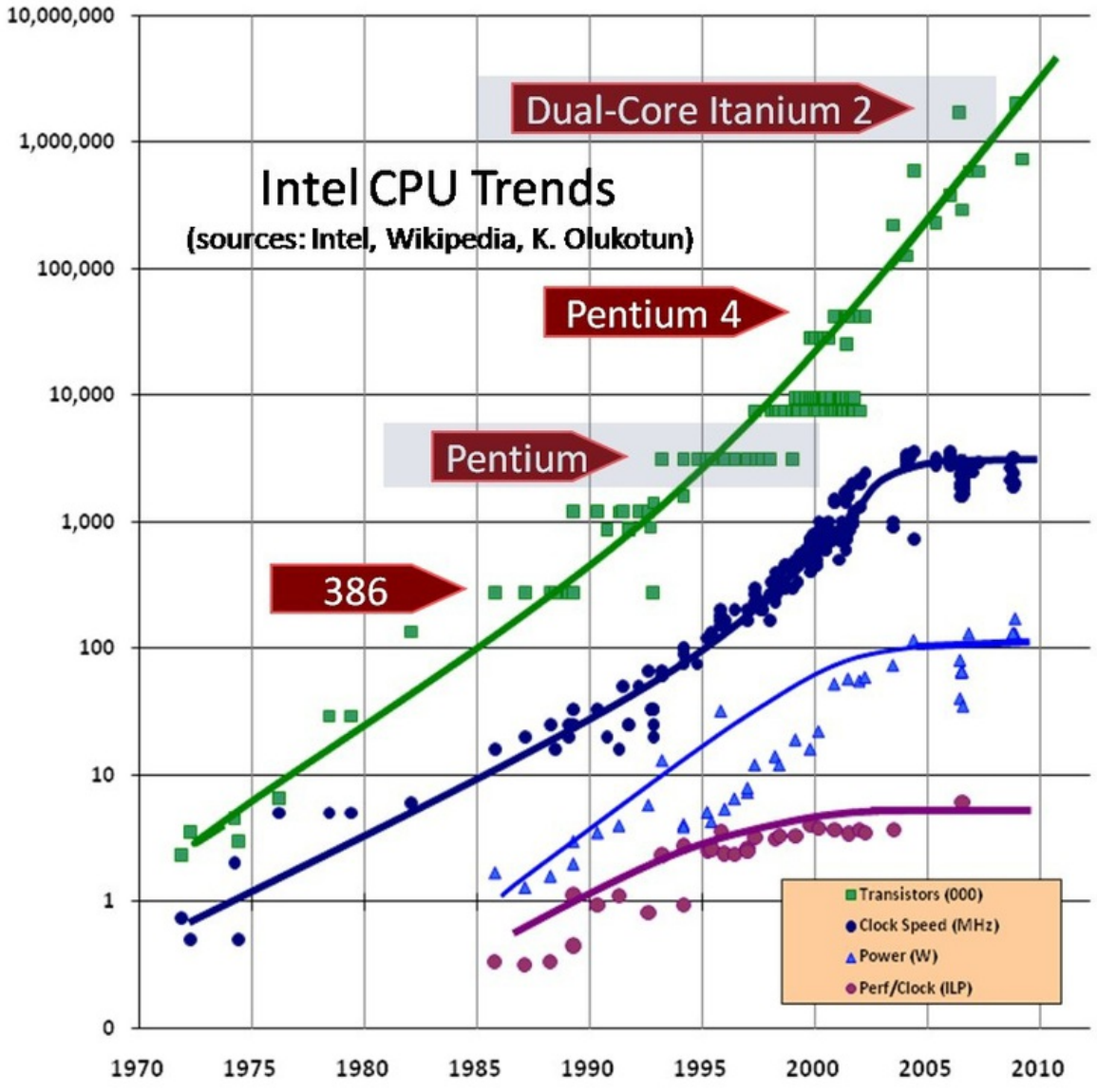
Предпосылки для возникновения и использования параллельного программирования:

- Приложения требуют увеличения производительности компьютеров
- **Производительность процессора и памяти ограничена физическими характеристиками применяемых материалов**
- Многие задачи содержат независимые компоненты, которые могут решаться одновременно (т.е. параллельно)

Производительность процессора и памяти ограничена физическими характеристиками применяемых материалов

В 1965 году Гордон Мур, один из сооснователей компании «Intel», сделал вывод на основе своих наблюдений о том, каждый год удваивается выгодное число транзисторов на одном кристалле. Немного позже эту закономерность назвали «законом Мура». Через 10 лет после публикации результатов наблюдений автор сделал уточнение: «Когда пройдет 1980 год, плотность транзисторов будет удваиваться через каждых два года». А коллега Гордона Мура Дэвид Хаус добавил: «Удвоение количества транзисторов в центральном процессоре в продолжение каждых 24 месяцев будет приводить к увеличению в два раза их производительности в продолжение каждых 18 месяцев».

Рост скорости процессоров

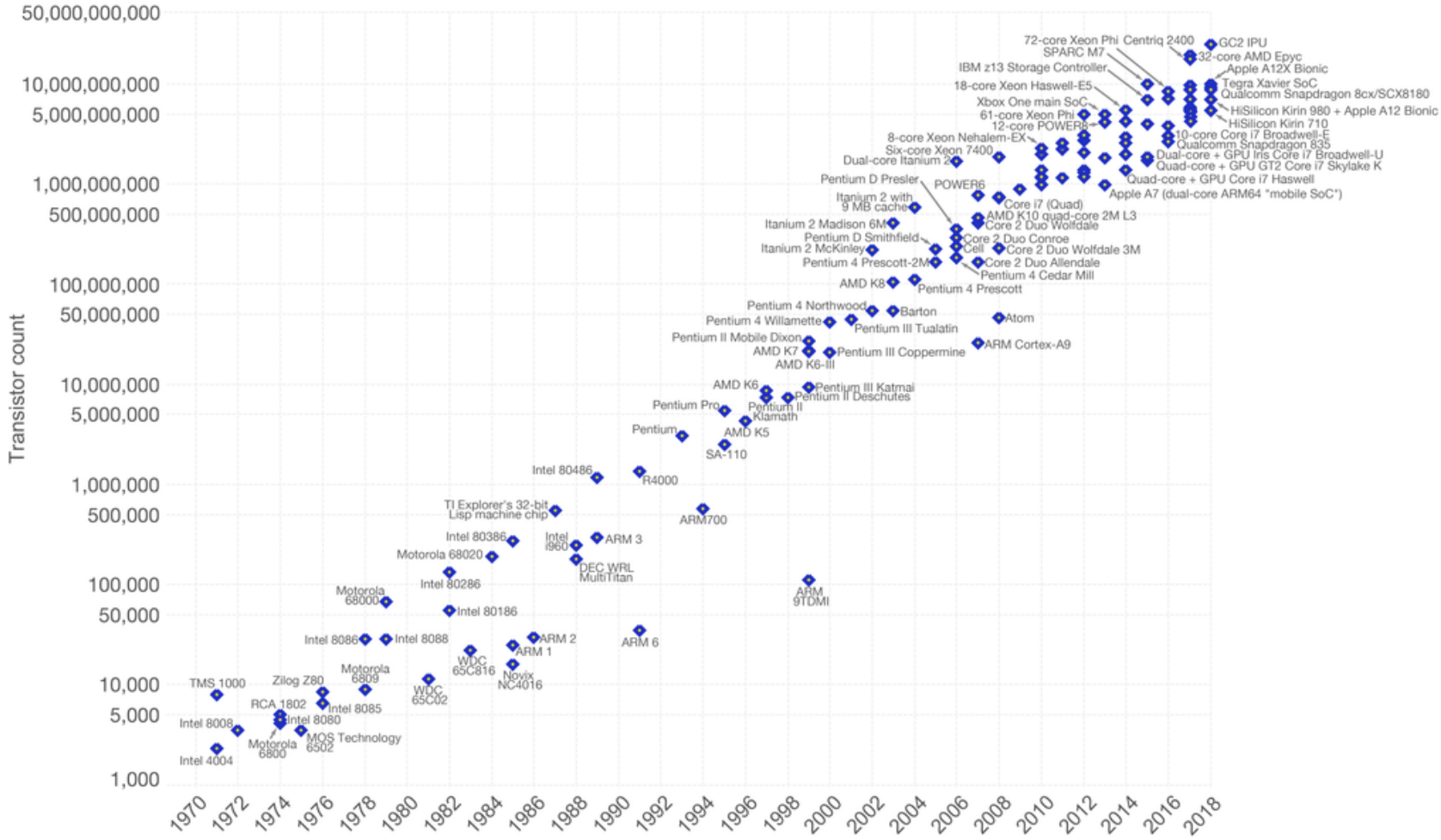


Рост скорости процессоров

Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

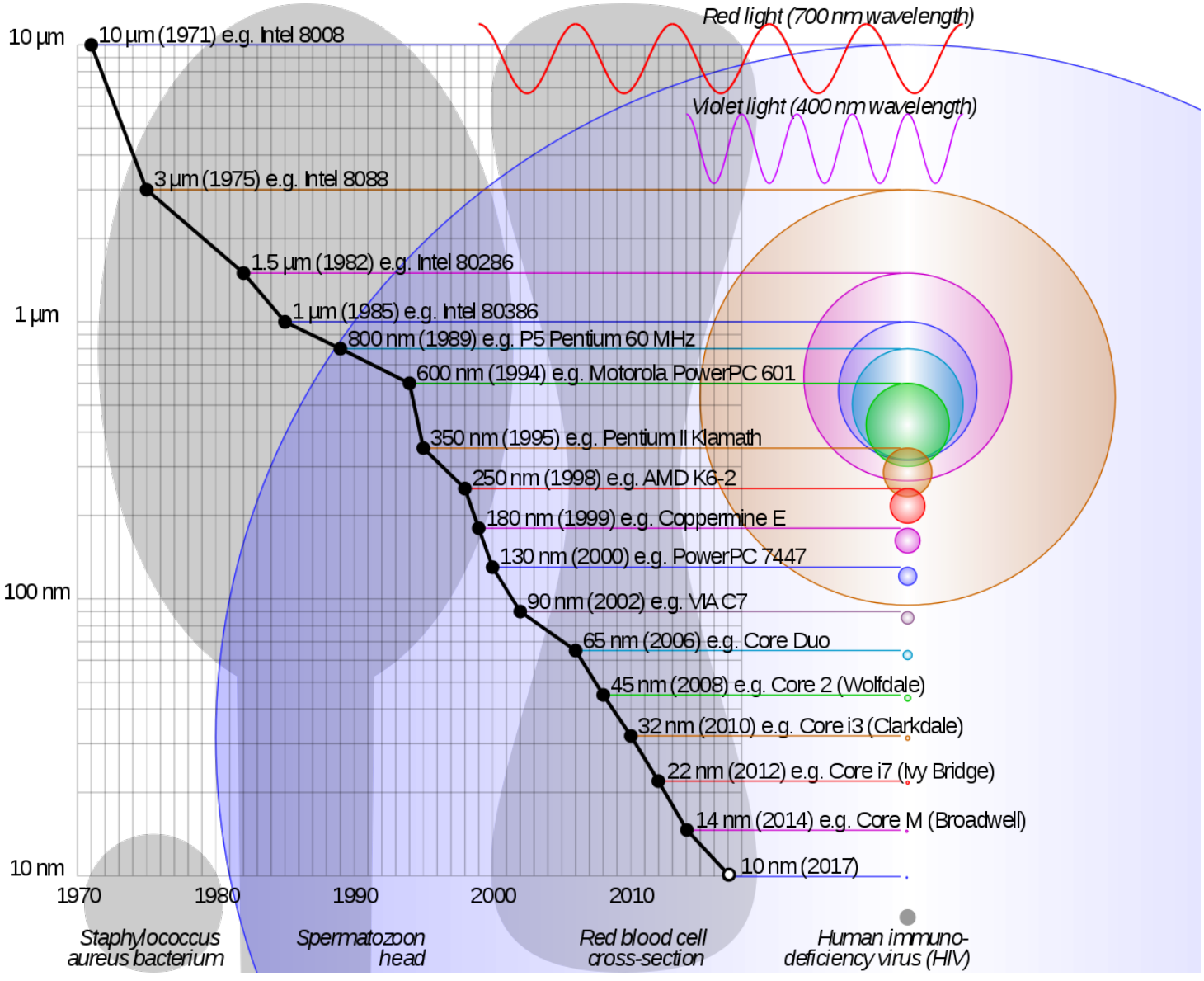
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

Our World in Data

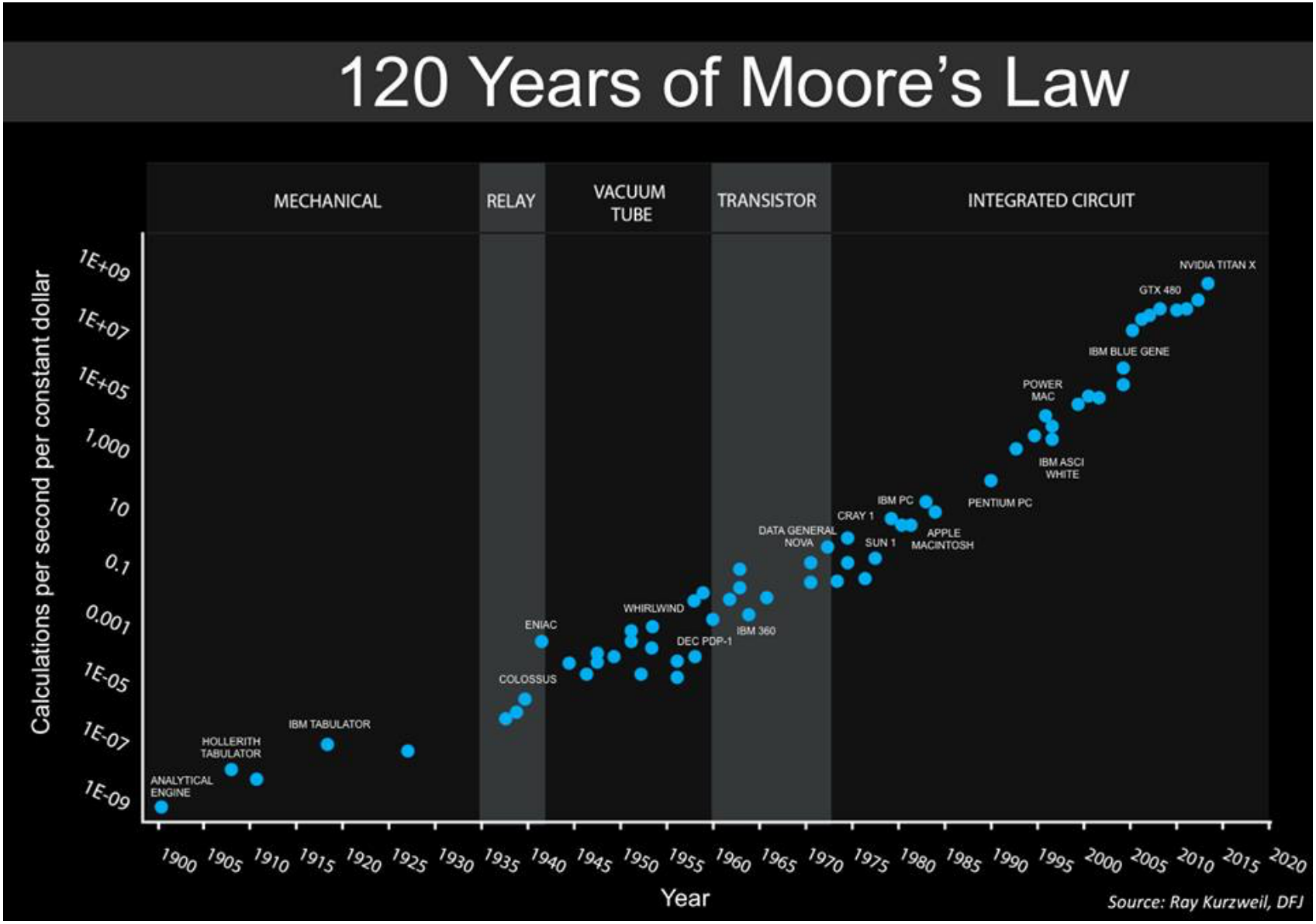


Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Уменьшение размера



Рост производительности?

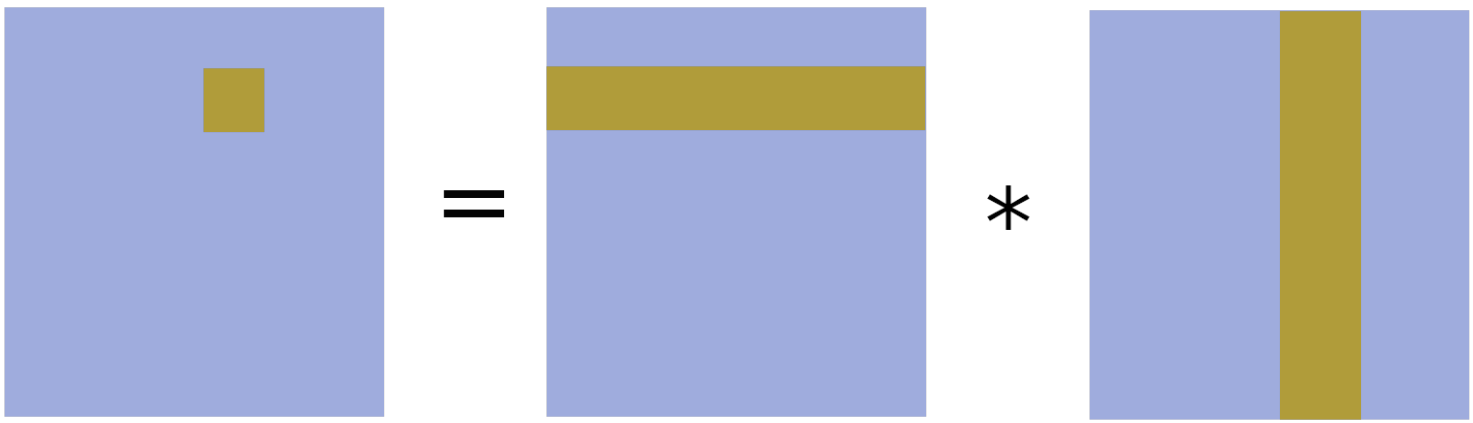


Мета-закон Мура :)

Предпосылки для возникновения и использования параллельного программирования:

- Приложения требуют увеличения производительности компьютеров
- Производительность процессора и памяти ограничена физическими характеристиками применяемых материалов
- **Многие задачи содержат независимые компоненты, которые могут решаться одновременно (т.е. параллельно)**

Многие задачи содержат независимые компоненты, которые могут решаться одновременно (т.е. параллельно):



```
std::random_device rd; std::mt19937 gen(rd());
std::uniform_real_distribution<double>
    uid(0., 1.);
constexpr size_t n = 32 * 1024 * 1024;
std::vector<int> a (n);
std::vector<int> b (n);
std::vector<int> c (n);
std::generate(a.begin(),a.end(),
    [&uid,&gen]()->double {return uid(gen);}
);
std::generate(b.begin(),b.end(),
    [&uid,&gen]()->double {return uid(gen);}
);
for (size_t i = 0; i != n; ++i) {
    c[i] = a[i] + b[i];
}
```

Какие есть варианты ускорить основной цикл?

- Потоки (руками или через OPENMP)

Какие есть варианты ускорить основной цикл?

- Потоки (руками или через OPENMP)
- SSE+AVX инструкции процессора

Какие есть варианты ускорить основной цикл?

- Потоки (руками или через OPENMP)
- SSE+AVX инструкции процессора
- OPENMPI

Какие есть варианты ускорить основной цикл?

- Потоки (руками или через OPENMP)
- SSE+AVX инструкции процессора
- OPENMPI
- Cuda (графические карточки)

Какие есть варианты ускорить основной цикл?

- Потоки (руками или через OPENMP)
- SSE+AVX инструкции процессора
- OPENMPI
- Cuda (графические карточки)
- ...

Раздел 1. План.

Теоретические основы высокопроизводительных вычислений

- Предпосылки для возникновения и использования параллельного программирования
- **Параллельное программирование и высокопроизводительные вычисления**
- Границы применимости параллельного программирования
- Способы оценки производительности и эффективности

Параллельное программирование и высокопроизводительные вычисления

- История развития вычислительных систем пришла к естественному решению – увеличивать число компонент оборудования, участвующего в решении задач.
- В частности, увеличивается число функциональных устройств одного процессора и общее число процессоров.
- Параллельные вычисления – вычисления на системах, содержащих несколько параллельно работающих вычислителей.

Спектр задач параллельного программирования

• Математическое моделирование:
– Газовая и гидро-динамика.
– Химическая физика.
– Процессы в полупроводниках.
– Имитационное моделирование в экономике.
– Биология.
• Оптимизация:
– Дискретное и линейное программирование.
– Общая задача нахождения экстремума.
• Оптимальный поиск:
– Дискретная оптимизация.
– Распознавание образов.
– Автоматическая верификация и доказательство теорем.
...

Повышение производительности

Как повышать производительность вычислений в наше время?

- Писать более эффективные программы :)

Повышение производительности

Как повышать производительность вычислений в наше время?

- Писать более эффективные программы :)
- Использовать параллельное программирование во всех его видах (суперкомпьютерный кодизайн)

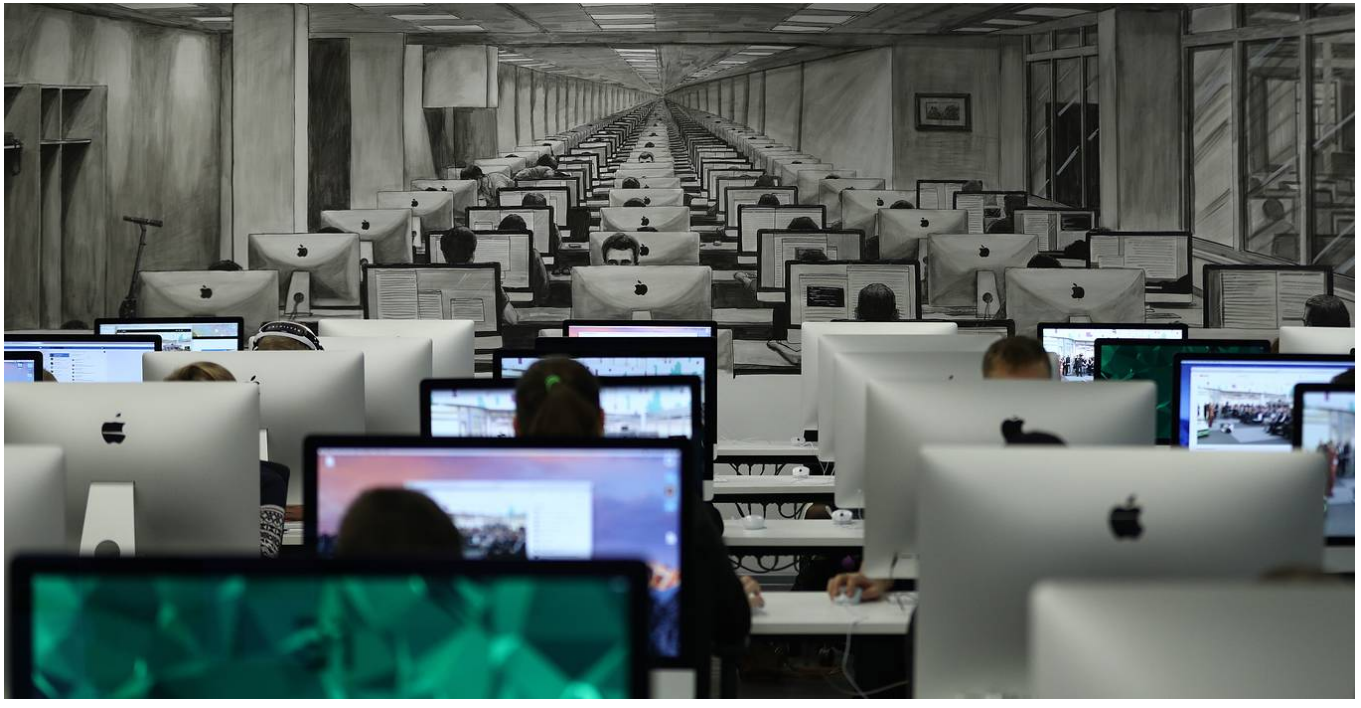
Повышение производительности

Как повышать производительность вычислений в наше время?

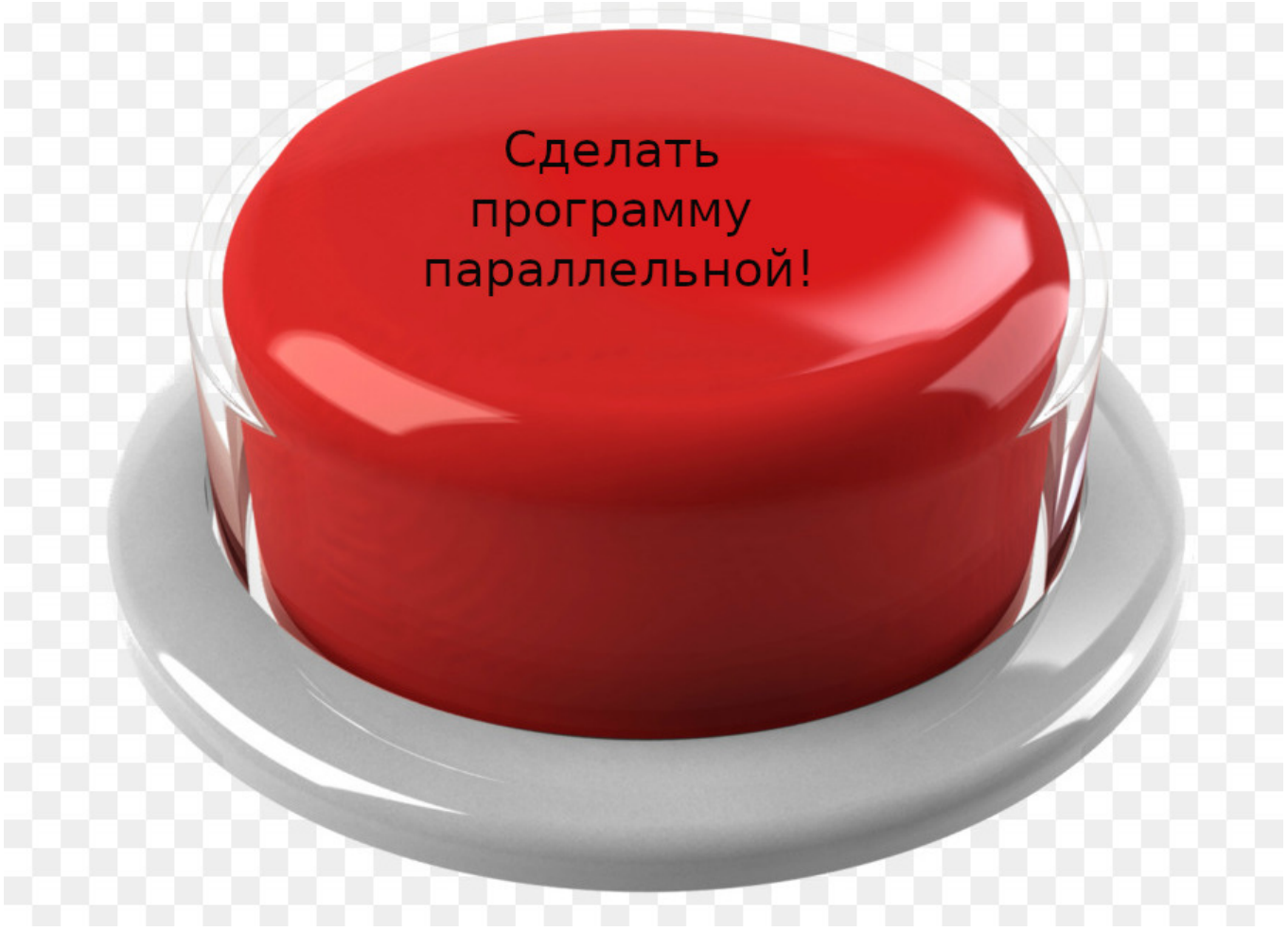
- Писать более эффективные программы :)
- Использовать параллельное программирование во всех его видах (суперкомпьютерный кодизайн)
- Разработать прорывную технологию

Параллельное программирование – это как?

Параллельное программирование – это как?



Параллельное программирование – это как?



Параллельное программирование – это как?

- Не существует универсального способа задействовать параллелизацию.
- Чтобы эффективно писать параллельные программы, необходимо понимать устройство той вычислительной системы, на которой вы собираетесь запускать программу.
- Также необходимо понимать, с какими типичными проблемами может сталкиваться параллельное программирование, и уметь их обходить.
- Не помешают и методы оценки того, насколько эффективно параллельное программирование для конкретной задачи

Раздел 1. План.

Теоретические основы высокопроизводительных вычислений

- Предпосылки для возникновения и использования параллельного программирования
- Параллельное программирование и высокопроизводительные вычисления
- **Границы применимости параллельного программирования**
- Способы оценки производительности и эффективности

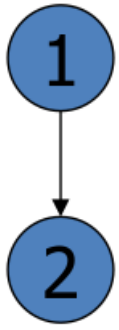
Границы применимости параллельного программирования

- Не всякая программа хорошо поддается параллелизации, возможности для параллельного исполнения могут быть ограничены
- Почему?

Зависимости в программах

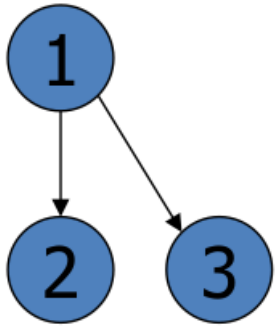
Зависимость по данным:

```
1: a = 1;
2: b = a;
```



Зависимость по управлению:

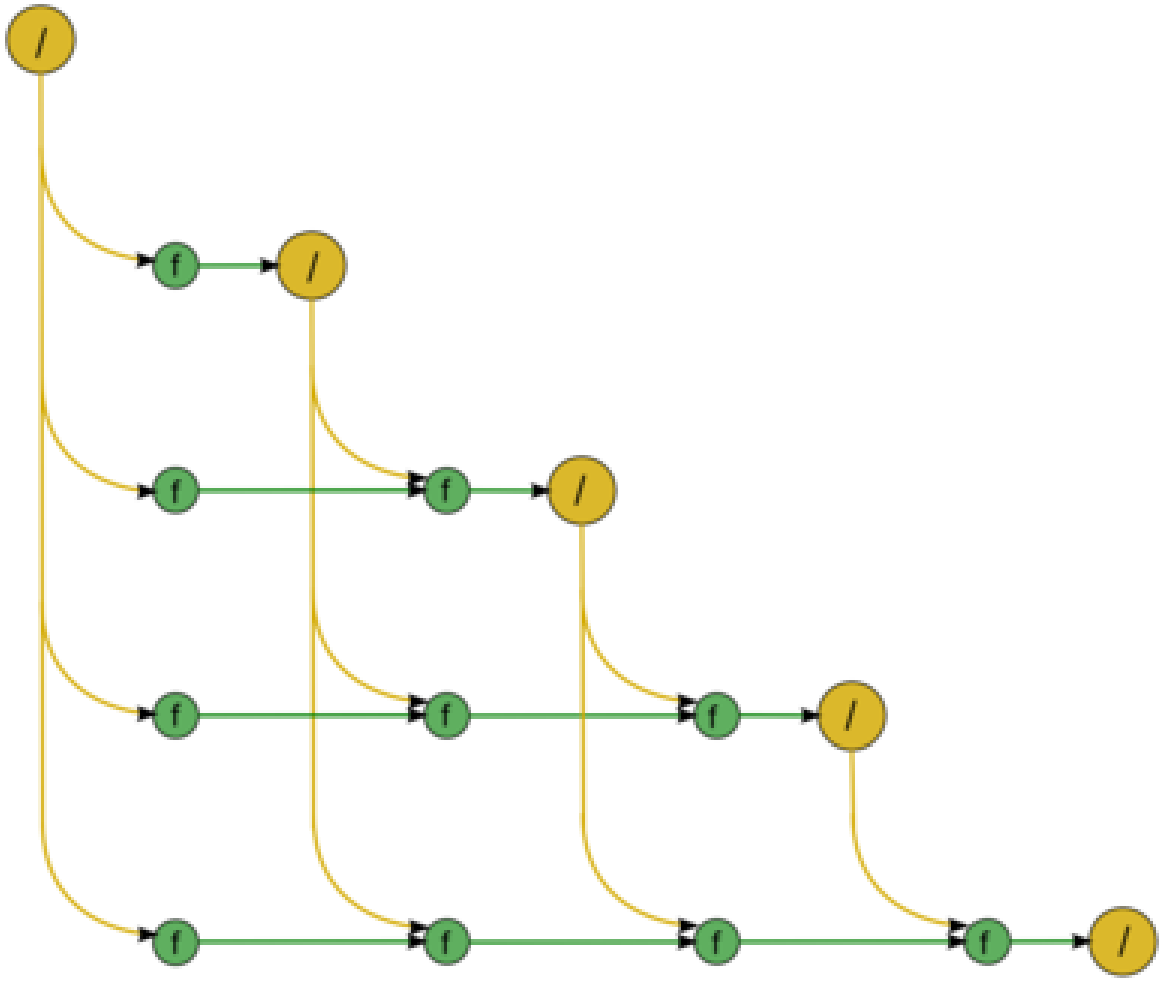
```
1: if(a) {
2:   x = c + d;
3:   y = 1;
4: }
```



Зависимости в программах

- Граф алгоритма (информационный граф) – ориентированный граф, состоящий из вершин, соответствующих операциям алгоритма, и направленных дуг, соответствующих передаче данных.

Зависимости в программах



Зависимости в программах

- Граф алгоритма (информационный граф) – ориентированный граф, состоящий из вершин, соответствующих операциям алгоритма, и направленных дуг, соответствующих передаче данных.
- Исследование информационного графа – один из методов поиска скрытого параллелизма в программе

Терминология, связанная с высокопроизводительными вычислениями

Терминология, связанная с высокопроизводительными вычислениями

- Ускорение (наблюдаемое) $S(n) = t(1)/t(n)$, где $t(1)$ – время последовательного выполнения программы, а $t(n)$ – время параллельного выполнения программы

Терминология, связанная с высокопроизводительными вычислениями

- Ускорение (наблюдаемое) $S(n) = t(1)/t(n)$, где $t(1)$ – время последовательного выполнения программы, а $t(n)$ – время параллельного выполнения программы
- Линейное ускорение – когда $S(n) = n$ – число участвующих ядер. Сверхлинейное ускорение – когда $S(n) > n$. Как такое может быть?

Сверхлинейное ускорение

Одной из причин такого явления может быть неравноправность выполнения последовательной и параллельной программ. Например, при решении задачи на одном процессоре оказывается недостаточно оперативной памяти для хранения всех обрабатываемых данных и, как результат, необходимым становится использование более медленной внешней памяти (в случае же использования нескольких процессоров оперативной памяти может оказаться достаточно за счет разделения данных между процессорами).

Сверхлинейное ускорение

Еще одной причиной сверхлинейного ускорения может быть нелинейный характер зависимости сложности решения задачи в зависимости от объема обрабатываемых данных. Так, например, известный алгоритм пузырьковой сортировки характеризуется квадратичной зависимостью количества необходимых операций от числа упорядочиваемых данных. Как результат, при распределении сортируемого массива между процессорами может быть получено ускорение, превышающее число процессоров. Источником сверхлинейного ускорения может быть и различие вычислительных схем последовательного и параллельного методов.

Закон Амдала (1967)

В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента

Пусть β – доля последовательных вычислений, а W – общий объем работы. Тогда

Закон Амдала (1967)

В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента

Пусть β – доля последовательных вычислений, а W – общий объем работы. Тогда

$$S(n) \leq \frac{W}{\beta * W + (1 - \beta) * W/n} = \frac{n}{\beta * n + (1 - \beta)} \leq \frac{1}{\beta}$$

Закон Амдала (1967)

В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого медленного фрагмента

Пусть β – доля последовательных вычислений, а W – общий объем работы. Тогда

$$S(n) \leq \frac{W}{\beta * W + (1-\beta) * W/n} = \frac{n}{\beta * n + (1-\beta)} \leq \frac{1}{\beta}$$

Так, если половина кода — последовательная, то общий прирост никогда не превысит двух.

Терминология (продолжение)

- Эффективность – отношение ускорения к числу процессоров. Показывает насколько эффективно используются аппаратные ресурсы.

$$E(n) = \frac{S(n)}{n} \leq 1$$

Терминология (продолжение)

- Эффективность – отношение ускорения к числу процессоров. Показывает насколько эффективно используются аппаратные ресурсы.

$$E(n) = \frac{S(n)}{n} \leq 1$$

- Масштабируемость – способность системы увеличивать свою производительность при увеличении числа ресурсов. Грубо говоря, программа масштабируема, если ее эффективность при увеличении числа ресурсов близка к единице (а ускорение близко к линейному). Программа может быть масштабируема до каких-то пределов, после чего терять масштабируемость (зависит от специфики задачи)

Накладные расходы

- Стоимость вычислений – число процессоров n, умноженное на время исполнения программы в параллельном режиме: $c(n) = n * t(n)$

Накладные расходы

- Стоимость вычислений – число процессоров n, умноженное на время исполнения программы в параллельном режиме: $c(n) = n * t(n)$
- Накладные расходы – это разница между стоимостью и временем исполнения на одном процессоре: $b(n) = c(n) - t(1) = n * t(n) - t(1)$

Накладные расходы

- Стоимость вычислений – число процессоров n, умноженное на время исполнения программы в параллельном режиме: $c(n) = n * t(n)$
- Накладные расходы – это разница между стоимостью и временем исполнения на одном процессоре: $b(n) = c(n) - t(1) = n * t(n) - t(1)$

$$E(n) = S(n)/n = (t(1)/t(n))/n) = t(1)/(t(n) * n)$$
$$= t(1)/(b(n) + t(1)) = 1 - b(n)/(b(n) + t(1))$$

Накладные расходы

- Стоимость вычислений – число процессоров n, умноженное на время исполнения программы в параллельном режиме: $c(n) = n * t(n)$
- Накладные расходы – это разница между стоимостью и временем исполнения на одном процессоре: $b(n) = c(n) - t(1) = n * t(n) - t(1)$

$$E(n) = S(n)/n = (t(1)/t(n))/n) = t(1)/(t(n) * n)$$
$$= t(1)/(b(n) + t(1)) = 1 - b(n)/(b(n) + t(1))$$

- Эффективность тем выше, тем ниже накладные расходы. Нужно бороться с ними и исследовать их причину

Что мешает эффективной параллелизации?

Подведем итоги

Что мешает эффективной параллелизации?

Подведем итоги

- Закон Амдала (последовательные части программы)

Что мешает эффективной параллелизации?

Подведем итоги

- Закон Амдала (последовательные части программы)
- Накладные расходы на коммуникации (латентность, пропускная способность)

Что мешает эффективной параллелизации?

Подведем итоги

- Закон Амдала (последовательные части программы)
- Накладные расходы на коммуникации (латентность, пропускная способность)
- Неравномерность загрузки (load balancing) процессоров

Что мешает эффективной параллелизации?

Подведем итоги

- Закон Амдала (последовательные части программы)
- Накладные расходы на коммуникации (латентность, пропускная способность)
- Неравномерность загрузки (load balancing) процессоров
- Предел декомпозиции данных.

Раздел 1. План.

Теоретические основы высокопроизводительных вычислений

- Предпосылки для возникновения и использования параллельного программирования
- Параллельное программирование и высокопроизводительные вычисления
- Границы применимости параллельного программирования
- **Способы оценки производительности и эффективности**

Способы оценки производительности и эффективности

Способы оценки производительности и эффективности

- МВС – многопроцессорные вычислительные системы
- HPC (High-performance computing) – высокопроизводительные вычисления
- MIPS (Million Instructions Per Second) – количество миллионов инструкций процессора в секунду.
- FLOPS (FLoating-point Operations Per Second) – количество операций с плавающей точкой в секунду.
- По русски так уже и пишем, флопсы. Есть килофлопсы, мегафлопсы, гигафлопсы, терафлопсы и даже экзафлопсы

Производительность пиковая и реальная

- Пиковая производительность – максимальное количество операций, которые вычислительное устройство может выполнить за единицу времени.
- Реальная производительность – количество операций, которое вычислительное устройство реально выполняет.
- Загруженность = (реальная производительность)/(пиковая производительность)

Способы оценки производительности

- Производительность системы может сильно зависеть от типа выполняемой задачи

Способы оценки производительности

- Производительность системы может сильно зависеть от типа выполняемой задачи
- Оценка производительности осуществляется с помощью специализированных тестов

Способы оценки производительности

- Производительность системы может сильно зависеть от типа выполняемой задачи
- Оценка производительности осуществляется с помощью специализированных тестов
- Для десктопных компьютеров обычно используется серия тестов, включая компрессию, обработку видео и оценка производительности в различных играх

Способы оценки производительности

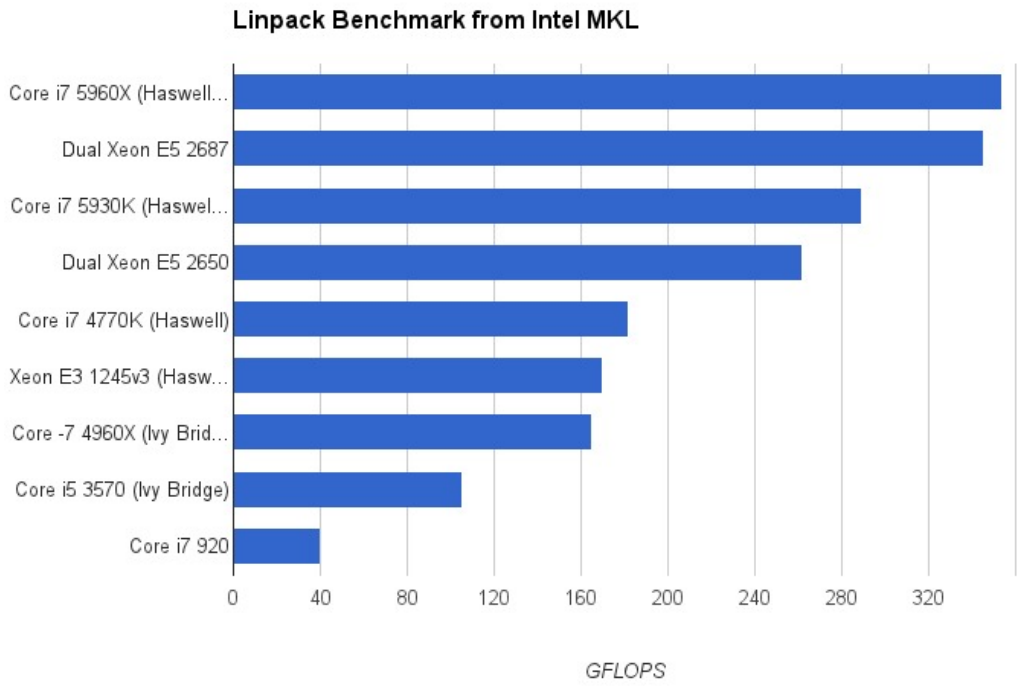
- Если рассматривается суперкомпьютер, то конечный пользователь обычно не использует его целиком, поэтому для него фактически не так важна производительность целиком, а ее отношение к количеству активных пользователей

Способы оценки производительности

- Если рассматривается суперкомпьютер, то конечный пользователь обычно не использует его целиком, поэтому для него фактически не так важна производительность целиком, а ее отношение к количеству активных пользователей
- Тем не менее, есть ряд общепринятых тестов, определяющих производительность вычислительного устройства, наиболее известный – LINPACK и его реализация HPL для суперкомпьютеров с использованием MPI

Производительность десктопных процессоров

Intel MKL на 2014 год



На данный момент достигнут 1 терафлопс на топовых десктопных процессорах – как у суперкомпьютера начала века.

Производительность десктопных процессоров

Intel MKL на 2014 год

Производительность суперкомпьютера преодолела экзафлопс в 2022 (Frontier)! Но об этом мы поговорим в следующем разделе, когда разберемся, как устроены суперкомпьютеры.

А что на мобильном?

Картинка многолетней давности:

	Samsung Galaxy S III	HTC One X (AT&T)	HTC One X (global)	HTC One S (global)	Samsung Galaxy Note
					
Quadrant (v2)	5,189	5183	4906	5053	3854
Linpack single-thread (MFLOPS)	51.337	103.77	48.54	103.88	64.3
Linpack multi-thread (MFLOPS)	101.606	214.53	150.54	222.22	95.66
NenaMark2 (fps)	58.8	58.7	47.6	61	32.8
NenaMark1 (fps)	59.8	58.6	59.5	60.8	56.6
Vellamo	2094	2350	1617	2452	901
SunSpider 9.1 (ms, lower numbers are better)	-	1709	1772.5	1742.5	2902

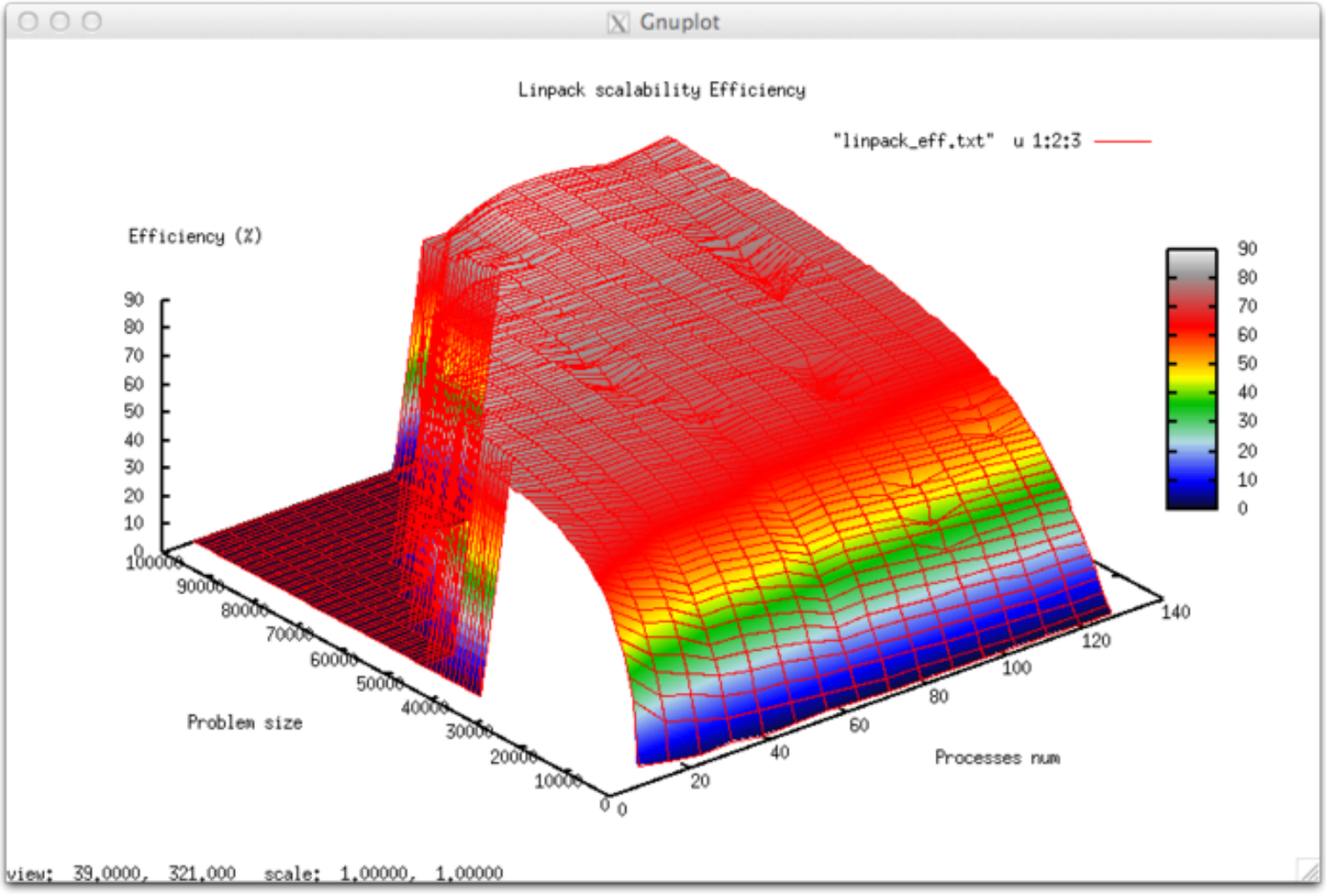
А что у вас? `http://linpack.hpc.msu.ru/`

Эффективность Linpack

Всегда ли программа вроде Linpack масштабируема?
Имеет ли смысл запускать подобные сравнения?

Эффективность Linpack

Информация с сайта `https://algowiki-project.org/`



Эффективность Linpack

Всегда ли программа вроде Linpack масштабируема? Имеет ли смысл запускать подобные сравнения?

- В данном случае необходимо правильно подобрать размер матрицы. Но в общем случае может быть куда больше сложностей с масштабируемостью!

Домашнее задание

Внимание! Задание!

- Требуется произвести замер эффективности теста Linpack для вашего вычислительного устройства
- Для этого необходимо, с одной стороны, рассчитать его теоретическую пиковую производительность, а, с другой стороны, прогнать тест Linpack на разных размерах матриц
- В качестве устройства может быть как мобильный телефон (ссылка на тест выше), так и ваш компьютер. Я не даю детальные инструкции, как запустить Linpack на компьютере, ищите в интернете! - но “бонусов” так получите больше.
- Если кто-то готов сам реализовать Linpack, будет очень много бонусов!
- Результаты заполняются на сайте курса.

Вопросы?