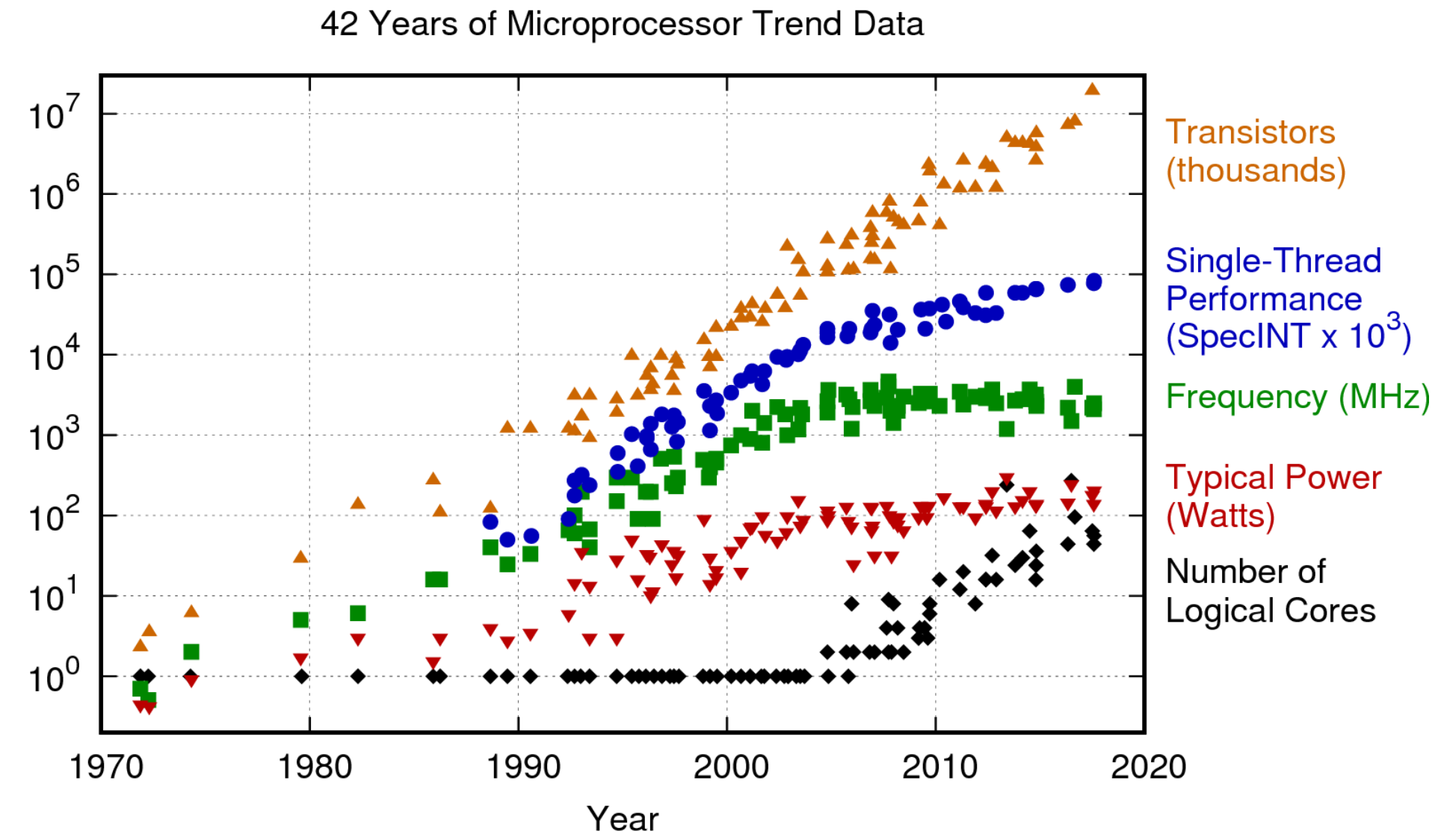
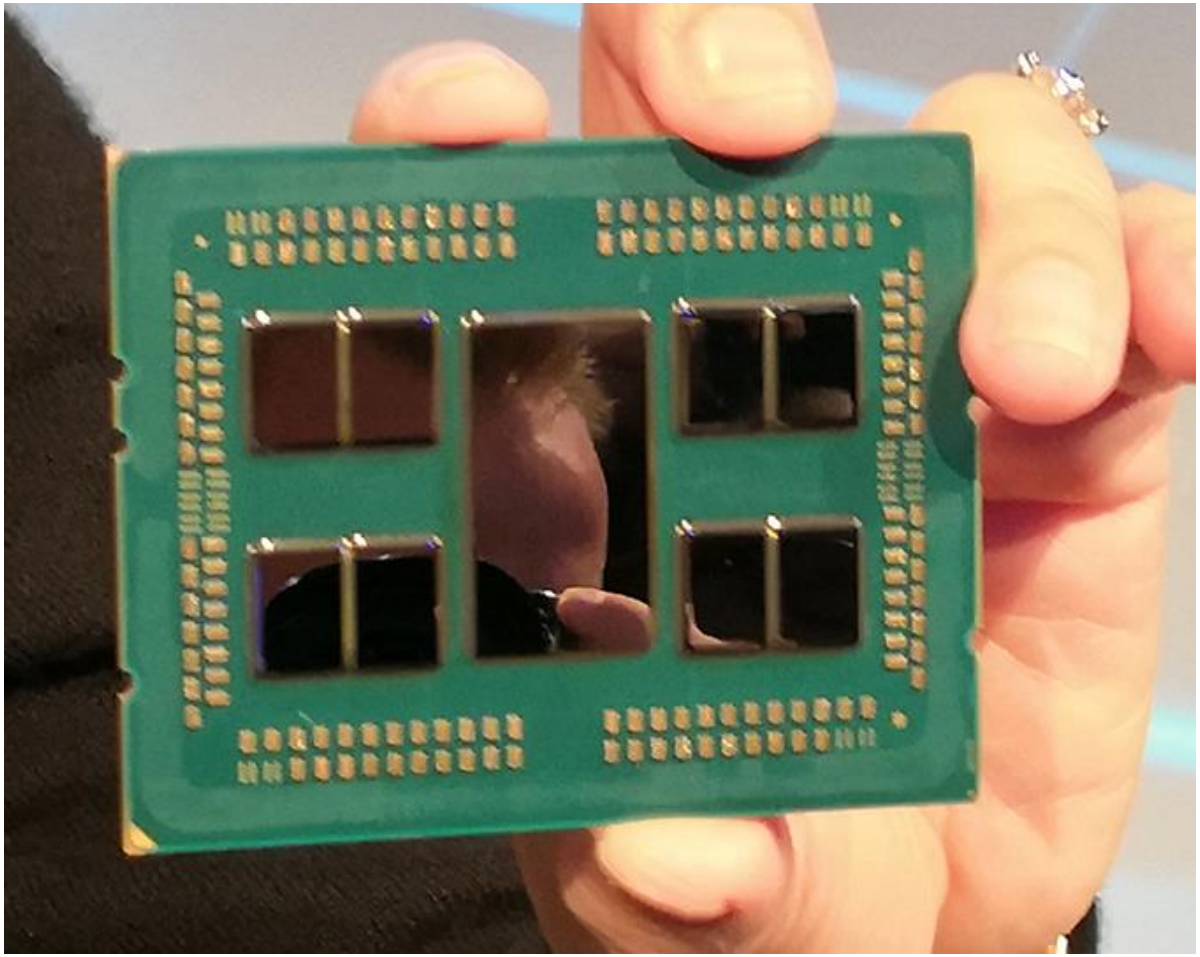
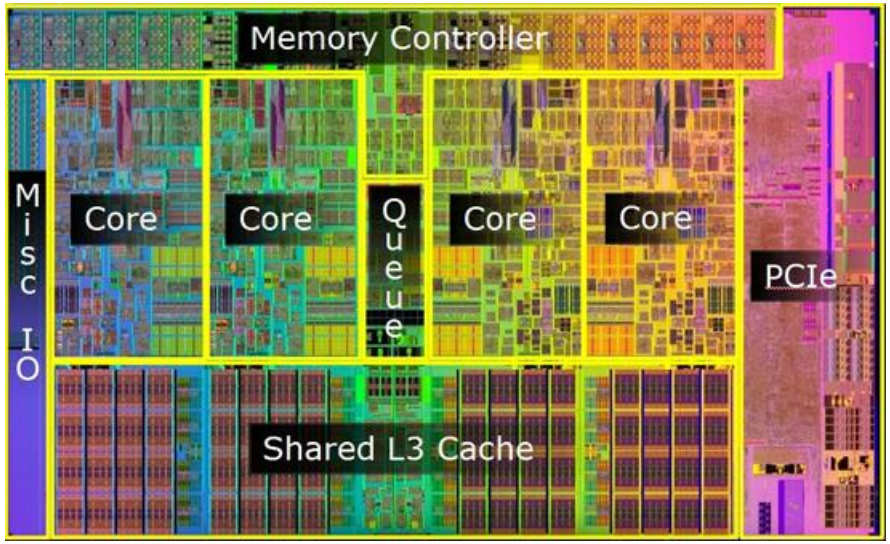
Многопоточные вычисления
Введение.

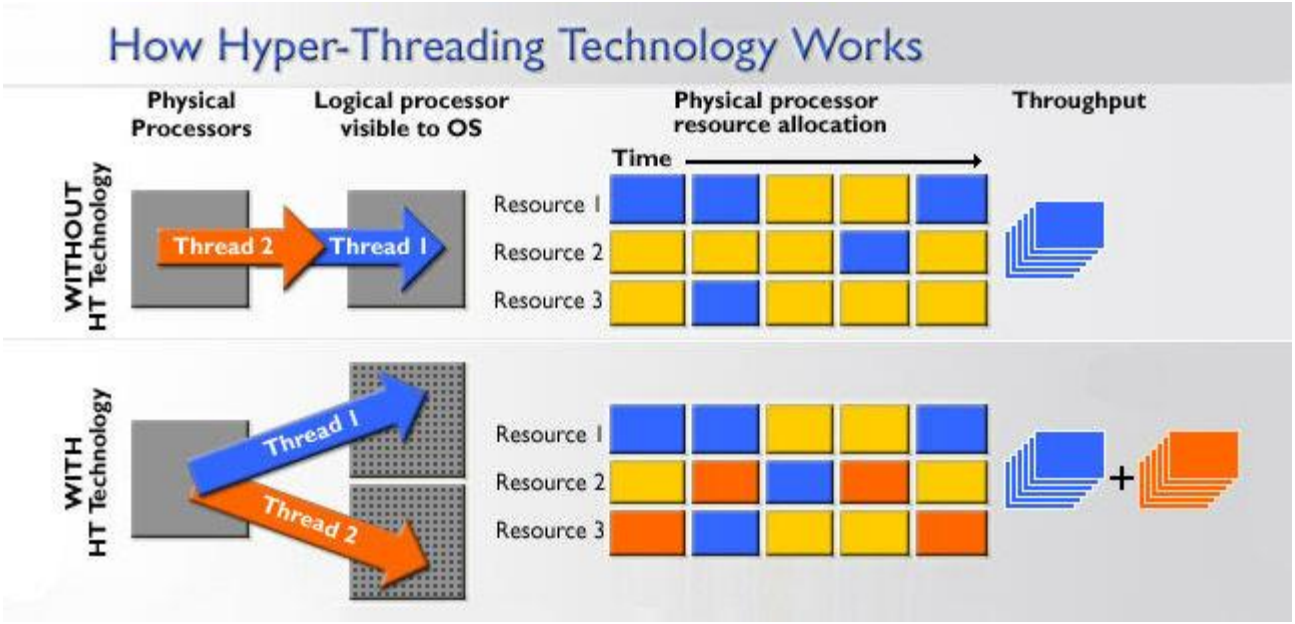


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp




Memory Controller
Misc IO
Core
Core
Queue
Core
Core
PCIe
Shared L3 Cache

Hyper threading



- Поток – многозначный термин
  - Поток выполнения
  - Поток задач
  - Поток ввода-вывода

Базовые понятия

- Процесс (process)
- Поток выполнения (thread)
  - Поток ядра
  - Пользовательский поток
- Волокна (fiber)
- Многозадачность (свойство ОС, потоковая и процессная)
- Многопоточность (свойство ОС или программы)

Многопоточность платформы (процессора)

- Временная – только один поток на конвейере
  - Крупнозернистая (*Coarse-grained multithreading*, *Blocked multithreading)*
  - Тонкозернистая (*Fine-grained multithreading*, *Interleaved multithreading*)
- Одновременная – несколько потоков на конвейере (*Simultaneous Multithreading)*

Модели многопоточности

- 1:1 – потоки ядра
- N:1 – пользовательские потоки
- N:M – смешанная

Контекст

Контекст – совокупность собственных ресурсов.

Контексты различных единиц выполнения:

Процесс: память, дескрипторы файлов и устройств, объекты ядра, окна, ….

Поток: стек, регистры процессора и (если нужно) память.

Волокна: иногда, память, либо нет собственных ресурсов (использует ресурсы потока, в котором работает)

Проблемы многопоточных программ

- Гонка - два потока одновременны пытаются выполнить неатомарный доступ к данным.
  - решение: примитивы синхронизации; неблокирующие структуры данных.
- Блокирующий ввод-вывод – актуально для пользовательских потоков.
  - решение: использование асинхронного ввода-вывода; M:N многопоточность
- Усложнение отладки: трудно воспроизводимые ошибки вероятностного характера, сложность пошагового выполнения

Зачем потоки?

- Дробление и распараллеливание вычислений (сокращение времени обработки)
- Одновременная обработка нескольких запросов (повышение производительности)
- Проблема ожидания
  - Памяти, ввода-вывода, …
- Более полное использование ресурсов

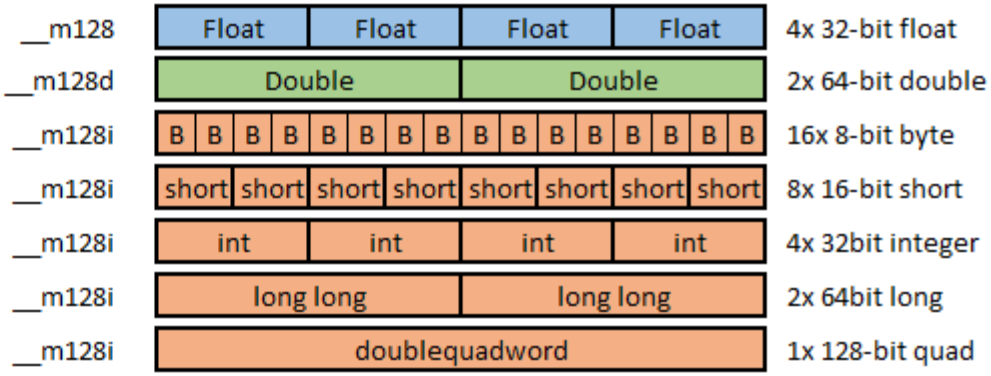
Другие методы решения тех же проблемм

- Векторизация вычислений **(SIMD)**
  - MMX, SSE, AVX
- Специальные процессоры
  - Вычисления на видеокарте (GPGPU)
  - Программируемые (FPGA)
  - Обработка сигналов (DSP), заказные (ASIC)
- Ко-рутины

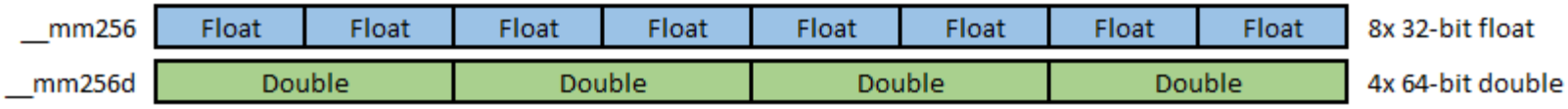
Векторные операции (SIMD)

- Одна и та же операция применяется к вектору(-ам) чисел
- Хорошо для обработки больших массивов данных (чисел и немного текста)
- Поддерживается большинством современных процессоров, включая ARM
- Данные должны быть выровнены
- Ветвления и условия не поддерживаются

SSE Data Types (16 XMM Registers)



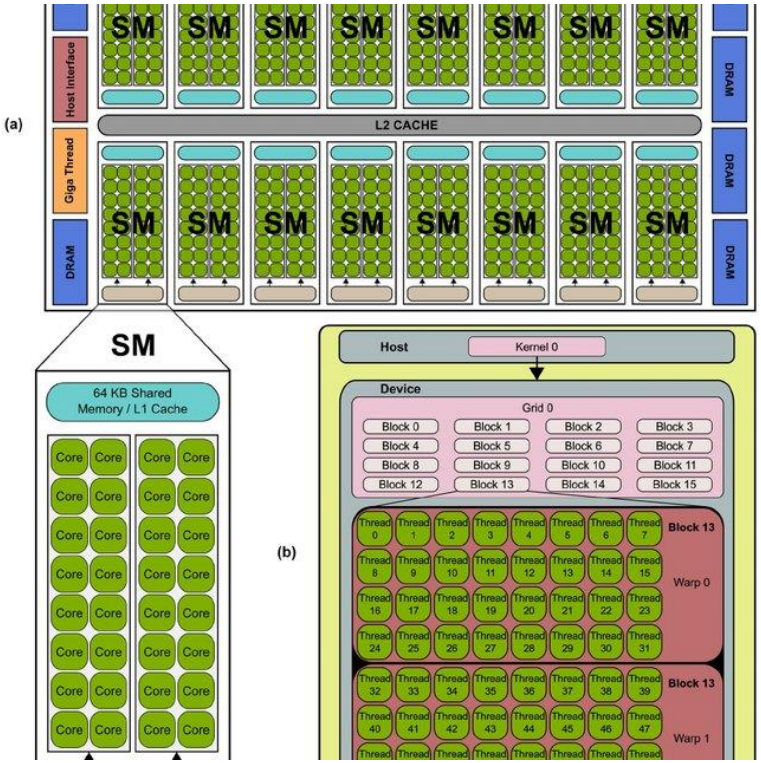
AVX Data Types (16 YMM Registers)



__mm256i *256-bit Integer registers. It behaves similarly to __m128i.Out of scope in AVX, useful on AVX2*

GPGPU

- почти как обычные ядра,
  но много
- Поддержка условий, ветвлений
- Нет поддержки примитивов
  синхронизации
- более сложный стэк технологий
  (компилятор, язык,
взимодействие)
- память, отдельная от CPU
- Идеально для обработки огромных
  массивов данных



Ко рутины

- специальным образом организованные модули
- могут сохранять состояние передавать управление
- «Легкая многопоточность»
- поддерживаются не всеми языками программирования

Эффективность распараллеливания

Ограничители скорости при распараллеливании

- Последовательные фрагменты кода
  - Управление
  - Инициализация, чтение и вывод данных
  - Агрегация результатов
- Накладные расходы
  - Запуск потоков
  - Обмен данными (исходными и результатом, особенно для массивов)
  - Пользовательская синхронизация
- Производительность памяти и кеша
- Файловый ввод-вывод, работа с устройствами
- Работа библиотеками, не поддерживающими одновременные вызовы из нескольких потоков
  - ограничение скорости вызывается при защите таких вызовов синхронизацией

Пример: типичная лабораторная

```
double t_start = omp_get_wtime();

int counter = 0;

int *m = new int[N];
for (int i = 0; i < N; ++i)
    m[i] = rand() % 10;

#pragma omp parallel for
for (int i = 0; i < N; ++)
    if (m[i] == 0)
        counter ++;

delete[] m;
double t_end = omp_get_wtime();

cout << "counter:" << counter << ebndl;
cout << "time: " << t_end - t_start << endl;
```

rand()

Линейный конгруэнтный генератор (побочный эффект — глобальная переменная):

```
#define RAND_MAX 0xffffffff
static unsigned long int next = 1;
int rand(void) {
next = next * 214013  + 2531011 ;
return (unsigned int)next % RAND_MAX;
}
```

Закон Амдала

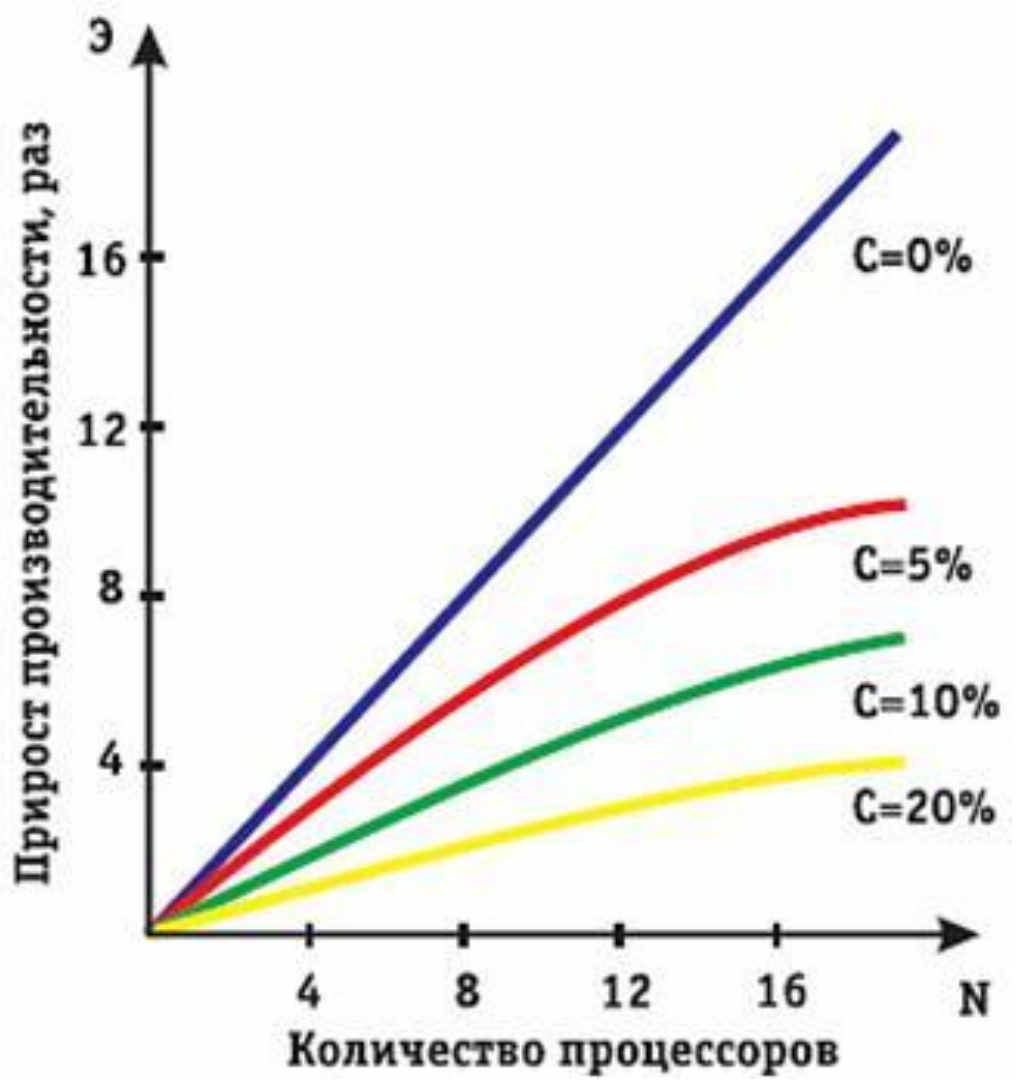
α - доля последовательных вычислений

p - число вычислительных узлов

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

α p	10	100	1000
0	10	100	1000
10%	5.263	9.174	9.910
25%	3.077	3.883	3.988
40%	2.174	2.463	2.496

Закон Амдала



Пример 2: сортировка подсчетом

```
void count_sort(int* a, int N, int M) {
    int counters[M];
    for (int i = 0; i < M; i++)
        counters[i] = 0;
    for(int i = 0; i < N; i++)
        ++ counters[a[i]];
    int k = 0;
    for (int i = 0; i < M; i++)
        for (int j = 0; j < counters[i]; j++, k++)
            a[k] = i;
}
```

Сложность алгоритма: O(N + M)

Рекомендации

- Выберите оптимальный алгоритм, не надо тратить время на оптимизацию и распараллеливание заведомо неэффективных алгоритмов
- Оцените потенциал распараллеливания, стоит ли оно того
- Оптимизировать или распараллеливать? Или одновременно?
- Работу с файлами и устройствами лучше делать однопоточным, иногда это единственный возможный вариант

Работа с потоками в WinAPI

- Handle CreateThread(ThreadAttributes, StackSize, StartAddress, Parameter, CreationFlags, ThreadId);
  - Pointer to variable!!!
- TerminateThread
- SuspendThread
- ResumeThread
- Sleep(milliseconds)

Замеры времени

- Прогрев
- Единый запуск
- Замеры времени
  - напр, omp_get_wtime()
- Прогонять несколько раз
- Release…

Подходы к распараллеливанию

Виды распараллеливания вычислений

- Thread-based (почти везде)
  - Process based (python)
- Task-based (например, omp tasks)
- Async model (например, .NET async/await)

Thread-based

- Поток создается на основе пользовательской процедуры (точка входа, потоковая функция)
- Указанная процедура работает параллельно остальным потокам
- По завершении функции, поток завершается
- Передача данных в/из потока – через общую память, параметр(ы) функции и механизмы IPC
- Трудоемко, но полный контроль над потоками
- Можно реализовать сложные сценарии взаимодействия потоков

Task-based programming

- Выделяются относительно небольшие задачи, которые отправляются менеджеру задач
- Менеджер использует потоки (обычно, из пула) для выполнения задач и передает назад результаты их выполнения
- Нет контроля над потоками, но менеджер берет на себя передачу данных, упрощается синхронизация

Высокоуровневые библиотеки

- OpenMP, MPI, TBB, …

Автоматическое распараллеливание и векторизация

Потоки в C++

- WinApi CreateThread (специфично для Windwows)
- std::thread (стандарт)
- OpenMP
- Куча библиотек для потоков и задач