



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО

ЦЕНТР КОМПЕТЕНЦИЙ ONEAPI В ННГУ

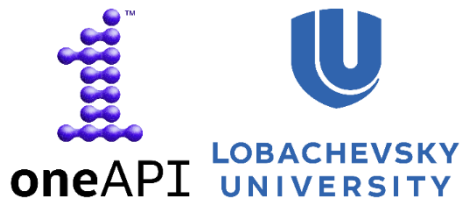
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО
ЦЕНТР КОМПЕТЕНЦИЙ oneAPI в ННГУ
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

***ВВЕДЕНИЕ В АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ
И ОПТИМИЗАЦИЮ ПРОГРАММ***

**Метрики производительности,
методика их сбора и анализа**



Волокитин В.Д.,
Мееров И.Б.

Содержание

- ❑ Производительность аппаратных средств
- ❑ Производительность программного обеспечения
- ❑ Анализ производительности
- ❑ Roofline-модель

ПРОИЗВОДИТЕЛЬНОСТЬ АППАРАТНЫХ СРЕДСТВ

Основные метрики

- ❑ **Тактовая частота (GHz)** – скорость работы аппаратного средства (CPU, памяти)
- ❑ **Вычислительная скорость (GFLOPs)** – количество операций, выполняемых за секунду
 - Целочисленные, с плавающей запятой (single precision, double precision)
- ❑ **Пропускная способность памяти (GB/s)** – объем передаваемых данных за секунду
 - Есть ли разница между чтением и записью?
 - NUMA-архитектура
- ❑ **Мощность (Watt)** – количество потребляемой энергии
- ❑ **Производные метрики:**
 - Flop/Byte, Flop/Watt ...

Закон Мура

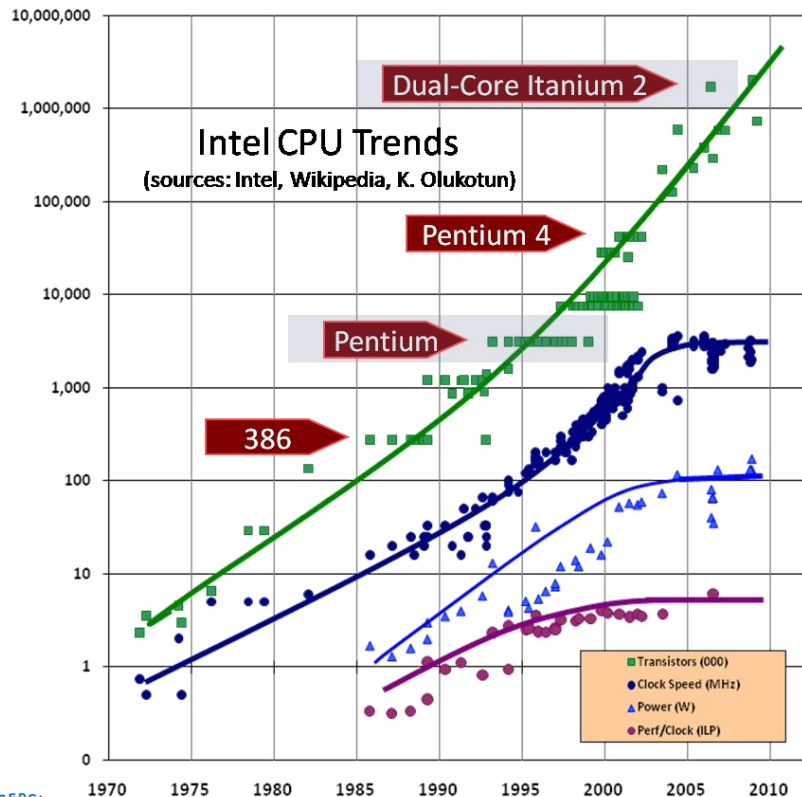
□ Закон Мура:

- Количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 18 месяцев (1965 год)
- В 1975 Гордон Мур изменил срок удвоения на 24 месяца

□ Что это означает на самом деле?



Эволюция процессоров



- ❑ До сих пор закон Мура правдив, но он распространяется только на транзисторы
- ❑ Закон Мура не распространяется на:
 - Скорость памяти
 - Мощность
 - Количество инструкций на один такт (ILP)

Источник: <http://www.gotw.ca/publications/concurrency-ddj.htm>

Теоретический пик производительности

- Пиковая производительность вычислений:

$$P * C * Vec * ILM * Clock,$$

где:

P – количество процессоров;

C – количество ядер на процессоре;

Vec – длина векторного регистра;

ILM – количество инструкций на один такт;

$Clock$ – частота процессора.

- Пиковая пропускная способность памяти:

$$MFreq * BPC * MWidht,$$

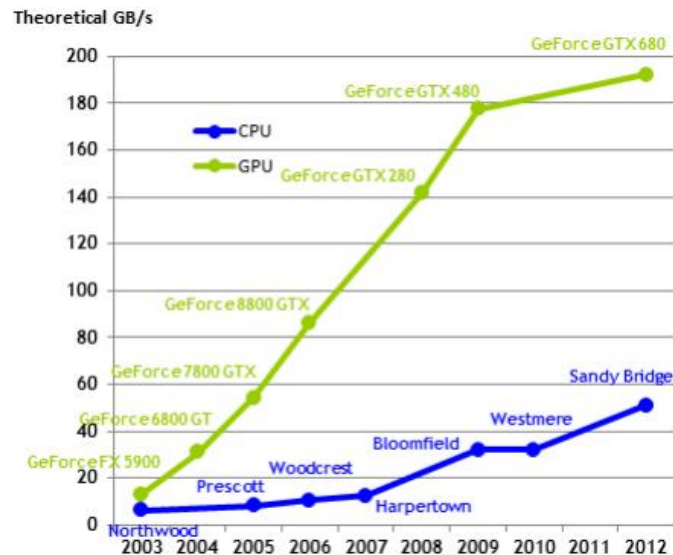
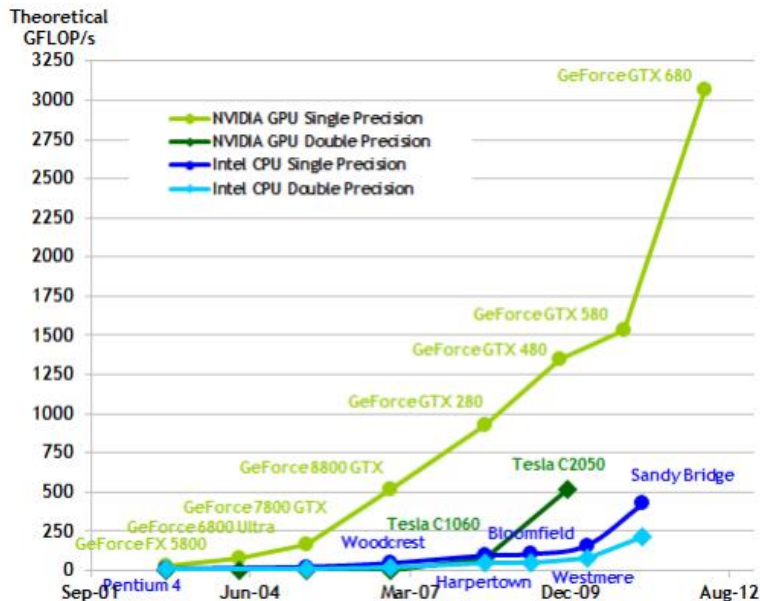
где:

$MFreq$ – частота памяти (не путать с частотой процессора);

BPC – количество каналов памяти;

$MWidht$ – ширина шины.

GPU vs. CPU



□ Все ли так хорошо на GPU?

Источник: Nvidia Cuda C Programming Guide version 4.2

Реальная производительность

- ❑ Пиковая производительность достигается:
 - При 100% задействовании вычислительных блоков
 - При 100% параллелизме
 - При передаче данных с максимальной пропускной способностью (на всех уровнях)
- ❑ Какая производительность в реальности?
 - Ни одно приложение не может работать с пиковой производительностью
- ❑ Как оценить реальную производительность системы?
 - Бенчмарки (Linpack, HPCG – высокопроизводительная линейная алгебра, Graph – работа с графами, обход в ширину)

ПРОИЗВОДИТЕЛЬНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Метрики производительности ПО

❑ **Пользователи** (Насколько хорошо работает приложение?):

- Время работы
- Ускорение
- Масштабируемость

❑ **Разработчики** (Можно ли сделать лучше?):

- Вычислительная сложность
- Близость к пиковой производительности

❑ **Заказчики** (На сколько хорошо используется моя инфраструктура и тратится мой бюджет?):

- Использование ресурсов (вычислительных, денежных и т.д.)

Ускорение

□ Закон Амдала:

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

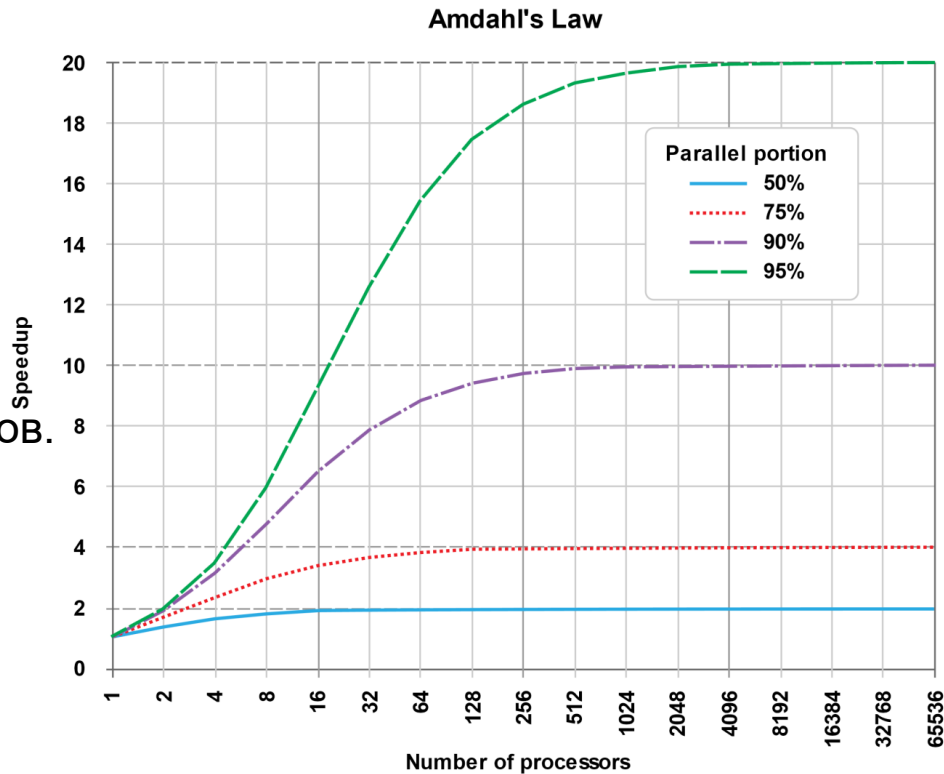
где:

S_p - ускорение параллельного кода;

α – доля последовательного кода;

p – количество параллельных ресурсов.

Источник: https://en.wikipedia.org/wiki/Amdahl%27s_law



Ускорение (2)

□ **Ускорение** на практике:

$$S = \frac{T_s}{T_p},$$

где:

T_s – время работы последовательной программы;

T_p – время работы параллельной программы.

□ Какое бывает ускорение:

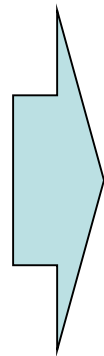
- $S < p$ – сублинейное
- $S = p$ – линейное

Возможно ли $S > p$ – сверхлинейное?

Накладные расходы

- ❑ Межпроцессорные взаимодействия:
 - Коммуникации
 - Синхронизации
- ❑ Дисбаланс нагрузки:
 - Неравномерное распределение задач

- ❑ Дополнительные накладные расходы:
 - Выделение памяти
 - Распределение данных
- ❑ Как оценить накладные расходы?



Простой в работе

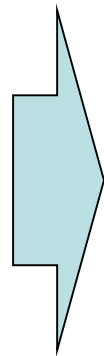
Накладные расходы

- ❑ Межпроцессорные взаимодействия:

- Коммуникации
- Синхронизации

- ❑ Дисбаланс нагрузки:

- Неравномерное распределение задач



Простой в работе

- ❑ Дополнительные накладные расходы:

- Выделение памяти
- Распределение данных

- ❑ Как оценить накладные расходы?

$$T_o = pT_p - T_s$$

Эффективность

- **Эффективность** – соотношение между результатами и использованными ресурсами:

$$E = \frac{S}{p} = \frac{T_s}{pT_p}$$

Показывает насколько хорошо параллельная реализация использует параллелизм машины

- $E \leq 1$

$$E = \frac{1}{1 + \frac{T_o}{T_s}}$$

- T_s для параллельной реализации неизменно
- Увеличение $p \Rightarrow$ увеличение накладных расходов \Rightarrow снижение эффективности

Масштабируемость

- ❑ **Масштабируемость** – показывает, как изменится время работы приложения при изменении объема задачи и доступных ресурсов
- ❑ **Сильная масштабируемость** – показывает зависимость времени решения задачи от числа доступных ресурсов при фиксированном размере задачи
- ❑ **Слабая масштабируемость** – показывает зависимость времени решения задачи от числа доступных ресурсов при фиксированном размере задачи на единицу ресурса

АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ

Применение

- ❑ Ограничения при анализе производительности:
 - Замер производительности соответствует уникальному набору:
 - Приложение
 - Вычислительная машина
 - Набор данных
- ❑ Во время разработки кода:
 - Понимание узких мест приложений
 - Позволяет узнать, как улучшить свое приложение
 - Позволяет понять, когда остановиться в оптимизации кода
- ❑ Для прогнозирования:
 - Предсказать поведение приложения:
 - Для разных конфигураций вычислительных машин
 - Для разных применений

Относительные метрики

❑ Метрики:

- Время работы
- Ускорение
- Эффективность
- Масштабируемость

❑ **Относительная производительность !!!**

- ❑ Позволяют сравнивать только один и тот же алгоритм
- ❑ Не учитывают применение программного обеспечения в иных задачах

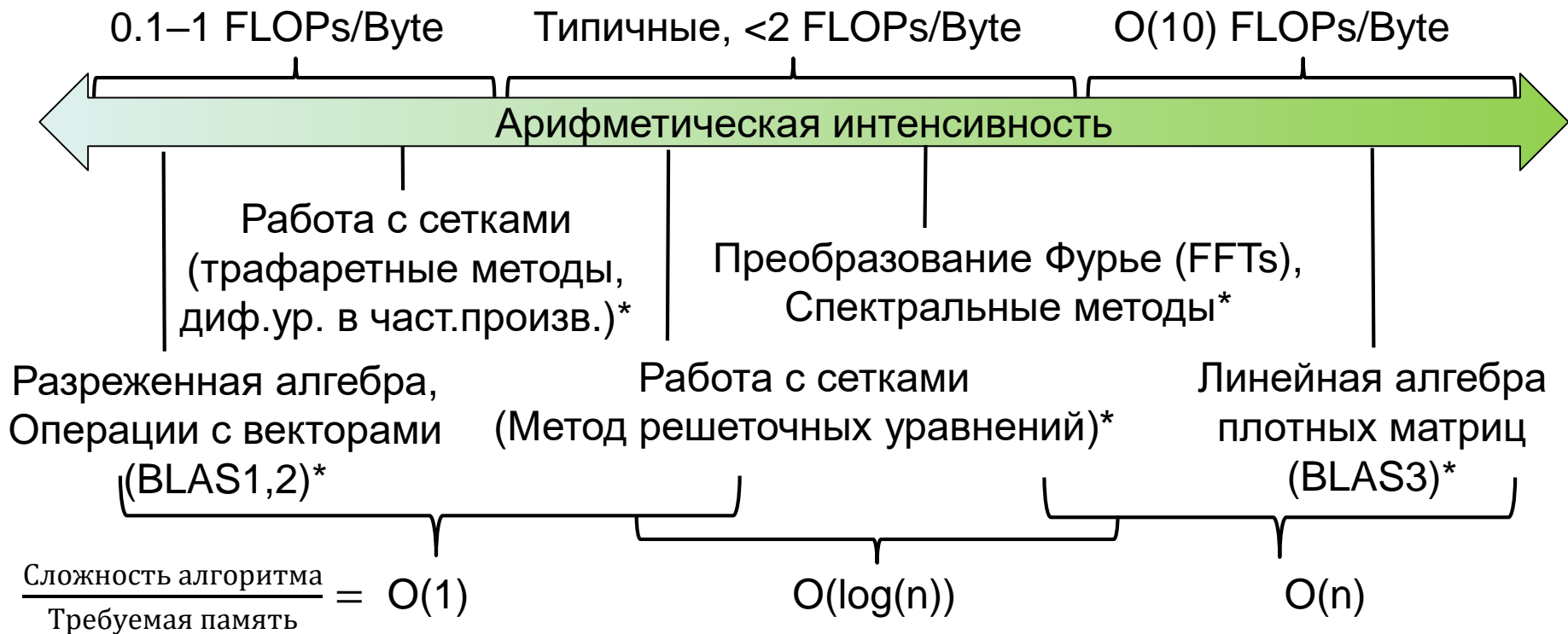
Оценка производительности

- ❑ Вычислительная производительность:
 - Достигнутая: $FLOPs = FLOP/T$
 - Эффективность: $E_c = FLOPs/Peak_FLOPs$
- ❑ Пропускная способность
 - Достигнутая: $BW = (READ + WRITE)/T$
 - Эффективность: $E_{bw} = BW/Peak_BW$
- ❑ Достигнутая производительность позволяет сравнивать разные алгоритмы
- ❑ Эффективность использования ресурсов позволяет учитывать характеристики вычислительной машины

Арифметическая интенсивность

- ❑ Арифметическая интенсивность – количество выполненных операций (обычно с плавающей запятой) на байт переданных данных
- ❑ Характеристика приложения, а не алгоритма
- ❑ Содержит только полезные операции. Не учитывает накладные расходы
- ❑ По арифметической интенсивности видно узкое место приложения:
 - Вычисления
 - Необходимо добавить вычислительных мощностей
 - Память
 - Необходимо увеличить пропускную способность памяти

Арифметическая интенсивность (2)



*Приблизительная арифметическая интенсивность при эффективной реализации алгоритма

Пример

❑ Умножение плотных квадратных матриц:

```
void MatMult(double* a, double* b, double* c, size_t size) {  
    for (size_t i = 0; i < size; i++)  
        for (size_t j = 0; j < size; j++) {  
            double sum = 0.0;  
            for (size_t k = 0; k < size; k++)  
                sum += a[i * size + k] * b[k * size + j];  
            c[i * size + j] = sum;  
        }  
}
```

Пример. Память

- ❑ Умножение плотных квадратных матриц:

```
void MatMult(double* a, double* b, double* c, size_t size) {  
    for (size_t i = 0; i < size; i++)  
        for (size_t j = 0; j < size; j++) {  
            double sum = 0.0;  
            for (size_t k = 0; k < size; k++)  
                sum += a[i * size + k] * b[k * size + j];  
            c[i * size + j] = sum;  
        }  
}
```

- ❑ Количество загрузок памяти: $load = 8 \times (N^3 + N^3)$
- ❑ Количество записей в память: $store = 8 \times N^2$
- ❑ Что может происходить с sum ?

Пример. Вычисления

- ❑ Умножение плотных квадратных матриц:

```
void MatMult(double* a, double* b, double* c, size_t size) {  
    for (size_t i = 0; i < size; i++)  
        for (size_t j = 0; j < size; j++) {  
            double sum = 0.0;  
            for (size_t k = 0; k < size; k++)  
                sum += a[i * size + k] * b[k * size + j];  
            c[i * size + j] = sum;  
        }  
}
```

- ❑ Операции с целыми: $INTOP = 3 \times N^3 + 3 \times N^3 + 3 \times N^2 + N^3$
- ❑ Операции с плавающей запятой: $FLOP = FMA \times N^3$
- ❑ FMA – вычисление $a * b + c$ (1 операция)

Пример. Арифметическая интенсивность

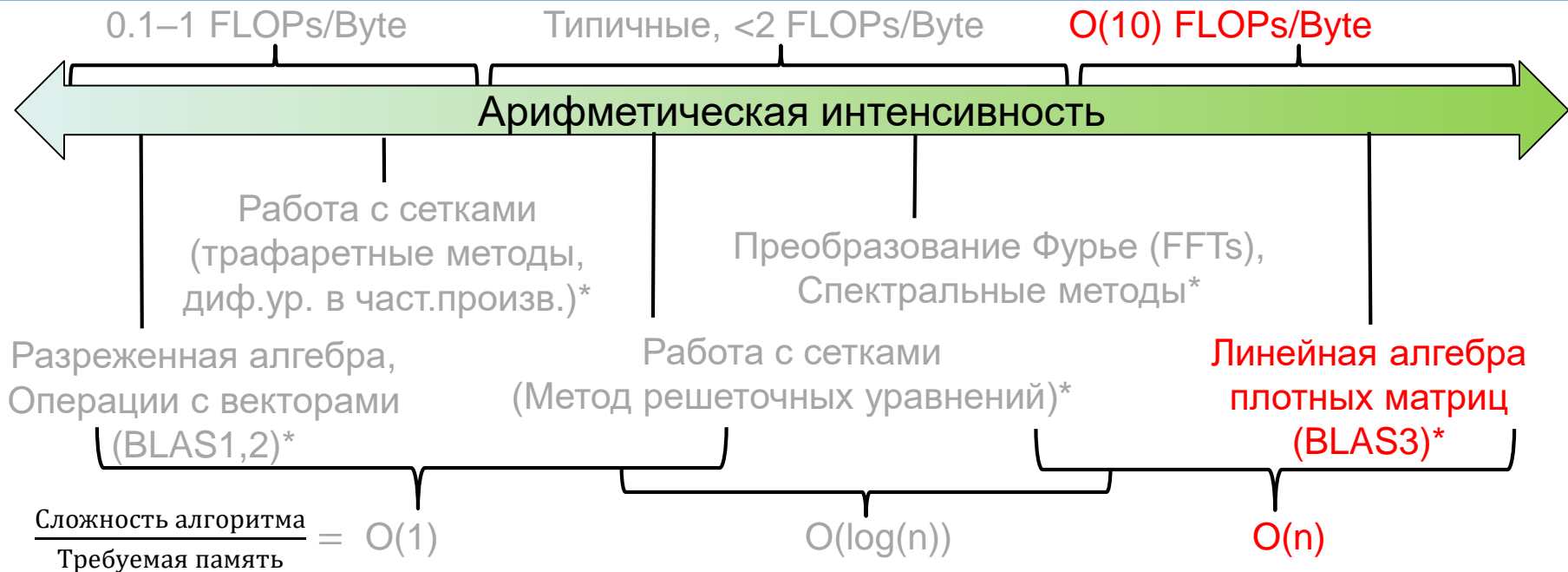
- Умножение плотных квадратных матриц:

```
void MatMult(double* a, double* b, double* c, size_t size) {  
    for (size_t i = 0; i < size; i++)  
        for (size_t j = 0; j < size; j++) {  
            double sum = 0.0;  
            for (size_t k = 0; k < size; k++)  
                sum += a[i * size + k] * b[k * size + j];  
            c[i * size + j] = sum;  
        }  
}
```

- Теоретическая арифметическая интенсивность:

$$\frac{FLOP}{load + store} = \frac{FMA \times N^3}{8 \times (2 \times N^3 + N^2)} \quad O(1)$$

Пример. Арифметическая интенсивность (2)



❑ Теоретическая арифметическая интенсивность:

$$\frac{FLOP}{load + store} = \frac{FMA \times N^3}{8 \times (2 \times N^3 + N^2)} \quad O(1)$$

ROOFLINE-МОДЕЛЬ

Определение

- ❑ **Roofline-модель** – визуальная интуитивно понятная модель производительности
 - Учитывает характеристики приложения через арифметическую интенсивность
 - Учитывает платформу через технические характеристики оборудования
- ❑ Определяет границы производительности
- ❑ Дает понимание о возможных оптимизациях
- ❑ Roofline-модель – это график в осях арифметической или операционной интенсивности (FLOPs/Byte) и производительности (GFLOPs)

Williams S., Waterman A., Patterson D. Roofline: an insightful visual performance model for multicore architectures //Communications of the ACM. – 2009. – Т. 52. – №. 4. – С. 65-76.

Вычислительные пики

- Для примера рассмотрим платформу:

- 2 x Intel Xeon Platinum 8260L (2x24 ядра, 2.4 GHz, AVX512)

- Пик скалярных вычислений:

$$Scalar Peak = 2(p) * 24(c) * 2(ILM) * 2.4GHz = 230.4 GFLOPs$$

- Пик векторных вычислений (двойная точность):

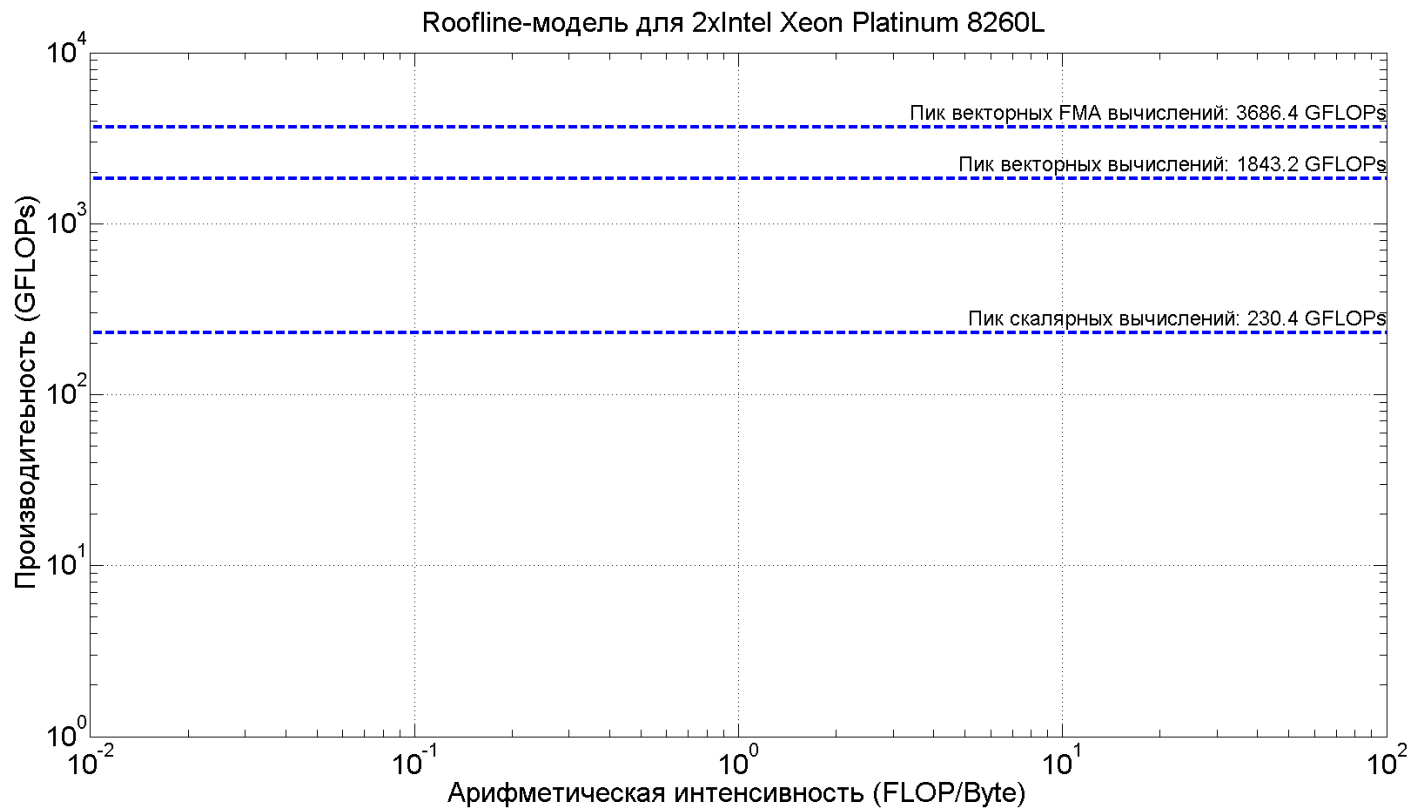
$$Vector Peak = 2 * 24 * 2 * 8(vec) * 2.4GHz = 1843.2 GFLOPs$$

- Пик векторных вычислений с FMA (двойная точность):

$$Vector FMA Peak = 2 * 24 * 4(ILM + FMA) * 8 * 2.4GHz = 3686.4 GFLOPs \\ = 3.6 TFLOPs$$

- FMA – вычисление $a * b + c$, которое может выполняться в современных архитектурах как одна операция без промежуточного округления

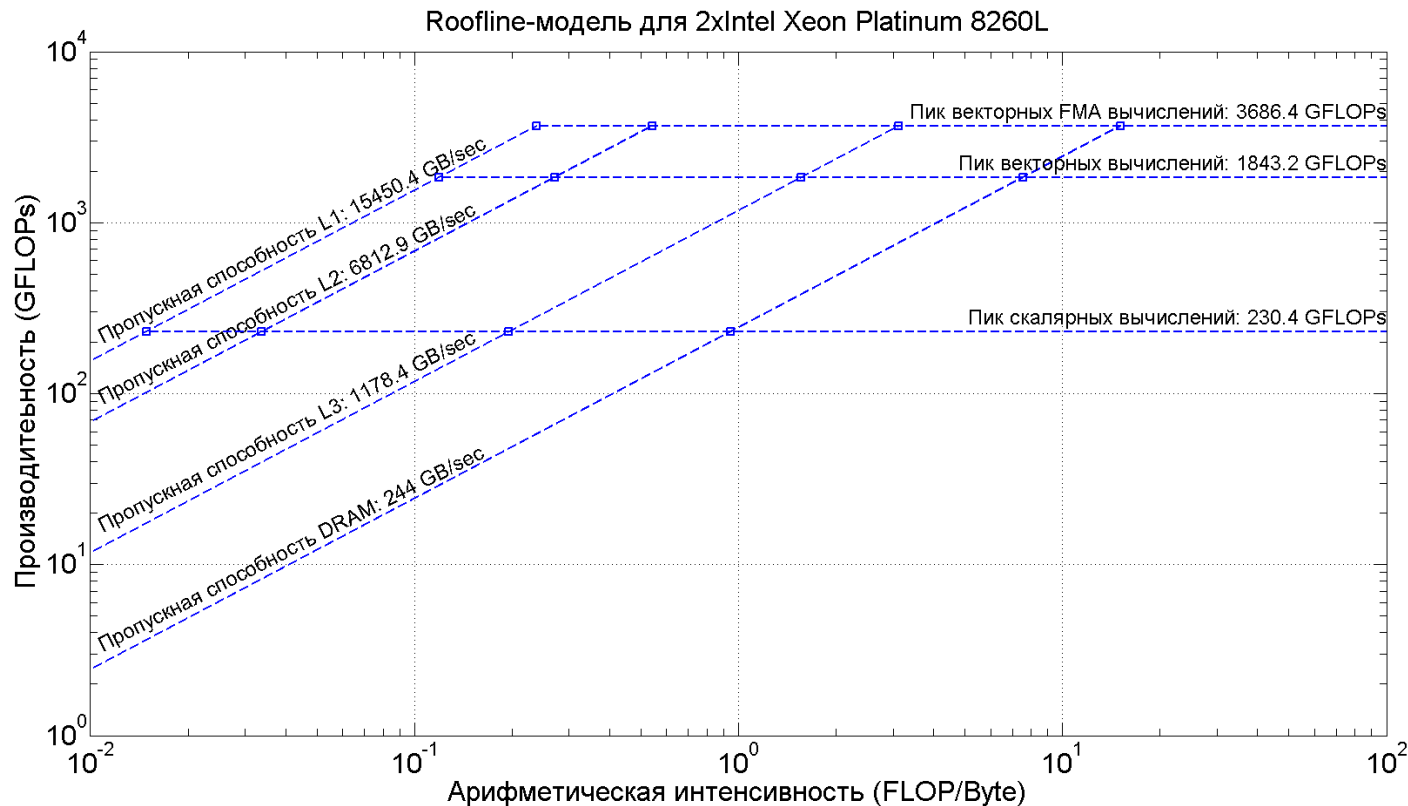
Вычислительные пики (2)



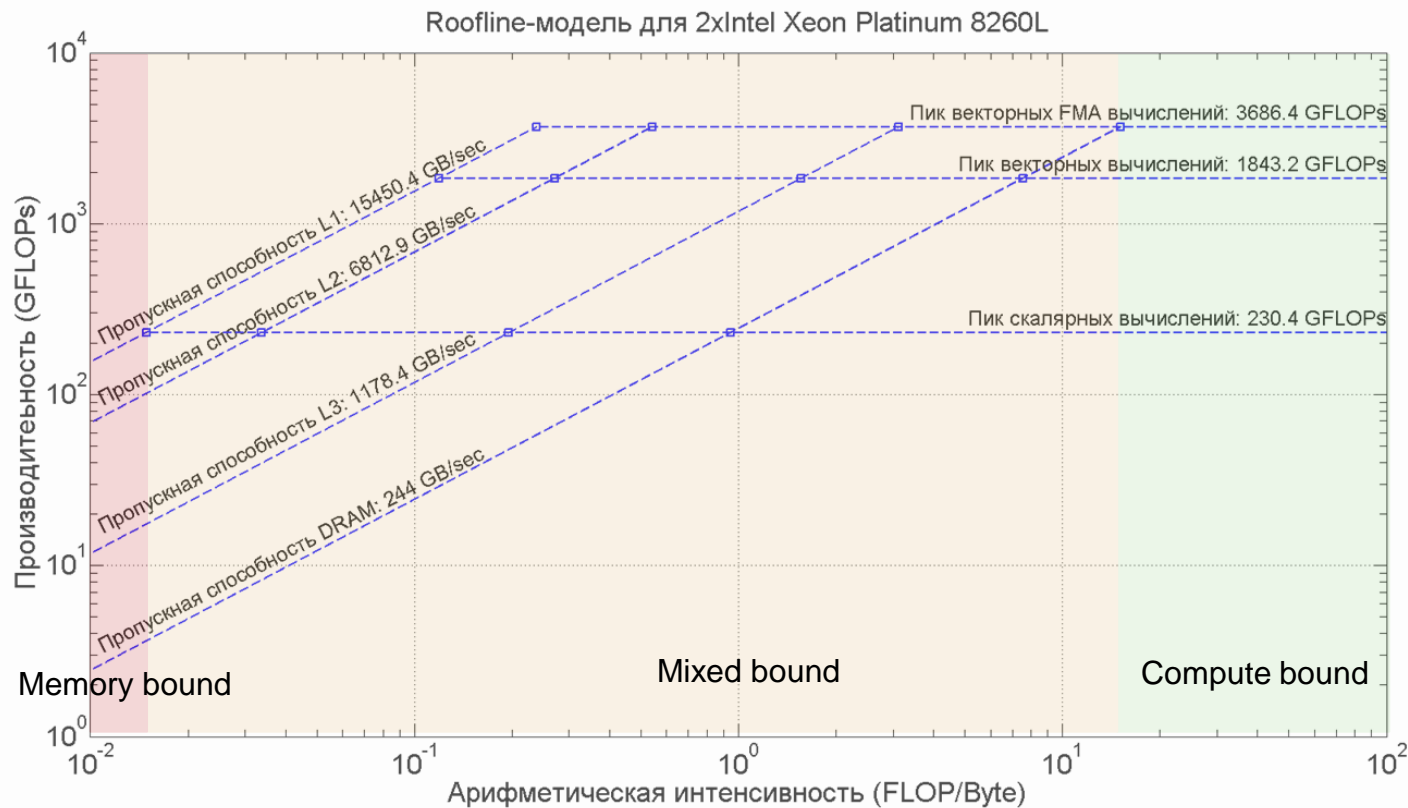
Пропускная способность памяти

- ❑ Теоретическую оценку пропускной способности памяти не используют для Roofline-модели, так как она может сильно отличаться от практических значений
- ❑ Используют значения, полученные с помощью бенчмарков
- ❑ Для рассматриваемой платформы:
 - Пропускная способность DRAM: 244 GB/sec
 - Пропускная способность кэша L3: 1178.4 GB/sec
 - Пропускная способность кэша L2: 6812.9 GB/sec
 - Пропускная способность кэша L1: 15450.4 GB/sec
- ❑ Как на графике отобразить пропускную способность памяти?

Пропускная способность памяти (2)



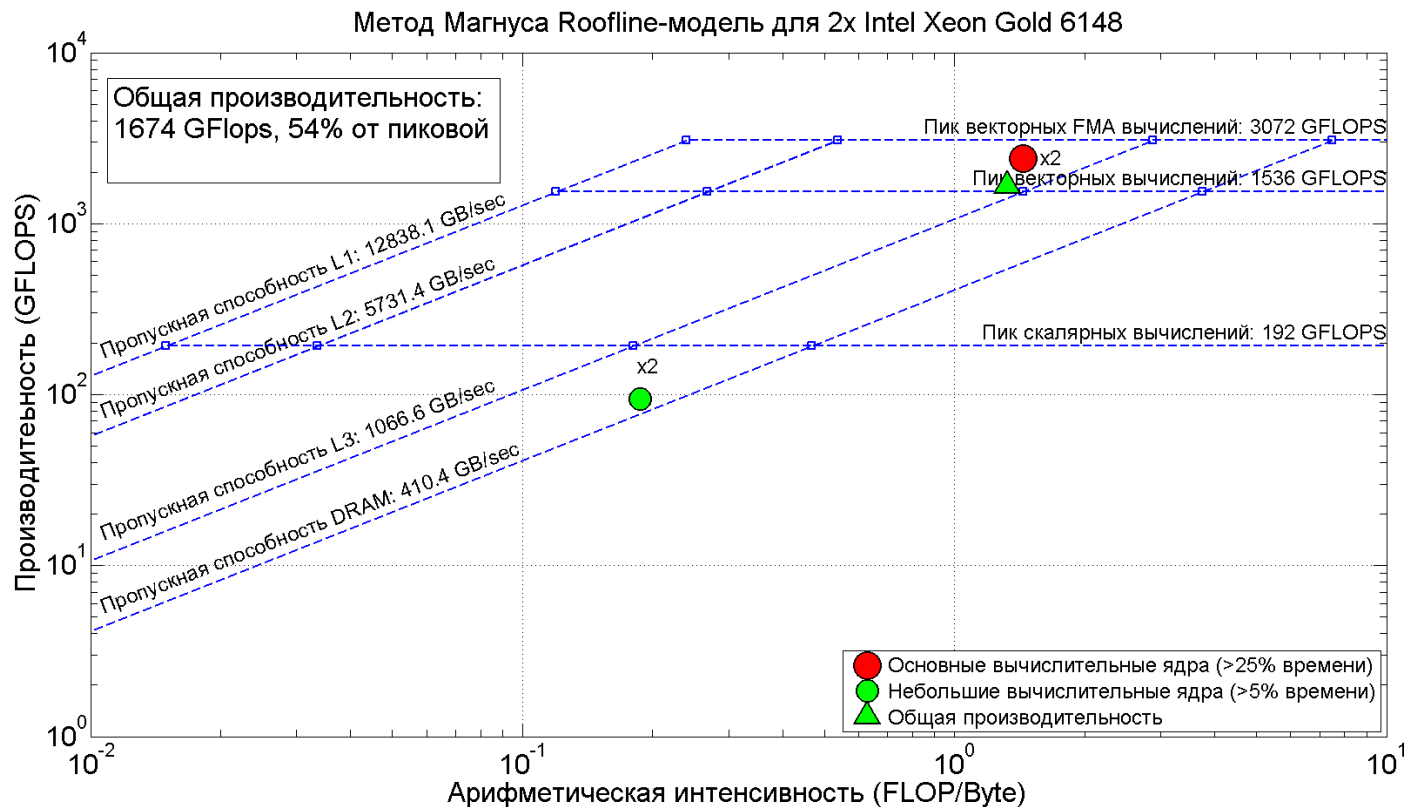
Узкие места программы



Циклы и функции приложения

- ❑ Каждый цикл в программе обладает следующими характеристиками:
 - Арифметическая интенсивность (зависит от скомпилированного кода)
 - Объем вычислений
 - Время выполнения
- ➡ Производительность цикла
- ❑ Каждая функция в программе может быть представлена как совокупность своих циклов и функций
 - ❑ Все циклы и функции приложения могут быть отображены на графике Roofline-модели

Циклы и функции приложения. Пример



ЗАКЛЮЧЕНИЕ

Заключение

- ❑ Рассмотрены основные метрики производительности аппаратных средств:
 - Вычислительная скорость
 - Пропускная способность памяти
 - Мощность
- ❑ Рассмотрены основные метрики производительности программного обеспечения:
 - Ускорение и эффективность
 - Масштабируемость
 - Арифметическая интенсивность
- ❑ Рассмотрена Roofline-модель

Литература

1. Dongarra J. J., Luszczek P., Petitet A. The LINPACK benchmark: past, present and future //Concurrency and Computation: practice and experience. – 2003.
2. McCalpin J. D. et al. Memory bandwidth and machine balance in current high-performance computers //IEEE computer society technical committee on computer architecture (TCCA) newsletter. – 1995.
3. Ueno K., Suzumura T. Highly scalable graph search for the Graph500 benchmark //Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing. – 2012.
4. Williams S., Waterman A., Patterson D. Roofline: an insightful visual performance model for multicore architectures //Communications of the ACM. – 2009.
5. Lo Y. J. et al. Roofline model toolkit: A practical tool for architectural and program analysis //International Workshop on Performance Modeling, Benchmarking and Simulation of High-Performance Computer Systems. – Springer, Cham, 2014.

Авторский коллектив

- ❑ Мееров Иосиф Борисович, к.т.н., доцент, зам. зав. каф. МОСТ
- ❑ Сысоев Александр Владимирович, к.т.н., доцент каф. МОСТ
- ❑ Линев Алексей Владимирович, зав. лаб. интернета вещей, каф. ПРИН
- ❑ Волокитин Валентин Дмитриевич, программист лаборатории СТиВВ, каф. МОСТ
- ❑ Козинов Евгений Александрович, к.т.н., преподаватель каф. МОСТ
- ❑ Панова Елена Анатольевна, инженер лаборатории СТиВВ, каф. МОСТ

Контакты

Нижегородский государственный университет

<http://www.unn.ru>

Центр компетенций oneAPI в ННГУ

<http://hpc-education.unn.ru/ru/центр-компетенций-oneapi-в-ннгу>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>