ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

Основы вычислительной физики

лекция 16

С.В. Смирнов, ФФ НГУ

Повышение скорости счёта

- Оптимизация средствами компилятора (/O2)
- Оптимизация параметров (шаг сетки, пределы, …)
- Упрощение физической модели
- Использование более совершенных алгоритмов
- Оптимизация программ с использованием профилировщиков
- Использование более производительных ЭВМ
  - Повышение тактовой частоты
  - Параллельные вычисления

Повышение скорости счёта

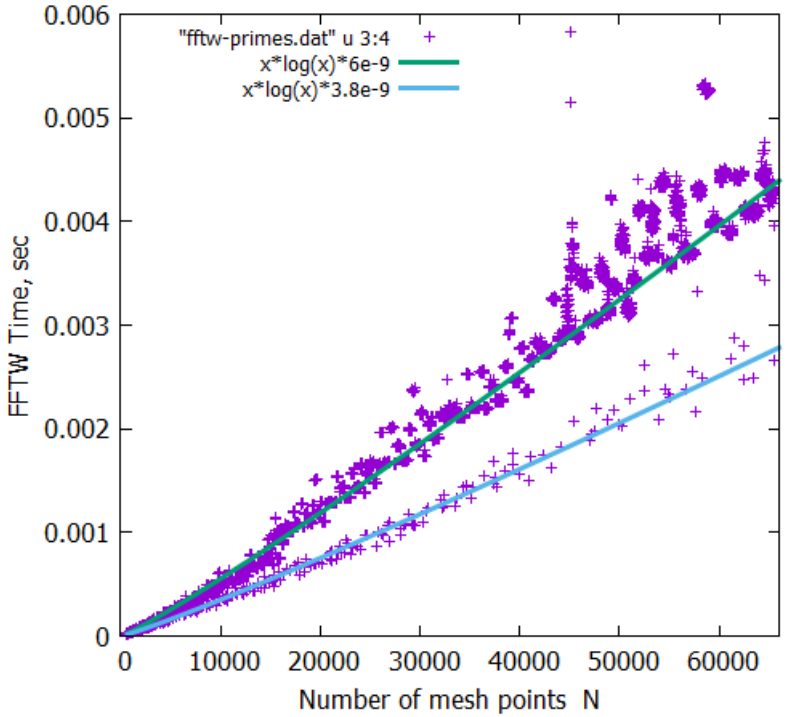
- Использование C/C++ & эффективных библиотек
- Оптимизация средствами компилятора (/O2)
- Оптимизация параметров (шаг сетки, пределы, …)
- Упрощение физической модели
- Использование более совершенных алгоритмов
- Оптимизация программ с использованием профилировщиков
- Использование более производительных ЭВМ
  - Повышение тактовой частоты
  - Параллельные вычисления

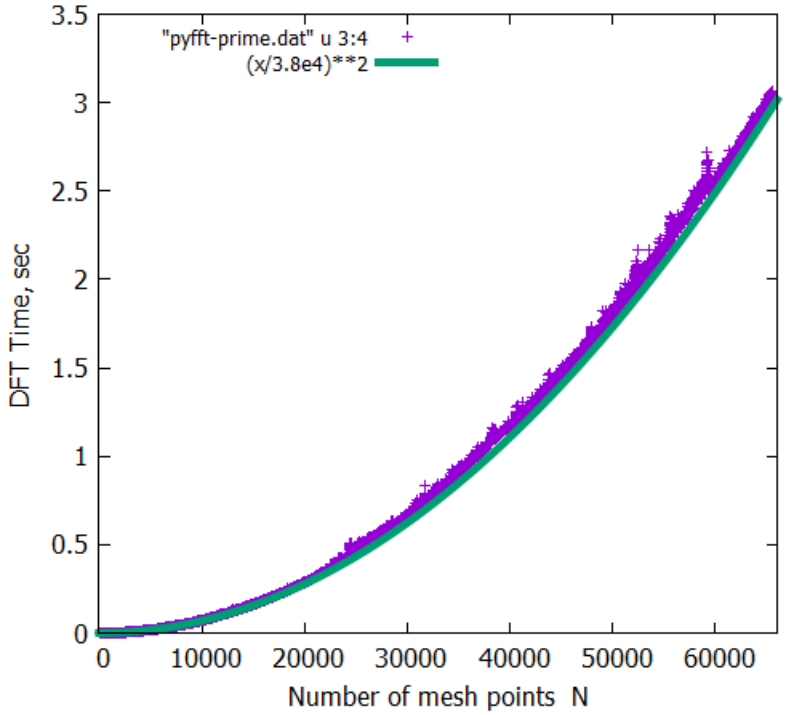
Повышение скорости счёта

- Использование C/C++ & эффективных библиотек

пример: DFT: fftw *vs.* scipy.fftpack

Case 1: число узлов N – простое



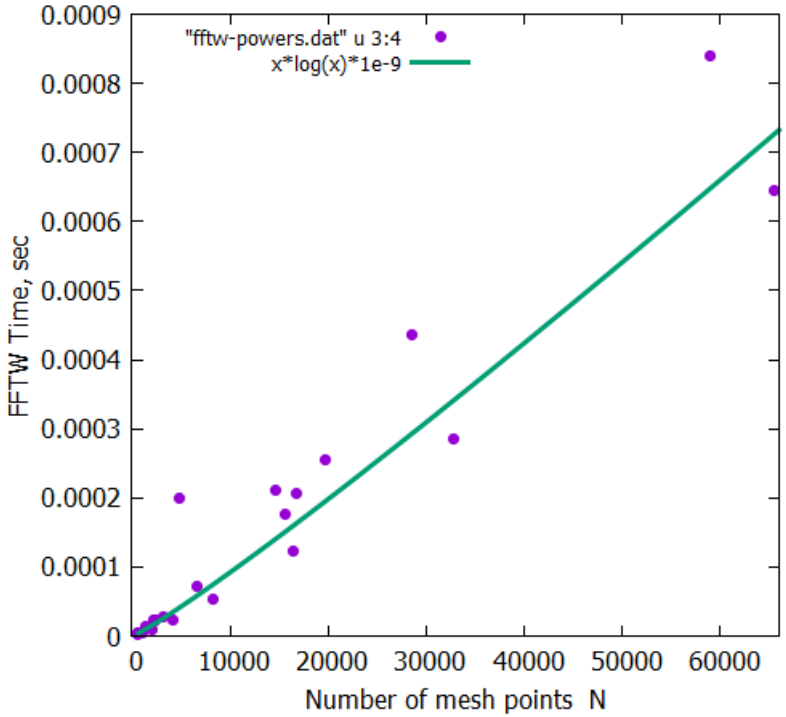


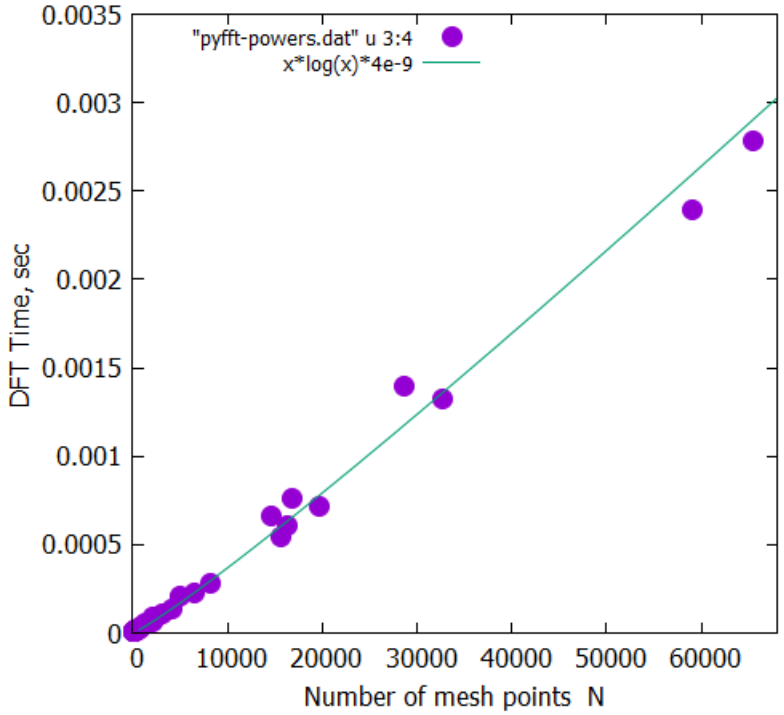
Повышение скорости счёта

- Использование C/C++ & эффективных библиотек

пример: DFT: fftw *vs.* scipy.fftpack

Case 2: число узлов N – степень 2 (3, 5, 7, ..)



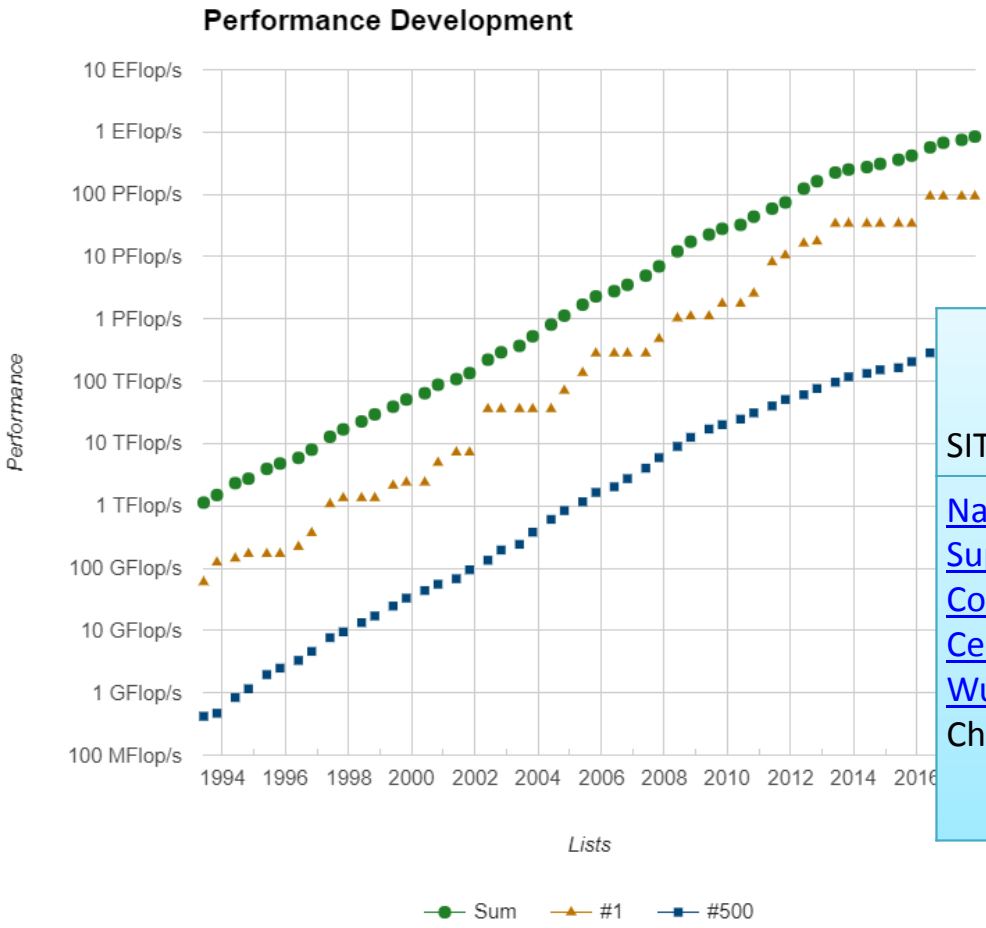


Повышение скорости счёта

- ✓ Использование C/C++ & эффективных библиотек
- Оптимизация средствами компилятора (/O2)
- Оптимизация параметров (шаг сетки, пределы, …)
- Упрощение физической модели
- Использование более совершенных алгоритмов
- Оптимизация программ с использованием профилировщиков
- Использование более производительных ЭВМ
  - Повышение тактовой частоты
  - Параллельные вычисления

Суперкомпьютеры

экспоненциальный рост производительности



Кластер НГУ: 27.5 (**17.3**) Tflop/s
18е место в ТОП-50 СНГ в 2008 г

#1 / TOP-500 (November 2016):

SITE	CORES	RMAX TFLOP/S	RPEAK TFLOP/S	POWER (KW)
National Super Computer Center in Wuxi China	10,649,600	**93,014**	125,436	15,371*

#500 (November 2017): **349** Tflop/s

http://www.top500.org/statistics/perfdevel/

*4% от мощности Новосибирской ГЭС

Суперкомпьютеры

экспоненциальный рост производительности

http://www.top500.org/statistics/perfdevel/

Кластер НГУ: 27.5 (**17.3**) Tflop/s
18е место в ТОП-50 СНГ в 2008 г

#1 / TOP-500 (November 2016):

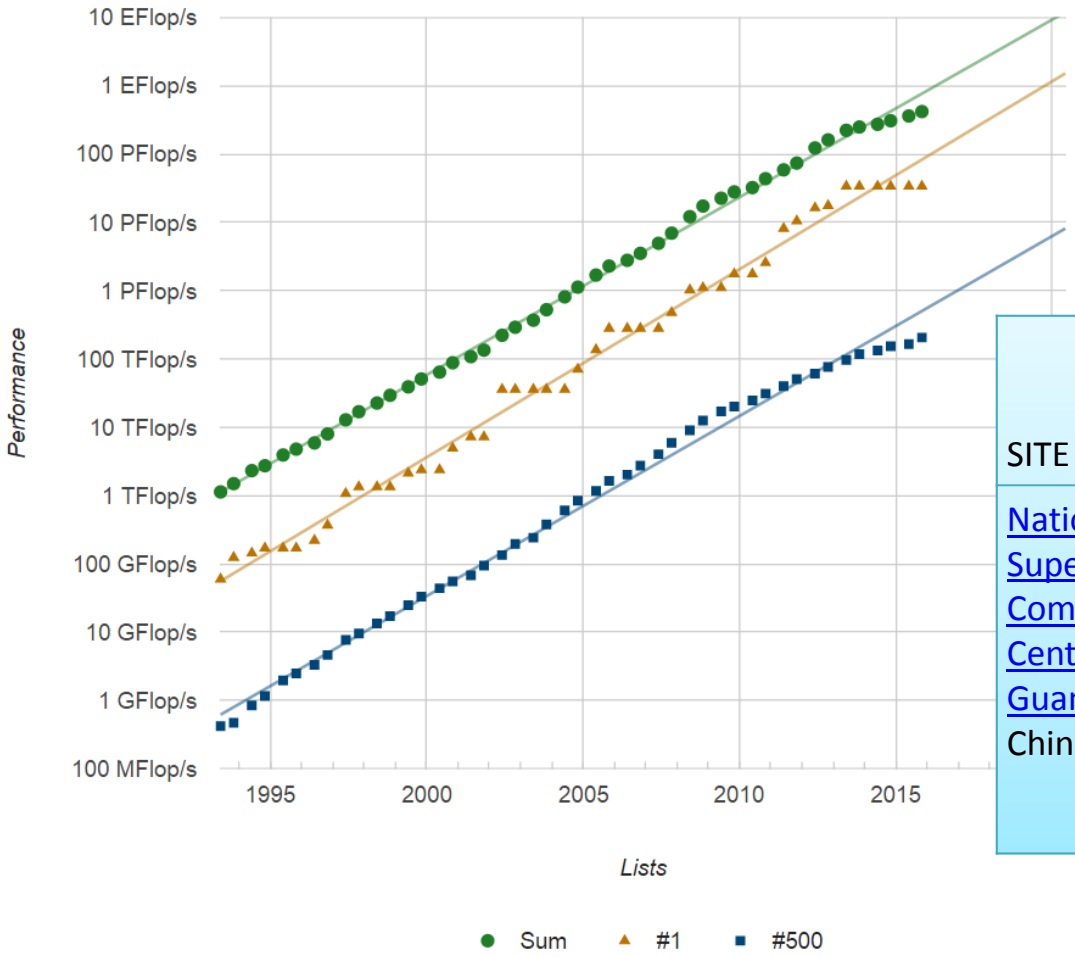
SITE	CORES	RMAX TFLOP/S	RPEAK TFLOP/S	POWER (KW)
National Super Computer Center in Wuxi China	10,649,600	**93,014**	125,436	15,371*

#500 (November 2016): **349** Tflop/s

*4% от мощности Новосибирской ГЭС

Суперкомпьютеры

экспоненциальный рост производительности



http://www.top500.org/statistics/perfdevel/

Кластер НГУ: 27.5 (**17.3**) Tflop/s
18е место в ТОП-50 СНГ в 2008 г

#1 / TOP-500 (November 2015):

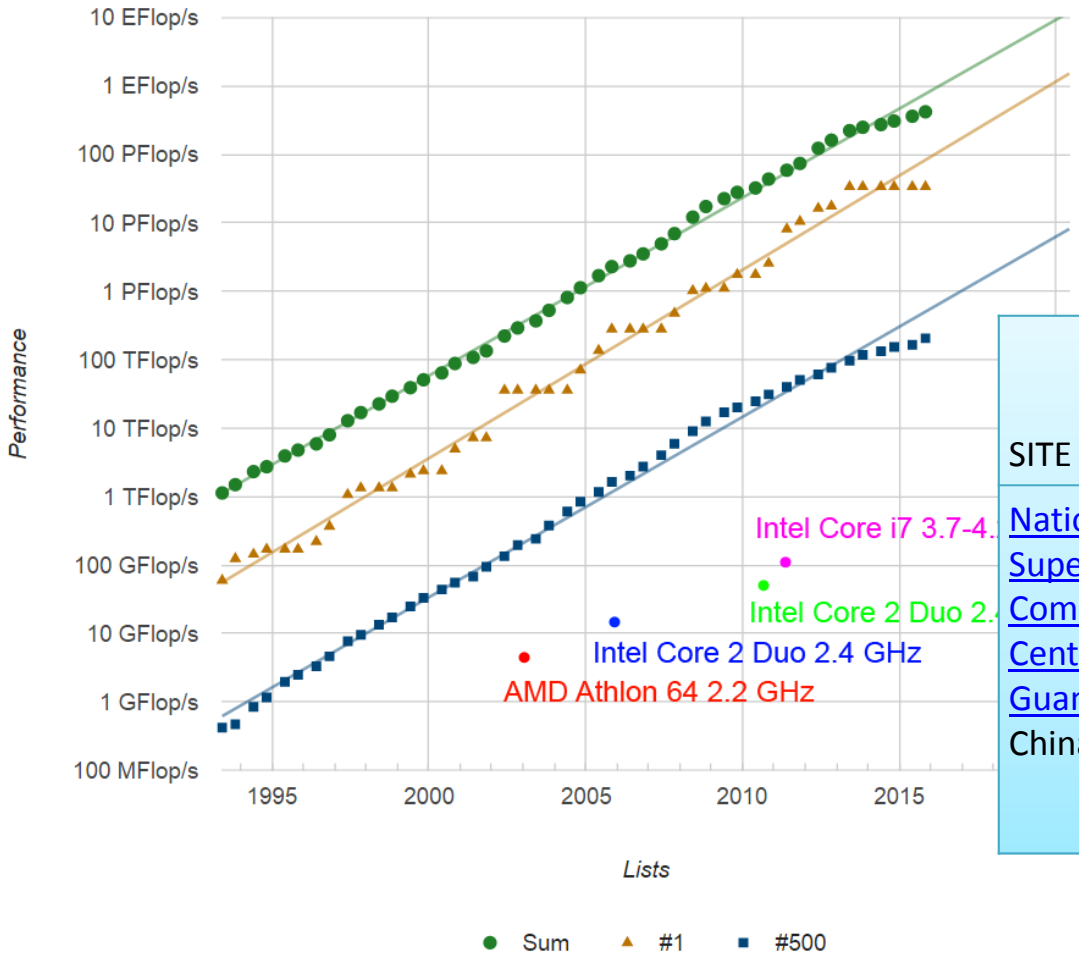
SITE	CORES	RMAX TFLOP/S	RPEAK TFLOP/S	POWER (KW)
National Super Computer Center in Guangzhou China	3,120,000	**33,862**	54,902	17,808*

#500 (November 2015): **206** Tflop/s

*4% от мощности Новосибирской ГЭС

Суперкомпьютеры

экспоненциальный рост производительности



Кластер НГУ: 27.5 (**17.3**) Tflop/s
18е место в ТОП-50 СНГ в 2008 г

#1 / TOP-500 (November 2014):

SITE	CORES	RMAX TFLOP/S	RPEAK TFLOP/S	POWER (KW)
National Super Computer Center in Guangzhou China	3,120,000	**33,862**	54,902	17,808

#500 (November 2014): **153** Tflop/s

http://www.top500.org/statistics/perfdevel/

Суперкомпьютеры

кластер НГУ: внешний вид оборудования

Суперкомпьютеры

кластер НГУ: серверная полка

64 вычислительных узла первой очереди, запущенной в 2008г, установлены в 4х серверные полки (шасси) HP BladeSystem c7000, каждая из которых содержит:

16 блэйд-серверов HP BL460c, имеющих по 16 ГБ оперативной памяти и по два 4х-ядерных процессора Xeon 5355 (2.66 ГГц).
Коммутатор сети Ethernet GbE2c Blade Switch for HP c-Class BladeSystem.
Коммутатор сети InfiniBand.
Модуль управления шасси Onboard Administrator.
6 блоков питания, работающих в режиме горячей замены.

Суперкомпьютеры

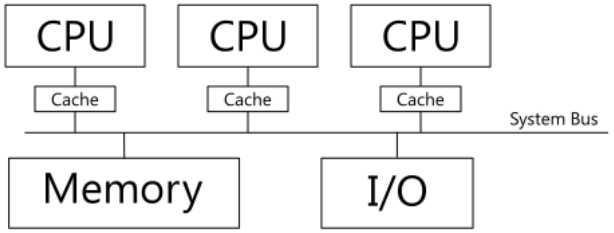
BLUEGENE/L: LAWRENCE LIVERMORE NATIONAL LABORATORY

No. 1 system from November 2004 to November 2007

Таксономия (классификация) Флинна

основные типы архитектур ЭВМ по наличию параллелизма

- SISD – Single Instruction Stream over a Single Data Stream
- SIMD – Single Instruction, Multiple Data
  - SM-SIMD (shared memory SIMD) – векторные процессоры (MMX & MMXE, 3DNow!, SSE (Streaming SIMD Extensions))
  - DM-SIMD (distributed memory SIMD) – матричные процессоры (GPU)
- MISD – Multiple Instruction, Single Data
- MIMD – Multiple Instruction, Multiple Data
  - SM-MIMD: мультипроцессоры, SMP-серверы (Symmetric Multiprocessing)
  - DM-MIMD: кластеры



Таксономия (классификация) Флинна

основные типы архитектур ЭВМ по наличию параллелизма

- ✓ SISD – Single Instruction Stream over a Single Data Stream

- SIMD – Single Instruction, Multiple Data
  - SM-SIMD (shared memory SIMD) – векторные процессоры (MMX & MMXE, 3DNow!, SSE (Streaming SIMD Extensions))
  - ➢ DM-SIMD (distributed memory SIMD) – матричные процессоры (GPU)

- MISD – Multiple Instruction, Single Data

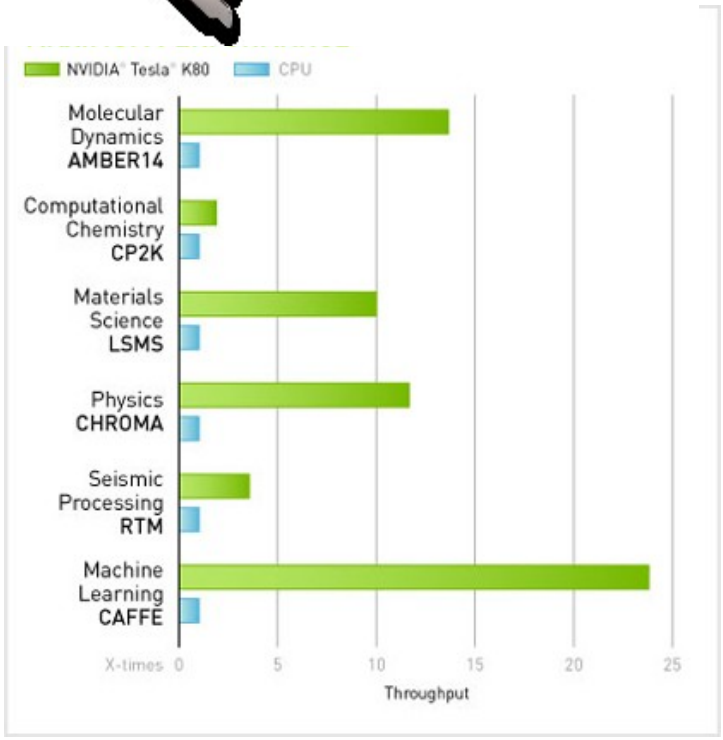
- MIMD – Multiple Instruction, Multiple Data
  - SM-MIMD: мультипроцессоры, SMP-серверы (Symmetric Multiprocessing)
  - DM-MIMD: кластеры

GPU: DM-SIMD

САМЫЕ БЫСТРЫЕ В МИРЕ ГРАФИЧЕСКИЕ УСКОРИТЕЛИ

Производительность в операциях двойной точности до 5,3 терафлопс и производительность в операциях с одинарной точностью до 10,6 терафлопс на Nvidia Tesla P100.
Большой объем встроенной памяти: 16 ГБ
Высокая пропускная способность памяти: 732 Гбит/с на Nvidia Tesla P100 GPU).

3584 ядер CUDA,
Max потребляемая мощность: 300W

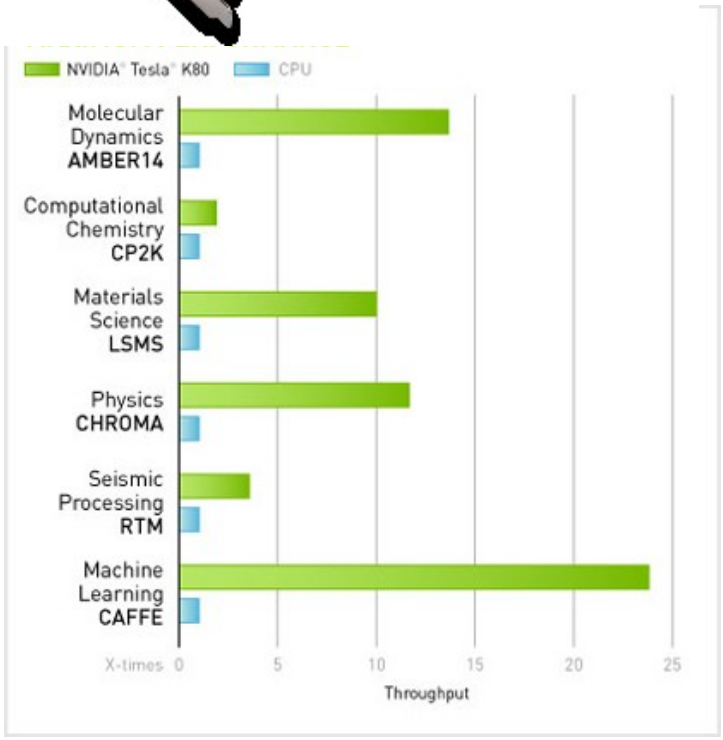


GPU: DM-SIMD

САМЫЕ БЫСТРЫЕ В МИРЕ ГРАФИЧЕСКИЕ УСКОРИТЕЛИ

Производительность в операциях двойной точности до 2,91 терафлопс и производительность в операциях с одинарной точностью до 8,74 терафлопс на графическом ускорителе Tesla K80.
Большой объем встроенной памяти: 24 ГБ
Высокая пропускная способность памяти: 480 Гбит/с на Tesla K80 GPU).

GPU: 2x Kepler GK210, 2x2496=4992 ядер CUDA



GPU: DM-SIMD

ТЕХНОСИТИ
PCI-E NVIDIA GeForce GTX1080 Ti 11264MB DDR5X Palit
Цена: 66 т.р.

Производительность в операциях двойной точности до 0.33 терафлопс и производительность в операциях с одинарной точностью до 10.6 терафлопс. (данные из Википедии)
Объем встроенной памяти: 11 ГБ ~~24 ГБ~~
Высокая пропускная способность памяти: 484 Гбит/с).

GPU: 3584 ядер CUDA ~~2x2496=4992~~
Потребляемая мощность: 250W

GPU: DM-SIMD

ТЕХНОСИТИ
PCI-E NVIDIA GeForce GTX980 OC 4096MB DDR5 GigaByte
Цена: ~~37~~ 44 т.р. (2016 г)

Производительность в операциях двойной точности до 0.14 ~~2,91~~ терафлопс и производительность в операциях с одинарной точностью до 4.6 ~~8,74~~ терафлопс. (данные из Википедии)
Объем встроенной памяти: 4 ГБ ~~24 ГБ~~
Высокая пропускная способность памяти: 224 ~~480~~ Гбит/с на Tesla K80 GPU).

GPU: 2048 ядер CUDA ~~2x2496=4992 ядер~~ CUDA

GPU: DM-SIMD

ТЕХНОСИТИ
MSI PCI-E GTX 980TI GAMING 6G nVidia GeForce GTX 980TI 6144Mb 384bit GDDR5
Цена: 60 т.р. (2016 г.)

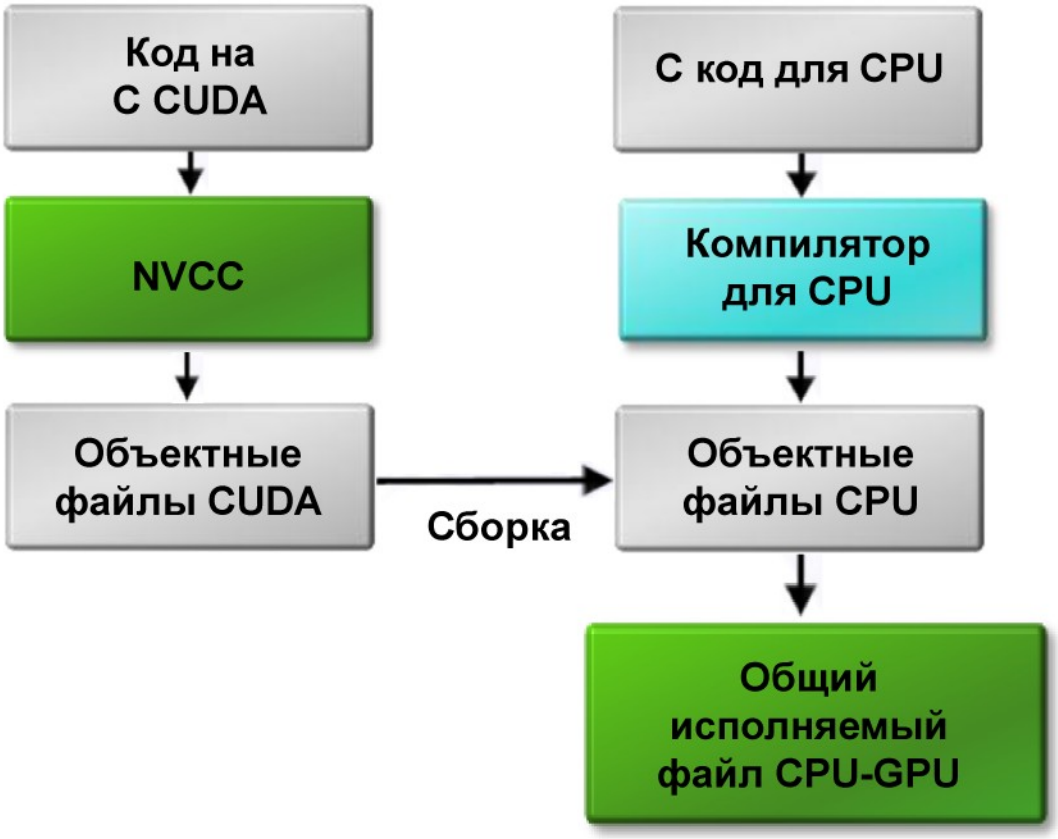
Производительность в операциях двойной точности до 0.18 терафлопс и производительность в операциях с одинарной точностью до 5.6 терафлопс. (данные из Википедии)

Объем встроенной памяти: 6 ГБ ~~4~~ ~~24 ГБ~~
Высокая пропускная способность памяти: 336 ~~224~~ ~~480~~ Гбит/с на Tesla K80 GPU).

GPU: 2816 ядер CUDA ~~2048~~ ~~2x2496=4992 ядер~~ CUDA

CUDA

процесс сборки приложения



CUDA

платформа параллельных вычислений от NVIDIA

- Параллельная часть кода выполняется как большое количество (*тысячи*) нитей (threads)
- Нити группируются в блоки (*blocks*) фиксированного размера (***blockDim***); количество потоков в блоке ограничено (типичные значения для разного оборудования – 512 и 1024 нитей в блоке)
- Блоки объединяются в сеть блоков (*grid*)
- Код ядра выполняется на сетке из блоков
- Каждая нить и блок имеют уникальный идентификатор (***threadIdx*** и ***blockIdx***)
- Возможна 1D (аудио), 2D (изображения и видео) и 3D (физ.моделирование) топология блока и нитей
  ID thread = x + y*Dx + z*Dx*Dy
- Блоки могут использовать быструю **shared** пямять
- Внутри блока потоки могут синхронизоваться

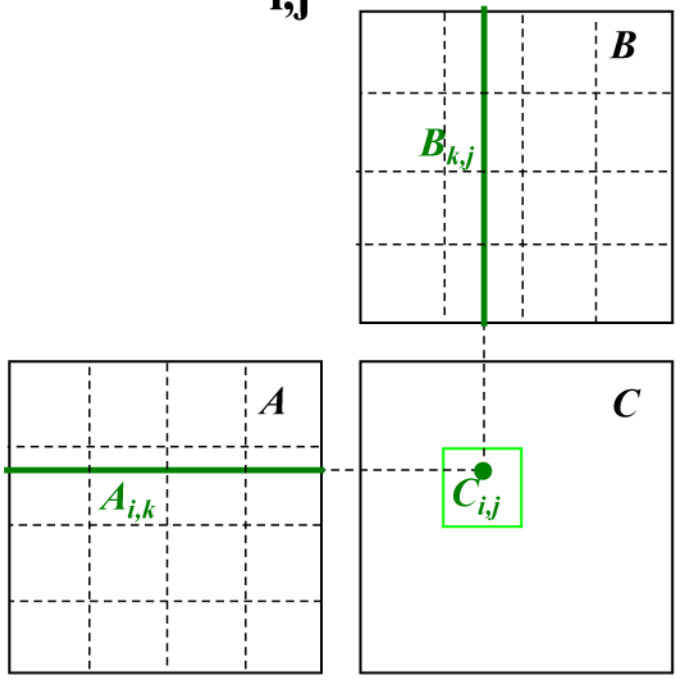
CUDA

платформа параллельных вычислений от NVIDIA

Тип памяти	Доступ	Уровень выделения	Скорость работы
Регистры	R/W	Per-thread	Высокая(on-chip)
Локальная	**R/W**	**Per-thread**	**Низкая (DRAM)**
Shared	R/W	Per-block	Высокая(on-chip)
Глобальная	**R/W**	**Per-grid**	**Низкая (DRAM)**
Constant	R/O	Per-grid	Высокая(L1 cache)
Texture	R/O	Per-grid	Высокая(L1 cache)

Пример: умножение матриц

- **Произведение двух квадратных матриц *A* и *B* размера *N**N*, *N* кратно 16**
- **Матрицы расположены в глобальной памяти**
- **По одной нити на каждый элемент $C_{i,j}$**
- **2D блок – 16*16**
- **2D *grid***



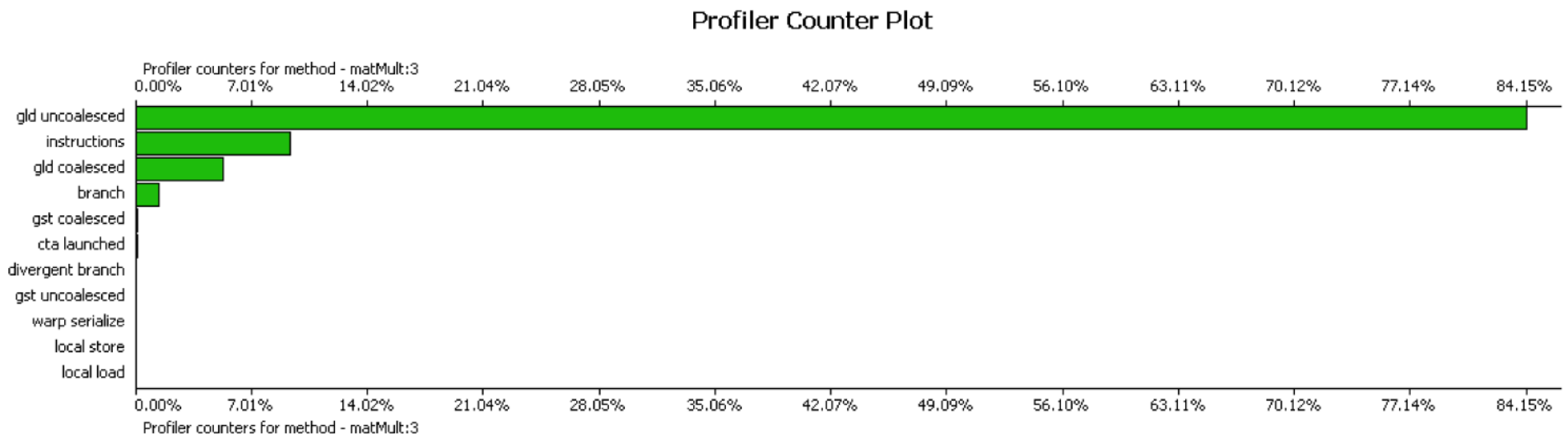
Умножение матриц

```
#define BLOCK_SIZE 16

__global__ void matMult ( float * a, float * b, int n,
                          float * c )
{
  int   bx  = blockIdx.x;
  int   by  = blockIdx.y;
  int   tx  = threadIdx.x;
  int   ty  = threadIdx.y;
  float sum = 0.0f;
  int   ia  = n * BLOCK_SIZE * by + n * ty;
  int   ib  = BLOCK_SIZE * bx + tx;
  int   ic  = n * BLOCK_SIZE * by + BLOCK_SIZE * bx;

  for ( int k = 0; k < n; k++ )
    sum += a [ia + k] * b [ib + k*n];

  c [ic + n * ty + tx] = sum;
}
```

Умножение матриц

```
int          numBytes = N * N * sizeof ( float );
float      * adev, * bdev, * cdev ;
dim3         threads ( BLOCK_SIZE, BLOCK_SIZE );
dim3         blocks ( N / threads.x, N / threads.y);

cudaMalloc       ( (void**)&adev, numBytes ); // allocate DRAM
cudaMalloc       ( (void**)&bdev, numBytes ); // allocate DRAM
cudaMalloc       ( (void**)&cdev, numBytes ); // allocate DRAM
                                              // copy from CPU to DRAM
cudaMemcpy ( adev, a, numBytes, cudaMemcpyHostToDevice );
cudaMemcpy ( bdev, b, numBytes, cudaMemcpyHostToDevice );

matMult<<<blocks, threads>>> ( adev, bdev, N, cdev );

cudaThreadSynchronize();
cudaMemcpy             ( c, cdev, numBytes, cudaMemcpyDeviceToHost );

cudaFree               ( adev );
cudaFree               ( bdev );
cudaFree               ( cdev );
```

Используем CUDA Profiler



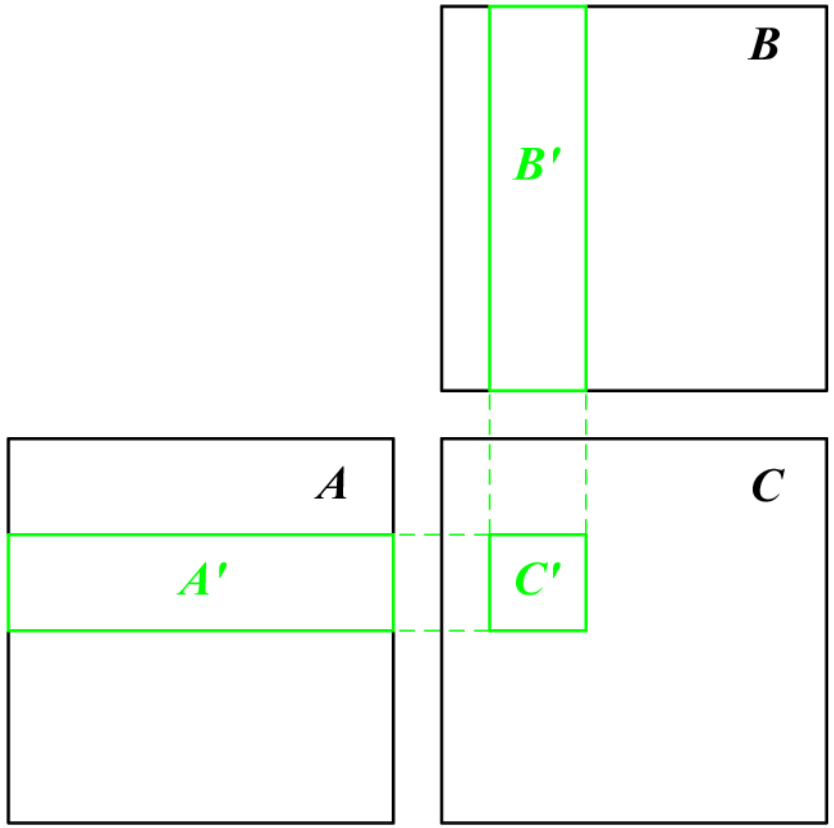
- Легко видно, что основное время (84.15%) ушло на чтение из глобальной памяти
- Непосредственно вычисления заняли всего около 10%

Типичный паттерн использования

1. Загрузить необходимые данные в shared-память (из глобальной)
2. __syncthreads ()
3. Выполнить вычисления над загруженными данными
4. __syncthreads ()
5. Записать результат в глобальную память

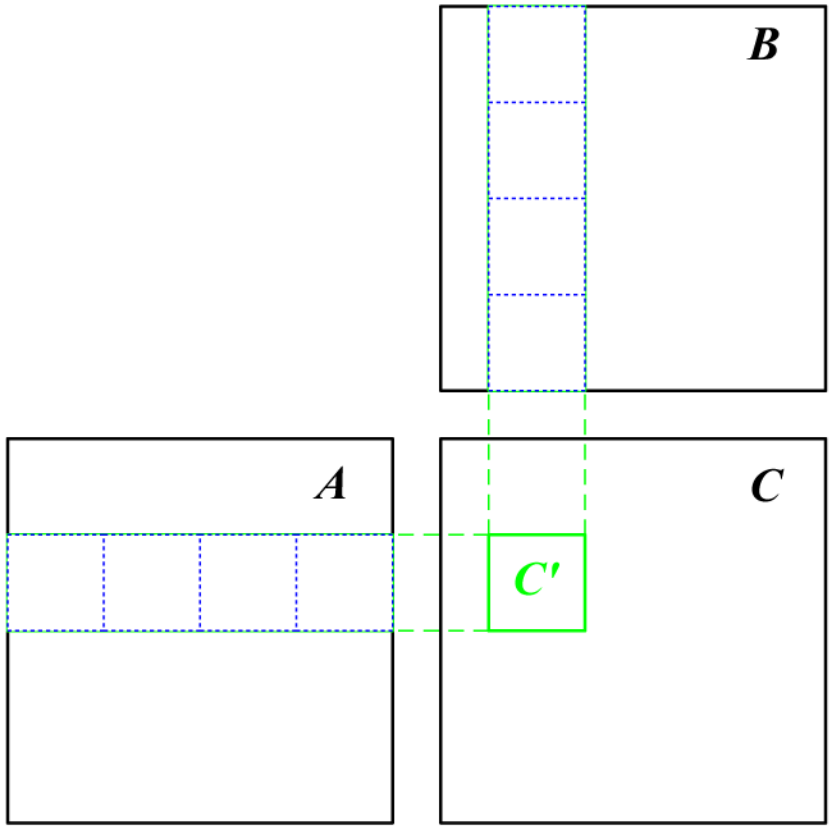
Простейшая реализация

- При вычислении C’ постоянно используются одни и те же элементы из A и B
  - По много раз считываются из глобальной памяти
- Эти многократно используемые элементы формируют полосы в матрицах A и B
- Размер такой полосы N*16 и для реальных задач даже одна такая полоса не помещается в shared-память



Эффективная реализация

- «Разделяй и властвуй»
- Разбиваем каждую полосу на квадратные матрицы (16*16)
- Тогда требуемая подматрица произведения C’ может быть представлена как сумма произведений таких матриц 16*16
- Для работы нужно только две матрицы 16*16 в **shared**-памяти



$$C' = A'_1 * B'_1 + \ldots + A'_{N/16} * B'_{N/16}$$

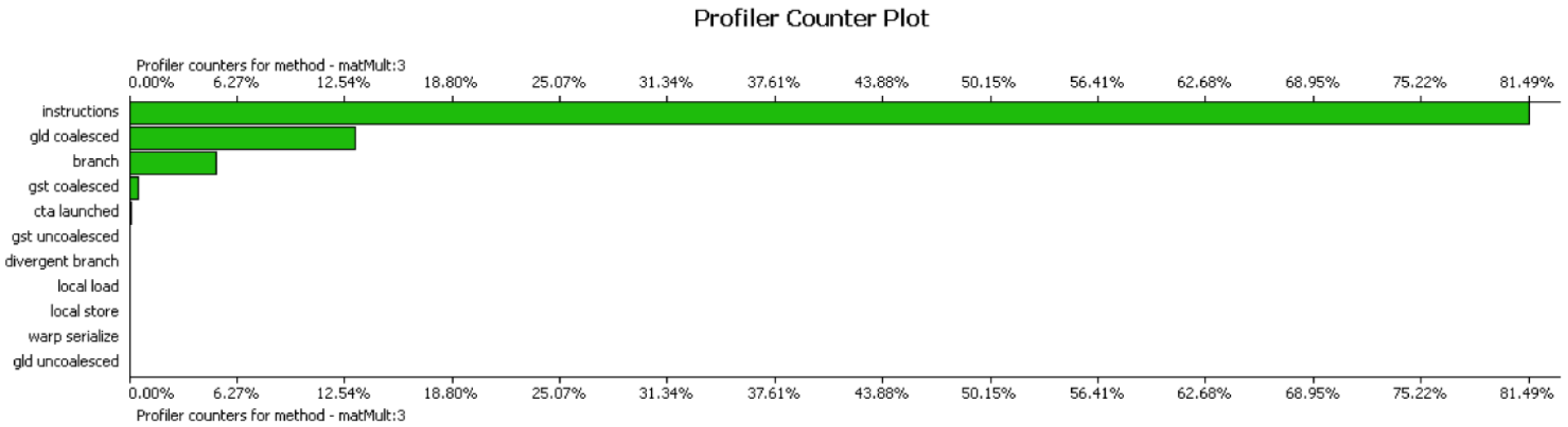
Эффективная реализация

```
__global__ void matMult ( float * a, float * b, int n, float * c ) {
  int bx     = blockIdx.x,  by = blockIdx.y;
  int tx     = threadIdx.x, ty = threadIdx.y;
  int aBegin = n * BLOCK_SIZE * by;
  int aEnd   = aBegin + n - 1;
  int bBegin = BLOCK_SIZE * bx;
  int aStep  = BLOCK_SIZE, bStep  = BLOCK_SIZE * n;
  float sum  = 0.0f;
  for ( int ia = aBegin, ib = bBegin; ia <= aEnd; ia += aStep, ib += bStep ){
    __shared__ float as [BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float bs [BLOCK_SIZE][BLOCK_SIZE];
    as [ty][tx] = a [ia + n * ty + tx];
    bs [ty][tx] = b [ib + n * ty + tx];
    __syncthreads ();     // Synchronize to make sure the matrices are loaded
   for ( int k = 0; k < BLOCK_SIZE; k++ )
     sum += as [ty][k] * bs [k][tx];
   __syncthreads ();      // Synchronize to make sure submatrices not needed
  }
  c [n * BLOCK_SIZE * by + BLOCK_SIZE * bx + n * ty + tx] = sum;
}
```

Эффективная реализация

- **На каждый элемент**
  - 2*N арифметических операций
  - 2*N/16 обращений к глобальной памяти
- **Поскольку разные warp'ы могут выполнять разные команды нужна явная синхронизация всех нитей блока**
- **Быстродействие выросло более чем на порядок (2578 vs 132 миллисекунд)**

Эффективная реализация



- Теперь основное время (81.49%) ушло на вычисления
- Чтение из памяти стало *coalesced* и заняло всего 12.5 %

CUDA

платформа параллельных вычислений от NVIDIA

PROs:

+ Ускорение в несколько раз (≈ на порядок величины)
+ Для ряда задач более эффективное решение в flops/$ и flops/W

CONs:

— Сложность программирования, необходимость специального дизайна и оптимизации программ
— Ограниченная применимость (SIMD, «лёгкие» нити)
— Расчёты здесь-и-сейчас – стандарт развивается, возможности оборудования постоянно расширяются.
— Ограниченная масштабируемость

Таксономия (классификация) Флинна

основные типы архитектур ЭВМ по наличию параллелизма

- ✓ SISD – Single Instruction Stream over a Single Data Stream

- SIMD – Single Instruction, Multiple Data
  - SM-SIMD (shared memory SIMD) – векторные процессоры (MMX & MMXE, 3DNow!, SSE (Streaming SIMD Extensions))
  - ✓ DM-SIMD (distributed memory SIMD) – матричные процессоры (GPU)

- MISD – Multiple Instruction, Single Data

- MIMD – Multiple Instruction, Multiple Data
  - ➢ SM-MIMD: мультипроцессоры, SMP-серверы (Symmetric Multiprocessing)
  - DM-MIMD: кластеры

OpenMP

стандарт параллельных вычислений для SM-SIMD/SM-MIMD

- Поддержка наиболее распространенными компиляторами (msvc, gcc / g++, …): компиляция с ключом `-fopenmp`
- Отличие от «обычных» программ на Си – директивы `#pragma`

Основные директивы `#pragma`, функции, константы:

- создание потоков (директива parallel),
- распределение работы между потоками (директивы for и section),
- конструкции для управления работой с данными (выражения shared и private для определения класса памяти переменных),
- конструкции для синхронизации потоков (директивы critical, atomic и barrier),
- процедуры библиотеки поддержки времени выполнения (например, omp_get_thread_num),
- переменные окружения (напр., OMP_NUM_THREADS).

```
#pragma omp parallel
{
    …
    #pragma omp atomic
    count = count + n;
}
```

OpenMP

стандарт параллельных вычислений для SM-SIMD/SM-MIMD

PROs:

+ Поддерживается наиболее распространенными компиляторами
+ Простота использования
  (нужно написать #pragma omp parallel перед циклом)
+ Многие современные ЭВМ – многопроцессорные, а процессоры – многоядерные => можно ускорить свою программу уже сегодня!

CONs:

— Относительно высокая стоимость flops/$
— Очень ограниченная масштабируемость

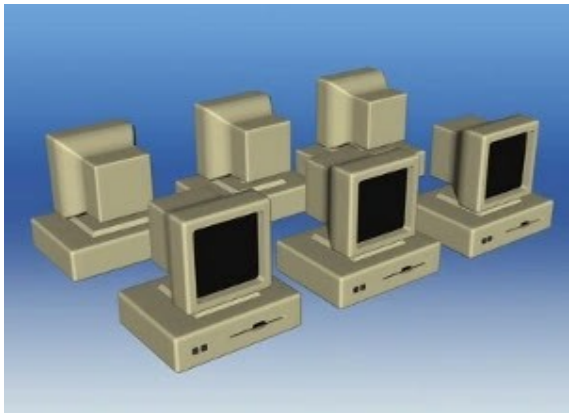
Таксономия (классификация) Флинна

основные типы архитектур ЭВМ по наличию параллелизма

- ✓ SISD – Single Instruction Stream over a Single Data Stream
- ✓ SIMD – Single Instruction, Multiple Data
  - ✓ SM-SIMD (shared memory SIMD) – векторные процессоры (MMX & MMXE, 3DNow!, SSE (Streaming SIMD Extensions))
  - ✓ DM-SIMD (distributed memory SIMD) – матричные процессоры (GPU)
- MISD – Multiple Instruction, Single Data
- MIMD – Multiple Instruction, Multiple Data
  - SM-MIMD: мультипроцессоры, SMP-серверы (Symmetric Multiprocessing)
  - ➢ DM-MIMD: кластеры

MPI, Message Passing Interface

стандарт де-факто & API для DM-MIMD систем



- Полноценные независимые процессы с раздельными пространствами памяти
- Выполнение разных процессов в т.ч. на различных физических устройствах
- Взаимодействие процессов по сети TCP/IP (передача данных, синхронизация)
- MPI позволяет абстрагироваться от передачи TCP/IP сообщений, вызывая высокоуровневые функции API (MPI_Send, MPI_Bcast, MPI_Recv, MPI_Gather, MPI_Reduce, MPI_Barrier, …)
- Целый ряд бесплатных и коммерческих реализаций MPI
- Унифицированный способ компиляции (mpiCC) за счет использования mpi-selector (try mpi-selector --list, --query)

MPI, Message Passing Interface

стандарт де-факто & API для DM-MIMD систем

PROs:

+ Возможность использования в компьютерных сетях, в т.ч. Разнородных и со сложной топологией
+ Возможность совместного использования с OMP & GPU
+ Относительная простота использования
+ Масштабируемость

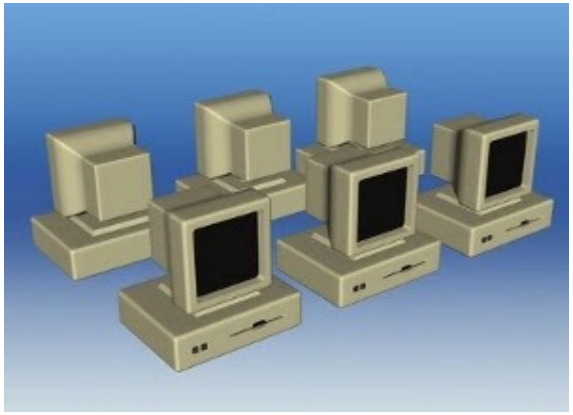
CONs:

— Ограничения скорости межпроцессных коммуникаций шириной полосы сетей данных
— Относительно высокая стоимость flops/$

Использование независимых процессов

PROs:

+ Простота программирования
+ Возможность задействования разнородного оборудования
+ Неограниченная масштабируемость



CONs:

— Ограниченная применимость
(подходит для большого числа однотипных задач с вариацией параметров; счёт с усреднением по случайным реализациям / шумам и т.п.)

— Высокая стоимость flops/$

Повышение скорости счёта

- Использование C/C++ & эффективных библиотек
- Оптимизация средствами компилятора (/O2)
- Оптимизация параметров (шаг сетки, пределы, …)
- Упрощение физической модели
- Использование более совершенных алгоритмов
- Оптимизация программ с использованием профилировщиков
- Использование более производительных ЭВМ
  - Повышение тактовой частоты
  - Параллельные вычисления