Самарский государственный аэрокосмический университет
имени академика С. П. Королева
Кафедра технической кибернетики

Обзор высокоуровневых платформ, использующих вычислительные мощности GPU

Каранашев Мухамед Русланович
mukhamed.karanashev@gmail.com

План доклада:

1. Проблема доступности GPGPU
2. Развитие инструментария разработки на CUDA
3. Обзор высокоуровневого инструментария для вычислений на GPU: C++, Java, Fortran, Python, Matlab
4. Выбор инструмента.

Доступность GPU

Графические процессоры, поддерживающие вычисления общего назначения, широко распространены.

NVIDIA	AMD/ATi
GeForce 9800	Radeon HD 5770
GeForce 8800	Radeon HD 4850
GeForce GTX 260	Radeon HD 5850
GeForce GTX 460	Radeon HD 6950

Источник: http://store.steampowered.com/hwsurvey/videocard/

Стандартный способ утилизации этих ресурсов: CUDA SDK и разработка на CUDA C:

- слишком низкий уровень;
- вынуждает вручную разбивать программу на части, выполняющиеся на CPU и GPU.

Вычисления на GPU

2002 Языки программирования шейдеров (**Cg/HLSL/GLSL**)

2004 **Brook** – диалект C, работающий поверх DirectX/OpenGL

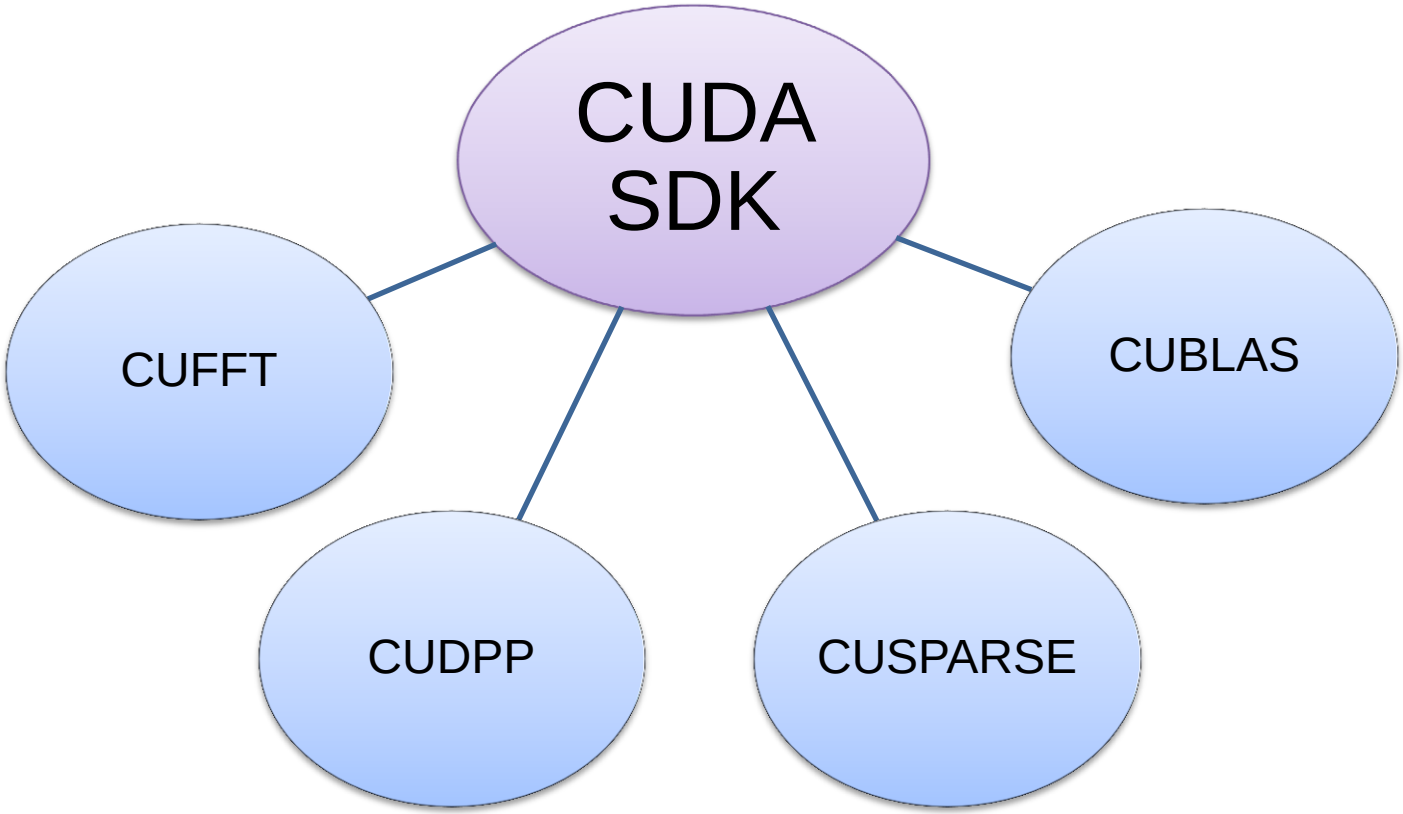
2006 **ATI “Close to Metal”** – низкоуровневый интерфейс для GPGPU.

2007 **AMD Stream SDK 1.0** – набор средств разработки для потоковых процессоров FireStream

NVIDIA CUDA SDK 1.0 – набор средств разработки для потоковых процессоров CUDA.

2008 **OpenCL 1.0** – фреймворк для разработки в гетерогенных средах.

Вычисления на GPU: CUDA



Общий недостаток: частность, «библиотечность» решений

C++

(ручное управление памятью, высокая производительность, ООП)

Thrust (http://code.google.com/p/thrust/)

NVIDIA Research, 2008

Лицензия: Apache Licence 2.0 (open-source)

- нет необходимости работать с ядрами GPU;
- STL-подобный дизайн (контейнеры/итераторы/алгоритмы);
- прозрачные операции с данными в памяти GPU;
- набор утилитных функций и алгоритмов;
- поставляется с CUDA SDK 4.0.

Thrust: пример

```
class random {
public:
  int operator() () {
    return rand() % 10;
  }
};

template <typename T> struct square {
  __host__ __device__ T operator() (T x) {
    return x * x;
  }
};

...
thrust::host_vector<int> data(DATA_SIZE);
thrust::generate(data.begin(), data.end(), random());

thrust::device_vector<int> gpudata = data;

int final_sum = thrust::transform_reduce(gpudata.begin(), gpudata.end(),
    square<int>(), 0, thrust::plus<int>());
```

Функтор для генерации случайного числа в диапазоне 0..10

Функтор возведения в квадрат

Генерация случайного вектора целых чисел на CPU

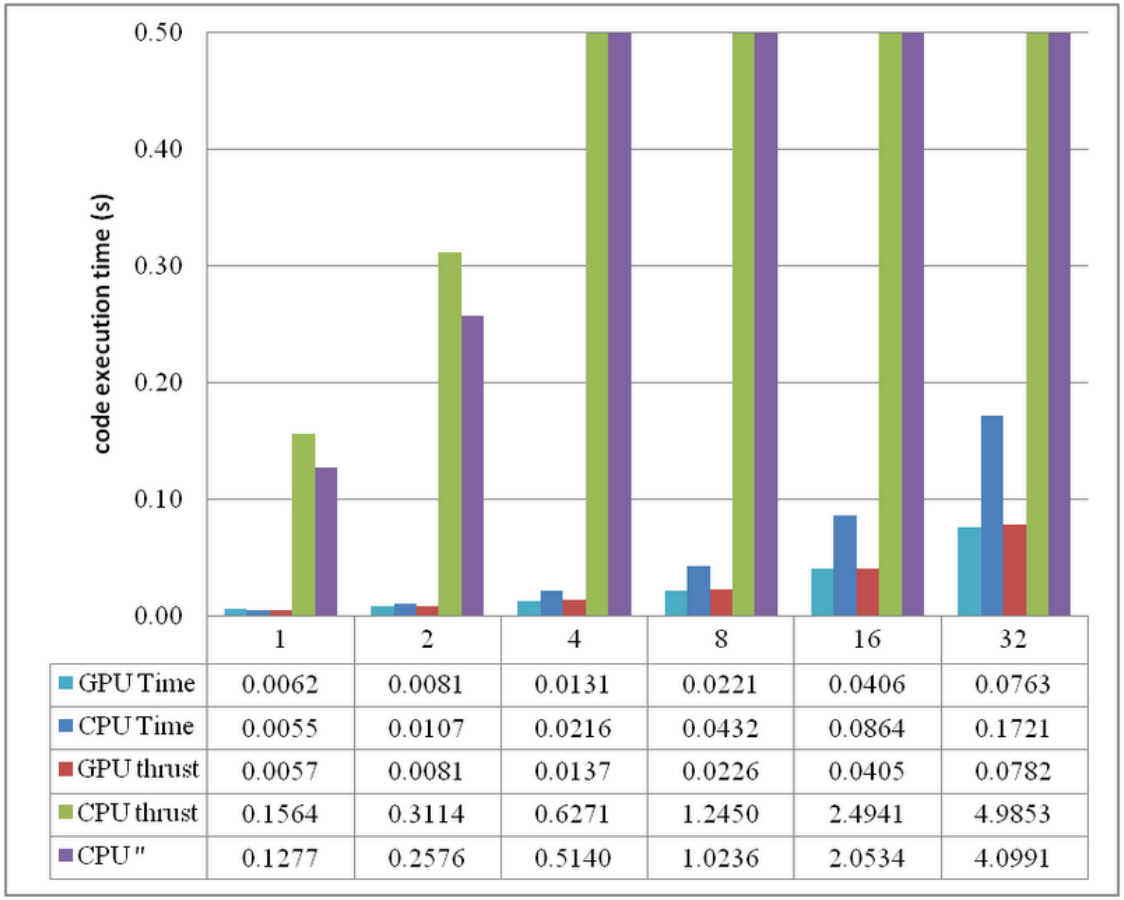
Перенос вектора на GPU

Подсчет суммы квадратов элементов на GPU

Источник: *A Brief Test on the Code Efficiency of CUDA and Thrust. Wayne Wood, 2010*
http://www.codeproject.com/KB/Parallel_Programming/test-on-thrust-efficiency.aspx?msg=3484497

Thrust: производительность

GPU Time	«чистая» CUDA
CPU Time	«чистый» C
GPU thrust	Thrust на CUDA
CPU Thrust	Thrust на CPU
CPU “	«чистый» C со структурами данных Thrust



	1	2	4	8	16	32
GPU Time	0.0062	0.0081	0.0131	0.0221	0.0406	0.0763
CPU Time	0.0055	0.0107	0.0216	0.0432	0.0864	0.1721
GPU thrust	0.0057	0.0081	0.0137	0.0226	0.0405	0.0782
CPU thrust	0.1564	0.3114	0.6271	1.2450	2.4941	4.9853
CPU "	0.1277	0.2576	0.5140	1.0236	2.0534	4.0991

Источник: *A Brief Test on the Code Efficiency of CUDA and Thrust. Wayne Wood, 2010*
http://www.codeproject.com/KB/Parallel_Programming/test-on-thrust-efficiency.aspx?msg=3484497

C++

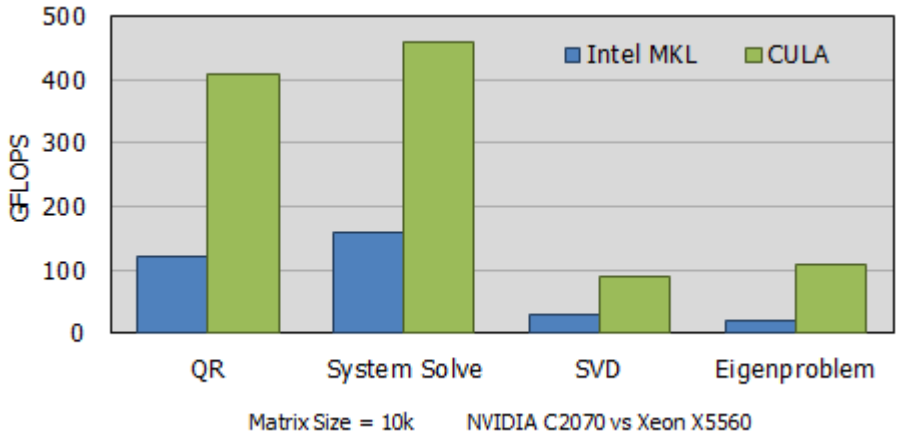
CULA (http://www.culatools.com/)

EM Photonics, 2009

Лицензия: коммерческая

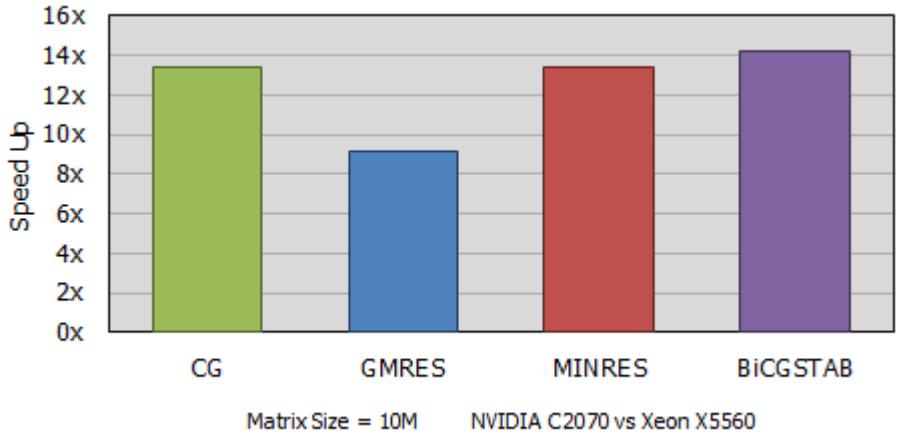
- **CULA Dense** – реализация LAPACK для CUDA, включает часто используемые алгоритмы операции линейной алгебры;
- **CULA Sparse** – набор реализаций итерационных методов решения СЛАУ с матрицами разреженного вида;
- Возможность использования в C/C++, Fortran, Matlab, Python.

CULA: производительность



Производительность CULA Dense

Источник:
Dense « CULA
(http://www.culatools.com/dense/)



Ускорение CULA Sparse

Источник:
Sparse « CULA
(http://www.culatools.com/sparse/)

C++

LibJacket (http://www.accelereyes.com/products/libjacket)

AccelerEyes, 2009
Лицензия: коммерческая

- построена на основе **CULA**;
- большой набор встроенных библиотек:
  - линейная алгебра,
  - обработка изображений/сигналов,
  - статистический анализ;
  - …
- возможность параллелизации циклов;
- поддержка нескольких GPU на одном узле и в кластере;
- возможность использования в C, C++, Python и Fortran.

LibJacket: пример

```
f32 A = f32::rand(n, n);
f32 B = f32::rand(n, n);

f32 X = lsolve(A, B);


float res = max_vector(abs(mtimes(A, X) - B));
```

Объявление и генерация случайных вещественных матриц размером n×n

Решение системы AX = B
A, X и B - матрицы

Вычисление максимальной ошибки ответа

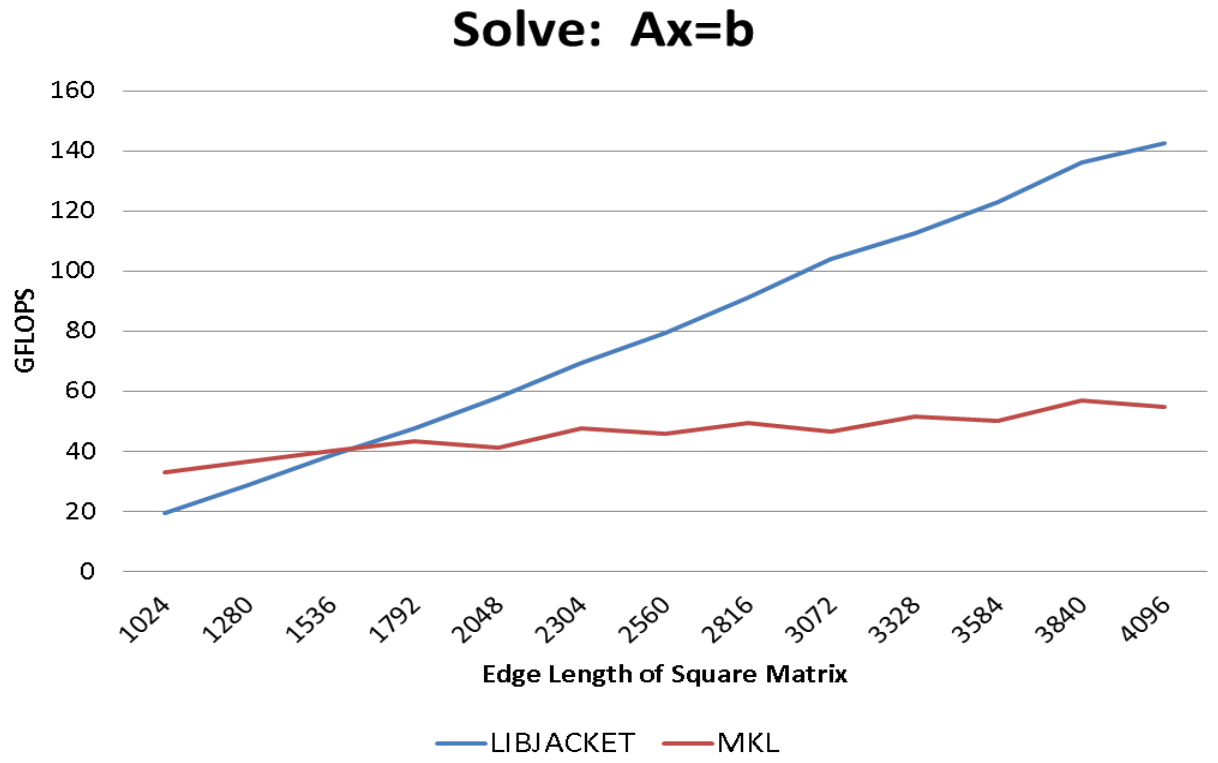
```
int n = 20e6;

f32 x = f32::rand(n,1), y = f32::rand(n,1);

float pi = 4.0 * sum_vector(sqrt(x*x + y*y) < 1) / n;
printf("pi = %g\n", pi);
```

Нахождение числа π
методом Монте-Карло

Источник: *LIBJACKET Documentation*
http://wiki.accelereyes.com/wiki/libjacket/

LibJacket: производительность

Решение Ax = b: NVIDIA Tesla C2050 vs. Intel i7-950 (3.06 GHz)
(LibJacket vs Intel Math Kernel Library)



Источник: *LibJacket Benchmarks*
(http://www.accelereyes.com/products/benchmarks_libjacket)

CULA vs LibJacket

- **LibJacket** использует **CULA** и предоставляет доступ ко всем ее базовым операциям через собственный интерфейс;
- Кроме того, **LibJacket** содержит прикладные подпрограммы:
  - обработки изображений (математическая морфология, геометрические преобразования, линейная и медианная фильтрация, арифметические операции);
  - обработки сигналов (одно-, двух- и многомерное преобразование Фурье, свертка, фильтрация);
  - алгебраические (векторное произведение, произведение Кронекера);
  - сортировки;
  - работы с матрицами (слияние, генерация перестановок, выделение диагоналей и треугольников);
  - специальных операций (тригонометрические, показательные, логические, поэлементные арифметические операции);
- **CULA Sparse** частично поддерживается расширением **LibJacket SLA**.

Java

(автоматическое управление памятью, средняя производительность, ООП, большой набор прикладных библиотек)

JCuda (http://jcuda.org/)

2009

Лицензия: MIT Licence (open-source)

- «привязка» к CUDA API:
  - API для работы с памятью GPU;
  - возможность запуска скомпилированных CUDA C-ядер.
- потенциальная необходимость писать на CUDA C;
- инфраструктура для дополнительных «привязок»:
  - JCublas;
  - JCufft;
  - …

JCuda: пример

```
Pointer deviceData = new Pointer();
JCuda.cudaMalloc(deviceData, memorySize);
float hostData[] = createInputData();
JCuda.cudaMemcpy(deviceData, Pointer.to(hostData), memorySize,
    cudaMemcpyKind.cudaMemcpyHostToDevice);

cufftHandle plan = new cufftHandle();
JCufft.cufftPlan1d(plan, complexElements, cufftType.CUFFT_C2C, 1);
JCufft.cufftExecC2C(plan, deviceData, deviceData, JCufft.CUFFT_FORWARD);

JCuda.cudaMemcpy(Pointer.to(hostData), deviceData, memorySize,
    cudaMemcpyKind.cudaMemcpyDeviceToHost);
JCuda.cudaFree(deviceData);
```

Выделение памяти на GPU

Генерация данных и копирование их на GPU

Вычисление ДПФ с помощью CUFFT

Копирование данных с GPU

Освобождение памяти на GPU

Источник: *JCuda Runtime API* (http://jcuda.org/jcuda/JCuda.html)

Python

(высокая скорость разработки, низкая производительность,
большое количество прикладных библиотек)

PyCUDA (http://mathema.tician.de/software/pycuda)

Andreas Klöckner, 2009
Лицензия: MIT Licence (open-source)

- работа памятью GPU;
- набор примитивных функций (min/max/триг./эксп.);
- интеграция с **NumPy**;
- интерфейс работы с компилятором CUDA C;
- трансляция ошибок CUDA в исключения Python.

PyCUDA: пример

```
a = numpy.random.randn(4,4)
a = a.astype(numpy.float32)
a_gpu = cuda.mem_alloc(a.nbytes)
cuda.memcpy_htod(a_gpu, a)

mod = SourceModule("""
   __global__ void doublify(float *a)
   {
     int idx = threadIdx.x + threadIdx.y*4;
     a[idx] *= 2;
   }
   """)
func = mod.get_function("doublify")
func(a_gpu, block=(4,4,1))
```

Генерация случайной вещественной матрицы 4×4

Выделение памяти на и копирование матрицы на GPU

Создание объекта – CUDA C ядра

Запуск ядра над матрицей на GPU

Использование встроенных функций PyCUDA:

```
a_gpu = gpuarray.to_gpu(numpy.random.randn(4,4).astype(numpy.float32))
a_doubled = (2*a_gpu).get()
```

Источник: *PyCUDA v2011.1.1 documentation: Tutorial Introduction* (http://documen.tician.de/pycuda/tutorial.html)

Matlab

(широкое распространение в научной среде,
большая база реализованных алгоритмов)

GPUmat (http://gp-you.org/)

The GP-you Group, 2010

Лицензия: freeware

- многие (но не все) встроенные функций Matlab могут работать на GPU;
- компилятор для собственных функций;
- прозрачная работа с памятью GPU;
- не поддерживаются встроенные в Matlab операции линейной алгебры.

GPUmat: пример

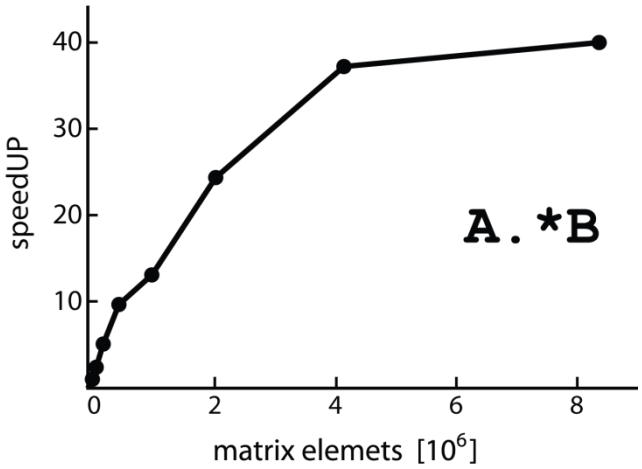
```
A = rand(100,100,GPUsingle);
B = rand(100,100,GPUsingle);
C = A.*B;
Ch = single(C);
```

Матрицы типа GPUsingle хранятся на GPU

Операция выполняется на GPU

Явное приведение типа копирует матрицу в память CPU

$$speedUP = \frac{T_{CPU}}{T_{GPU}}$$



Intel Core 2 Duo 6600 2.4 GHz

vs.

NVIDIA 8800GTX

Источник: *GPUmat. Performance* (http://gp-you.org/index.php?option=com_content&view=article&id=46&Itemid=54)

Matlab

Parallel Computing Toolbox (PCT)

(http://www.mathworks.com/products/parallel-computing/**)**

MathWorks, 2004

Лицензия: коммерческая

- расширение Matlab для использования многоядерных процессоров, кластеров и графических устройств для производства вычислений;
- ограниченное число встроенных функций, работающих на GPU;
- не поддерживается оператор среза массива;
- возможность запуска на нескольких GPU и на кластере хостов с GPU.

PCT: пример

```
function time = timeSolve(A, b)
    tic;
    x = A\b;
    time = toc;
end
% …
[A, b] = getData(n, clz);
A = gpuArray(A);
b = gpuArray(b);

time = timeSolve(A, b)
```

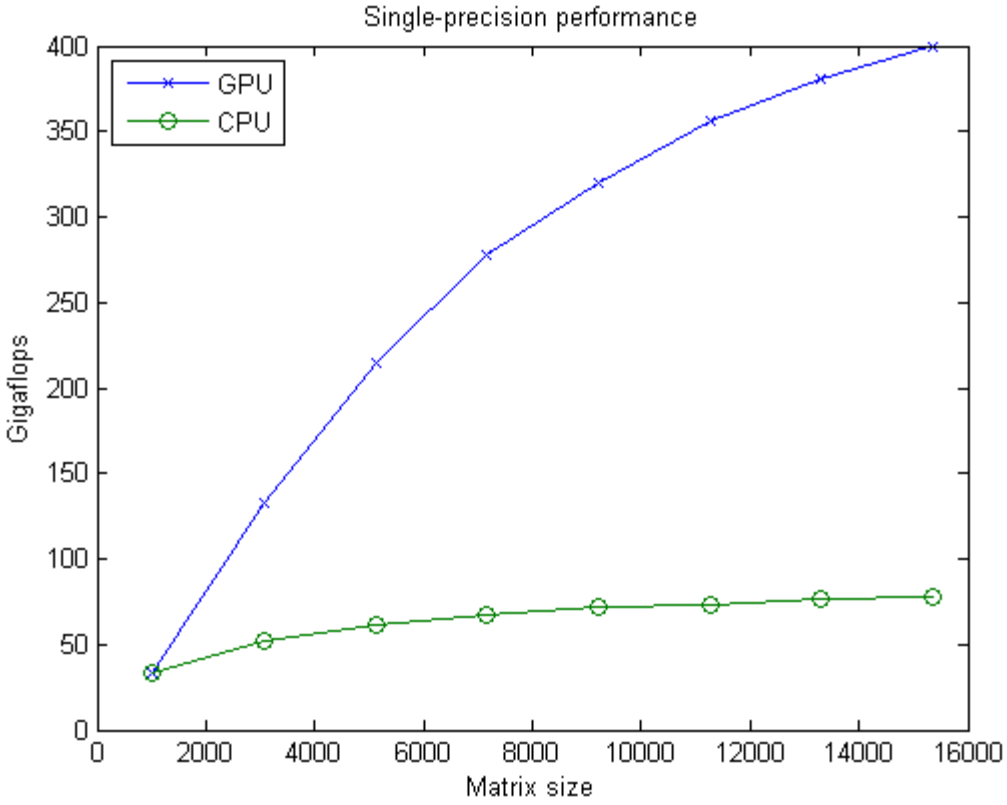
Функция, подсчитывающая время выполнения операции A^{-1} ·b

Генерация данных и их перенос на GPU

Подсчет времени вычисления решения системы Ax = b

Источник: *Parallel Computing Toolbox – Benchmarking A\b on GPU* (http://www.mathworks.com/products/demos/shipping/distcomp/paralleldemo_gpu_backslash.html)

PCT: производительность



Источник: *Parallel Computing Toolbox – Benchmarking A\b on GPU* (http://www.mathworks.com/products/demos/shipping/distcomp/paralleldemo_gpu_backslash.html)

Matlab

Jacket (http://www.accelereyes.com/products/jacket)

AccelerEyes, 2009

Лицензия: коммерческая

- значительно более широкий набор высокопроизводительных реализаций функций/алгоритмов;
- параллелизация циклов с декомпозицией по данным;
- возможность создания исполняемых файлов;
- возможность запуска на нескольких GPU и на кластере хостов с GPU (необходимо PCT).

Сравнение с GPUmat: http://www.accelereyes.com/products/compare_gpumat

Сравнение с Parallel Computing Toolbox: http://www.accelereyes.com/products/compare

Jacket: пример

```
G = gdouble(C);
…
G = fft(G);
G = G * G;

C = double(G);
```

Явное указание типа создает матрицу в памяти GPU

Операции выполняются на GPU

Явное указание типа создает копирует матрицу в память CPU

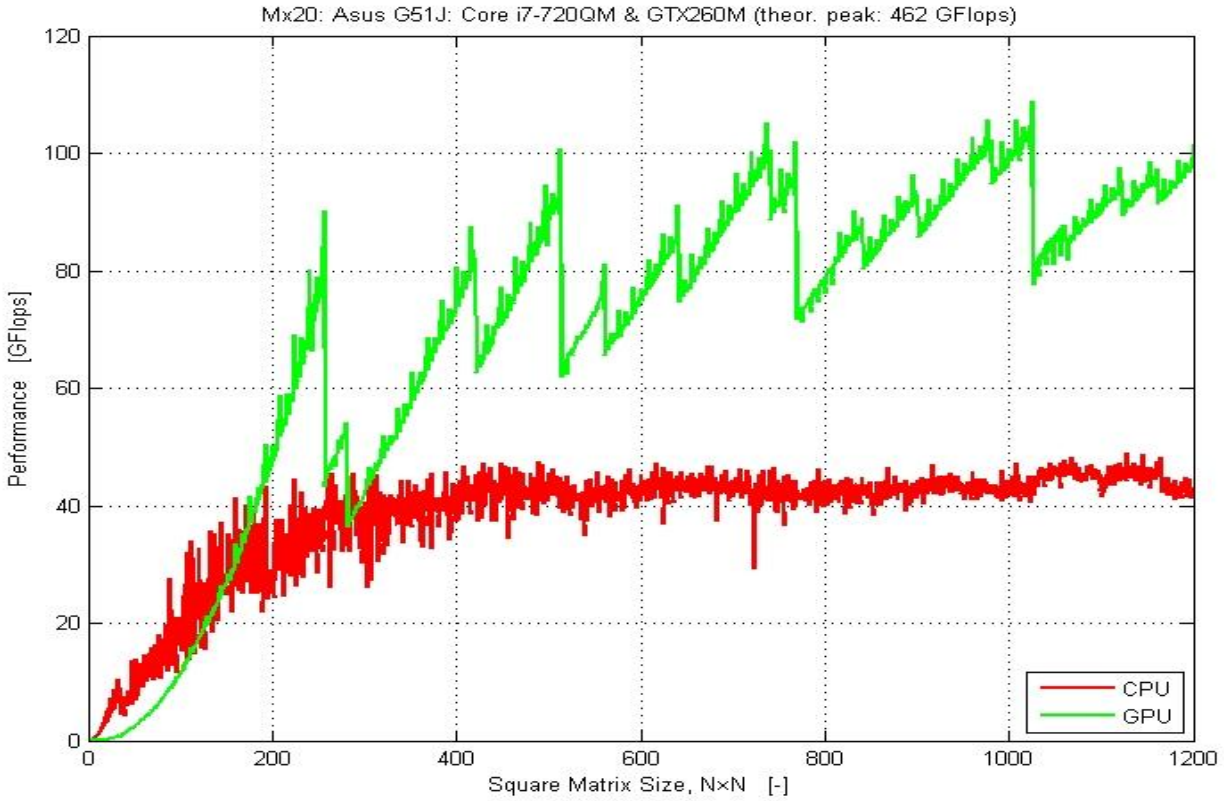
```
NSET = 1000000;
X = grand( 1, NSET );
Y = grand( 1, NSET );

distance_from_zero = sqrt( X.*X + Y.*Y );
inside_circle = (distance_from_zero <= 1);

pi = 4 * sum(inside_circle) / NSET;
```

Нахождение числа π методом Монте-Карло

Источник: *Jacket by Example* (http://wiki.accelereyes.com/wiki/index.php/Jacket_by_Example)

Jacket: производительность

Перемножение 20 матриц



Источник: Jacket Floating Point Performance (GFlops), 2010 (http://wiki.accelereyes.com/wiki/index.php/Jacket_Floating_Point_Performance_(GFlops))

Выбор инструмента

Задачи

1. Прототипирование.
2. Интеграция с существующим решением.
3. Разработка высокопроизводительного алгоритма с использованием конкретной платформы GPGPU.

Прототипирование

- **PyCUDA**
  - легкость освоения языка;
  - наличие примитивных функций;
- **GPUmat, PCT, Jacket**
  - легкость написания алгоритма в среде Matlab.

Интеграция с существующим решением

- C++
  - **Thrust**
    - простота работы с памятью GPU;
    - необходимость использования предопределенных структур данных.
- C++, Fortran
  - **LibJacket**
    - легкость переноса существующего алгоритма на GPU;
    - широкий набор встроенных библиотечных функций.
- Matlab
  - **GPUmat, PCT**
    - легкость переноса существующего алгоритма на GPU.
  - **Jacket**
    - широкий набор встроенных библиотечных функций.

Высокопроизводительная реализация алгоритма

- **CULA**
  - возможность использования при опыте работы с LAPACK.
- **JCuda, PyCUDA**
  - возможность оперировать на низком уровне с CUDA C;
  - автоматическое управление памятью CPU.
- **LibJacket**
  - широкий набор встроенных функций;
  - упрощенная работа с памятью GPU;
  - возможность работы на C++, Fortran и Python.

Выбор инструмента

Область применения / Инструмент	Прототипирование	Интеграция	Высоко-производительная реализация
Thrust		✓	
CULA			✓
LibJacket		✓	✓
JCuda			✓
PyCUDA	✓		✓
GPUmat	✓	✓	
PCT	✓	✓	
Jacket	✓	✓	