Посещаемость

Не забываем отмечаться и оставлять отзывы.

Спасибо!

ПРОГРАММИРОВАНИЕ CUDA C/C++, АНАЛИЗ ИЗОБРАЖЕНИЙ И DEEP LEARNING

Лекция №3

Спасёнов Алексей

Прикладные CUDA библиотеки и OpenACC

1. Библиотеки cuBLAS, cuRAND, cuSPARSE, MAGMA
2. Библиотека Thrust
3. Введение в OpenACC
4. Библиотека OpenCV

Программная модель CUDA

Compute Unified Device Architecture

GPU Computing Applications

Libraries and Middleware

cuFFT cuBLAS cuRAND cuSPARSE	CULA MAGMA	Thrust NPP	VSIPL SVM OpenCurrent	PhysX OptiX	iray	MATLAB Mathmatica

Programming Languages

C	C++	Fortran	Java Python Wrappers	Directives (e.g. OpenACC)

NVIDIA GPU

CUDA Parallel Computing Architecture

Прикладные CUDA библиотеки (1/2)

CUDA Toolkit

- cuFFT - Быстрое Преобразование Фурье
- cuBLAS - Библиотека линейной алгебры
- cuRAND - Генератор случайных чисел
- cuSPARSE - Работа с разряженными матрицами
- Thrust - Библиотека шаблонов
- NPP - Обработка сигналов, изображений

cuDNN - Примитивы для глубинных нейронных сетей (https://developer.nvidia.com/)

Прикладные CUDA библиотеки (2/2)

Сторонние библиотеки

- MAGMA - Линейная алгебра с плотными матрицами, аналог LAPACK
- CUSP - Линейная алгебра с разреженными матрицами
- OpenCurrent - Решатели уравнений в частных производных на регулярных сетках
- Accelereyes ArrayFire - Контейнеры для хранения данных и операции над ними

Библиотека cuBLAS

Basic Linear Algebra Subprograms

Основные операции линейной алгебры

- Создание матриц и векторных объектов
- Заполнение матриц и векторов данными
- Определение ранга матрицы
- Преобразование матриц
- Решение СЛАУ

Библиотека cuBLAS

Перемножение матриц

$$\boldsymbol{C = \alpha AB + \beta C}$$

- $\boldsymbol{\alpha, \beta}$ - скалярные величины
- $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ - матрицы

Библиотека cuBLAS

Функция MAIN

```
1.   int main()
2.   { ...
3.       int m = M; // Матрица A размерностью MxK
4.       int n = N; // Матрица B размерностью KxN
5.       int k = K; // Матрица C размерностью MxN
6.       cudaError_t  cudaStat;   //       cudaMalloc статус
7.       cublasStatus_t stat;     //       cuBLAS статус функциии
8.       cublasHandle_t handle;   //       cuBLAS контекст
```

Библиотека cuBLAS

Функция MAIN

```
9.  // Выделение памяти на host
10. a = (float *) malloc (m*k* sizeof (float)); // Память для матрицы A
11. b = (float *) malloc (k*n* sizeof (float)); // Память для матрицы B
12. c = (float *) malloc (m*n* sizeof (float)); // Память для матрицы C
13.  { ... } // Генерация матрицы A
14.  { ... } // Генерация матрицы B
15.  { ... } // Генерация матрицы C
```

Библиотека cuBLAS

Функция MAIN

```
16.  // Выделение памяти на device
17.   cudaStat = cudaMalloc((void **)& d_a, m*k* sizeof(*a)); // Память для матрицы A
18.   cudaStat = cudaMalloc((void **)& d_b, k*n* sizeof(*b)); // Память для матрицы B
19.   cudaStat = cudaMalloc((void **)& d_c, m*n* sizeof(*c)); // Память для матрицы C
20.    // Копирование данных с host на device
21.    stat = cublasSetMatrix(m, k, sizeof (*a), a, m, d_a, m);
22.    stat = cublasSetMatrix(k, n, sizeof (*b), b, k, d_b, k);
23.    stat = cublasSetMatrix(m, n, sizeof (*c), c, m, d_c, m);
```

Библиотека cuBLAS

Функция MAIN

```
24.  // Скалярные величины
25.  float al  = 1.0f;
26.  float bet = 1.0f;
27.  // Перемножение матриц с использованием cuBLAS
28.   stat = cublasSgemm( handle, CUBLAS_OP_N, CUBLAS_OP_N,
29.      m, n, k, &al, d_a, m, d_b, k, &bet, d_c, m);
30.   // Копирование данных с device на host
31.   stat = cublasGetMatrix(m, n, sizeof (*c), d_c, m, c, m);
```

Библиотека cuBLAS

Функция MAIN

```
32. // Освобождение памяти на host и device
33.   cudaFree ( d_a );
34.   cudaFree ( d_b );
35.   cudaFree ( d_c );
36.   free ( a );
37.   free ( b );
38.   free ( c );
39.  // Уничтожение cuBLAS контекста
40.   cublasDestroy ( handle );
```

Библиотека cuBLAS

Анализ результатов

CPU Core i7-4770K 4.00GHz (1 ядро) GPU GeForce GTX 780Ti

Лекция 2		cuBLAS	
* GPU time:	18.87 ms	* GPU time:	4.56 ms
Rate:	195 x	Rate:	811 x
** GPU time:	16.93 ms	** GPU time:	1.66 ms
Rate:	219 x	Rate:	2229 x

CPU time: 3700 ms

* - Время выполнения с учётом копирования данных «HOST» - «DEVICE»

** - Время выполнения ТОЛЬКО операции умножения матриц

Библиотека cuRAND (1/3)

Генерация случайных чисел

Реализовано 4 алгоритма генерации случайных чисел:

- Генератор «Вихрь Мерсенна»
- Генератор псевдо-случайных чисел XORWOW
- Генераторы квази-случайных чисел Соболя
- MRG32k3a (Multiple Recursive Generator)

Библиотека cuRAND (2/3)

Генерация случайных чисел

Распределения:

- Равномерное распределение
- Нормальное распределение
- Логарифмически нормальное распределение
- Распределения одинарной и двойной точности
- Распределение Пуассона

Библиотека cuRAND (3/3)

Генерация случайных чисел

- Работает как на стороне GPU, так и на стороне CPU
- При одних и тех же начальных условиях последовательности чисел будут одинаковые. (на GPU и CPU)

Библиотека cuRAND

Использование:

1) Создание генератора
   - GPU: curandCreateGenerator()
   - CPU: curandCreateGeneratorHost()
2) Установка параметров
3) Выделение памяти
   - GPU: cudaMalloc()
   - CPU: cudaMallocHost()
4) Генерация последовательности
5) Использование результата
6) Освобождение памяти
7) Удаление генератора
   - curandDestroyGenerator()

Библиотека cuRAND

Функция MAIN

```
1.  #include <cuda.h>
2.  #include <curand.h>
3.  int main()
4.  {
5.      ...
6.      curandGenerator_t gen; // cuRAND генератор
7.      ...
8.      // Выделение памяти на host и device
9.      hostData = (float *) calloc (n, sizeof(float));
10.     cudaMalloc ((void **)&devData, n*sizeof(float));
11.     ...
```

Библиотека cuRAND

Функция MAIN

```
12.     // Создание генератора и заполнение массива данными
13.     curandCreateGenerator (&gen, CURAND_RNG_PSEUDO_DEFAULT);
14.     curandSetPseudoRandomGeneratorSeed (gen, 1234ULL);
15.     curandGenerateUniform (gen, devData, n);
16.     ...
17.     // копирование данных с device на host
18.     cudaMemcpy (hostData, devData, mem_size, cudaMemcpyDeviceToHost);
19.     // Уничтожение генератора
20.     curandDestroyGenerator (gen);
21.     cudaFree ( devData );  free ( hostData ); // Освобождение памяти
22. }
```

Библиотека cuRAND

Анализ результатов

CPU Core i7-4770K 4.00GHz (1 ядро) GPU GeForce GTX 780Ti

* GPU time:	12.47 ms	** GPU time:	18.76 ms
* CPU time:	21.00 ms	** CPU time:	5258.00 ms
Rate:	1.68 x	Rate:	280.28 x

* - Размер массива 1024x1024

** - Размер массива 16384x16384

Библиотека cuRAND

Генерация случайных чисел «на лету»

```
1.  #include <cuda.h>
2.  #include <curand_kernel.h> // #include <curand.h>


3.   // Инициализация cuRAND-состояний
4.   __global__ void stateKernel (curandState *state) {
5.       int tid = blockIdx.x * blockDim.x + threadIdx.x;
6.       curand_init (1234, tid, 0, &state[tid]);
7.   }
```

Библиотека cuRAND

Генерация случайных чисел «на лету»

```
8.   __global__ void generateKernel(curandState *state, int *data) {
9.       int tid = blockIdx.x * blockDim.x + threadIdx.x;
10.      int value = 0;
11.      float x = 0.f;

12.      curandState localState = state[tid];
13.
14.      x = curand_uniform (&localState);  // Равномерное распределение
15.      value = static_cast<int> (x * 10); // 0...10

16.      state[tid] = localState;

17.      data[tid] = value;
18.  }
```

Библиотека cuRAND

Генерация случайных чисел «на лету»

```
1.  int main() {
2.      curandState *devStates;
3.      int *devData, *hostData;

4.      int arraySize = 512 * 10;
5.
6.      int N_Threads = 512;
7.      int N_Blocks = 0;

8.      if (((arraySize) % N_Threads) == 0) {
9.              N_Blocks = ((arraySize) / N_Threads);
10.     }
11.     else {
12.             N_Blocks = ((arraySize) / N_Threads) + 1;
13.     }

14.     dim3 Threads(N_Threads);
15.     dim3 Blocks(N_Blocks);
```

Библиотека cuRAND

Генерация случайных чисел «на лету»

```
16.     hostData = (int*)calloc(arraySize, sizeof(int));

17.     cudaMalloc ((void**)&devData, arraySize*sizeof(int));
18.     cudaMemset (devData, 0, arraySize*sizeof(int));

19.     cudaMalloc ((void**)&devStates, arraySize*sizeof(curandState));
20.     stateKernel <<< Blocks, Threads >>> (devStates);

21.     generateKernel <<< Blocks, Threads >>> (devStates, devData);

22.     cudaMemcpy (hostData, devData, arraySize*sizeof(int), cudaMemcpyDeviceToHost);
```

Библиотека cuRAND

Генерация случайных чисел «на лету»

```
23.     cudaFree(devStates);
24.     cudaFree(devData);
25.     free(hostData);

26.     return 0;
27. }
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <cuda_runtime.h>
4.  #include "cusparse.h"

5.  int main(){
6.      cudaError_t cudaStat1,cudaStat2,cudaStat3,cudaStat4,cudaStat5,cudaStat6;
7.      cusparseStatus_t status;
8.      cusparseHandle_t handle=0;
9.      cusparseMatDescr_t descr=0;
```

http://docs.nvidia.com/cuda/cusparse/#appendix-b-cusparse-library-c---example

Библиотека cuSPARSE

Работа с разреженными матрицами

```
10.     int * cooRowIndexHostPtr=0;
11.     int * cooColIndexHostPtr=0;
12.     double * cooValHostPtr=0;

13.     int * cooRowIndex=0;
14.     int * cooColIndex=0;
15.     double * cooVal=0;

16.     int * xIndHostPtr=0;
17.     double * xValHostPtr=0;
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
18.     double * yHostPtr=0;
19.     int * xInd=0;
20.     double * xVal=0;
21.     double * y=0;
22.     int * csrRowPtr=0;
23.     double * zHostPtr=0;
24.     double * z=0;
25.     int n, nnz, nnz_vector;
26.     double dzero =0.0;
27.     double dtwo =2.0;
28.     double dthree=3.0; double dfive =5.0;
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
29.     /* Разреженная матрица   */
30.     /* |1.0     2.0 3.0|
31.        |    4.0         |
32.        |5.0     6.0 7.0|
33.        |    8.0     9.0| */
34.        n=4; nnz=9;
35.        cooRowIndexHostPtr = (int *) malloc (nnz*sizeof(cooRowIndexHostPtr[0]));
36.        cooColIndexHostPtr = (int *) malloc (nnz*sizeof(cooColIndexHostPtr[0]));
37.        cooValHostPtr = (double*) malloc (nnz*sizeof(cooValHostPtr[0]));
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
38.     cooRowIndexHostPtr[0]=0; cooColIndexHostPtr[0]=0; cooValHostPtr[0]=1.0;
39.     cooRowIndexHostPtr[1]=0; cooColIndexHostPtr[1]=2; cooValHostPtr[1]=2.0;
40.     cooRowIndexHostPtr[2]=0; cooColIndexHostPtr[2]=3; cooValHostPtr[2]=3.0;
41.     cooRowIndexHostPtr[3]=1; cooColIndexHostPtr[3]=1; cooValHostPtr[3]=4.0;
42.     cooRowIndexHostPtr[4]=2; cooColIndexHostPtr[4]=0; cooValHostPtr[4]=5.0;
43.     cooRowIndexHostPtr[5]=2; cooColIndexHostPtr[5]=2; cooValHostPtr[5]=6.0;
44.     cooRowIndexHostPtr[6]=2; cooColIndexHostPtr[6]=3; cooValHostPtr[6]=7.0;
45.     cooRowIndexHostPtr[7]=3; cooColIndexHostPtr[7]=1; cooValHostPtr[7]=8.0;
46.     cooRowIndexHostPtr[8]=3; cooColIndexHostPtr[8]=3; cooValHostPtr[8]=9.0;
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
47.     /* Создание разреженного и полного вектора */
48.     /* xVal= [100.0 200.0 400.0] (sparse)
49.        xInd= [0 1 3 ]
50.        y = [10.0 20.0 30.0 40.0 | 50.0 60.0 70.0 80.0] (dense) */
51.     nnz_vector = 3;
52.     xIndHostPtr = (int *) malloc (nnz_vector*sizeof(xIndHostPtr[0]));
53.     xValHostPtr = (double *) malloc (nnz_vector*sizeof(xValHostPtr[0]));
54.     yHostPtr = (double *) malloc (2*n *sizeof(yHostPtr[0]));
55.     zHostPtr = (double *) malloc (2*(n+1) *sizeof(zHostPtr[0]));
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
56.     yHostPtr[0] = 10.0;   xIndHostPtr[0]=0;   xValHostPtr[0]=100.0;
57.     yHostPtr[1] = 20.0;   xIndHostPtr[1]=1;   xValHostPtr[1]=200.0;
58.     yHostPtr[2] = 30.0;
59.     yHostPtr[3] = 40.0;   xIndHostPtr[2]=3;   xValHostPtr[2]=400.0;
60.     yHostPtr[4] = 50.0;
61.     yHostPtr[5] = 60.0;
62.     yHostPtr[6] = 70.0;
63.     yHostPtr[7] = 80.0;
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
64.     cudaStat1 = cudaMalloc((void**)&cooRowIndex,nnz*sizeof(cooRowIndex[0]));
65.     cudaStat2 = cudaMalloc((void**)&cooColIndex,nnz*sizeof(cooColIndex[0]));
66.     cudaStat3 = cudaMalloc((void**)&cooVal, nnz*sizeof(cooVal[0]));
67.     cudaStat4 = cudaMalloc((void**)&y, 2*n*sizeof(y[0]));
68.     cudaStat5 = cudaMalloc((void**)&xInd,nnz_vector*sizeof(xInd[0]));
69.     cudaStat6 = cudaMalloc((void**)&xVal,nnz_vector*sizeof(xVal[0]));

70.     // Проверка cudaStat1, cudaStat2, cudaStat3, …
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
71. cudaStat1 = cudaMemcpy(cooRowIndex, cooRowIndexHostPtr,
                           (size_t)(nnz*sizeof(cooRowIndex[0])), cudaMemcpyHostToDevice);
72. cudaStat2 = cudaMemcpy(cooColIndex, cooColIndexHostPtr,
                           (size_t)(nnz*sizeof(cooColIndex[0])), cudaMemcpyHostToDevice);
73. cudaStat3 = cudaMemcpy(cooVal, cooValHostPtr,
                           (size_t)(nnz*sizeof(cooVal[0])), cudaMemcpyHostToDevice);
74. cudaStat4 = cudaMemcpy(y, yHostPtr, (size_t)(2*n*sizeof(y[0])), cudaMemcpyHostToDevice);
75. cudaStat5 = cudaMemcpy(xInd, xIndHostPtr,
                           (size_t)(nnz_vector*sizeof(xInd[0])), cudaMemcpyHostToDevice);
76. cudaStat6 = cudaMemcpy(xVal, xValHostPtr,
                            (size_t)(nnz_vector*sizeof(xVal[0])), cudaMemcpyHostToDevice);
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
77.     /* Инициализируем библиотеку CUSPARSE */
78.     status = cusparseCreate(&handle);

79.     /* Создаём дескриптор матрицы */
80.     status = cusparseCreateMatDescr(&descr);

81.     cusparseSetMatType(descr, CUSPARSE_MATRIX_TYPE_GENERAL);
82.     cusparseSetMatIndexBase(descr, CUSPARSE_INDEX_BASE_ZERO);
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
85.     /* Преобразование матрицы из формата COO в формат CSR */
86.     cudaStat1 = cudaMalloc((void**)&csrRowPtr,(n+1)*sizeof(csrRowPtr[0]));

87.     status = cusparseXcoo2csr(handle,cooRowIndex,nnz,n, csrRowPtr, CUSPARSE_INDEX_BASE_ZERO);

88.     int devId;
89.     cudaDeviceProp prop;
90.     cudaError_t cudaStat;
91.     cudaStat = cudaGetDevice(&devId);
92.     cudaStat = cudaGetDeviceProperties( &prop, devId) ;
93.     int cc = 100*prop.major + 10*prop.minor;      // Проверка CC > 130
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
94.     /* Умножение разреженного вектора на плотный */
95.     status= cusparseDsctr(handle, nnz_vector, xVal, xInd, &y[n], CUSPARSE_INDEX_BASE_ZERO);
96.     //y = [10 20 30 40 | 100 200 70 400]

97.      /* Умножение разреженной матрицы на плотный вектор */
98.      status= cusparseDcsrmv(handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
99.                             n, n, nnz, &dtwo, descr,
100.                            cooVal, csrRowPtr, cooColIndex, &y[0], &dthree, &y[n]);

101.     //y = [10 20 30 40 | 680 760 1230 2240]
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
102.    cudaMemcpy(yHostPtr, y, (size_t)(2*n*sizeof(y[0])), cudaMemcpyDeviceToHost);

103.    /* Умножение разреженной матрицы на плотный вектор-столбец */
104.    cudaStat1 = cudaMalloc((void**)&z, 2*(n+1)*sizeof(z[0]));
105.    cudaStat1 = cudaMemset((void *)z,0, 2*(n+1)*sizeof(z[0]));

106.    status= cusparseDcsrmm(handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
107.                           n, 2, n, nnz, &dfive, descr, cooVal,
108.                           csrRowPtr, cooColIndex, y, n, &dzero, z, n+1);
```

Библиотека cuSPARSE

Работа с разреженными матрицами

```
109.     /* Копируем (z) */
110.     cudaStat1 = cudaMemcpy(zHostPtr, z,
111.                            (size_t)(2*(n+1)*sizeof(z[0])), cudaMemcpyDeviceToHost);

112.     /* Уничтожаем дескриптор матрицы */
113.     status = cusparseDestroyMatDescr(descr);
114.     status = cusparseDestroy(handle);
115. }
```

Библиотека MAGMA

Ускорение линейной алгебры нового поколения

- Высокая производительность и точность
- Поддержка арифметических операций с различными уровнями точности
- Гибридные алгоритмы, использующие GPU и CPU

HTTP://ICL.CS.UTK.EDU/MAGMA/SOFTWARE/

Библиотека MAGMA

Функция MAIN

Решение СЛАУ

$$A\,X = B$$

```
1.  #include <magma.h>
2.  #include <magma_lapack.h>

3.  int main()
4.  {...
5.      magma_init ();                          // Инициализация MAGMA
6.      magma_timestr_t start,end;
7.      ...
8.      magma_int_t m = 2*8192;                 // Размерность матрицы A(mxm)
9.      magma_int_t n = 1;                      // Размерность матрицы B(mxn)
10.     ...
11.     magma_int_t mm = m*m;
12.     magma_int_t mn = m*n;
13.     ...
14.     magma_int_t ISEED [4] = {0 ,0 ,0 ,1};  // Seed для генерации
```

Библиотека MAGMA

Функция MAIN

```
15.     // Выделение памяти на host
16.     magma_smalloc_pinned (&a , mm );
17.     magma_smalloc_pinned (&b , mm );
18.     magma_smalloc_pinned (&c , mm );
19.     // Генерация матрицы A
20.     lapackf77_slarnv (&ione ,ISEED ,&mm ,a);
21.     // Генерация единичной матрицы B
22.     lapackf77_slaset ( MagmaUpperLowerStr ,&m ,&n ,& alpha ,& alpha ,b ,&m);
```

Библиотека MAGMA

Функция MAIN

```
23.     start = get_current_time ();                  // Время начала расчётов
24.     magma_sgesv (m,n,a,m,piv,c,m,&info);          // Решение системы
25.     end = get_current_time ();                    // Время окончания расчетов
26.     gpu_time = GetTimerValue (start ,end )/1e3;   // Перевод времени
27.     ...
28.     // Освобождение памяти
29.     magma_free_pinned (a);
30.     magma_free_pinned (b);
31.     magma_free_pinned (c);
32.     return 0;
33. }
```

Библиотека Thrust

- Высокоуровневый интерфейс для программирования на GPU
- STL-подобная библиотека обработки данных на GPU
- Реализация как для GPU, так и для CPU

Библиотека THRUST

Сортировка массива

```
1.  #include <time.h>
2.  #include <thrust/host_vector.h>
3.  #include <thrust/device_vector.h>
4.  #include <thrust/generate.h>
5.  #include <thrust/sort.h>
6.  #include <thrust/copy.h>
7.  #include <thrust/reduce.h>

8.  int main() {
9.      float CPUstart, CPUstop;
10.     float GPUstart, GPUstop;
11.     float CPUtime = 0.0f;
12.     float GPUtime = 0.0f;
```

Библиотека THRUST

Сортировка массива

```
13.     thrust::host_vector<int> hostVector(32<<22);
14.     thrust::generate(hostVector.begin(), hostVector.end(), rand);

15.     thrust::device_vector<int> deviceVector = hostVector;

16.     CPUstart = clock();

17.     thrust::sort(hostVector.begin(), hostVector.end());

18.     CPUstop = clock();
19.     CPUtime = 1000.*(CPUstop - CPUstart) / CLOCKS_PER_SEC;
20.     printf("CPU time : %.3f ms\n", CPUtime);
```

Библиотека Thrust

Сортировка массива

```
21.     GPUstart = clock();
22.
23.     thrust::sort(deviceVector.begin(), deviceVector.end());
24.
25.     GPUstop = clock();
26.     GPUtime = 1000.*(GPUstop - GPUstart) / CLOCKS_PER_SEC;
27.     printf(“GPU time : %.3f ms\n", GPUtime);

28.     printf("ArraySize :  %d \n", 32<<22);
29.     printf("Rate : %.3f \n", CPUtime/GPUtime);
30.     return 0;
31. }
```

Библиотека Thrust

Анализ результатов

CPU Core i7-3610QM 2.30GHz GPU GeForce 650M

GPU time: 658.533 ms

CPU time: 4457.360 ms

Rate: 6.767

Библиотека THRUST

Сумма элементов массива

```
1.  #include <time.h>
2.  #include <thrust/host_vector.h>
3.  #include <thrust/device_vector.h>
4.  #include <thrust/generate.h>
5.  #include <thrust/sort.h>
6.  #include <thrust/copy.h>
7.  #include <thrust/reduce.h>

8.  int main() {
9.      float CPUstart, CPUstop;
10.     float GPUstart, GPUstop;
11.     float CPUtime = 0.0f, GPUtime = 0.0f;
12.     int CPUresult = 0, GPUresult = 0;
```

Библиотека THRUST

Сумма элементов массива

```
13.     thrust::host_vector<int> hostVector(32<<22);
14.     thrust::generate(hostVector.begin(), hostVector.end(), rand);

15.     thrust::device_vector<int> deviceVector = hostVector;

16.     CPUstart = clock();

17.     CPUresult = thrust::reduce(hostVector.begin(), hostVector.end(), 0,
                                                          thrust::plus<int>());
13.     CPUstop = clock();
14.     CPUtime = 1000.*(CPUstop - CPUstart) / CLOCKS_PER_SEC;
15.     printf("CPU time : %.3f ms\n", CPUtime);
```

Библиотека THRUST

Сумма элементов массива

```
21.     GPUstart = clock();
22.
23.     GPUresult = thrust::reduce(deviceVector.begin(), deviceVector.end(), 0,
                                                                thrust::plus<int>());
21.     GPUstop = clock();
22.     GPUtime = 1000.*(GPUstop - GPUstart) / CLOCKS_PER_SEC;
23.     printf(“GPU time : %.3f ms\n", GPUtime);

24.     printf("ArraySize :  %d \n", 32<<22);
25.     printf("Rate : %.3f \n", CPUtime/GPUtime);
26.     return 0;
27. }
```

Библиотека Thrust

Анализ результатов

CPU Core i7-3610QM 2.30GHz GPU GeForce 650M

GPU time: 2933.422 ms

CPU time: 6644.201 ms

Rate: 2.265

Библиотека THRUST

Сумма векторов

```
1.  int main() {
2.      float CPUstart, CPUstop;
3.      float GPUstart, GPUstop;
4.      float CPUtime = 0.0f, GPUtime = 0.0f;
5.      thrust::host_vector<int> hostVectorA(32<<20);
6.      thrust::host_vector<int> hostVectorB(32<<20);
7.      thrust::host_vector<int> hostVectorC(32<<20);
8.      thrust::generate(hostVectorA.begin(), hostVectorA.end(), rand);
9.      thrust::generate(hostVectorB.begin(), hostVectorB.end(), rand);
10.     CPUstart = clock();

11.     thrust::transform(hostVectorA.begin(), hostVectorA.end(),hostVectorB.begin(),
                                                hostVectorC.begin(), thrust::plus<int>());
12.     CPUstop = clock();
13.     CPUtime = 1000.*(CPUstop - CPUstart) / CLOCKS_PER_SEC;
14.     printf("CPU time : %.3f ms\n", CPUtime);
```

Библиотека THRUST

Сумма векторов

```
15.     thrust::device_vector<int> deviceVectorA = hostVectorA;
16.     thrust::device_vector<int> deviceVectorB = hostVectorB;
17.     thrust::device_vector<int> deviceVectorC(32<<20);
18.     GPUstart = clock();

19.     thrust::transform(deviceVectorA.begin(), deviceVectorA.end(), deviceVectorB.begin(),
                                                  deviceVectorC.begin(), thrust::plus<int>());

20.     GPUstop = clock();
21.     GPUtime = 1000.*(GPUstop - GPUstart) / CLOCKS_PER_SEC;
22.     printf("GPU time : %.3f ms\n", GPUtime);
23.     printf("ArraySize :  %d \n", 32<<22);
24.     printf("Rate : %.3f \n", CPUtime/GPUtime);
25.     return 0;
26. }
```

Библиотека Thrust

Анализ результатов

CPU Core i7-3610QM 2.30GHz GPU GeForce 650M

GPU time: 520.477 ms

CPU time: 8625.582 ms

Rate: 16.572

Библиотека Thrust

SAXPY

$$Z = aX + Y$$

- X, Y, Z — вектора
- a - скаляр

Библиотека THRUST

Сумма элементов массива

```
1.  struct saxpy {
2.      int a;
3.      saxpy(int a): a(a) {}
4.      __host__ __device__ float operator() (int x, int y)
5.      {
6.          return a*x + y;
7.      }
8.  };
```

Библиотека THRUST

Сумма элементов массива

```
1.      //...
2.      thrust::transform(hostVectorA.begin(), hostVectorA.end(),hostVectorB.begin(),
                                                        hostVectorC.begin(), saxpy(a));
3.      //...
4.      thrust::transform(deviceVectorA.begin(), deviceVectorA.end(), deviceVectorB.begin(),
                                                      deviceVectorC.begin(), saxpy(a));
5.      //...
```

Библиотека Thrust

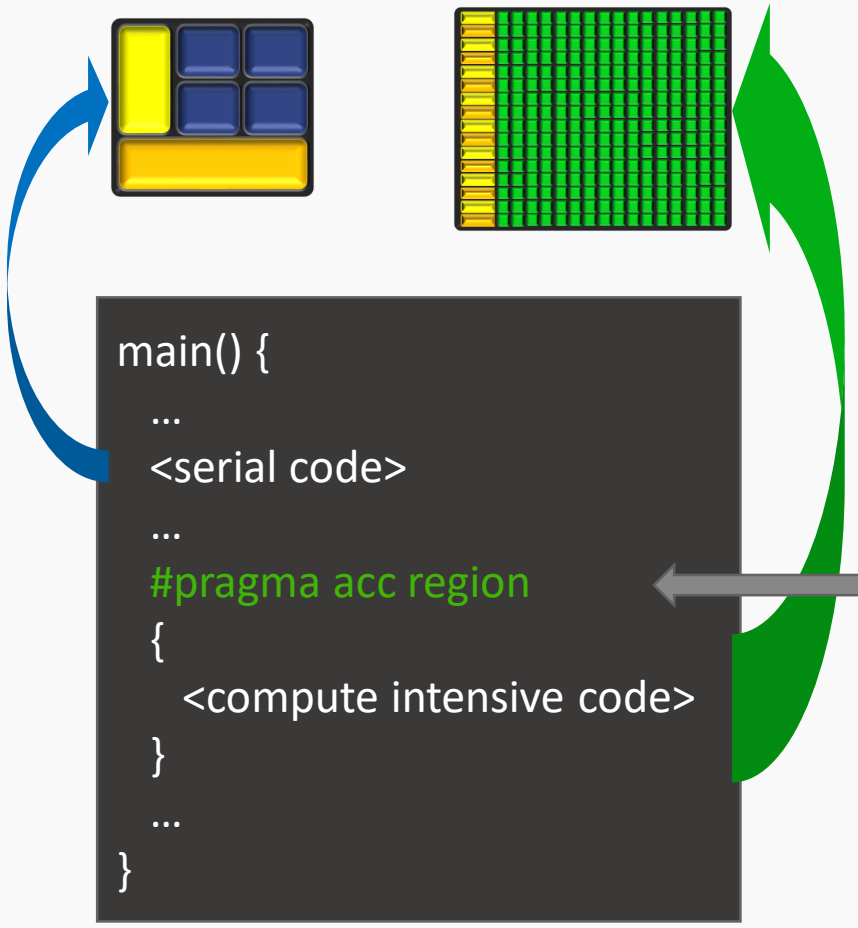
Анализ результатов

CPU Core i7-3610QM 2.30GHz GPU GeForce 650M

GPU time: 518.279 ms

CPU time: 8685.321 ms

Rate: 16.758



OpenACC

- CRAY, CAPS, PGI, NVIDIA
- Открытый стандарт
- Простота
- Использование на GPU

Выделение части кода, выполняемого на GPU

Основные директивы OpenACC

- PARALLEL
  - Основные атрибуты: IF, ASYNC, NUM_GANGS, NUM_WORKER, VECTOR_LENGTH, PRIVATE, FIRST_PRIVATE, REDUCTION
  - Атрибуты для данных: COPY, COPYIN, COPYOUT, CREATE, PRESENT, PRESENT_OR_COPY, PRESENT_OR_CREATE, DEVICEPTR, PRIVATE, FIRSTPRIVATE
- KERNELS
  - Основные атрибуты: IF, ASYNC
  - Атрибуты для данных: COPY, COPYIN, COPYOUT, CREATE, PRESENT, PRESENT_OR_COPY, PRESENT_OR_CREATE, DEVICEPTR, PRIVATE, FIRSTPRIVATE

OpenACC

#PRAGMA ACC <директива> атрибуты

- LOOP
  - Атрибуты: COLLAPSE, GANG, WORKER, VECTOR, SEQ, INDEPENDENT, PRIVATE, REDUCTION
- DATA
  - Атрибуты: ARRAY [начало:дина]

- Комбинированные директивы
  - PARALLEL LOOP
  - KERNELS LOOP

OpenACC

PGI компилятор языка «C»

pgcc –acc –ta=tesla:cuda7.0, time –Minfo=accel program.c

-acc - использование директив OpenACC

-ta - целевая архитектура, время

-Minfo - информация по оптимизации

Директива PARALLEL

```
#include <openacc.h>
//...
35.   #pragma acc parallel {
36.       for(int i=0; i<N; i++) a[i] = sqrtf(i);
37.       for(int j=0; j<N; j++) b[j] = tanf(j);
38.   }
```

```
35. Accelerator kernel generated
      35. #pragma acc loop vector(256) /* threadIdx.x */
      36. #pragma acc loop vector(256) /* threadIdx.x */
36. Generating present_or_copyout(a[:100000])
    Generating present_or_copyout(b[:100000])
    Generating Tesla code
37. Loop is parallelizable
38. Loop is parallelizable
```

Директива PARALLEL

```
#include <openacc.h>
//...
35.  #pragma acc kernels {
36.      for(int i=0; i<N; i++) a[i] = sqrtf(i);
37.      for(int j=0; j<N; j++) b[j] = tanf(j);
38.  }
```

```
35. Generating present_or_copyout(a[:100000])
    Generating present_or_copyout(b[:100000])
    Generating Tesla code
36. Loop is parallelizable
    Accelerator kernel generated
     36. #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
37. Loop is parallelizable
    Accelerator kernel generated
     37. #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```

Директива PARALLEL

```
#include <openacc.h>
// . . .
float a[N], b[N]
// . . .
40.  #pragma acc parallel {
41.      for(int i=0; i<N; i++) a[i] = sqrtf(i);
42.  }
43.  #pragma acc parallel {
44.      for(int j=0; j<N; j++) b[j] = tanf(a[i]);
45.  }
```

Директива PARALLEL

```
40. Accelerator kernel generated
      41. #pragma acc loop vector(256) /* threadIdx.x */
41. Generating present_or_copyout(a[:100000])
    Generating Tesla code
42. Loop is parallelizable
43. Accelerator kernel generated
      44. #pragma acc loop vector(256) /* threadIdx.x */
45. Generating present_or_copyout(b[:100000])
    Generating present_or_copyin(a[:100000])
46. Generating Tesla code
    Loop is parallelizable
```

Директива DATA

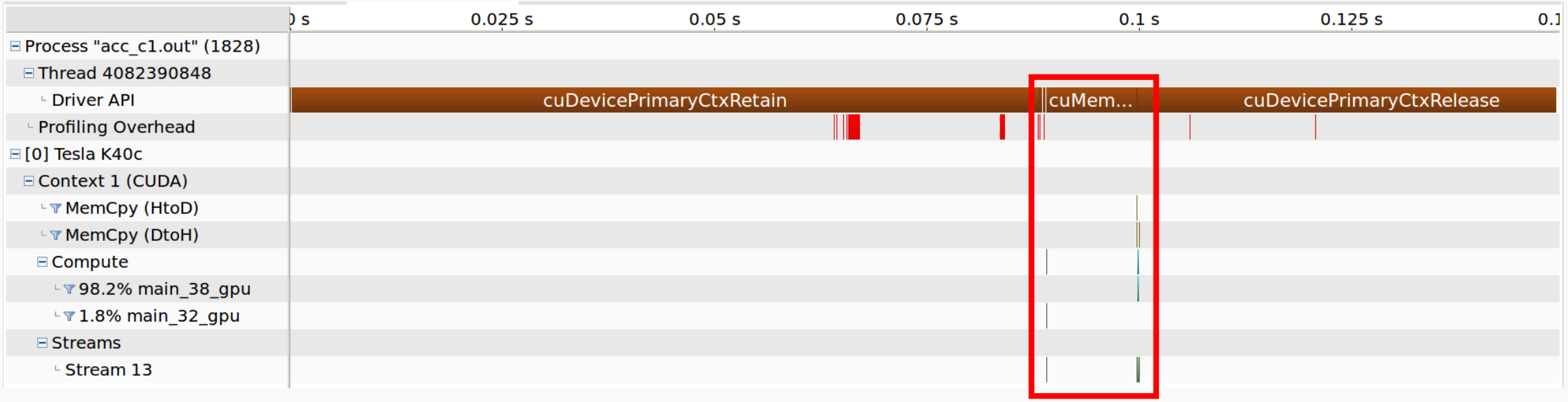
```
#include <openacc.h>
// . . .
float a[N], b[N]
// . . .
40. #pragma acc data copyout (a[0:N], b[0:N])
41. {
42.     #pragma acc parallel {
43.         for(int i=0; i<N; i++) a[i] = sqrtf(i); }
44.     #pragma acc parallel {
45.         for(int j=0; j<N; j++) b[j] = tanf(a[i]); }
46. }
```

Директива DATA

```
40. Generating present_or_copyout(a[:100000])
    Generating present_or_copyout(b[:100000])
41. Accelerator kernel generated
      42. #pragma acc loop vector(256) /* threadIdx.x */
42. Generating Tesla code
43. Loop is parallelizable
44. Accelerator kernel generated
      45. #pragma acc loop vector(256) /* threadIdx.x */
45. Generating Tesla code
46. Loop is parallelizable
```

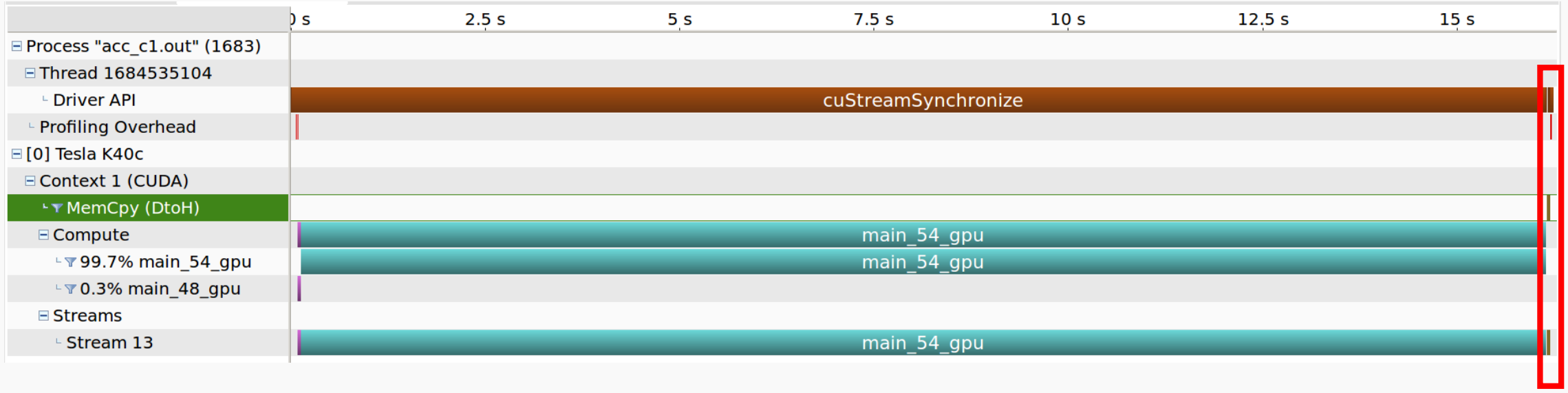
Visual Profiler

Копирование данных



Visual Profiler

Выполнение KERNEL-FUNCTION



Сравнение

- CUDA
- Библиотеки CUDA
- OpenACC
- OpenMP

Сравнение

Перемножение матриц (Лекция 2, Пример №4)

$$C = AB,$$

$$C_{i,j} = \sum_{k=0}^{N-1} A_{i,k} B_{k,j},$$

$$A_i = \sqrt{i},\ B_i = \sin(i),$$

$$i, j = 0,1, \dots, N-1$$

Сравнение

OpenMP

```
1.  float sum;

2.  #pragma omp parallel for shared (a, bT, c) private (n,m,k) {
3.      for(n=0; n<N; n++) {
4.          for(m=0; m<N; m++) {
5.              sum=0.f;
6.              for(k=0; k<N; k++) {
7.                  sum += a[k + n*N] * bT[k + m*N];
8.              }
9.              c[m + n*N] = sum;
10.         }
11.     }
12. }
```

Сравнение

OpenACC

```
1.  void mult(float *restrict c, float *a, float *bT, int N) {
2.      float sum;
3.      #pragma acc parallel loop present (c, a, bT) {
4.          for(n=0; n<N; n++) {
5.              for(m=0; m<N; m++) {
6.                  sum=0.f;
7.                  for(k=0; k<N; k++) { sum += a[k + n*N] * bT[k + m*N]; }
8.                  c[m + n*N] = sum;
9.              }
10.         }
11.     }
12. }
```

```
1.  int main() {
2.      // . . .
3.      #pragma acc data copyin (a[0:N], b[0:N]) copyout (c[0:N]) {
4.          mult(c, a, bT, N); }
5.      // . . .
6.  }
```

Сравнение

Анализ результатов

CPU Core i7-4770K 4.00GHz GPU GeForce GTX 780Ti

3700.00 ms	CPU-time
2963.00 ms	CPU-time OpenMP
176.428 ms	GPU-time OpenACC
18.87 ms	GPU-time CUDA
4.56 ms	GPU-time cuBLAS

OpenCV

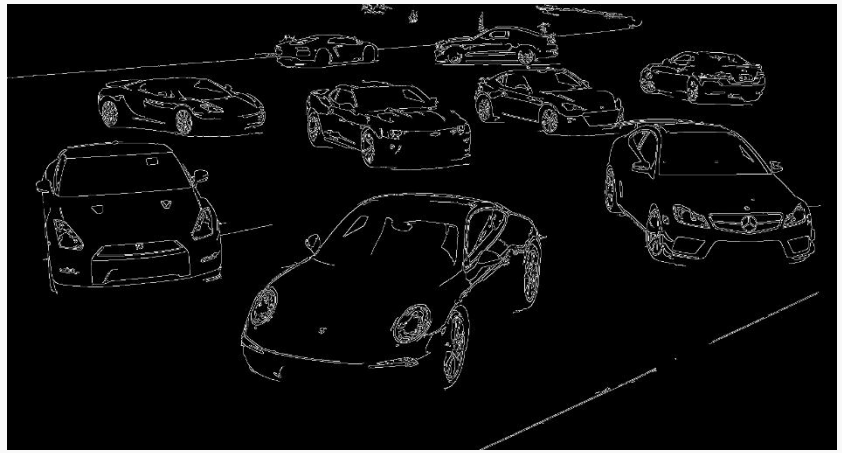
Применение:

- Обработка изображений и видео
- Обнаружение объектов
- Стерео-зрение
- Машинное обучение и кластеризация
- …

Обработка изображения CPU

```
1.  #include <opencv2/core/core.hpp>

2.  using namespace cv;

3.  int main() {
4.      Mat src = imread("./cars.jpg",0);
5.      if(!src.data) exit(-1);

6.      Mat dst;
7.      bilateralFilter(src,dst,-1, 50, 7);
8.      Canny(dst,dst, 35, 200, 3);

9.      imwrite("outCars.jpg",dst);

10.     return 0;
11. }
```

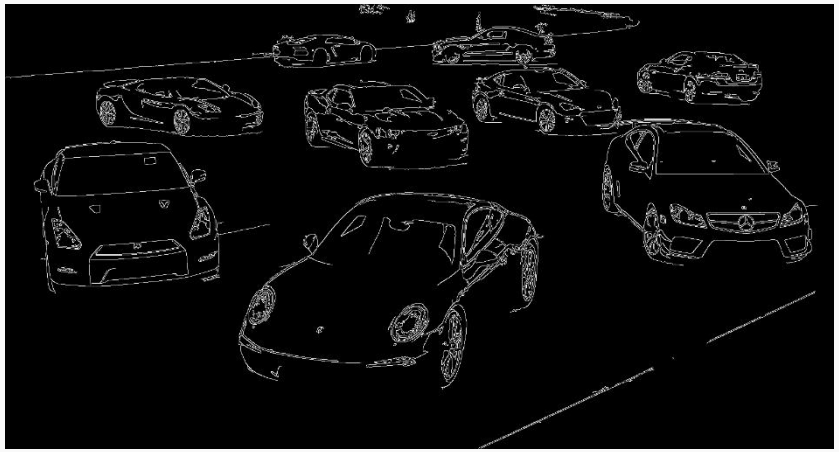
Результат



Обработка изображения GPU

```
1.  #include <opencv2/core/core.hpp>
2.  #include <opencv2/gpu/gpu.hpp>
3.  using namespace cv;
4.  int main() {
5.      if(getCudaEnabledDeviceCount() < 1) exit(-1);

6.      Mat src = imread("./cars.jpg",0);
7.      if(!src.data) exit(-1);

8.      gpu::GpuMat device_src(src);
9.      gpu::GpuMat device_dst;

10.     gpu::bilateralFilter(device_src, device_dst, -1, 50, 7);
11.     gpu::Canny(device_dst, device_dst, 35, 200, 3);

12.     Mat dst;
13.     device_dst.download(dst);

14.     imwrite("outCars.jpg",dst);
15.     return 0;
16. }
```

Результат



OpenCV

Сравнение

```
1.  #include <opencv2/core/core.hpp>

2.  using namespace cv;

3.  int main() {
4.      Mat src = imread("./cars.jpg",0);
5.      if(!src.data) exit(-1);

6.      Mat dst;
7.      bilateralFilter(src,dst,-1, 50, 7);
8.      Canny(dst,dst, 35, 200, 3);

9.      imwrite("outCars.jpg",dst);

10.     return 0;
11. }
```

1771.910 ms

12.002 ms

```
1.  #include <opencv2/core/core.hpp>
2.  #include <opencv2/gpu/gpu.hpp>
3.  using namespace cv;
4.  int main() {
5.      if(getCudaEnabledDeviceCount() < 1) exit(-1);

6.      Mat src = imread("./cars.jpg",
7.      if(!src.data) exit(-1);

8.      gpu::GpuMat device_src(src);
9.      gpu::GpuMat device_dst;

10.     gpu::bilateralFilter(device_src, device_dst, -1, 50, 7);
11.     gpu::Canny(device_dst, device_dst, 35, 200, 3);

12.     Mat dst;
13.     device_dst.download(dst);

14.     imwrite("outCars.jpg",dst);
15.     return 0;
16. }
```

Rate:
12.498 x

141.770 ms

10.699 ms

Контакты:
A.SPASENOV@CORP.MAIL.RU
ALEX_SPASENOV (SKYPE)

Спасибо за внимание!