Applied Parallel Computing LLC
http://parallel-computing.pro

Обзор библиотек для вычислений на GPU

к.т.н. Алексей Ивахненко

26 февраля, 2017

CUFFT Быстрое Преобразование Фурье

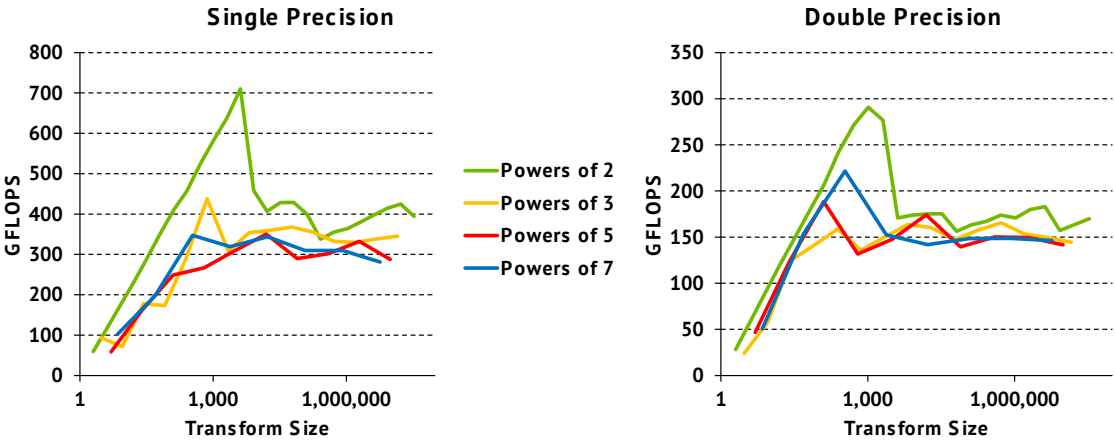
- 1D, 2D, 3D преобразование комплесных и действительных типов данных
- 1D преобразование поддерживает до 128 млн. элементов
- Гибкое расположение данных, благодаря изменяемым сдвигам между элементами и измерениям массивов
- FFT алгоритмы на базе Cooley-Tukey и Bluestein
- Потоковое асинхронное исполнение
- Преобразования одинарной и двойной точности
- Пакетное исполнение для множества одновременных преобразований
- Преобразования in-place и out-of-place
- Потокобезопасная, может быть вызвана из множества хостовых потоков

Дизайн очень схож с FFTW:

```
fftw_plan PlanA;
fftw_plan_dft_2d(N, M, &PlanA, data,
    data, FFT_FORWARD);
fftw_execute_dft(PlanA, data, data);
```

```
cufftPlan2d PlanA;
cufftCreatePlan(N, M, &PlanA, CUFFT_C2C);
cufftExecC2C(PlanA, d_data,
    d_data, CUFFT_FORWARD);
```

CUFFT 1D Complex Batched FFT Performance



- cuFFT 7.0 on K40m, Base clocks, ECC ON
- Batched transforms on 28M-33M total elements, input and output data on device
- Excludes time to create cuFFT plans

CURAND Генерация случайных чисел

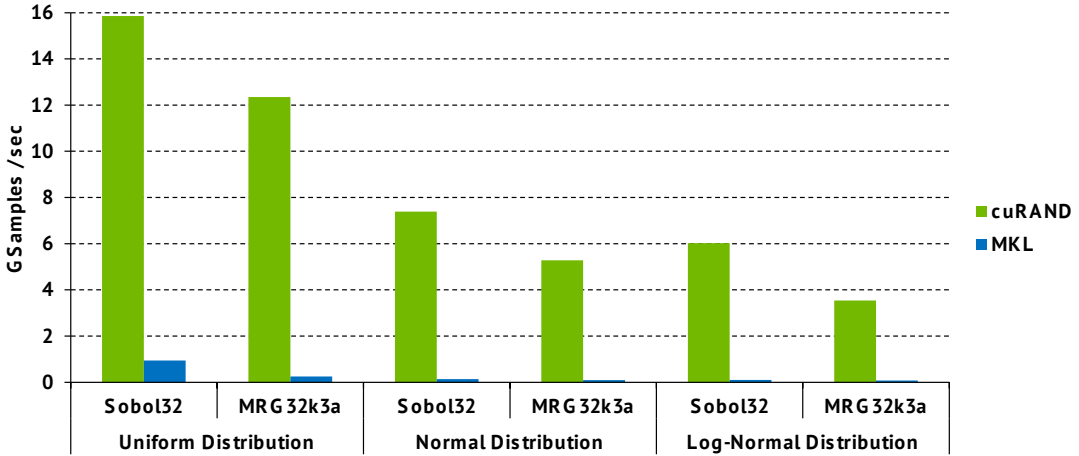
- Большой набор генераторов высококачественных последовательностей:
  - XORWOW, MRG323ka, MTGP32, scrambled Sobol
  - uniform, normal, log-normal
  - одинарная и двойная точность
- Два APIs:
  - Хост API: для генерации больших объемов чисел

```
#include "curand.h"
curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_DEFAULT);
curandGenerateUniform(gen, d_data, n);
```

  - CUDA kernel API: генерация чисел внутри ядра

```
#include “curand_kernel.”h
__global__ void generate_kernel(curandState *state)
{
    int id = threadIdx.x + blockIdx.x * blockDim.x;
    x = curand(&state[id]);
}
```

cuRAND Speedup vs. Intel MKL

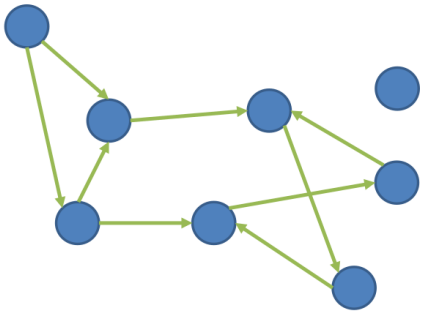


- cuRAND 7.0 on K40m, Base clocks, ECC ON, double-precision input and output data on device
- MKL 11.0.1 on Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo

nvGRAPH: Назнаение

- Эффективная резализация GPU-ускоренной библиотеки анализа графов для кибераналитики, работы с геномами, снализа социальных сетей и т.д.
- Все больше компаний используют данные о взаимодействии пользователя с интернетом и сенсорами $\Rightarrow$ требуется анализ в реальном времени коммерческих предпочтений.

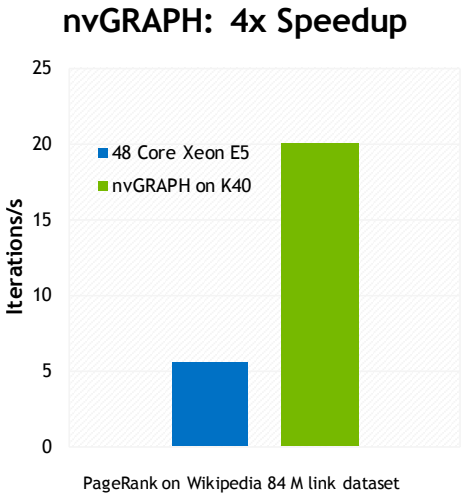
https://developer.nvidia.com/nvgraph



nvGRAPH: Производительность

- Обрабатывает графы с размером до 2.5 млрд. граней на GPU (24GB M40)
- Ускоряет множество различных приложений:

PageRank	Single Source Shortest Path	Single Source Widest Path
Search	Robotic Path Planning	IP Routing
Recommendation Engines	Power Network Planning	Chip Design
Social Ad Placement	Logistics & Supply Chain Planning	Traffic sensitive routing



nvGRAPH API

```
struct nvgraphCOOTopology32I_st
{
    int nvertices;
    int nedges;
    int *source_indices;
    int *destination_indices;
    nvgraphTag_t tag;
};

typedef struct ↩
    nvgraphCOOTopology32I_st *
nvgraphCOOTopology32I_t;

typedef enum
{
    NVGRAPH_CSR_32 = 0,
    NVGRAPH_CSC_32 = 1,
    NVGRAPH_COO_32 = 2
} nvgraphTopologyType_t;
```

```
// NVGRAPH_STATUS_SUCCESS
// NVGRAPH_STATUS_NOT_INITIALIZED
// NVGRAPH_STATUS_ALLOC_FAILED
// NVGRAPH_STATUS_INVALID_VALUE
// NVGRAPH_STATUS_ARCH_MISMATCH
// NVGRAPH_STATUS_MAPPING_ERROR
// NVGRAPH_STATUS_EXECUTION_FAILED
// NVGRAPH_STATUS_INTERNAL_ERROR
// NVGRAPH_STATUS_TYPE_NOT_SUPPORTED
// NVGRAPH_STATUS_NOT_CONVERGED
```

nvGRAPH API: nvgraphCreate

```
nvgraphStatus_t nvgraphCreate(nvgraphHandle_t *handle);

// NVGRAPH_STATUS_SUCCESS
// NVGRAPH_STATUS_ALLOC_FAILED
// NVGRAPH_STATUS_INTERNAL_ERROR
```

- Инициализирует библиотеку nvGRAPH, создает дескриптор контекста библиотеки nvGRAPH
- Необходимо вызвать до любых других вызовов nvGRAPH

nvGRAPH API: nvgraphDestroy

```
nvgraphStatus_t nvgraphDestroy(nvgraphHandle_t handle);

// NVGRAPH_STATUS_SUCCESS
// NVGRAPH_STATUS_INVALID_VALUE
```

- Освобождает использованные ресурсы
- Обычно эта функция является завершающей при работе с данным дескриптором библиотеки nvGRAPH

nvGRAPH API: nvgraphCreateGraphDescr

```
nvgraphStatus_t nvgraphCreateGraphDescr(
    nvgraphHandle_t handle, nvgraphGraphDescr_t *descrG);

// NVGRAPH_STATUS_SUCCESS
// NVGRAPH_STATUS_INVALID_VALUE
// NVGRAPH_STATUS_ALLOC_FAILED
```

- Создает дескриптор структуры графа
- Требуется для любой операции с графом

nvGRAPH API: nvgraphDestroyGraphDescr

```
nvgraphStatus_t nvgraphDestroyGraphDescr(
    nvgraphHandle_t handle, nvgraphGraphDescr_t descrG);

// NVGRAPH_STATUS_SUCCESS
// NVGRAPH_STATUS_TYPE_NOT_SUPPORTED
// NVGRAPH_STATUS_INVALID_VALUE
```

- Освобождает дескриптор графа, созданный с помощью nvgraphCreateGraphDescr()
- Не освобождает память, выделенную для графа до момента деинициализации дескриптора библиотеки nvGRAPH
- Вызовы, оперирующие уже деинициализированными графами возвразают NVGRAPH_STATUS_INVALID_VALUE

nvGRAPH API: остальные подпрограммы

- nvgraphSetGraphStructure()
- nvgraphGetGraphStructure()
- nvgraphConvertTopology()
- nvgraphConvertGraph()
- nvgraphAllocateEdgeData()
- nvgraphSetEdgeData()
- nvgraphGetEdgeData()
- nvgraphAllocateVertexData()
- nvgraphSetVertexData()
- nvgraphGetVertexData()
- nvgraphStatusGetString()

nvGRAPH: Порядок исполнения

- Инициализировать библиотеку с помощью nvgraphCreate()
- Загрузить граф на GPU
  - CSR (compressed sparse row) формат или
  - CSC(compressed column storage) или
  - nvgraphCreateGraphDescr()
- Прикрепить данные к вершинам, или граням графа с помощью nvgraphSetVertexData() и nvgraphSetEdgeData()
- Вызвать алгоритмы обработки графа
- Скопировать результаты на хост
- Освободить ресурсы использованныеnvGRAPH с помощью nvgraphDestroy()

nvGRAPH API: Алгоритмы

- ***nvgraphExtractSubgraphByVertex()***
  - Создает граф, извлекая подграф по заданным вершинам, состоит из номеров строк в массиве, представляющем изначальный граф
- ***nvgraphExtractSubgraphByEdge()***
  - Создает граф, извлекая подграф по заданным граням, состоит из номеров столбцовcol_ind[] в массиве, представляющем изначальный граф (CSR)
- ***nvgraphWidestPath()***
  - Нахождение широчайшего пути от вершины source_index ко всем остальным вершинам; Также известна как задача 'the bottleneck path problem' или 'the maximum capacity path problem'.
- ***nvgraphSssp()***
  - Single Source Shortest Path (SSSP) алгоритм рассчитывает кратчайшее расстояние от заданной вершины ко всем остальным

nvGRAPH SSSP Пример

```
#include <nvgraph.h>
#include <stdio.h>

int main(int argc, char **argv) {

const size_t n = 6, nnz = 10,
    vertex_numsets = 1, edge_numsets = 1;
float *sssp_1_h;
void** vertex_dim;
// nvgraph variables
nvgraphHandle_t handle;
nvgraphGraphDescr_t graph;
nvgraphCSCTopology32I_t CSC_input;
cudaDataType_t edge_dimT = CUDA_R_32F;
cudaDataType_t* vertex_dimT;
// Init host data
sssp_1_h = (float*)malloc(n*sizeof(float));
vertex_dim  = (void**)malloc(
    vertex_numsets * sizeof(void*));
vertex_dimT = (cudaDataType_t*)malloc(
    vertex_numsets * sizeof(cudaDataType_t));
CSC_input = (nvgraphCSCTopology32I_t) malloc(
    sizeof(struct nvgraphCSCTopology32I_st));
vertex_dim[0]= (void*)sssp_1_h;
vertex_dimT[0] = CUDA_R_32F;
float weights_h[] = {0.333333, 0.5, 0.333333, 0.5, 0.5,
    1.0, 0.333333, 0.5, 0.5, 0.5};
int destination_offsets_h[] = {0, 1, 3, 4, 6, 8, 10};
```

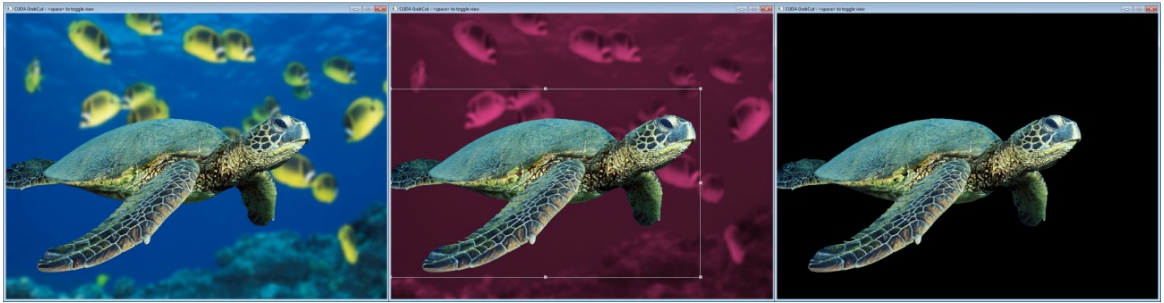
```
int source_indices_h[] = {2, 0, 2, 0, 4, 5, 2, 3, 3, 4};
nvgraphCreate(&handle);
nvgraphCreateGraphDescr (handle, &graph);
CSC_input->nvertices = n; CSC_input->nedges = nnz;
CSC_input->destination_offsets = destination_offsets_h;
CSC_input->source_indices = source_indices_h;
// Set graph connectivity and properties (tranfers)
nvgraphSetGraphStructure(handle, graph,
    (void*)CSC_input, NVGRAPH_CSC_32);
nvgraphAllocateVertexData(handle, graph,
    vertex_numsets, vertex_dimT);
nvgraphAllocateEdgeData  (handle, graph,
    edge_numsets, &edge_dimT);
nvgraphSetEdgeData(handle, graph, (void*)weights_h, 0);
// Solve
int source_vert = 0;
nvgraphSssp(handle, graph, 0,  &source_vert, 0);
// Get and print result
nvgraphGetVertexData(handle, graph, (void*)sssp_1_h, 0);
for (int i=0; i<n; ++i)
printf("%f\n",sssp_1_h[i]);
// Cleanup
free(sssp_1_h); free(vertex_dim);
free(vertex_dimT); free(CSC_input);
nvgraphDestroyGraphDescr(handle, graph);
nvgraphDestroy(handle);
return 0;
}
```

nvGRAPH SSSP Пример

```
$ ./nvgraph
0.000000
0.500000
0.500000
1.333333
0.833333
1.333333
```

NPP: NVIDIA Performance Primitives

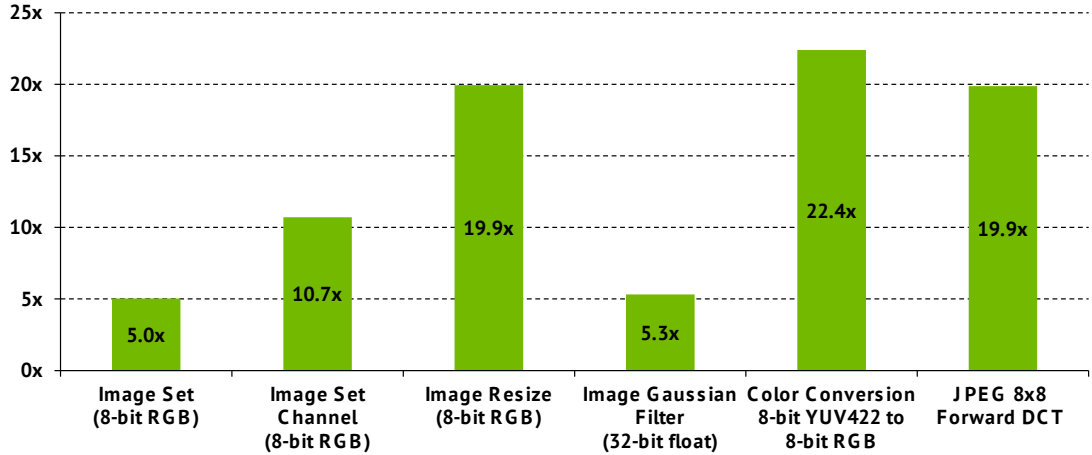
NVIDIA NPP – библиотека функций для обработки данных на CUDA



- Более 5000 подпрограмм обработки изображений и сигналов:

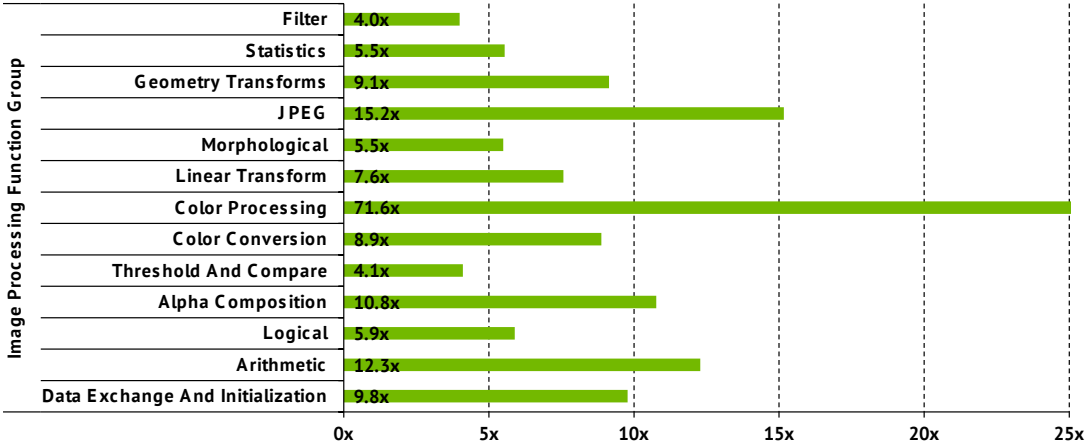
  преобразование цветов, геометрические преобразования, операции сдвига, линейные фильтры, статистика сигналов и изображений, JPEG, сегментация изображений, медианная фильтрация, BGR/YUV преобразования, 3D LUT конверсия цветов

NPP Speedup vs. Intel IPP



- NPP 7.0 on K40m, ECC ON, Base clocks, input and output data on device
- IPP 7.0 on Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo

NPP Speedup vs. Intel IPP



- NPP 7.0 on K40m, ECC ON, Base clocks, input and output data on device
- Each bar represents the average speedup over all routines in the function group
- IPP 7.0 on Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo

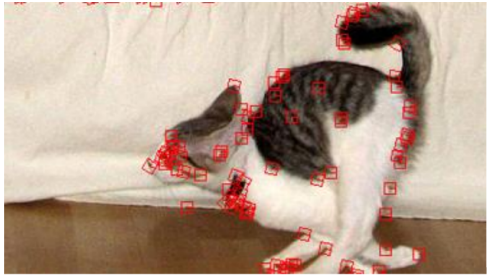
NPP

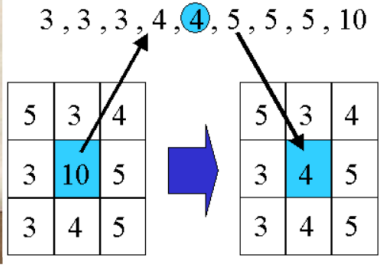
- Основной функционал библиотеки сосредоточен на обработке видео и изображений
- NPP со временем развивается и включает в себя новые более сложные задачи
- NPP написана для увеличения гибкости, при этом обеспечивает высокую производительность. Может быть использована одним из двух способов:
  - Отдельная библиотека для ускорения приложения на GPU с минимальными трудозатратами
  - Кооперативная библиотека для эффективного взаимодействия с другим GPU кодом

NPP Структура

- Основная библиотека NPPC содержит базовый функционал из файла <npp.h>, а так же функционал разделенный между двумя библиотеками.
- Библиотека обработки изображений NPPI:
  - Любые функции из <nppi.h> (или множества файлов с названием <nppi_xxx.h>)
- Библиотека обработки сигналов NPPS:
  - Любые функции из <npps.h> (или множества файлов с названием <npps_xxx.h>)
- Соответствующие статические и динамические библиотеки находятся в cuda/lib/libnpp* and cuda/lib64/libnpp* для 32- и 64-bit платформ

NPP: Фильтры изображений





NPP: Фильтр Гаусса

NPP Пример: Box Filter

```
// declare a host image object for an 8-bit grayscale image
npp::ImageCPU_8u_C1 oHostSrc;
// load gray-scale image from disk
npp::loadImage(sFilename, oHostSrc);
// declare a device image and copy construct from the host image, i.e. upload host to device
npp::ImageNPP_8u_C1 oDeviceSrc(oHostSrc);

// create struct with box-filter mask size
NppiSize oMaskSize = {5, 5};
// create struct with ROI size given the current mask
NppiSize oSizeROI = {oDeviceSrc.width() - oMaskSize.width + 1, oDeviceSrc.height() - oMaskSize.height + 1};
// allocate device image of appropriatedly reduced size
npp::ImageNPP_8u_C1 oDeviceDst(oSizeROI.width, oSizeROI.height);
// set anchor point inside the mask to (0, 0)
NppiPoint oAnchor = {0, 0};

// run box filter
NppStatus eStatusNPP;
eStatusNPP = nppiFilterBox_8u_C1R(oDeviceSrc.data(), oDeviceSrc.pitch(),
    oDeviceDst.data(), oDeviceDst.pitch(), oSizeROI, oMaskSize, oAnchor);
NPP_ASSERT(NPP_NO_ERROR == eStatusNPP);

// declare a host image for the result
npp::ImageCPU_8u_C1 oHostDst(oDeviceDst.size());
// and copy the device result data into it
oDeviceDst.copyTo(oHostDst.data(), oHostDst.pitch());
```

OpenCV

- OpenCV для CPU:

```
#include "opencv2/opencv.hpp"
```

- OpenCV для GPU:

```
#include "opencv2/gpu/gpu.hpp"
```

- Mat - класс для хранения данных на хосте
- gpu::GpuMat - класс для хранения данных на GPU. Идентичен хостовому Mat за исключением :
  - Не поддерживаются различные размерности (только 2D)
  - Нет функций, возвращающих ссылки на данные (ссылки в GPU не работают на CPU)
  - нет поддержки expression templates

OpenCV Пример

```
int main (int argc, char* argv[])
{
    try
    {
        cv::Mat src_host = cv::imread("file.png",
            CV_LOAD_IMAGE_GRAYSCALE);
        cv::gpu::GpuMat dst, src;
        src.upload(src_host);
        cv::gpu::threshold(src, dst, 128.0, 255.0,
            CV_THRESH_BINARY);
        cv::Mat result_host = dst;
        cv::imshow("Result", result_host);
        cv::waitKey();
    }
    catch(const cv::Exception& ex)
    {
        std::cout << "Error: " << ex.what() << std::endl;
    }
    return 0;
}
```