ПРИЛОЖЕНИЕ А

Исходный код библиотеки

FILE: AppConfig.hpp

PATH: AppConfig.hpp

EXTENSION: .hpp

SIZE: 5853 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Конфигурация приложения

class AppConfig

{

bool isInitialized{true};// Статус инициализации объекта конфигурации приложения

std::string message{"AppConfig status: OK"};// Строка с описанием статуса инициализации объекта конфигурации приложения

std::string fileConfig{"config.txt"};// Имя файла конфигурации

int compSystemId {1};// Идентификатор вычислительной системы

std::string dir\_data{"data"};// Каталог с данными

std::string dir\_computingSystemRepository{"ComputingSystemRepository"}; // Каталог с данными о вычислительных системах

std::string dir\_algTestingResultRepository{"AlgTestingResultRepository"}; // Каталог с данными о результатах вычислительных экспериментов

/// @brief Проверка существования каталогов

void CheckDirectories()

{

if(!FileSystemHelper::IsDirExists(dir\_data))

FileSystemHelper::CreateDir(dir\_data);

if(!FileSystemHelper::IsDirExists(GetDirComputingSystemRepository()))

FileSystemHelper::CreateDir(GetDirComputingSystemRepository());

if(!FileSystemHelper::IsDirExists(GetDirAlgTestingResultRepository()))

FileSystemHelper::CreateDir(GetDirAlgTestingResultRepository());

}

/// @brief Считывает конфигурацию из файла

/// @return true - успех; false - наличие ошибок считывания

bool ReadConfigFile()

{

std::ifstream f(fileConfig);

if(!f.is\_open())

{

message = "Config file \"" + fileConfig + "\" is not opened!";

return false;

}

// Проверка формата файла

std::string str;

f >> str;

if (str != "AppConfig")

{

message = "Config file \"" + fileConfig + "\" format is not AppConfig!";

return false;

}

// Считываем пары "Параметр Значение"

std::string param, value;

while(f >> param >> value)

{

//std::cout << param << ": " << value << std::endl;

if(param == "compSystemId")

{

try

{

compSystemId = std::stoi(value);

//std::cout << "!!! " << compSystemId << std::endl;

}

catch(const std::exception& e)

{

message = "Config file \"" + fileConfig + "\": compSystemId parameter is not recognized!";

return false;

}

}

else if (param == "dir\_data")

dir\_data = value;

else if (param == "dir\_algTestingResultRepository")

dir\_algTestingResultRepository = value;

else if (param == "dir\_computingSystemRepository")

dir\_computingSystemRepository = value;

else

{

message = "Config file \"" + fileConfig + "\": parameter \"" + param + "\" with value \"" + value + "\" is not recognized!";

return false;

}

}

return true;

}

public:

AppConfig()

{

// Проверка существования каталогов

CheckDirectories();

}

AppConfig(std::string fileName)

{

if(!FileSystemHelper::IsFileExists(fileName))

{

isInitialized = false;

message = "Error! Config file \"" + fileName + "\" not found!";

return;

}

fileConfig = fileName;

bool result = ReadConfigFile();

if(!result)

{

isInitialized = false;

return;

}

// Проверка существования каталогов

CheckDirectories();

}

bool IsInitialized() const

{

return isInitialized;

}

std::string GetMessage() const

{

return message;

}

// Возвращает путь к каталогу с репозиторием вычислительных систем

std::string GetDirComputingSystemRepository() const

{

std::string path = FileSystemHelper::CombinePath(dir\_data, dir\_computingSystemRepository);

return path;

}

// Возвращает путь к каталогу с репозиторием результатов тестовых запусков алгоритмов

std::string GetDirAlgTestingResultRepository() const

{

std::string path = FileSystemHelper::CombinePath(dir\_data, dir\_algTestingResultRepository);

return path;

}

void Print()

{

if(!isInitialized)

{

std::cout << "AppConfig: ["

<< "NOT INITIALIZED; "

<< message

<< "]" << std::endl;

return;

}

std::cout << "AppConfig: ["

<< "compSystemId: " << compSystemId << "; "

<< "dir\_data: " << dir\_data << "; "

<< "dir\_algTestingResultRepository: " << dir\_algTestingResultRepository << "; "

<< "dir\_computingSystemRepository: " << dir\_computingSystemRepository

<< "]" << std::endl;

}

};

==================================================

FILE: Application.hpp

PATH: Application.hpp

EXTENSION: .hpp

SIZE: 2887 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Приложение

class Application

{

// Главное меню

MainMenu menu;

// Конфигурация приложения

AppConfig appConfig;

// Репозиторий сведений о вычислительных системах

ComputingSystemRepository computingSystemRepository{false};

// Репозиторий алгоритмов

AlgorithmRepository algorithmRepository;

// Репозиторий реализаций алгоритмов

AlgorithmImplementationRepository algorithmImplementationRepository;

// Репозиторий сведений о тестовых запусках различных алгоритмов

AlgTestingResultRepository algTestingResultRepository{false};

public:

AppConfig& GetAppConfig()

{

return appConfig;

}

void Start()

{

// 1. Считываем конфигурацию из файла

std::string appConfigFileName {"config.txt"};

appConfig = AppConfig(appConfigFileName);

if(!appConfig.IsInitialized())

{

std::cerr << appConfig.GetMessage() << std::endl;

exit(-1);

}

std::cout << "Application initialization: OK" << std::endl;

// 2. Считываем сведения о вычислительной системе

computingSystemRepository = ComputingSystemRepository {appConfig.GetDirComputingSystemRepository()};

std::cout << "Computing system repository initialization: OK" << std::endl;

// 3. Инициализируем репозиторий алгоритмов

algorithmRepository = AlgorithmRepository{};

// 4. Инициализируем репозиторий реализаций алгоритмов

algorithmImplementationRepository = AlgorithmImplementationRepository{};

// 5. Считываем сведения о результатах тестовых запусков алгоритмов

algTestingResultRepository = AlgTestingResultRepository {appConfig.GetDirAlgTestingResultRepository()};

std::cout << "Computing system repository initialization: OK" << std::endl;

// 6. Запуск различных реализаций алгоритмов

AlgorithmImplementationExecutor algorithmImplementationExecutor{computingSystemRepository,

algorithmImplementationRepository, algTestingResultRepository};

// 7. Запускаем главное меню

menu.Start(appConfig,

computingSystemRepository,

algorithmRepository,

algorithmImplementationRepository,

algTestingResultRepository,

algorithmImplementationExecutor

);//\*/

}

};

==================================================

FILE: GlobalTestFunctions.hpp

PATH: GlobalTestFunctions.hpp

EXTENSION: .hpp

SIZE: 6139 bytes

----------------------------------------

CONTENT:

#pragma once

/////////////////////////////////

/// Проверка работы VectorGpu ///

bool TestVectorGpu()

{

// Добавить разные тесты

try

{

VectorGpu<double> v1{350000};

v1.InitByVal(0.001);

//v1.Print();

for(int i = 1; i <= 5; i++)

{

for(int j = 1; j <= 5; j++)

{

auto res = ArrayHelper::SumCuda(v1.Get\_dev\_data\_pointer(), v1.Length(),i,j);

std::cout << i << ", " << j << ": ";

//res.Print();

std::cout << res << std::endl;

}

}

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return false;

}

return true;

}

///////////////////////////////////////////////////////////

/// Тестирование функции суммирования элементов массива ///

bool TestSum()

{

TestParams testParams;

testParams.IterNum = 1;

// 1. Подготовка данных

unsigned Nthreads = 10;

size\_t size = 1000000000;

double elVal = 0.001;

VectorRam<double> v(size);

v.InitByVal(elVal);

//v.Print();

VectorGpu<double>\* vGpu\_p = nullptr;

try

{

vGpu\_p = new VectorGpu<double>(size);

vGpu\_p->InitByVal(elVal);

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

// 2. Запуск тестов и получение массива результатов

// 2.1 Последовательный алгоритм

auto testResults\_seq = TestHelper::LaunchSum(v, testParams);

std::cout << "Seq: testResults\_seq size = " << testResults\_seq.size() << std::endl;

for(auto& res : testResults\_seq)

res.Print();

// 2.2 Параллельный алгоритм std::thread

auto testResults\_par = TestHelper::LaunchSum(v, Nthreads, testParams);

std::cout << "Parallel: testResults size = " << testResults\_par.size() << std::endl;

for(auto& res : testResults\_par)

res.Print();

// 2.3 Параллельный алгоритм OpenMP

auto testResults\_par\_OpenMP = TestHelper::LaunchSumOpenMP(v, Nthreads, testParams);

std::cout << "Parallel OpenMP: testResults size = " << testResults\_par\_OpenMP.size() << std::endl;

for(auto& res : testResults\_par\_OpenMP)

res.Print();

// 2.4 Параллельный алгоритм Cuda

int numBlocks = 10;

auto testResults\_par\_Cuda = TestHelper::LaunchSumCuda(\*vGpu\_p, numBlocks, Nthreads, testParams);

std::cout << "Parallel CUDA: testResults size = " << testResults\_par\_Cuda.size() << std::endl;

for(auto& res : testResults\_par\_Cuda)

res.Print();

// 2.5 Параллельный алгоритм Cuda на 1 GPU с двумя видеочипами

//int numBlocks = 37;

/\*auto testResults\_par2\_Cuda = TestHelper::LaunchSumCudaMultiGpu(testParamsGpu);

std::cout << "Parallel CUDA LaunchSumCudaMultiGpu: testResults size = "

<< testResults\_par2\_Cuda.size() << std::endl;

for(auto& res : testResults\_par2\_Cuda)

res.Print();\*/

// Освобождаем видеопамять

vGpu\_p->Clear\_dev\_data();

// 3. Статистическая обработка результатов

CalculationStatistics stat\_seq{testResults\_seq};

std::cout << "CalculationStatistics seq: " << std::endl;

stat\_seq.Print();

CalculationStatistics stat\_par{testResults\_par};

std::cout << "CalculationStatistics parallel std::thread: " << std::endl;

stat\_par.Print();

CalculationStatistics stat\_par\_OpenMP;

try

{

stat\_par\_OpenMP = CalculationStatistics{testResults\_par\_OpenMP};

std::cout << "CalculationStatistics parallel OpenMP: " << std::endl;

stat\_par\_OpenMP.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

CalculationStatistics stat\_par\_Cuda;

try

{

stat\_par\_Cuda = CalculationStatistics{testResults\_par\_Cuda};

std::cout << "CalculationStatistics parallel Cuda: " << std::endl;

stat\_par\_Cuda.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

/\*CalculationStatistics stat\_par2\_Cuda;

try

{

stat\_par2\_Cuda = CalculationStatistics{testResults\_par2\_Cuda};

std::cout << "CalculationStatistics parallel Cuda LaunchSumCudaDevNum1GpuNum2: " << std::endl;

stat\_par2\_Cuda.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}\*/

// 4. Вычисляем ускорение и эффективность

std::cout << "--- std::thread ---" << std::endl;

ParallelCalcIndicators parallelCalcIndicators(stat\_seq, stat\_par, Nthreads);

parallelCalcIndicators.Print();

try

{

std::cout << "--- OpenMP ---" << std::endl;

ParallelCalcIndicators parallelCalcIndicators\_OpenMP(stat\_seq, stat\_par\_OpenMP, Nthreads);

parallelCalcIndicators\_OpenMP.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return false;

}

try

{

std::cout << "--- CUDA ---" << std::endl;

ParallelCalcIndicators parallelCalcIndicators\_Cuda(stat\_seq, stat\_par\_Cuda, numBlocks\*Nthreads);

parallelCalcIndicators\_Cuda.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return false;

}

/\*try

{

std::cout << "--- CUDA, 1 dev, 2 videochips ---" << std::endl;

ParallelCalcIndicators parallelCalcIndicators\_Cuda2(stat\_seq, stat\_par2\_Cuda, numBlocks\*Nthreads);

parallelCalcIndicators\_Cuda2.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return false;

}\*/

return true;

}

==================================================

FILE: TestHelper.hpp

PATH: TestHelper.hpp

EXTENSION: .hpp

SIZE: 3797 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

#include "CommonHelpers/FuncResult.hpp"

#include "Vectors/VectorRam.hpp"

#include "Vectors/VectorGpu.hpp"

#include "TestParams.hpp"

/// @brief Вспомогательный класс для запуска численных экспериментов

class TestHelper

{

public:

template<typename T>

static std::vector<FuncResult<T>> LaunchSum(VectorRam<T>& v, TestParams p)

{

std::cout << "-------LaunchSum(VectorRam<T>& v) Start ------" << std::endl;

auto iterNum = p.IterNum;

std::vector<FuncResult<T>> results;

for(unsigned i{0}; i < iterNum; i++)

{

FuncResult<T> res = VectorRamHelper::Sum(v);

results.push\_back(res);

}

std::cout << "-------LaunchSum(VectorRam<T>& v) End --------" << std::endl;

return results;

}

template<typename T>

static std::vector<FuncResult<T>> LaunchSum(VectorRam<T>& v, unsigned Nthreads, TestParams p)

{

std::cout << "-------LaunchSum(VectorRam<T>& v, unsigned Nthreads) Start ------" << std::endl;

auto iterNum = p.IterNum;

std::vector<FuncResult<T>> results;

for(unsigned i{0}; i < iterNum; i++)

{

FuncResult<T> res = VectorRamHelper::Sum(v, Nthreads);

results.push\_back(res);

}

std::cout << "-------LaunchSum(VectorRam<T>& v, unsigned Nthreads) End --------" << std::endl;

return results;

}

template<typename T>

static std::vector<FuncResult<T>> LaunchSumOpenMP(VectorRam<T>& v, unsigned Nthreads, TestParams p)

{

std::cout << "-------LaunchSumOpenMP(VectorRam<T>& v, unsigned Nthreads) Start ------" << std::endl;

std::vector<FuncResult<T>> results;

#ifdef \_OPENMP

for(unsigned i{0}; i < p.IterNum; i++)

{

FuncResult<T> res = VectorRamHelper::SumOpenMP(v, Nthreads);

results.push\_back(res);

}

#endif

std::cout << "-------LaunchSumOpenMP(VectorRam<T>& v, unsigned Nthreads) End --------" << std::endl;

return results;

}

template<typename T>

static std::vector<FuncResult<T>> LaunchSumCuda(VectorGpu<T>& v, unsigned NumBlocks, unsigned Nthreads, TestParams p)

{

std::cout << "-------LaunchSumCuda(VectorGpu<T>& v, unsigned NumBlocks, unsigned Nthreads, TestParams p) Start ------" << std::endl;

std::vector<FuncResult<T>> results;

#ifdef \_\_NVCC\_\_

auto iterNum = p.IterNum;

for(unsigned i{0}; i < iterNum; i++)

{

FuncResult<T> res = VectorGpuHelper::SumCuda(v, NumBlocks, Nthreads);

results.push\_back(res);

}

#endif

std::cout << "-------LaunchSumCuda(VectorGpu<T>& v, unsigned NumBlocks, unsigned Nthreads, TestParams p) End --------" << std::endl;

return results;

}

template<typename T>

static std::vector<FuncResult<T>> LaunchSumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params, TestParams p)

{

std::cout << "-------LaunchSumCudaSumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params) Start ------" << std::endl;

std::vector<FuncResult<T>> results;

#ifdef \_\_NVCC\_\_

auto iterNum = p.IterNum;

for(unsigned i{0}; i < iterNum; i++)

{

FuncResult<T> res = VectorGpuHelper::SumCudaMultiGpu(params);

results.push\_back(res);

}

#endif

std::cout << "-------LaunchSumCudaSumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params) End --------" << std::endl;

return results;

}

};

==================================================

FILE: TestParams.hpp

PATH: TestParams.hpp

EXTENSION: .hpp

SIZE: 243 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Параметры проведения численного эксперимента

struct TestParams

{

unsigned IterNum = 20;// Количество итераций

// --- Дополнить ---

};

==================================================

FILE: \_IncludeLib.hpp

PATH: \_IncludeLib.hpp

EXTENSION: .hpp

SIZE: 6146 bytes

----------------------------------------

CONTENT:

#include <iostream>

#include <fstream>

#include <thread>

#include <mutex>

#include <vector>

#include <chrono>

#include <functional>

#include <algorithm>

#include <cmath>

#include <fstream>

#include <string>

#include <map>

using namespace std::chrono;

#ifdef \_OPENMP

#include <omp.h>

#endif

#ifdef OPENBLAS

#include <cblas.h> // OpenBLAS

#endif

///////////// CUDA (начало) /////////////

#ifdef \_\_NVCC\_\_

#include "Cuda/kernels.cu"

#include <cublas\_v2.h>

#endif

#include "Cuda/CudaDeviceProperties.hpp"

#include "Cuda/CudaHelper.hpp"

#include "Cuda/CublasHelper.hpp"

///////////// CUDA (конец) /////////////

///// Вспомогательные типы

#include "CommonHelpers/\_IncludeCommonHelpers.hpp"

///// Дифференциальные уравнения

#include "DifferentialEquations/\_IncludeDifferentialEquations.hpp"

///// Геометрия расчетной области

#include "Geometry/\_IncludeGeometry.hpp"

///// Параметры проведения тестов производительности (начало) /////

#include "PerformanceTests/PerfTestParamsData.hpp"

#include "PerformanceTests/PerfTestParamsCpu.hpp"

#include "PerformanceTests/PerfTestParamsGpu.hpp"

#include "PerformanceTests/PerfTestParams.hpp"

#include "PerformanceTests/CalculationStatistics.hpp"

#include "PerformanceTests/ParallelCalcIndicators.hpp"

#include "PerformanceTests/PerfTestResults.hpp"

///// Параметры проведения тестов производительности (конец) /////

///// Модуль Math (начало) /////

#include "Math/MathObject.hpp"

#include "Math/Expression.hpp"

#include "Math/Constant.hpp"

#include "Math/Variable.hpp"

#include "Math/Negate.hpp"

#include "Math/BinaryExpression.hpp"

#include "Math/FuncExpression.hpp"

#include "Math/GridContext.hpp"

#include "Math/GridEvaluableObject.hpp"

#include "Math/GridOperator.hpp"

#include "Math/GridOperatorEvaluator.hpp"

#include "Math/MathHelper.hpp"

#include "Math/MathHelper\_ConsoleUI.hpp"

///// Модуль Math (конец) /////

////////// Функции (начало) ////////////

#include "Functions/FunctionDataType.hpp"

#include "Functions/FunctionDataTypes.hpp"

#include "Functions/FunctionArgument.hpp"

#include "Functions/FunctionArguments.hpp"

#include "Functions/Function.hpp"

////////// Функции (конец) ////////////

////////// Массивы (начало) ////////////

#include "Arrays/ArrayGpuProcessingParams.hpp"

#include "Arrays/ArrayHelper.hpp"

#include "Arrays/ArrayHelper\_ConsoleUI.hpp"

#include "Arrays/ArrayHelperFuncResult.hpp"

#include "Arrays/ArrayPerfTestHelper.hpp"

#include "Arrays/ArrayPerfTestHelper\_ConsoleUI.hpp"

#include "Arrays/ArrayBlockIndexes.hpp"

#include "Arrays/ArraysIndexMap.hpp"

#include "Arrays/DevMemArrPointer.hpp"

#include "Arrays/DevMemArrPointers.hpp"

////////// Массивы (конец) ////////////

////////// Модуль Scalars (Скалярные значения) ////////////

#include "Scalars/\_IncludeScalars.hpp"

////////// Модуль Vectors (Векторы) ////////////

#include "Vectors/\_IncludeVectors.hpp"

////////// Матрицы (начало) ////////////

#include "Matrices/MatrixDataLocation.hpp"

#include "Matrices/IMatrix.hpp"

#include "Matrices/MatrixRam.hpp"

#include "Matrices/MatrixRamZero.hpp"

#include "Matrices/MatrixRamE.hpp"

#include "Matrices/MatrixBlockRamGpus.hpp"

#include "Matrices/MatrixMapElement.hpp"

#include "Matrices/MatrixMap.hpp"

#include "Matrices/MatricesHelper.hpp"

#include "Matrices/MatricesHelper\_ConsoleUI.hpp"

////////// Матрицы (конец) ////////////

#include "TestParams.hpp"

#include "TestHelper.hpp"

//////// Вычислительная система (начало) ///////

#include "ComputingSystem/RamParams.hpp"

#include "ComputingSystem/CpuParams.hpp"

#include "ComputingSystem/GpuParams.hpp"

#include "ComputingSystem/ComputingSystemNode.hpp"

#include "ComputingSystem/ComputingSystem.hpp"

#include "ComputingSystem/ComputingSystemRepository.hpp"

//////// Вычислительная система (конец) ////////

//////// Задачи (начало) ///////

#include "Tasks/TaskGroup.hpp"

#include "Tasks/Task.hpp"

#include "Tasks/TaskDimensions.hpp"

//////// Задачи (конец) ///////

/////////////// Алгоритмы (начало) ///////////////////

#include "Algorithms/AlgorithmDataLocation.hpp"

#include "Algorithms/AlgorithmType.hpp"

#include "Algorithms/Algorithm.hpp"

#include "Algorithms/AlgorithmRepository.hpp"

#include "Algorithms/AlgorithmMetrics.hpp"

#include "Algorithms/AlgorithmImplementation.hpp"

#include "Algorithms/AlgorithmImplementationRepository.hpp"

#include "Algorithms/AlgorithmImplementationExecParams.hpp"

#include "Algorithms/AlgorithmImplementationExecutor.hpp"

#include "Algorithms/AlgorithmImplementationExecutorHelper.hpp"

/////////////// Алгоритмы (конец) ///////////////////

//// Результаты тестовых запусков алгоритмов (начало) /////

#include "AlgTestingResults/AlgTestingResult.hpp"

#include "AlgTestingResults/AlgTestingResultRepository.hpp"

///// Результаты тестовых запусков алгоритмов (конец) /////

////////// Глобальные функции (начало) ///////////

#include "GlobalTestFunctions.hpp"

////////// Глобальные функции (конец) ////////////

///// Решения модельных задач

#include "ModelProblems/\_IncludeModelProblems.hpp"

///////////// Приложение (начало) ////////////////

// Конфигурация приложения

#include "AppConfig.hpp"

// Меню (начало)

#include "Menu/MenuCommand.hpp"

#include "Menu/MenuCommandItem.hpp"

#include "Menu/MenuFunctions.hpp"

#include "Menu/MainMenu.hpp"

// Меню (конец)

// Класс "Приложение"

#include "Application.hpp"

///////////// Приложение (конец) ////////////////

==================================================

FILE: Algorithm.hpp

PATH: Algorithms\Algorithm.hpp

EXTENSION: .hpp

SIZE: 2071 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../Tasks/TaskGroup.hpp"

#include "../Tasks/Task.hpp"

#include "../Tasks/TaskDimensions.hpp"

#include "AlgorithmType.hpp"

#include "AlgorithmDataLocation.hpp"

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Сведения об алгоритме

struct Algorithm

{

// УИД алгоритма

unsigned id = 0;

// Группа задач

TaskGroup taskGroup;

// Задача

Task task;

// Размерности задачи

TaskDimensions taskDimensions {};

// Длина типа данных, используемая в алгоритме (float: 4; double: 8)

unsigned dataTypeLength = 0;

// Тип алгоритма (послед., параллельный и пр.)

AlgorithmType algorithmType;

// Место расположения исходных данных

AlgorithmDataLocation dataLocationInput;

// Место расположения результатов

AlgorithmDataLocation dataLocationOutput;

void Print(PrintParams pp)

{

std::cout << pp.startMes;

std::cout << "id" << pp.splitterKeyValue << id << pp.splitter;

std::cout << "taskGroup" << pp.splitterKeyValue << taskGroup << pp.splitter;

std::cout << "task" << pp.splitterKeyValue << task << pp.splitter;

std::cout << "taskDimensions" << pp.splitterKeyValue;

taskDimensions.Print(pp);

std::cout << pp.splitter;

std::cout << "dataTypeLength" << pp.splitterKeyValue << dataTypeLength << pp.splitter;

std::cout << "algorithmType" << pp.splitterKeyValue << algorithmType << pp.splitter;

std::cout << "dataLocationInput" << pp.splitterKeyValue << dataLocationInput << pp.splitter;

std::cout << "dataLocationOutput" << pp.splitterKeyValue << dataLocationOutput << pp.splitter;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: AlgorithmDataLocation.hpp

PATH: Algorithms\AlgorithmDataLocation.hpp

EXTENSION: .hpp

SIZE: 815 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Место расположения данных

enum class AlgorithmDataLocation

{

None, // 0 - Неинициализировано

Ram, // 1 - ОЗУ

Gpu, // 2 - видеопамять GPU

RamGpu // 3 - ОЗУ + видеопамять GPU

};

std::ostream& operator<<(std::ostream& os, AlgorithmDataLocation dataLocation)

{

switch (dataLocation)

{

case AlgorithmDataLocation::None:

os << "None";

break;

case AlgorithmDataLocation::Ram:

os << "Ram";

break;

case AlgorithmDataLocation::Gpu:

os << "Gpu";

break;

case AlgorithmDataLocation::RamGpu:

os << "RamGpu";

break;

default:

break;

}

return os;

}

==================================================

FILE: AlgorithmImplementation.hpp

PATH: Algorithms\AlgorithmImplementation.hpp

EXTENSION: .hpp

SIZE: 2288 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/PrintParams.hpp"

#include "../Functions/Function.hpp"

#include "AlgorithmImplementationExecParams.hpp"

#include "../AlgTestingResults/AlgTestingResult.hpp"

/// @brief Класс реализации алгоритма

/// (сопоставляет УИД алгоритма с функцией реализации)

class AlgorithmImplementation

{

unsigned id{};// УИД сопоставления

unsigned algorithmId{};// УИД алгоритма

// Объект функции, реализующей алгоритм

Function function{};

// Описание

std::string description{};

public:

AlgorithmImplementation()

{}

AlgorithmImplementation(unsigned id,

unsigned algorithmId,

std::string description,

Function function) :

id(id),

algorithmId(algorithmId),

function(function),

description(description)

{}

void Print(PrintParams pp)

{

std::cout << pp.startMes;

std::cout << "id" << pp.splitterKeyValue << id;

std::cout << pp.splitter;

std::cout << "algorithmId" << pp.splitterKeyValue << algorithmId;

std::cout << pp.splitter;

std::cout << "description" << pp.splitterKeyValue << description;

std::cout << pp.splitter;

std::cout << "function" << pp.splitterKeyValue;

function.Print(pp);

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

/// @brief Возвращает УИД сопоставления алгоритма и его реализации

/// @return УИД сопоставления (id)

unsigned GetId() const

{

return id;

}

/// @brief Возвращает объект функции

/// @return Объект типа Function

Function GetFunction()

{

return function;

}

AlgTestingResult Exec(AlgorithmImplementationExecParams params)

{

AlgTestingResult res = function.Exec(params);

std::cout << "\n\nAlgorithmImplementation::Exec str 69\n\n";

return res;

}

};

==================================================

FILE: AlgorithmImplementationExecParams.hpp

PATH: Algorithms\AlgorithmImplementationExecParams.hpp

EXTENSION: .hpp

SIZE: 387 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

/// @brief Параметры запуска реализации алгоритма

struct AlgorithmImplementationExecParams

{

// Аргументы функции, реализующей алгоритм

FunctionArguments functionArguments;

// Количество запусков функции

unsigned iterNumber{100};

};

==================================================

FILE: AlgorithmImplementationExecutor.hpp

PATH: Algorithms\AlgorithmImplementationExecutor.hpp

EXTENSION: .hpp

SIZE: 3840 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

//#include "../\_IncludeLib.hpp"

#include "../AlgTestingResults/AlgTestingResult.hpp"

#include "../AlgTestingResults/AlgTestingResultRepository.hpp"

/// @brief Класс для запуска алгоритма на заданной вычислительной системе

class AlgorithmImplementationExecutor

{

private:

// Ссылка на репозиторий вычислительных систем

ComputingSystemRepository& computingSystemRepository;

// УИД текущей вычислительной системы

unsigned computingSystemId{};

// Ссылка на репозиторий реализаций алгоритмов

AlgorithmImplementationRepository& algorithmImplementationRepository;

// Ссылка на репозиторий результатов тестовых запусков алгоритмов

AlgTestingResultRepository& algTestingResultRepository;

public:

AlgorithmImplementationExecutor(ComputingSystemRepository& computingSystemRepository,

AlgorithmImplementationRepository& algorithmImplementationRepository,

AlgTestingResultRepository& algTestingResultRepository)

: computingSystemRepository(computingSystemRepository),

algorithmImplementationRepository(algorithmImplementationRepository),

algTestingResultRepository(algTestingResultRepository)

{

}

/// @brief Проверяет готовность

/// @return

bool IsConfigured()

{

if(!(bool)computingSystemId)

return false;

return true;

}

/// @brief Устанавливает УИД текущей вычислительной системы

/// @param id УИД вычислительной системы

void SetComputingSystemId(unsigned id)

{

if (!computingSystemRepository.IsExists(id))

throw std::runtime\_error("ComputingSystem not found!");

computingSystemId = id;

}

/// @brief Устанавливает УИД текущей вычислительной системы

/// @param id УИД вычислительной системы

/// @return true - успех, false - неудача

bool TrySetComputingSystemId(unsigned id)

{

if (!computingSystemRepository.IsExists(id))

return false;

computingSystemId = id;

return true;

}

/// @brief Запускает реализацию алгоритма с заданными параметрами

/// на текущей вычислительной системе

/// @param AlgorithmImplementationId УИД реализации алгоритма

/// @return

AlgTestingResult Exec(unsigned AlgorithmImplementationId,

AlgorithmImplementationExecParams params)

{

AlgorithmImplementation alg = algorithmImplementationRepository.Get(AlgorithmImplementationId);

Function func = alg.GetFunction();

// Проверка соответствия тепов аргументов функции

if(!func.CheckArgumentsTypes(params.functionArguments.GetFunctionArgumentsDataTypes()))

{

std::cout << "\nError in AlgorithmImplementationExecutor::Exec(...)! Arguments types not equals!\n";

throw std::runtime\_error("Arguments types not equals!");

}

AlgTestingResult res = alg.Exec(params);

res.algorithmId = alg.GetId();

res.compSystemId = computingSystemId;

res.id = algTestingResultRepository.GetLastId() + 1;

res.Print();

algTestingResultRepository.Add(res);

return res;

}

};

==================================================

FILE: AlgorithmImplementationExecutorHelper.hpp

PATH: Algorithms\AlgorithmImplementationExecutorHelper.hpp

EXTENSION: .hpp

SIZE: 1385 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

struct AlgorithmImplementationExecutorHelper

{

static void Exec(AlgorithmImplementationExecutor& algorithmImplementationExecutor)

{

std::cout << "AlgorithmImplementationExecutorHelper::Exec()\n";

algorithmImplementationExecutor.SetComputingSystemId(1);

std::cout << "IsConfigured(): " << algorithmImplementationExecutor.IsConfigured();

std::cout << std::endl;

FunctionArgument arg\_void{};

FunctionArgument arg\_int{1};

FunctionArgument arg\_float{2.f};

FunctionArgument arg\_double{2000.123};

FunctionArgument arg\_ull{12345678909ull};

float\* arr\_float {new float[10]{1.1, 2.2, 3.3, 4, 5, 6, 7, 8, 9, 10}};

FunctionArgument arg\_ptr\_float{arr\_float};

double\* arr\_double = new double[10]{0.1, 0.2, 0.3, 0.4, 5, 6, 7, 8, 9, 10};

FunctionArgument arg\_ptr\_double{arr\_double};

FunctionArguments func\_args;

func\_args.Add(arg\_ptr\_float);

func\_args.Add(FunctionArgument{0ull});

func\_args.Add(FunctionArgument{9ull});

func\_args.Print(PrintParams{});

AlgorithmImplementationExecParams execParams;

execParams.functionArguments = func\_args;

AlgTestingResult res = algorithmImplementationExecutor.Exec(1, execParams);

res.Print();

}

};

==================================================

FILE: AlgorithmImplementationRepository.hpp

PATH: Algorithms\AlgorithmImplementationRepository.hpp

EXTENSION: .hpp

SIZE: 2969 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <map>

#include "AlgorithmImplementation.hpp"

#include "../CommonHelpers/PrintParams.hpp"

// Репозиторий реализаций алгоритмов.

// Сопоставляет УИД алгоритма с функцией, реализующей алгоритм

class AlgorithmImplementationRepository

{

std::map<unsigned, AlgorithmImplementation> data;

/// @brief Инициализация репозитория реализаций алгоритмов

void Init();

public:

AlgorithmImplementationRepository()

{

Init();

}

void Print(PrintParams pp)

{

std::cout << pp.startMes;

for(auto& el : data)

{

std::cout << el.first << pp.splitterKeyValue;

el.second.Print(PrintParams{}.SetIsEndl(false));

std::cout << pp.splitter;

if(pp.isEndl)

std::cout << std::endl;

}

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

/// @brief Проверяет наличие алгоритма с указанным УИД

/// @return

bool IsExists(unsigned id)

{

return data.count(id) > 0;

}

/// @brief Возвращает алгоритм по УИД

/// @param id

/// @return

AlgorithmImplementation Get(unsigned id)

{

return data[id];

}

/// @brief Запрашивает у пользователя id алгоритма и выводит в консоль сведения о нём

void Get()

{

unsigned id = ConsoleHelper::GetUnsignedIntFromUser("Enter algorithm implementation id: ");

AlgorithmImplementation algImpl = Get(id);

unsigned algImpl\_id = algImpl.GetId();

if(algImpl\_id == id && algImpl\_id != 0)

algImpl.Print(PrintParams{});

else

std::cout << "Algorithm implementation not found!" << std::endl;

}

/// @brief Добавляет реализацию алгоритма в репозиторий

/// @param algImpl

/// @return Результат (true - добавлен, false - не добавлен)

bool Add(AlgorithmImplementation algImpl)

{

unsigned algImpl\_id = algImpl.GetId();

if (algImpl\_id == 0 || IsExists(algImpl\_id))

return false;

data[algImpl\_id] = algImpl;

return true;

}

};

///////////////////////////////////////////////////////

void AlgorithmImplementationRepository::Init()

{

Function f1{ArrayHelper::Sum<float>};

AlgorithmImplementation algImpl\_01{1, 1, "ArrayHelper::Sum<float>", f1};

Add(algImpl\_01);

///////////////////////////////

Function f2{ArrayHelper::Sum<double>};

AlgorithmImplementation algImpl\_02{2, 2, "ArrayHelper::Sum<double>", f2};

Add(algImpl\_02);

}

==================================================

FILE: AlgorithmMetrics.hpp

PATH: Algorithms\AlgorithmMetrics.hpp

EXTENSION: .hpp

SIZE: 1457 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Метрики алгоритма

struct AlgorithmMetrics

{

// Объём дополнительной памяти ОЗУ

size\_t allocRam;

// Объём дополнительной памяти Gpu

size\_t allocGpu;

// количество считываний из памяти ОЗУ

size\_t readRam;

// количество считываний из глобальной памяти Gpu

size\_t readGpu;

// количество суммирований и вычитаний

size\_t arifmSumSub;

// количество умножений и делений

size\_t arifmMultDiv;

void Print(PrintParams pp)

{

std::cout << pp.startMes;

std::cout << "allocRam" << pp.splitterKeyValue << allocRam << pp.splitter;

std::cout << "allocGpu" << pp.splitterKeyValue << allocGpu << pp.splitter;

std::cout << "readRam" << pp.splitterKeyValue << readRam << pp.splitter;

std::cout << "readGpu" << pp.splitterKeyValue << readGpu << pp.splitter;

std::cout << "arifmSumSub" << pp.splitterKeyValue << arifmSumSub << pp.splitter;

std::cout << "arifmMultDiv" << pp.splitterKeyValue << arifmMultDiv << pp.splitter;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: AlgorithmRepository.hpp

PATH: Algorithms\AlgorithmRepository.hpp

EXTENSION: .hpp

SIZE: 2866 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <map>

#include "Algorithm.hpp"

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Репозиторий алгоритмов

class AlgorithmRepository

{

std::map<unsigned, Algorithm> data;

/// @brief Инициализация репозитория алгоритмов

void Init();

public:

AlgorithmRepository()

{

Init();

}

void Print(PrintParams pp)

{

std::cout << "void AlgorithmRepository::Print();\n";

for(auto& element : data)

{

element.second.Print(pp);

}

if (pp.isEndl)

std::cout << std::endl;

}

/// @brief Проверяет наличие алгоритма с указанным УИД

/// @return

bool IsExists(unsigned id)

{

return data.count(id) > 0;

}

/// @brief Возвращает алгоритм по УИД

/// @param id

/// @return

Algorithm Get(unsigned id)

{

return data[id];

}

/// @brief Запрашивает у пользователя id алгоритма и выводит в консоль сведения о нём

void Get()

{

unsigned id = ConsoleHelper::GetUnsignedIntFromUser("Enter algorithm id: ");

Algorithm alg = Get(id);

if(alg.id == id && alg.id != 0)

alg.Print(PrintParams{});

else

std::cout << "Algorithm not found!" << std::endl;

}

/// @brief Добавляет алгоритм в репозиторий

/// @param alg

/// @return Результат (true - добавлен, false - не добавлен)

bool Add(Algorithm alg)

{

if (alg.id == 0 || IsExists(alg.id))

return false;

data[alg.id] = alg;

return true;

}

};

///////////////////////////////////////////////////////

void AlgorithmRepository::Init()

{

Algorithm alg1;

alg1.id = 1;

alg1.taskGroup = TaskGroup::Array;

alg1.task = Task::Sum;

alg1.taskDimensions = TaskDimensions{1};

alg1.dataTypeLength = sizeof(float);

alg1.algorithmType = AlgorithmType::SeqCpu;

alg1.dataLocationInput = AlgorithmDataLocation::Ram;

alg1.dataLocationOutput = AlgorithmDataLocation::Ram;

Add(alg1);

Algorithm alg2;

alg2.id = 2;

alg2.taskGroup = TaskGroup::Array;

alg2.task = Task::Sum;

alg2.taskDimensions = TaskDimensions{1};

alg2.dataTypeLength = sizeof(double);

alg2.algorithmType = AlgorithmType::SeqCpu;

alg2.dataLocationInput = AlgorithmDataLocation::Ram;

alg2.dataLocationOutput = AlgorithmDataLocation::Ram;

Add(alg2);

}

==================================================

FILE: AlgorithmType.hpp

PATH: Algorithms\AlgorithmType.hpp

EXTENSION: .hpp

SIZE: 1125 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Тип алгоритма:

enum class AlgorithmType

{

None, // 0 - Неинициализировано

SeqCpu, // 1 - последовательный CPU

SeqGpuCuda, // 2 - последовательный GPU CUDA

ParCpuThread,// 3 - параллельный CPU std::thread

ParCpuOpenMP,// 4 - параллельный CPU OpenMP

ParGpuCuda // 5 - параллельный GPU CUDA

};

std::ostream& operator<<(std::ostream& os, AlgorithmType algType)

{

switch (algType)

{

case AlgorithmType::None:

os << "None";

break;

case AlgorithmType::SeqCpu:

os << "SeqCpu";

break;

case AlgorithmType::SeqGpuCuda:

os << "SeqGpuCuda";

break;

case AlgorithmType::ParCpuThread:

os << "ParCpuThread";

break;

case AlgorithmType::ParCpuOpenMP:

os << "ParCpuOpenMP";

break;

case AlgorithmType::ParGpuCuda:

os << "ParGpuCuda";

break;

default:

break;

}

return os;

}

==================================================

FILE: AlgTestingResult.hpp

PATH: AlgTestingResults\AlgTestingResult.hpp

EXTENSION: .hpp

SIZE: 2506 bytes

----------------------------------------

CONTENT:

#pragma once

#include <sstream>

#include "../PerformanceTests/CalculationStatistics.hpp"

/// @brief Результаты тестового запуска алгоритма

struct AlgTestingResult

{

// УИД тестового запуска

size\_t id = 0;

// УИД вычислительной системы

unsigned compSystemId = 0;

// УИД алгоритма

unsigned algorithmId = 0;

// Количество потоков CPU

unsigned threadsNumCpu = 0;

// Количество блоков GPU

unsigned threadBlocksNumGpu = 0;

// Количество нитей GPU в блоке

unsigned threadsNumGpu = 0;

// Статистики вычислительного эксперимента

CalculationStatistics calculationStatistics;

AlgTestingResult()

{

}

AlgTestingResult(std::string strToParse)

{

std::stringstream obj\_ss(strToParse);

obj\_ss >> id;

obj\_ss >> compSystemId;

obj\_ss >> algorithmId;

obj\_ss >> threadsNumCpu;

obj\_ss >> threadBlocksNumGpu;

obj\_ss >> threadsNumGpu;

obj\_ss >> calculationStatistics.numIter;

obj\_ss >> calculationStatistics.minValue;

obj\_ss >> calculationStatistics.median;

obj\_ss >> calculationStatistics.avg;

obj\_ss >> calculationStatistics.percentile\_95;

obj\_ss >> calculationStatistics.maxValue;

obj\_ss >> calculationStatistics.stdDev;

}

void Print()

{

std::cout << "id: " << id << "; ";

std::cout << "compSystemId: " << compSystemId << "; ";

std::cout << "algorithmId: " << algorithmId << "; ";

std::cout << "threadsNumCpu: " << threadsNumCpu << "; ";

std::cout << "threadBlocksNumGpu: " << threadBlocksNumGpu << "; ";

std::cout << "threadsNumGpu: " << threadsNumGpu << "; ";

calculationStatistics.Print();

std::cout << std::endl;

}

friend std::ofstream& operator<<(std::ofstream& fout, const AlgTestingResult& data)

{

fout << data.id << " "

<< data.compSystemId << " "

<< data.algorithmId << " "

<< data.threadsNumCpu << " "

<< data.threadBlocksNumGpu << " "

<< data.threadsNumGpu << " ";

fout << data.calculationStatistics;

fout << "\n";

return fout;

}

};

==================================================

FILE: AlgTestingResultRepository.hpp

PATH: AlgTestingResults\AlgTestingResultRepository.hpp

EXTENSION: .hpp

SIZE: 6414 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Репозиторий результатов тестовых запусков алгоритмов

class AlgTestingResultRepository

{

bool isInitialized = false;

std::string dir\_name = "AlgTestingResultRepository";// Каталог с данными

std::string file\_name = "data.txt"; // Файл с данными

std::vector<AlgTestingResult> cache; // Кэш данных в памяти

// Ключ - compSystemId;

// значение - вектор индексов УИД тестовых запусков

// вычислительной системы compSystemId

std::map<unsigned, std::vector<size\_t>> compSystemIndex;

/// @brief Проверка существования каталогов

void CheckDirectories()

{

if (!isInitialized) return;

if(!FileSystemHelper::IsDirExists(dir\_name))

FileSystemHelper::CreateDir(dir\_name);

std::string filePath = GetFullPath();

if(!FileSystemHelper::IsFileExists(filePath))

FileSystemHelper::CreateFile(dir\_name, file\_name, "");

}

public:

AlgTestingResultRepository(bool isInitialized = true)

: isInitialized(isInitialized)

{

CheckDirectories();

}

AlgTestingResultRepository(std::string dir\_name)

: dir\_name(dir\_name)

{

isInitialized = true;

CheckDirectories();

}

void PrintConfig()

{

std::cout << "isInitialized: " << isInitialized << "; "

<< "dir\_name: " << dir\_name << "; "

<< "file\_name: " << file\_name << std::endl;

}

/// @brief Считывает значение пути к каталогу с данными

/// @param dir

std::string Get\_dir\_name()

{

return dir\_name;

}

/// @brief Возвращает полный путь к файлу с данными

std::string GetFullPath()

{

return FileSystemHelper::CombinePath(dir\_name, file\_name);

}

/// @brief Возвращает наибольший использованный УИД тестового запуска

/// @return

size\_t GetLastId()

{

std::ifstream fin(GetFullPath());

if(!fin.is\_open())

throw std::runtime\_error("File not opened!");

size\_t id\_max = 0;

while(!fin.eof())

{

std::string line;

std::getline(fin,line);

//std::cout << line << std::endl;

if(line.size() < 2)

continue;

AlgTestingResult algTestingResult(line);

if(algTestingResult.id > id\_max)

id\_max = algTestingResult.id;

}

return id\_max;

}

/// @brief Устанавливает значение пути к каталогу с данными

/// @param dir

void Set\_dir\_name(std::string dir)

{

dir\_name = dir;

}

/// @brief Записывает результаты тестового запуска в файл

/// @param data

/// @return

bool Add(AlgTestingResult& data)

{

try

{

std::string filePath = GetFullPath();

std::ofstream fout(filePath, std::ios::app);

fout << data;

fout.close();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return false;

}

return true;

}

/// @brief Запуск по команде меню

void Add()

{

AlgTestingResult res;

res.id = GetLastId() + 1;

res.compSystemId = 222;

res.algorithmId = 333;

bool result = Add(res);

if(result)

std::cout << "Item with id=" + std::to\_string(res.id) + " added." << std::endl;

else

std::cout << "Error in adding item with id=" + std::to\_string(res.id) << std::endl;

}

AlgTestingResult Find(size\_t id)

{

std::ifstream fin(GetFullPath());

if(!fin.is\_open())

throw std::runtime\_error("File not opened!");

while(!fin.eof())

{

std::string line;

std::getline(fin,line);

//std::cout << line << std::endl;

if(line.size() < 2)

continue;

AlgTestingResult algTestingResult(line);

if(algTestingResult.id == id)

return algTestingResult;

}

throw std::runtime\_error("AlgTestingResult entry with id=" + std::to\_string(id) + " not found!");

}

/// @brief Поиск записи в файле по команде меню

void Find()

{

size\_t id = ConsoleHelper::GetUnsignedLongLongFromUser("Enter id: ");

//std::cout << "ull: " << id << std::endl;

try

{

AlgTestingResult entry = Find(id);

entry.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

/// @brief Проверка существования записи с указанным id

bool IsExists(size\_t id)

{

std::ifstream fin(GetFullPath());

if(!fin.is\_open())

throw std::runtime\_error("File not opened!");

while(!fin.eof())

{

std::string line;

std::getline(fin,line);

if(line.size() < 2)

continue;

std::stringstream obj\_ss(line);

size\_t cur\_id;

obj\_ss >> cur\_id;

if(cur\_id == id)

return true;

}

return false;

}

/// @brief Проверка существования записи с указанным id по еоманде меню

void IsExists()

{

size\_t id = ConsoleHelper::GetUnsignedLongLongFromUser("Enter id: ");

bool isExists = IsExists(id);

if(isExists)

std::cout << "Item with id=" + std::to\_string(id) + " exists." << std::endl;

else

std::cout << "Item with id=" + std::to\_string(id) + " not exists." << std::endl;

}

};

==================================================

FILE: ArrayBlockIndexes.hpp

PATH: Arrays\ArrayBlockIndexes.hpp

EXTENSION: .hpp

SIZE: 1094 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Индекс и размер блока, локальный индекс элемента в блоке

struct ArrayBlockIndexes

{

// Индекс блока

unsigned blockIndex{};

// Размер блока

unsigned long long blockLength{};

// Локальный индекс элемента в блоке

unsigned long long localIndex{};

void Print(PrintParams pp = PrintParams{})

{

pp.PrintStartMessage();

pp.PrintKeyValue("blockIndex", blockIndex);

pp.PrintSplitter();

pp.PrintKeyValue("blockLength", blockLength);

pp.PrintSplitter();

pp.PrintKeyValue("localIndex", localIndex);

pp.PrintEndMessage();

pp.PrintIsEndl();

}

/// @brief Возвращает флаг инициализации объекта

/// @return Успех, если размер блока > 0

bool IsInitialized()

{

return (bool)blockLength;

}

};

==================================================

FILE: ArrayGpuProcessingParams.hpp

PATH: Arrays\ArrayGpuProcessingParams.hpp

EXTENSION: .hpp

SIZE: 667 bytes

----------------------------------------

CONTENT:

#pragma once

// Параметры запуска функции обработки массива на GPU

template<typename T>

struct ArrayGpuProcessingParams

{

unsigned deviceId;

T\* dev\_arr;

size\_t indStart;

size\_t indEnd;

unsigned blocksNum;

unsigned threadsNum;

void Print()

{

std::cout << "[";

std::cout << deviceId << "; ";

std::cout << dev\_arr << "; ";

std::cout << indStart << "; ";

std::cout << indEnd << "; ";

std::cout << blocksNum << "; ";

std::cout << threadsNum << "; ";

std::cout << "]";

std::cout << std::endl;

}

};

==================================================

FILE: ArrayHelper.hpp

PATH: Arrays\ArrayHelper.hpp

EXTENSION: .hpp

SIZE: 49332 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <vector>

#include <thread>

#include "DevMemArrPointer.hpp"

/// @brief Структура для хранения методов обработки массивов T\*

struct ArrayHelper

{

////////////////////////// Вывод массивов в консоль (начало) /////////////////////////////

template<typename T>

static void PrintArrayRam(T\* data, size\_t indStart, size\_t length)

{

std::cout << "[";

for (size\_t i = indStart; i < indStart+length-1; i++)

{

std::cout << data[i] << " ";

}

std::cout << data[indStart+length-1];

std::cout << "]\n";

}

template<typename T>

static void PrintArrayRam(T\* data, size\_t length)

{

PrintArrayRam(data, 0, length);

}

///////// Вывод значений элементов массивов GPU в консоль

template<typename T>

static void PrintArrayGpu(T\* data, size\_t indStart, size\_t length, int deviceId = 0)

{

#ifdef \_\_NVCC\_\_

if(deviceId > 0)

{

std::thread th{[&](){

cudaSetDevice(deviceId);

kernel\_print<T><<<1,1>>>(data, indStart, length);

cudaDeviceSynchronize();

}};

th.join();

}

else

{

kernel\_print<T><<<1,1>>>(data, indStart, length);

cudaDeviceSynchronize();

}

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Вывод массивов в консоль (конец) /////////////////////////////

////////////////////////// Создание и освобождение массивов (начало) /////////////////////////////

/// @brief Выделяет память для массива в RAM

/// @tparam T Тип элементов массива

/// @param size Количество элементов

/// @return Указатель на созданный массив

template<typename T>

static T\* CreateArrayRam(unsigned long long size)

{

return new T[size];

}

template<typename T>

static void DeleteArrayRam(T\*& arrayRam)

{

if(arrayRam == nullptr)

return;

delete[] arrayRam;

arrayRam = nullptr;

}

// Работа с закреплённой памятью (начало)

/// @brief Выделяет закреплённую память для массива в RAM

/// @tparam T Тип элементов массива

/// @param size Количество элементов

/// @return Указатель на созданный массив

template<typename T>

static T\* CreateArrayRamPinned(unsigned long long size)

{

#ifdef \_\_NVCC\_\_

if(size == 0)

return nullptr;

T\* h\_aPinned = nullptr;

size\_t bytes = size \* sizeof(T);

cudaMallocHost((void\*\*)&h\_aPinned, bytes);

if(CudaHelper::IsErrors())

return nullptr;

return h\_aPinned;

#else

return nullptr;

#endif

}

template<typename T>

static void DeleteArrayRamPinned(T\*& arrayRamPinned)

{

#ifdef \_\_NVCC\_\_

cudaFreeHost(arrayRamPinned);

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

// Работа с закреплённой памятью (конец)

/// @brief Выделяет память для массива на текущем GPU

/// @tparam T Тип элементов массива

/// @param size Количество элементов

/// @return Указатель на созданный массив

template<typename T>

static T\* CreateArrayGpu(unsigned long long size)

{

#ifdef \_\_NVCC\_\_

if (size == 0)

{

std::string mes = "Cannot initialize array of 0 elements";

//std::cerr << mes << std::endl;

throw std::logic\_error(mes);

}

//std::cout << "Allocating GPU memory: " << size << " \* " << sizeof(T) << " = " << size\*sizeof(T) << " bytes... ";

T\* dev\_array = nullptr;

cudaMalloc(&dev\_array, size\*sizeof(T));

std::string msg("Could not allocate device memory for GPU array: ");

msg += std::to\_string(size\*sizeof(T));

msg += " bytes not allocated!\n";

cudaCheckErrors(msg.c\_str());

cudaDeviceSynchronize();

std::cout << "OK\n";

return dev\_array;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

/// @brief Выделяет память для массива на GPU

/// @tparam T Тип элементов массива

/// @param size Количество элементов

/// @param deviceId Идентификатор устройства

/// @return Указатель на созданный массив

template<typename T>

static T\* CreateArrayGpu(unsigned long long size, int deviceId)

{

#ifdef \_\_NVCC\_\_

if (size == 0)

{

std::string mes = "Cannot initialize array of 0 elements";

//std::cerr << mes << std::endl;

throw std::logic\_error(mes);

}

T\* dev\_array = nullptr;

if(deviceId == 0)

{

dev\_array = CreateArrayGpu<T>(size);

}

else

{

std::thread th{

[&](){

// Set CUDA device.

//cudaSetDevice(deviceId);

//cudaCheckErrors("!!!Cannot set CUDA device\n");

if(CudaHelper::SetDevice(deviceId))

dev\_array = CreateArrayGpu<T>(size);

}

};

th.join();

}

return dev\_array;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

/// @brief Освобождает массив на текущем GPU

/// @tparam T Указатель на массив в GPU

template<typename T>

static void DeleteArrayGpu(T\*& arrayGpu)

{

#ifdef \_\_NVCC\_\_

//std::cout << "Clearing gpu array " << arrayGpu << ": ";

if (arrayGpu == nullptr)

return;

cudaFree(arrayGpu);

cudaCheckErrors("Error in cudaFree!");

arrayGpu = nullptr;

//std::cout << "OK (" << arrayGpu << ")\n";

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

/// @brief Освобождает массив на GPU c

/// @tparam T Указатель на массив в GPU

/// @tparam deviceId Идентификатор GPU

template<typename T>

static void DeleteArrayGpu(T\*& arrayGpu, int deviceId)

{

#ifdef \_\_NVCC\_\_

if (deviceId == 0)

{

DeleteArrayGpu(arrayGpu);

return;

}

std::thread th{

[&](){

cudaSetDevice(deviceId);

cudaCheckErrors("Error in cudaSetDevice!");

DeleteArrayGpu(arrayGpu);

}

};

th.join();

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Создание и освобождение массивов (конец) /////////////////////////////

////////////////////////// Инициализация массивов (начало) /////////////////////////////

/// @brief Инициализирует массив array значениями value

/// @tparam T

/// @param array

/// @param size

/// @param value

template<typename T>

static void InitArrayRam(T\* array,

unsigned long long size,

T value)

{

for (unsigned long long i = 0ull; i < size; i++)

{

array[i] = value;

}

}

/// @brief Заполняет массив dev\_array на текущем GPU значением value

/// @tparam T Тип элементов массива

/// @param dev\_array Указатель на инициализируемый массив

/// @param size Количество элементов массива

/// @param value Значение, присваиваемое всем элементам массива dev\_array

template<typename T>

static void InitArrayGpu(T\* dev\_array,

unsigned long long size,

T value)

{

CudaHelper::InitByValue(dev\_array, size, value);

}

template<typename T>

static void InitArrayGpu(T\* dev\_array,

unsigned long long size,

T value,

int deviceId)

{

#ifdef \_\_NVCC\_\_

if(deviceId == 0)

{

InitArrayGpu(dev\_array, size, value);

}

else

{

std::thread th{

[&](){

// Set curent CUDA device.

cudaError\_t cudaResult = cudaSetDevice(deviceId);

if (cudaResult != cudaSuccess)

{

fprintf(stderr, "Cannot set current CUDA device, status = %d: %s\n",

cudaResult, cudaGetErrorString(cudaResult));

throw std::runtime\_error("Cannot set current CUDA device");

}

InitArrayGpu(dev\_array, size, value);

}

};

th.join();

}

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static void InitArray(DevMemArrPointer<T>& devMemArrPointer, T value)

{

if(!devMemArrPointer.IsInitialized())

throw std::runtime\_error("devMemArrPointer is not initialized!");

switch (devMemArrPointer.dataLocation)

{

case DataLocation::RAM:

InitArrayRam(devMemArrPointer.ptr, devMemArrPointer.length, value);

break;

case DataLocation::GPU0:

InitArrayGpu(devMemArrPointer.ptr, devMemArrPointer.length, value, 0);

break;

case DataLocation::GPU1:

InitArrayGpu(devMemArrPointer.ptr, devMemArrPointer.length, value, 1);

break;

case DataLocation::GPU2:

InitArrayGpu(devMemArrPointer.ptr, devMemArrPointer.length, value, 2);

break;

case DataLocation::GPU3:

InitArrayGpu(devMemArrPointer.ptr, devMemArrPointer.length, value, 3);

break;

default:

break;

}

}

////////////////////////// Инициализация массивов (конец) /////////////////////////////

////////////////////////// Считывание элементов массивов (начало) /////////////////////

template<typename T>

static T GetValueRAM(T\* arrayRam, unsigned long long index)

{

return arrayRam[index];

}

template<typename T>

static T GetValueGPU(T\* arrayGpu, unsigned long long index, unsigned deviceId = 0)

{

#ifdef \_\_NVCC\_\_

T value;

if(deviceId == 0)

{

CopyGpuToRam(arrayGpu + index, &value, 1);

}

else

{

std::thread th{

[&]() {

cudaSetDevice(deviceId);

CopyGpuToRam(arrayGpu + index, &value, 1);

}

};

th.join();

}

return value;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Считывание элементов массивов (конец) /////////////////////

////////////////////////// Установка значений элементов массивов (начало) /////////////////////

template<typename T>

static void SetValueRAM(T\* arrayRam, unsigned long long index, T value)

{

arrayRam[index] = value;

}

template<typename T>

static void SetValueGPU(T\* arrayGpu, unsigned long long index, unsigned deviceId, T value)

{

#ifdef \_\_NVCC\_\_

if(deviceId == 0)

{

CopyRamToGpu(&value, arrayGpu + index, 1);

}

else

{

std::thread th{

[&]() {

cudaSetDevice(deviceId);

CopyRamToGpu(&value, arrayGpu + index, 1);

}

};

th.join();

}

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Установка значений элементов массивов (конец) /////////////////////

////////////////////////// Копирование массивов (начало) /////////////////////////////

template<typename T>

static void CopyRamToGpu(T\* arrayRam, T\* arrayGpu, size\_t length)

{

#ifdef \_\_NVCC\_\_

size\_t dataSize = length \* sizeof(T);

cudaMemcpy(arrayGpu, arrayRam, dataSize, cudaMemcpyKind::cudaMemcpyHostToDevice);

cudaCheckErrors("Error in cudaMemcpy()");

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static void CopyRamToGpu(T\* arrayRam, T\* arrayGpu,

size\_t ind\_start, size\_t length, int deviceId = 0)

{

#ifdef \_\_NVCC\_\_

if(deviceId == 0)

{

CopyRamToGpu(arrayRam + ind\_start, arrayGpu + ind\_start, length);

}

else

{

std::thread th{

[&]() {

cudaSetDevice(deviceId);

CopyRamToGpu(arrayRam + ind\_start, arrayGpu + ind\_start, length);

}

};

th.join();

}

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static void CopyGpuToRam(T\* arrayGpu, T\* arrayRam, size\_t length)

{

#ifdef \_\_NVCC\_\_

size\_t dataSize = length \* sizeof(T);

cudaMemcpy(arrayRam, arrayGpu, dataSize, cudaMemcpyKind::cudaMemcpyDeviceToHost);

cudaCheckErrors("Error in cudaMemcpy()");

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static void CopyGpuToRam(T\* arrayGpu, T\* arrayRam,

size\_t ind\_start, size\_t length, int deviceId = 0)

{

#ifdef \_\_NVCC\_\_

if(deviceId == 0)

{

CopyGpuToRam(arrayGpu + ind\_start, arrayRam + ind\_start, length);

}

else

{

std::thread th{

[&]() {

cudaSetDevice(deviceId);

CopyGpuToRam(arrayGpu + ind\_start, arrayRam + ind\_start, length);

}

};

th.join();

}

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Копирование массивов (конец) /////////////////////////////

////////////////////////// Сравнение массивов (начало) /////////////////////////////

template<typename T>

static bool IsEqualsRamRam(T\* arrayRam1, T\*arrayRam2,

size\_t length, double eps = 0.00000001)

{

for (size\_t i = 0; i < length; i++)

{

if(fabs(arrayRam2[i] - arrayRam1[i]) > eps)

return false;

}

return true;

}

/// @brief Сравнивает содержимое массивов, расположенных на RAM и GPU

/// @tparam T

/// @param arrayRam

/// @param arrayGpu

/// @param length

/// @param eps

/// @return

template<typename T>

static bool IsEqualsRamGpu(T\* arrayRam, T\* arrayGpu,

size\_t length, double eps = 0.00000001)

{

#ifdef \_\_NVCC\_\_

bool isEquals = true;

const unsigned blockSize = 1000000;

unsigned blocksNum = length / blockSize;

unsigned lastBlockSize = length % blockSize;

T\* arrayRamTmp = new T[blockSize];

for (size\_t blockInd = 0; blockInd < blocksNum; blockInd++)

{

CopyGpuToRam(arrayGpu+blockInd\*blockSize, arrayRamTmp, blockSize);

isEquals = IsEqualsRamRam(arrayRam+blockInd\*blockSize, arrayRamTmp, blockSize);

if(!isEquals) break;

}

if(isEquals && lastBlockSize > 0)

{

CopyGpuToRam(arrayGpu+blocksNum\*blockSize, arrayRamTmp, lastBlockSize);

isEquals = IsEqualsRamRam(arrayRam+blocksNum\*blockSize, arrayRamTmp, lastBlockSize);

}

delete[] arrayRamTmp;

return isEquals;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Сравнение массивов (конец) /////////////////////////////

////////////////////////// Суммирование элементов массива (начало) /////////////////////////////

///// Последовательное суммирование на CPU /////

template<typename T>

static T Sum(T\* data, size\_t indStart, size\_t indEnd)

{

T result = 0;

for (size\_t i = indStart; i <= indEnd; i++)

{

result += data[i];

}

return result;

}

template<typename T>

static T Sum(T\* data, size\_t size)

{

T result = Sum(data, 0, size-1);

return result;

}

///////////////////////////////////////////////

///// Суммирование с помощью std::thread на CPU //////

// Структура для передачи аргументов в потоковую функцию

template<typename T>

struct SumThreadArgs

{

T\* data;

size\_t indStart;

size\_t indEnd;

T& sum;

std::mutex& m;

SumThreadArgs(T\* data,

size\_t indStart,

size\_t indEnd,

T& sum,

std::mutex& m) :

data(data),

indStart(indStart),

indEnd(indEnd),

sum(sum),

m(m)

{}

};

// Функция для исполнения потоком std::thread

template<typename T>

static void SumThread(SumThreadArgs<T> args)

{

T\* data = args.data;

auto indStart = args.indStart;

auto indEnd = args.indEnd;

T local\_sum = 0;

for (size\_t i = indStart; i <= indEnd; i++)

{

local\_sum += data[i];

}

{

std::lock\_guard<std::mutex> lock(args.m);

args.sum += local\_sum;

}

}

template<typename T>

static T Sum(T\* data, size\_t indStart, size\_t indEnd, unsigned threadsNum)

{

std::mutex m;

T sum = 0;

size\_t blockSize = indEnd - indStart + 1;

std::vector<std::thread> threads;

size\_t thBlockSize = blockSize / threadsNum;

for (size\_t i = 0; i < threadsNum; i++)

{

size\_t thIndStart = i \* thBlockSize;

size\_t thIndEnd = thIndStart + thBlockSize - 1;

if(i == threadsNum - 1)

thIndEnd = indEnd;

SumThreadArgs<T> args(data, thIndStart, thIndEnd, sum, m);

threads.push\_back(std::thread(SumThread<T>, args));

}

for(auto& th : threads)

{

th.join();

}

return sum;

}

template<typename T>

static T Sum(T\* data, size\_t size, unsigned threadsNum)

{

return Sum(data, 0, size - 1, threadsNum);

}

///////////////////////////////////////////////

///// Суммирование с помощью OpenMP на CPU /////

template<typename T>

static T SumOpenMP(T\* data, size\_t indStart, size\_t indEnd, unsigned threadsNum)

{

#ifdef \_OPENMP

omp\_set\_num\_threads(threadsNum);

T sum = 0;

#pragma omp parallel for reduction(+:sum)

for (long long i = (long long)indStart; i <= (long long)indEnd; i++)

{

sum += data[i];

}

return sum;

#else

throw std::runtime\_error("OpenMP not supported!");

#endif

}

template<typename T>

static T SumOpenMP(T\* data, size\_t size, unsigned threadsNum)

{

return SumOpenMP(data, 0, size - 1, threadsNum);

}

///// Суммирование с помощью Cuda /////

// Суммирование на одном GPU

template<typename T>

static T SumCuda(T\* dev\_arr, size\_t indStart, size\_t indEnd, unsigned blocksNum, unsigned threadsNum)

{

#ifdef \_\_NVCC\_\_

size\_t length = indEnd - indStart + 1;

#ifdef DEBUG

std::cout << "T Sum(" << dev\_arr << ", "

<< length << ", "<< blocksNum << ", "

<< threadsNum << ")" <<std::endl;

#endif

T sum{0};

//T\* dev\_sum;

//cudaMalloc(&dev\_sum, sizeof(T));

//cudaMemcpy(d, h, size, cudaMemcpyHostToDevice);

// Выделяем в распределяемой памяти каждого SM массив для хранения локальных сумм каждого потока блока

unsigned shared\_mem\_size = threadsNum \* sizeof(T);

#ifdef DEBUG

std::cout << "shared\_mem\_size = " << shared\_mem\_size << std::endl;

#endif

// Выделяем в RAM и глобальной памяти GPU массив для локальных сумм каждого блока

T\* block\_sum = (T\*)malloc(blocksNum \* sizeof(T));

T\* dev\_block\_sum;

cudaMalloc(&dev\_block\_sum, blocksNum \* sizeof(T));

kernel\_sum<<<blocksNum, threadsNum, shared\_mem\_size>>>(dev\_arr, length, dev\_block\_sum);

//cudaMemcpy(&sum, dev\_sum, sizeof(T), cudaMemcpyDeviceToHost);

cudaMemcpy(block\_sum, dev\_block\_sum, blocksNum \* sizeof(T), cudaMemcpyDeviceToHost);

for(unsigned i=0; i<blocksNum;i++)

{

//std::cout << "block\_sum[" << i << "] = " << block\_sum[i] << std::endl;

sum += block\_sum[i];

}

#ifdef DEBUG

std::cout << "SumCuda: Sum is " << sum << std::endl;

#endif

free(block\_sum);

cudaFree(dev\_block\_sum);

return sum;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static T SumCuda(T\* data, size\_t size, unsigned blocksNum, unsigned threadsNum)

{

return SumCuda(data, 0, size - 1, blocksNum, threadsNum);

}

// Суммирование на нескольких GPU

template<typename T>

static T SumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)

{

//std::cout << "SumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)\n\n";

#ifdef \_\_NVCC\_\_

T sum{0};

auto gpuNum = params.size();

std::vector<std::thread> threads;

std::mutex mutex;

for(int i = 0; i < gpuNum; i++)

{

threads.push\_back(std::thread{[i, &mutex, &params, &sum]() {

cudaSetDevice(i);

T gpu\_sum = SumCuda(params[i].dev\_arr,

params[i].indStart,

params[i].indEnd,

params[i].blocksNum,

params[i].threadsNum );

mutex.lock();

//std::cout << "thread " << i <<": ";

//params[i].Print();

//std::cout << "gpu\_sum = " << gpu\_sum <<"\n";

sum += gpu\_sum;

mutex.unlock();

}});

}

/\*

unsigned deviceId;

T\* dev\_arr;

size\_t indStart;

size\_t indEnd;

unsigned blocksNum;

unsigned threadsNum;

\*/

for(auto& thread : threads)

{

thread.join();

}

return sum;

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

// Суммирование на GPU cuBLAS

template<typename T>

static T SumCublas(cublasHandle\_t cublasH,

T\* dev\_arr, size\_t indStart, size\_t indEnd)

{

#ifdef \_\_NVCC\_\_

/\*std::cout << "!!!SumCublas(): cublasH: " << cublasH

<< "; dev\_arr: " << dev\_arr

<< "; indStart: " << indStart

<< "; indEnd: " << indEnd

<< std::endl;\*/

T result = 0;

//cublasHandle\_t cublasH = nullptr;

//cublasStatus\_t cublasStat = cublasCreate(&cublasH);

//CublasHelper::CheckCublasStatus(cublasStat, "CUBLAS initialization failed\n");

const int incx = 1;

cublasStatus\_t cublasStat;

if(typeid(T)==typeid(double))

{

cublasStat = cublasDasum(cublasH, indEnd-indStart+1, (double\*)dev\_arr, incx, &result);

}

else if(typeid(T)==typeid(float))

{

float\* dev\_arr\_float = (float\*)dev\_arr;

float result\_float = 0;

cublasStat = cublasSasum(cublasH, indEnd-indStart+1, dev\_arr\_float, incx, &result\_float);

result = (T) result\_float;

}

else

throw std::runtime\_error("typeid(T) not supported by cublas!");

CublasHelper::CheckCublasStatus(cublasStat, "cublas sum failed\n");

return result;

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static T SumCublas(cublasHandle\_t cublasH,

T\* dev\_arr, size\_t length)

{

return SumCublas(cublasH, dev\_arr, 0, length-1);

}

template<typename T>

static T SumCublas(cublasHandle\_t cublasH, ArrayGpuProcessingParams<T> params)

{

T sum = SumCublas(cublasH, params.dev\_arr, params.indStart, params.indEnd);

return sum;

}

// Суммирование на нескольких GPU с помощью CuBLAS

template<typename T>

static T SumCublasMultiGpu(std::vector<cublasHandle\_t> cublasHandles,

std::vector<T\*> dev\_arrays,

std::vector<size\_t> indStarts,

std::vector<size\_t> indEnds)

{

//std::cout << "SumCublasMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)\n\n";

#ifdef \_\_NVCC\_\_

T sum{0};

auto gpuNum = cublasHandles.size();

std::vector<std::thread> threads;

std::mutex mutex;

for(int i = 0; i < gpuNum; i++)

{

cublasHandle\_t cublasHandle = cublasHandles[i];

T\* dev\_arr = dev\_arrays[i];

size\_t indStart = indStarts[i];

size\_t indEnd = indEnds[i];

std::cout << "!!!SumCublasMultiGpu(): cublasHandle: " << cublasHandle

<< "; dev\_arr: " << dev\_arr

<< "; indStart: " << indStart

<< "; indEnd: " << indEnd

<< std::endl;

/\*std::cout << "!!!SumCublasMultiGpu(): cublasHandles[i]: " << cublasHandles[i]

<< "; dev\_arrays[i]: " << dev\_arrays[i]

<< "; indStarts[i]: " << indStarts[i]

<< "; indEnds[i]: " << indEnds[i]

<< std::endl;\*/

threads.push\_back(std::thread{[&mutex,

&sum, i, cublasHandle,

dev\_arr, indStart, indEnd]() {

cudaSetDevice(i);

T gpu\_sum = SumCublas(cublasHandle,

dev\_arr,

indStart,

indEnd);

mutex.lock();

//std::cout << "thread " << i <<": ";

//params[i].Print();

//std::cout << "gpu\_sum = " << gpu\_sum <<"\n";

sum += gpu\_sum;

mutex.unlock();

}});

}

for(auto& thread : threads)

{

thread.join();

}

return sum;

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Суммирование элементов массива (конец) /////////////////////////////

////////////////////////// Скалярное произведение элементов массива (начало) /////////////////////////////

template<typename T>

static T ScalarProductRamSeq(T\* arrayRam1, T\* arrayRam2, size\_t length)

{

T scalarProduct{0};

for (size\_t i = 0; i < length; i++)

{

scalarProduct += arrayRam1[i] \* arrayRam2[i];

}

return scalarProduct;

}

template<typename T>

static T ScalarProductRamParThread(T\* arrayRam1, T\* arrayRam2, size\_t length, unsigned threadsNum)

{

T scalarProduct{0};

std::mutex mutex;

std::vector<std::thread> threads;

size\_t blockSize = length / threadsNum;

for (size\_t i = 0; i < threadsNum; i++)

{

threads.push\_back(

std::thread{

[=, &mutex, &scalarProduct](){

T localSum = ScalarProductRamSeq(arrayRam1 + i\*blockSize,

arrayRam2 + i\*blockSize,

(i < threadsNum-1) ? blockSize : blockSize + length % threadsNum);

{

std::lock\_guard<std::mutex> guard{mutex};

scalarProduct += localSum;

}

}

}

);

}

for(auto& thread : threads)

{

thread.join();

}

return scalarProduct;

}

template<typename T>

static T ScalarProductRamParOpenMP(T\* array1Ram, T\* array2Ram, size\_t length, unsigned threadsNum)

{

#ifdef \_OPENMP

omp\_set\_num\_threads(threadsNum);

T scalarProduct = 0;

#pragma omp parallel for reduction(+:scalarProduct)

for (long long i = 0; i < (long long)length; i++)

{

scalarProduct += array1Ram[i]\*array2Ram[i];

}

return scalarProduct;

#else

throw std::runtime\_error("OpenMP not supported!");

#endif

}

static double ScalarProductRamCublas(double\* arrayRam1, double\* arrayRam2, size\_t length)

{

#ifdef OPENBLAS

double scalarProduct = cblas\_ddot(length, arrayRam1, 1, arrayRam2, 1);

return scalarProduct;

#else

throw std::runtime\_error("OpenBlas not supported!");

#endif

}

template<typename T>

static T ScalarProductGpuParCuda(T\* arrayGpu1, T\* arrayGpu2, size\_t length,

unsigned kernelBlocks, unsigned kernelThreads)

{

#ifdef \_\_NVCC\_\_

// Выделяем в распределяемой памяти каждого SM массив для хранения локальных сумм каждого потока блока

unsigned shared\_mem\_size = kernelThreads \* sizeof(T);

// Выделяем в RAM и глобальной памяти GPU массив для локальных сумм каждого блока

T\* blockSumsRam = CreateArrayRam<T>(kernelBlocks);

T\* blockSumsGpu = CreateArrayGpu<T>(kernelBlocks);

// Запуск ядра вычисления скалярного произведения

kernel\_scalar\_product<<<kernelBlocks, kernelThreads, shared\_mem\_size>>>(arrayGpu1, arrayGpu2, length, blockSumsGpu);

cudaCheckErrors("Error in kernel\_scalar\_product!\n");

// Копируем частичные суммы из GPU в RAM

std::cout << "Starting CopyGpuToRam... ";

CopyGpuToRam(blockSumsGpu, blockSumsRam, kernelBlocks);

std::cout << "OK\n";

T result = Sum(blockSumsRam, kernelBlocks);

// Освобождаем память

DeleteArrayRam(blockSumsRam);

DeleteArrayGpu(blockSumsGpu);

return result;

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static FuncResult<T> ScalarProductGpuParCuda(size\_t length,

unsigned kernelBlocks, unsigned kernelThreads)

{

#ifdef \_\_NVCC\_\_

T\* arrayGpu1 = CreateArrayGpu<T>(length);

T\* arrayGpu2 = CreateArrayGpu<T>(length);

ArrayHelper::InitArrayGpu(arrayGpu1, length, (T)10.0);

ArrayHelper::InitArrayGpu(arrayGpu2, length, (T)0.1);

auto start = high\_resolution\_clock::now();

T scalarProduct = ScalarProductGpuParCuda(arrayGpu1, arrayGpu2, length, kernelBlocks, kernelThreads);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

DeleteArrayGpu(arrayGpu1);

DeleteArrayGpu(arrayGpu2);

return FuncResult<T>{true, scalarProduct, t};

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static T ScalarProductMultiGpuParCuda(

std::vector<T\*> array1Gpus,

std::vector<T\*> array2Gpus,

std::vector<size\_t> lengthGpus,

unsigned kernelBlocks,

unsigned kernelThreads)

{

#ifdef \_\_NVCC\_\_

T scalarProduct{0};

auto gpuNum = array1Gpus.size();

std::vector<std::thread> threads;

std::mutex mutex;

for(int i = 0; i < gpuNum; i++)

{

threads.push\_back(std::thread{

[i, &mutex, &array1Gpus, &array2Gpus,

&lengthGpus, kernelBlocks, kernelThreads, &scalarProduct](){

cudaSetDevice(i);

std::cout << i << "; "

<< array1Gpus[i] << "; "

<< array2Gpus[i] << "; "

<< lengthGpus[i] << "; "

<< kernelBlocks << "; "

<< kernelThreads

<< std::endl;

T gpu\_scalarProduct = ScalarProductGpuParCuda(

array1Gpus[i],

array2Gpus[i],

lengthGpus[i],

kernelBlocks,

kernelThreads

);

mutex.lock();

//std::cout << "thread " << i <<": ";

//params[i].Print();

//std::cout << "scalarProduct = " << scalarProduct <<"\n";

scalarProduct += gpu\_scalarProduct;

mutex.unlock();

}});

}

for(auto& thread : threads)

{

thread.join();

}

return scalarProduct;

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static FuncResult<T> ScalarProductMultiGpuParCuda(size\_t length,

unsigned kernelBlocks, unsigned kernelThreads, std::vector<double> kGpuData)

{

#ifdef \_\_NVCC\_\_

std::vector<size\_t> gpuDataLength;

size\_t gpuDataLengthDistribution = length;

int gpuNum = kGpuData.size();

for (size\_t i = 0; i < gpuNum; i++)

{

std::cout << "kGpuData[" << i << "]: " << kGpuData[i] << "\n";

size\_t gpuDataLengthElement = kGpuData[i] \* length;

if(i == gpuNum - 1)

{

gpuDataLengthElement = gpuDataLengthDistribution;

}

gpuDataLength.push\_back(gpuDataLengthElement);

gpuDataLengthDistribution -= gpuDataLengthElement;

}

std::vector<T\*> array1Gpus;

std::vector<T\*> array2Gpus;

for (size\_t i = 0; i < gpuDataLength.size(); i++)

{

std::cout << "GPU " << i << ": "

<< gpuDataLength[i]

<< " from " << length << std::endl;

T\* array1Gpu = CreateArrayGpu<T>(gpuDataLength[i], i);

array1Gpus.push\_back(array1Gpu);

T\* array2Gpu = CreateArrayGpu<T>(gpuDataLength[i], i);

array2Gpus.push\_back(array2Gpu);

}

std::cout << "Arrays created!\n";

for (size\_t i = 0; i < gpuDataLength.size(); i++)

{

std::cout << "GPU " << i << ": "

<< gpuDataLength[i]

<< " from " << length << std::endl;

ArrayHelper::InitArrayGpu(array1Gpus[i], gpuDataLength[i], (T)10.0, i);

ArrayHelper::InitArrayGpu(array2Gpus[i], gpuDataLength[i], (T)0.1, i);

}

std::cout << "Arrays initialized!\n";

auto start = high\_resolution\_clock::now();

T scalarProduct = ScalarProductMultiGpuParCuda(array1Gpus, array2Gpus, gpuDataLength, kernelBlocks, kernelThreads);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

for (size\_t i = 0; i < gpuDataLength.size(); i++)

{

std::cout << "GPU " << i << ": "

<< gpuDataLength[i]

<< " from " << length << std::endl;

DeleteArrayGpu(array1Gpus[i], i);

DeleteArrayGpu(array2Gpus[i], i);

}

std::cout << "Arrays deleted!\n";

return FuncResult<T>{true, scalarProduct, t};

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static T ScalarProductGpuCublas(cublasHandle\_t cublasHandle,

T\* array1Gpu, T\* array2Gpu, size\_t length)

{

#ifdef \_\_NVCC\_\_

T result = 0;

int incx = 1;

int incy = 1;

cublasStatus\_t cublasStatus;

if(typeid(T)==typeid(double))

{

double\* array1Gpu\_double = (double\*)array1Gpu;

double\* array2Gpu\_double = (double\*)array2Gpu;

double result\_double = 0;

cublasStatus = cublasDdot(cublasHandle, length,

array1Gpu\_double, incx,

array2Gpu\_double, incy,

&result\_double);

result = (T) result\_double;

}

else if(typeid(T)==typeid(float))

{

float\* array1Gpu\_float = (float\*)array1Gpu;

float\* array2Gpu\_float = (float\*)array2Gpu;

float result\_float = 0;

cublasStatus = cublasSdot(cublasHandle, length,

array1Gpu\_float, incx,

array2Gpu\_float, incy,

&result\_float);

result = (T) result\_float;

}

else

throw std::runtime\_error("typeid(T) not supported by cublas!");

CublasHelper::CheckCublasStatus(cublasStatus, "cublas dot failed\n");

return result;

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

template<typename T>

static FuncResult<T> ScalarProductGpuCublas(size\_t length)

{

#ifdef \_\_NVCC\_\_

cublasHandle\_t cublasHandle = CublasHelper::CublasCreate();

T\* array1Gpu = CreateArrayGpu<T>(length);

T\* array2Gpu = CreateArrayGpu<T>(length);

ArrayHelper::InitArrayGpu(array1Gpu, length, (T)10.0);

ArrayHelper::InitArrayGpu(array2Gpu, length, (T)0.1);

auto start = high\_resolution\_clock::now();

T scalarProduct = ScalarProductGpuCublas(cublasHandle, array1Gpu, array2Gpu, length);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

DeleteArrayGpu(array1Gpu);

DeleteArrayGpu(array2Gpu);

CublasHelper::CublasDestroy(cublasHandle);

return FuncResult<T>{true, scalarProduct, t};

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

// Скалярное произведение на нескольких GPU с помощью CuBLAS

template<typename T>

static T ScalarProductMultiGpuCublas(std::vector<cublasHandle\_t> cublasHandles,

std::vector<T\*> dev\_arrays\_1,

std::vector<T\*> dev\_arrays\_2,

std::vector<size\_t> dev\_arrays\_lengths)

{

//std::cout << "ScalarProductMultiGpuCublas(...)\n\n";

#ifdef \_\_NVCC\_\_

T scalarProduct{0};

auto gpuNum = cublasHandles.size();

std::vector<std::thread> threads;

std::mutex mutex;

for(int i = 0; i < gpuNum; i++)

{

cublasHandle\_t cublasHandle = cublasHandles[i];

T\* dev\_arr\_1 = dev\_arrays\_1[i];

T\* dev\_arr\_2 = dev\_arrays\_2[i];

size\_t length = dev\_arrays\_lengths[i];

std::cout << "!!!ScalarProductMultiGpuCublas():"

<< " cublasHandle: " << cublasHandle

<< "; dev\_arr\_1: " << dev\_arr\_1

<< "; dev\_arr\_2: " << dev\_arr\_2

<< "; length: " << length

<< std::endl;

threads.push\_back(std::thread{[&mutex,

&scalarProduct, i, cublasHandle,

dev\_arr\_1, dev\_arr\_2, length]() {

cudaSetDevice(i);

T gpu\_scalarProduct = ScalarProductGpuCublas(cublasHandle,

dev\_arr\_1,

dev\_arr\_2,

length);

mutex.lock();

//std::cout << "thread " << i <<": ";

//params[i].Print();

//std::cout << "gpu\_scalarProduct = " << gpu\_scalarProduct <<"\n";

scalarProduct += gpu\_scalarProduct;

mutex.unlock();

}});

}

for(auto& thread : threads)

{

thread.join();

}

return scalarProduct;

//#ifdef \_\_NVCC\_\_

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

////////////////////////// Скалярное произведение элементов массива (конец) /////////////////////////////

////////////////////////// Умножение каждого элемента массива на число (начало) ////////////////////////

template<typename T, typename S>

static void MultiplyRam(T\* arrayRam, unsigned long long length, S scalar)

{

for (size\_t i = 0; i < length; i++)

{

arrayRam[i] \*= scalar;

}

}

template<typename T, typename S>

static void MultiplyRamParallel(T\* arrayRam, unsigned long long length,

S scalar, unsigned cpuThreadsNumber)

{

std::vector<std::thread> threads;

auto blockSize = length/cpuThreadsNumber;

auto blockSizeLast = length - blockSize \* (cpuThreadsNumber-1);

auto arrayRamTh = arrayRam;

for (size\_t i = 0; i < cpuThreadsNumber; i++)

{

if(i == cpuThreadsNumber - 1)

{

blockSize = blockSizeLast;

}

threads.push\_back(

std::thread(

[=](){

for (size\_t i = 0; i < blockSize; i++)

{

arrayRamTh[i] \*= scalar;

}

}

)

);

arrayRamTh += blockSize;

}

for(auto& thread : threads)

{

if(thread.joinable())

thread.join();

}

}

template<typename T, typename S>

static void MultiplyGpu(T\* arrayGpu, unsigned long long length,

unsigned deviceId, S scalar)

{

if(CudaHelper::IsCudaSupported())

{

if(deviceId == 0)

{

CudaHelper::Multiply(arrayGpu, length, scalar);

}

else

{

std::thread th{

[&]() {

//cudaSetDevice(deviceId);

CudaHelper::SetDevice(deviceId);

CudaHelper::Multiply(arrayGpu, length, scalar);

}

};

th.join();

}

}

else

throw std::runtime\_error("CUDA not supported!");

}

template<typename T, typename S>

static void Multiply(DevMemArrPointer<T> devMemArrPointer, S scalar)

{

switch (devMemArrPointer.dataLocation)

{

case DataLocation::RAM:

MultiplyRam(devMemArrPointer.ptr, devMemArrPointer.length, scalar);

break;

case DataLocation::GPU0:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 0, scalar);

break;

case DataLocation::GPU1:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 1, scalar);

break;

case DataLocation::GPU2:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 2, scalar);

break;

case DataLocation::GPU3:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 3, scalar);

break;

default:

break;

}

}

template<typename T, typename S>

static void MultiplyParallel(DevMemArrPointer<T> devMemArrPointer,

S scalar,

unsigned cpuThreadsNumber)

{

switch (devMemArrPointer.dataLocation)

{

case DataLocation::RAM:

MultiplyRamParallel(devMemArrPointer.ptr, devMemArrPointer.length, scalar, cpuThreadsNumber);

break;

case DataLocation::GPU0:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 0, scalar);

break;

case DataLocation::GPU1:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 1, scalar);

break;

case DataLocation::GPU2:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 2, scalar);

break;

case DataLocation::GPU3:

MultiplyGpu(devMemArrPointer.ptr, devMemArrPointer.length, 3, scalar);

break;

default:

break;

}

}

////////////////////////// Умножение каждого элемента массива на число (конец) ////////////////////////

/\* --- Другие алгоритмы --- \*/

};

==================================================

FILE: ArrayHelperFuncResult.hpp

PATH: Arrays\ArrayHelperFuncResult.hpp

EXTENSION: .hpp

SIZE: 1339 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../CommonHelpers/FuncResult.hpp"

#include "ArrayHelper.hpp"

struct ArrayHelperFuncResult

{

template<typename T>

static

FuncResult<T> SumOpenMP(T\* data, size\_t size, unsigned threadsNum)

{

try

{

auto start = high\_resolution\_clock::now();

T result = ArrayHelper::SumOpenMP(data, size, threadsNum);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto time\_mks = duration.count();

return FuncResult<T>(true, result, time\_mks);

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return FuncResult<T>();

}

}

template<typename T>

static

FuncResult<T> SumCublas(cublasHandle\_t cublasH, T\* data, size\_t size)

{

try

{

auto start = high\_resolution\_clock::now();

T result = ArrayHelper::SumCublas(cublasH, data, size);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto time\_mks = duration.count();

return FuncResult<T>(true, result, time\_mks);

} catch(const std::exception& e) {

std::cerr << e.what() << '\n';

return FuncResult<T>();

}

}};

==================================================

FILE: ArrayHelper\_ConsoleUI.hpp

PATH: Arrays\ArrayHelper\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 36927 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <chrono>

#include "../CommonHelpers/ConsoleHelper.hpp"

#include "../Cuda/CudaHelper.hpp"

#include "ArrayHelper.hpp"

/// @brief Структура для хранения консольного пользовательского интерфейса для методов класса ArrayHelper, обрабатывающих массивы T\*.

struct ArrayHelper\_ConsoleUI

{

/// @brief Выделение закрепленной памяти

static void CreateArrayRamPinned\_ConsoleUI()

{

std::cout << "CreateArrayRamPinned\_ConsoleUI\n";

if(!CudaHelper::IsCudaSupported())

{

std::cout << "Cuda not supported!" << std::endl;

return;

}

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length: ");

double\* arrPinned = ArrayHelper::CreateArrayRamPinned<double>(length);

if(!arrPinned)

{

std::cout << "Pinned RAM memory not allocated!" << std::endl;

return;

}

arrPinned[0] = 0.1;

std::cout << "arrPinned[0] = 0.1;\n";

std::cout << "arrPinned[0] = " << arrPinned[0] <<";\n";

ArrayHelper::DeleteArrayRamPinned(arrPinned);

std::cout << "Pinned RAM memory cleared!\n";

}

/// @brief Копирование данных из RAM в GPU

static void CopyRamToGpu\_ConsoleUI()

{

std::cout << "CopyRamToGpu\_ConsoleUI\n";

try

{

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

std::cout << "cudaDeviceNumber: " << cudaDeviceNumber << std::endl;

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length: ");

//double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

double value = 0;

int memoryType = ConsoleHelper::GetIntFromUser("Enter type of ram alloc (1-paged; 2-pinned): ");

// Инициализируем массив в RAM

double\* arrayRam = nullptr;

if(memoryType == 1)

arrayRam = new double[length];

else if (memoryType == 2)

arrayRam = ArrayHelper::CreateArrayRamPinned<double>(length);

else

{

std::cout << "type of ram alloc not recognized: " << memoryType << std::endl;

return;

}

for (size\_t i = 0; i < length; i++)

{

arrayRam[i] = value+0.1\*i;

}

std::cout << "arrayRam[0]: " << arrayRam[0] << std::endl;

std::cout << "arrayRam[length-1]: " << arrayRam[length-1] << std::endl;

for(int i = 0; i < cudaDeviceNumber; i++)

{

std::cout << "--- Starting work with GPU " << i << " ---\n";

std::cout << "Creating array on GPU " << i << "... ";

double\* arrayGpu = ArrayHelper::CreateArrayGpu<double>(length, i);

std::cout << "OK\n";

std::cout << "--- Copy to GPU" << i << " starting...\n";

auto start = high\_resolution\_clock::now();

ArrayHelper::CopyRamToGpu(arrayRam, arrayGpu, 0, length, i);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "Time, mks: " << t << std::endl;

std::cout << "Ram: ";

if(length>20)

ArrayHelper::PrintArrayRam(arrayRam, 0, 20);

else

ArrayHelper::PrintArrayRam(arrayRam, 0, length);

std::cout << "GPU " << i << ": ";

if(length>20)

ArrayHelper::PrintArrayGpu(arrayGpu, 0, 20, i);

else

ArrayHelper::PrintArrayGpu(arrayGpu, 0, length, i);

//arrayRam[length-1]+=0.00001;

bool isEquals = ArrayHelper::IsEqualsRamGpu(arrayRam, arrayGpu, length);

if (isEquals)

std::cout << "Success! Arrays are equals!\n";

else

std::cout << "Error! Arrays are not equals!\n";

std::cout << "--------------------------------\n";

ArrayHelper::DeleteArrayGpu(arrayGpu, i);

}

if(memoryType == 1)

delete[] arrayRam;

else if (memoryType == 2)

ArrayHelper::DeleteArrayRamPinned(arrayRam);

}

catch(const std::exception& e)

{

std::cerr << e.what() << std::endl;

}

}

/// @brief Копирование данных из GPU в RAM

static void CopyGpuToRam\_ConsoleUI()

{

std::cout << "CopyGpuToRam\_ConsoleUI\n";

try

{

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

std::cout << "cudaDeviceNumber: " << cudaDeviceNumber << std::endl;

size\_t size = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array size: ");

double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

// Инициализируем массив в RAM

double\* arrayRam = new double[size];

for (size\_t i = 0; i < size; i++)

{

arrayRam[i] = value;

}

//std::cout << "arrayRam[0]: " << arrayRam[0] << std::endl;

//std::cout << "arrayRam[size-1]: " << arrayRam[size-1] << std::endl;

for(int i = 0; i < cudaDeviceNumber; i++)

{

std::cout << "--- Starting work with GPU " << i << " ---\n";

std::cout << "Creating array on GPU " << i << "... ";

double\* dev\_array = ArrayHelper::CreateArrayGpu<double>(size, i);

std::cout << "OK\n";

std::cout << "Copy from RAM to GPU" << i << " starting...";

{

auto start = high\_resolution\_clock::now();

ArrayHelper::CopyRamToGpu(arrayRam, dev\_array, 0, size, i);

auto stop = high\_resolution\_clock::now();

std::cout << "OK\n";

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "Time, mks: " << t << std::endl;

}

std::cout << "Copy from GPU " << i << " to Ram starting...";

double\* arrayRamTmp = new double[size];

{

auto start = high\_resolution\_clock::now();

ArrayHelper::CopyGpuToRam(dev\_array, arrayRamTmp, 0, size, i);

auto stop = high\_resolution\_clock::now();

std::cout << "OK\n";

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "Time, mks: " << t << std::endl;

}

std::cout << "Ram: ";

if(size>20)

ArrayHelper::PrintArrayRam(arrayRam, 0, 20);

else

ArrayHelper::PrintArrayRam(arrayRam, 0, size);

std::cout << "GPU " << i << ": ";

if(size>20)

ArrayHelper::PrintArrayGpu(dev\_array, 0, 20, i);

else

ArrayHelper::PrintArrayGpu(dev\_array, 0, size, i);

std::cout << "Ram copied: ";

if(size>20)

ArrayHelper::PrintArrayRam(arrayRamTmp, 0, 20);

else

ArrayHelper::PrintArrayRam(arrayRamTmp, 0, size);

bool isEquals = ArrayHelper::IsEqualsRamRam(arrayRam, arrayRamTmp, size);

if(isEquals)

std::cout << "Checking equals: OK\n";

else

std::cout << "Checking equals: FALSE\n";

delete[] arrayRamTmp;

std::cout << "--------------------------------\n";

}

delete[] arrayRam;

}

catch(const std::exception& e)

{

std::cerr << e.what() << std::endl;

}

}

/// @brief Работа с функцией SumOpenMP

static void SumOpenMP\_ConsoleUI()

{

// Вызов функции суммирования с помощью OpenMP

try

{

size\_t size = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array size: ");

double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

int Nthreads = ConsoleHelper::GetIntFromUser("Enter num threads: ");

double\* data = new double[size];

for (size\_t i = 0; i < size; i++)

{

data[i] = value;

}

auto start = high\_resolution\_clock::now();

double sum = ArrayHelper::SumOpenMP(data, 0, size, Nthreads);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "ArrayRamHelper::SumOpenMP(data, 0, size, Nthreads): " << sum << std::endl;

std::cout << "Expected sum: " << size\*value << std::endl;

std::cout << "Time, mks: " << t << std::endl;

}

catch(const std::exception& e)

{

std::cerr << e.what() << std::endl;

}

}

/// @brief Работа с функцией SumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)

static void SumCudaMultiGpu\_ConsoleUI()

{

std::cout << "SumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)\n";

// Вызов функции суммирования с помощью Cuda на нескольких GPU

try

{

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

//cudaDeviceNumber = 1;

std::cout << "cudaDeviceNumber: " << cudaDeviceNumber << std::endl;

double expectedResult = 0;

std::vector<ArrayGpuProcessingParams<double>> params;

for(int i = 0; i < cudaDeviceNumber; i++)

{

std::cout << "--- Init " << i << " array starting...\n";

//size\_t size = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array size: ");

//double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

//int blocksNum = ConsoleHelper::GetIntFromUser("Enter num blocks: ");

//int threadsNum = ConsoleHelper::GetIntFromUser("Enter num threads: ");

size\_t size = 500000000ull;

double value = 0.001;

int blocksNum = 34;

int threadsNum = 16;

expectedResult += size\*value;

ArrayGpuProcessingParams<double> param;

param.deviceId = i;

param.indStart = 0;

param.indEnd = size-1;

param.blocksNum = blocksNum;

param.threadsNum = threadsNum;

try

{

param.dev\_arr = ArrayHelper::CreateArrayGpu<double>(size, i);

std::cout << "array " << i << " created\n";

std::cout << "First 10 elements of " << i << " array: ";

ArrayHelper::PrintArrayGpu(param.dev\_arr, 0, 10, i);

ArrayHelper::InitArrayGpu(param.dev\_arr, size, value, i);

std::cout << "array " << i << " initialized\n";

std::cout << "First 10 elements of " << i << " array: ";

ArrayHelper::PrintArrayGpu(param.dev\_arr, 0, 10, i);

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

std::exit(-1);

}

params.push\_back(param);

params[i].Print();

std::cout << "--- Initializing " << i << " array completed!\n";

}

auto start = high\_resolution\_clock::now();

double sum = ArrayHelper::SumCudaMultiGpu(params);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "ArrayRamHelper::SumCudaMultiGpu(...): " << sum << std::endl;

std::cout << "Expected sum: " << expectedResult << std::endl;

std::cout << "Time, mks: " << t << std::endl;

}

catch(const std::exception& e)

{

std::cerr << e.what() << std::endl;

}

}

/// @brief Работа с функцией SumCublas

static void SumCublas\_ConsoleUI()

{

std::cout << "SumCublas(...)\n";

// Вызов функции суммирования с помощью Cuda на нескольких GPU

try

{

bool isCudaSupported = CudaHelper::IsCudaSupported();

if(!isCudaSupported)

{

std::cout << "Cuda is not supported!" << std::endl;

return;

}

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

//cudaDeviceNumber = 1;

int deviceId = 0;

std::cout << "cudaDeviceNumber: " << cudaDeviceNumber << std::endl;

size\_t size = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array size: ");

//size\_t size = 200000000ull;

std::cout << "size: " << size << std::endl;

//double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

double value = 0.001;

//int blocksNum = ConsoleHelper::GetIntFromUser("Enter num blocks: ");

//int threadsNum = ConsoleHelper::GetIntFromUser("Enter num threads: ");

int blocksNum = 34;

int threadsNum = 16;

ArrayGpuProcessingParams<double> params;

params.deviceId = deviceId;

params.indStart = 0;

params.indEnd = size-1;

params.blocksNum = blocksNum;

params.threadsNum = threadsNum;

params.dev\_arr = ArrayHelper::CreateArrayGpu<double>(size, params.deviceId);

std::cout << "Array on device " << params.deviceId << " created!\n";

std::cout << "First 10 elements: ";

ArrayHelper::PrintArrayGpu(params.dev\_arr, 0, 10, params.deviceId);

ArrayHelper::InitArrayGpu(params.dev\_arr, size, value, params.deviceId);

std::cout << "array " << params.deviceId << " initialized\n";

std::cout << "First 10 elements of " << params.deviceId << " array: ";

ArrayHelper::PrintArrayGpu(params.dev\_arr, 0, 10, params.deviceId);

std::cout << "Initializing array completed!\n";

cublasHandle\_t cublasH = CublasHelper::CublasCreate();

double sum = 0;

auto start = high\_resolution\_clock::now();

sum = ArrayHelper::SumCublas(cublasH, params);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "ArrayHelper::SumCuBLAS(...): " << sum << std::endl;

std::cout << "Expected sum: " << size\*value << std::endl;

std::cout << "Time, mks: " << t << std::endl;

CudaHelper::CudaFree(params.dev\_arr);

CublasHelper::CublasDestroy(cublasH);

}

catch(const std::exception& e)

{

std::cerr << e.what() << std::endl;

}

}

/// @brief Работа с функцией SumCublasMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)

static void SumCublasMultiGpu\_ConsoleUI()

{

std::cout << "SumCublasMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)\n";

// Вызов функции суммирования с помощью Cublas на нескольких GPU

try

{

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

//cudaDeviceNumber = 1;

std::cout << "cudaDeviceNumber: " << cudaDeviceNumber << std::endl;

double expectedResult = 0;

std::vector<cublasHandle\_t> cublasHandles;

std::vector<double\*> dev\_arrays;

std::vector<size\_t> indStarts;

std::vector<size\_t> indEnds;

for(int i = 0; i < cudaDeviceNumber; i++)

{

cublasHandle\_t cublasHandle = CublasHelper::CublasCreate(i);

cublasHandles.push\_back(cublasHandle);

std::cout << "--- Init " << i << " array starting...\n";

//size\_t size = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array size: ");

//double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

size\_t size = 500000000ull;

double value = 0.001;

expectedResult += size\*value;

try

{

double\* dev\_arr = ArrayHelper::CreateArrayGpu<double>(size, i);

std::cout << "array " << i << " created\n";

std::cout << "First 10 from " << size <<" elements of " << i << " array: ";

ArrayHelper::PrintArrayGpu(dev\_arr, 0, 10, i);

ArrayHelper::InitArrayGpu(dev\_arr, size, value, i);

std::cout << "array " << i << " initialized\n";

std::cout << "First 10 from " << size <<" elements of " << i << " array: ";

ArrayHelper::PrintArrayGpu(dev\_arr, 0, 10, i);

std::cout << "--- Initializing " << i << " array completed!\n";

dev\_arrays.push\_back(dev\_arr);

indStarts.push\_back(0);

indEnds.push\_back(size-1);

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

std::exit(-1);

}

}

auto start = high\_resolution\_clock::now();

double sum = ArrayHelper::SumCublasMultiGpu(cublasHandles,

dev\_arrays, indStarts, indEnds);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "ArrayRamHelper::SumCudaMultiGpu(...): " << sum << std::endl;

std::cout << "Expected sum: " << expectedResult << std::endl;

std::cout << "Time, mks: " << t << std::endl;

}

catch(const std::exception& e)

{

std::cerr << e.what() << std::endl;

}

}

/// @brief Скалярное произведение векторов, расположенных в RAM

static void ScalarProductRamSeq\_ConsoleUI()

{

std::cout << "ScalarProductRamRamSeq\_ConsoleUI\n";

size\_t size = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array size: ");

double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

// Инициализируем массив в RAM

double\* arrayRam1 = new double[size];

double\* arrayRam2 = new double[size];

for (size\_t i = 0; i < size; i++)

{

arrayRam1[i] = value;

arrayRam2[i] = 1/value;

}

auto start = high\_resolution\_clock::now();

double scalarProduct = ArrayHelper::ScalarProductRamSeq(arrayRam1, arrayRam2, size);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "Time, mks: " << t << std::endl;

std::cout << "scalarProduct: " << scalarProduct << std::endl;

delete[] arrayRam1;

delete[] arrayRam2;

}

/// @brief Скалярное произведение векторов, расположенных в RAM, параллельно, std::thread

static void ScalarProductRamParThread\_ConsoleUI()

{

try

{

std::cout << "ScalarProductRamRamParThread\_ConsoleUI\n";

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length: ");

//double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

double value = 0.1;

size\_t threadsNum = ConsoleHelper::GetUnsignedIntFromUser("Enter number of threads: ");

// Инициализируем массив в RAM

double\* arrayRam1 = new double[length];

double\* arrayRam2 = new double[length];

for (size\_t i = 0; i < length; i++)

{

arrayRam1[i] = value;

arrayRam2[i] = 1/value;

}

auto start = high\_resolution\_clock::now();

double scalarProduct = ArrayHelper::ScalarProductRamParThread(arrayRam1, arrayRam2, length, threadsNum);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "Time, mks: " << t << std::endl;

std::cout << "scalarProduct: " << scalarProduct << std::endl;

delete[] arrayRam1;

delete[] arrayRam2;

}

catch(const std::exception& e)

{

std::cout << e.what() << '\n';

}

}

/// @brief Скалярное произведение векторов, расположенных в RAM, параллельно, OpenMP

static void ScalarProductRamParOpenMP\_ConsoleUI()

{

try

{

std::cout << "ScalarProductRamParOpenMP\_ConsoleUI\n";

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length: ");

//double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

double value = 0.1;

size\_t threadsNum = ConsoleHelper::GetUnsignedIntFromUser("Enter number of OpenMP threads: ");

// Инициализируем массив в RAM

double\* arrayRam1 = new double[length];

double\* arrayRam2 = new double[length];

for (size\_t i = 0; i < length; i++)

{

arrayRam1[i] = value;

arrayRam2[i] = 1/value;

}

auto start = high\_resolution\_clock::now();

double scalarProduct = ArrayHelper::ScalarProductRamParOpenMP(arrayRam1, arrayRam2, length, threadsNum);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "Time, mks: " << t << std::endl;

std::cout << "scalarProduct: " << scalarProduct << std::endl;

delete[] arrayRam1;

delete[] arrayRam2;

}

catch(const std::exception& e)

{

std::cout << e.what() << '\n';

}

}

/// @brief Скалярное произведение векторов, расположенных в RAM, параллельно, OpenBlas

static void ScalarProductRamOpenBlas\_ConsoleUI()

{

try

{

std::cout << "ScalarProductRamOpenBlas\_ConsoleUI\n";

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length: ");

//double value = ConsoleHelper::GetDoubleFromUser("Enter value: ","Error! Enter double value");

double value = 0.1;

//size\_t threadsNum = ConsoleHelper::GetUnsignedIntFromUser("Enter number of OpenMP threads: ");

// Инициализируем массив в RAM

double\* arrayRam1 = new double[length];

double\* arrayRam2 = new double[length];

for (size\_t i = 0; i < length; i++)

{

arrayRam1[i] = value;

arrayRam2[i] = 1/value;

}

std::cout << "\ncblas\_ddot\n";

double scalarProduct = 0;

try

{

auto start = high\_resolution\_clock::now();

scalarProduct = ArrayHelper::ScalarProductRamCublas(arrayRam1, arrayRam2, length);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "Time, mks: " << t << std::endl;

std::cout << "scalarProduct: " << scalarProduct << std::endl;

}

catch (const std::exception& exc)

{

std::cout << exc.what() << std::endl;

}

delete[] arrayRam1;

delete[] arrayRam2;

}

catch(const std::exception& e)

{

std::cout << e.what() << '\n';

}

}

/// @brief Скалярное произведение векторов, расположенных в GPU, параллельно, Cuda

static void ScalarProductGpuParCuda\_ConsoleUI()

{

std::cout << "ScalarProductGpuParCuda\_ConsoleUI()\n";

if(!CudaHelper::IsCudaSupported())

{

std::cout << "CUDA not supported!\n";

return;

}

try

{

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter arrays length: ");

unsigned kernelBlocks = ConsoleHelper::GetUnsignedIntFromUser("Enter number of CUDA blocks: ");

unsigned kernelThreads = ConsoleHelper::GetUnsignedIntFromUser("Enter number of CUDA threads in block: ");

auto resFloat = ArrayHelper::ScalarProductGpuParCuda<float>(length, kernelBlocks, kernelThreads);

std::cout << "float: ";

resFloat.Print();

auto resDouble = ArrayHelper::ScalarProductGpuParCuda<double>(length, kernelBlocks, kernelThreads);

std::cout << "double: ";

resDouble.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

/// @brief Скалярное произведение векторов, расположенных в нескольких GPU, параллельно, Cuda

static void ScalarProductMultiGpuParCuda\_ConsoleUI()

{

std::cout << "ScalarProductMultiGpuParCuda\_ConsoleUI()\n";

if(!CudaHelper::IsCudaSupported())

{

std::cout << "CUDA not supported!\n";

return;

}

try

{

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter arrays length: ");

unsigned kernelBlocks = ConsoleHelper::GetUnsignedIntFromUser("Enter number of CUDA blocks: ");

unsigned kernelThreads = ConsoleHelper::GetUnsignedIntFromUser("Enter number of CUDA threads in block: ");

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

std::vector<double> kGpuData;// Коэффициент распределения данных между GPU

double kGpuDistrubution = 1.0;

for (int i = 0; i < cudaDeviceNumber; i++)

{

std::string msg = "Enter k GPU " + std::to\_string(i);

msg += " [";

msg += CudaHelper::GetCudaDeviceName(i);

msg += "]";

msg += "(0.." + std::to\_string(kGpuDistrubution) + "): ";

double kGpu = ConsoleHelper::GetDoubleFromUser(msg);

if(kGpu<0)

kGpu = 0;

else if(kGpu>kGpuDistrubution)

kGpu=kGpuDistrubution;

kGpuDistrubution -= kGpu;

kGpuData.push\_back(kGpu);

std::cout << "Accepted: " << kGpu << "; ";

std::cout << "Remain: " << kGpuDistrubution << "\n";

}

auto resFloat = ArrayHelper::ScalarProductMultiGpuParCuda<float>(length, kernelBlocks, kernelThreads, kGpuData);

std::cout << "float: ";

resFloat.Print();

auto resDouble = ArrayHelper::ScalarProductMultiGpuParCuda<double>(length, kernelBlocks, kernelThreads, kGpuData);

std::cout << "double: ";

resDouble.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

/// @brief Скалярное произведение векторов, расположенных в GPU, параллельно, Cublas

static void ScalarProductGpuCublas\_ConsoleUI()

{

std::cout << "ScalarProductGpuCublas\_ConsoleUI()\n";

if(!CudaHelper::IsCudaSupported())

{

std::cout << "CUDA not supported!\n";

return;

}

try

{

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter arrays length: ");

auto resFloat = ArrayHelper::ScalarProductGpuCublas<float>(length);

std::cout << "float: ";

resFloat.Print();

auto resDouble = ArrayHelper::ScalarProductGpuCublas<double>(length);

std::cout << "double: ";

resDouble.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

/// @brief Работа с функцией ScalarProductMultiGpuCublas

static void ScalarProductMultiGpuCublas\_ConsoleUI()

{

std::cout << "ScalarProductMultiGpuCublas\_ConsoleUI()\n";

try

{

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

//cudaDeviceNumber = 1;

std::cout << "cudaDeviceNumber: " << cudaDeviceNumber << std::endl;

if(cudaDeviceNumber < 2)

{

std::cout << "GPU number must be greater 2!\n";

return;

}

std::vector<cublasHandle\_t> cublasHandles;

std::vector<double\*> dev\_arrays\_1;

std::vector<double\*> dev\_arrays\_2;

std::vector<size\_t> dev\_arrays\_lengths;

size\_t length = ConsoleHelper::GetUnsignedLongLongFromUser("Enter arrays length: ");

double expectedResult = length;

double value\_1 = 0.001;

double value\_2 = 1/value\_1;

for(int i = 0; i < cudaDeviceNumber; i++)

{

cublasHandle\_t cublasHandle = CublasHelper::CublasCreate(i);

cublasHandles.push\_back(cublasHandle);

std::cout << "--- Init data on GPU " << i << " ---\n";

double kGpu = ConsoleHelper::GetDoubleFromUser("Enter kGpu (0...1): ","Error! Enter double value");

size\_t size = length \* kGpu;

if(i==cudaDeviceNumber-1)

size = length - length \* kGpu \* i;

try

{

double\* dev\_arr\_1 = ArrayHelper::CreateArrayGpu<double>(size, i);

std::cout << "array 1 on GPU " << i << " created\n";

std::cout << "First 10 from " << size <<" elements of array 1 on GPU " << i << ": ";

ArrayHelper::PrintArrayGpu(dev\_arr\_1, 0, 10, i);

ArrayHelper::InitArrayGpu(dev\_arr\_1, size, value\_1, i);

std::cout << "array 1 on GPU " << i << " initialized\n";

std::cout << "First 10 from " << size <<" elements of array 2 on GPU " << i << ": ";

ArrayHelper::PrintArrayGpu(dev\_arr\_1, 0, 10, i);

std::cout << "--- Initializing array 1 on GPU " << i << " completed!\n";

dev\_arrays\_1.push\_back(dev\_arr\_1);

double\* dev\_arr\_2 = ArrayHelper::CreateArrayGpu<double>(size, i);

std::cout << "array 2 on GPU " << i << " created\n";

std::cout << "First 10 from " << size <<" elements of array 2 on GPU " << i << ": ";

ArrayHelper::PrintArrayGpu(dev\_arr\_2, 0, 10, i);

ArrayHelper::InitArrayGpu(dev\_arr\_2, size, value\_2, i);

std::cout << "array 1 on GPU " << i << " initialized\n";

std::cout << "First 10 from " << size <<" elements of array 2 on GPU " << i << ": ";

ArrayHelper::PrintArrayGpu(dev\_arr\_2, 0, 10, i);

std::cout << "--- Initializing array 2 on GPU " << i << " completed!\n";

dev\_arrays\_2.push\_back(dev\_arr\_2);

dev\_arrays\_lengths.push\_back(size);

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

std::exit(-1);

}

}

auto start = high\_resolution\_clock::now();

double scalarProduct = ArrayHelper::ScalarProductMultiGpuCublas(cublasHandles,

dev\_arrays\_1, dev\_arrays\_2, dev\_arrays\_lengths);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

std::cout << "ArrayRamHelper::ScalarProductMultiGpuCublas(...): " << scalarProduct << std::endl;

std::cout << "Expected scalarProduct: " << expectedResult << std::endl;

std::cout << "Time, mks: " << t << std::endl;

// Освобождение ресурсов

for (size\_t i = 0; i < dev\_arrays\_1.size(); i++)

{

ArrayHelper::DeleteArrayGpu(dev\_arrays\_1[i], i);

ArrayHelper::DeleteArrayGpu(dev\_arrays\_2[i], i);

}

CublasHelper::CublasDestroy(cublasHandles);

}

catch(const std::exception& e)

{

std::cerr << e.what() << std::endl;

}

}

};

==================================================

FILE: ArrayPerfTestHelper.hpp

PATH: Arrays\ArrayPerfTestHelper.hpp

EXTENSION: .hpp

SIZE: 4406 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

struct ArrayPerfTestHelper

{

static PerfTestResults PerfTest\_SumOpenMP(PerfTestParams perfTestParams)

{

std::cout << "ArrayPerfTestHelper::PerfTest\_SumOpenMP()\n";

size\_t arrLengthMin = perfTestParams.perfTestParamsData.arrayLengthMin;

size\_t arrLengthMax = perfTestParams.perfTestParamsData.arrayLengthMax;

size\_t arrLengthStep = perfTestParams.perfTestParamsData.arrayLengthStep;

for (size\_t dataLength = arrLengthMin;

dataLength <= arrLengthMax;

dataLength += arrLengthStep)

{// Цикл по размеру массива (начало)

std::cout << "---------- dataLength: " << dataLength << std::endl;

auto array = ArrayHelper::CreateArrayRam<double>(dataLength);

ArrayHelper::InitArrayRam(array, dataLength, 0.001);

auto cpuThreadNumMin = perfTestParams.perfTestParamsCpu.cpuThreadsNumMin;

auto cpuThreadNumMax = perfTestParams.perfTestParamsCpu.cpuThreadsNumMax;

auto cpuThreadNumStep = perfTestParams.perfTestParamsCpu.cpuThreadsNumStep;

for (auto cpuThreadsNum = cpuThreadNumMin;

cpuThreadsNum <= cpuThreadNumMax;

cpuThreadsNum += cpuThreadNumStep)

{// Цикл по количеству потоков CPU (начало)

std::cout << "----- cpuThreadsNum: " << cpuThreadsNum << std::endl;

std::vector<FuncResult<double>> results;

unsigned iterNumber = perfTestParams.iterNumber;

for (unsigned iterCnt = 0; iterCnt < iterNumber; iterCnt += 1)

{// Цикл по количеству итераций (начало)

auto result = ArrayHelperFuncResult::SumOpenMP(array, dataLength, cpuThreadsNum);

results.push\_back(result);

result.Print();

}// Цикл по количеству итераций (конец)

CalculationStatistics stat(results);

stat.Print();

ParallelCalcIndicators parallelCalcIndicators{};

parallelCalcIndicators.Print();

std::cout << "-----" << std::endl;

}// Цикл по количеству потоков CPU (конец)

ArrayHelper::DeleteArrayRam(array);

std::cout << "----------" << std::endl;

}// Цикл по размеру массива (конец)

PerfTestResults results;

return results;

}

static PerfTestResults PerfTest\_SumCublas(PerfTestParams perfTestParams)

{

std::cout << "ArrayPerfTestHelper::PerfTest\_SumCublas()\n";

cublasHandle\_t cublasH = CublasHelper::CublasCreate();

size\_t arrLengthMin = perfTestParams.perfTestParamsData.arrayLengthMin;

size\_t arrLengthMax = perfTestParams.perfTestParamsData.arrayLengthMax;

size\_t arrLengthStep = perfTestParams.perfTestParamsData.arrayLengthStep;

for (size\_t dataLength = arrLengthMin;

dataLength <= arrLengthMax;

dataLength += arrLengthStep)

{// Цикл по размеру массива (начало)

std::cout << "---------- dataLength: " << dataLength << std::endl;

auto array = ArrayHelper::CreateArrayGpu<double>(dataLength);

ArrayHelper::InitArrayGpu(array, dataLength, 0.001);

std::vector<FuncResult<double>> results;

unsigned iterNumber = perfTestParams.iterNumber;

for (unsigned iterCnt = 0; iterCnt < iterNumber; iterCnt += 1)

{// Цикл по количеству итераций (начало)

auto result = ArrayHelperFuncResult::SumCublas(cublasH, array, dataLength);

results.push\_back(result);

result.Print();

}// Цикл по количеству итераций (конец)

CalculationStatistics stat(results);

stat.Print();

ParallelCalcIndicators parallelCalcIndicators{};

parallelCalcIndicators.Print();

std::cout << "-----" << std::endl;

ArrayHelper::DeleteArrayGpu(array);

std::cout << "----------" << std::endl;

}// Цикл по размеру массива (конец)

CublasHelper::CublasDestroy(cublasH);

PerfTestResults results;

return results;

}

};

==================================================

FILE: ArrayPerfTestHelper\_ConsoleUI.hpp

PATH: Arrays\ArrayPerfTestHelper\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 3571 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

struct ArrayPerfTestHelper\_ConsoleUI

{

static void SumOpenMP\_ConsoleUI()

{

std::cout << "ArrayPerfTestHelper\_ConsoleUI::SumOpenMP\_ConsoleUI()\n";

unsigned iterNum = ConsoleHelper::GetUnsignedIntFromUser("Enter iterations number: ");

unsigned long long arrayLengthMin = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length min: ");

unsigned long long arrayLengthMax = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length max: ");

unsigned long long arrayLengthStep = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length step: ");

unsigned cpuThreadNumMin = ConsoleHelper::GetUnsignedIntFromUser("Enter num cpu threads min: ");

unsigned cpuThreadNumMax = ConsoleHelper::GetUnsignedIntFromUser("Enter num cpu threads max: ");

unsigned cpuThreadNumStep = ConsoleHelper::GetUnsignedIntFromUser("Enter num cpu threads step: ");

/\*std::cout

<< arrayLengthMin << " "

<< arrayLengthMax << " "

<< arrayLengthStep << " "

<< cpuThreadNumMin << " "

<< cpuThreadNumMax << " "

<< cpuThreadNumStep << " "

<< iterNum << " "

<< std::endl;\*/

DataTypes dataTypes;

dataTypes.Add(DataTypeEnum::dt\_float);

dataTypes.Add(DataTypeEnum::dt\_double);

PerfTestParamsData perfTestParamsData(dataTypes, arrayLengthMin, arrayLengthMax, arrayLengthStep);

PerfTestParamsCpu perfTestParamsCpu(cpuThreadNumMin, cpuThreadNumMax, cpuThreadNumStep);

PerfTestParams perfTestParams(iterNum, perfTestParamsData, perfTestParamsCpu);

perfTestParams.Print();

PerfTestResults results = ArrayPerfTestHelper::PerfTest\_SumOpenMP(perfTestParams);

results.Print();

}

static void SumCublas\_ConsoleUI()

{

std::cout << "ArrayPerfTestHelper\_ConsoleUI::SumCublas\_ConsoleUI()\n";

unsigned iterNum = ConsoleHelper::GetUnsignedIntFromUser("Enter iterations number: ");

unsigned long long arrayLengthMin = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length min: ");

unsigned long long arrayLengthMax = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length max: ");

unsigned long long arrayLengthStep = ConsoleHelper::GetUnsignedLongLongFromUser("Enter array length step: ");

/\*unsigned cpuThreadNumMin = ConsoleHelper::GetUnsignedIntFromUser("Enter num cpu threads min: ");

unsigned cpuThreadNumMax = ConsoleHelper::GetUnsignedIntFromUser("Enter num cpu threads max: ");

unsigned cpuThreadNumStep = ConsoleHelper::GetUnsignedIntFromUser("Enter num cpu threads step: ");\*/

/\*std::cout

<< arrayLengthMin << " "

<< arrayLengthMax << " "

<< arrayLengthStep << " "

<< cpuThreadNumMin << " "

<< cpuThreadNumMax << " "

<< cpuThreadNumStep << " "

<< iterNum << " "

<< std::endl;\*/

DataTypes dataTypes;

dataTypes.Add(DataTypeEnum::dt\_float);

dataTypes.Add(DataTypeEnum::dt\_double);

PerfTestParamsData perfTestParamsData(dataTypes, arrayLengthMin, arrayLengthMax, arrayLengthStep);

//PerfTestParamsGpu perfTestParamsGpu(cpuThreadNumMin, cpuThreadNumMax, cpuThreadNumStep);

//PerfTestParams perfTestParams(iterNum, perfTestParamsData, perfTestParamsCpu);

PerfTestParams perfTestParams(iterNum, perfTestParamsData);

perfTestParams.Print();

PerfTestResults results = ArrayPerfTestHelper::PerfTest\_SumCublas(perfTestParams);

results.Print();

}

};

==================================================

FILE: ArraysIndexMap.hpp

PATH: Arrays\ArraysIndexMap.hpp

EXTENSION: .hpp

SIZE: 1968 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <vector>

#include "ArrayBlockIndexes.hpp"

/// @brief Карта индексов

class ArraysIndexMap

{

std::vector<std::vector<unsigned long long>> indexMap;

public:

/// @brief Добавляет строку индексов

/// @param indStart Индекс первого элемента

/// @param indEnd Индекс последнего элемента

void AddIndexes(unsigned long long indStart, unsigned long long indEnd)

{

std::vector<unsigned long long> row;

row.push\_back(indStart);

row.push\_back(indEnd);

indexMap.push\_back(row);

}

/// @brief Возвращает объект, содержащий индексы блока, размер блока и локальный индекс

/// @param globalIndex Глобальный индекс элемента

/// @return ArrayBlockIndexes

ArrayBlockIndexes GetArrayBlockIndexes(unsigned long long globalIndex) const

{

ArrayBlockIndexes arrayBlockIndexes;

for (size\_t bi = 0; bi < indexMap.size(); bi++)

{

auto indStart = indexMap[bi][0];

auto indEnd = indexMap[bi][1];

if (globalIndex < indStart || globalIndex > indEnd)

continue;

arrayBlockIndexes.blockIndex = bi;

arrayBlockIndexes.blockLength = indEnd - indStart + 1;

arrayBlockIndexes.localIndex = globalIndex - indStart;

break;

}

return arrayBlockIndexes;

}

void Print()

{

std::cout << "ArraysIndexMap::Print()" << std::endl;

for (size\_t i = 0; i < indexMap.size(); i++)

{

auto& row = indexMap[i];

std::cout << row[0] << " " << row[1] << std::endl;

}

std::cout << "-----------------------" << std::endl;

}

};

==================================================

FILE: DevMemArrPointer.hpp

PATH: Arrays\DevMemArrPointer.hpp

EXTENSION: .hpp

SIZE: 3316 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/DataLocation.hpp"

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Указатель на массив, расположенный в памяти устройства вычислительного узла (RAM или GPU)

template<typename T>

struct DevMemArrPointer

{

/// Идентификатор указателя

unsigned id = 0;

// Место расположения данных

DataLocation dataLocation = DataLocation::None;

// Указатель на массив

T\* ptr = nullptr;

// Количество элементов

unsigned long long length = 0;

DevMemArrPointer()

{}

DevMemArrPointer(unsigned id,

DataLocation dataLocation,

T\* ptr,

unsigned long long length)

: id(id), dataLocation(dataLocation),

ptr(ptr), length(length)

{

}

/// @brief Возвращает флаг инициализации указателя

/// @return

bool IsInitialized() const

{

if( dataLocation == DataLocation::None

|| ptr == nullptr

|| length == 0)

{

return false;

}

return true;

}

/// @brief Возвращает флаг сброшенности указателя

/// @return

bool IsReset() const

{

if( dataLocation == DataLocation::None

&& ptr == nullptr

&& length == 0)

{

return false;

}

return true;

}

/// @brief Сбрасывает указатель в исходное неинициализированное состояние

void Reset()

{

dataLocation = DataLocation::None;

ptr = nullptr;

length = 0;

}

/// @brief Возвращает объём памяти, занимаемый структурой

/// @return unsigned long long (объём в байтах)

unsigned long long GetSizeStruct() const

{

return sizeof(\*this);

}

/// @brief Возвращает объём памяти, занимаемый массивом

/// @return

unsigned long long GetSizeData() const

{

return sizeof(T) \* length;

}

/// @brief Выводит в консоль сведения об указателе

/// @param pp

void Print(PrintParams pp = PrintParams{}) const

{

std::cout << pp.startMes;

std::cout << "id" << pp.splitterKeyValue << id;

std::cout << pp.splitter;

std::cout << "dataLocation" << pp.splitterKeyValue << dataLocation;

std::cout << pp.splitter;

std::cout << "ptr" << pp.splitterKeyValue << ptr;

std::cout << pp.splitter;

std::cout << "length" << pp.splitterKeyValue << length;

std::cout << pp.splitter;

std::cout << "GetSizeStruct()" << pp.splitterKeyValue << GetSizeStruct();

std::cout << pp.splitter;

std::cout << "GetSizeData()" << pp.splitterKeyValue << GetSizeData();

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: DevMemArrPointers.hpp

PATH: Arrays\DevMemArrPointers.hpp

EXTENSION: .hpp

SIZE: 12207 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/DataLocation.hpp"

#include "ArrayHelper.hpp"

#include "DevMemArrPointer.hpp"

#include "ArrayBlockIndexes.hpp"

template<typename T>

class DevMemArrPointers

{

// Массив указателей на части вектора, расположенные в различных областях памяти

std::vector<DevMemArrPointer<T>> dataPointers;

/// @brief Очищает dataPointers от сброшенных в исходное состояние объектов DevMemArrPointer<T>

void RemoveFreeDataPointers()

{

bool isClean = false;

while(!isClean)

{

isClean = true;

for (size\_t i = 0; i < dataPointers.size(); i++)

{

if(!dataPointers[i].IsReset())

{

isClean = false;

dataPointers.erase(dataPointers.begin() + i);

break;

}

}

}

}

public:

DevMemArrPointers()

{

}

void InitByVal(T value)

{

for (auto& devMemArrPointer : dataPointers)

{

if(!devMemArrPointer.IsInitialized())

continue;

ArrayHelper::InitArray(devMemArrPointer, value);

}

}

void Print() const

{

std::cout << "DevMemArrPointers::Print()" << std::endl;

std::cout << "dataPointers: ";

if(dataPointers.size() == 0)

std::cout << "none";

for (size\_t i = 0; i < dataPointers.size(); i++)

{

dataPointers[i].Print();

}

std::cout << std::endl;

}

size\_t GetSize() const

{

unsigned long long size = 0;

for (auto& devMemArrPointer : dataPointers)

{

size += devMemArrPointer.length;

}

return size;

}

/// @brief Возвращает количество выделенных блоков памяти

/// @return size\_t

auto GetDataPointersNum()

{

return dataPointers.size();

}

///// Выделение блоков памяти /////

/// @brief Выделяет непрерывный блок памяти

/// @param id Идентификатор блока

/// @param dataLocation Место расположения блока данных

/// @param length Количество элементов в блоке

/// @return DevMemArrPointer

DevMemArrPointer<T> AllocMem(unsigned id,

DataLocation dataLocation,

unsigned long long length)

{

T\* ptr = nullptr;

try

{

switch (dataLocation)

{

case DataLocation::RAM:

ptr = ArrayHelper::CreateArrayRam<T>(length);

break;

case DataLocation::GPU0:

ptr = ArrayHelper::CreateArrayGpu<T>(length, 0);

break;

case DataLocation::GPU1:

ptr = ArrayHelper::CreateArrayGpu<T>(length, 1);

break;

case DataLocation::GPU2:

ptr = ArrayHelper::CreateArrayGpu<T>(length, 2);

break;

case DataLocation::GPU3:

ptr = ArrayHelper::CreateArrayGpu<T>(length, 3);

break;

default:

break;

}

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return DevMemArrPointer<T>{};

}

if(!ptr)

return DevMemArrPointer<T>{};

DevMemArrPointer<T> dmptr(id, dataLocation, ptr, length);

dataPointers.push\_back(dmptr);

return dmptr;

}

/// @brief Добавляет непрерывный блок данных

/// @param dataLocation Место расположения блока данных

/// @param length Количество элементов в блоке

/// @return bool - Результат выполнения операции (true - успех)

bool AddBlock(DataLocation dataLocation,

unsigned long long length)

{

auto newBlockId = GetDataPointersNum();

auto newBlock = AllocMem(newBlockId, dataLocation, length);

if(newBlock.IsInitialized())

return true;

return false;

}

///////////////////////////////////

///// Освобождение памяти /////

/// @brief Освобождает зарезервированную память

void Clear(DevMemArrPointer<T>& devMemArrPointer)

{

std::cout << "Clear ";

devMemArrPointer.Print();

std::cout << std::endl;

try

{

switch (devMemArrPointer.dataLocation)

{

case DataLocation::RAM:

ArrayHelper::DeleteArrayRam<T>(devMemArrPointer.ptr);

break;

case DataLocation::GPU0:

ArrayHelper::DeleteArrayGpu<T>(devMemArrPointer.ptr, 0);

break;

case DataLocation::GPU1:

ArrayHelper::DeleteArrayGpu<T>(devMemArrPointer.ptr, 1);

break;

case DataLocation::GPU2:

ArrayHelper::DeleteArrayGpu<T>(devMemArrPointer.ptr, 2);

break;

case DataLocation::GPU3:

ArrayHelper::DeleteArrayGpu<T>(devMemArrPointer.ptr, 3);

break;

default:

break;

}

if(!devMemArrPointer.ptr)

{

devMemArrPointer.Reset();

}

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

/// @brief Освобождает всю зарезервированную память

void Clear()

{

// Очищаем зарезервированную память

for(auto& dataPointer : dataPointers)

{

Clear(dataPointer);

}

// Очищаем контейнер dataPointers

RemoveFreeDataPointers();

}

///////////////////////////////////

/// @brief Строит карту индексов

/// @return std::vector<std::vector<unsigned long long>>

ArraysIndexMap GetArraysIndexMap() const

{

// Строим карту индексов

// 0 999

// 1000 1099 и т.д.

ArraysIndexMap indexMap;

unsigned long long i = 0;

for (size\_t dp = 0; dp < dataPointers.size(); dp++)

{

auto indStart = i;

i += dataPointers[dp].length - 1;

auto indEnd = i;

indexMap.AddIndexes(indStart, indEnd);

i++;

}

return indexMap;

}

/// @brief Возвращает значение по глобальному индексу

/// @param globalIndex Глобальный индекс

/// @return

T GetValue(unsigned long long globalIndex) const

{

ArraysIndexMap map = GetArraysIndexMap();

//map.Print();

// Определяем индекс блока и локальный индекс элемента в блоке

ArrayBlockIndexes indexes = map.GetArrayBlockIndexes(globalIndex);

//std::cout << "globalIndex: [" << globalIndex << "]: ";

//indexes.Print();

if(!indexes.IsInitialized())

throw std::runtime\_error("Error in finding ArrayBlockIndexes by globalIndex!");

T value = GetValue(indexes.blockIndex, indexes.localIndex);

return value;

}

/// @brief Возвращает значение по индексу блока и локальному индексу

/// @param blockIndex

/// @param localIndex

/// @return

T GetValue(unsigned blockIndex, unsigned long long localIndex) const

{

auto& devMemArrPointer = dataPointers[blockIndex];

T value;

switch (devMemArrPointer.dataLocation)

{

case DataLocation::RAM:

value = ArrayHelper::GetValueRAM(devMemArrPointer.ptr, localIndex);

break;

case DataLocation::GPU0:

value = ArrayHelper::GetValueGPU(devMemArrPointer.ptr, localIndex, 0);

break;

case DataLocation::GPU1:

value = ArrayHelper::GetValueGPU(devMemArrPointer.ptr, localIndex, 1);

break;

case DataLocation::GPU2:

value = ArrayHelper::GetValueGPU(devMemArrPointer.ptr, localIndex, 2);

break;

case DataLocation::GPU3:

value = ArrayHelper::GetValueGPU(devMemArrPointer.ptr, localIndex, 3);

break;

default:

throw std::runtime\_error("Wrong DataLocation!");

}

return value;

}

/// @brief Устанавливает значение по глобальному индексу

/// @param globalIndex

/// @param value

/// @return

bool SetValue(unsigned long long globalIndex, T value)

{

ArraysIndexMap map = GetArraysIndexMap();

//map.Print();

// Определяем индекс блока и локальный индекс элемента в блоке

ArrayBlockIndexes indexes = map.GetArrayBlockIndexes(globalIndex);

//std::cout << "globalIndex: [" << globalIndex << "]: ";

//indexes.Print();

if(!indexes.IsInitialized())

return false;

bool isValueSetted = SetValue(indexes.blockIndex, indexes.localIndex, value);

return isValueSetted;

}

bool SetValue(unsigned blockIndex, unsigned long long localIndex, T value)

{

try

{

auto& devMemArrPointer = dataPointers[blockIndex];

switch (devMemArrPointer.dataLocation)

{

case DataLocation::RAM:

ArrayHelper::SetValueRAM(devMemArrPointer.ptr, localIndex, value);

break;

case DataLocation::GPU0:

ArrayHelper::SetValueGPU(devMemArrPointer.ptr, localIndex, 0, value);

break;

case DataLocation::GPU1:

ArrayHelper::SetValueGPU(devMemArrPointer.ptr, localIndex, 1, value);

break;

case DataLocation::GPU2:

ArrayHelper::SetValueGPU(devMemArrPointer.ptr, localIndex, 2, value);

break;

case DataLocation::GPU3:

ArrayHelper::SetValueGPU(devMemArrPointer.ptr, localIndex, 3, value);

break;

default:

throw std::runtime\_error("Wrong DataLocation!");

}

return true;

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

return false;

}

}

template<typename S>

void Multiply(S scalar, bool isParallel = false)

{

if(!isParallel)

{

for (auto devMemArrPointer : dataPointers)

{

ArrayHelper::Multiply(devMemArrPointer, scalar);

}

}

else

{

std::vector<std::thread> threads;

for (auto devMemArrPointer : dataPointers)

{

threads.push\_back(

std::thread{

[=](){

ArrayHelper::MultiplyParallel(devMemArrPointer, scalar, 10);

}

}

);

}

for(auto& th : threads)

{

if(th.joinable())

th.join();

}

}

}

};

==================================================

FILE: ConsoleHelper.hpp

PATH: CommonHelpers\ConsoleHelper.hpp

EXTENSION: .hpp

SIZE: 5362 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Вспомогательный класс для работы с консолью

struct ConsoleHelper

{

static void PrintLine(std::string string)

{

std::cout << string << std::endl;

}

static void WaitAnyKey(std::string message = "Press Enter to continue...")

{

std::cout << message;

std::cout << std::flush;

int ch = std::getchar();

while(1)

{

ch = std::getchar();

std::cout << "\nch: " << ch << "\n";

if(ch == 10)

break;

};

}

template<typename T>

static void PrintKeyValue(std::string key, T value,

std::string splitter = ": ",

bool isEndl = true)

{

std::cout << key << splitter << value;

if(isEndl)

std::cout << std::endl;

}

/// @brief Запрашивает у пользователя целое число

/// @param message Сообщение для пользователя

/// @param errorMessage Сообщение об ошибке

/// @return Введённое пользователем число

static std::string GetStringFromUser(std::string message)

{

std::cout << message;

std::string userInput;

if(char(std::cin.peek()) == '\n')

std::cin.ignore();

if (std::cin.fail())

{

std::cin.clear();

std::cin.ignore(32767, '\n');

}

getline(std::cin, userInput);

return userInput;

}

static bool GetBoolFromUser(std::string message, std::string errorMessage = "Error! Enter bool value (y, n, 0, 1)")

{

while (1)

{

try

{

std::cout << message;

std::string userInput;

std::cin >> userInput;

if(userInput == "y" || userInput == "1")

return true;

if(userInput == "n" || userInput == "0")

return false;

std::cout << errorMessage << std::endl;

}

catch(const std::exception& e)

{

std::cout << errorMessage << std::endl;

}

}

}

/// @brief Запрашивает у пользователя целое число

/// @param message Сообщение для пользователя

/// @param errorMessage Сообщение об ошибке

/// @return Введённое пользователем число

static int GetIntFromUser(std::string message, std::string errorMessage = "Error! Enter integer number")

{

while (1)

{

try

{

std::cout << message;

std::string userInput;

std::cin >> userInput;

int value = std::stoi(userInput);

return value;

}

catch(const std::exception& e)

{

std::cout << errorMessage << std::endl;

}

}

}

static unsigned GetUnsignedIntFromUser(std::string message, std::string errorMessage = "Error! Enter integer number")

{

while (1)

{

try

{

std::cout << message;

std::string userInput;

std::cin >> userInput;

if(isdigit(userInput[0]))

{

unsigned value = std::stoul(userInput);

return value;

}

std::cout << errorMessage << std::endl;

}

catch(const std::exception& e)

{

std::cout << errorMessage << std::endl;

}

}

}

static size\_t GetUnsignedLongLongFromUser(std::string message, std::string errorMessage = "Error! Enter integer number")

{

while (1)

{

try

{

std::cout << message;

std::string userInput;

std::cin >> userInput;

if(isdigit(userInput[0]))

{

size\_t value = std::stoull(userInput);

return value;

}

std::cout << errorMessage << std::endl;

}

catch(const std::exception& e)

{

std::cout << errorMessage << std::endl;

}

}

}

static double GetDoubleFromUser(std::string message = "Enter double value: ", std::string errorMessage = "Error! Enter double number")

{

while (1)

{

try

{

std::cout << message;

std::string userInput;

std::cin >> userInput;

if(isdigit(userInput[0]))

{

double value = std::stod(userInput);

return value;

}

std::cout << errorMessage << std::endl;

}

catch(const std::exception& e)

{

std::cout << errorMessage << std::endl;

}

}

}

};

==================================================

FILE: DataLocation.hpp

PATH: CommonHelpers\DataLocation.hpp

EXTENSION: .hpp

SIZE: 1158 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Место хранения данных (векторов, матриц и пр.)

enum class DataLocation

{

None = -2, // Данные нигде не хранятся (нулевая, единичная матрицы и пр.)

RAM = -1,

GPU0 = 0, // Видеопамять GPU0

GPU1 = 1, // Видеопамять GPU1

GPU2 = 2, // Видеопамять GPU2

GPU3 = 3 // Видеопамять GPU3

};

std::ostream& operator<<(std::ostream& os, DataLocation dl)

{

switch (dl)

{

case DataLocation::None:

os << "DataLocation::None";

break;

case DataLocation::RAM:

os << "DataLocation::RAM";

break;

case DataLocation::GPU0:

os << "DataLocation::GPU0";

break;

case DataLocation::GPU1:

os << "DataLocation::GPU1";

break;

case DataLocation::GPU2:

os << "DataLocation::GPU2";

break;

case DataLocation::GPU3:

os << "DataLocation::GPU3";

break;

default:

break;

}

return os;

}

==================================================

FILE: DataType.hpp

PATH: CommonHelpers\DataType.hpp

EXTENSION: .hpp

SIZE: 1539 bytes

----------------------------------------

CONTENT:

#pragma once

#include "DataTypeEnum.hpp"

#include "PrintParams.hpp"

struct DataType

{

DataTypeEnum dataTypeEnum;

DataType(DataTypeEnum dataTypeEnum) :

dataTypeEnum(dataTypeEnum)

{}

static bool TryParse(DataType& dataType, std::string str)

{

if(str == "float")

{

dataType.dataTypeEnum = DataTypeEnum::dt\_float;

return true;

}

else if(str == "double")

{

dataType.dataTypeEnum = DataTypeEnum::dt\_double;

return true;

}

return false;

}

/// @brief Возвращает объём памяти, занимаемый одним элементом данного типа

/// @return

unsigned GetSize() const

{

if (dataTypeEnum == DataTypeEnum::dt\_float)

return sizeof(float);

else if (dataTypeEnum == DataTypeEnum::dt\_double)

return sizeof(double);

throw std::runtime\_error("DataType::GetSize(): DataTypeEnum not recognized!");

}

/// @brief Выводит в консоль сведения о типе данных

/// @param pp

void Print(PrintParams pp = PrintParams{}) const

{

std::cout << pp.startMes;

std::cout << dataTypeEnum;

std::cout << pp.endMes;

if(pp.isEndl) std::cout << std::endl;

}

};

std::ostream& operator<<(std::ostream& os, DataType dts)

{

os << dts.dataTypeEnum;

return os;

}

==================================================

FILE: DataTypeEnum.hpp

PATH: CommonHelpers\DataTypeEnum.hpp

EXTENSION: .hpp

SIZE: 858 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

enum class DataTypeEnum

{

dt\_void,

dt\_int,

dt\_ull,

dt\_float,

dt\_ptr\_float,

dt\_double,

dt\_ptr\_double

};

std::ostream& operator<<(std::ostream& os, DataTypeEnum dt)

{

switch (dt)

{

case DataTypeEnum::dt\_void:

os << "void";

break;

case DataTypeEnum::dt\_int:

os << "int";

break;

case DataTypeEnum::dt\_ull:

os << "ull";

break;

case DataTypeEnum::dt\_float:

os << "float";

break;

case DataTypeEnum::dt\_ptr\_float:

os << "float\*";

break;

case DataTypeEnum::dt\_double:

os << "double";

break;

case DataTypeEnum::dt\_ptr\_double:

os << "double\*";

break;

default:

break;

}

return os;

}

==================================================

FILE: DataTypes.hpp

PATH: CommonHelpers\DataTypes.hpp

EXTENSION: .hpp

SIZE: 1388 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

#include "DataType.hpp"

#include "PrintParams.hpp"

/// @brief Список типов данных

class DataTypes

{

std::vector<DataType> dataTypes;

public:

void Add(DataType dataType)

{

dataTypes.push\_back(dataType);

}

void Print(PrintParams pp = PrintParams{}) const

{

std::cout << pp.startMes;

for (size\_t i = 0; i < dataTypes.size(); i++)

{

std::cout << i << pp.splitterKeyValue << dataTypes[i];

if(i<dataTypes.size()-1)

std::cout << "; ";

}

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

unsigned Count() const

{

return dataTypes.size();

}

DataType operator[](unsigned index) const

{

if(index >= Count())

{

std::cout << "\nError! Index out of range\n";

throw std::runtime\_error("Error in FunctionDataTypes::operator[]. Out of range!");

}

return dataTypes[index];

}

};

std::ostream& operator<<(std::ostream& os, DataTypes dts)

{

std::cout << "(";

for(unsigned i = 0; i < dts.Count(); i++)

{

std::cout << dts[i];

if (i < dts.Count() - 1)

std::cout << ", ";

}

std::cout << ")";

return os;

}

==================================================

FILE: DimensionEnum.hpp

PATH: CommonHelpers\DimensionEnum.hpp

EXTENSION: .hpp

SIZE: 541 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Размерность для геометрии и пр.

enum class Dimension

{

D1 = 1, // 1D

D2 = 2, // 2D

D3 = 3 // 3D

};

std::ostream& operator<<(std::ostream& os, Dimension dim)

{

switch (dim)

{

case Dimension::D1:

os << "1D";

break;

case Dimension::D2:

os << "2D";

break;

case Dimension::D3:

os << "3D";

break;

default:

break;

}

return os;

}

==================================================

FILE: FileSystemHelper.hpp

PATH: CommonHelpers\FileSystemHelper.hpp

EXTENSION: .hpp

SIZE: 6919 bytes

----------------------------------------

CONTENT:

#pragma once

//////////////// Файловая система (начало) ///////////////////

/// @brief Класс для работы с файловой системой

//#include <filesystem> // C++17

#include <fstream>

#include "sys/stat.h"

#if defined(\_WIN32)

#include<windows.h>

#endif

class FileSystemHelper

{

public:

/// @brief Комбинирует имя папки и имя файла в путь к файлу

/// @param dir\_name

/// @param file\_name

/// @return

static std::string CombinePath(const std::string& dir\_name, const std::string& file\_name)

{

return std::string{dir\_name + "/" + file\_name};

}

static std::string CombinePath()

{

std::cout << "Enter dir name: ";

std::string dirName;

std::cin >> dirName;

std::cout << "Enter file name: ";

std::string fileName;

std::cin >> fileName;

std::string path = CombinePath(dirName, fileName);

std::cout << "Path: " << path << std::endl;

return path;

}

/// @brief Проверяет существование файла

/// @return true - существует; false - не существует

static bool IsFileExists(const std::string& path\_file)

{

std::ifstream iff(path\_file);

//std::cout << "iff.good()" << iff.good() << std::endl;

return iff.good();

// C++17

/\*if(std::filesystem::exists(path\_file))

return true;

else

return false;\*/

}

/// @brief Проверяет существование файла

/// @return true - существует; false - не существует

static bool IsFileExists(const std::string& dir\_name, const std::string& file\_name)

{

auto filePath = CombinePath(dir\_name, file\_name);

if(IsFileExists(filePath))

return true;

else

return false;

}

static bool IsFileExists()

{

std::cout << "Enter file name: ";

std::string fileName;

std::cin >> fileName;

bool isExists = IsFileExists(fileName);

if(isExists)

std::cout << "File exists (true)" << std::endl;

else

std::cout << "File not exists (false)" << std::endl;

return isExists;

}

/// @brief Проверяет существование каталога

/// @return true - существует; false - не существует

static bool IsDirExists(const std::string& path\_dir)

{

std::string filePath = CombinePath(path\_dir,"tmp");

std::ofstream fout(filePath,std::ios::app);

bool isExists = fout.good();

fout.close();

if (isExists)// Удаляем временный файл

{

remove(filePath.c\_str());

}

return isExists;

// C++17

//if(std::filesystem::exists(path\_dir))

// return true;

//return false;

}

static bool IsDirExists()

{

std::cout << "Enter dir name: ";

std::string dirName;

std::cin >> dirName;

bool isExists = IsDirExists(dirName);

if(isExists)

std::cout << "Directory exists (true)" << std::endl;

else

std::cout << "Directory not exists (false)" << std::endl;

return isExists;

}

/// @brief Создаёт каталог

/// @return Результат создания нового каталога

static bool CreateDir(const std::string& path\_dir)

{

if(IsDirExists(path\_dir))

return false;

int errCode = 0;

#if defined(\_WIN32)

bool res = CreateDirectory(path\_dir.c\_str(), nullptr);

return res;

#else

errCode = mkdir(path\_dir.c\_str(), S\_IRWXU);

#endif

bool result = !(bool)errCode;

return result;

//return std::create\_directory(path\_dir);

}

static bool CreateDir()

{

std::cout << "Enter dir name: ";

std::string dirName;

std::cin >> dirName;

bool res = CreateDir(dirName);

if(res)

std::cout << "Directory created (true)" << std::endl;

else

std::cout << "Directory not created (false)" << std::endl;

return res;

}

static bool CreateFile(const std::string& dir\_name, const std::string& file\_name, const std::string& string\_data)

{

//auto filePath = CombinePath(dir\_name, file\_name);

std::string filePath = file\_name;

if(dir\_name.size()>0)

filePath = CombinePath(dir\_name, file\_name);

std::cout << "filePath: " << filePath << std::endl;

std::ofstream fout(filePath);

if(string\_data != "")

fout << string\_data;

fout.close();

return true;

}

static bool CreateFile()

{

std::cout << "Enter dir name (. - current dir): ";

std::string dirName;

std::cin >> dirName;

std::cout << "Enter file name: ";

std::string fileName;

std::cin >> fileName;

bool res = CreateFile(dirName, fileName, "");

if(res)

std::cout << "File created (true)" << std::endl;

else

std::cout << "File not created (false)" << std::endl;

return res;

}

static bool RemoveFile(const std::string& dir\_name, const std::string& file\_name)

{

auto filePath = CombinePath(dir\_name, file\_name);

int errCode = remove(filePath.c\_str());

return !(bool)errCode;

}

static bool RemoveFile()

{

std::cout << "Enter dir name: ";

std::string dirName;

std::cin >> dirName;

std::cout << "Enter file name: ";

std::string fileName;

std::cin >> fileName;

bool res = RemoveFile(dirName, fileName);

if(res)

std::cout << "File removed (true)" << std::endl;

else

std::cout << "File not removed (false)" << std::endl;

return res;

}

static bool RemoveDir(const std::string& dir\_name)

{

int errCode = remove(dir\_name.c\_str());

return !(bool)errCode;

}

static bool RemoveDir()

{

std::cout << "Enter dir name: ";

std::string dirName;

std::cin >> dirName;

bool res = RemoveDir(dirName);

if(res)

std::cout << "Directory removed (true)" << std::endl;

else

std::cout << "Directory not removed (false)" << std::endl;

return res;

}

};

/////////////////// Файловая система (конец) ///////////////////

==================================================

FILE: FuncResult.hpp

PATH: CommonHelpers\FuncResult.hpp

EXTENSION: .hpp

SIZE: 949 bytes

----------------------------------------

CONTENT:

#pragma once

#include "PrintParams.hpp"

template<typename T>

struct FuncResult

{

bool status{};

T result{};

long long time{};

FuncResult()

{ }

FuncResult(bool status, T result, long long time) :

status(status), result(result), time(time)

{ }

void Print(PrintParams pp = PrintParams{})

{

std::cout << pp.startMes;

std::cout << "status" << pp.splitterKeyValue << std::boolalpha << status;

std::cout << pp.splitter;

std::cout << "result" << pp.splitterKeyValue << result;

std::cout << pp.splitter;

std::cout << "time" << pp.splitterKeyValue << time << " mks";

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

static bool compare(const FuncResult<T>& left, const FuncResult<T>& right)

{

return left.time < right.time;

}

};

==================================================

FILE: LibSupport.hpp

PATH: CommonHelpers\LibSupport.hpp

EXTENSION: .hpp

SIZE: 947 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Структура для хранения поддерживаемых библиотек

struct LibSupport

{

bool IsOpenMP = false;// Поддержка OpenMP

bool IsCuda = false;// Поддержка CUDA

bool IsOpenBlas = false;// Поддержка OpenBlas

LibSupport()

{

// OpenMP

#ifdef \_OPENMP

IsOpenMP = true;

#endif

// CUDA

#ifdef \_\_NVCC\_\_

IsCuda = true;

#endif

// OpenBLAS

#ifdef OPENBLAS\_VERSION

IsOpenBlas = true;

#endif

}

void Print()

{

std::cout << "Supported libs: ";

if (IsOpenMP)

std::cout << "OpenMP ";

if (IsCuda)

std::cout << "CUDA ";

if (IsOpenBlas)

std::cout << "OpenBlas ";

std::cout << std::endl;

}

};

==================================================

FILE: MeasurementUnitEnum.hpp

PATH: CommonHelpers\MeasurementUnitEnum.hpp

EXTENSION: .hpp

SIZE: 643 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Размерность для геометрии и пр.

enum class MeasurementUnitEnum

{

Meter = 1, // Метры

Pascal = 2, // Паскали

Gramm = 3 // Граммы

};

std::ostream& operator<<(std::ostream& os, MeasurementUnitEnum mu)

{

switch (mu)

{

case MeasurementUnitEnum::Meter:

os << "m.";

break;

case MeasurementUnitEnum::Pascal:

os << "Pa.";

break;

case MeasurementUnitEnum::Gramm:

os << "g.";

break;

default:

break;

}

return os;

}

==================================================

FILE: PhysicalQuantityEnum.hpp

PATH: CommonHelpers\PhysicalQuantityEnum.hpp

EXTENSION: .hpp

SIZE: 706 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Размерность для геометрии и пр.

enum class PhysicalQuantityEnum

{

None, // Безразмерная величина, о.е.

T = 1, // Температура, град. Цельсия

P = 2 // Давление, Па

};

std::ostream& operator<<(std::ostream& os, PhysicalQuantityEnum dim)

{

switch (dim)

{

case PhysicalQuantityEnum::None:

os << "None";

break;

case PhysicalQuantityEnum::T:

os << "T";

break;

case PhysicalQuantityEnum::P:

os << "P";

break;

default:

break;

}

return os;

}

==================================================

FILE: PrintParams.hpp

PATH: CommonHelpers\PrintParams.hpp

EXTENSION: .hpp

SIZE: 811 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

struct PrintParams

{

std::string startMes = "[";

std::string splitterKeyValue = ": ";

std::string splitter = "; ";

std::string endMes = "]";

bool isEndl = true;

PrintParams& SetIsEndl(bool isEndLine = true)

{

isEndl = isEndLine;

return \*this;

}

void PrintStartMessage()

{

std::cout << startMes;

}

void PrintEndMessage()

{

std::cout << endMes;

}

void PrintSplitter()

{

std::cout << splitter;

}

void PrintKeyValue(std::string key, unsigned value)

{

std::cout << key << splitterKeyValue << value;

}

void PrintIsEndl()

{

if (isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: \_IncludeCommonHelpers.hpp

PATH: CommonHelpers\\_IncludeCommonHelpers.hpp

EXTENSION: .hpp

SIZE: 371 bytes

----------------------------------------

CONTENT:

#pragma once

#include "LibSupport.hpp"

#include "FuncResult.hpp"

#include "PrintParams.hpp"

#include "ConsoleHelper.hpp"

#include "FileSystemHelper.hpp"

#include "DataTypeEnum.hpp"

#include "DataType.hpp"

#include "DataTypes.hpp"

#include "DataLocation.hpp"

#include "DimensionEnum.hpp"

#include "MeasurementUnitEnum.hpp"

#include "PhysicalQuantityEnum.hpp"

==================================================

FILE: ComputingSystem.hpp

PATH: ComputingSystem\ComputingSystem.hpp

EXTENSION: .hpp

SIZE: 4836 bytes

----------------------------------------

CONTENT:

#pragma once

#include <map>

/// @brief Вычислительная система

class ComputingSystem

{

int id{0};// Идентификатор вычислительной системы

std::string name{"TestSystem"}; // Наименование вычислительной системы

std::string description{"TestSystem description"}; // Описание вычислительной системы

std::string file\_name{"ComputingSystem.txt"};// Имя файла с описанием вычислительной системы

std::map<unsigned, ComputingSystemNode> nodes;// Вычислительные узлы

public:

ComputingSystem()

{}

ComputingSystem(int id,

std::string name,

std::string description,

std::string file\_name = "ComputingSystem.txt"

) : id(id),

name(name),

description(description),

file\_name(file\_name)

{}

/// @brief Добавляет вычислительный узел в вычислительную систему

void AddNode(ComputingSystemNode node)

{

nodes[node.GetId()] = node;

}

/// @brief Выводит в консоль сведения о вычислительной системе

void Print()

{

std::cout << "Computing system details:"

<< "\nid: " << id

<< "\nname: " << name

<< "\ndescription: " << description

<< "\nfile\_name: " << file\_name

<< std::endl;

std::cout << "Nodes number: " << GetNodesNumber() << std::endl;

for ( auto& node : nodes)

{

node.second.Print(PrintParams{"--- Node [", ": ", "; ", "] ---"});

}

}

/// @brief Возвращает количество узлов вычислительной системы

/// @return

unsigned GetNodesNumber()

{

unsigned cnt{0};

/\*for ( auto& node : nodes)

{

cnt++;

}\*/

cnt = nodes.size();

return cnt;

}

/// @brief Возвращает идентификатор вычислительной системы

/// @return

int GetId() const

{

return id;

}

/// @brief Устанавливает идентификатор вычислительной системы

/// @param id

void SetId(int id)

{

this->id = id;

}

/// @brief Записать сведения о вычислительной системе

/// @param dir\_name Каталог для записи

/// @return

bool Serialize(const std::string& dir\_name)

{

// Создаём каталог dir\_name/id

std::string path\_dir = FileSystemHelper::CombinePath(dir\_name, std::to\_string(id));

bool result = FileSystemHelper::CreateDir(path\_dir);

if (!result)

{

std::cerr << "Cannot create dir " << path\_dir << std::endl;

return false;

}

std::string data = std::to\_string(id) + "\n" + name + "\n" + description + "\n";

FileSystemHelper::CreateFile(path\_dir, file\_name, data);

return true;

}

static ComputingSystem Deserialize(const std::string& dir\_name,

const int id,

const std::string& file\_name = "ComputingSystem.txt")

{

ComputingSystem computingSystem;

std::string dir\_Path = FileSystemHelper::CombinePath(dir\_name,

std::to\_string(id));

std::string filePath = FileSystemHelper::CombinePath(dir\_Path, file\_name);

std::ifstream fin(filePath);

if(!fin.good())

throw std::runtime\_error("Error in opening file " + filePath);

int f\_id;

std::string f\_name;

std::string f\_description;

//fin >> f\_id;

std::string f\_id\_str;

std::getline(fin, f\_id\_str);

f\_id = std::stoi(f\_id\_str);

if(f\_id != id)

throw std::runtime\_error("Error in file " + filePath);

//fin >> f\_name;

std::getline(fin, f\_name);

std::getline(fin, f\_description);

return ComputingSystem(f\_id, f\_name, f\_description);

}

static ComputingSystem GetDataFromUser()

{

int id = ConsoleHelper::GetIntFromUser("Enter computing system id: ");

std::string name = ConsoleHelper::GetStringFromUser("Enter computing system name: ");

std::string description = ConsoleHelper::GetStringFromUser("Enter computing system description: ");

return ComputingSystem(id, name, description);

}

};

==================================================

FILE: ComputingSystemNode.hpp

PATH: ComputingSystem\ComputingSystemNode.hpp

EXTENSION: .hpp

SIZE: 2382 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <map>

#include "../CommonHelpers/PrintParams.hpp"

#include "GpuParams.hpp"

/// @brief Узел вычислительной системы

class ComputingSystemNode

{

/// @brief УИД вычислительного узла

unsigned id{0};

/// @brief Количество потоков CPU, задействуемых в вычислениях

unsigned threadsNum{0};

/// @brief Максимальный объём RAM, доступной к задействованию в вычислениях

unsigned RamSize{0};

/// @brief Сведения о центральном процессоре

CpuParams cpuParams;

/// @brief Сведения об оперативной памяти (RAM)

RamParams ramParams;

/// @brief Сведения о GPU, задействуемых в вычислениях

std::map<unsigned, GpuParams> Gpus;

public:

unsigned GetId() const

{

return id;

}

unsigned GetGpuNum() const

{

return Gpus.size();

}

bool IsGpuExists(unsigned id)

{

if(Gpus.count(id)>0)

return true;

return false;

}

bool AddGpu(GpuParams gpu)

{

if(IsGpuExists(gpu.id))

return false;

Gpus[gpu.id] = gpu;

return true;

}

void AddCpu(CpuParams cpuParameters)

{

cpuParams = cpuParameters;

}

void AddRam(RamParams ramParameters)

{

ramParams = ramParameters;

}

void Print(PrintParams pp)

{

//std::cout << "----- ComputingSystemNode::Print(PrintParameters pp) -----" << std::endl;

std::cout << pp.startMes;

std::cout << "id" << pp.splitterKeyValue << id;

std::cout << pp.splitter;

std::cout << "GPU number" << pp.splitterKeyValue << GetGpuNum();

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

cpuParams.Print(PrintParams{"CPU: ["});

ramParams.Print(PrintParams{"RAM: ["});

for(auto& gpu : Gpus)

{

gpu.second.Print(PrintParams{"GPU: ["});

}

std::cout << "---------------------------------------------------------" << std::endl;

}

};

==================================================

FILE: ComputingSystemRepository.hpp

PATH: ComputingSystem\ComputingSystemRepository.hpp

EXTENSION: .hpp

SIZE: 9690 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Репозиторий сведений о вычислительных системах

class ComputingSystemRepository

{

bool isInitialized = false;

std::string dir\_name = "ComputingSystemRepository";// Имя каталога со сведениями о вычислительных системах

std::string file\_name = "List.txt";// Имя файла со сведениями о вычислительных системах

std::vector<int> computerSystemIds;// Вектор идентификаторов вычислительных систем

// Кэш сведений о вычислительных системах

std::map<unsigned, ComputingSystem> computingSystemCache;

/// @brief Проверка существования каталогов

void CheckDirectories()

{

if(!FileSystemHelper::IsDirExists(dir\_name))

FileSystemHelper::CreateDir(dir\_name);

}

void CheckFiles()

{

if(!FileSystemHelper::IsFileExists(dir\_name, file\_name))

{

bool result = FileSystemHelper::CreateFile(dir\_name, file\_name, "ComputingSystemRepository");

if (!result)

{

std::cerr << "File " + file\_name + " in directory " + dir\_name + " is not created!";

exit(-1);

}

}

}

/// @brief Считывает содержимое файла со сведениями о вычислительных системах

/// @return

bool ReadFile()

{

std::string filePath = FileSystemHelper::CombinePath(dir\_name, file\_name);

std::ifstream f(filePath);

if(!f.is\_open())

{

std::string message = "File \"" + filePath + "\" is not opened!";

std::cerr << message << std::endl;

return false;

}

// Проверка формата файла

std::string str;

f >> str;

if (str != "ComputingSystemRepository")

{

std::string message = "File \"" + filePath + "\" format is not AppConfig!";

std::cerr << message << std::endl;

return false;

}

// Считываем пары "Параметр Значение"

int value;

while(f >> value)

{

//std::cout << value << std::endl;

computerSystemIds.push\_back(value);

}

return true;

}

/// @brief Записывает new\_id в конец файла

/// @param new\_id

/// @return

bool AddIdToFile(const int new\_id)

{

std::string filePath = FileSystemHelper::CombinePath(dir\_name, file\_name);

std::ofstream fout(filePath,std::ios::app);

if(!fout.is\_open())

{

return false;

}

fout << '\n' << new\_id ;

fout.close();

return true;

}

public:

ComputingSystemRepository(bool isInitialized = true)

: isInitialized(isInitialized)

{

CheckAndReadIfInitialized();

}

ComputingSystemRepository(std::string dir\_name)

: dir\_name(dir\_name)

{

isInitialized = true;

CheckAndReadIfInitialized();

}

void CheckAndReadIfInitialized()

{

if(isInitialized)

{

CheckDirectories();

CheckFiles();

ReadFile();

Init();

}

}

bool IsExists(int computingSystemId) const

{

for(auto& id : computerSystemIds)

{

if(id == computingSystemId)

return true;

}

return false;

}

bool TryAddComputingSystem(ComputingSystem& computingSystem)

{

int new\_id = computingSystem.GetId();

// Если уже есть информация о вычислительной системе

// с таким идентификатором, информацию не добавляем

// и возвращаем false

if (IsExists(new\_id))

return false;

// Записать данные о выч. системе в каталог dir\_name

computingSystem.Serialize(dir\_name);

AddIdToFile(new\_id);

computerSystemIds.push\_back(new\_id);

return true;

}

ComputingSystem GetComputingSystem(int id)

{

if(!IsExists(id))

throw std::logic\_error("Computing system not found!");

try

{

auto entry = computingSystemCache[id];

return entry;

}

catch(const std::exception& e)

{

//std::cerr << e.what() << '\n';

}

return ComputingSystem::Deserialize(dir\_name, id);

}

/// @brief 2 Print config

void PrintConfig()

{

std::cout << "dir\_name: " << dir\_name << "; ";

std::cout << "file\_name: " << file\_name << std::endl;

}

/// @brief 3 Print computing system list

void PrintList()

{

std::cout << "Computing system ids: [";

for(auto& id : computerSystemIds)

std::cout << id << " ";

std::cout << "]" << std::endl;

}

/// @brief 4 Print computing system details

void PrintDetails()

{

std::cout << "PrintDetails()" << std::endl;

int id = ConsoleHelper::GetIntFromUser("Enter computing system id: ");

if(!IsExists(id))

{

std::cout << "Not found!" << std::endl;

return;

}

try

{

ComputingSystem computingSystem = GetComputingSystem(id);

computingSystem.Print();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

/// @brief 5 Add computing system

void Add()

{

std::cout << "Add()" << std::endl;

ComputingSystem computingSystem = ComputingSystem::GetDataFromUser();

if(TryAddComputingSystem(computingSystem))

{

std::cout << "Computing system " << computingSystem.GetId() << " added." << std::endl;

}

else

{

std::cout << "Error in adding computing system " << computingSystem.GetId() << "!" << std::endl;

}

}

/// @brief 6 Change computing system

void Change()

{

std::cout << "ComputingSystemRepository::Change()" << std::endl;

}

/// @brief 7 Remove computing system

void Remove()

{

std::cout << "ComputingSystemRepository::Remove()" << std::endl;

}

/// @brief 8 Is computing system exists

void IsExists()

{

int compSystemId = ConsoleHelper::GetIntFromUser("Enter computing system id: ", "Error! Enter integer number!");

bool isExists = IsExists(compSystemId);

std::cout << "id: " << compSystemId << "; ";

std::cout << "isExists: " << isExists << std::endl;

}

/// @brief 9 Clear computing system repository

void Clear()

{

std::cout << "ComputingSystemRepository::Clear()" << std::endl;

bool result = FileSystemHelper::RemoveDir(dir\_name);

if(result)

std::cout << "Clearing success!" << std::endl;

else

std::cout << "Clearing error!" << std::endl;

}

/// @brief 10 Init computing system repository. Fill repository computing systems data.

void Init()

{

std::cout << "ComputingSystemRepository::Init()" << std::endl;

////////////////////////////////

ComputingSystem cs1{1, "i3-8G-MX250-2G","Notebook i3-8G-MX250-2G"};

CpuParams cpu1cs1n1{};

cpu1cs1n1.id = 0;

cpu1cs1n1.name = "Intel Core i3-10110U 2.1GHz";

cpu1cs1n1.ThreadsNumber = 4;

RamParams ramcs1n1{};

ramcs1n1.RamSizeGb = 8;

ramcs1n1.RamBandwidthGbS = 19.2;

GpuParams gpu1cs1n1{};

gpu1cs1n1.id = 0;

gpu1cs1n1.name = "NVIDIA GeForce MX250";

gpu1cs1n1.VRamSizeGb = 3.9;

gpu1cs1n1.SmNumber = 3;

gpu1cs1n1.PeakMemoryBandwidthGbS = 48.064;

ComputingSystemNode cs1n1{};

cs1n1.AddGpu(gpu1cs1n1);

cs1n1.AddCpu(cpu1cs1n1);

cs1n1.AddRam(ramcs1n1);

cs1.AddNode(cs1n1);

computingSystemCache[cs1.GetId()] = cs1;

computerSystemIds.push\_back(cs1.GetId());

////////////////////////////////

////////////////////////////////

ComputingSystem cs2{2, "i5-32G-RTX2060S-8G","PC i5-32G-RTX2060S-8G"};

CpuParams cpu1cs2n1{};

cpu1cs2n1.id = 0;

cpu1cs2n1.name = "Intel Core i5-6600 3.3GHz";

cpu1cs2n1.ThreadsNumber = 4;

RamParams ramcs2n1{};

ramcs2n1.RamSizeGb = 32;

ramcs2n1.RamBandwidthGbS = 19.2;

GpuParams gpu1cs2n1{};

gpu1cs2n1.id = 0;

gpu1cs2n1.name = "NVIDIA GeForce RTX 2060 SUPER";

gpu1cs2n1.VRamSizeGb = 7.9;

gpu1cs2n1.SmNumber = 34;

gpu1cs2n1.PeakMemoryBandwidthGbS = 448.064;

ComputingSystemNode cs2n1{};

cs2n1.AddGpu(gpu1cs2n1);

cs2n1.AddCpu(cpu1cs2n1);

cs2n1.AddRam(ramcs2n1);

cs2.AddNode(cs2n1);

computingSystemCache[cs2.GetId()] = cs2;

computerSystemIds.push\_back(cs2.GetId());

////////////////////////////////

}

};

==================================================

FILE: CpuParams.hpp

PATH: ComputingSystem\CpuParams.hpp

EXTENSION: .hpp

SIZE: 929 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Параметры центрального процессора

struct CpuParams

{

/// @brief УИД CPU

unsigned id{0};

/// @brief Наименование CPU

std::string name{""};

/// @brief Количество поток

unsigned ThreadsNumber{0};

void Print(PrintParams pp)

{

//std::cout << "CpuParams::Print(PrintParams pp)" << std::endl;

std::cout << pp.startMes;

std::cout << "id" << pp.splitterKeyValue << id;

std::cout << pp.splitter;

std::cout << "name" << pp.splitterKeyValue << name;

std::cout << pp.splitter;

std::cout << "ThreadsNumber" << pp.splitterKeyValue << ThreadsNumber;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: GpuParams.hpp

PATH: ComputingSystem\GpuParams.hpp

EXTENSION: .hpp

SIZE: 1493 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Параметры видеоадаптера

struct GpuParams

{

/// @brief УИД GPU

unsigned id{0};

/// @brief Наименование GPU

std::string name{""};

/// @brief Объём видеопамяти, доступной для вычислений, Гбайт

unsigned VRamSizeGb{0};

/// @brief Количество потоковых мультипроцессоров

unsigned SmNumber{0};

/// @brief Пиковая пропускная способность видеопамяти, Гб/с

double PeakMemoryBandwidthGbS{0};

void Print(PrintParams pp)

{

//std::cout << "GpuParams::Print(PrintParams pp)" << std::endl;

std::cout << pp.startMes;

std::cout << "id" << pp.splitterKeyValue << id;

std::cout << pp.splitter;

std::cout << "name" << pp.splitterKeyValue << name;

std::cout << pp.splitter;

std::cout << "VRamSizeGb" << pp.splitterKeyValue << VRamSizeGb;

std::cout << pp.splitter;

std::cout << "SmNumber" << pp.splitterKeyValue << SmNumber;

std::cout << pp.splitter;

std::cout << "PeakMemoryBandwidthGbS" << pp.splitterKeyValue << PeakMemoryBandwidthGbS;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: RamParams.hpp

PATH: ComputingSystem\RamParams.hpp

EXTENSION: .hpp

SIZE: 857 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Параметры центрального процессора

struct RamParams

{

/// @brief Объём доступной для вычислений RAM, Гб

unsigned RamSizeGb{0};

/// @brief Пропускная способность RAM, Гб/c

double RamBandwidthGbS{0};

void Print(PrintParams pp)

{

//std::cout << "CpuParams::Print(PrintParams pp)" << std::endl;

std::cout << pp.startMes;

std::cout << "RamSizeGb" << pp.splitterKeyValue << RamSizeGb;

std::cout << pp.splitter;

std::cout << "RamBandwidthGbS" << pp.splitterKeyValue << RamBandwidthGbS;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: CublasHelper.hpp

PATH: Cuda\CublasHelper.hpp

EXTENSION: .hpp

SIZE: 2755 bytes

----------------------------------------

CONTENT:

#pragma once

#ifndef \_\_NVCC\_\_

struct cublasHandle\_t{};

struct cublasStatus\_t{};

#endif

/// @brief Класс для хранения вспомогательных функций CuBLAS

struct CublasHelper

{

static void CheckCublasStatus(cublasStatus\_t cublasStat, std::string msg = "CUBLAS error")

{

#ifdef \_\_NVCC\_\_

if (cublasStat != CUBLAS\_STATUS\_SUCCESS)

{

std::cout << msg;

throw std::runtime\_error(msg);

}

#else

std::cout << "CublasHelper::CublasDestroy(): CUDA is not supported!" << std::endl;

#endif

}

/// @brief Инициализирует CuBLAS

/// @return

static cublasHandle\_t CublasCreate()

{

#ifdef \_\_NVCC\_\_

cublasHandle\_t cublasH = nullptr;

cublasStatus\_t cublasStat = cublasCreate(&cublasH);

CublasHelper::CheckCublasStatus(cublasStat, "CUBLAS initialization failed\n");

return cublasH;

#else

std::string msg{"CublasHelper::CublasCreate(): CUDA is not supported!"};

std::cout << msg << std::endl;

throw std::runtime\_error(msg);

#endif

}

/// @brief Инициализирует CuBLAS

/// @return

static cublasHandle\_t CublasCreate(int deviceId)

{

#ifdef \_\_NVCC\_\_

cublasHandle\_t cublasH = nullptr;

cublasStatus\_t cublasStat;

if(deviceId == 0)

{

cublasH = CublasCreate();

return cublasH;

}

std::thread th{

[&](){

cudaSetDevice(deviceId);

cublasStat = cublasCreate(&cublasH);

}

};

th.join();

CublasHelper::CheckCublasStatus(cublasStat, "CUBLAS initialization failed\n");

return cublasH;

#else

std::string msg{"CublasHelper::CublasCreate(int deviceId): CUDA is not supported!"};

std::cout << msg << std::endl;

throw std::runtime\_error(msg);

#endif

}

/// @brief Освобождает ресурсы CuBLAS

/// @param cublasH

static void CublasDestroy(cublasHandle\_t cublasH)

{

#ifdef \_\_NVCC\_\_

cublasDestroy(cublasH);

#else

std::cout << "CublasHelper::CublasDestroy(): CUDA is not supported!" << std::endl;

#endif

}

/// @brief Освобождает ресурсы CuBLAS

/// @param cublasHandles

static void CublasDestroy(std::vector<cublasHandle\_t> cublasHandles)

{

for (size\_t i = 0; i < cublasHandles.size(); i++)

{

CublasDestroy(cublasHandles[i]);

}

}

};

==================================================

FILE: CudaDeviceProperties.hpp

PATH: Cuda\CudaDeviceProperties.hpp

EXTENSION: .hpp

SIZE: 3091 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/////////////// CUDA ////////////////

/// @brief Структура для хранения параметров видеокарты

struct CudaDeviceProperties

{

bool IsInitialized = false;

int Major;

int Minor;

std::string Name;

size\_t TotalGlobalMem;

size\_t SharedMemoryPerBlock;

size\_t RegsPerBlock;

int WarpSize;

size\_t MemPitch;

size\_t MaxThreadsPerBlock;

int MultiProcessorCount;

bool DeviceOverlap;

int AsyncEngineCount;// Number of asynchronous engines

size\_t MemoryClockRate;//Memory Clock Rate (KHz)

int MemoryBusWidth;//Memory Bus Width (bits)

double GetPeakMemoryBandwidthGBs()

{

return 2.0\*MemoryClockRate\*(MemoryBusWidth/8)/1.0e6;

}

void Print()

{

if(!IsInitialized)

{

std::cout << "CudaDeviceProperties object is not initialized!" << std::endl;

return;

}

std::cout << "Major revision number: " << Major << std::endl;

std::cout << "Minor revision number: " << Minor << std::endl;

std::cout << "Name: " << Name << std::endl;

std::cout << "Total global memory: " << TotalGlobalMem << std::endl;

std::cout << "Total shared memory per block: " << SharedMemoryPerBlock << std::endl;

std::cout << "Total registers per block: " << RegsPerBlock << std::endl;

std::cout << "Warp size: " << WarpSize << std::endl;

std::cout << "Maximum memory pitch: " << MemPitch << std::endl;

std::cout << "Maximum threads per block: " << MaxThreadsPerBlock << std::endl;

/\*for (int i = 0; i < 3; ++i)

printf("Maximum dimension %d of block: %d\n", i, devProp.maxThreadsDim[i]);

for (int i = 0; i < 3; ++i)

printf("Maximum dimension %d of grid: %d\n", i, devProp.maxGridSize[i]);

printf("Clock rate: %d\n", devProp.clockRate);

printf("Total constant memory: %u\n", devProp.totalConstMem);

printf("Texture alignment: %u\n", devProp.textureAlignment);

printf("Concurrent copy and execution: %s\n", (devProp.deviceOverlap ? "Yes" : "No"));\*/

std::cout << "Number of multiprocessors: " << MultiProcessorCount << std::endl;

//printf("Kernel execution timeout: %s\n", (devProp.kernelExecTimeoutEnabled ? "Yes" : "No"));

std::cout << "Number of asynchronous engines: " << AsyncEngineCount << std::endl;

std::cout << "Memory Clock Rate (KHz): " << MemoryClockRate << std::endl;

std::cout << "Memory Bus Width (bits): " << MemoryBusWidth << std::endl;

std::cout << "Peak Memory Bandwidth (GB/s): " << GetPeakMemoryBandwidthGBs() << std::endl;

}

};

==================================================

FILE: CudaHelper.hpp

PATH: Cuda\CudaHelper.hpp

EXTENSION: .hpp

SIZE: 11647 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "CudaDeviceProperties.hpp"

/////////////// CUDA ////////////////

#ifdef \_\_NVCC\_\_

#define cudaCheckErrors(msg) \

do { \

cudaError\_t \_\_err = cudaGetLastError(); \

if (\_\_err != cudaSuccess) { \

fprintf(stderr, "Fatal error: %s (%s at %s:%d)\n", \

msg, cudaGetErrorString(\_\_err), \

\_\_FILE\_\_, \_\_LINE\_\_); \

fprintf(stderr, "\*\*\* FAILED - THROWING EXCEPTION \*\*\*\n"); \

throw std::runtime\_error(msg); \

} \

} while (0)

#endif

/// @brief Класс для хранения вспомогательных функций Cuda

struct CudaHelper

{

/// @brief Проверка наличия ошибок

/// @return true - ошибки; false - успех, ошибок нет

static bool IsErrors()

{

#ifdef \_\_NVCC\_\_

cudaError\_t err = cudaGetLastError();

if (err != cudaSuccess)

return true;

return false;

#else

return true;

#endif

}

/// @brief Определяет поддержку CUDA

/// @return true - поддерживается, false - не поддерживается

static bool IsCudaSupported()

{

bool isCudaSupported = false;

#ifdef \_\_NVCC\_\_

isCudaSupported = true;

#endif

return isCudaSupported;

}

/// @brief Возвращает количество Cuda-совместимых устройств

/// @return Количество Cuda-совместимых устройств

static int GetCudaDeviceNumber()

{

int devCount = 0;

#ifdef \_\_NVCC\_\_

cudaGetDeviceCount(&devCount);

#endif

return devCount;

}

/// @brief Устанавливает текущее устройство (GPU)

/// @param deviceId Индекс устройства

/// @return true - успех

static bool SetDevice(unsigned deviceId)

{

#ifdef \_\_NVCC\_\_

cudaSetDevice(deviceId);

cudaError\_t err = cudaGetLastError();

if (err != cudaSuccess)

return false;

return true;

#else

return false;

#endif

}

/// @brief Возвращает структуру с параметрами видеокарты

/// @param deviceId Идентификатор Cuda-устройства

/// @return Объект CudaDeviceProperties с параметрами видеокарты (поле IsInitialized = true в случае успеха, иначе IsInitialized = false)

static CudaDeviceProperties GetCudaDeviceProperties(int deviceId = 0)

{

CudaDeviceProperties prop;

#ifdef \_\_NVCC\_\_

// Get device properties

printf("\nCUDA Device #%d\n", deviceId);

cudaDeviceProp devProp;

cudaGetDeviceProperties(&devProp, deviceId);

prop.IsInitialized = true;

prop.Major = devProp.major;

prop.Minor = devProp.minor;

prop.Name = std::string(devProp.name);

prop.TotalGlobalMem = devProp.totalGlobalMem;

prop.SharedMemoryPerBlock = devProp.sharedMemPerBlock;

prop.RegsPerBlock = devProp.regsPerBlock;

prop.WarpSize = devProp.warpSize;

prop.MemPitch = devProp.memPitch;

prop.MaxThreadsPerBlock = devProp.maxThreadsPerBlock;

//for (int i = 0; i < 3; ++i)

// printf("Maximum dimension %d of block: %d\n", i, devProp.maxThreadsDim[i]);

//for (int i = 0; i < 3; ++i)

// printf("Maximum dimension %d of grid: %d\n", i, devProp.maxGridSize[i]);

//printf("Clock rate: %d\n", devProp.clockRate);

//printf("Total constant memory: %u\n", devProp.totalConstMem);

//printf("Texture alignment: %u\n", devProp.textureAlignment);

prop.DeviceOverlap = devProp.deviceOverlap;

prop.MultiProcessorCount = devProp.multiProcessorCount;

//printf("Kernel execution timeout: %s\n", (devProp.kernelExecTimeoutEnabled ? "Yes" : "No"));//\*/

prop.AsyncEngineCount = devProp.asyncEngineCount;

prop.MemoryClockRate = devProp.memoryClockRate;

prop.MemoryBusWidth = devProp.memoryBusWidth;

#endif

return prop;

}

/// @brief Возвращает имя GPU по переданному идентификатору

/// @param deviceId Идентификатор GPU

/// @return Имя GPU

static std::string GetCudaDeviceName(int deviceId = 0)

{

#ifdef \_\_NVCC\_\_

auto cudaDeviceProperties = GetCudaDeviceProperties(deviceId);

return cudaDeviceProperties.Name;

#else

std::cout << "GetCudaDeviceName(): CUDA is not supported!" << std::endl;

return "";

#endif

}

// Print device properties

static void PrintCudaDeviceProperties(int deviceId = 0)

{

#ifdef \_\_NVCC\_\_

// Get device properties

std::cout << "\nCUDA Device #" << deviceId << std::endl;

cudaDeviceProp devProp;

cudaGetDeviceProperties(&devProp, deviceId);

std::cout << "Major revision number: " << devProp.major << std::endl;

std::cout << "Minor revision number: " << devProp.minor << std::endl;

std::cout << "Name: " << devProp.name << std::endl;

std::cout << "Total global memory: " << devProp.totalGlobalMem << std::endl;

std::cout << "Total shared memory per block: " << devProp.sharedMemPerBlock << std::endl;

std::cout << "Total registers per block: " << devProp.regsPerBlock << std::endl;

std::cout << "Warp size: " << devProp.warpSize << std::endl;

std::cout << "Maximum memory pitch: " << devProp.memPitch << std::endl;

std::cout << "Maximum threads per block: " << devProp.maxThreadsPerBlock << std::endl;

/\*for (int i = 0; i < 3; ++i)

printf("Maximum dimension %d of block: %d\n", i, devProp.maxThreadsDim[i]);

for (int i = 0; i < 3; ++i)

printf("Maximum dimension %d of grid: %d\n", i, devProp.maxGridSize[i]);

printf("Clock rate: %d\n", devProp.clockRate);

printf("Total constant memory: %u\n", devProp.totalConstMem);

printf("Texture alignment: %u\n", devProp.textureAlignment);

printf("Concurrent copy and execution: %s\n", (devProp.deviceOverlap ? "Yes" : "No"));\*/

std::cout << "Number of multiprocessors: " << devProp.multiProcessorCount << std::endl;

//printf("Kernel execution timeout: %s\n", (devProp.kernelExecTimeoutEnabled ? "Yes" : "No"));

std::cout << "Number of asynchronous engines: " << devProp.asyncEngineCount << std::endl;

std::cout << "Memory Clock Rate (KHz): " << devProp.memoryClockRate << std::endl;

std::cout << "Memory Bus Width (bits): " << devProp.memoryBusWidth << std::endl;

std::cout << "Peak Memory Bandwidth (GB/s): " << 2.0\*devProp.memoryClockRate\*(devProp.memoryBusWidth/8)/1.0e6 << std::endl;

#else

std::cout << "printDevProp(): CUDA is not supported!" << std::endl;

#endif

}

static void PrintCudaDeviceProperties\_ConsoleUI()

{

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

std::cout << "CudaDeviceNumber: "

<< cudaDeviceNumber

<< std::endl;

std::cout << "Enter deviceId (0..." << cudaDeviceNumber-1 << "): ";

int deviceId;

std::cin >> deviceId;

PrintCudaDeviceProperties(deviceId);

}

static void WriteGpuSpecs(std::ofstream& out)

{

#ifdef \_\_NVCC\_\_

out << "WriteGpuSpecs()" << std::endl;

int nDevices;

cudaGetDeviceCount(&nDevices);

for (int i = 0; i < nDevices; i++)

{

cudaDeviceProp prop;

cudaGetDeviceProperties(&prop, i);

out << "Device Number: " << i << std::endl;

out << " Device name: " << prop.name << std::endl;

out << " Compute capability: " << prop.major << "." << prop.minor << std::endl;

out << " MultiProcessorCount: " << prop.multiProcessorCount << std::endl;

out << " asyncEngineCount: " << prop.asyncEngineCount<< " (Number of asynchronous engines)" << std::endl;

out << " Memory Clock Rate (KHz): " << prop.memoryClockRate << std::endl;

out << " Memory Bus Width (bits): " << prop.memoryBusWidth << std::endl;

out << " Peak Memory Bandwidth (GB/s): "

<< 2.0\*prop.memoryClockRate\*(prop.memoryBusWidth/8)/1.0e6 << std::endl;

}

//#ifdef \_\_NVCC\_\_

#else

out << "printDevProp(): CUDA is not supported!" << std::endl;

#endif

}

/// @brief Записывает параметры видеокарты в текстовый файл gpu-specs.txt

static void WriteGpuSpecsToTxtFile\_ConsoleUI()

{

int cudaDeviceNumber = CudaHelper::GetCudaDeviceNumber();

std::cout << "Cuda devices number: " << cudaDeviceNumber << std::endl;

//CudaHelper::PrintCudaDeviceProperties();

if(cudaDeviceNumber > 0)

{

for(int i = 0; i < cudaDeviceNumber; i++)

{

auto devProps = CudaHelper::GetCudaDeviceProperties();

devProps.Print();

}

std::ofstream f("gpu-specs.txt");

CudaHelper::WriteGpuSpecs(f);

f.close();

}

}

template<typename T>

static void CudaFree(T\* dev\_arr)

{

#ifdef \_\_NVCC\_\_

cudaFree(dev\_arr);

#endif

}

template<typename T>

static void InitByValue(T\* arrayGpu,

unsigned long long length,

T value)

{

#ifdef \_\_NVCC\_\_

unsigned threadsNum = 1024;

unsigned blocksNum = length / threadsNum;

if(length % threadsNum)

blocksNum++;

kernel\_init\_by\_value<<<blocksNum, threadsNum>>>(arrayGpu, length, value);

cudaError\_t cudaResult = cudaGetLastError();

if (cudaResult != cudaSuccess)

{

std::string msg("Could not init GPU array by value: ");

msg += cudaGetErrorString(cudaResult);

throw std::runtime\_error(msg);

}

#else

std::cout << "CUDA is not supported!" << std::endl;

#endif

}

template<typename T, typename S>

static void Multiply(T\* arrayGpu, unsigned long long length, S scalar)

{

#ifdef \_\_NVCC\_\_

unsigned threadsNum = 1024;

unsigned blocksNum = length / threadsNum;

if(length % threadsNum)

blocksNum++;

kernel\_multiply<<<blocksNum, threadsNum>>>(arrayGpu, length, scalar);

cudaDeviceSynchronize();

#else

std::cout << "CUDA is not supported!" << std::endl;

#endif

}

};

/////////////////// CUDA (END) /////////////////////////

==================================================

FILE: kernels.cu

PATH: Cuda\kernels.cu

EXTENSION: .cu

SIZE: 7748 bytes

----------------------------------------

CONTENT:

#pragma once

template <typename T>

\_\_device\_\_ T\* shared\_memory\_proxy()

{

extern \_\_shared\_\_ unsigned char memory[];

return reinterpret\_cast<T\*>(memory);

}

// cuda-ядро для инициализации одномерного массива числом

template<typename T>

\_\_global\_\_

void kernel\_array\_init\_by\_value(T\* data, size\_t indStart, size\_t length, T value)

{

int th\_i = blockIdx.x \* blockDim.x + threadIdx.x;

if (th\_i == 0)

{

//printf("GPU: print\_kernel() vectorGpu.\_size = %d\n", vectorGpu.GetSize());

T\* \_dev\_data\_pointer = data;

unsigned long long indEnd = indStart + length - 1;

//printf("[%d..", (long)indStart);

//printf("%d]: ", (long)indEnd);

for(unsigned long long i = indStart; i <= indEnd; i++)

{

\_dev\_data\_pointer[i] = value;

}

//printf(" initialized by %f\n", value);

}

}

// cuda-ядро для вывода одномерного массива в консоль

template<typename T>

\_\_global\_\_

void kernel\_print(T\* data, size\_t indStart, size\_t length)

{

int th\_i = blockIdx.x \* blockDim.x + threadIdx.x;

if (th\_i == 0)

{

//printf("GPU: print\_kernel() vectorGpu.\_size = %d\n", vectorGpu.GetSize());

T\* \_dev\_data\_pointer = data;

size\_t indEnd = indStart + length - 1;

printf("[%d..", (long)indStart);

printf("%d]: ", (long)indEnd);

for(size\_t i = indStart; i <= indEnd; i++)

{

printf("%f ", \_dev\_data\_pointer[i]);

}

printf("\n");

}

}

template<typename T>

\_\_global\_\_ void kernel\_sum(T\* dev\_arr, size\_t length, T\* dev\_block\_sum)

{

// Массив в распределенной памяти GPU

// для хранения локальных сумм отдельных потоков блока

extern \_\_shared\_\_ T shared\_array[];

//printf("\nkernel\_sum: length = %ld\n", length);

const int tid = threadIdx.x + blockDim.x \* blockIdx.x;

//printf("\nkernel\_sum: tid = %ld\n", tid);

const int number\_of\_threads = gridDim.x \* blockDim.x;

int n\_elem\_per\_thread = length / number\_of\_threads;

#ifdef DEBUG

if(tid == 0)

{

printf("\nkernel\_sum: dev\_arr = %p\n", dev\_arr);

printf("\nkernel\_sum: length = %d\n", length);

printf("\nkernel\_sum: dev\_block\_sum = %p\n", dev\_block\_sum);

printf("\nkernel\_sum: number\_of\_threads = %d\n", number\_of\_threads);

printf("\nkernel\_sum: n\_elem\_per\_thread = %d\n", n\_elem\_per\_thread);

}

#endif

unsigned long long block\_start\_idx = n\_elem\_per\_thread \* blockIdx.x \* blockDim.x;

unsigned long long thread\_start\_idx = block\_start\_idx

+ threadIdx.x \* n\_elem\_per\_thread;

unsigned long long thread\_end\_idx = thread\_start\_idx + n\_elem\_per\_thread;

if(tid == number\_of\_threads - 1)

{

thread\_end\_idx = length;

}

if(thread\_end\_idx > length) thread\_end\_idx = length;

#ifdef DEBUG

printf("\nkernel\_sum: i = %d [%d .. %d]\n", tid, thread\_start\_idx, thread\_end\_idx);

#endif

T localResult{0};

for(size\_t i = thread\_start\_idx; i < thread\_end\_idx; i++)

{

localResult += dev\_arr[i];

}

#ifdef DEBUG

printf("\nkernel\_sum: i = %d, localResult = %f\n", tid, localResult);

#endif

shared\_array[threadIdx.x] = localResult;

\_\_syncthreads();

// Просматриваем содержимое распределяемой памяти

#ifdef DEBUG

if(threadIdx.x == 0)

{

for(int i = 0; i < blockDim.x; i++)

{

printf("\nkernel\_sum: %d (b%d, t%d) shared\_array[%d] = %f\n", tid, blockIdx.x, threadIdx.x, i, shared\_array[i]);

}

}

#endif

if(threadIdx.x == 0)

{

T block\_result = 0;

for(int i = 0; i < blockDim.x; i++)

{

block\_result += shared\_array[i];

#ifdef DEBUG

printf("\nkernel\_sum: shared\_array[%d] = %f\n", tid, shared\_array[i]);

#endif

}

#ifdef DEBUG

printf("\nkernel\_sum: %d, block\_result = %f\n", tid, block\_result);

#endif

dev\_block\_sum[blockIdx.x] = block\_result;

}

}

template<typename T>

\_\_global\_\_ void kernel\_scalar\_product(T\* arrayGpu1, T\* arrayGpu2, size\_t length, T\* blockSumsGpu)

{

// Массив в распределенной памяти GPU

// для хранения локальных сумм отдельных потоков блока

//extern \_\_shared\_\_ T shared\_array[];

auto shared\_array = shared\_memory\_proxy<T>();

//printf("\nkernel\_sum: length = %ld\n", length);

const int tid = threadIdx.x + blockDim.x \* blockIdx.x;

//printf("\nkernel\_sum: tid = %ld\n", tid);

const int number\_of\_threads = gridDim.x \* blockDim.x;

int n\_elem\_per\_thread = length / number\_of\_threads;

#ifdef DEBUG

if(tid == 0)

{

printf("\nkernel\_sum: arrayGpu1 = %p\n", arrayGpu1);

printf("\nkernel\_sum: length = %d\n", length);

printf("\nkernel\_sum: dev\_block\_sum = %p\n", dev\_block\_sum);

printf("\nkernel\_sum: number\_of\_threads = %d\n", number\_of\_threads);

printf("\nkernel\_sum: n\_elem\_per\_thread = %d\n", n\_elem\_per\_thread);

}

#endif

unsigned long long block\_start\_idx = n\_elem\_per\_thread \* blockIdx.x \* blockDim.x;

unsigned long long thread\_start\_idx = block\_start\_idx

+ threadIdx.x \* n\_elem\_per\_thread;

unsigned long long thread\_end\_idx = thread\_start\_idx + n\_elem\_per\_thread;

if(tid == number\_of\_threads - 1)

{

thread\_end\_idx = length;

}

if(thread\_end\_idx > length) thread\_end\_idx = length;

#ifdef DEBUG

printf("\nkernel\_sum: i = %d [%d .. %d]\n", tid, thread\_start\_idx, thread\_end\_idx);

#endif

T localResult{0};

for(size\_t i = thread\_start\_idx; i < thread\_end\_idx; i++)

{

localResult += arrayGpu1[i]\*arrayGpu2[i];

}

#ifdef DEBUG

printf("\nkernel\_sum: i = %d, localResult = %f\n", tid, localResult);

#endif

shared\_array[threadIdx.x] = localResult;

\_\_syncthreads();

// Просматриваем содержимое распределяемой памяти

#ifdef DEBUG

if(threadIdx.x == 0)

{

for(int i = 0; i < blockDim.x; i++)

{

printf("\nkernel\_sum: %d (b%d, t%d) shared\_array[%d] = %f\n", tid, blockIdx.x, threadIdx.x, i, shared\_array[i]);

}

}

#endif

if(threadIdx.x == 0)

{

T block\_result = 0;

for(int i = 0; i < blockDim.x; i++)

{

block\_result += shared\_array[i];

#ifdef DEBUG

printf("\nkernel\_sum: shared\_array[%d] = %f\n", tid, shared\_array[i]);

#endif

}

#ifdef DEBUG

printf("\nkernel\_sum: %d, block\_result = %f\n", tid, block\_result);

#endif

blockSumsGpu[blockIdx.x] = block\_result;

}

}

template<typename T>

\_\_global\_\_ void kernel\_init\_by\_value(T\* arrayGpu, unsigned long long length, T value)

{

auto tid = threadIdx.x + blockIdx.x \* blockDim.x;

if(tid < length)

arrayGpu[tid] = value;

}

template<typename T, typename S>

\_\_global\_\_ void kernel\_multiply(T\* arrayGpu, unsigned long long length, S scalar)

{

auto tid = threadIdx.x + blockIdx.x \* blockDim.x;

if(tid < length)

arrayGpu[tid] \*= scalar;//printf("!");

}

==================================================

FILE: DiffEqFunc2D.hpp

PATH: DifferentialEquations\DiffEqFunc2D.hpp

EXTENSION: .hpp

SIZE: 1277 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <vector>

#include "../CommonHelpers/\_IncludeCommonHelpers.hpp"

/// @brief Функция правой части дифф. уравнения (заданная аналитически)

class DiffEqFunc2D : public IDiffEqFunction

{

/// @brief Указатель на функцию

double (\*f)(double, double);

public:

DiffEqFunc2D(double (\*f)(double, double))

: f(f)

{

}

/// @brief Возвращает размерность объекта функции

Dimension GetDimension() const override

{

return Dimension::D2;

}

double GetValue(double x, double y) const

{

if(!f)

throw std::runtime\_error("f not allowed!");

return f(x, y);

}

/// @brief Возвращает значение функции в точке

double GetValue(std::vector<double> coordinates) const override

{

if(coordinates.size()<2)

return 0;

return GetValue(coordinates[0], coordinates[1]);

}

void Print() const override

{

std::cout << "function address: " << f << std::endl;

std::cout << "dimension: " << GetDimension() << std::endl;

}

};

==================================================

FILE: DiffEqFunc2DPointSources.hpp

PATH: DifferentialEquations\DiffEqFunc2DPointSources.hpp

EXTENSION: .hpp

SIZE: 1688 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

/// @brief Функция правой части дифф. уравнения (точечные источники)

class DiffEqFunc2DPointSources : public IDiffEqFunction

{

// Массив данных в формате

// x1 y1 f1 ...

std::vector<double> data;

public:

/// @brief Возвращает размерность объекта функции

Dimension GetDimension() const override

{

return Dimension::D2;

}

void AddPointSource(double x, double y, double f)

{

data.push\_back(x);

data.push\_back(y);

data.push\_back(f);

}

unsigned int GetNumPointSources() const

{

return data.size() / 3;

}

double GetValue(double x, double y, double eps = 0.00000001) const

{

for(auto i{0ull}; i < data.size(); i += 3)

{

auto \_x = data[i];

auto \_y = data[i + 1];

auto \_f = data[i + 2];

if(std::abs(x - \_x) < eps && std::abs(y - \_y) < eps)

return \_f;

}

return 0.0;

}

/// @brief Возвращает значение функции в точке

double GetValue(std::vector<double> coordinates) const override

{

if(coordinates.size()<2)

return 0;

return GetValue(coordinates[0], coordinates[1]);

}

void Print() const override

{

for(auto i{0ull}; i < data.size(); i += 3)

{

std::cout << data[i] << " "

<< data[i + 1] << " "

<< data[i + 2] << std::endl;

}

}

};

==================================================

FILE: DifferentialEquationsRepository.hpp

PATH: DifferentialEquations\DifferentialEquationsRepository.hpp

EXTENSION: .hpp

SIZE: 613 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "\_IncludeDifferentialEquations.hpp"

/// @brief Репозиторий дифференциальных уравнений

class DifferentialEquationsRepository

{

public:

static Poisson2D GetPoisson2D(PhysicalQuantityEnum physicalQuantity,

IDiffEqFunction\* f)

{

return Poisson2D(physicalQuantity, f);

}

/// @brief Выводит в консоль сведения об объекте

void Print() const

{

std::cout << "DifferentialEquationsRepository" << std::endl;

}

};

==================================================

FILE: DifferentialEquations\_ConsoleUI.hpp

PATH: DifferentialEquations\DifferentialEquations\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 1384 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

struct DifferentialEquations\_ConsoleUI

{

static void Poisson2D\_ConsoleUI()

{

std::cout << "Poisson2D\_ConsoleUI()\n";

/\*auto f = [](double x, double y)

{

if (x < 10 )

return 111.1;

return 0.0;

};\*/

auto f = new DiffEqFunc2DPointSources();

f->AddPointSource(10,20,100);

f->AddPointSource(20,10,80);

auto poisson2D = DifferentialEquationsRepository::GetPoisson2D(PhysicalQuantityEnum::T, f);

poisson2D.Print();

}

static void DiffEqFunc2DPointSources\_ConsoleUI()

{

std::cout << "DiffEqFunc2DPointSources\_ConsoleUI()\n";

auto f = new DiffEqFunc2DPointSources();

f->AddPointSource(10,20,100);

f->AddPointSource(20,10,80);

f->Print();

((IDiffEqFunction\*)f)->Print();

}

static void DiffEqFunc2D\_ConsoleUI()

{

std::cout << "DiffEqFunc2D\_ConsoleUI()\n";

auto f = [](double x, double y)

{

return 10 \* sin(x) + 5 \* cos(y);

};

IDiffEqFunction\* idf = new DiffEqFunc2D(f);

idf->Print();

std::cout << "f(0.1, 0.2) [10 \* sin(x) + 5 \* cos(y)] = "

<< idf->GetValue(std::vector<double>{0.1, 0.2}) << std::endl;

}

};

==================================================

FILE: IDiffEqFunction.hpp

PATH: DifferentialEquations\IDiffEqFunction.hpp

EXTENSION: .hpp

SIZE: 601 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

#include "../CommonHelpers/\_IncludeCommonHelpers.hpp"

// Интерфейс функции правой части

class IDiffEqFunction

{

public:

/// @brief Возвращает значение функции в точке

virtual double GetValue(std::vector<double>) const = 0;

/// @brief Возвращает размерность объекта функции

virtual Dimension GetDimension() const = 0;

/// @brief Выводит в консоль сведения об объекте

virtual void Print() const = 0;

};

==================================================

FILE: Poisson2D.hpp

PATH: DifferentialEquations\Poisson2D.hpp

EXTENSION: .hpp

SIZE: 980 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/\_IncludeCommonHelpers.hpp"

/// @brief Уравнение Пуассона для двумерной области

class Poisson2D

{

// Физическая величина

PhysicalQuantityEnum physicalQuantity;

// Указатель на функцию правой части

// double (\*f)(double, double);

IDiffEqFunction\* f;

public:

Poisson2D(PhysicalQuantityEnum physicalQuantity,

IDiffEqFunction\* f)

: physicalQuantity(physicalQuantity), f(f)

{

}

/// @brief Возвращает размерность объекта функции

Dimension GetDimension() const

{

return Dimension::D2;

}

void Print() const

{

std::cout << "Poisson2D:" << std::endl;

std::cout << "physicalQuantity: " << physicalQuantity << std::endl;

std::cout << "f address: " << f << std::endl;

}

};

==================================================

FILE: \_IncludeDifferentialEquations.hpp

PATH: DifferentialEquations\\_IncludeDifferentialEquations.hpp

EXTENSION: .hpp

SIZE: 242 bytes

----------------------------------------

CONTENT:

#pragma once

#include "IDiffEqFunction.hpp"

#include "DiffEqFunc2D.hpp"

#include "DiffEqFunc2DPointSources.hpp"

#include "Poisson2D.hpp"

#include "DifferentialEquationsRepository.hpp"

#include "DifferentialEquations\_ConsoleUI.hpp"

==================================================

FILE: Function.hpp

PATH: Functions\Function.hpp

EXTENSION: .hpp

SIZE: 5249 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <functional>

#include "../CommonHelpers/PrintParams.hpp"

#include "FunctionDataType.hpp"

#include "FunctionDataTypes.hpp"

#include "../Algorithms/AlgorithmImplementationExecParams.hpp"

#include "../AlgTestingResults/AlgTestingResult.hpp"

class Function

{

// Указатель на функцию, реализующую алгоритм

void\* func = nullptr;

// Тип возвращаемого значения

FunctionDataType returnType;

// Список типов аргументов функции

FunctionDataTypes argumentsTypes;

public:

Function()

{}

template<typename T>

Function(T(\*function)(T\*, size\_t, size\_t))

{

func = (void\*)function;

if(typeid(T) == typeid(float))

{

returnType = FunctionDataType::fdt\_float;

argumentsTypes.Add(FunctionDataType::fdt\_ptr\_float);

}

else if(typeid(T) == typeid(double))

{

returnType = FunctionDataType::fdt\_double;

argumentsTypes.Add(FunctionDataType::fdt\_ptr\_double);

}

else

{

throw std::runtime\_error("Function argument type not realized");

}

argumentsTypes.Add(FunctionDataType::fdt\_ull);

argumentsTypes.Add(FunctionDataType::fdt\_ull);

}

/// @brief Возвращает количество аргументов функции

/// @return

unsigned GetArgumentsTypesCount() const

{

return argumentsTypes.Count();

}

/// @brief Проверка типов аргументов функции

/// @return

bool CheckArgumentsTypes(FunctionDataTypes argsTypes) const

{

if(GetArgumentsTypesCount() != argsTypes.Count())

return false;

for(unsigned i{0}; i < GetArgumentsTypesCount(); i++)

{

if(argumentsTypes[i] != argsTypes[i])

return false;

}

return true;

}

AlgTestingResult Exec(AlgorithmImplementationExecParams params)

{

FunctionArguments functionArguments = params.functionArguments;

if(argumentsTypes.Count()==3)

{

if(argumentsTypes[0] == FunctionDataType::fdt\_ptr\_float

&& argumentsTypes[1] == FunctionDataType::fdt\_ull

&& argumentsTypes[2] == FunctionDataType::fdt\_ull)

{

// Преобразовываем указатель func к нужному виду

// float\* (\*func\_ptr)(size\_t, size\_t);

auto func\_ptr = (float (\*)(float\*, size\_t, size\_t))func;

float\* arg0 = functionArguments.GetArgumentValue<float\*>(0);

std::cout << "arg0 = " << arg0 << std::endl;

size\_t arg1 = functionArguments.GetArgumentValue<size\_t>(1);

std::cout << "arg1 = " << arg1 << std::endl;

size\_t arg2 = functionArguments.GetArgumentValue<size\_t>(2);

std::cout << "arg2 = " << arg2 << std::endl;

std::vector<FuncResult<float>> results;

for(unsigned i{0}; i < params.iterNumber; i++)

{

bool calcStatus = true;

auto start = high\_resolution\_clock::now();

float result\_f = func\_ptr(arg0, arg1, arg2);

auto stop = high\_resolution\_clock::now();

std::cout << "!!! result\_f = " << result\_f << std::endl;

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

FuncResult<float> funcResF(calcStatus, result\_f, t);

results.push\_back(funcResF);

}

//

CalculationStatistics stats{results};

AlgTestingResult algTestingResult;

algTestingResult.calculationStatistics = stats;

algTestingResult.Print();

return algTestingResult;

}

else

{

std::cout << "\n\nNot realized!\n";

throw std::runtime\_error("Error! Function::Exec(...) argumentsTypes.Count()==3 types combination not realized!");

}

}

else

{

std::cout << "\n\nNot realized!\n";

throw std::runtime\_error("Error! Function::Exec(...) argumentsTypes.Count() not realized!");

}

//FuncResult<float> res;

//return res;

}

void Print(PrintParams pp) const

{

std::cout << pp.startMes;

std::cout << "func" << pp.splitterKeyValue << func;

std::cout << pp.splitter;

std::cout << "returnType" << pp.splitterKeyValue << returnType;

std::cout << pp.splitter;

std::cout << "argumentsTypes" << pp.splitterKeyValue << argumentsTypes;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: FunctionArgument.hpp

PATH: Functions\FunctionArgument.hpp

EXTENSION: .hpp

SIZE: 4214 bytes

----------------------------------------

CONTENT:

#pragma once

#include <typeinfo>

#include "../Functions/FunctionDataType.hpp"

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Аргумент функции

struct FunctionArgument

{

FunctionDataType dataType;

void\* data = nullptr;

FunctionArgument()

{

dataType = FunctionDataType::fdt\_void;

Print(PrintParams{});

}

FunctionArgument(float argument)

{

dataType = FunctionDataType::fdt\_float;

auto ptr = new float;

\*ptr = argument;

data = (void\*)ptr;

Print(PrintParams{});

}

template<typename T>

FunctionArgument(T argument)

{

std::cout << "FunctionArgument(T argument): argument = " << argument << std::endl;

if(typeid(T)==typeid(int))

dataType = FunctionDataType::fdt\_int;

else if(typeid(T)==typeid(float))

dataType = FunctionDataType::fdt\_float;

else if(typeid(T)==typeid(float\*))

dataType = FunctionDataType::fdt\_ptr\_float;

else if(typeid(T)==typeid(double))

dataType = FunctionDataType::fdt\_double;

else if(typeid(T)==typeid(double\*))

dataType = FunctionDataType::fdt\_ptr\_double;

else if(typeid(T)==typeid(unsigned long long))

dataType = FunctionDataType::fdt\_ull;

else

{

std::cout << "\nError in FunctionArgument constructor!\n";

//std::cout << "typeid(T): " << typeid(argument).name() << std::cout;

throw std::runtime\_error("Type not recognized!");

}

auto ptr = new T;

\*ptr = argument;

data = (void\*)ptr;

Print(PrintParams{});

}

~FunctionArgument()

{

switch (dataType)

{

case FunctionDataType::fdt\_float:

delete (float\*)data;

break;

default:

break;

}

}

template<typename T>

T GetValue()

{

return \*(T\*)data;

}

void Print(PrintParams pp)

{

std::cout << pp.startMes;

std::cout << dataType;

std::cout << pp.splitter;

std::cout << data;

std::cout << pp.splitter;

switch (dataType)

{

case FunctionDataType::fdt\_void:

std::cout << "void";

break;

case FunctionDataType::fdt\_int:

std::cout << GetValue<int>();

break;

case FunctionDataType::fdt\_float:

std::cout << GetValue<float>();

break;

case FunctionDataType::fdt\_ptr\_float:

std::cout << GetValue<float\*>();

break;

case FunctionDataType::fdt\_double:

std::cout << GetValue<double>();

break;

case FunctionDataType::fdt\_ptr\_double:

std::cout << GetValue<double\*>();

break;

case FunctionDataType::fdt\_ull:

std::cout << GetValue<size\_t>();

break;

default:

std::cout << "\nError in FunctionArgument::Print()! Type not found!\n" << std::endl;

throw std::runtime\_error("Add type in switch of FunctionArgument::Print()");

//break;

}

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

std::ostream& operator<<(std::ostream& os, FunctionArgument arg)

{

switch (arg.dataType)

{

case FunctionDataType::fdt\_void:

os << "void";

break;

case FunctionDataType::fdt\_float:

os << arg.GetValue<float>();

break;

case FunctionDataType::fdt\_ptr\_float:

os << arg.GetValue<float\*>();

break;

case FunctionDataType::fdt\_double:

os << arg.GetValue<double>();

break;

case FunctionDataType::fdt\_ptr\_double:

os << arg.GetValue<double\*>();

break;

case FunctionDataType::fdt\_ull:

os << arg.GetValue<size\_t>();

break;

default:

break;

}

os << "(" << arg.dataType << ")";

return os;

}

==================================================

FILE: FunctionArguments.hpp

PATH: Functions\FunctionArguments.hpp

EXTENSION: .hpp

SIZE: 1195 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

#include "../Functions/FunctionArgument.hpp"

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Аргументы функции

class FunctionArguments

{

std::vector<FunctionArgument> functionArguments;

public:

FunctionDataTypes GetFunctionArgumentsDataTypes() const

{

FunctionDataTypes argDataTypes;

for (size\_t i = 0; i < functionArguments.size(); i++)

{

argDataTypes.Add(functionArguments[i].dataType);

}

return argDataTypes;

}

void Add(FunctionArgument arg)

{

functionArguments.push\_back(arg);

}

FunctionArgument Get(unsigned index) const

{

return functionArguments[index];

}

template<typename T>

T GetArgumentValue(unsigned index)

{

FunctionArgument arg = Get(index);

T argValue = arg.GetValue<T>();

return argValue;

}

void Print(PrintParams pp)

{

for (size\_t i = 0; i < functionArguments.size(); i++)

{

std::cout << i << ": " << functionArguments[i] << "; ";

}

std::cout << std::endl;

}

};

==================================================

FILE: FunctionDataType.hpp

PATH: Functions\FunctionDataType.hpp

EXTENSION: .hpp

SIZE: 1034 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Перечисление типов данных для описания прототипов функций

enum class FunctionDataType

{

fdt\_void,

fdt\_int,

fdt\_float,

fdt\_ptr\_float,

fdt\_double,

fdt\_ptr\_double,

fdt\_ull

};

std::ostream& operator<<(std::ostream& os, FunctionDataType fdt)

{

switch (fdt)

{

case FunctionDataType::fdt\_void:

os << "void";

break;

case FunctionDataType::fdt\_int:

os << "int";

break;

case FunctionDataType::fdt\_float:

os << "float";

break;

case FunctionDataType::fdt\_ptr\_float:

os << "float\*";

break;

case FunctionDataType::fdt\_double:

os << "double";

break;

case FunctionDataType::fdt\_ptr\_double:

os << "double\*";

break;

case FunctionDataType::fdt\_ull:

os << "size\_t";

break;

default:

break;

}

return os;

}

==================================================

FILE: FunctionDataTypes.hpp

PATH: Functions\FunctionDataTypes.hpp

EXTENSION: .hpp

SIZE: 1365 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

#include "../Functions/FunctionDataType.hpp"

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Список типов аргументов функции

class FunctionDataTypes

{

std::vector<FunctionDataType> functionDataTypes;

public:

void Add(FunctionDataType dataType)

{

functionDataTypes.push\_back(dataType);

}

void Print(PrintParams pp = PrintParams{}) const

{

for (size\_t i = 0; i < functionDataTypes.size(); i++)

{

std::cout << i << ": " << functionDataTypes[i] << "; ";

}

std::cout << std::endl;

}

unsigned Count() const

{

return functionDataTypes.size();

}

FunctionDataType operator[](unsigned index) const

{

if(index >= Count())

{

std::cout << "\nError! Index out of range\n";

throw std::runtime\_error("Error in FunctionDataTypes::operator[]. Out of range!");

}

return functionDataTypes[index];

}

};

std::ostream& operator<<(std::ostream& os, FunctionDataTypes fdts)

{

std::cout << "(";

for(unsigned i = 0; i < fdts.Count(); i++)

{

std::cout << fdts[i];

if (i < fdts.Count() - 1)

std::cout << ", ";

}

std::cout << ")";

return os;

}

==================================================

FILE: G2DRectangle.hpp

PATH: Geometry\G2DRectangle.hpp

EXTENSION: .hpp

SIZE: 1199 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Прямоугольник

class G2DRectangle : public IGeometry

{

/// @brief Длина

double Lx;

/// @brief Ширина

double Ly;

public:

/// @brief

/// @param Lx

/// @param Ly

G2DRectangle(double Lx, double Ly)

: Lx(Lx), Ly(Ly)

{}

~G2DRectangle()

{

//std::cout << "G2DRectangle::~G2DRectangle()\n";

}

/// @brief Возвращает размерность объекта геометрии

Dimension GetDimension() const override

{

return Dimension::D2;

}

/// @brief Выводит в консоль сведения об объекте

void Print() const override

{

std::cout << "G2DRectangle object description:" << std::endl;

std::cout << "- type name: " << typeid(this).name() << std::endl;

std::cout << "- address: " << this << std::endl;

std::cout << "- dimension: " << GetDimension() << std::endl;

std::cout << "- Lx: " << Lx << " " << GetMeasurementUnitEnum() << std::endl;

std::cout << "- Ly: " << Ly << " " << GetMeasurementUnitEnum() << std::endl;

}

};

==================================================

FILE: GeometryComposition2D.hpp

PATH: Geometry\GeometryComposition2D.hpp

EXTENSION: .hpp

SIZE: 1047 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

#include "../CommonHelpers/\_IncludeCommonHelpers.hpp"

#include "IGeometryLocation.hpp"

/// @brief Размещение объектов геометрии в двумерном пространстве

class GeometryComposition2D : public IGeometryComposition

{

public:

/// @brief Выводит в консоль сведения об объекте и его значение

void Print() const override

{

std::cout << "GeometryComposition2D::Print()\n";

}

/// @brief Возвращает размерность объектов геометрии

Dimension GetDimension() const override

{

return Dimension::D2;

}

/// @brief Возвращает единицу измерения, используемую для описания объекта геометрии

/// @return MeasurementUnitEnum

MeasurementUnitEnum GetMeasurementUnitEnum() const

{

return MeasurementUnitEnum::Meter;

}

};

==================================================

FILE: GeometryHelper\_ConsoleUI.hpp

PATH: Geometry\GeometryHelper\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 357 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "\_IncludeGeometry.hpp"

struct GeometryHelper\_ConsoleUI

{

static void Geometry2DRectangle\_Console\_UI()

{

std::cout << "Geometry2DRectangle\_Console\_UI()\n";

IGeometry\* g2dRectangle = new G2DRectangle(2, 1);

g2dRectangle->Print();

delete g2dRectangle;

}

};

==================================================

FILE: GeometryLocation2D.hpp

PATH: Geometry\GeometryLocation2D.hpp

EXTENSION: .hpp

SIZE: 520 bytes

----------------------------------------

CONTENT:

#pragma once

#include "IGeometry.hpp"

#include "Location2D.hpp"

#include "IGeometryLocation.hpp"

class GeometryLocation2D : public IGeometryLocation

{

Location2D location;

public:

GeometryLocation2D(IGeometry\* geometry, double x, double y)

{

this->geometry = geometry;

location.x = x;

location.y = y;

}

~GeometryLocation2D()

{

delete this->geometry;

}

ILocation\* GetLocation() override

{

return &location;

}

};

==================================================

FILE: IGeometry.hpp

PATH: Geometry\IGeometry.hpp

EXTENSION: .hpp

SIZE: 876 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Абстрактный класс, моделирующий геометрию расчетной области

class IGeometry

{

public:

/// @brief Виртуальный деструктор

virtual ~IGeometry() = default;

/// @brief Выводит в консоль сведения об объекте и его значение

virtual void Print() const = 0;

/// @brief Возвращает размерность объекта геометрии

virtual Dimension GetDimension() const = 0;

/// @brief Возвращает единицу измерения, используемую для описания объекта геометрии

/// @return MeasurementUnitEnum

MeasurementUnitEnum GetMeasurementUnitEnum() const

{

return MeasurementUnitEnum::Meter;

}

};

==================================================

FILE: IGeometryComposition.hpp

PATH: Geometry\IGeometryComposition.hpp

EXTENSION: .hpp

SIZE: 1788 bytes

----------------------------------------

CONTENT:

#pragma once

#include <vector>

#include "../CommonHelpers/\_IncludeCommonHelpers.hpp"

#include "IGeometryLocation.hpp"

/// @brief Абстрактный класс, моделирующий размещение объектов геометрии в пространстве

class IGeometryComposition

{

std::vector<IGeometryLocation\*> elements;

public:

/// @brief Выводит в консоль сведения об объекте и его значение

virtual void Print() const = 0;

/// @brief Возвращает размерность объекта геометрии

virtual Dimension GetDimension() const = 0;

/// @brief Возвращает единицу измерения, используемую для описания объекта геометрии

/// @return MeasurementUnitEnum

MeasurementUnitEnum GetMeasurementUnitEnum() const

{

return MeasurementUnitEnum::Meter;

}

/// @brief Добавляет объект геометрии в расчетную область по заданнй координате

/// @param geometry

/// @param x

/// @param y

void Add(IGeometry\* geometry, double x, double y)

{

IGeometryLocation\* geometryLocation = new GeometryLocation2D(geometry, x, y);

elements.push\_back(geometryLocation);

}

/// @brief Вывод сведений об объект в консоль

void Print()

{

std::cout << "IGeometryComposition address: " << this << std::endl;

std::cout << "Geometry elements count: " << elements.size() << std::endl;

for(auto i = 0ull; i < elements.size(); i++)

{

elements[i]->Print();

}

}

};

==================================================

FILE: IGeometryLocation.hpp

PATH: Geometry\IGeometryLocation.hpp

EXTENSION: .hpp

SIZE: 588 bytes

----------------------------------------

CONTENT:

#pragma once

class IGeometryLocation

{

public:

IGeometry\* geometry;

//ILocation\* location;

virtual ILocation\* GetLocation() = 0;

/// @brief Вывод сведений об объекте в консоль

void Print()

{

std::cout << "IGeometryLocation address: " << this << std::endl;

std::cout << "IGeometry address: " << geometry << std::endl;

geometry->Print();

ILocation\* location = GetLocation();

std::cout << "ILocation address: " << location << std::endl;

location->Print();

}

};

==================================================

FILE: ILocation.hpp

PATH: Geometry\ILocation.hpp

EXTENSION: .hpp

SIZE: 216 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Интерфейс для объектов, описывающих расположение в пространстве

class ILocation

{

public:

virtual void Print() const = 0;

};

==================================================

FILE: Location2D.hpp

PATH: Geometry\Location2D.hpp

EXTENSION: .hpp

SIZE: 560 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Координаты расположения объекта геометрии в пространстве

class Location2D : public ILocation

{

public:

double x{0};

double y{0};

Location2D()

{}

Location2D(double x, double y)

: x(x), y(y)

{}

void Print() const override

{

std::cout << "Location2D: ";

std::cout << this << "; ";

std::cout << "x = " << x << "; ";

std::cout << "y = " << y << ".";

std::cout << std::endl;

}

};

==================================================

FILE: \_IncludeGeometry.hpp

PATH: Geometry\\_IncludeGeometry.hpp

EXTENSION: .hpp

SIZE: 315 bytes

----------------------------------------

CONTENT:

#pragma once

#include "IGeometry.hpp"

#include "G2DRectangle.hpp"

#include "ILocation.hpp"

#include "Location2D.hpp"

#include "IGeometryLocation.hpp"

#include "GeometryLocation2D.hpp"

#include "IGeometryComposition.hpp"

#include "GeometryComposition2D.hpp"

#include "GeometryHelper\_ConsoleUI.hpp"

==================================================

FILE: BinaryExpression.hpp

PATH: Math\BinaryExpression.hpp

EXTENSION: .hpp

SIZE: 1615 bytes

----------------------------------------

CONTENT:

#pragma once

#include "Expression.hpp"

/// @brief Выражение с двумя операндами

/// @tparam E1

/// @tparam OP +, -, \*, /

/// @tparam E2

template<class E1, class OP, class E2>

struct BinaryExpression :

Expression<BinaryExpression<E1, OP, E2> >

{

BinaryExpression(const Expression<E1> &expr1,

const OP &op,

const Expression<E2> &expr2)

: expr1(expr1.Self()),

op(op),

expr2(expr2.Self())

{}

double operator()(double x) const

{

return op(expr1(x), expr2(x));

}

private:

const E1 expr1;

const OP op;

const E2 expr2;

};

#define DEFINE\_BIN\_OP(oper, OP) \

\

template<class E1, class E2> \

BinaryExpression<E1, std::OP<double>, E2> operator oper \

(const Expression<E1> &expr1, const Expression<E2> &expr2) \

{ \

return BinaryExpression<E1, std::OP<double>, E2> \

(expr1, std::OP<double>(), expr2); \

} \

\

template<class E> \

BinaryExpression<E, std::OP<double>, Constant> operator oper \

(const Expression<E> &expr, double value) \

{ \

return BinaryExpression<E, std::OP<double>, Constant> \

(expr, std::OP<double>(), Constant(value)); \

} \

\

template<class E> \

BinaryExpression<Constant, std::OP<double>, E> operator oper \

(double value, const Expression<E> &expr)\

{ \

return BinaryExpression<Constant, std::OP<double>, E> \

(Constant(value), std::OP<double>(), expr); \

}

DEFINE\_BIN\_OP(+, plus)

DEFINE\_BIN\_OP(-, minus)

DEFINE\_BIN\_OP(\*, multiplies)

DEFINE\_BIN\_OP(/, divides)

==================================================

FILE: Constant.hpp

PATH: Math\Constant.hpp

EXTENSION: .hpp

SIZE: 416 bytes

----------------------------------------

CONTENT:

#pragma once

#include "Expression.hpp"

/// @brief Константа (в выражении)

struct Constant : Expression<Constant>

{

Constant(double value) : value(value){}

double operator()(double x) const

{

return value; // Возвращаемое значение не зависит от значения переменной.

}

private:

double value;

};

==================================================

FILE: Expression.hpp

PATH: Math\Expression.hpp

EXTENSION: .hpp

SIZE: 179 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MathObject.hpp"

/// @brief Маркерный класс "Выражение"

/// @tparam E

template<class E>

struct Expression : MathObject<E>{};

==================================================

FILE: FuncExpression.hpp

PATH: Math\FuncExpression.hpp

EXTENSION: .hpp

SIZE: 772 bytes

----------------------------------------

CONTENT:

#pragma once

#include "Expression.hpp"

/// @brief Функция (в выражении)

/// @tparam E

template<class E>

struct FuncExpression : Expression<FuncExpression<E>>

{

typedef double (\*func\_t)(double);

FuncExpression(const Expression<E> &expr, func\_t func) :

expr(expr.Self()), func(func)

{}

double operator()(double x) const

{

return func(expr(x));

}

private:

const E expr;

func\_t func;

};

#define DEFINE\_FUNC(func) \

\

template<class E> \

FuncExpression<E> func(const Expression<E> &expr) \

{ \

return FuncExpression<E>(expr, std::func); \

}

DEFINE\_FUNC(sin)

DEFINE\_FUNC(cos)

DEFINE\_FUNC(tan)

DEFINE\_FUNC(atan)

DEFINE\_FUNC(exp)

DEFINE\_FUNC(log)

DEFINE\_FUNC(sqrt)

==================================================

FILE: GridContext.hpp

PATH: Math\GridContext.hpp

EXTENSION: .hpp

SIZE: 105 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MathObject.hpp"

template<class GC>

struct GridContext : MathObject<GC>

{};

==================================================

FILE: GridEvaluableObject.hpp

PATH: Math\GridEvaluableObject.hpp

EXTENSION: .hpp

SIZE: 366 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MathObject.hpp"

/// @brief Объект, вычисляемый на сетке. Маркерный класс.

/// @tparam EO Тип вычисляемого объекта

/// @tparam Proxy Прокси-объект (облегченный)

template<class EO, class Proxy = EO>

struct GridEvaluableObject : MathObject<EO, Proxy>{};

==================================================

FILE: GridOperator.hpp

PATH: Math\GridOperator.hpp

EXTENSION: .hpp

SIZE: 818 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MathObject.hpp"

// Предварительное определение GridOperatorEvaluator

template<class GO, class EO>

struct GridOperatorEvaluator;

template<class GO, class Proxy>

struct GridOperator : MathObject<GO, Proxy>

{

template<typename T>

struct GetValueType

{

typedef T type;

};

template<class EO>

GridOperatorEvaluator<GO, EO>

operator()(const GridEvaluableObject<EO, typename EO::proxy\_type>& eobj)

{

return GridOperatorEvaluator<GO, EO>(\*this, eobj);

}

};

#define REIMPLEMENT\_GRID\_EVAL\_OPERATOR() \

template<class EO> \

GridOperatorEvaluator<type, EO> \

operator()(const GridEvaluableObject<EO, typename EO::proxy\_type>& eobj) const \

{ \

return base\_type::operator()(eobj); \

}

==================================================

FILE: GridOperatorEvaluator.hpp

PATH: Math\GridOperatorEvaluator.hpp

EXTENSION: .hpp

SIZE: 1146 bytes

----------------------------------------

CONTENT:

#pragma once

#include "GridEvaluableObject.hpp"

/// @brief Вычислитель

/// @tparam GO

/// @tparam EO

template<class GO, class EO>

struct GridOperatorEvaluator : GridEvaluableObject< GridOperatorEvaluator<GO, EO> >

{

typedef GridOperatorEvaluator type;

typedef GridEvaluableObject<type> base\_type;

typedef typename GO::template get\_value\_type<typename EO::value\_type>::type value\_type;

typedef GridOperator<GO, typename GO::proxy\_type> op\_type;

typedef GridEvaluableObject<EO, typename EO::proxy\_type> eobj\_type;

GridOperatorEvaluator(const op\_type& op, const eobj\_type& eobj)

: op\_proxy(op.get\_proxy()),

eobj\_proxy(eobj.get\_proxy())

{}

template<class GC>

value\_type operator()(size\_t i, size\_t j, size\_t k,

const GridContext<GC>& context) const

{

return op\_proxy(i, j, k, eobj\_proxy, context);

}

value\_type operator()(size\_t i, size\_t j, size\_t k) const

{

return op\_proxy(i, j, k, eobj\_proxy);

}

private:

const typename GO::proxy\_type op\_proxy;

const typename EO::proxy\_type eobj\_proxy;

};

==================================================

FILE: MathHelper.hpp

PATH: Math\MathHelper.hpp

EXTENSION: .hpp

SIZE: 471 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MathObject.hpp"

#include "Expression.hpp"

#include "Constant.hpp"

#include "Variable.hpp"

#include "Negate.hpp"

#include "BinaryExpression.hpp"

#include "FuncExpression.hpp"

#include "GridContext.hpp"

#include "GridEvaluableObject.hpp"

#include "GridOperator.hpp"

#include "GridOperatorEvaluator.hpp"

/// @brief Вспомогательный класс для работы с модулем Math

struct MathHelper

{

};

==================================================

FILE: MathHelper\_ConsoleUI.hpp

PATH: Math\MathHelper\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 1582 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "MathHelper.hpp"

const double PI = 3.141592653589793238463; //value of pi

template<class E>

void f(const Expression<E> &expr0)

{

const E &expr = expr0.Self();

std::cout << "expr(3): " << expr(3) << std::endl;

std::cout << "expr(1.5): " << expr(1.5) << std::endl;

}

/// @brief Вспомогательный класс для работы с модулем Math

struct MathHelper\_ConsoleUI

{

/// @brief Работа с классом MathObject

static void MathObject\_ConsoleUI()

{

std::cout << "--- void MathObject\_ConsoleUI() ---" << std::endl;

/\*struct Scalar : MathObject<Scalar>

{

int value = 10;

};

Scalar a;

std::cout << "Scalar a; a.value: " << a.value << std::endl;

std::cout << "a.Self().value: " << a.Self().value << std::endl;

std::cout << "a.GetProxy().value: " << a.GetProxy().value << std::endl;\*/

Variable x;

std::cout << "f(x) = x" << std::endl;

f(x);

std::cout << "f(x) = x+1.5" << std::endl;

auto expr1 = x + 1.5;

f(expr1);

double res = expr1(10);

std::cout << "double res = expr1(10): " << res << std::endl;

f(sin(x \* x + PI));

auto expr2 = 5 \* cos(-x \* (x + 1));

f(expr2);

//Variable y;

//auto expr\_x\_y = 2\*x - y/3;

//std::cout<< "f(x,y) = 2\*x - y/3; f(10, 30) = " << expr\_x\_y(10, 30) << std::endl;//err

}

};

==================================================

FILE: MathObject.hpp

PATH: Math\MathObject.hpp

EXTENSION: .hpp

SIZE: 524 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Математический объект. Базовый класс.

/// @tparam T

/// @tparam TProxy

template<class T, class TProxy = T>

struct MathObject

{

typedef T FinalType;

typedef TProxy ProxyType;

FinalType& Self()

{

return static\_cast<FinalType&>(\*this);

}

const FinalType& Self() const

{

return static\_cast<const FinalType&>(\*this);

}

ProxyType GetProxy() const

{

return Self();

}

};

==================================================

FILE: Negate.hpp

PATH: Math\Negate.hpp

EXTENSION: .hpp

SIZE: 472 bytes

----------------------------------------

CONTENT:

#pragma once

#include "Expression.hpp"

/// @brief Отрицание (в выражении)

/// @tparam E

template<class E>

struct Negate : Expression<Negate<E> >

{

Negate(const Expression<E> &expr)

: expr(expr.Self()){}

double operator()(double x) const

{

return -expr(x);

}

private:

const E expr;

};

template<class E>

Negate<E> operator-(const Expression<E> &expr)

{

return Negate<E>(expr);

}

==================================================

FILE: Variable.hpp

PATH: Math\Variable.hpp

EXTENSION: .hpp

SIZE: 219 bytes

----------------------------------------

CONTENT:

#pragma once

#include "Expression.hpp"

/// @brief Переменная (в выражении)

struct Variable : Expression<Variable>

{

double operator()(double x) const

{

return x;

}

};

==================================================

FILE: IMatrix.hpp

PATH: Matrices\IMatrix.hpp

EXTENSION: .hpp

SIZE: 2527 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../CommonHelpers/PrintParams.hpp"

#include "MatrixType.hpp"

/// @brief Интерфейс "Матрица"

class IMatrix

{

public:

/// @brief Возвращает тип матрицы

/// @return MatrixType

virtual MatrixType GetMatrixType() const = 0;

/// @brief Возвращает объём занятой оперативной памяти в байтах

/// @return Объём занятой оперативной памяти в байтах

virtual unsigned long long GetSize() const = 0;

/// @brief Возвращает количество строк M

/// @return M - количество строк

virtual unsigned long long GetM() const = 0;

/// @brief Возвращает количество столбцов N

/// @return N - количество строк

virtual unsigned long long GetN() const = 0;

/// @brief Возвращает значение элемента матрицы по указанному индексу

/// @param i Индекс строки

/// @param j Индекс столбца

/// @return Элемент (i, j)

virtual double GetValue(unsigned long long i, unsigned long long j) const = 0;

/// @brief Возвращает значение элемента матрицы по указанному индексу

/// @param i Индекс строки

/// @param j Индекс столбца

/// @return Элемент (i, j)

virtual double operator()(unsigned long long i, unsigned long long j) const = 0;

/// @brief Выводит в консоль сведения о матрице

virtual void Print(PrintParams pp = PrintParams{}) const = 0;

/// @brief Выводит в консоль значения элементов матрицы

virtual void PrintMatrix() const = 0;

/// @brief Выводит в консоль значения элементов матрицы в указанном диапазоне

/// @param ind\_row\_start Индекс стартовой строки

/// @param num\_rows Количество строк

/// @param ind\_col\_start Индекс стартового столбца

/// @param num\_cols Количество столбцов

virtual void PrintMatrix(unsigned long long ind\_row\_start,

unsigned long long num\_rows,

unsigned long long ind\_col\_start,

unsigned long long num\_cols) const = 0;

};

==================================================

FILE: MatricesHelper.hpp

PATH: Matrices\MatricesHelper.hpp

EXTENSION: .hpp

SIZE: 252 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MatrixRamZero.hpp"

#include "MatrixRamE.hpp"

#include "MatrixBlockRamGpus.hpp"

/// @brief Вспомогательный класс для работы с матрицами

class MatricesHelper

{

public:

};

==================================================

FILE: MatricesHelper\_ConsoleUI.hpp

PATH: Matrices\MatricesHelper\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 2648 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MatricesHelper.hpp"

/// @brief Вспомогательный класс для работы с матрицами

class MatricesHelper\_ConsoleUI

{

public:

/// @brief Тестирование класса MatrixRamZero

static void MatrixRamZeroTesting()

{

std::cout << "--- void MatrixRamZeroTesting() ---" << std::endl;

MatrixRamZero z{4, 6};

z.Print();

MatrixRamZero\* z\_ptr = new MatrixRamZero{3, 5};

z\_ptr->Print();

MatrixRam\* mrz\_ptr = new MatrixRamZero{2, 4};

mrz\_ptr->Print();

IMatrix\* iz\_ptr = (IMatrix\*)z\_ptr;

iz\_ptr->Print();

}

/// @brief Тестирование класса MatrixRamE

static void MatrixRamETesting()

{

std::cout << "--- void MatrixRamETesting() ---" << std::endl;

MatrixRamE z{4, 6};

z.Print();

std::cout << "z(0,0) = " << z(0,0) <<std::endl;

std::cout << "z(0,1) = " << z(0,1) <<std::endl;

std::cout << "z(1,0) = " << z(1,0) <<std::endl;

std::cout << "z(1,1) = " << z(1,1) <<std::endl;

MatrixRamE\* z\_ptr = new MatrixRamE{3, 5};

z\_ptr->Print();

MatrixRam\* mrz\_ptr = new MatrixRamE{2, 4};

mrz\_ptr->Print();

IMatrix\* iz\_ptr = (IMatrix\*)z\_ptr;

iz\_ptr->Print();

}

/// @brief Класс MatrixBlockRamGpus - блочная матрица в RAM+GPUs

static void MatrixBlockRamGpus\_ConsoleUI()

{

std::cout << "MatricesHelper\_ConsoleUI::MatrixBlockRamGpus\_ConsoleUI()" << std::endl;

try

{

unsigned mb = 4; //ConsoleHelper::GetUnsignedIntFromUser("Enter number of blocks by row, mb: ");

unsigned nb = 4; //ConsoleHelper::GetUnsignedIntFromUser("Enter number of blocks by column, mb: ");

unsigned block\_size = 5;

// Создаём нулевую блочную матрицу нужной размерности

MatrixBlockRamGpus matrix(mb, nb, block\_size);

matrix.Print();

std::cout << "matrix.AddE(1,2);" << std::endl;

matrix.AddE(1,2);

matrix.Print();

std::cout << "matrix.AddE(1,1);" << std::endl;

matrix.AddE(1,1);

matrix.Print();

std::cout << "matrix.AddE(1,3);" << std::endl;

matrix.AddE(1,3);

matrix.Print();

matrix.PrintMatrix();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

}

};

==================================================

FILE: MatrixBlockRamGpus.hpp

PATH: Matrices\MatrixBlockRamGpus.hpp

EXTENSION: .hpp

SIZE: 5478 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "IMatrix.hpp"

#include "MatrixMap.hpp"

/// @brief Блочная матрица на выч. узле (RAM+GPUs)

class MatrixBlockRamGpus : IMatrix

{

unsigned mb;

unsigned nb;

unsigned n;

// Карта блочной матрицы

MatrixMap matrixMap;

public:

MatrixBlockRamGpus(unsigned mb, unsigned nb, unsigned n)

: mb(mb), nb(nb), n(n)

{

// Добавляем mb строк в карту матрицы

matrixMap.SetRowsNumber(mb);

}

virtual MatrixType GetMatrixType() const override

{

return MatrixType::MatrixBlockRamGpus;

}

/// @brief Возвращает объём занятой оперативной памяти в байтах

/// @return Объём занятой оперативной памяти в байтах

virtual unsigned long long GetSize() const override

{

throw std::runtime\_error("Not realized!");

}

unsigned long long GetM() const override

{

return mb\*n;

}

unsigned long long GetN() const override

{

return nb\*n;

}

/// @brief Возвращает индекс блока

/// @param i Индекс строки элемента

/// @param j Индекс столбца элемента

/// @return

std::pair<unsigned long long, unsigned long long> GetBlockIndexes(unsigned long long i,

unsigned long long j) const

{

return std::pair<unsigned long long, unsigned long long>{i/mb, j/nb};

}

std::pair<unsigned long long, unsigned long long> GetElementInBlockIndexes(unsigned long long i,

unsigned long long j) const

{

return std::pair<unsigned long long, unsigned long long>{i%mb, j%nb};

}

/// @brief Возвращает значение элемента матрицы по указанному индексу

/// @param i Индекс строки

/// @param j Индекс столбца

/// @return Элемент (i, j)

virtual double GetValue(unsigned long long i, unsigned long long j) const override

{

// Вычисляем индекс блока

std::pair<unsigned long long, unsigned long long> indBlock = GetBlockIndexes(i, j);

// Вычисляем индексы элемента внутри блока

std::pair<unsigned long long, unsigned long long> indElementInBlock = GetElementInBlockIndexes(i, j);

MatrixType matrixType = matrixMap.GetMatrixType(indBlock.first, indBlock.second);

switch (matrixType)

{

case MatrixType::Zero:

return 0;

//break;

case MatrixType::E:

if(indElementInBlock.first==indElementInBlock.second)

return 1;

return 0;

//break;

default:

break;

}

//

//

//

//

return 0;

}

virtual double operator()(unsigned long long i, unsigned long long j) const override

{

return GetValue(i, j);

}

void Print(PrintParams pp = PrintParams{}) const override

{

std::cout << "MatrixBlockRamGpus:";

std::cout << pp.startMes;

std::cout << "this"<< pp.splitterKeyValue << this;

std::cout << pp.splitter;

std::cout << "M" << pp.splitterKeyValue << GetM();

std::cout << pp.splitter;

std::cout << "N" << pp.splitterKeyValue << GetN();

std::cout << pp.splitter;

std::cout << "mb" << pp.splitterKeyValue << mb;

std::cout << pp.splitter;

std::cout << "nb" << pp.splitterKeyValue << nb;

std::cout << pp.splitter;

std::cout << "n" << pp.splitterKeyValue << n;

std::cout << pp.splitter;

std::cout << "matrixMap" << pp.splitterKeyValue;

std::cout << "\n";

matrixMap.Print(mb, nb);

std::cout << pp.splitter;

std::cout << "\n";

matrixMap.Print();

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

void PrintMatrix() const override

{

PrintMatrix(0, GetM(), 0, GetN());

}

void PrintMatrix(unsigned long long ind\_row\_start,

unsigned long long num\_rows,

unsigned long long ind\_col\_start,

unsigned long long num\_cols) const override

{

std::cout << "\nPrinting matrix: \n";

std::cout << "Rows: " << ind\_row\_start << "..";

std::cout << (ind\_row\_start + num\_rows - 1) << "; ";

std::cout << "Cols: " << ind\_col\_start << "..";

std::cout << (ind\_col\_start + num\_cols - 1) << "]\n";

for (auto i = ind\_row\_start; i < ind\_row\_start + num\_rows; i++)

{

for (auto j = ind\_col\_start; j < ind\_col\_start + num\_cols; j++)

{

std::cout << GetValue(i, j) << " ";

}

std::cout << "\n";

}

}

/////////

/// @brief Добавляет единичную матрицу по указанным координатам

/// @param bi Индекс строки

/// @param bj Индекс столбца

void AddE(unsigned long long bi, unsigned long long bj)

{

matrixMap.AddE(bi, bj);

}

/////////

};

==================================================

FILE: MatrixDataLocation.hpp

PATH: Matrices\MatrixDataLocation.hpp

EXTENSION: .hpp

SIZE: 1204 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Место хранения матрицы

enum class MatrixDataLocation

{

None = -2, // Данные нигде не хранятся (нулевая, единичная матрицы и пр.)

RAM = -1,

GPU0 = 0, // Видеопамять GPU0

GPU1 = 1, // Видеопамять GPU1

GPU2 = 2, // Видеопамять GPU2

GPU3 = 3 // Видеопамять GPU3

};

std::ostream& operator<<(std::ostream& os, MatrixDataLocation fdt)

{

switch (fdt)

{

case MatrixDataLocation::None:

os << "MatrixDataLocation::None";

break;

case MatrixDataLocation::RAM:

os << "MatrixDataLocation::RAM";

break;

case MatrixDataLocation::GPU0:

os << "MatrixDataLocation::GPU0";

break;

case MatrixDataLocation::GPU1:

os << "MatrixDataLocation::GPU1";

break;

case MatrixDataLocation::GPU2:

os << "MatrixDataLocation::GPU2";

break;

case MatrixDataLocation::GPU3:

os << "MatrixDataLocation::GPU3";

break;

default:

break;

}

return os;

}

==================================================

FILE: MatrixMap.hpp

PATH: Matrices\MatrixMap.hpp

EXTENSION: .hpp

SIZE: 3910 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <vector>

#include "MatrixMapElement.hpp"

/// @brief Карта матрицы

class MatrixMap

{

std::vector<std::vector<MatrixMapElement>> mapElements;

public:

void Print() const

{

for (size\_t ib = 0; ib < mapElements.size(); ib++)

{

std::cout << ib << ": ";

for (size\_t jb = 0; jb < mapElements[ib].size(); jb++)

{

std::cout << "["

<< mapElements[ib][jb].columnIndex

<< "("

<< mapElements[ib][jb].matrixType

<< ")] ";

}

std::cout << std::endl;

}

}

/// @brief Вывод в консоль карты блочной матрицы

void Print(unsigned mb, unsigned nb) const

{

std::cout << "MatrixMap" << std::endl;

for (size\_t ib = 0; ib < mb; ib++)

{

for (size\_t jb = 0; jb < nb; jb++)

{

MatrixType mtype = GetMatrixType(ib, jb);

switch (mtype)

{

case MatrixType::Zero:

std::cout << "Z ";

break;

case MatrixType::E:

std::cout << "E ";

break;

default:

break;

}

}

std::cout << std::endl;

}

}

/// @brief Возвращает тип матрицы, расположенной в блочной матрице по указанному индексу

/// @param ib

/// @param jb

/// @return

MatrixType GetMatrixType(unsigned ib, unsigned jb) const

{

// Проверка

if(ib >= mapElements.size())

{

std::cout << "ib: " << ib;

std::cout << "\nmapElements.size(): " << mapElements.size();

throw std::runtime\_error("Error in mapElements size!");

}

// Выбираем строку блочной матрицы

auto& blockMatrixRow = mapElements[ib];

// Перебираем блоки в текущей строке

for (size\_t j = 0; j < blockMatrixRow.size(); j++)

{

auto curBlock = blockMatrixRow[j];

if(jb < curBlock.columnIndex)

return MatrixType::Zero;

if(jb == curBlock.columnIndex)

return curBlock.matrixType;

}

return MatrixType::Zero;

}

void SetRowsNumber(unsigned mb)

{

mapElements.clear();

for (size\_t i = 0; i < mb; i++)

{

mapElements.push\_back(std::vector<MatrixMapElement>{});

}

}

/// @brief Добавляет единичную матрицу по указанным координатам

/// @param bi Индекс строки

/// @param bj Индекс столбца

void AddE(unsigned ib, unsigned jb)

{

auto insertingElement = MatrixMapElement{jb, MatrixType::E};

// Выбираем строку блочной матрицы

auto& blockMatrixRow = mapElements[ib];

// Перебираем блоки в текущей строке

for (size\_t j = 0; j < blockMatrixRow.size(); j++)

{

auto curBlock = blockMatrixRow[j];

//curCol = curBlock.columnIndex;

if(jb < curBlock.columnIndex)

{

blockMatrixRow.insert(blockMatrixRow.begin()+j,insertingElement);

return;

}

}

blockMatrixRow.push\_back(insertingElement);

}

};

==================================================

FILE: MatrixMapElement.hpp

PATH: Matrices\MatrixMapElement.hpp

EXTENSION: .hpp

SIZE: 825 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "MatrixType.hpp"

#include "MatrixDataLocation.hpp"

#include "IMatrix.hpp"

/// @brief Элемент карты блочной матрицы

struct MatrixMapElement

{

// Индекс столбца блочной матрицы

unsigned columnIndex = 0;

// Тип матрицы

MatrixType matrixType = MatrixType::Zero;

// Место хранения данных матрицы

MatrixDataLocation matrixDataLocation = MatrixDataLocation::None;

// Указатель на объект матрицы

IMatrix\* matrixPtr = nullptr;

MatrixMapElement()

{}

MatrixMapElement(unsigned columnIndex,

MatrixType matrixType)

: columnIndex(columnIndex),

matrixType(matrixType)

{}

};

==================================================

FILE: MatrixRam.hpp

PATH: Matrices\MatrixRam.hpp

EXTENSION: .hpp

SIZE: 2269 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../CommonHelpers/PrintParams.hpp"

#include "IMatrix.hpp"

/// @brief Класс "Матрица в RAM"

class MatrixRam : IMatrix

{

public:

const unsigned long long M{};

const unsigned long long N{};

MatrixRam()

{ }

MatrixRam(unsigned long long M,

unsigned long long N) :

M(M), N(N)

{

}

/// @brief Возвращает объём занятой оперативной памяти в байтах

/// @return Объём занятой оперативной памяти в байтах

unsigned long long GetSize() const override

{

long long size = sizeof(this);

return size;

}

void Print(PrintParams pp = PrintParams{}) const override

{

std::cout << "void MatrixRam::Print() const " << std::endl;

std::cout << pp.startMes;

std::cout << "M" << pp.splitterKeyValue << GetM();

std::cout << pp.splitter;

std::cout << "N" << pp.splitterKeyValue << GetN();

std::cout << pp.splitter;

std::cout << "GetMatrixType()" << pp.splitterKeyValue << GetMatrixType();

std::cout << pp.splitter;

std::cout << "GetSize()" << pp.splitterKeyValue << GetSize();

std::cout << pp.endMes;

PrintMatrix();

if(pp.isEndl)

std::cout << std::endl;

}

void PrintMatrix() const override

{

PrintMatrix(0, GetM(), 0, GetN());

}

void PrintMatrix(unsigned long long ind\_row\_start,

unsigned long long num\_rows,

unsigned long long ind\_col\_start,

unsigned long long num\_cols) const override

{

std::cout << "\nPrinting matrix: \n";

std::cout << "Rows: " << ind\_row\_start << "..";

std::cout << (ind\_row\_start + num\_rows - 1) << "; ";

std::cout << "Cols: " << ind\_col\_start << "..";

std::cout << (ind\_col\_start + num\_cols - 1) << "]\n";

for (auto i = ind\_row\_start; i < ind\_row\_start + num\_rows; i++)

{

for (auto j = ind\_col\_start; j < ind\_col\_start + num\_cols; j++)

{

std::cout << GetValue(i, j) << " ";

}

std::cout << "\n";

}

}

};

==================================================

FILE: MatrixRamE.hpp

PATH: Matrices\MatrixRamE.hpp

EXTENSION: .hpp

SIZE: 1494 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MatrixRam.hpp"

/// @brief Единичная матрица

class MatrixRamE : public MatrixRam

{

public:

MatrixRamE(unsigned long long M, unsigned long long N)

: MatrixRam(M, N)

{

}

virtual MatrixType GetMatrixType() const override

{

return MatrixType::E;

}

unsigned long long GetM() const override

{

return M;

}

unsigned long long GetN() const override

{

return N;

}

/// @brief Возвращает значение элемента матрицы по указанному индексу

/// @param i Индекс строки

/// @param j Индекс столбца

/// @return Элемент (i, j)

virtual double GetValue(unsigned long long i, unsigned long long j) const override

{

if(i >= M || j >= N)

{

std::cout << "!!!!! i: " << i << "; j: " << j << std::endl;

throw std::runtime\_error("ZeroMatrix::GetValue() error!");

}

if(i==j)

return 1;

return 0;

}

virtual double operator()(unsigned long long i, unsigned long long j) const override

{

if(i >= M || j >= N)

{

std::cout << "!!!!! i: " << i << "; j: " << j << std::endl;

throw std::runtime\_error("ZeroMatrix::operator() error!");

}

if(i==j)

return 1;

return 0;

}

};

==================================================

FILE: MatrixRamZero.hpp

PATH: Matrices\MatrixRamZero.hpp

EXTENSION: .hpp

SIZE: 1414 bytes

----------------------------------------

CONTENT:

#pragma once

#include "MatrixRam.hpp"

/// @brief Нулевая матрица

class MatrixRamZero : public MatrixRam

{

public:

MatrixRamZero(unsigned long long M, unsigned long long N)

: MatrixRam(M, N)

{

}

unsigned long long GetM() const override

{

return M;

}

unsigned long long GetN() const override

{

return N;

}

/// @brief Возвращает значение элемента матрицы по указанному индексу

/// @param i Индекс строки

/// @param j Индекс столбца

/// @return Элемент (i, j)

virtual double GetValue(unsigned long long i, unsigned long long j) const override

{

if(i >= M || j >= N)

{

std::cout << "!!!!! i: " << i << "; j: " << j << std::endl;

throw std::runtime\_error("ZeroMatrix::GetValue() error!");

}

return 0;

}

virtual double operator()(unsigned long long i, unsigned long long j) const override

{

if(i >= M || j >= N)

{

std::cout << "!!!!! i: " << i << "; j: " << j << std::endl;

throw std::runtime\_error("ZeroMatrix::operator() error!");

}

return 0;

}

virtual MatrixType GetMatrixType() const override

{ return MatrixType::Zero; }

};

==================================================

FILE: MatrixType.hpp

PATH: Matrices\MatrixType.hpp

EXTENSION: .hpp

SIZE: 952 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Тип матрицы

enum class MatrixType

{

Zero, // Нулевая матрица

E, // Единичная матрица

Diagonal, // Диагональная матрица

MatrixBlockRamGpus // Блочная матрица с размещением данных в RAM и нескольких GPU на одном вычислительном узле

};

std::ostream& operator<<(std::ostream& os, MatrixType fdt)

{

switch (fdt)

{

case MatrixType::Zero:

os << "MatrixType::Zero";

break;

case MatrixType::E:

os << "MatrixType::E";

break;

case MatrixType::Diagonal:

os << "MatrixType::Diagonal";

break;

case MatrixType::MatrixBlockRamGpus:

os << "MatrixType::MatrixBlockRamGpus";

break;

default:

break;

}

return os;

}

==================================================

FILE: MainMenu.hpp

PATH: Menu\MainMenu.hpp

EXTENSION: .hpp

SIZE: 11789 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Главное меню приложения

class MainMenu

{

// Список команд меню

std::vector<MenuCommandItem> menuCommands;

//MenuCommand command = MenuCommand::None;// Выбранная команда меню

MenuCommandItem command;// Выбранная команда меню

/// @brief Распознаёт команду

/// @param commandString

/// @return

bool RecognizeCommand(std::string commandString)

{

command.Reset();

for(auto& menuItem : menuCommands)

{

if(menuItem.CheckKey(commandString))

{

command = menuItem;

return true;

}

}

return false;

}

/// @brief Исполняет команду

void RunCommand()

{

if(command.func == nullptr)

return;

std::cout << "----- Starting: " << command.desc << "-----------" << std::endl;

command.func();

std::cout << "-------------------------------------" << std::endl;

}

/// @brief Выводит в консоль справочную информацию

void PrintHelp()

{

std::cout << "----- Command list -----" << std::endl;

for(auto& menuItem : menuCommands)

{

for(auto& key : menuItem.keys)

{

std::cout << key << " ";

}

std::cout << "\t" << menuItem.desc << std::endl;

}

}

public:

MainMenu()

{

// Инициализация меню

MenuCommandItem item1;

item1.comm = MenuCommand::Help;

item1.keys = {std::to\_string((int)MenuCommand::Help),"?","h","help"};

item1.func = nullptr;

item1.desc = "Print help";

menuCommands.push\_back(item1);

MenuCommandItem item2;

item2.comm = MenuCommand::Exit;

item2.keys = {std::to\_string((int)MenuCommand::Exit),"q","exit"};

item2.func = nullptr;

item2.desc = "Exit from menu";

menuCommands.push\_back(item2);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::PrintLibSupport,

{std::to\_string((int)MenuCommand::PrintLibSupport),"libs"},

MenuFunctions::PrintLibSupport,

"Print supported libs (OpenMP, Cuda etc.)"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::CudaHelper,

{std::to\_string((int)MenuCommand::CudaHelper),"CudaHelper"},

MenuFunctions::CudaHelper,

"Class CudaHelper"

}

);

/\*menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::PrintGpuParameters,

{std::to\_string((int)MenuCommand::PrintGpuParameters),"gpu"},

MenuFunctions::PrintGpuParameters,

"Print default (0) Cuda-device properties"

}

);\*/

/\*menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::WriteGpuSpecsToTxtFile,

{std::to\_string((int)MenuCommand::WriteGpuSpecsToTxtFile),"gpu"},

MenuFunctions::WriteGpuSpecsToTxtFile,

"Write GPU specification to txt file gpu-specs.txt"

}

);\*/

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::ArrayHelper,

{std::to\_string((int)MenuCommand::ArrayHelper),"ArrayHelper"},

MenuFunctions::ArrayHelper,

"Class ArrayHelper"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::ArrayPerfTestHelper,

{std::to\_string((int)MenuCommand::ArrayPerfTestHelper),"ArrayPerfTestHelper"},

MenuFunctions::ArrayPerfTestHelper,

"Class ArrayPerfTestHelper"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::Testing\_TestVectorGpu,

{std::to\_string((int)MenuCommand::Testing\_TestVectorGpu),"test-vec-gpu"},

MenuFunctions::Testing\_TestVectorGpu,

"Testing VectorGpu class"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::VectorsHelper\_ConsoleUI,

{std::to\_string((int)MenuCommand::VectorsHelper\_ConsoleUI),"vectors"},

MenuFunctions::VectorsHelper\_ConsoleUI,

"Vectors module (VectorsHelper\_ConsoleUI)"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::Testing\_Matrices,

{std::to\_string((int)MenuCommand::Testing\_Matrices),"matrices"},

MenuFunctions::MatricesHelper\_ConsoleUI,

"MatricesHelper\_ConsoleUI"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::Math,

{std::to\_string((int)MenuCommand::Math),"math"},

MenuFunctions::MathHelper\_ConsoleUI,

"Math module (MathHelper\_ConsoleUI)"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::Testing\_TestSum,

{std::to\_string((int)MenuCommand::Testing\_TestSum),"test-sum"},

MenuFunctions::Testing\_TestSum,

"Testing sum functions"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::Application\_Config,

{std::to\_string((int)MenuCommand::Application\_Config),"app-conf"},

nullptr,

"Application configuration"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::ComputingSystemRepository\_Config,

{std::to\_string((int)MenuCommand::ComputingSystemRepository\_Config),"cs-repo-conf"},

nullptr,

"Computing system repository configuration"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::AlgTestingResultRepository\_Config,

{std::to\_string((int)MenuCommand::AlgTestingResultRepository\_Config),"algtr-repo-conf"},

nullptr,

"AlgTestingResultRepository configuration"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::Testing\_FileSystemHelper,

{std::to\_string((int)MenuCommand::Testing\_FileSystemHelper),"fs-hlp"},

MenuFunctions::Testing\_FileSystemHelper,

"Testing FileSystemHelper"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::AlgorithmRepository,

{std::to\_string((int)MenuCommand::AlgorithmRepository),"alg-repo"},

nullptr,

"Testing AlgorithmRepository"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::AlgorithmImplementationRepository,

{std::to\_string((int)MenuCommand::AlgorithmImplementationRepository),"alg-impl-repo"},

nullptr,

"AlgorithmImplementationRepository"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::AlgorithmImplementationExecutor,

{std::to\_string((int)MenuCommand::AlgorithmImplementationExecutor),"alg-impl-exec"},

nullptr,

"AlgorithmImplementationExecutor"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::GeometryHelper\_ConsoleUI,

{std::to\_string((int)MenuCommand::GeometryHelper\_ConsoleUI),"geometry"},

MenuFunctions::GeometryHelper\_ConsoleUI,

"Geometry module (GeometryHelper\_ConsoleUI)"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::DifferentialEquations\_ConsoleUI,

{std::to\_string((int)MenuCommand::DifferentialEquations\_ConsoleUI),"diff-eq"},

MenuFunctions::DifferentialEquations\_ConsoleUI,

"DifferentialEquations module (DifferentialEquations\_ConsoleUI)"

}

);

menuCommands.push\_back(

MenuCommandItem

{

MenuCommand::ModelProblems\_ConsoleUI,

{std::to\_string((int)MenuCommand::ModelProblems\_ConsoleUI),"mod-prob"},

MenuFunctions::ModelProblems\_ConsoleUI,

"ModelProblems module (ModelProblems\_ConsoleUI)"

}

);

}

/// @brief Запуск главного меню

void Start(AppConfig& appConfig,

ComputingSystemRepository& compSysRepo,

AlgorithmRepository& algorithmRepository,

AlgorithmImplementationRepository& algorithmImplementationRepo,

AlgTestingResultRepository& algTestingResultRepo,

AlgorithmImplementationExecutor& algorithmImplementationExecutor)

{

std::cout << "--- Main Menu ('1', '?', 'h' or 'help' for print help)---" << std::endl;

std::string commandString;// Введённая пользователем команда

while(command.comm != MenuCommand::Exit)

{

std::cout << "> ";

std::cin >> commandString;

if ( !RecognizeCommand(commandString))// Распознаём команду

{

std::cout << "Error! Command not recognized! Please enter command again. '?' or 'help' for print help." << std::endl;

continue;

}

switch (command.comm)

{

case MenuCommand::Help:

PrintHelp();

break;

case MenuCommand::Application\_Config:

MenuFunctions::Application\_Config(appConfig);

break;

case MenuCommand::ComputingSystemRepository\_Config:

MenuFunctions::ComputingSystemRepository\_Config(compSysRepo);

break;

case MenuCommand::AlgTestingResultRepository\_Config:

MenuFunctions::AlgTestingResultRepository\_Config(algTestingResultRepo);

case MenuCommand::AlgorithmRepository:

MenuFunctions::AlgorithmRepository(algorithmRepository);

case MenuCommand::AlgorithmImplementationRepository:

MenuFunctions::Menu\_AlgorithmImplementationRepository(algorithmImplementationRepo);

case MenuCommand::AlgorithmImplementationExecutor:

MenuFunctions::Menu\_AlgorithmImplementationExecutor(algorithmImplementationExecutor);

default:

RunCommand();

break;

}

}

std::cout << "--- Good bye! ---" << std::endl;

}

};

==================================================

FILE: MenuCommand.hpp

PATH: Menu\MenuCommand.hpp

EXTENSION: .hpp

SIZE: 2436 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Перечисление команд меню

enum class MenuCommand

{

None, // Не выбрано

Help, // Вывод в консоль справки

Exit, // Выход из меню

PrintLibSupport, // Вывод в консоль списка поддерживаемых библиотек

CudaHelper, // Работа с классом CudaHelper

//PrintGpuParameters, // Вывод в консоль параметров GPU

//WriteGpuSpecsToTxtFile, // Записывает параметры видеокарт в текстовый файл gpu-specs.txt

ArrayHelper, // Работа с классом ArrayHelper

ArrayPerfTestHelper, // Работа с классом ArrayPerfTestHelper

Testing\_TestVectorGpu, // Тестирование класса VectorGpu

VectorsHelper\_ConsoleUI, // Работа с модулем Vectors

Testing\_Matrices, // Тестирование классов матриц

Math, // Работа с модулем Math

Testing\_TestSum, // Тестирование функций суммирования

Application\_Config, // Конфигурация приложения

ComputingSystemRepository\_Config, // Конфигурирование хранилища сведений о вычислительных системах

AlgTestingResultRepository\_Config, // Работа с хранилищем результатов тестовых запусков

Testing\_FileSystemHelper, // Тестирование вспомогательного класса для работы с файловой системой

AlgorithmRepository, // Тестирование репозитория алгоритмов

AlgorithmImplementationRepository, // Работа с репозиторием реализаций алгоритмов

AlgorithmImplementationExecutor,// Запуск различных реализаций алгоритмов

GeometryHelper\_ConsoleUI, // Работа с модулем Geometry

DifferentialEquations\_ConsoleUI, // Работа с модулем DifferentialEquations

ModelProblems\_ConsoleUI // Работа с модулем ModelProblems

};

==================================================

FILE: MenuCommandItem.hpp

PATH: Menu\MenuCommandItem.hpp

EXTENSION: .hpp

SIZE: 1018 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Элемент меню

struct MenuCommandItem

{

MenuCommand comm = MenuCommand::None;// Команда

std::vector<std::string> keys;// Список ключей

std::function<void()> func;// Вызываемая функция

std::string desc;// Описание команды

MenuCommandItem()

{}

MenuCommandItem(MenuCommand comm,

std::vector<std::string> keys,

std::function<void()> func,

std::string desc)

: comm(comm), keys(keys), func(func), desc(desc)

{}

void Reset()

{

comm = MenuCommand::None;

keys = {};

func = nullptr;

desc = "Command not choosed!";

}

bool CheckKey(const std::string& str)

{

bool isKey = false;

for(auto& key : keys)

{

if(key == str)

{

isKey = true;

break;

}

}

return isKey;

}

};

==================================================

FILE: MenuFunctions.hpp

PATH: Menu\MenuFunctions.hpp

EXTENSION: .hpp

SIZE: 57278 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../GlobalTestFunctions.hpp"

#include "../Algorithms/AlgorithmImplementationExecutor.hpp"

#include "../Algorithms/AlgorithmImplementationExecutorHelper.hpp"

#include "../Matrices/MatricesHelper.hpp"

#include "../Vectors/VectorsHelper\_ConsoleUI.hpp"

/// @brief Функции меню

struct MenuFunctions

{

/// @brief Выводит параметры GPU

/\*static void PrintGpuParameters()

{

CudaHelper::PrintCudaDeviceProperties();

}\*/

/// @brief Выводит в консоль список поддерживаемых библиотек

static void PrintLibSupport()

{

// Определяем перечень поддерживаемых библиотек

LibSupport support;

support.Print();// Выводим список поддерживаемых библиотек

}

/// @brief Работа с классом CudaHelper

static void CudaHelper()

{

std::cout << "----- CudaHelper -----\n"

<< "1 Back to main menu\n"

<< "2 IsCudaSupported()\n"

<< "3 GetCudaDeviceNumber()\n"

<< "4 GetCudaDeviceProperties(int deviceId = 0)\n"

<< "5 WriteGpuSpecsToTxtFile\_ConsoleUI()\n"

<< "6 -\n"

<< "7 -\n"

<< "8 -\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 IsCudaSupported()\n";

std::cout << std::boolalpha

<< CudaHelper::IsCudaSupported()

<< std::endl;

break;

case 3:

std::cout << "Command: 3 GetCudaDeviceNumber()\n";

std::cout << std::boolalpha

<< CudaHelper::GetCudaDeviceNumber()

<< std::endl;

break;

case 4:

std::cout << "Command: 4 GetCudaDeviceProperties(int deviceId = 0)\n";

CudaHelper::PrintCudaDeviceProperties\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 WriteGpuSpecsToTxtFile\_ConsoleUI()\n";

CudaHelper::WriteGpuSpecsToTxtFile\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 -\n";

//FileSystemHelper::IsDirExists();

break;

case 7:

std::cout << "Command: 7 -\n";

//FileSystemHelper::RemoveFile();

break;

case 8:

std::cout << "Command: 8 -\n";

//FileSystemHelper::RemoveDir();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Тестирование функций класса ArrayHelper

static void ArrayHelper()

{

std::cout << "----- ArrayHelper -----\n"

<< "1 Back to main menu\n"

<< "2 ArrayHelper::CreateArrayRamPinned\_ConsoleUI\n"

<< "3 ArrayHelper::SumOpenMP\n"

<< "4 ArrayHelper::SumCudaMultiGpu\n"

<< "5 ArrayHelper::SumCublas\n"

<< "6 ArrayHelper::SumCublasMultiGpu\n"

<< "7 ArrayHelper::CopyRamToGpu\n"

<< "8 ArrayHelper::CopyGpuToRam\n"

<< "9 ArrayHelper::ScalarProductRamSeq\n"

<< "10 ArrayHelper::ScalarProductRamParThread\n"

<< "11 ArrayHelper::ScalarProductRamParOpenMP\n"

<< "12 ArrayHelper::ScalarProductGpuParCuda\n"

<< "13 ArrayHelper::ScalarProductMultiGpuParCuda\n"

<< "14 ArrayHelper::ScalarProductGpuCublas\n"

<< "15 ArrayHelper::ScalarProductMultiGpuCublas\n"

<< "16 ArrayHelper::ScalarProductRamOpenBlas\_ConsoleUI\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 ArrayHelper::CreateArrayRamPinned\_ConsoleUI()\n";

ArrayHelper\_ConsoleUI::CreateArrayRamPinned\_ConsoleUI();

break;

case 3:

std::cout << "Command: 3 ArrayHelper::SumOpenMP()\n";

ArrayHelper\_ConsoleUI::SumOpenMP\_ConsoleUI();

break;

case 4:

std::cout << "Command: 4 ArrayHelper::SumCudaMultiGpu()\n";

ArrayHelper\_ConsoleUI::SumCudaMultiGpu\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 ArrayHelper::SumCublas()\n";

ArrayHelper\_ConsoleUI::SumCublas\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 ArrayHelper::SumCublasMultiGpu()\n";

ArrayHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 ArrayHelper::CopyRamToGpu()\n";

ArrayHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 ArrayHelper::CopyGpuToRam()\n";

ArrayHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 ArrayHelper::ScalarProductRamSeq\n";

ArrayHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 ArrayHelper::ScalarProductRamParThread\n";

ArrayHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 ArrayHelper::ScalarProductRamParOpenMP\_ConsoleUI\n";

ArrayHelper\_ConsoleUI::ScalarProductRamParOpenMP\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 ArrayHelper::ScalarProductGpuParCuda\n";

ArrayHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 ArrayHelper::ScalarProductMultiGpuParCuda\n";

ArrayHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 ArrayHelper::ScalarProductGpuCublas\n";

ArrayHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 15:

std::cout << "Command: 15 ArrayHelper::ScalarProductMultiGpuCublas\n";

ArrayHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

case 16:

std::cout << "Command: 16 ArrayHelper::ScalarProductRamOpenBlas\_ConsoleUI\n";

ArrayHelper\_ConsoleUI::ScalarProductRamOpenBlas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Тестирование функций класса ArrayPerfTestHelper

static void ArrayPerfTestHelper()

{

std::cout << "----- ArrayPerfTestHelper -----\n"

<< "1 Back to main menu\n"

<< "2 ArrayPerfTestHelper::SumOpenMP\n"

//<< "3 ArrayPerfTestHelper::SumCudaMultiGpu\n"

<< "4 ArrayPerfTestHelper::SumCublas\n"

//<< "5 ArrayPerfTestHelper::SumCublasMultiGpu\n"

//<< "6 ArrayPerfTestHelper::CopyRamToGpu\n"

//<< "7 ArrayPerfTestHelper::CopyGpuToRam\n"

//<< "8 ArrayPerfTestHelper::ScalarProductRamSeq\n"

//<< "9 ArrayPerfTestHelper::ScalarProductRamParThread\n"

//<< "10 ArrayPerfTestHelper::ScalarProductRamParOpenMP\n"

//<< "11 ArrayPerfTestHelper::ScalarProductGpuParCuda\n"

//<< "12 ArrayPerfTestHelper::ScalarProductMultiGpuParCuda\n"

//<< "13 ArrayPerfTestHelper::ScalarProductGpuCublas\n"

//<< "14 ArrayPerfTestHelper::ScalarProductMultiGpuCublas\n";

;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 ArrayPerfTestHelper::SumOpenMP()\n";

ArrayPerfTestHelper\_ConsoleUI::SumOpenMP\_ConsoleUI();

break;

case 3:

std::cout << "Command: 3 ArrayPerfTestHelper::SumCudaMultiGpu()\n";

//ArrayPerfTestHelper\_ConsoleUI::SumCudaMultiGpu\_ConsoleUI();

break;

case 4:

std::cout << "Command: 4 ArrayPerfTestHelper::SumCublas()\n";

ArrayPerfTestHelper\_ConsoleUI::SumCublas\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 ArrayPerfTestHelper::SumCublasMultiGpu()\n";

//ArrayPerfTestHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 ArrayPerfTestHelper::CopyRamToGpu()\n";

//ArrayPerfTestHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 ArrayPerfTestHelper::CopyGpuToRam()\n";

//ArrayPerfTestHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 ArrayPerfTestHelper::ScalarProductRamSeq\n";

//ArrayPerfTestHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 ArrayPerfTestHelper::ScalarProductRamParThread\n";

//ArrayPerfTestHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 ArrayPerfTestHelper::ScalarProductRamParOpenMP\_ConsoleUI\n";

//ArrayPerfTestHelper\_ConsoleUI::ScalarProductRamParOpenMP\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 ArrayPerfTestHelper::ScalarProductGpuParCuda\n";

//ArrayPerfTestHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 ArrayPerfTestHelper::ScalarProductMultiGpuParCuda\n";

//ArrayPerfTestHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 ArrayPerfTestHelper::ScalarProductGpuCublas\n";

//ArrayPerfTestHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 ArrayPerfTestHelper::ScalarProductMultiGpuCublas\n";

//ArrayPerfTestHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Запускает тест работоспособности VectorGpu

static void Testing\_TestVectorGpu()

{

// Запускаем тест работоспособности VectorGpu

if(TestVectorGpu())

std::cout << "VectorGpu correct!" << std::endl;

else

std::cout << "VectorGpu not correct!" << std::endl;

}

/// @brief Работа с модулем Vectors

static void VectorsHelper\_ConsoleUI()

{

std::cout << "----- VectorsHelper\_ConsoleUI -----\n"

<< "1 Back to main menu\n"

<< "2 VectorRam\n"

//<< "3 VectorsHelper\_ConsoleUI::MatrixRamETesting\n"

//<< "4 VectorsHelper\_ConsoleUI::SumCublas\n"

//<< "5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu\n"

//<< "6 VectorsHelper\_ConsoleUI::CopyRamToGpu\n"

//<< "7 VectorsHelper\_ConsoleUI::CopyGpuToRam\n"

//<< "8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n"

//<< "9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n"

<< "10 VectorRamGpus\n"

//<< "11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n"

//<< "12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n"

//<< "13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n"

//<< "14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 VectorRam\n";

VectorsHelper\_ConsoleUI::VectorRam\_Console\_UI();

break;

case 3:

std::cout << "Command: 3 VectorsHelper\_ConsoleUI::MatrixRamETesting()\n";

//VectorsHelper\_ConsoleUI::MatrixRamETesting();

break;

case 4:

std::cout << "Command: 4 VectorsHelper\_ConsoleUI::SumCublas()\n";

//VectorsHelper\_ConsoleUI::SumCublas\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu()\n";

//VectorsHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 VectorsHelper\_ConsoleUI::CopyRamToGpu()\n";

//VectorsHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 VectorsHelper\_ConsoleUI::CopyGpuToRam()\n";

//VectorsHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 VectorRamGpus\n";

VectorsHelper\_ConsoleUI::VectorRamGpus\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Запускает тесты работоспособности классов матриц

static void MatricesHelper\_ConsoleUI()

{

std::cout << "Matrices" << std::endl;

std::cout << "-- MatrixRamZeroTesting()" << std::endl;

MatricesHelper\_ConsoleUI::MatrixRamZeroTesting();

std::cout << "-- MatrixRamETesting()" << std::endl;

MatricesHelper\_ConsoleUI::MatrixRamETesting();

std::cout << "----- ArrayPerfTestHelper -----\n"

<< "1 Back to main menu\n"

<< "2 MatricesHelper\_ConsoleUI::MatrixRamZeroTesting\n"

<< "3 MatricesHelper\_ConsoleUI::MatrixRamETesting\n"

//<< "4 MatricesHelper\_ConsoleUI::SumCublas\n"

//<< "5 MatricesHelper\_ConsoleUI::SumCublasMultiGpu\n"

//<< "6 MatricesHelper\_ConsoleUI::CopyRamToGpu\n"

//<< "7 MatricesHelper\_ConsoleUI::CopyGpuToRam\n"

//<< "8 MatricesHelper\_ConsoleUI::ScalarProductRamSeq\n"

//<< "9 MatricesHelper\_ConsoleUI::ScalarProductRamParThread\n"

<< "10 MatricesHelper\_ConsoleUI::MatrixBlockRamGpus\_ConsoleUI\n"

//<< "11 MatricesHelper\_ConsoleUI::ScalarProductGpuParCuda\n"

//<< "12 MatricesHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n"

//<< "13 MatricesHelper\_ConsoleUI::ScalarProductGpuCublas\n"

//<< "14 MatricesHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 MatricesHelper\_ConsoleUI::MatrixRamZeroTesting()\n";

MatricesHelper\_ConsoleUI::MatrixRamZeroTesting();

break;

case 3:

std::cout << "Command: 3 MatricesHelper\_ConsoleUI::MatrixRamETesting()\n";

MatricesHelper\_ConsoleUI::MatrixRamETesting();

break;

case 4:

std::cout << "Command: 4 MatricesHelper\_ConsoleUI::SumCublas()\n";

//MatricesHelper\_ConsoleUI::SumCublas\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 MatricesHelper\_ConsoleUI::SumCublasMultiGpu()\n";

//MatricesHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 MatricesHelper\_ConsoleUI::CopyRamToGpu()\n";

//MatricesHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 MatricesHelper\_ConsoleUI::CopyGpuToRam()\n";

//MatricesHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 MatricesHelper\_ConsoleUI::ScalarProductRamSeq\n";

//MatricesHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 MatricesHelper\_ConsoleUI::ScalarProductRamParThread\n";

//MatricesHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 MatricesHelper\_ConsoleUI::MatrixBlockRamGpus\_ConsoleUI\n";

MatricesHelper\_ConsoleUI::MatrixBlockRamGpus\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 MatricesHelper\_ConsoleUI::ScalarProductGpuParCuda\n";

//MatricesHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 MatricesHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n";

//MatricesHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 MatricesHelper\_ConsoleUI::ScalarProductGpuCublas\n";

//MatricesHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 MatricesHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

//MatricesHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Запускает тесты работоспособности классов матриц

static void MathHelper\_ConsoleUI()

{

std::cout << "----- MathHelper\_ConsoleUI -----\n"

<< "1 Back to main menu\n"

<< "2 MathHelper\_ConsoleUI::MathObject\n"

//<< "3 MathHelper\_ConsoleUI::MatrixRamETesting\n"

//<< "4 MathHelper\_ConsoleUI::SumCublas\n"

//<< "5 MathHelper\_ConsoleUI::SumCublasMultiGpu\n"

//<< "6 MathHelper\_ConsoleUI::CopyRamToGpu\n"

//<< "7 MathHelper\_ConsoleUI::CopyGpuToRam\n"

//<< "8 MathHelper\_ConsoleUI::ScalarProductRamSeq\n"

//<< "9 MathHelper\_ConsoleUI::ScalarProductRamParThread\n"

//<< "10 MathHelper\_ConsoleUI::ScalarProductRamParOpenMP\n"

//<< "11 MathHelper\_ConsoleUI::ScalarProductGpuParCuda\n"

//<< "12 MathHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n"

//<< "13 MathHelper\_ConsoleUI::ScalarProductGpuCublas\n"

//<< "14 MathHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 MathHelper\_ConsoleUI::MathObject()\n";

MathHelper\_ConsoleUI::MathObject\_ConsoleUI();

break;

case 3:

std::cout << "Command: 3 MathHelper\_ConsoleUI::MatrixRamETesting()\n";

//MathHelper\_ConsoleUI::MatrixRamETesting();

break;

case 4:

std::cout << "Command: 4 MathHelper\_ConsoleUI::SumCublas()\n";

//MathHelper\_ConsoleUI::SumCublas\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 MathHelper\_ConsoleUI::SumCublasMultiGpu()\n";

//MathHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 MathHelper\_ConsoleUI::CopyRamToGpu()\n";

//MathHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 MathHelper\_ConsoleUI::CopyGpuToRam()\n";

//MathHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 MathHelper\_ConsoleUI::ScalarProductRamSeq\n";

//MathHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 MathHelper\_ConsoleUI::ScalarProductRamParThread\n";

//MathHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 MathHelper\_ConsoleUI::ScalarProductRamParOpenMP\_ConsoleUI\n";

//MathHelper\_ConsoleUI::ScalarProductRamParOpenMP\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 MathHelper\_ConsoleUI::ScalarProductGpuParCuda\n";

//MathHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 MathHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n";

//MathHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 MathHelper\_ConsoleUI::ScalarProductGpuCublas\n";

//MathHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 MathHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

//MathHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Запускает функцию тестирования суммирования элементов массивов

static void Testing\_TestSum()

{

// Запускаем функцию тестирования суммирования элементов массивов

if(TestSum())

std::cout << "TestSum correct!" << std::endl;

else

std::cout << "TestSum not correct!" << std::endl;

}

/// @brief Конфигурирование приложения

static void Application\_Config(AppConfig& config)

{

std::cout << "----- Application configuration -----\n"

<< "1 Back to main menu\n"

<< "2 Print config" << std::endl;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

config.Print();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Конфигурирование приложения

static void ComputingSystemRepository\_Config(ComputingSystemRepository& repo)

{

std::cout << "----- Computing system repository configuration -----\n"

<< "1 Back to main menu\n"

<< "2 Print config\n"

<< "3 Print computing system list\n"

<< "4 Print computing system details\n"

<< "5 Add computing system\n"

<< "6 Change computing system\n"

<< "7 Remove computing system\n"

<< "8 Is computing system exists\n"

<< "9 Clear computing system repository\n"

<< "10 Init computing system repository\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 Print config\n";

repo.PrintConfig();

break;

case 3:

std::cout << "Command: 3 Print computing system list\n";

repo.PrintList();

break;

case 4:

std::cout << "Command: 4 Print computing system details\n";

repo.PrintDetails();

break;

case 5:

std::cout << "Command: 5 Add computing system\n";

repo.Add();

break;

case 6:

std::cout << "Command: 6 Change computing system\n";

repo.Change();

break;

case 7:

std::cout << "Command: 7 Remove computing system\n";

repo.Remove();

break;

case 8:

std::cout << "Command: 8 Is computing system exists\n";

repo.IsExists();

break;

case 9:

std::cout << "Command: 9 Clear computing system repository \n";

repo.Clear();

break;

case 10:

std::cout << "Command: 10 Init computing system repository \n";

repo.Init();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

static void AlgTestingResultRepository\_Config(AlgTestingResultRepository& repo)

{

std::cout << "----- AlgTestingResultRepository configuration -----\n"

<< "1 Back to main menu\n"

<< "2 Print config\n"

<< "3 Get last Id\n"

<< "4 Find alg testing result\n"

<< "5 Add test alg result data\n"

<< "6 Change AlgTestingResultRepository\n"

<< "7 Remove AlgTestingResultRepository\n"

<< "8 Is AlgTestingResultRepository exists\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 Print config\n";

repo.PrintConfig();

break;

case 3:

std::cout << "Command: 3 Get last Id\n";

try

{

std::cout << repo.GetLastId() << std::endl;

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

break;

case 4:

std::cout << "Command: 4 Find alg testing result\n";

repo.Find();

break;

case 5:

std::cout << "Command: 5 Add test alg result data\n";

repo.Add();

break;

case 6:

std::cout << "Command: 6 Change alg testing result\n";

//repo.Change();

break;

case 7:

std::cout << "Command: 7 Remove alg testing result\n";

//repo.Remove();

break;

case 8:

std::cout << "Command: 8 Is alg testing result exists\n";

repo.IsExists();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

// Тестирование функциональности класса FileSystemHelper

static void Testing\_FileSystemHelper()

{

std::cout << "----- FileSystemHelper -----\n"

<< "1 Back to main menu\n"

<< "2 CombinePath\n"

<< "3 CreateFile\n"

<< "4 IsFileExists\n"

<< "5 CreateDir\n"

<< "6 IsDirExists\n"

<< "7 RemoveFile\n"

<< "8 RemoveDir\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 CombinePath\n";

FileSystemHelper::CombinePath();

break;

case 3:

std::cout << "Command: 3 CreateFile\n";

FileSystemHelper::CreateFile();

break;

case 4:

std::cout << "Command: 4 IsFileExists\n";

FileSystemHelper::IsFileExists();

break;

case 5:

std::cout << "Command: 5 CreateDir\n";

FileSystemHelper::CreateDir();

break;

case 6:

std::cout << "Command: 6 IsDirExists\n";

FileSystemHelper::IsDirExists();

break;

case 7:

std::cout << "Command: 7 RemoveFile\n";

FileSystemHelper::RemoveFile();

break;

case 8:

std::cout << "Command: 8 RemoveDir\n";

FileSystemHelper::RemoveDir();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

static void AlgorithmRepository(AlgorithmRepository& repo)

{

std::cout << "----- AlgorithmRepository configuration -----\n"

<< "1 Back to main menu\n"

<< "2 Print algorithms\n"

<< "3 Get algorithm\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 Print config\n";

repo.Print(PrintParams{});

break;

case 3:

std::cout << "Command: 3 Get algorithm\n";

try

{

repo.Get();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Работа с репозиторием реализаций алгоритмов

/// @param repo Объект типа AlgorithmImplementationRepository

static void Menu\_AlgorithmImplementationRepository(AlgorithmImplementationRepository& repo)

{

std::cout << "----- AlgorithmImplementationRepository configuration -----\n"

<< "1 Back to main menu\n"

<< "2 Print algorithm implementations\n"

<< "3 Get algorithm implementation\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "2 Print algorithm implementations\n";

repo.Print(PrintParams{"[\n"});

break;

case 3:

std::cout << "3 Get algorithm implementation\n";

try

{

//repo.Get();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Запуск различных реализаций алгоритмов

/// @param repo Объект типа AlgorithmImplementationRepository

static void Menu\_AlgorithmImplementationExecutor(AlgorithmImplementationExecutor& algorithmImplementationExecutor)

{

std::cout << "----- AlgorithmImplementationExecutor -----\n"

<< "1 Back to main menu\n"

<< "2 Exec T\* Sum\n"

<< "3 Get T\* Sum results\n";

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "2 Exec T\* Sum\n";

AlgorithmImplementationExecutorHelper::Exec(algorithmImplementationExecutor);

break;

case 3:

std::cout << "3 Get T\* Sum results\n";

try

{

//repo.Get();

}

catch(const std::exception& e)

{

std::cerr << e.what() << '\n';

}

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Работа с модулем Geometry

static void GeometryHelper\_ConsoleUI()

{

std::cout << "----- GeometryHelper\_ConsoleUI -----\n"

<< "1 Back to main menu\n"

<< "2 Geometry2D rectangle\n"

//<< "3 VectorsHelper\_ConsoleUI::MatrixRamETesting\n"

//<< "4 VectorsHelper\_ConsoleUI::SumCublas\n"

//<< "5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu\n"

//<< "6 VectorsHelper\_ConsoleUI::CopyRamToGpu\n"

//<< "7 VectorsHelper\_ConsoleUI::CopyGpuToRam\n"

//<< "8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n"

//<< "9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n"

//<< "10 VectorRamGpus\n"

//<< "11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n"

//<< "12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n"

//<< "13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n"

//<< "14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 Geometry2D rectangle\n";

GeometryHelper\_ConsoleUI::Geometry2DRectangle\_Console\_UI();

break;

case 3:

std::cout << "Command: 3 VectorsHelper\_ConsoleUI::MatrixRamETesting()\n";

//VectorsHelper\_ConsoleUI::MatrixRamETesting();

break;

case 4:

std::cout << "Command: 4 VectorsHelper\_ConsoleUI::SumCublas()\n";

//VectorsHelper\_ConsoleUI::SumCublas\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu()\n";

//VectorsHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 VectorsHelper\_ConsoleUI::CopyRamToGpu()\n";

//VectorsHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 VectorsHelper\_ConsoleUI::CopyGpuToRam()\n";

//VectorsHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 VectorRamGpus\n";

VectorsHelper\_ConsoleUI::VectorRamGpus\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Работа с модулем DifferntialEquations

static void DifferentialEquations\_ConsoleUI()

{

std::cout << "----- DifferntialEquations\_ConsoleUI -----\n"

<< "1 Back to main menu\n"

<< "2 Poisson2D\n"

<< "3 DiffEqFunc2DPointSources\n"

<< "4 DiffEqFunc2D\n"

//<< "5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu\n"

//<< "6 VectorsHelper\_ConsoleUI::CopyRamToGpu\n"

//<< "7 VectorsHelper\_ConsoleUI::CopyGpuToRam\n"

//<< "8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n"

//<< "9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n"

//<< "10 VectorRamGpus\n"

//<< "11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n"

//<< "12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n"

//<< "13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n"

//<< "14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: 2 Poisson2D\n";

DifferentialEquations\_ConsoleUI::Poisson2D\_ConsoleUI();

break;

case 3:

std::cout << "Command: 3 DiffEqFunc2DPointSources\n";

DifferentialEquations\_ConsoleUI::DiffEqFunc2DPointSources\_ConsoleUI();

break;

case 4:

std::cout << "Command: 4 DiffEqFunc2D\n";

DifferentialEquations\_ConsoleUI::DiffEqFunc2D\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu()\n";

//VectorsHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 VectorsHelper\_ConsoleUI::CopyRamToGpu()\n";

//VectorsHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 VectorsHelper\_ConsoleUI::CopyGpuToRam()\n";

//VectorsHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 VectorRamGpus\n";

VectorsHelper\_ConsoleUI::VectorRamGpus\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

/// @brief Работа с модулем ModelProblems

static void ModelProblems\_ConsoleUI()

{

std::cout << "----- ModelProblems\_ConsoleUI -----\n"

<< "1 Back to main menu\n"

<< "2 Poisson2D in rectangle (border cond. 1111)\n"

//<< "3 DiffEqFunc2DPointSources\n"

//<< "4 DiffEqFunc2D\n"

//<< "5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu\n"

//<< "6 VectorsHelper\_ConsoleUI::CopyRamToGpu\n"

//<< "7 VectorsHelper\_ConsoleUI::CopyGpuToRam\n"

//<< "8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n"

//<< "9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n"

//<< "10 VectorRamGpus\n"

//<< "11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n"

//<< "12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n"

//<< "13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n"

//<< "14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

;

int command = 0;

while(command != 1)

{

std::cout << ">> ";

std::string commandString;

std::cin >> commandString;

try

{

command = std::stoi(commandString);

}

catch(const std::exception& e)

{

command = 0;

}

switch (command)

{

case 1:

std::cout << "Back to main menu" << std::endl;

break;

case 2:

std::cout << "Command: Poisson2D in rectangle (border cond. 1111)\n";

ModelProblems\_ConsoleUI::Poisson2D\_Rectangle\_bc1111\_ConsoleUI();

break;

case 3:

std::cout << "Command: 3 DiffEqFunc2DPointSources\n";

//DifferentialEquations\_ConsoleUI::DiffEqFunc2DPointSources\_ConsoleUI();

break;

case 4:

std::cout << "Command: 4 DiffEqFunc2D\n";

//DifferentialEquations\_ConsoleUI::DiffEqFunc2D\_ConsoleUI();

break;

case 5:

std::cout << "Command: 5 VectorsHelper\_ConsoleUI::SumCublasMultiGpu()\n";

//VectorsHelper\_ConsoleUI::SumCublasMultiGpu\_ConsoleUI();

break;

case 6:

std::cout << "Command: 6 VectorsHelper\_ConsoleUI::CopyRamToGpu()\n";

//VectorsHelper\_ConsoleUI::CopyRamToGpu\_ConsoleUI();

break;

case 7:

std::cout << "Command: 7 VectorsHelper\_ConsoleUI::CopyGpuToRam()\n";

//VectorsHelper\_ConsoleUI::CopyGpuToRam\_ConsoleUI();

break;

case 8:

std::cout << "Command: 8 VectorsHelper\_ConsoleUI::ScalarProductRamSeq\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamSeq\_ConsoleUI();

break;

case 9:

std::cout << "Command: 9 VectorsHelper\_ConsoleUI::ScalarProductRamParThread\n";

//VectorsHelper\_ConsoleUI::ScalarProductRamParThread\_ConsoleUI();

break;

case 10:

std::cout << "Command: 10 VectorRamGpus\n";

VectorsHelper\_ConsoleUI::VectorRamGpus\_ConsoleUI();

break;

case 11:

std::cout << "Command: 11 VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuParCuda\_ConsoleUI();

break;

case 12:

std::cout << "Command: 12 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuParCuda\_ConsoleUI();

break;

case 13:

std::cout << "Command: 13 VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductGpuCublas\_ConsoleUI();

break;

case 14:

std::cout << "Command: 14 VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\n";

//VectorsHelper\_ConsoleUI::ScalarProductMultiGpuCublas\_ConsoleUI();

break;

default:

std::cout << "Command not recognized!" << std::endl;

break;

}

}

}

};

==================================================

FILE: ModelProblems\_ConsoleUI.hpp

PATH: ModelProblems\ModelProblems\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 1931 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../Geometry/\_IncludeGeometry.hpp"

/// @brief Консольный пользовательский интерфейс для решения модельных задач

struct ModelProblems\_ConsoleUI

{

/// @brief Уравнение Пуассона в прямоугольнике с граничными условиями 1го рода

static void Poisson2D\_Rectangle\_bc1111\_ConsoleUI()

{

std::cout << "Poisson2D\_Rectangle\_bc1111\_ConsoleUI()\n";

// 1. Создаём объект, описывающий геометрию расчетной области

// Прямоугольник 2\*1

IGeometry\* rectangle = new G2DRectangle(2, 1);

// Расположение в точке (1, 2)

IGeometryComposition\* geomComp = new GeometryComposition2D();

geomComp->Add(rectangle, 1, 2);

geomComp->Print();

// 2. Описываем граничные условия

// 3. Задаём уравнение в непрерывной форме

// Задаём искомую физическую величину

// Задаём функцию правой части

// Задаём параметры расчетной сетки

// IGridParams\* gridParams\_ptr = new CalculationGrid2DUniformParams(0.1, 0.2);

// IGrid\* grid = GridFactory::Create2DUniformGrid(geomComp, gridParams\_ptr)

// Задаём схему дискретизации по пространству

// Задаём схему дискретизации граничных условий

// 8. Формируем СЛАУ

// 9. Решаем СЛАУ

// 10. Сохраняем результаты расчета в файл

// 11. Создаём визуализацию

}

};

==================================================

FILE: Poisson2D\_Rectangle.hpp

PATH: ModelProblems\Poisson2D\_Rectangle.hpp

EXTENSION: .hpp

SIZE: 16 bytes

----------------------------------------

CONTENT:

#pragma once

==================================================

FILE: \_IncludeModelProblems.hpp

PATH: ModelProblems\\_IncludeModelProblems.hpp

EXTENSION: .hpp

SIZE: 54 bytes

----------------------------------------

CONTENT:

#pragma once

#include "ModelProblems\_ConsoleUI.hpp"

==================================================

FILE: CalculationStatistics.hpp

PATH: PerformanceTests\CalculationStatistics.hpp

EXTENSION: .hpp

SIZE: 3923 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <vector>

#include "../CommonHelpers/FuncResult.hpp"

/// @brief Статистические параметры результатов численного эксперимента

struct CalculationStatistics

{

// Количество запусков численного эксперимента

unsigned numIter = 0;

// Минимальное значение

double minValue = 0;

// Среднее арифметическое

double avg = 0;

// Медиана

double median = 0;

// 95 процентиль

double percentile\_95 = 0;

// Максимальное значение

double maxValue = 0;

// Среднеквадратическое отклонение

double stdDev = 0;

CalculationStatistics()

{}

template<typename T>

CalculationStatistics(std::vector<FuncResult<T>> results)

{

auto resultsSize = results.size();

if (resultsSize == 0)

throw std::logic\_error("results size is 0");

// Проверяем корректность результатов

for(unsigned i = 1; i < resultsSize; i++)

{

if(results[i].status == false)

throw std::logic\_error("results[i].Status = 0");

if( fabs((results[i].result - results[0].result) / (double)results[0].result) > 0.0001 )

throw std::logic\_error("fabs((results[i].result - results[0].result) / results[0].Result) > 0.0001");

}

//print(std::string("---Before sort---"), results);

// Сортируем results

std::sort(results.begin(), results.end(), FuncResult<T>::compare);

//print(std::string("---After sort---"), results);

//std::cout << "----------" << std::endl;

minValue = results[0].time;

maxValue = results[resultsSize - 1].time;

if(resultsSize % 2 == 0)

{

median = (results[resultsSize / 2 - 1].time + results[resultsSize / 2].time)/2;

}

else

{

median = results[resultsSize / 2].time;

}

// Вычисляем среднее арифметическое

double sum = 0;

for(auto& item : results)

sum += item.time;

avg = sum / resultsSize;

// Вычисляем стандартное отклонение

double sumSq = 0;

for(auto& item : results)

sumSq += pow(item.time - avg, 2);

stdDev = sqrt(sumSq / resultsSize);

// Вычисляем 95 перцентиль

double rang95 = 0.95\*(resultsSize-1) + 1;

unsigned rang95okrVniz = (unsigned)floor(rang95);

percentile\_95 = results[rang95okrVniz-1].time + (rang95-rang95okrVniz)\*(results[rang95okrVniz].time - results[rang95okrVniz-1].time);// Доделать

//Print();

}

void Print()

{

std::cout << "numIter: " << numIter << "; "

<< "minValue: " << minValue << "; "

<< "median: " << median << "; "

<< "avg: " << avg << "; "

<< "percentile\_95: " << percentile\_95 << "; "

<< "maxValue: " << maxValue << "; "

<< "stdDev: " << stdDev << "; ";

}

friend std::ofstream& operator<<(std::ofstream& fout, const CalculationStatistics& data)

{

fout << data.numIter << " "

<< data.minValue << " "

<< data.median << " "

<< data.avg << " "

<< data.percentile\_95 << " "

<< data.maxValue << " "

<< data.stdDev << " ";

return fout;

}

};

==================================================

FILE: ParallelCalcIndicators.hpp

PATH: PerformanceTests\ParallelCalcIndicators.hpp

EXTENSION: .hpp

SIZE: 1813 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Показатели параллельного вычислительного процесса (ускорение, эффективность)

struct ParallelCalcIndicators

{

unsigned Nthreads{};

double Smin{};

double Smax{};

double Savg{};

double Smedian{};

double Sperc95{};

double Emin{};

double Emax{};

double Eavg{};

double Emedian{};

double Eperc95{};

ParallelCalcIndicators()

{}

ParallelCalcIndicators(CalculationStatistics& stat\_seq,

CalculationStatistics& stat\_par,

unsigned Nthreads) : Nthreads(Nthreads)

{

Smin = stat\_seq.minValue / stat\_par.minValue;

Smax = stat\_seq.maxValue / stat\_par.maxValue;

Savg = stat\_seq.avg / stat\_par.avg;

Smedian = stat\_seq.median / stat\_par.median;

Sperc95 = stat\_seq.percentile\_95 / stat\_par.percentile\_95;

Emin = Smin / Nthreads;

Emax = Smax / Nthreads;

Eavg = Savg / Nthreads;

Emedian = Smedian / Nthreads;

Eperc95 = Sperc95 / Nthreads;

}

void Print()

{

std::cout << "N threads: " << Nthreads << std::endl;

std::cout << "Smin: " << Smin << std::endl;

std::cout << "Smax: " << Smax << std::endl;

std::cout << "Savg: " << Savg << std::endl;

std::cout << "Smedian: " << Smedian << std::endl;

std::cout << "Sperc95: " << Sperc95 << std::endl;

std::cout << "Emin: " << Emin << std::endl;

std::cout << "Emax: " << Emax << std::endl;

std::cout << "Eavg: " << Eavg << std::endl;

std::cout << "Emedian: " << Emedian << std::endl;

std::cout << "Eperc95: " << Eperc95 << std::endl;

}

};

==================================================

FILE: PerfTestParams.hpp

PATH: PerformanceTests\PerfTestParams.hpp

EXTENSION: .hpp

SIZE: 2161 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Параметры выполнения тестов производительности

struct PerfTestParams

{

// Количество итераций

unsigned iterNumber;

// Параметры варьирования данных

PerfTestParamsData perfTestParamsData;

// Параметры варьирования количества потоков CPU

PerfTestParamsCpu perfTestParamsCpu;

// Параметры варьирования параметров GPU

PerfTestParamsGpu perfTestParamsGpu;

PerfTestParams()

{}

PerfTestParams(unsigned iterNumber,

PerfTestParamsData perfTestParamsData,

PerfTestParamsCpu perfTestParamsCpu,

PerfTestParamsGpu perfTestParamsGpu) :

iterNumber(iterNumber),

perfTestParamsData(perfTestParamsData),

perfTestParamsCpu(perfTestParamsCpu),

perfTestParamsGpu(perfTestParamsGpu)

{}

PerfTestParams(unsigned iterNumber,

PerfTestParamsData perfTestParamsData,

PerfTestParamsCpu perfTestParamsCpu) :

iterNumber(iterNumber),

perfTestParamsData(perfTestParamsData),

perfTestParamsCpu(perfTestParamsCpu)

{}

PerfTestParams(unsigned iterNumber,

PerfTestParamsData perfTestParamsData) :

iterNumber(iterNumber),

perfTestParamsData(perfTestParamsData)

{}

void Print(PrintParams pp = PrintParams{})

{

std::cout << pp.startMes;

std::cout << "iterNumber" << pp.splitterKeyValue << iterNumber;

std::cout << pp.splitter;

std::cout << "perfTestParamsData" << pp.splitterKeyValue;

perfTestParamsData.Print();

std::cout << pp.splitter;

std::cout << "perfTestParamsCpu" << pp.splitterKeyValue;

perfTestParamsCpu.Print();

std::cout << pp.splitter;

std::cout << "perfTestParamsGpu" << pp.splitterKeyValue;

perfTestParamsGpu.Print();

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: PerfTestParamsCpu.hpp

PATH: PerformanceTests\PerfTestParamsCpu.hpp

EXTENSION: .hpp

SIZE: 1304 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Параметры варьирования потоков CPU

struct PerfTestParamsCpu

{

// Минимальное количество потоков CPU

unsigned cpuThreadsNumMin = 1;

// Максимальное количество потоков CPU

unsigned cpuThreadsNumMax = 1;

// Шаг изменения количества потоков CPU

unsigned cpuThreadsNumStep = 1;

PerfTestParamsCpu()

{}

PerfTestParamsCpu(unsigned cpuThreadsNumMin,

unsigned cpuThreadsNumMax,

unsigned cpuThreadsNumStep) :

cpuThreadsNumMin(cpuThreadsNumMin),

cpuThreadsNumMax(cpuThreadsNumMax),

cpuThreadsNumStep(cpuThreadsNumStep)

{}

void Print(PrintParams pp = PrintParams{})

{

std::cout << pp.startMes;

std::cout << "cpuThreadsNumMin" << pp.splitterKeyValue << cpuThreadsNumMin;

std::cout << pp.splitter;

std::cout << "cpuThreadsNumMax" << pp.splitterKeyValue << cpuThreadsNumMax;

std::cout << pp.splitter;

std::cout << "cpuThreadsNumStep" << pp.splitterKeyValue << cpuThreadsNumStep;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: PerfTestParamsData.hpp

PATH: PerformanceTests\PerfTestParamsData.hpp

EXTENSION: .hpp

SIZE: 1723 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../CommonHelpers/DataTypes.hpp"

/// @brief Параметры варьирования диапазона данных

struct PerfTestParamsData

{

// Типы данных

DataTypes dataTypes;

// Минимальное количество элементов в контейнере

unsigned long long arrayLengthMin = 100000000ull;

// Максимальное количество элементов в контейнере

unsigned long long arrayLengthMax = 1000000000ull;

// Шаг изменения количества элементов в контейнере

unsigned long long arrayLengthStep = 100000000ull;

PerfTestParamsData()

{}

PerfTestParamsData(DataTypes dataTypes,

unsigned long long arrayLengthMin,

unsigned long long arrayLengthMax,

unsigned long long arrayLengthStep) :

dataTypes(dataTypes),

arrayLengthMin(arrayLengthMin),

arrayLengthMax(arrayLengthMax),

arrayLengthStep(arrayLengthStep)

{}

void Print(PrintParams pp = PrintParams{})

{

std::cout << pp.startMes;

std::cout << "dataTypes" << pp.splitterKeyValue;

dataTypes.Print();

std::cout << pp.splitter;

std::cout << "arrayLengthMin" << pp.splitterKeyValue << arrayLengthMin;

std::cout << pp.splitter;

std::cout << "arrayLengthMax" << pp.splitterKeyValue << arrayLengthMax;

std::cout << pp.splitter;

std::cout << "arrayLengthStep" << pp.splitterKeyValue << arrayLengthStep;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: PerfTestParamsGpu.hpp

PATH: PerformanceTests\PerfTestParamsGpu.hpp

EXTENSION: .hpp

SIZE: 2220 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Параметры варьирования блоков и потоков GPU

struct PerfTestParamsGpu

{

// Минимальное количество блоков GPU

unsigned gpuBlockNumMin = 1;

// Максимальное количество блоков GPU

unsigned gpuBlockNumMax = 1;

// Шаг изменения количества блоков GPU

unsigned gpuBlockNumStep = 1;

// Минимальное количество потоков GPU

unsigned gpuThreadNumMin = 1;

// Максимальное количество потоков GPU

unsigned gpuThreadNumMax = 1;

// Шаг изменения количества потоков GPU

unsigned gpuThreadNumStep = 1;

PerfTestParamsGpu()

{}

PerfTestParamsGpu(unsigned gpuBlockNumMin,

unsigned gpuBlockNumMax,

unsigned gpuBlockNumStep,

unsigned gpuThreadNumMin,

unsigned gpuThreadNumMax,

unsigned gpuThreadNumStep) :

gpuBlockNumMin(gpuBlockNumMin),

gpuBlockNumMax(gpuBlockNumMax),

gpuBlockNumStep(gpuBlockNumStep),

gpuThreadNumMin(gpuThreadNumMin),

gpuThreadNumMax(gpuThreadNumMax),

gpuThreadNumStep(gpuThreadNumStep)

{}

void Print(PrintParams pp = PrintParams{})

{

std::cout << pp.startMes;

std::cout << "gpuBlockNumMin" << pp.splitterKeyValue << gpuBlockNumMin;

std::cout << pp.splitter;

std::cout << "gpuBlockNumMax" << pp.splitterKeyValue << gpuBlockNumMax;

std::cout << pp.splitter;

std::cout << "gpuBlockNumStep" << pp.splitterKeyValue << gpuBlockNumStep;

std::cout << pp.splitter;

std::cout << "gpuThreadNumMin" << pp.splitterKeyValue << gpuThreadNumMin;

std::cout << pp.splitter;

std::cout << "gpuThreadNumMax" << pp.splitterKeyValue << gpuThreadNumMax;

std::cout << pp.splitter;

std::cout << "gpuThreadNumStep" << pp.splitterKeyValue << gpuThreadNumStep;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: PerfTestResults.hpp

PATH: PerformanceTests\PerfTestResults.hpp

EXTENSION: .hpp

SIZE: 432 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Результаты вычислительного эксперимента

struct PerfTestResults

{

CalculationStatistics calculationStatistics;

ParallelCalcIndicators parallelCalcIndicators;

void Print(PrintParams pp = PrintParams{})

{

std::cout << pp.startMes;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

};

==================================================

FILE: IScalar.hpp

PATH: Scalars\IScalar.hpp

EXTENSION: .hpp

SIZE: 833 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Абстрактный класс, моделирующий скалярное значение в разных видах памяти

/// @tparam T

template<typename T>

class IScalar

{

public:

/// @brief Место расположения данных

DataLocation dataLocation = DataLocation::RAM;

// /// @brief Выводит в консоль сведения об объекте и его значение

virtual void Print() const = 0;

/// @brief Возвращает значение скаляра

virtual T GetValue() const = 0;

/// @brief Устанавливает значение скаляра

virtual bool SetValue(T value) = 0;

/// @brief Очищает массивы данных

virtual void ClearData() = 0;

};

==================================================

FILE: ScalarRam.hpp

PATH: Scalars\ScalarRam.hpp

EXTENSION: .hpp

SIZE: 1178 bytes

----------------------------------------

CONTENT:

#pragma once

template<typename T>

class ScalarRam : public IScalar<T>

{

public:

T data;

ScalarRam(T value = 0) : data(value)

{

}

~ScalarRam()

{

}

/// @brief Выводит в консоль сведения об объекте

void Print() const override

{

std::cout << "ScalarRam object description:" << std::endl;

std::cout << "type name: " << typeid(this).name() << std::endl;

std::cout << "address: " << this << std::endl;

std::cout << "sizeof data element: " << sizeof(T) << std::endl;

std::cout << "dataLocation: " << this->dataLocation << std::endl;

}

/// @brief Возвращает значение

T GetValue() const override

{

return data;

}

/// @brief Устанавливает значение

bool SetValue(T value) override

{

data = value;

return true;

}

/// @brief Очищает массивы данных и устанавливает размер вектора в 0

void ClearData() override

{

}

};

==================================================

FILE: \_IncludeScalars.hpp

PATH: Scalars\\_IncludeScalars.hpp

EXTENSION: .hpp

SIZE: 68 bytes

----------------------------------------

CONTENT:

#pragma once

#include "IScalar.hpp"

#include "ScalarRam.hpp"

==================================================

FILE: Task.hpp

PATH: Tasks\Task.hpp

EXTENSION: .hpp

SIZE: 808 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Задача (копирование, суммирование и пр.)

enum class Task

{

None,

Init,// Инициализация

Copy,// Копирование

Sum, // Суммирование

Min, // Минимум

Max // Максимум

};

std::ostream& operator<<(std::ostream& os, Task tg)

{

switch (tg)

{

case Task::None:

os << "None";

break;

case Task::Init:

os << "Init";

break;

case Task::Copy:

os << "Copy";

break;

case Task::Sum:

os << "Sum";

break;

case Task::Min:

os << "Min";

break;

case Task::Max:

os << "Max";

break;

default:

break;

}

return os;

}

==================================================

FILE: TaskDimensions.hpp

PATH: Tasks\TaskDimensions.hpp

EXTENSION: .hpp

SIZE: 4952 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include <fstream>

#include "../CommonHelpers/PrintParams.hpp"

/// @brief Размерности задачи

struct TaskDimensions

{

// Используется ли пространственная ось Ox

bool is\_used\_X = false;

// Используется ли пространственная ось Oy

bool is\_used\_Y = false;

// Используется ли пространственная ось Oz

bool is\_used\_Z = false;

// Используется ли ось времени Ot

bool is\_used\_t = false;

/// @brief Возвращает суммарное количество измерений задачи

/// @return

unsigned GetDimensionsNumber() const

{

return (unsigned)is\_used\_X + (unsigned)is\_used\_Y + (unsigned)is\_used\_Z + (unsigned)is\_used\_t;

}

/// @brief Является ли задача стационарной

/// @return true - стационарная, false - нестационарная

bool IsStationaryProblem()

{

return !is\_used\_t;

}

/// @brief Является ли задача нестационарной

/// @return true - нестационарная, false - стационарная

bool IsNonStationaryProblem()

{

return is\_used\_t;

}

/// @brief Является ли задача одномерной

/// @return true - одномерная, false - неодномерная

bool Is1DProblem()

{

if (GetDimensionsNumber() == 1)

return true;

return false;

}

void Print(PrintParams pp)

{

std::cout << pp.startMes;

std::cout << "DimensionsNumber" << pp.splitterKeyValue << GetDimensionsNumber() << pp.splitter;

std::cout << "is\_used\_X" << pp.splitterKeyValue << is\_used\_X << pp.splitter;

std::cout << "is\_used\_Y" << pp.splitterKeyValue << is\_used\_Y << pp.splitter;

std::cout << "is\_used\_Z" << pp.splitterKeyValue << is\_used\_Z << pp.splitter;

std::cout << "is\_used\_t" << pp.splitterKeyValue << is\_used\_t << pp.splitter;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

friend std::ofstream& operator<<(std::ofstream& fout, const TaskDimensions& data)

{

fout << data.GetDimensionsNumber() << " "

<< data.is\_used\_X << " "

<< data.is\_used\_Y << " "

<< data.is\_used\_Z << " "

<< data.is\_used\_t;

return fout;

}

};

/\*

/// @brief Размерности задачи

struct TaskDimensions

{

unsigned dim = 1;// 1 - 1D, 2 - 2D, 3 - 3D, 4 - 3D+t

size\_t x = 1;// Количество элементов по x

size\_t y = 1;// Количество элементов по y

size\_t z = 1;// Количество элементов по z

size\_t t = 1;// Количество элементов по t

/// @brief Возвращает суммарный размер задачи

/// @return

size\_t GetFullSize()

{

return x \* y \* z \* t;

}

/// @brief Является ли задача стационарной

/// @return true - стационарная, false - нестационарная

bool IsStationaryProblem()

{

if (t > 1)

return false;

return true;

}

/// @brief Является ли задача нестационарной

/// @return true - нестационарная, false - стационарная

bool IsNonStationaryProblem()

{

return !IsStationaryProblem();

}

/// @brief Является ли задача стационарной

/// @return true - стационарная, false - нестационарная

bool Is1DProblem()

{

if (y == 1 && z == 1)

return true;

return false;

}

void Print(PrintParams pp)

{

std::cout << pp.startMes;

std::cout << "dim" << pp.splitterKeyValue << dim << pp.splitter;

std::cout << "x" << pp.splitterKeyValue << x << pp.splitter;

std::cout << "y" << pp.splitterKeyValue << y << pp.splitter;

std::cout << "z" << pp.splitterKeyValue << z << pp.splitter;

std::cout << "t" << pp.splitterKeyValue << t << pp.splitter;

std::cout << pp.endMes;

if(pp.isEndl)

std::cout << std::endl;

}

friend std::ofstream& operator<<(std::ofstream& fout, const TaskDimensions& data)

{

fout << data.dim << " "

<< data.x << " "

<< data.y << " "

<< data.z << " "

<< data.t;

return fout;

}

};

\*/

==================================================

FILE: TaskGroup.hpp

PATH: Tasks\TaskGroup.hpp

EXTENSION: .hpp

SIZE: 984 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Группы задач (работа с векторами, матрицами ипр.)

enum class TaskGroup

{

None,

Array,// Массив вида T\* array

Vector,

VecVec,

Matrix,

MatVec,

VecMat,

MatMat

};

std::ostream& operator<<(std::ostream& os, TaskGroup tg)

{

switch (tg)

{

case TaskGroup::None:

os << "None";

break;

case TaskGroup::Array:

os << "Array";

break;

case TaskGroup::Vector:

os << "Vector";

break;

case TaskGroup::VecVec:

os << "VecVec";

break;

case TaskGroup::Matrix:

os << "Matrix";

break;

case TaskGroup::MatVec:

os << "MatVec";

break;

case TaskGroup::VecMat:

os << "VecMat";

break;

case TaskGroup::MatMat:

os << "MatMat";

break;

default:

break;

}

return os;

}

==================================================

FILE: IVector.hpp

PATH: Vectors\IVector.hpp

EXTENSION: .hpp

SIZE: 2062 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Абстрактный класс, моделирующий вектор в N-мерном пространстве

/// @tparam T

template<typename T>

class IVector

{

public:

/// @brief Тип вектора

VectorType vectorType = VectorType::VectorRow;

/// @brief Место расположения данных

DataLocation dataLocation = DataLocation::RAM;

/// @brief Транспонирует вектор (вектор-столбец превращает в вектор-строку и наоборот)

void Transpose()

{

vectorType = (VectorType)!(bool)vectorType;

}

/// @brief Инициализирует все элементы вектора указанным значением

virtual void InitByVal(T val) = 0;

// /// @brief Выводит в консоль сведения об объекте

virtual void Print() const = 0;

/// @brief Выводит в консоль элементы вектора в заданном диапазоне

virtual void PrintData(unsigned long long indStart,

unsigned long long length) const = 0;

/// @brief Выводит в консоль все элементы вектора

virtual void PrintData() const = 0;

/// @brief Возвращает длину вектора (количество элементов)

virtual size\_t Length() const = 0;

/// @brief Возвращает значение элемента вектора, расположенного по указанному индексу

virtual T GetValue(unsigned long long index) const = 0;

/// @brief Устанавливает значение элемента вектора, расположенного по указанному индексу

virtual bool SetValue(unsigned long long index, T value) = 0;

/// @brief Очищает массивы данных и устанавливает размер вектора в 0

virtual void ClearData() = 0;

};

==================================================

FILE: IVectorHelper.hpp

PATH: Vectors\IVectorHelper.hpp

EXTENSION: .hpp

SIZE: 2671 bytes

----------------------------------------

CONTENT:

#pragma once

#include "../Scalars/\_IncludeScalars.hpp"

#include "\_IncludeVectors.hpp"

/// @brief Вспомогательный класс для работы с векторами через интерфейс IVector

struct IVectorHelper

{

/// @brief Создаёт новый вектор, склеивая векторы-аргументы

/// @tparam T

/// @param v1

/// @param v2

/// @return

template<typename T>

static

IVector<T>\* Split(IVector<T>\* v1, IVector<T>\* v2,

DataLocation newVectorDataLocation = DataLocation::RAM)

{

size\_t resultVectorLength = v1->Length() + v2->Length();

IVector<T>\* IVectorSplitResultPtr = nullptr;

// Если результирующий вектор должен располагпться в RAM

if(newVectorDataLocation == DataLocation::RAM)

{

IVectorSplitResultPtr = new VectorRam<T>(resultVectorLength);

if( v1->dataLocation == DataLocation::RAM &&

v2->dataLocation == DataLocation::RAM)

{

size\_t i = 0;

for(size\_t v1i = 0; v1i < v1->Length(); v1i++)

{

auto value = v1->GetValue(v1i);

IVectorSplitResultPtr->SetValue(i, value);

i++;

}

for(size\_t v2i = 0; v2i < v2->Length(); v2i++)

{

auto value = v2->GetValue(v2i);

IVectorSplitResultPtr->SetValue(i, value);

i++;

}

return IVectorSplitResultPtr;

}

}

throw std::runtime\_error("IVectorHelper::Split(): Not realized!");

}

/// Вычисляет скалярное произведение векторов

template<typename T>

static

IScalar<T>\* Dot(IVector<T>\* v1, IVector<T>\* v2,

DataLocation resultDataLocation = DataLocation::RAM)

{

IScalar<T>\* resultPtr = nullptr;

// Если результирующий объект должен располагаться в RAM

if(resultDataLocation == DataLocation::RAM)

{

resultPtr = new ScalarRam<T>();

if( v1->dataLocation == DataLocation::RAM &&

v2->dataLocation == DataLocation::RAM)

{

size\_t i = 0;

return resultPtr;

}

}

throw std::runtime\_error("IVectorHelper::Sum(): Not realized!");

}

};

==================================================

FILE: VectorGpu.hpp

PATH: Vectors\VectorGpu.hpp

EXTENSION: .hpp

SIZE: 9437 bytes

----------------------------------------

CONTENT:

#pragma once

/// @brief Вектор (в GPU)

/// @tparam T Тип элементов вектора

template<typename T>

class VectorGpu : public IVector<T>

{

public:

// Количество элементов вектора

size\_t \_size = 0;

// Указатель на массив в видеопамяти

T\* \_dev\_data = nullptr;

// Флаг инициализации вектора

// false - неинициализирован, true - инициализирован

bool \_isInitialized = false;

VectorGpu(size\_t size) : \_size(size)

{

#ifdef \_\_NVCC\_\_

//std::cout << "VectorGpu(size\_t size) constructor started...\n";

if (\_size == 0)

{

std::string mes = "Cannot initialize vector of \_size = 0";

//std::cerr << mes << std::endl;

throw std::logic\_error(mes);

}

cudaError\_t cudaResult = cudaMalloc(&\_dev\_data, size\*sizeof(T));

if (cudaResult != cudaSuccess)

{

std::string msg("Could not allocate device memory for VectorGpu: ");

msg += cudaGetErrorString(cudaResult);

throw std::runtime\_error(msg);

}

//std::cout << "VectorGpu(size\_t size): Device memory for VectorGpu allocated!\n";

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

VectorGpu(VectorRam<T> vecRam) : \_size(vecRam.GetSize())

{

#ifdef \_\_NVCC\_\_

std::cout << "VectorGpu(VectorRam<T> vecRam) constructor started...\n";

if (\_size == 0)

{

std::string mes = "Cannot initialize vector of \_size = 0";

//std::cerr << mes << std::endl;

throw std::logic\_error(mes);

}

cudaError\_t cudaResult = cudaMalloc(&\_dev\_data, \_size\*sizeof(T));

if (cudaResult != cudaSuccess)

{

std::string msg("Could not allocate device memory for VectorGpu: ");

msg += cudaGetErrorString(cudaResult);

throw std::runtime\_error(msg);

}

std::cout << "VectorGpu(VectorRam<T> vecRam): Device memory for VectorGpu allocated!\n";

// Копируем данные в видеопамять

cudaResult = cudaMemcpy(\_dev\_data, vecRam.Get\_data\_pointer(), \_size\*sizeof(T), cudaMemcpyHostToDevice);

if (cudaResult != cudaSuccess)

{

std::string msg("Could not copy data from RAM to device memory: ");

msg += cudaGetErrorString(cudaResult);

throw std::runtime\_error(msg);

}

//std::cout << "cudaMemCpy OK!\n";

// Устанавливаем флаг инициализации вектора

\_isInitialized = true;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

~VectorGpu()

{

//std::cout << "~VectorGpu(): " << this << " destructed!\n";

}

/// @brief Проверяет состояние вектора

bool CheckState()

{

if(!\_isInitialized)

return false;

if(\_size < 1)

return false;

if(\_dev\_data == nullptr)

return false;

return true;

}

/// @brief Возвращает сумму элементов вектора

/\*FuncResultScalar<T> Sum(unsigned blocksNum, unsigned threadsNum)

{

if(!CheckState())

throw std::logic\_error("Vector is not initialized!");

std::chrono::steady\_clock::time\_point begin = std::chrono::steady\_clock::now();

T result = CudaHelper<T>::Sum(\_dev\_data, \_size, blocksNum, threadsNum);

std::chrono::steady\_clock::time\_point end = std::chrono::steady\_clock::now();

//std::cout << "Time difference = " << std::chrono::duration\_cast<std::chrono::microseconds>(end - begin).count() << "[us]" << std::endl;

FuncResultScalar<T> res{true, result, std::chrono::duration\_cast<std::chrono::microseconds>(end - begin).count()};

//res.Print();

return res;

}\*/

/// @brief Освобождаем массив в видеопамяти

void Clear\_dev\_data()

{

#ifdef \_\_NVCC\_\_

if(\_dev\_data != nullptr)

{

cudaFree(\_dev\_data);

\_dev\_data = nullptr;

\_isInitialized = false;

//std::cout << "Device memory for VectorGpu cleared!\n";

}

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

/// @brief Возвращает указатель на данные в видеопамяти

/// @return

#ifdef \_\_NVCC\_\_

\_\_host\_\_ \_\_device\_\_

#endif

T\* Get\_dev\_data\_pointer()

{

return \_dev\_data;

}

/\*#ifdef \_\_NVCC\_\_

\_\_host\_\_ \_\_device\_\_

#endif\*/

size\_t Length() const override

{

return \_size;

}

/// @brief Инициализирует вектор числом

void InitByVal(double value) override

{

#ifdef \_\_NVCC\_\_

// Создаём временный массив

T\* tmp = new T[\_size];

// Инициализируем временный массив

for (size\_t i = 0; i < \_size; i++)

{

tmp[i] = value;

//std::cout << tmp[i] << " ";

}

//std::cout << std::endl;

// Копируем данные из временного массива в видеопамять

cudaError\_t cudaResult = cudaMemcpy(\_dev\_data, tmp, \_size\*sizeof(T), cudaMemcpyHostToDevice);

if (cudaResult != cudaSuccess)

{

std::string msg("Could not copy data from RAM to device memory: ");

msg += cudaGetErrorString(cudaResult);

throw std::runtime\_error(msg);

}

//std::cout << "cudaMemCpy OK!\n";

// Освобождаем временный массив

delete[] tmp;

// Устанавливаем флаг инициализации вектора

\_isInitialized = true;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

/// @brief Инициализирует вектор числами из диапазона от start до end

void InitVectorByRange(double start, double end)

{

#ifdef \_\_NVCC\_\_

// Создаём временный массив

T\* tmp = new T[\_size];

size\_t cnt = 0;

// Инициализируем временный массив

auto step = (end-start)/(\_size-1);

for (auto i = start; i < end+step/2; i+=step)

{

tmp[cnt++] = i;

//std::cout << tmp[cnt-1] << " ";

}

std::cout << std::endl;

// Копируем данные из временного массива в видеопамять

cudaError\_t cudaResult = cudaMemcpy(\_dev\_data, tmp, \_size\*sizeof(T), cudaMemcpyHostToDevice);

if (cudaResult != cudaSuccess)

{

std::string msg("Could not copy data from RAM to device memory: ");

msg += cudaGetErrorString(cudaResult);

throw std::runtime\_error(msg);

}

//std::cout << "cudaMemCpy OK!\n";

// Освобождаем временный массив

delete[] tmp;

// Устанавливаем флаг инициализации вектора

\_isInitialized = true;

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

void Print() const override

{

#ifdef \_\_NVCC\_\_

kernel\_print<T><<<1,1>>>(\_dev\_data, 0, \_size);

cudaDeviceSynchronize();

#else

throw std::runtime\_error("CUDA not supported!");

#endif

}

/// @brief Выводит в консоль элементы вектора в заданном диапазоне

void PrintData(unsigned long long indStart,

unsigned long long length) const override

{

throw std::runtime\_error("Not realized!");

}

/// @brief Выводит в консоль все элементы вектора

void PrintData() const override

{

PrintData(0, Length());

}

/// @brief Возвращает значение элемента вектора, расположенного по указанному индексу

T GetValue(unsigned long long index) const override

{

throw std::runtime\_error("Not realized!");

}

/// @brief Устанавливает значение элемента вектора, расположенного по указанному индексу

bool SetValue(unsigned long long index, T value) override

{

throw std::runtime\_error("Not realized!");

}

/// @brief Очищает массивы данных и устанавливает размер вектора в 0

void ClearData() override

{

throw std::runtime\_error("Not realized!");

//delete[] data;

//data = nullptr;

//this->length = 0;

}

};

==================================================

FILE: VectorGpuHelper.hpp

PATH: Vectors\VectorGpuHelper.hpp

EXTENSION: .hpp

SIZE: 1512 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/FuncResult.hpp"

#include "VectorGpu.hpp"

class VectorGpuHelper

{

public:

template<typename T>

static

FuncResult<T> SumCuda(VectorGpu<T>& v, size\_t indStart, size\_t indEnd, unsigned NumBlocks, unsigned Nthreads)

{

bool calcStatus = true;

auto start = high\_resolution\_clock::now();

T result = ArrayHelper::SumCuda(v.\_dev\_data, indStart, indEnd, NumBlocks, Nthreads);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

return FuncResult<T>(calcStatus, result, t);

}

template<typename T>

static

FuncResult<T> SumCuda(VectorGpu<T>& v, unsigned NumBlocks, unsigned Nthreads)

{

return SumCuda(v, 0, v.Length() - 1, NumBlocks, Nthreads);

}

/////////////

// Суммирование на нескольких GPU

template<typename T>

static

FuncResult<T> SumCudaMultiGpu(std::vector<ArrayGpuProcessingParams<T>> params)

{

bool calcStatus = true;

auto start = high\_resolution\_clock::now();

T result = ArrayHelper::SumCudaMultiGpu(params);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

return FuncResult<T>(calcStatus, result, t);

}

};

==================================================

FILE: VectorRam.hpp

PATH: Vectors\VectorRam.hpp

EXTENSION: .hpp

SIZE: 3193 bytes

----------------------------------------

CONTENT:

#pragma once

template<typename T>

class VectorRam : public IVector<T>

{

public:

T\* data;

size\_t length;

VectorRam(size\_t length) : length(length)

{

data = new T[length];

}

~VectorRam()

{

if(data)

delete[] data;

}

void InitByVal(T val) override

{

for (size\_t i = 0; i < length; i++)

{

data[i] = val;

}

}

/// @brief Выводит в консоль сведения об объекте

void Print() const override

{

std::cout << "VectorRam object description:" << std::endl;

std::cout << "type name: " << typeid(this).name() << std::endl;

std::cout << "address: " << this << std::endl;

std::cout << "vector type: " << this->vectorType << std::endl;

std::cout << "length: " << length << std::endl;

std::cout << "sizeof 1 data element: " << sizeof(T) << std::endl;

std::cout << "size of data: " << sizeof(T) \* Length() << std::endl;

std::cout << "dataLocation: " << this->dataLocation << std::endl;

}

/// @brief Выводит в консоль элементы вектора в заданном диапазоне

void PrintData(unsigned long long indStart,

unsigned long long length) const override

{

if(indStart + length > Length())

{

throw std::runtime\_error("Exception in PrintData()! Out of range: indStart + length > Length()");

}

std::string splitter = " ";

if(this->vectorType == VectorType::VectorColumn)

splitter = "\n";

for (size\_t i = indStart; i < length; i++)

{

std::cout << data[i] << splitter;

}

std::cout << std::endl;

}

/// @brief Выводит в консоль все элементы вектора

void PrintData() const override

{

PrintData(0, Length());

}

/// @brief Возвращает длину вектора (количество элементов)

/// @return

size\_t Length() const override

{

return length;

}

/// @brief Возвращает значение элемента вектора, расположенного по указанному индексу

T GetValue(unsigned long long index) const override

{

if (index >= Length())

throw std::out\_of\_range("SetValue(): Index out of range!");

return data[index];

}

/// @brief Устанавливает значение элемента вектора, расположенного по указанному индексу

bool SetValue(unsigned long long index, T value) override

{

if (index >= Length())

return false;

data[index] = value;

return true;

}

/// @brief Очищает массивы данных и устанавливает размер вектора в 0

void ClearData() override

{

delete[] data;

data = nullptr;

this->length = 0;

}

};

==================================================

FILE: VectorRamGpus.hpp

PATH: Vectors\VectorRamGpus.hpp

EXTENSION: .hpp

SIZE: 4121 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "IVector.hpp"

#include "../CommonHelpers/DataLocation.hpp"

#include "../Arrays/DevMemArrPointers.hpp"

#include "../Arrays/ArraysIndexMap.hpp"

template<typename T>

class VectorRamGpus : public IVector<T>

{

// Контейнер указателей на части вектора, расположенные в различных областях памяти

DevMemArrPointers<T> devMemArrPointers;

public:

VectorRamGpus()

{

}

void InitByVal(T value) override

{

devMemArrPointers.InitByVal(value);

}

void Print() const override

{

std::cout << "VectorRamGpus::Print()" << std::endl;

std::cout << this << std::endl;

std::cout << "vectorType: " << this->vectorType << std::endl;

std::cout << "devMemArrPointers: ";

devMemArrPointers.Print();

std::cout << std::endl;

}

/// @brief Выводит в консоль элементы вектора в заданном диапазоне

void PrintData(unsigned long long indStart,

unsigned long long length) const override

{

std::string elementSplitter = " ";

if(this->vectorType == VectorType::VectorColumn)

elementSplitter = "\n";

auto lastIndexGlobal = Length() - 1;

for (unsigned long long i = indStart; i < indStart + length; i++)

{

if (i>lastIndexGlobal)

break;

std::cout << GetValue(i);

std::cout << elementSplitter;

}

if (elementSplitter != "\n")

std::cout << std::endl;

}

/// @brief Выводит в консоль все элементы вектора

void PrintData() const

{

PrintData(0, Length());

}

/// @brief Возвращает количество элементов в векторе

/// @return size\_t

size\_t Length() const override

{

return devMemArrPointers.GetSize();

}

/// @brief Возвращает значение элемента вектора, расположенного по указанному индексу

T GetValue(unsigned long long index) const override

{

T value = devMemArrPointers.GetValue(index);

return value;

}

/// @brief Устанавливает значение элемента вектора, расположенного по указанному индексу

bool SetValue(unsigned long long index, T value) override

{

bool isSetted = devMemArrPointers.SetValue(index, value);

return isSetted;

}

/// @brief Транспонирует вектор

void Transpose()

{

this->vectorType = (VectorType)!(bool)this->vectorType;

}

/// @brief Добавляет элементы в вектор

/// @param dataLocation Место расположения элементов вектора

/// @param length Количество добавляемых элементов

/// @return bool - Результат выполнения операции (true - успех)

bool Add(DataLocation dataLocation,

unsigned long long length)

{

auto result = devMemArrPointers.AddBlock(dataLocation, length);

return result;

}

/// @brief Освобождает всю зарезервированную память

void Clear()

{

devMemArrPointers.Clear();

}

template<typename S>

VectorRamGpus& Multiply(S scalar, bool isParallel = false)

{

devMemArrPointers.Multiply(scalar, isParallel);

return \*this;

}

/// @brief Очищает массивы данных и устанавливает размер вектора в 0

void ClearData() override

{

throw std::runtime\_error("Not realized!");

//delete[] data;

//data = nullptr;

//this->length = 0;

}

};

==================================================

FILE: VectorRamGpusHelper.hpp

PATH: Vectors\VectorRamGpusHelper.hpp

EXTENSION: .hpp

SIZE: 210 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "VectorRamGpus.hpp"

/// @brief Вспомогательный класс для работы с классом VectorRamGpus

struct VectorRamGpusHelper

{

};

==================================================

FILE: VectorRamHelper.hpp

PATH: Vectors\VectorRamHelper.hpp

EXTENSION: .hpp

SIZE: 2258 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "../CommonHelpers/FuncResult.hpp"

#include "VectorRam.hpp"

class VectorRamHelper

{

public:

template<typename T>

static

FuncResult<T> Sum(VectorRam<T>& v, size\_t indStart, size\_t indEnd)

{

bool calcStatus = true;

auto start = high\_resolution\_clock::now();

T result = ArrayHelper::Sum(v.data, indStart, indEnd);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

return FuncResult<T>(calcStatus, result, t);

}

template<typename T>

static

FuncResult<T> Sum(VectorRam<T>& v)

{

return Sum(v, 0, v.Length() - 1);

}

template<typename T>

static

FuncResult<T> Sum(VectorRam<T>& v, size\_t indStart, size\_t indEnd, unsigned threadsNum)

{

bool calcStatus = true;

auto start = high\_resolution\_clock::now();

T result = ArrayHelper::Sum(v.data, indStart, indEnd, threadsNum);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

return FuncResult<T>(calcStatus, result, t);

}

template<typename T>

static

FuncResult<T> Sum(VectorRam<T>& v, unsigned threadsNum)

{

return Sum(v, 0, v.Length() - 1, threadsNum);

}

/////////////////// OpenMP ////////////////////

template<typename T>

static

FuncResult<T> SumOpenMP(VectorRam<T>& v, size\_t indStart, size\_t indEnd, unsigned threadsNum)

{

bool calcStatus = true;

auto start = high\_resolution\_clock::now();

T result = ArrayHelper::SumOpenMP(v.data, indStart, indEnd, threadsNum);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>(stop - start);

auto t = duration.count();

return FuncResult<T>(calcStatus, result, t);

}

template<typename T>

static

FuncResult<T> SumOpenMP(VectorRam<T>& v, unsigned threadsNum)

{

return SumOpenMP(v, 0, v.Length() - 1, threadsNum);

}

};

==================================================

FILE: VectorsHelper\_ConsoleUI.hpp

PATH: Vectors\VectorsHelper\_ConsoleUI.hpp

EXTENSION: .hpp

SIZE: 7330 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

#include "\_IncludeVectors.hpp"

#include "../CommonHelpers/DataLocation.hpp"

struct VectorsHelper\_ConsoleUI

{

static void VectorRam\_Console\_UI()

{

std::cout << "VectorRam\_Console\_UI" << std::endl;

ConsoleHelper::PrintLine("Creating VectorRam object with 10 elements");

VectorRam<double> v1(10);

v1.Print();

v1.InitByVal(5);

std::cout << "v1.SetValue(1, -10.123)" << std::endl;

v1.SetValue(1, -10.123);

std::cout << "v1.GetValue(1): " << v1.GetValue(1) << std::endl;

v1.PrintData(0, v1.Length());

IVector<double>\* IVector1Ptr = &v1;

IVector1Ptr->Transpose();

std::cout << "typeid(IVector1Ptr).name(): " << typeid(IVector1Ptr).name() << std::endl;

IVector1Ptr->Print();

IVector1Ptr->PrintData(0, IVector1Ptr->Length());

IVector<double>\* IVectorSplitResultPtr = IVectorHelper::Split(IVector1Ptr, IVector1Ptr, DataLocation::RAM);

IVectorSplitResultPtr->Print();

IVectorSplitResultPtr->PrintData();

ConsoleHelper::PrintLine("ClearData()");

IVectorSplitResultPtr->ClearData();

IVectorSplitResultPtr->Print();

IVectorSplitResultPtr->PrintData();

}

static void VectorRamGpus\_ConsoleUI()

{

std::cout << "VectorRamGpus\_ConsoleUI" << std::endl;

bool res;

ConsoleHelper::PrintLine("VectorRamGpus<double> v1;");

VectorRamGpus<double> v1;

ConsoleHelper::PrintLine("v1.Print();");

v1.Print();

ConsoleHelper::PrintLine("----------\n");

//ConsoleHelper::WaitAnyKey();

unsigned N\_RAM = 800000000;

ConsoleHelper::PrintKeyValue("N\_RAM", N\_RAM);

ConsoleHelper::PrintLine("auto res = v1.Add(DataLocation::RAM, N\_RAM);");

res = v1.Add(DataLocation::RAM, N\_RAM);

ConsoleHelper::PrintKeyValue("res", res);

ConsoleHelper::PrintLine("v1.Print();");

v1.Print();

ConsoleHelper::PrintLine("----------\n");

//ConsoleHelper::WaitAnyKey();

unsigned N\_GPU0 = 800000000;

ConsoleHelper::PrintKeyValue("N\_GPU0", N\_GPU0);

ConsoleHelper::PrintLine("res = v1.Add(DataLocation::GPU0, N\_GPU0);");

res = v1.Add(DataLocation::GPU0, N\_GPU0);

ConsoleHelper::PrintKeyValue("res", res);

ConsoleHelper::PrintLine("v1.Print();");

v1.Print();

ConsoleHelper::PrintLine("----------\n");

//ConsoleHelper::WaitAnyKey();

unsigned N\_GPU1 = N\_GPU0;

ConsoleHelper::PrintKeyValue("N\_GPU1", N\_GPU1);

ConsoleHelper::PrintLine("res = v1.Add(DataLocation::GPU1, N\_GPU1);");

//res = v1.Add(DataLocation::GPU1, N\_GPU1);

ConsoleHelper::PrintKeyValue("res", res);

ConsoleHelper::PrintLine("v1.Print();");

v1.Print();

ConsoleHelper::PrintLine("----------\n");

//ConsoleHelper::WaitAnyKey();

ConsoleHelper::PrintLine("auto size = v1.Size();");

auto size = v1.Length();

ConsoleHelper::PrintKeyValue("size", size);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::WaitAnyKey();

ConsoleHelper::PrintLine("v1.PrintData(0, size);");

//v1.PrintData(0, size);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.InitByVal(0.01);");

v1.InitByVal(0.01);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.PrintData(0, size-1);");

//v1.PrintData(0, size-1);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.Transpose();");

v1.Transpose();

ConsoleHelper::PrintLine("v1.Print();");

v1.Print();

std::cout << "v1.PrintData(0, 5);" << std::endl;

v1.PrintData(0, 5);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.Transpose();");

v1.Transpose();

ConsoleHelper::PrintLine("v1.Print();");

v1.Print();

ConsoleHelper::PrintLine("v1.PrintData(0, 5);");

v1.PrintData(0, 5);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("bool isValueSetted = v1.SetValue(1, 123.45);");

bool isValueSetted = v1.SetValue(1, 123.45);

ConsoleHelper::PrintKeyValue("isValueSetted", isValueSetted);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("auto 1 = v1.GetValue(1);");

auto val = v1.GetValue(1);

ConsoleHelper::PrintKeyValue("val", val);

ConsoleHelper::PrintLine("----------\n");

std::cout << "bool isValueSetted = v1.SetValue(11, 23.455);" << std::endl;

isValueSetted = v1.SetValue(11, 23.455);

ConsoleHelper::PrintKeyValue("isValueSetted", isValueSetted);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.GetValue(11);");

val = v1.GetValue(11);

ConsoleHelper::PrintKeyValue("val", val);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("bool isValueSetted = v1.SetValue(size-1, 23.456);");

isValueSetted = v1.SetValue(size-1, 23.456);

ConsoleHelper::PrintKeyValue("isValueSetted", isValueSetted);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("val = v1.GetValue(size-1);");

val = v1.GetValue(size-1);

ConsoleHelper::PrintKeyValue("val", val);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.PrintData(0, size);");

//v1.PrintData(0, size);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.Multiply(2);");

auto start1 = std::chrono::high\_resolution\_clock::now();

v1.Multiply(2);

auto stop1 = std::chrono::high\_resolution\_clock::now();

auto duration = duration\_cast<std::chrono::microseconds>(stop1 - start1);

long long time\_mks = duration.count();

ConsoleHelper::PrintKeyValue("time\_mks", time\_mks);

ConsoleHelper::PrintLine("v1.PrintData(N\_RAM - 10, 20);");

v1.PrintData(N\_RAM - 10, 20);

ConsoleHelper::PrintLine("----------\n");

ConsoleHelper::PrintLine("v1.Multiply(2, true);");

auto start2 = std::chrono::high\_resolution\_clock::now();

v1.Multiply(2, true);

auto stop2 = std::chrono::high\_resolution\_clock::now();

auto duration2 = duration\_cast<std::chrono::microseconds>(stop2 - start2);

time\_mks = duration2.count();

ConsoleHelper::PrintKeyValue("time\_mks", time\_mks);

ConsoleHelper::PrintLine("v1.PrintData(N\_RAM - 10, 20);");

v1.PrintData(N\_RAM - 10, 20);

ConsoleHelper::PrintLine("----------\n");

//bool isClear = ConsoleHelper::GetBoolFromUser("Do you want clear vector data? (y/n)");

bool isClear = true;

if(isClear)

{

ConsoleHelper::PrintLine("v1.Clear();");

v1.Clear();

}

ConsoleHelper::PrintLine("v1.Print();");

v1.Print();

}

};

==================================================

FILE: VectorType.hpp

PATH: Vectors\VectorType.hpp

EXTENSION: .hpp

SIZE: 588 bytes

----------------------------------------

CONTENT:

#pragma once

#include <iostream>

/// @brief Перечисление "Типы векторов"

enum class VectorType

{

VectorRow, // Вектор-строка

VectorColumn // Вектор-столбец

};

std::ostream& operator<<(std::ostream& os, VectorType vectorType)

{

switch (vectorType)

{

case VectorType::VectorRow:

os << "VectorType::VectorRow";

break;

case VectorType::VectorColumn:

os << "VectorType::VectorColumn";

break;

default:

break;

}

return os;

}

==================================================

FILE: \_IncludeVectors.hpp

PATH: Vectors\\_IncludeVectors.hpp

EXTENSION: .hpp

SIZE: 425 bytes

----------------------------------------

CONTENT:

#pragma once

////////// Векторы (начало) ////////////

#include "VectorType.hpp"

#include "IVector.hpp"

#include "VectorRam.hpp"

#include "VectorGpu.hpp"

#include "VectorRamHelper.hpp"

#include "VectorGpuHelper.hpp"

#include "VectorRamGpus.hpp"

#include "VectorRamGpusHelper.hpp"

#include "IVectorHelper.hpp"

#include "VectorsHelper\_ConsoleUI.hpp"

////////// Векторы (конец) ////////////

==================================================

TOTAL FILES PROCESSED: 128