Министерство образования и науки Российской Федерации

Новосибирский национальный исследовательский государственный университет

Основы параллельного программирования

Отчет по лабораторной работе № 2

Студент: Олимпиев Ю. Ю.

Преподаватель: Артюхов А. А.

Новосибирск, 2023 г.

1. **Цель работы**

Разработать и исследовать параллельные программы решения СЛАУ методом сопряженных градиентов с применением одной из библиотек, реализующих стандарты OpenMP.

1. **Краткое описание подходов к организации решения прикладной задачи параллельными взаимодействующими процессами**

Реализованы 2 подхода к организации параллельной программы при умножении матрицы на вектор:

1. Порождение потоков исполнения при локальной необходимости (распараллеливание непосредственно вычисляющего кода).
2. Однократное порождение потоков исполнения перед общим блоком вычислений.

В прикладной программе использованы возможности бибилотек blas и gsl.

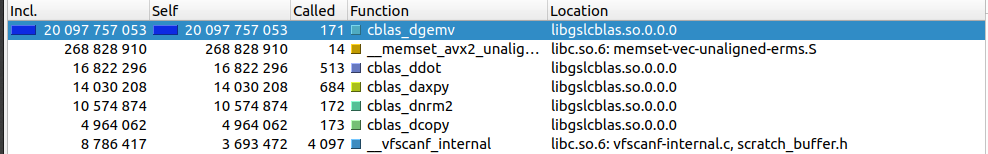
1. **Исследование производительности программ**

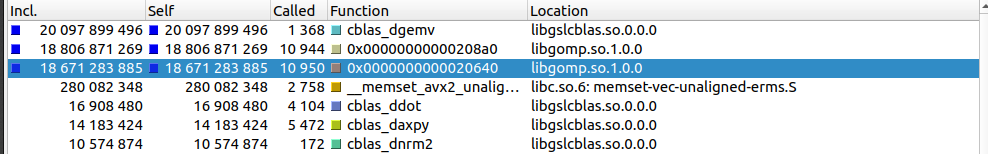
Результаты измерений времени вычислений, ускорения и загруженности для обоих вариантов реализации параллельного кода с использованием средств OpenMP представлены на графиках ниже.

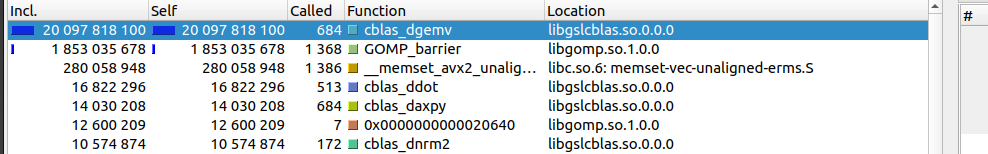
Результаты измерений времени вычислений, ускорения и загруженности для обоих вариантов реализации параллельного кода с использованием средств OpenMP представлены на графиках ниже.

1. **Профилирование**

Данные о профилях вызовов функций в каждом варианте реализации представлены ниже.

Рисунок 1: Профиль вызовов для последовательной версии программы

Рисунок 2: Профиль вызовов для первой версии параллельной программы

Рисунок 3: Профиль вызовов для второй версии параллельной программы

По данным о зываовах функции в различных версиях программы видно, что основное время работы программы занимают вычисления операций вида матрица-вектор. Кроме того, в первой версии программы соразмерной нагрузкой выступают функции библиотеки libgomp (предположительно методы для конструирования и деконструирования потоков), а во второй версии программы большую часть времени работы программы занимает синхронизация посредством барьеров.

1. **Заключение**

Реализованная на языке Си программа позволяет за обозримое время моделировать распределение тепла в пластине, разрешая СЛАУ с некоторой наперед заданной точностью.

Оба варианта реализации параллельного алгоритма дают соизмеримые результаты ускорения и загруженности исполнителей, аномалии в замерах объясняются алгоритмом вычислений (его итерационной природой) и относительной малостью времени вычислений относительно времени на обслуживание параллелизма за счёт использования библиотек gsl и blas.

➜ sudo perf stat -B -e cache-references,cache-misses ./sequentional/Exec/app matrixes/matrix\_64\_64.dat

Performance counter stats for ./sequentional/Exec/app matrixes/matrix\_64\_64.dat&apos:

1 111 382 427 cache-references

8 727 056 cache-misses # 0,785 % of all cache refs

10 134 016 961 cycles

20 384 588 162 instructions # 2,01 insn per cycle

2 923 080 539 branches

32 935 faults

8 migrations

4,239754374 seconds time elapsed

4,165050000 seconds user

0,072018000 seconds sys

➜ sudo perf stat -B -e cache-references,cache-misses./parallel\_1/Exec/app matrixes/matrix\_64\_64.dat

2

Performance counter stats for ./parallel\_1/Exec/app matrixes/matrix\_64\_64.dat:

1 633 534 839 cache-references

17 707 461 cache-misses # 1,084 % of all cache refs

36 420 228 435 cycles

31 848 429 425 instructions # 0,87 insn per cycle

5 010 699 378 branches

32 976 faults

137 migrations

2,200918100 seconds time elapsed

15,094096000 seconds user

0,191266000 seconds sys

➜ sudo perf stat -B -e cache-references,cache-misses ./parallel\_2/Exec/app matrixes/matrix\_64\_64.dat

Performance counter stats for ./parallel\_2/Exec/app matrixes/matrix\_64\_64.dat:

1 161 858 299 cache-references

12 636 088 cache-misses # 1,088 % of all cache refs

25 512 752 687 cycles

22 506 703 189 instructions # 0,88 insn per cycle

3 523 992 111 branches

32 968 faults

116 migrations

1,603124175 seconds time elapsed

10,622744000 seconds user

0,146821000 seconds sys

➜ **lab2** **git:(main)** sudo perf stat -B -e cache-references,cache-misses ./parallel\_2/Exec/app matrixes/matrix\_128\_128.dat

Performance counter stats for./parallel\_2/Exec/app matrixes/matrix\_128\_128.dat:

41 098 152 057 cache-references

440 298 951 cache-misses # 1,071 % of all cache refs

597 362 014 698 cycles

647 994 170 038 instructions # 1,08 insn per cycle

93 965 534 839 branches

524 670 faults

590 migrations

34,703371562 seconds time elapsed

231,163012000 seconds user

2,737971000 seconds sys

➜ **lab2** **git:(main)** sudo perf stat -B -e cache-references,cache-misses ./parallel\_1/Exec/app matrixes/matrix\_128\_128.dat

Performance counter stats for ./parallel\_1/Exec/app matrixes/matrix\_128\_128.dat:

54 821 869 183 cache-references

526 857 902 cache-misses # 0,961 % of all cache refs

866 085 660 708 cycles

924 119 542 409 instructions # 1,07 insn per cycle

135 370 803 516 branches

524 664 faults

1 243 migrations

47,901587199 seconds time elapsed

330,302886000 seconds user

2,618067000 seconds sys

1. **Приложения**

**seq.c**

**// Copyright 2023 Olimpiev Y. Y.**

**#include <assert.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <gsl/gsl\_blas.h>**

**#include <gsl/gsl\_matrix.h>**

**#include <gsl/gsl\_vector.h>**

**void pretty\_gsl\_matrix\_fprintf(FILE\* out, gsl\_matrix\* matrix, const char\* format) {**

**assert(out);**

**assert(matrix);**

**for (size\_t row = 0; row < matrix->size1; row++) {**

**for (size\_t col = 0; col < matrix->size2; col++) {**

**fprintf(out, format, gsl\_matrix\_get(matrix, row, col));**

**}**

**fprintf(out, "\n");**

**}**

**}**

**gsl\_matrix\* ReadGridMatrix(FILE\* in, size\_t rowsAmount, size\_t colsAmount) {**

**assert(in);**

**gsl\_matrix\* gridMatrix = gsl\_matrix\_calloc(rowsAmount, colsAmount);**

**assert(gridMatrix);**

**if (gsl\_matrix\_fscanf(in, gridMatrix) != 0) {**

**perror("Problem with grid matrix reading.\n");**

**return NULL;**

**}**

**#ifdef DEBUG**

**FILE\* out = fopen("gridmatrix.dat", "w");**

**assert(out);**

**pretty\_gsl\_matrix\_fprintf(out, gridMatrix, "%lf ");**

**fclose(out);**

**#endif**

**return gridMatrix;**

**}**

**double ConjugateGradientsMethodIteration(**

**gsl\_matrix\* A,**

**gsl\_vector\* x,**

**gsl\_vector\* r,**

**gsl\_vector\* z,**

**gsl\_vector\* tmpVec,**

**double bNorm) {**

**double err = 0.0;**

**double alpha = 0.0;**

**double betta = 0.0;**

**double tmp = 0.0;**

**gsl\_blas\_dgemv(CblasNoTrans, 1.0, A, z, 0.0, tmpVec);**

**// Calc (r\_n, r\_n). double tmp <- (r\_n, r\_n)**

**gsl\_blas\_ddot(r, r, &tmp);**

**// Calc (A \* z\_n, z\_n).**

**gsl\_blas\_ddot(tmpVec, z, &alpha);**

**// Calc (r\_n, r\_n) / (A \* z\_n, z\_n).**

**alpha = tmp / alpha;**

**// Calc x\_(n + 1) = x\_n + aplha \* z\_n.**

**gsl\_blas\_daxpy(alpha, z, x);**

**// Calc r\_(n + 1) = r\_n - aplha \* (A \* z\_n).**

**gsl\_blas\_daxpy(-alpha, tmpVec, r);**

**// Calc (r\_(n + 1), r\_(n + 1)).**

**gsl\_blas\_ddot(r, r, &betta);**

**// Calc betta\_(n + 1) = (r\_(n + 1), r\_(n + 1)) / (r\_n, r\_n).**

**betta /= tmp;**

**gsl\_vector\_set\_zero(tmpVec);**

**gsl\_blas\_daxpy(betta, z, tmpVec);**

**gsl\_blas\_daxpy(1.0, r, tmpVec);**

**gsl\_vector\_memcpy(z, tmpVec);**

**err = gsl\_blas\_dnrm2(r) / bNorm;**

**return err;**

**}**

**gsl\_vector\* ConjugateGradientsMethod(gsl\_matrix\* A, gsl\_vector\* B, gsl\_vector\* X) {**

**assert(A);**

**assert(B);**

**assert(X);**

**double eps = 0.00001;**

**gsl\_vector\* tmpVec = gsl\_vector\_calloc(B->size);**

**assert(tmpVec);**

**gsl\_vector\* r = gsl\_vector\_calloc(B->size);**

**assert(r);**

**// Calc r = b - Ax. But x = (0), so r = b.**

**gsl\_vector\_memcpy(r, B);**

**gsl\_vector\* z = gsl\_vector\_calloc(B->size);**

**assert(z);**

**gsl\_vector\_memcpy(z, r);**

**double err = 0.0;**

**double normB = gsl\_blas\_dnrm2(B);**

**do {**

**err = ConjugateGradientsMethodIteration(A, X, r, z, tmpVec, normB);**

**} while (eps < err);**

**gsl\_vector\_free(r);**

**gsl\_vector\_free(z);**

**gsl\_vector\_free(tmpVec);**

**return X;**

**}**

**gsl\_matrix\* BuildKernelMatrix(size\_t rowsAmount, size\_t colsAmount) {**

**size\_t kernelMatrixSize = colsAmount \* rowsAmount;**

**gsl\_matrix\* kernelMatrix = gsl\_matrix\_calloc(kernelMatrixSize, kernelMatrixSize);**

**assert(kernelMatrix);**

**for (size\_t row = 0; row < kernelMatrixSize - colsAmount; row++) {**

**// Set three diagonals.**

**gsl\_matrix\_set(kernelMatrix, row, row, -4.0);**

**gsl\_matrix\_set(kernelMatrix, row, row + 1, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row + 1, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row + colsAmount, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row, row + colsAmount, 1.0);**

**}**

**for (size\_t row = kernelMatrixSize - colsAmount; row < kernelMatrixSize; row++) {**

**// Set three diagonals.**

**gsl\_matrix\_set(kernelMatrix, row, row, -4.0);**

**gsl\_matrix\_set(kernelMatrix, row, row - 1, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row - 1, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row - colsAmount, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row, row - colsAmount, 1.0);**

**}**

**#ifdef DEBUG**

**FILE\* out = fopen("kernelmatrix.dat", "w");**

**assert(out);**

**pretty\_gsl\_matrix\_fprintf(out, kernelMatrix, "%lf ");**

**fclose(out);**

**#endif**

**return kernelMatrix;**

**}**

**gsl\_vector\* BuildAnswerVector(size\_t rowsAmount, size\_t colsAmount) {**

**gsl\_vector\* answerVector = gsl\_vector\_calloc(rowsAmount \* colsAmount);**

**assert(answerVector);**

**#ifdef DEBUG**

**FILE\* out = fopen("answervector.dat", "w");**

**assert(out);**

**gsl\_vector\_fprintf(out, answerVector, "%lf ");**

**fclose(out);**

**#endif**

**return answerVector;**

**}**

**gsl\_vector\* BuildCoeffsVector(gsl\_matrix\* gridMatrix) {**

**size\_t vectorSize = gridMatrix->size1 \* gridMatrix->size2;**

**gsl\_vector\* coeffsVector = gsl\_vector\_calloc(vectorSize);**

**assert(coeffsVector);**

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

**gsl\_vector\_set(coeffsVector, i, gsl\_matrix\_get(gridMatrix, i / gridMatrix->size2, i % gridMatrix->size2));**

**}**

**#ifdef DEBUG**

**FILE\* out = fopen("coeffsvector.dat", "w");**

**assert(out);**

**gsl\_vector\_fprintf(out, coeffsVector, "%lf ");**

**fclose(out);**

**#endif**

**return coeffsVector;**

**}**

**gsl\_vector\* CalcGridHeatDistribution(gsl\_matrix\* gridMatrix) {**

**assert(gridMatrix);**

**gsl\_matrix\* kernelMatrix = BuildKernelMatrix(gridMatrix->size1, gridMatrix->size2);**

**assert(kernelMatrix);**

**gsl\_vector\* X = BuildAnswerVector(gridMatrix->size1, gridMatrix->size2);**

**assert(X);**

**gsl\_vector\* B = BuildCoeffsVector(gridMatrix);**

**assert(B);**

**gsl\_vector\* ret = ConjugateGradientsMethod(kernelMatrix, B, X);**

**gsl\_matrix\_free(kernelMatrix);**

**gsl\_vector\_free(B);**

**return ret;**

**}**

**int main(int argc, char\* argv[]) {**

**size\_t colsAmount = 0;**

**size\_t rowsAmount = 0;**

**gsl\_matrix\* gridMatrix = NULL;**

**FILE\* in = (argc == 1) ? stdin : fopen(argv[1], "r");**

**assert(in);**

**if (fscanf(in, "%zu %zu", &rowsAmount, &colsAmount) != 2) {**

**perror("Invalid matrix size input.\n");**

**return EXIT\_FAILURE;**

**}**

**gridMatrix = ReadGridMatrix(in, rowsAmount, colsAmount);**

**if (argc != 1) fclose(in);**

**gsl\_vector\* result = CalcGridHeatDistribution(gridMatrix);**

**if (result) {**

**gsl\_vector\_fprintf(stdout, result, "%4lf ");**

**gsl\_vector\_free(result);**

**}**

**gsl\_matrix\_free(gridMatrix);**

**return EXIT\_SUCCESS;**

**}**

**prll\_1.c**

**// Copyright 2023 Olimpiev Y. Y.**

**#include <assert.h>**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <gsl/gsl\_blas.h>**

**#include <gsl/gsl\_matrix.h>**

**#include <gsl/gsl\_vector.h>**

**#include "builders.h"**

**#include <mpich/mpi.h>**

**#include <math.h>**

**void prll\_gsl\_blas\_dgemv(**

**CBLAS\_TRANSPOSE\_t trans,**

**double alpha,**

**gsl\_matrix\* subA,**

**gsl\_vector\* x,**

**double beta,**

**gsl\_vector\* y,**

**gsl\_vector\* answerPart,**

**int cols,**

**int\* scounts,**

**int\* displs) {**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**int size = 0;**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);**

**if (0 != rank && !x) {**

**x = gsl\_vector\_calloc(cols);**

**assert(x);**

**}**

**// Not shared, need in every iteration.**

**MPI\_Bcast(x->data, cols, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**// Calculation.**

**gsl\_blas\_dgemv(trans, alpha, subA, x, beta, answerPart);**

**void\* recv = (0 == rank) ? y->data : NULL;**

**MPI\_Gatherv(answerPart->data, scounts[rank], MPI\_DOUBLE, recv, scounts, displs, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**}**

**double ConjugateGradientsMethodIteration(**

**gsl\_matrix\* subA,**

**gsl\_vector\* x,**

**gsl\_vector\* asnwerPart,**

**gsl\_vector\* r,**

**gsl\_vector\* z,**

**gsl\_vector\* tmpVec,**

**double bNorm,**

**int cols,**

**int\* scounts,**

**int\* displs) {**

**double err = 0.0;**

**double alpha = 0.0;**

**double betta = 0.0;**

**double tmp = 0.0;**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**prll\_gsl\_blas\_dgemv(CblasNoTrans, 1.0, subA, z, 0.0, tmpVec, asnwerPart, cols, scounts, displs);**

**if (0 == rank) {**

**// Calc (r\_n, r\_n). double tmp <- (r\_n, r\_n)**

**gsl\_blas\_ddot(r, r, &tmp);**

**// Calc (A \* z\_n, z\_n).**

**gsl\_blas\_ddot(tmpVec, z, &alpha);**

**// Calc (r\_n, r\_n) / (A \* z\_n, z\_n).**

**alpha = tmp / alpha;**

**// Calc x\_(n + 1) = x\_n + aplha \* z\_n.**

**gsl\_blas\_daxpy(alpha, z, x);**

**// Calc r\_(n + 1) = r\_n - aplha \* (A \* z\_n).**

**gsl\_blas\_daxpy(-alpha, tmpVec, r);**

**// Calc (r\_(n + 1), r\_(n + 1)).**

**gsl\_blas\_ddot(r, r, &betta);**

**// Calc betta\_(n + 1) = (r\_(n + 1), r\_(n + 1)) / (r\_n, r\_n).**

**betta /= tmp;**

**gsl\_vector\_set\_zero(tmpVec);**

**gsl\_blas\_daxpy(betta, z, tmpVec);**

**gsl\_blas\_daxpy(1.0, r, tmpVec);**

**gsl\_vector\_memcpy(z, tmpVec);**

**err = gsl\_blas\_dnrm2(r) / bNorm;**

**}**

**MPI\_Bcast(&err, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**return err;**

**}**

**gsl\_vector\* ConjugateGradientsMethod(gsl\_matrix\* A, gsl\_vector\* B, gsl\_vector\* X) {**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**int size = 0;**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);**

**gsl\_vector\* tmpVec = NULL;**

**gsl\_vector\* r = NULL;**

**gsl\_vector\* z = NULL;**

**int rows = 0;**

**int cols = 0;**

**int lastChildRows = 0;**

**double normB = 0.0;**

**if (0 == rank) {**

**assert(A);**

**assert(B);**

**assert(X);**

**tmpVec = gsl\_vector\_calloc(B->size);**

**assert(tmpVec);**

**r = gsl\_vector\_calloc(B->size);**

**assert(r);**

**// Calc r = b - Ax. But x = (0), so r = b.**

**gsl\_vector\_memcpy(r, B);**

**z = gsl\_vector\_calloc(B->size);**

**assert(z);**

**gsl\_vector\_memcpy(z, r);**

**normB = gsl\_blas\_dnrm2(B);**

**cols = A->size2;**

**rows = trunc((double) A->size1 / (double) size);**

**lastChildRows = A->size2 - (size - 1) \* rows;**

**}**

**MPI\_Bcast(&rows, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Bcast(&cols, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Bcast(&lastChildRows, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**// Send matrix.**

**int\* displs = (int\*)malloc(size\*sizeof(int));**

**int\* scounts = (int\*)malloc(size\*sizeof(int));**

**if (!displs || !scounts) {**

**printf("Error: malloc failed\n");**

**exit(1);**

**}**

**for (int i = 0; i < size; ++i) {**

**scounts[i] = rows \* cols;**

**if (i == size - 1) {**

**scounts[i] = lastChildRows \* cols;**

**}**

**displs[i] = (i == 0) ? 0 : displs[i-1] + scounts[i-1];**

**}**

**gsl\_matrix\* subA = gsl\_matrix\_alloc(scounts[rank]/cols, cols);**

**assert(subA);**

**gsl\_vector\* answerPart = gsl\_vector\_calloc(scounts[rank]/cols);**

**assert(answerPart);**

**void\* dest = (0 == rank) ? A->data : NULL;**

**MPI\_Scatterv(dest, scounts, displs, MPI\_DOUBLE, subA->data, scounts[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**for (int i = 0; i < size; ++i) {**

**scounts[i] /= cols;**

**displs[i] = (i == 0) ? 0 : displs[i-1] + scounts[i-1];**

**}**

**double eps = 0.00001;**

**double err = 0.0;**

**do {**

**err = ConjugateGradientsMethodIteration(subA, X, answerPart, r, z, tmpVec, normB, cols, scounts, displs);**

**} while (eps < err);**

**gsl\_vector\_free(r);**

**gsl\_vector\_free(z);**

**gsl\_vector\_free(tmpVec);**

**free(displs);**

**free(scounts);**

**gsl\_matrix\_free(subA);**

**gsl\_vector\_free(answerPart);**

**return X;**

**}**

**gsl\_vector\* CalcGridHeatDistribution(gsl\_matrix\* gridMatrix) {**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**gsl\_matrix\* kernelMatrix = NULL;**

**gsl\_vector\* X = NULL;**

**gsl\_vector\* B = NULL;**

**if (0 == rank) {**

**assert(gridMatrix);**

**kernelMatrix = BuildKernelMatrix(gridMatrix->size1, gridMatrix->size2);**

**assert(kernelMatrix);**

**X = BuildAnswerVector(gridMatrix->size1, gridMatrix->size2);**

**assert(X);**

**B = BuildCoeffsVector(gridMatrix);**

**assert(B);**

**}**

**gsl\_vector\* ret = ConjugateGradientsMethod(kernelMatrix, B, X);**

**gsl\_matrix\_free(kernelMatrix);**

**gsl\_vector\_free(B);**

**return ret;**

**}**

**int main(int argc, char\* argv[]) {**

**MPI\_Init(&argc, &argv);**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**gsl\_matrix\* gridMatrix = NULL;**

**if (0 == rank) {**

**size\_t colsAmount = 0;**

**size\_t rowsAmount = 0;**

**FILE\* in = (argc == 1) ? stdin : fopen(argv[1], "r");**

**assert(in);**

**if (fscanf(in, "%zu %zu", &rowsAmount, &colsAmount) != 2) {**

**perror("Invalid matrix size input.\n");**

**return EXIT\_FAILURE;**

**}**

**gridMatrix = ReadGridMatrix(in, rowsAmount, colsAmount);**

**if (argc != 1) fclose(in);**

**}**

**double start = 0.0;**

**double finish = 0.0;**

**if (rank == 0) start = MPI\_Wtime();**

**gsl\_vector\* result = CalcGridHeatDistribution(gridMatrix);**

**if (rank == 0) finish = MPI\_Wtime();**

**if (0 == rank && result) {**

**gsl\_vector\_fprintf(stdout, result, "%4lf ");**

**printf("Time is: %lf seconds.\n", finish - start);**

**gsl\_vector\_free(result);**

**}**

**gsl\_matrix\_free(gridMatrix);**

**MPI\_Finalize();**

**return EXIT\_SUCCESS;**

**}**

**builders.c**

**//Copyright 2023 Olimpiev Y. Y.**

**#include "builders.h"**

**#include <assert.h>**

**void pretty\_gsl\_matrix\_fprintf(FILE\* out, gsl\_matrix\* matrix, const char\* format) {**

**assert(out);**

**assert(matrix);**

**for (size\_t row = 0; row < matrix->size1; row++) {**

**for (size\_t col = 0; col < matrix->size2; col++) {**

**fprintf(out, format, gsl\_matrix\_get(matrix, row, col));**

**}**

**fprintf(out, "\n");**

**}**

**}**

**gsl\_matrix\* BuildKernelMatrix(size\_t rowsAmount, size\_t colsAmount) {**

**size\_t kernelMatrixSize = colsAmount \* rowsAmount;**

**gsl\_matrix\* kernelMatrix = gsl\_matrix\_calloc(kernelMatrixSize, kernelMatrixSize);**

**assert(kernelMatrix);**

**for (size\_t row = 0; row < kernelMatrixSize - colsAmount; row++) {**

**// Set three diagonals.**

**gsl\_matrix\_set(kernelMatrix, row, row, -4.0);**

**gsl\_matrix\_set(kernelMatrix, row, row + 1, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row + 1, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row + colsAmount, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row, row + colsAmount, 1.0);**

**}**

**for (size\_t row = kernelMatrixSize - colsAmount; row < kernelMatrixSize; row++) {**

**// Set three diagonals.**

**gsl\_matrix\_set(kernelMatrix, row, row, -4.0);**

**gsl\_matrix\_set(kernelMatrix, row, row - 1, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row - 1, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row - colsAmount, row, 1.0);**

**gsl\_matrix\_set(kernelMatrix, row, row - colsAmount, 1.0);**

**}**

**#ifdef DEBUG**

**FILE\* out = fopen("kernelmatrix.dat", "w");**

**assert(out);**

**pretty\_gsl\_matrix\_fprintf(out, kernelMatrix, "%lf ");**

**fclose(out);**

**#endif**

**return kernelMatrix;**

**}**

**gsl\_vector\* BuildAnswerVector(size\_t rowsAmount, size\_t colsAmount) {**

**gsl\_vector\* answerVector = gsl\_vector\_calloc(rowsAmount \* colsAmount);**

**assert(answerVector);**

**#ifdef DEBUG**

**FILE\* out = fopen("answervector.dat", "w");**

**assert(out);**

**gsl\_vector\_fprintf(out, answerVector, "%lf ");**

**fclose(out);**

**#endif**

**return answerVector;**

**}**

**gsl\_vector\* BuildCoeffsVector(gsl\_matrix\* gridMatrix) {**

**size\_t vectorSize = gridMatrix->size1 \* gridMatrix->size2;**

**gsl\_vector\* coeffsVector = gsl\_vector\_calloc(vectorSize);**

**assert(coeffsVector);**

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

**gsl\_vector\_set(coeffsVector, i, gsl\_matrix\_get(gridMatrix, i / gridMatrix->size2, i % gridMatrix->size2));**

**}**

**#ifdef DEBUG**

**FILE\* out = fopen("coeffsvector.dat", "w");**

**assert(out);**

**gsl\_vector\_fprintf(out, coeffsVector, "%lf ");**

**fclose(out);**

**#endif**

**return coeffsVector;**

**}**

**gsl\_matrix\* ReadGridMatrix(FILE\* in, size\_t rowsAmount, size\_t colsAmount) {**

**assert(in);**

**gsl\_matrix\* gridMatrix = gsl\_matrix\_calloc(rowsAmount, colsAmount);**

**assert(gridMatrix);**

**if (gsl\_matrix\_fscanf(in, gridMatrix) != 0) {**

**perror("Problem with grid matrix reading.\n");**

**return NULL;**

**}**

**#ifdef DEBUG**

**FILE\* out = fopen("gridmatrix.dat", "w");**

**assert(out);**

**pretty\_gsl\_matrix\_fprintf(out, gridMatrix, "%lf ");**

**fclose(out);**

**#endif**

**return gridMatrix;**

**}**

**builders.h**

**// Copyright 2023 Olimpiev Y. Y.**

**#pragma once**

**#include <gsl/gsl\_matrix.h>**

**#include <gsl/gsl\_vector.h>**

**gsl\_matrix\* BuildKernelMatrix(size\_t rowsAmount, size\_t colsAmount);**

**gsl\_vector\* BuildAnswerVector(size\_t rowsAmount, size\_t colsAmount);**

**gsl\_vector\* BuildCoeffsVector(gsl\_matrix\* gridMatrix);**

**void pretty\_gsl\_matrix\_fprintf(FILE\* out, gsl\_matrix\* matrix, const char\* format);**

**gsl\_matrix\* ReadGridMatrix(FILE\* in, size\_t rowsAmount, size\_t colsAmount);**

**prll2.c**

**// Copyright 2023 Olimpiev Y. Y.**

**#include <assert.h>**

**#include <stdio.h>**

**#include <string.h>**

**#include <stdlib.h>**

**#include <gsl/gsl\_blas.h>**

**#include <gsl/gsl\_matrix.h>**

**#include <gsl/gsl\_vector.h>**

**#include "builders.h"**

**#include <mpich/mpi.h>**

**#include <math.h>**

**void prll\_gsl\_blas\_dgemv(**

**CBLAS\_TRANSPOSE\_t trans,**

**double alpha,**

**gsl\_matrix\* subA,**

**gsl\_vector\* x,**

**double beta,**

**gsl\_vector\* y,**

**gsl\_vector\* answerPart,**

**int\* scounts,**

**int\* displs) {**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**int size = 0;**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);**

**gsl\_vector\* xBlock = gsl\_vector\_calloc(scounts[rank]);**

**assert(xBlock);**

**gsl\_vector\* answerBlock = gsl\_vector\_calloc(scounts[rank]);**

**assert(answerBlock);**

**// Send parts of vector x to all threads.**

**void\* sendbuf = (0 == rank) ? x->data: NULL;**

**MPI\_Scatterv(sendbuf, scounts, displs, MPI\_DOUBLE, xBlock->data, scounts[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**gsl\_vector\_set\_all(answerPart, 0.0);**

**for (int i = 0; i < size; i++) {**

**gsl\_matrix\_view currMatrix = gsl\_matrix\_submatrix(subA, 0, displs[(i + rank) % size], scounts[i], scounts[i]);**

**gsl\_blas\_dgemv(trans, alpha, &currMatrix.matrix, xBlock, beta, answerBlock);**

**gsl\_blas\_daxpy(1.0, answerBlock, answerPart);**

**int left = (rank + size - 1) % size;**

**int right = (rank + size + 1) % size;**

**MPI\_Sendrecv\_replace(xBlock->data, scounts[rank], MPI\_DOUBLE, left, 1, right, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);**

**}**

**void\* recv = (0 == rank) ? y->data : NULL;**

**MPI\_Gatherv(answerPart->data, scounts[rank], MPI\_DOUBLE, recv, scounts, displs, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**gsl\_vector\_free(xBlock);**

**gsl\_vector\_free(answerBlock);**

**}**

**double ConjugateGradientsMethodIteration(**

**gsl\_matrix\* subA,**

**gsl\_vector\* x,**

**gsl\_vector\* asnwerPart,**

**gsl\_vector\* r,**

**gsl\_vector\* z,**

**gsl\_vector\* tmpVec,**

**double bNorm,**

**int\* scounts,**

**int\* displs) {**

**double err = 0.0;**

**double alpha = 0.0;**

**double betta = 0.0;**

**double tmp = 0.0;**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**prll\_gsl\_blas\_dgemv(CblasNoTrans, 1.0, subA, z, 0.0, tmpVec, asnwerPart, scounts, displs);**

**if (0 == rank) {**

**// Calc (r\_n, r\_n). double tmp <- (r\_n, r\_n).**

**gsl\_blas\_ddot(r, r, &tmp);**

**// Calc (A \* z\_n, z\_n).**

**gsl\_blas\_ddot(tmpVec, z, &alpha);**

**// Calc (r\_n, r\_n) / (A \* z\_n, z\_n).**

**alpha = tmp / alpha;**

**// Calc x\_(n + 1) = x\_n + aplha \* z\_n.**

**gsl\_blas\_daxpy(alpha, z, x);**

**// Calc r\_(n + 1) = r\_n - aplha \* (A \* z\_n).**

**gsl\_blas\_daxpy(-alpha, tmpVec, r);**

**// Calc (r\_(n + 1), r\_(n + 1)).**

**gsl\_blas\_ddot(r, r, &betta);**

**// Calc betta\_(n + 1) = (r\_(n + 1), r\_(n + 1)) / (r\_n, r\_n).**

**betta /= tmp;**

**// TODO: explain this shit.**

**gsl\_vector\_set\_zero(tmpVec);**

**gsl\_blas\_daxpy(betta, z, tmpVec);**

**gsl\_blas\_daxpy(1.0, r, tmpVec);**

**gsl\_vector\_memcpy(z, tmpVec);**

**err = gsl\_blas\_dnrm2(r) / bNorm;**

**}**

**MPI\_Bcast(&err, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**return err;**

**}**

**gsl\_vector\* ConjugateGradientsMethod(gsl\_matrix\* A, gsl\_vector\* B, gsl\_vector\* X) {**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**int size = 0;**

**MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);**

**gsl\_vector\* tmpVec = NULL;**

**gsl\_vector\* r = NULL;**

**gsl\_vector\* z = NULL;**

**int rows = 0;**

**int cols = 0;**

**int lastChildRows = 0;**

**double normB = 0.0;**

**if (0 == rank) {**

**assert(A);**

**assert(B);**

**assert(X);**

**tmpVec = gsl\_vector\_calloc(B->size);**

**assert(tmpVec);**

**r = gsl\_vector\_calloc(B->size);**

**assert(r);**

**// Calc r = b - Ax. But x = (0), so r = b.**

**// TODO: explain this shit.**

**gsl\_vector\_memcpy(r, B);**

**z = gsl\_vector\_calloc(B->size);**

**assert(z);**

**gsl\_vector\_memcpy(z, r);**

**normB = gsl\_blas\_dnrm2(B);**

**cols = A->size2;**

**rows = trunc((double) A->size1 / (double) size);**

**lastChildRows = A->size2 - (size - 1) \* rows;**

**}**

**MPI\_Bcast(&rows, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Bcast(&cols, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**MPI\_Bcast(&lastChildRows, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);**

**// Send matrix.**

**int\* displs = (int\*)malloc(size\*sizeof(int));**

**int\* scounts = (int\*)malloc(size\*sizeof(int));**

**assert(displs);**

**assert(scounts);**

**for (int i = 0; i < size; ++i) {**

**scounts[i] = rows \* cols;**

**if (i == size - 1) {**

**scounts[i] = lastChildRows \* cols;**

**}**

**displs[i] = (i == 0) ? 0 : displs[i-1] + scounts[i-1];**

**}**

**gsl\_matrix\* subA = gsl\_matrix\_alloc(scounts[rank] / cols, cols);**

**assert(subA);**

**gsl\_vector\* answerPart = gsl\_vector\_calloc(scounts[rank] / cols);**

**assert(answerPart);**

**// Send matrix.**

**void\* dest = (0 == rank) ? A->data : NULL;**

**MPI\_Scatterv(dest, scounts, displs, MPI\_DOUBLE, subA->data, scounts[rank], MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);**

**for (int i = 0; i < size; ++i) {**

**scounts[i] /= cols;**

**displs[i] = (i == 0) ? 0 : displs[i-1] + scounts[i-1];**

**}**

**double eps = 0.00001;**

**double err = 0.0;**

**do {**

**err = ConjugateGradientsMethodIteration(subA, X, answerPart, r, z, tmpVec, normB, scounts, displs);**

**} while (eps < err);**

**gsl\_vector\_free(r);**

**gsl\_vector\_free(z);**

**gsl\_vector\_free(tmpVec);**

**free(displs);**

**free(scounts);**

**gsl\_matrix\_free(subA);**

**gsl\_vector\_free(answerPart);**

**return X;**

**}**

**gsl\_vector\* CalcGridHeatDistribution(gsl\_matrix\* gridMatrix) {**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**gsl\_matrix\* kernelMatrix = NULL;**

**gsl\_vector\* X = NULL;**

**gsl\_vector\* B = NULL;**

**if (0 == rank) {**

**assert(gridMatrix);**

**kernelMatrix = BuildKernelMatrix(gridMatrix->size1, gridMatrix->size2);**

**assert(kernelMatrix);**

**X = BuildAnswerVector(gridMatrix->size1, gridMatrix->size2);**

**assert(X);**

**B = BuildCoeffsVector(gridMatrix);**

**assert(B);**

**}**

**gsl\_vector\* ret = ConjugateGradientsMethod(kernelMatrix, B, X);**

**gsl\_matrix\_free(kernelMatrix);**

**gsl\_vector\_free(B);**

**return ret;**

**}**

**int main(int argc, char\* argv[]) {**

**MPI\_Init(&argc, &argv);**

**int rank = 0;**

**MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);**

**gsl\_matrix\* gridMatrix = NULL;**

**if (0 == rank) {**

**size\_t colsAmount = 0;**

**size\_t rowsAmount = 0;**

**FILE\* in = (argc == 1) ? stdin : fopen(argv[1], "r");**

**assert(in);**

**if (fscanf(in, "%zu %zu", &rowsAmount, &colsAmount) != 2) {**

**perror("Invalid matrix size input.\n");**

**return EXIT\_FAILURE;**

**}**

**gridMatrix = ReadGridMatrix(in, rowsAmount, colsAmount);**

**if (argc != 1) fclose(in);**

**}**

**double start = 0.0;**

**double finish = 0.0;**

**if (rank == 0) start = MPI\_Wtime();**

**gsl\_vector\* result = CalcGridHeatDistribution(gridMatrix);**

**if (0 == rank && result) {**

**finish = MPI\_Wtime();**

**gsl\_vector\_fprintf(stdout, result, "%4lf ");**

**printf("Time is: %lf in seconds.\n", finish - start);**

**gsl\_vector\_free(result);**

**}**

**gsl\_matrix\_free(gridMatrix);**

**MPI\_Finalize();**

**return EXIT\_SUCCESS;**

**}**