Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет»
(Национальный исследовательский университет)
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра математического и компьютерного моделирования

ОТЧЕТ ПО НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ
ЮУрГУ-010400.68.2017.049.001 КР

Руководитель, д.ф.-м.н.,
профессор
_____ А.В. Панюков
« » _____2016 г.

Автор проекта
студент группы ЕТ-213
_____ В.А. Безбородов
« » _____2016 г.

Проект защищен
с оценкой
_____
« » _____2016 г.

Челябинск, 2016

Содержание выполненных работ

В дипломной работе *«Параллельная реализация метода эллипсоидов для задач оптимизации большой размерности»* был произведен анализ вычислительной сложности выполняемых операций алгоритма метода эллипсоидов. На основе результатов анализа была разработана параллельная реализация метода эллипсоидов, адаптированная для решения задач оптимизации большой размерности на многопроцессорных и/или многоядерных вычислительных системах с общей разделяемой памятью.

Структура и содержание выполненных работ направлены на *развитие идей* дипломной работы, сформулированных как направления дальнейших исследований. В частности, предлагалось повысить эффективность работы с типами данных произвольной точности, применить последние алгоритмические разработки в области параллельного умножения матриц, а также использовать модификации метода эллипсоидов для решения одномерных задач оптимизации.

Ядро существующей реализации было полностью переписано на C [8]. Исправлены найденные недочеты и уязвимости. Поскольку C не поддерживает сложные структуры данных [9], а предоставляет для этого низкоуровневый механизм структур (`struct`, POD), был введен тип данных, представляющий матрицы в программном коде (см. листинг 1).

```
/**
 * Matrix structure.
 */
struct mtx
{
    mpfr_t* storage;     ///!< internal storage
    size_t nrows;        ///!< number of rows
    size_t ncols;        ///!< number of columns
};
```

Листинг 1. Представление матриц в программном коде: *storage* – это указатель на выделенный участок памяти, *nrows* и *ncols* используются для адресной арифметики.

Элементы матрицы имеют тип `mpfr_t`. Это специализированный тип данных из библиотеки GNU MPFR Library [1] – портативной библиотеки, на-

писанной на C, для арифметики произвольной точности над числами с плавающей запятой. Она основана на GNU MP Library [2]. Библиотека предоставляет класс чисел с плавающей запятой с точной семантикой. Основными преимуществами библиотеки MPFR от аналогов являются:

- код MPFR портативен, т.е. результат любой операции не зависит от размера машинного слова `mp_bits_per_limb` (64 для большинства современных процессоров);
- точность в битах может быть задана явно для любых допустимых значений для каждой переменной (включая очень маленькую точность);
- библиотека MPFR предоставляет четыре режима округления из стандарта IEEE 754-1985 как для базовых операций, так и для других математических функций.

Для работы с введенным типом матриц был разработан простой базовый интерфейс библиотеки. Сигнатуры некоторых методов представлены в листинге 2.

```
// memory handling
int mtx_init(struct mtx* const m,
        size_t rows, size_t columns, mpfr_prec_t prec);

int mtx_clear(struct mtx const m);

// matrix I/O
int mtx_fprint(FILE* stream, struct mtx const m);
int mtx_fscan(FILE* stream, struct mtx m, char const* delim);

// assignment
int mtx_fill(struct mtx m, mpfr_t val, mpfr_t diagval);
int mtx_fill_d(struct mtx m, double val, double diagval);

// copying
int mtx_copy(struct mtx rop, struct mtx const op);

// multiplication
int mtx_mul(struct mtx rop,
        struct mtx const op1, struct mtx const op2);

int mtx_mulval(struct mtx rop, struct mtx const op1, mpfr_t op2);
```

```
// addition
int mtx_add(struct mtx rop,
        struct mtx const op1, struct mtx const op2);

// transposition
int mtx_tr(struct mtx rop, struct mtx const op);
```

Листинг 2. Интерфейс библиотеки для работы с матрицами.

Все функции сгруппированы в соответствии с типом выполняемых операций:

- **управление памятью** – создание/удаление матриц с выделением/освобождением динамической памяти (параметр `prec` задает число бит, используемых для представления мантиссы числа с плавающей точкой, параметр может принимать любые значения от `MPFR_PREC_MIN` до `MPFR_PREC_MAX`);
- **средства ввода/вывода** – чтение/запись матриц из/в различные источники;
- **операции присваивания** – функции заполнения матриц значением (параметр `diagval` используется для заполнения элементов на диагонали, если матрица квадратная);
- **базовые операции** – матричное умножение, умножение на число, сложение, транспонирование и копирование.

Каждая функция в качестве возвращаемого значения использует ненулевое значение при возникновении ошибки.

Большинство функций библиотеки MPFR принимают дополнительный параметр, называемый *порядок округления*. Библиотека поддерживает следующие пять режимов:

- **MPFR_RNDN** – округление к ближайшему (roundTiesToEven из IEEE 754-2008),
- **MPFR_RNDZ** – округление к нулю (roundTowardZero из IEEE 754-2008),
- **MPFR_RNDU** – округление к плюс бесконечности (roundTowardPositive из IEEE 754-2008),

- **MPFR_RNDD** – округление к минус бесконечности (roundTowardNegative из IEEE 754-2008),
- **MPFR_RNDA** – округление от нуля.

Благодаря своей природе, большинство матричных операций могут быть выполнены параллельно без гонки за данными, захвата ресурсов и с минимальным набором разделяемых данных. По этой причине большинство функций из листинга 2 были распараллелены для ускорения.

Сегодня существует много способов организации параллелизма. Одним из наиболее популярных, современных и гибких является интерфейс OpenMP [3], поддерживающий мультиплатформенное мультипроцессорное программирование на C, C++ и Fortran [6] для распределенных систем с разделяемой памятью на большинстве современных платформ, включая Solaris, AIX, HP-UX, Linux, OS X и Windows. Интерфейс состоит из набора директив компилятора, библиотечных функций и переменных окружения, определяющих поведение времени исполнения.

OpenMP использует портативную, расширяемую модель и предоставляет программистам простой и гибкий интерфейс для разработки параллельных приложений для различных платформ, от стандартного настольного десктопа до суперкомпьютера.

Приложения, созданные с помощью гибридной модели параллельного программирования, могут быть запущены на кластере, использующем как OpenMP, так и MPI, при этом OpenMP обеспечивает параллелизм внутри многоядерного узла, в то время как MPI обеспечивает параллелизм между разными узлами. Также существует возможность запустить OpenMP на распределенных системах с разделяемой памятью [5] для трансляции OpenMP в MPI [4, 7] либо для расширения OpenMP для систем с неразделяемой памятью.

Все сложности организации многопоточного кода библиотека работы с матрицами скрывает «за кулисами». Код, который необходимо выполнить параллельно, помечается как *параллельная область OpenMP*. Для назначения отдельной работы одному или всем потокам используются специальные конструкции. Директивы `omp for` или `omp do` необходимы для разделения

итераций цикла между потоками. Пример объявления параллельного цикла представлен в листинге 3.

```
    size_t i, j, k;

#pragma omp parallel for
    shared(rop) private(i,j,k) schedule(static)

    for (i = 0; i < op1.nrows; ++i)
    {
        for (j = 0; j < op2.ncols; ++j)
        {
            // do something ...
        }
    }
```

Листинг 3. Параллельно выполняемый цикл.

Поскольку OpenMP является моделью программирования для разделяемой памяти, большинство переменных по умолчанию видимы для всех потоков. Но иногда для избежания гонки потоков или для передачи значений между последовательными частями и параллельной областью необходимы закрытые переменные. Управление расположением данных реализуется через указания атрибута разделения данных. Вот некоторые типы:

- **shared** – данные внутри параллельной области разделяемы, что означает их доступность для всех потоков одновременно. По умолчанию, все переменные разделяемы за исключением счетчика итераций цикла.
- **private** – данные внутри параллельной области являются локальными для каждого потока. Это означает, что каждый поток будет иметь собственную копию данных и использовать их как временные переменные. Закрытые данные не инициализируются, и их значение не может быть использовано вне параллельного региона. По умолчанию счетчики итераций цикла являются закрытыми в OpenMP.

Одним из *режимов планирования* является `schedule(type, chunk)`. Он используется по умолчанию для циклов `do` и `for`. Итерации назначаются потокам в соответствии с режимом планирования. При использовании режима `static`, итерации назначаются потокам перед началом выполнения

цикла. По умолчанию итерации распределяются между потоками равномерно. Хотя, указание целочисленного значения в качестве параметра `chunk` приведет к назначению указанного количества последовательных итераций каждому потоку.

Именно таким образом библиотека работы с матрицами разрешает вопрос обработки матриц большого размера за приемлемое время. Разумеется, не все операции могут быть выполнены параллельно (например, ввод/вывод матриц).

Направления будущих исследований

Необходимо применить метод эллипсоидов, построенный на основе параллельной библиотеки для работы с матрицами, для задач линейного программирования больших размерностей, собрать необходимые метрики (время работы, количество затрачиваемых итераций) и сравнить эффективность алгоритма с традиционными алгоритмами решения ЗЛП (симплекс-метод).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. URL: http://www.mpfr.org (дата обращения: 13.12.2016).
2. URL: https://gmplib.org/ (дата обращения: 13.12.2016).
3. URL: http://www.openmp.org (дата обращения: 14.12.2016).
4. Basumallik, A. Programming distributed memory systems using openmp. / A. Basumallik, S. Min, R. Eigenmann // Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium. New York: IEEE Press. — 2007.
5. Costa, J. Running openmp applications efefficient on an everything-shared sdsm. / J.J. Costa // Journal of Parallel and Distributed Computing. Elsevier. — 2006. — no. 66 (5), 647658.
6. Gagne, A. S. Operating system concepts (9-th ed.). / A. S. Gagne, G. Peter Baer Galvin. — Hoboken, N.J.: Wiley. — pp. 181-182.
7. Wang, J. Openmp compiler for distributed memory architectures. / J. Wang, C. Hu, J. Li // Science China Information Sciences. Springer Publishing. — 2010. — no. 53 (5): 932944.
8. Керниган, Б. Язык программировании Си. – 2-е изд. / Б. Керниган, Д. Ритчи. — М.: Вильямс, 2007. — с. 304.
9. Шилдт, Г. С. С: полное руководство, классическое издание. / Г. С. Шилдт. — М.: Вильямс, 2010. — с. 704.