Нижегородский государственный университет им. Н.И. Лобачевского

Параллельные численные методы

Лабораторная работа: Умножение разреженной матрицы на вектор

При поддержке компании Intel

Кустикова В.Д., Мееров И.Б., Сысоев А.В.,
кафедра математического обеспечения ЭВМ

Содержание

- Введение
- Цели работы
- Задача умножения разреженной матрицы на вектор
- Программная реализация
  - Генерация матриц, вспомогательные функции
  - Последовательная реализация
  - Параллельная версия с использованием OpenMP
  - Параллельная версия с использованием библиотеки Intel® Threading Building Blocks
  - Параллельная версия с использованием возможностей технологии Intel® Cilk Plus
  - Реализация с использованием библиотеки Intel® Array Building Blocks

Введение

- Параллелизм на уровне ядер процессора. Многопоточность.
  - OpenMP,
  - Intel® Threading Building Blocks (Intel® TBB),
  - Intel® Cilk Plus (в Cilk для обозначения потоков вводится специальный термин «*worker*»),
  - Intel® Array Building Blocks (Intel® ArBB).
- Параллелизм на уровне данных. Векторизация.
  - Intel® Cilk Plus (CEAN),
  - Intel® Array Building Blocks (Intel® ArBB).

Цели работы

- Изучение некоторых технологий параллельного программирования в системах с общей памятью на примере задачи умножения разреженной матрицы на вектор.

Тестовая инфраструктура

Процессор	2 четырехъядерных процессора Intel Xeon E5520 (2.27 GHz)
Память	16 Gb
Операционная система	Microsoft Windows 7
Среда разработки	Microsoft Visual Studio 2008
Компилятор, профилировщик, отладчик	Intel Parallel Studio XE
Библиотеки	Intel MKL v. 10.2.5.035 Intel® ArBB 1.0 Beta 4 Intel® Threading Building Blocks 3.0 for Windows, Update 3 (в составе Intel® Parallel Studio XE 2011)

Постановка задачи

ЗАДАЧА УМНОЖЕНИЯ РАЗРЕЖЕННОЙ МАТРИЦЫ НА ВЕКТОР

Постановка задачи

- Пусть A – разреженная квадратная матрица размера $N \times N$, x – плотный вектор, состоящий из N элементов.
- Требуется найти вектор $b = A \cdot x$.

- Для хранения разреженной матрицы предлагается использовать CRS формат.

CRS формат хранения разреженной матрицы (1)

- Формат хранения CSR (Compressed Sparse Rows) или CRS (Compressed Row Storage) призван устранить некоторые недоработки координатного представления.
- Используются три массива.
  - первый массив хранит значения элементов построчно (строки рассматриваются по порядку сверху вниз),
  - второй – номера столбцов для каждого элемента,
  - третий заменяет номера строк, используемые в координатном формате, на индекс начала каждой строки.
- Количество элементов массива **RowIndex** равно N + 1.

CRS формат хранения разреженной матрицы (2)

- Количество элементов массива **RowIndex** равно $N + 1$.
- i-ый элемент массива **RowIndex** указывает на начало *i*-ой строки.
- Элементы строки i в массиве **Value** находятся по индексам от **RowIndex[i]** до **RowIndex[i + 1] – 1** включительно.
- Таким образом обрабатывается случай пустых строк, а также добавляется «лишний» элемент в массив **RowIndex** – устраняется особенность при доступе к элементам последней строки. **RowIndex[N] = NZ.**

CRS формат хранения разреженной матрицы (3)

- Оценим объем необходимой памяти.
  - Плотное представление: $M = 8\ N^2$ байт.
  - В координатном формате: $M = 16\ NZ$ байт.
  - В формате CRS:
    $M = 8\ NZ + 4\ NZ + 4\ (N + 1) = 12\ NZ + 4\ N + 4$.
- В часто встречающемся случае, когда $N + 1 < NZ$, данный формат является более эффективным, чем координатный, с точки зрения объема используемой памяти.

CRS формат хранения разреженной матрицы (4)

Пример:

A

1				2	
		3	4		
			8		5
	7	1			6

Структура хранения:

1	2	3	4	8	5	7	1	6

Value

0	4	2	3	3	5	1	2	5

Col

0	2	4	4	6	6	9

RowIndex

CRS формат хранения разреженной матрицы (5)

- Индексация массивов в стиле языка C – с нуля.
- Элементы в строке упорядочиваются по номеру столбца.
- В матрице хранятся числа типа **double**.

```
struct crsMatrix
{
  int N;  // Размер матрицы (N x N)
  int NZ; // Кол-во ненулевых элементов
  // Массив значений (размер NZ)
  double* Value;
  // Массив номеров столбцов (размер NZ)
  int* Col;
  // Массив индексов строк (размер N + 1)
  int* RowIndex;
};
```

Генерация разреженных матриц (1)

- Как выбрать тестовые матрицы? Есть проблемы:
  - Воспроизводимость результатов.
  - Репрезентативность бенчмарка.
  - Приемлемое время работы – не слишком большое, не слишком малое (секунды).
  - …
- Решение:
  - Для достижения целей ЛР достаточно ограничиться некоторым классом (классами) матриц.
  - Не претендуем на лучшее время работы.
  - По возможности ориентируемся на время работы аналогичных программ в индустриальных библиотеках.

Генерация разреженных матриц (2)

- Будем формировать матрицу A при помощи датчика случайных чисел. Данная задача включает два этапа:
  - построение *портрета (шаблона) матрицы*,
  - и наполнение этого портрета конкретными значениями.

Портрет матрицы

X				X	
		X	X		
			X		X
	X	X			X

Структура хранения портрета:

0	4	2	3	3	5	1	2	5

Col

0	2	4	4	6	6	9

RowIndex

Генерация разреженных матриц (3)

```
// Генерирует квадратную матрицу в формате CRS
// (3 массива, индексация с нуля)
// В каждой строке cntInRow ненулевых элементов
void GenerateRegularCRS(int seed, int N,
                        int cntInRow, crsMatrix& mtx);
```

Вспомогательные функции (1)

1. Инициализация матрицы – выделение памяти под структуру данных для хранения матрицы в формате CRS.

```
void InitializeMatrix(int N, int NZ, crsMatrix &mtx)
{
  mtx.N = N;
  mtx.NZ = NZ;
  mtx.Value    = new double[NZ];
  mtx.Col      = new int[NZ];
  mtx.RowIndex = new int[N + 1];
}
```

Вспомогательные функции (2)

2. Инициализация вектора – выделение памяти под структуру данных для хранения вектора.

3. Удаление матрицы – освобождение выделенной ранее памяти.

4. Удаление вектора – освобождение выделенной ранее памяти.

5. Генерация вектора – формирование компонент вектора. Отметим, что значения элементов вектора генерируются в тех же пределах, что и элементы матрицы.

Вспомогательные функции (3)

6. Сравнение векторов – вычисление максимального расхождения компонент двух векторов.

```
int CompareVectors(double* vec1, double* vec2,
                   int n, double &diff)
{
  diff = 0.0;
  for (int i = 0; i < n; i++)
    if (diff < fabs(vec1[i] - vec2[i]))
      diff = fabs(vec1[i] - vec2[i]);

  return 0;
}
```

Вспомогательные функции (4)

7. Умножение разреженной матрицы в формате CRS на вектор – эталонная версия. Для проверки собственных реализаций алгоритма умножения разреженной матрицы на вектор необходимо обеспечить контроль за правильностью результата. Будем использовать функцию **mkl_dcsrgemv()** в качестве эталона. Изучим код в среде MS VS.

8. Запись разреженной матрицы в файл. Реализация данной операции позволяет не тратить время на генерацию матрицы при запуске различных версий умножения разреженной матрицы на вектор, а просто один раз сгенерировать и сохранить матрицу, а потом загружать ее из файла.

Вспомогательные функции (5)

9. Чтение разреженной матрицы из файла.

10. Запись исходного вектора в файл. Также позволяет не генерировать повторно вектор в процессе проведения экспериментов на разных версиях умножения.

11. Чтение вектора из файла.

Программная реализация

ПОСЛЕДОВАТЕЛЬНАЯ РЕАЛИЗАЦИЯ

Последовательная реализация. Создание проекта (1)

- Запустите приложение **Microsoft Visual Studio 2008**.
- В меню **File** выполните команду **New→Project…**.
- В диалоговом окне **New Project** в типах проекта выберите **Win32**, в шаблонах **Win32 Console Application**, в поле **Solution** введите **15_SparseMatrDenseVec**, в поле **Name – 01_Sequence**, в поле **Location** укажите путь к папке с лабораторными работами курса – **c:\ParallelCalculus**. Нажмите **OK**.
- В диалоговом окне **Win32 Application Wizard** нажмите **Next** (или выберите **Application Settings** в дереве слева) и установите флаг **Empty Project**. Нажмите **Finish**.

Последовательная реализация. Создание проекта (2)

- В окне **Solution Explorer** в папке **Source Files** выполните команду контекстного меню **Add→New Item…**. В дереве категорий слева выберите **Code**, в шаблонах справа – **C++ File (.cpp)**, в поле **Name** введите имя файла **main**. Нажмите **Add**.
- Аналогично добавьте файлы **sparse.h** и **sparse.cpp**.
- Добавьте объявление структуры данных для хранения матрицы в CRS формате и прототип функции умножения разреженной матрицы на векторы в файл **sparse.h** (реализация будет в **sparse.cpp**).

Последовательная реализация. Основная функция

- Через аргумент командной строки будем передавать порядок матрицы и количество ненулевых элементов в каждой строке матрицы A, а также в качестве необязательных параметров – имена файлов с матрицей и вектором, чтобы повторную генерацию заменить чтением из файлов. Изучим код в среде MS VS.
- Логика работы основной функции достаточно простая:
  - Генерация или считывание из файлов матрицы и вектора.
  - Выполнение умножения полученной матрицы на вектор с помощью разрабатываемой версии и эталонной версии MKL.
  - Сравнение результатов выполнения операции.

Последовательная реализация. Функция умножения разреженной матрицы на вектор (1)

- Добавим последовательную реализацию умножения разреженной матрицы в формате CRS на плотный вектор в файл **sparse.cpp**.
- Объявление соответствующей функции разместим в файле **sparse.h**.

Последовательная реализация. Функция умножения разреженной матрицы на вектор (2)

```
int Multiplicate(crsMatrix A, double *x, double *b,
  double &time)
{
  clock_t start, finish;
  start = clock();
  for (int i = 0; i < A.N; i++)
  {
    b[i] = 0.0;
    for (int j=A.RowIndex[i]; j<A.RowIndex[i + 1]; j++)
      b[i] += A.Value[j] * x[A.Col[j]];
  }
  finish = clock();
  time = (double(finish - start)) / CLOCKS_PER_SEC;

  return 0;
}
```

Результаты экспериментов

- Запустим корректную реализацию на максимально возможном в выбранной инфраструктуре размере матрицы N = 200 000 с числом ненулевых элементов NZ = 5000.

Программная реализация

ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ С ИСПОЛЬЗОВАНИЕМ OPENMP

Параллельная реализация. Создание и настройка проекта

- Создадим в рамках решения **15_SparseMatrDenseVec** проект **02_OpenMP**.
- Добавим в него файлы с именами, идентичными тем, что были созданы в предыдущем разделе, и скопируем в них исходный код, разработанный ранее.
- В настройках проекта подключите возможность использования технологии OpenMP.
  - В дереве **Configuration Properties** перейдите к разделу **C/C++→Language** и в поле **OpenMP Support** справа выберите вариант: **Generate Parallel Code (/openmp, equiv. to /Qopenmp)**.

Параллельная реализация (1)

- Распараллелим цикл с известным числом повторений с помощью директивы **#pragma omp parallel for**.
- Дополнительно установим опцию **num_threads**, чтобы можно было управлять количеством создаваемых потоков и в дальнейшем провести анализ масштабируемости.

```
int Multiplicate(crsMatrix A, double *x, double *b,
  int numThreads, double &time) {
  …
  #pragma omp parallel for num_threads(numThreads)
  for (int i = 0; i < A.N; i++)
  {
      …
  }
  …
```

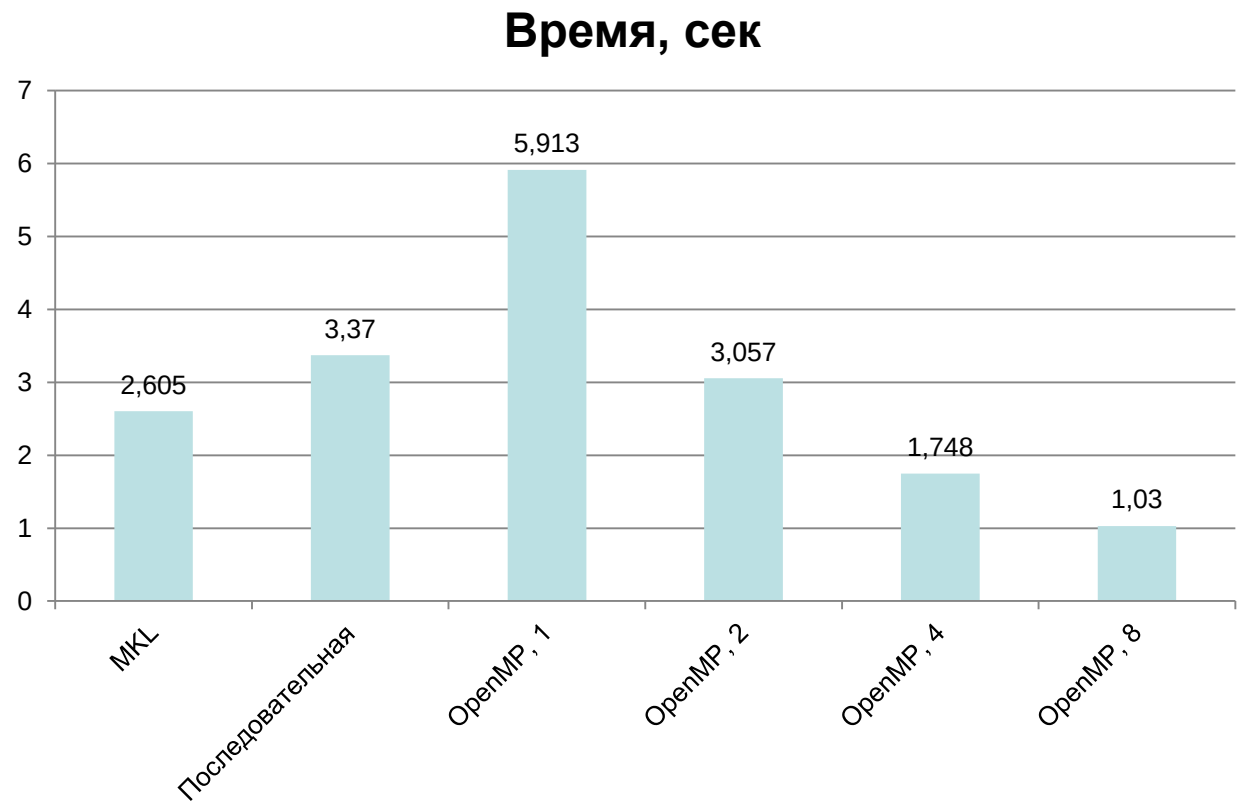
Параллельная реализация (2)

- Внесем необходимые изменения в основную функцию.

```
int main(int argc, char *argv[]) {
  if (argc < 4)
  {
    printf("Invalid input parameters\n");
    return 1;
  }
  int N = atoi(argv[1]);
  int NZ = atoi(argv[2]);
  int numThreads = atoi(argv[3]);
  ...
  Multiplicate(A, x, b, numThreads, timeM);
  ...
  return 0;
}
```

Результаты экспериментов

- Запустим корректную реализацию на максимально возможном в выбранной инфраструктуре размере матрицы N = 200 000 с числом ненулевых элементов NZ = 5000.



Программная реализация

ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ С ИСПОЛЬЗОВАНИЕМ БИБЛИОТЕКИ INTEL® TBB

Параллельная версия. Создание проекта

- В рамках решения **15_SparseMatrDenseVec** создадим проект **03_TBB**. Затем добавим в него файлы аналогичные тем, что были созданы при разработке предыдущей реализации, и скопируем в них исходный код, разработанный в последовательной версии.

Параллельная версия. Подключение библиотеки TBB к проекту в MS VS 2008

- Установить путь до заголовочных файлов библиотеки TBB (**Configuration Properties→C/C++→General→ Additional Include Directories**).
- Установить путь до **.lib** файлов (**Configuration Properties→Linker →General→Additional Library Directories**).
- Прописать название библиотеки **tbb.lib**, с которой будет собираться проект (**Configuration Properties→Linker→ Input→Additional Dependencies**).

Параллельная версия. Подключение заголовочных файлов библиотеки TBB

- В файле с объявлением функции умножения (**sparse.h**) укажите необходимые заголовочные файлы библиотеки, в котором определена функция **parallel_for**, одномерное итерационное пространство и функция динамического распределения нагрузки **tbb::affinity_partitioner()**.

```
#include "tbb/parallel_for.h"
#include "tbb/blocked_range.h"
#include "tbb/partitioner.h"
```

Параллельная версия (1)

- Будем использовать схему распараллеливания по строкам матрицы (соответствует внешнему циклу последовательной реализации).

- Для распараллеливания циклов с известным числом итераций используется функция **tbb::parallel_for**, которой в качестве входного параметра необходимо передавать объект некоторого класса-функтора и итерационное пространство.
- В качестве итерационного пространства будем использовать стандартное одномерное итерационное пространство, реализованное в библиотеке TBB.

Параллельная версия (2)

- Полями класса-функтора будут являться матрица в CRS-формате, вектор, на который умножается матрица, и результирующий вектор.
- В классе необходимо реализовать конструктор и метод **operator()**, который принимает на вход итерационное пространство. Ниже представлена программная реализация класса-функтора.

Параллельная версия. Реализация класса-функтора (1)

```
class MultiplicateFunctor
{
private:
  crsMatrix A;
  double *x;
  double *b;
public:
  MultiplicateFunctor(const crsMatrix _A,
    double *_x, double *_b)
  {
    A = _A;
    x = _x;
    b = _b;
  }
…
```

Параллельная версия. Реализация класса-функтора (2)

```
void operator()(const tbb::blocked_range<int> &r) const
  {
    int i, j;
    for (i = r.begin(); i < r.end(); i++)
    {
      b[i] = 0.0;
      for (j=A.RowIndex[i]; j<A.RowIndex[i + 1]; j++)
        b[i] += A.Value[j] * x[A.Col[j]];
    }
  }
};
```

Параллельная версия. Функция умножения разреженной матрицы на вектор

- Заменим вложенные циклы последовательной реализации вычисления произведения матрицы на вектор вызовом функции **tbb::parallel_for**.

```
int Multiplicate(crsMatrix A, double *x, double *b,
  double &time)
{
  …
  tbb::parallel_for<tbb::blocked_range<int>,
                    MultiplicateFunctor>(
                    tbb::blocked_range<int>(0, A.N),
                    MultiplicateFunctor(A, x, b),
                    tbb::affinity_partitioner());
  …
}
```

Параллельная версия. Инициализация и завершение библиотеки

```
...
#include "tbb/task_scheduler_init.h"
...
int main(int argc, char *argv[]) {
  ...
  int N = atoi(argv[1]);
  int NZ = atoi(argv[2]);
  int numThreads = atoi(argv[3]);
  ...
  tbb::task_scheduler_init
    init(tbb::task_scheduler_init::deferred);
  init.initialize(numThreads);
  Multiplicate(A, x, b, timeM);
  init.terminate();
  ...
  return 0;
}
```

Результаты экспериментов

- Запустим корректную реализацию на максимально возможном в выбранной инфраструктуре размере матрицы N = 200 000 с числом ненулевых элементов NZ = 5000.

Программная реализация

ПАРАЛЛЕЛЬНАЯ ВЕРСИЯ С ИСПОЛЬЗОВАНИЕМ INTEL® CILK PLUS

Параллельная версия. Создание проекта

- В рамках решения **15_SparseMatrDenseVec** создадим новый проект **04_Cilk**.
- Добавим в него файлы с такими же именами, что были созданы при разработке последовательной реализации, и скопируем в них исходный код, разработанный в последовательной версии.

Параллельная версия. Подключение возможности использования Cilk Plus

- Применить компилятор **Intel® Windows C/C++ Compiler**, входящий в состав **Intel® Parallel Composer XE 2011**:
  - В окне **Solution Explorer** выберите проект и выполните команду контекстного меню **Intel® Parallel Composer→Use Intel® C++…**.
  - В диалоговом окне **Confirmation** нажмите **OK**.
- В контекстном меню проекта в окне **Solution Explorer** выберите пункт **Properties** и выполните команду **Configuration Properties→C/C++→Language.** Для свойств **Disable Intel® Cilk Plus Keywords for Serial Semantics** и **Disable All Intel® Language Extensions** установите значение **No**.

Параллельная версия. Подключение заголовочных файлов

- В файле, содержащем прототип функции умножения (**sparse.h**) подключите заголовочный файл **cilk.h**.

```
#include <cilk/cilk.h>
```

Параллельная версия. Функция умножения разреженной матрицы на вектор

- Для распараллеливания циклов с известным количеством повторений в Cilk Plus используется синтаксическая конструкция **cilk_for**. По сути, параллельная реализация получается из последовательной заменой **for** на **cilk_for**.

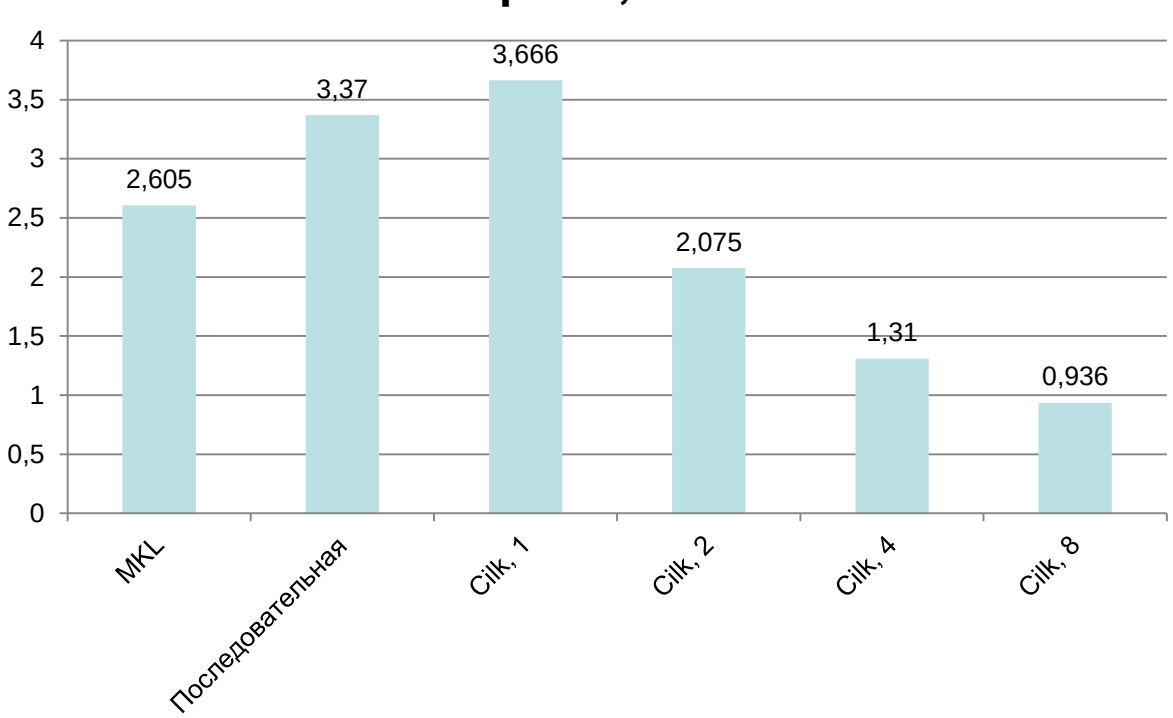
```
int Multiplicate(crsMatrix A, double *x, double *b,
  int numThreads, double &time) {
  …
  cilk_for (int i = 0; i < A.N; i++)
  {
      …
  }
  …
```

Параллельная версия. Установка количества обработчиков

```
#include <cilk/cilk_api.h>
...
int main(int argc, char *argv[]){
  ...
  int NZ = atoi(argv[2]);
  int numThreads = atoi(argv[3]);
  char *mtxFileName = NULL;
  char *vecFileName = NULL;
  if (argc > 4 && argc < 7)
  {
    …
  }
  ...
  char nt[3];
  itoa(numThreads, nt, 10);
  __cilkrts_set_param("nworkers", nt);
  InitializeVector(N, &b);
  ...
```

Результаты экспериментов

- Запустим корректную реализацию на максимально возможном в выбранной инфраструктуре размере матрицы N = 200 000 с числом ненулевых элементов NZ = 5000.



Программная реализация

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ С ИСПОЛЬЗОВАНИЕМ INTEL® ARBB

Создание проекта

- Как и ранее, перед разработкой реализации создадим новый проект **05_ArBB** в рамках решения **15_SparseMatrDenseVec**. Далее создадим файлы с именами, аналогичными тем, что создавались при разработке последовательной версии. Скопируем в них код последовательной реализации.

Подключение возможности использования библиотеки ArBB

- Установить путь до заголовочных файлов библиотеки ArBB (**Configuration Properties→C/C++→General→ Additional Include Directories**).
- Указать путь до **.lib** файлов (**Configuration Properties→Linker →General→Additional Library Directories**).
- Прописать название библиотеки **arbb.lib**, с которой должен собираться проект (**Configuration Properties→Linker→ Input→Additional Dependencies**).

Схема распараллеливания по данным в случае плотной матрицы

- Умножение плотной матрицы на вектор – это набор скалярных произведений строк матрицы на вектор.
- Скалярное произведение представляет собой покомпонентное умножение векторов с последующим суммированием.
- Таким образом, здесь явно выделяются две векторные операции – покомпонентное умножение векторов и суммирование компонент вектора.

- Замечание: умножение разреженной матрицы почти ничем не отличается от случая плотной.

Схема распараллеливания в случае разреженной матрицы (1)

- При реализации умножения вычисляется скалярное произведение не всей строки матрицы, а вектора, содержащего ненулевые элементы строки, на соответствующие компоненты заданного вектора.
- С другой стороны, т.к. используется CRS-формат для хранения разреженной матрицы, то множество ненулевых элементов представляется вектором.
- Преобразуем плотный вектор, на который умножается матрица, так, чтобы в нем были продублированы компоненты, соответствующие номерам столбцов, содержащих ненулевые элементы.

Схема распараллеливания в случае разреженной матрицы (2)

- Тогда операцию умножения матрицы на вектор можно рассматривать как скалярное произведение двух векторов с последующим суммированием частей вектора, которые отвечают строкам матрицы.
- Именно эту идею предлагается использовать при разработке параллельной реализации с помощью библиотеки ArBB.

Программная реализация. Структура хранения разреженной матрицы

- Добавим новую структуру данных **crsMatrixArBB** для хранения матрицы в формате CRS, в которой используются типы данных библиотеки ArBB. Аналог структуры **crsMatrix**, в которой отсутствуют значения числа строк и столбцов матрицы, т.к. их всегда можно получить с помощью метода **length()**.

```
struct crsMatrixArBB
{
  // Массив значений (размер NZ)
  dense<f64> Value;
  // Массив номеров столбцов (размер NZ)
  dense<i32> Col;
  // Массив индексов строк (размер N + 1)
  dense<i32> RowIndex;
};
```

Программная реализация. ArBB-функция умножения разреженной матрицы на вектор (1)

- Входные параметры функции:
  - матрица типа **crsMatrixArBB**,
  - плотный вектор **x**,
  - вектор **b** для записи результата.

- Код ArBB-функции содержит вызов только четырех функций.

Программная реализация. ArBB-функция умножения разреженной матрицы на вектор (2)

- Дублирование компонент вектора **x** согласно номерам столбцов, в которых расположены ненулевые элементы строк. Для этого используется функция **gather**.
- Умножение вектора ненулевых элементов матрицы и вектора с продублированными компонентами.
- Преобразование регулярного контейнера в нерегулярный с использованием функции **reshape_nested_offsets** – разбиение регулярного контейнера на вектора разной длины, которая соответствует числу ненулевых элементов в каждой строке матрицы.
- Суммирование элементов каждого вектора, входящего в состав нерегулярного контейнера с помощью функции **add_reduce**.

Программная реализация. ArBB-функция умножения разреженной матрицы на вектор (3)

```
void ArBBMultiplicate(crsMatrixArBB A,
                      dense<f64> x, dense<f64> &b)
{
  dense<f64> x_arbb = gather(x, A.Col);
  x_arbb = x_arbb * A.Value;
  nested<f64> row_blocks =
          reshape_nested_offsets(x_arbb, A.RowIndex);
  b = add_reduce(row_blocks);
}
```

Программная реализация. Функции преобразования

- Разработаем методы, которые преобразуют матрицу из **crsMatrix** в **crsMatrixArBB** и вектор из **double*** в **dense<f64>**.
- Для этого реализуем функции, которые привязывают регулярные контейнеры к соответствующим областям памяти с помощью функции **bind()**.
- Прототипы функций имеют вид:

```
int crsMatrix2crsMatrixArBB(crsMatrix A,
                            crsMatrixArBB &B);

int dArray2densef64(double *a, int n, dense<f64> &b);
```

Программная реализация. Функция умножения разреженной матрицы на вектор (1)

- Фактически в данной функции выполняется лишь преобразование матрицы и векторов к типам ArBB, а также вызов ArBB-функции с помощью механизма **call()**.

```
int Multiplicate(crsMatrix A, double *x, double *b,
  double &time)
{
  int ret_code;
  clock_t start, finish;
  crsMatrixArBB A_arbb;
  dense<f64> b_arbb, x_arbb;

  crsMatrix2crsMatrixArBB(A, A_arbb);
  dArray2densef64(b, A.N, b_arbb);
  dArray2densef64(x, A.N, x_arbb);
```

Программная реализация. Функция умножения разреженной матрицы на вектор (1)

```
  start = clock();
  try  {
    call(ArBBMultiplicate)(A_arbb, x_arbb, b_arbb);
    ret_code = 0;
  }
  catch (const std::exception& e)  {
    printf("%s\n", e.what());
    ret_code = -1;
  }
  catch (...)  {
    printf("Unknown error\n");
    ret_code = -1;
  }
  finish = clock();
  time = (double(finish - start)) / CLOCKS_PER_SEC;
  return ret_code;
}
```

Результаты экспериментов

- Эксперименты показали, что максимальный размер матрицы, на котором она успешно отрабатывает, составляет N=60 000 при количестве ненулевых элементов NZ=500. При бóльших размерах возникают проблемы работы с памятью внутри библиотеки. По словам разработчиков библиотеки ArBB проблемы будет исправлены по мере перехода от Beta-версии к релизу продукта.

Дополнительные задания (1)

- Выполните параллельную реализацию алгоритма умножения плотных матриц с использованием технологии OpenMP, Cilk и TBB.
- Напишите две реализации скалярного умножения плотных векторов с использованием функций call и map библиотеки ArBB.
- Реализуйте алгоритм умножения плотных матриц с помощью библиотеки ArBB. Примечание: рассмотрите задачу умножения матриц как многократное вычисление скалярных произведений строк первой матрицы на столбцы второй.

Дополнительные задания (2)

- Оцените число кэш-промахов с ростом числа потоков в OpenMP-реализации с целью выяснения причин нелинейной масштабируемости. Воспользуйтесь инструментом Intel Parallel Studio XE.
- Объясните причину отсутствия линейного ускорения параллельных реализаций, разработанных с помощью OpenMP, Cilk и TBB.
- Разработайте генератор матриц другой структуры (например, с нарастающим числом ненулевых элементов) и проведите эксперименты со всеми разработанными реализациями.

Вопросы

- ???

Авторский коллектив

- Кустикова Валентина Дмитриевна,
  ассистент кафедры
  Математического обеспечения ЭВМ факультета ВМК ННГУ.
  valentina.kustikova@gmail.com
- Мееров Иосиф Борисович,
  к.т.н., доцент, зам. зав. кафедры
  Математического обеспечения ЭВМ факультета ВМК ННГУ.
  meerov@vmk.unn.ru
- Сысоев Александр Владимирович,
  ассистент кафедры
  Математического обеспечения ЭВМ факультета ВМК ННГУ.
  sysoyev@vmk.unn.ru