



Нижегородский государственный университет им. Н.И. Лобачевского

Параллельные численные методы

Лабораторная работа Разреженное матричное умножение

При поддержке компании Intel

Мееров И.Б., Сысоев А.В.
Кафедра математического обеспечения ЭВМ
При участии Я. Сафоновой

Содержание

- ❑ Введение
- ❑ Цель работы
- ❑ Тестовая инфраструктура
- ❑ Постановка задачи
- ❑ Форматы хранения
- ❑ Генерация тестовых задач
- ❑ Программная реализация
- ❑ Задания для самостоятельной работы
- ❑ Литература



1. Приступаем к работе



Введение. Алгебра разреженных матриц

- **Алгебра разреженных матриц (*Sparse algebra*)** – важный раздел математики, имеющий очевидное практическое применение.
- Разреженные матрицы возникают естественным образом при постановке и решении задач из разных научных и инженерных областей:
 - При постановке и решении оптимизационных задач.
 - При численном решении дифференциальных уравнений в частных производных.
 - В теории графов.
 - Есть и другие случаи использования (см., например, обзор в *Тьюарсон Р. Разреженные матрицы. – М.: Мир, 1977*).



Введение. Понятие разреженной матрицы...

- ❑ Встречаются разные формулировки:
 - Разреженной называют матрицу, имеющую малый процент ненулевых элементов.
 - Матрица размера $N \times N$ называется разреженной, если количество ее ненулевых элементов есть $O(N)$.
 - Известны и другие определения.
- ❑ Приведенные варианты являются не вполне точными в математическом смысле.
- ❑ На практике классификация матрицы зависит не только от количества ненулевых элементов.



Введение. Понятие разреженной матрицы...

- ❑ На практике классификация матрицы зависит не только от количества ненулевых элементов, но и от:
 - размера матрицы,
 - распределения ненулевых элементов,
 - архитектуры конкретной вычислительной системы и используемых алгоритмов (см., например, более подробное обоснование в *Писсанецки С. Технология разреженных матриц. — М.: Мир, 1988*).
- ❑ Важно не просто дать матрице звучное название: разреженная или плотная. Важно определиться с выбором структуры хранения.



Введение. Понятие разреженной матрицы

- ❑ В ряде случаев матрица имеет регулярную структуру (например, ленточная матрица), что позволяет разработать специализированную структуру хранения.
- ❑ Иногда размерность матрицы такова, что ее плотное представление попросту не укладывается в память.
- ❑ Если и разреженное, и плотное представление допустимы, а количество ненулевых элементов невелико, необходимо проанализировать, какая структура хранения будет более эффективна в рамках используемых алгоритмов и конкретной архитектуры.
- ❑ В зависимости от результатов этого анализа можно рассматривать матрицу либо как разреженную, либо как плотную.



Введение. Сложность задачи

- ❑ Эффективные методы хранения и обработки разреженных матриц на протяжении последних десятилетий вызывают интерес у широкого круга исследователей.
- ❑ Для учета разреженной структуры приходится существенно усложнять как методы хранения, так и алгоритмы обработки.
- ❑ Многие тривиальные с точки зрения программирования алгоритмы в разреженном случае становятся весьма сложными.
- ❑ Для оптимизации производительности приходится разрабатывать нетривиальные алгоритмы и использовать возможности современной аппаратуры.



Цель работы...

Изучение некоторых принципов хранения и алгоритмов обработки разреженных матриц.

- ❑ Изучение способов хранения разреженных матриц.
- ❑ Разработка программной инфраструктуры для проведения экспериментов – генератора тестовых задач, автоматизации анализа корректности путем сравнения с эталонной версией из библиотеки Intel Math Kernel Library (MKL) и др.
- ❑ Разработка «наивной» программной реализации (по определению) разреженного матричного умножения.



Цель работы

Изучение некоторых принципов хранения и алгоритмов обработки разреженных матриц.

- ❑ Демонстрация алгоритмической оптимизации в разреженном матричном умножении.
- ❑ Ознакомление с идеологией двухфазной обработки разреженных матриц – разделением на *символическую* и *численную фазу*. Выполнение соответствующей программной реализации.
- ❑ Распараллеливание разреженного матричного умножения в системах с общей памятью с использованием OpenMP и Intel Threading Building Blocks (TBB).
- ❑ Изучение способов балансировки нагрузки с использованием OpenMP и TBB.



2. Постановка задачи



Постановка задачи

- Пусть A и B – квадратные матрицы размера $N \times N$, в которых процент ненулевых элементов мал (уточним позже).
- Будем считать, что элементы матриц A и B – вещественные числа (double).
- Требуется найти матрицу $C = A * B$, где символ $*$ соответствует матричному умножению.
- Нередко в процессе умножения двух разреженных матриц получается плотная матрица. В рамках данной работы предполагается, что матрица C также является разреженной.



Пример: $C = A * B$

A

1				2	
		3	4		
			2		5
	7				

*

B

3					1
		1			
			3		
1		3			
	2			4	

=

C

5		6			1
			12		
	10		6	20	
		7			

3. Тестовая инфраструктура



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
Математическая библиотека	Intel MKL v. 10.2.5.035



4. Форматы хранения



Форматы хранения. Общие замечания

- ❑ Изучению способов хранения разреженных матриц посвящено немало литературы, например:
 - Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений. – М.: Мир, 1984.
 - Писсанецки С. Технология разреженных матриц. — М.: Мир, 1988.
 - Тьюарсон Р. Разреженные матрицы. – М.: Мир, 1977.
- ❑ В лабораторной работе мы изучим несколько широко распространенных форматов хранения, в том числе поддерживаемых математическими библиотеками и пакетами (Intel MKL, PETSc, Matlab и др.).
- ❑ При описании мы будем ориентироваться на матрицы размера $N \times N$, в которых NZ элементов являются ненулевыми, $NZ \ll N^2$.



Форматы хранения. Координатный формат...

- ❑ Один из наиболее простых для понимания форматов хранения разреженных матриц – координатный формат. Элементы матрицы и ее структура хранятся в трех массивах, содержащих значения, их X и Y координаты.
- ❑ Оценим объем необходимой памяти (M):
 - Плотное представление: $M = 8 N^2$ байт.
 - В координатном формате:
 $M = 8 NZ + 4 NZ + 4 NZ = 16 NZ$ байт (предполагается, что $N < 2^{32}$; в противном случае необходимо использовать 64-битный вариант беззнакового целого для хранения индексов).
 - Ясно, что $16 NZ \ll 8 N^2$, если $NZ \ll N^2$.



Форматы хранения. Координатный формат...

Пример:

A

1				2	
		3	4		
			8		5
	7	1			6

Структура хранения:

1	2	3	4	8	5	7	1	6
---	---	---	---	---	---	---	---	---

Value

0	0	1	1	3	3	5	5	5
---	---	---	---	---	---	---	---	---

Row

0	4	2	3	3	5	1	2	5
---	---	---	---	---	---	---	---	---

Col

Форматы хранения. Координатный формат

- ❑ Координатный формат можно использовать
 - как в упорядоченном виде, когда в массиве **Value** хранятся элементы матрицы построчно (например, слева направо, сверху вниз),
 - так и в неупорядоченном.
- ❑ Неупорядоченное представление существенно упрощает операции вставки/удаления новых элементов, но приводит к переборным поискам.
- ❑ Упорядоченный вариант позволяет быстрее находить все элементы нужной строки (столбца), но приводит к перепаковкам при вставках/удалениях элементов.



Форматы хранения. Формат CRS (CSR)...

- ❑ Формат хранения CSR (Compressed Sparse Rows) или CRS (Compressed Row Storage), призван устранить некоторые недоработки координатного представления.
- ❑ Так, по-прежнему 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 $N + 1$.

Форматы хранения. Формат CRS (CSR)...

- ❑ 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.
- ❑ Т.о. обрабатывается случай пустых строк, а также добавляется «лишний» элемент в массив **RowIndex** – устраняется особенность при доступе к элементам последней строки. **RowIndex**[N] = NZ .



Форматы хранения. Формат CRS (CSR)...

- 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.

Форматы хранения. Формат CRS (CSR)

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

Форматы хранения. Модификации CRS...

- ❑ Возможна индексация как с нуля, так и с единицы (C/Fortran).
- ❑ В базовом варианте CRS строки рассматриваются по порядку, но элементы внутри каждой строки могут быть как упорядочены по номеру столбца, так и не упорядочены. Далее в работе мы будем использовать упорядоченный вариант.
- ❑ Модификация CRS с четырьмя массивами. Четвертый массив хранит индексы элементов, идущих в конце строки. Строки могут не быть упорядочены – быстрая перестановка.



Форматы хранения. Модификации CRS

- ❑ В некоторых источниках рассмотренный формат упоминается как Yale format, в некоторых – как AIJ.
- ❑ Иногда применяется модификация, состоящая в том, что массивы хранятся в памяти в другом порядке.
- ❑ Все, что было описано выше, может быть с равным успехом применено не к строкам, а к столбцам. В итоге получается столбцовый формат CSC (CCS) вместо строчного CSR (CRS).
- ❑ Для ряда алгоритмов более удобен строчный формат, для ряда – столбцовый.



Форматы хранения. Работа с CRS

Пример: $y = Ax$.

$x[Col[j]]$ – кеш-промахи (см. работы Дж. Деммель и др.)

```
// Цикл по строкам матрицы A
for (i = 0; i < N; i++)
{
    // Вычисляем i-ю компоненту вектора y
    sum = 0;
    j1 = RowIndex[i]; j2 = RowIndex[i + 1];
    for (j = j1; j < j2; j++)
        sum = sum + Value[j] * x[Col[j]];
    y[i] = sum;
}
```

Форматы хранения. Заключение...

- ❑ Операции вставки/удаления элементов приводят к перепакеткам, что является недостатком рассмотренного представления.
- ❑ Известны также диагональный формат хранения, разные варианты блочного представления – разреженная структура из плотных блоков (BCRS), плотная структура из разреженных блоков и т.д.
- ❑ Далее в работе мы будем использовать формат CRS в виде трех массивов, расположенных в памяти в порядке **Value, Col, RowIndex**.



Форматы хранения. Заключение

- ❑ 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;
};
```

5. Генерация тестовых задач



Генерация тестовых задач...

- ❑ 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 выбрать ограничиться некоторым классом (классами) матриц.
 - Не претендуем на лучшее время работы.
 - По возможности ориентируемся на время работы аналогичных программ в индустриальных библиотеках.



Генерация тестовых задач...

- ❑ Пусть матрицы A и B содержат не более 1% ненулевых элементов.
- ❑ На заполнение результирующей матрицы C в экспериментах будем обращать меньшее внимание, но позаботимся о том, чтобы оно не превышало 10% (не совсем разреженная матрица, но подойдет для иллюстрации работы алгоритма).
- ❑ Будем формировать матрицы A и B при помощи датчика случайных чисел. Данная задача включает два этапа:
 - построение *портрета (шаблона) матрицы*,
 - и наполнение этого портрета конкретными значениями.



Генерация тестовых задач...

- Будем формировать матрицы A и B 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 тестовых задач...

- ❑ Пусть $N = 10000$. Для формирования портрета матрицы A применим следующую схему: будем генерировать случайным образом позиции 50 ненулевых элементов в каждой строке матрицы.
- ❑ Для формирования портрета матрицы B применим другой подход: будем наращивать количество ненулевых элементов от строки к строке по кубическому закону так, чтобы последняя строка содержала максимальное количество (50) ненулевых элементов.
- ❑ Размер матрицы и количество ненулевых элементов параметризуем.
- ❑ В итоге необходимы 2 генератора для матриц разных классов.



Генерация тестовых задач...

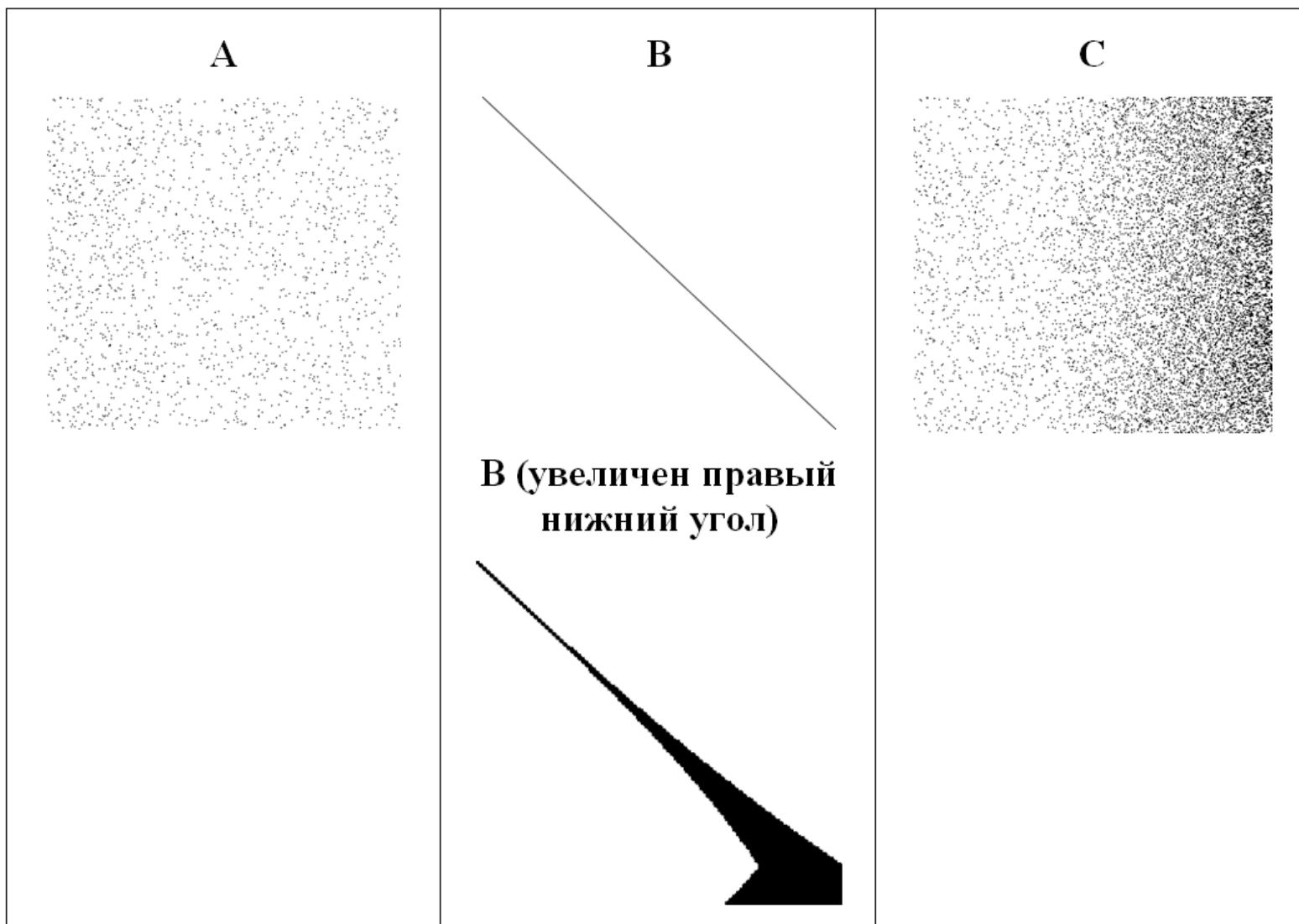
```
// Генерирует квадратную матрицу в формате CRS
// (3 массива, индексация с нуля)
// В каждой строке cntInRow ненулевых элементов
void GenerateRegularCRS(int seed, int N,
                       int cntInRow, crsMatrix& mtx);
```

```
// Генерирует квадратную матрицу в формате CRS
// (3 массива, индексация с нуля)
// Число ненулевых элементов в строках растёт
// от 1 до cntInRow. Закон роста – кубическая парабола
void GenerateSpecialCRS(int seed, int N,
                       int cntInRow, crsMatrix& mtx);
```

Генерация тестовых задач. Тестовые матрицы

Матрица	Количество ненулевых элементов	Процент ненулевых элементов
A	500 000	0,5%
B	130 775	0,13%
C	5 681 531	5,68%

Генерация тестовых задач. Тестовые матрицы



6. Вспомогательные функции



Вспомогательные функции...

- ❑ Инициализация матрицы
- ❑ Удаление матрицы
- ❑ Сравнение двух разреженных матриц в формате CRS
- ❑ Умножение двух разреженных матриц в формате CRS – эталонная версия

Вспомогательные функции...

Инициализация матрицы

```
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...

Удаление матрицы

```
void FreeMatrix(crsMatrix &mtx)
{
    delete[] mtx.Value;
    delete[] mtx.Col;
    delete[] mtx.RowIndex;
}
```

Вспомогательные функции...

- ❑ Сравнение двух разреженных матриц в формате CRS.
- ❑ **Вариант 1.** Копируем обе матрицы в плотные, после чего сравниваем поэлементно, возвращаем максимальную разность значений соответствующих элементов (diff).
- ❑ Return value – совпал ли размер.

```
int CompareMatrix(crsMatrix mtx1, crsMatrix mtx2,  
                  double &diff)
```

- ❑ Проблема – долго, может не хватить памяти.

Вспомогательные функции...

- ❑ Сравнение двух разреженных матриц в формате CRS.
- ❑ **Вариант 2.** Необходимо реализовать вычитание разреженных матриц, далее найти максимум невязки.
- ❑ Будем использовать **mkl_dcsradd()** из Intel MKL.
- ❑ **mkl_dcsradd()** выполняет сложение двух разреженных матриц: $C = A + \text{beta} * \text{op}(B)$, где
 - **beta** – вещественный множитель (в нашем случае -1),
 - параметр **op** позволяет указать, что матрица **B** должна быть транспонирована (в нашем случае это не требуется).
- ❑ **mkl_dcsradd()** работает только с 1-based CRS, реализуем преобразование 0-based ↔ 1-based.



Вспомогательные функции...

```
// Принимает 2 квадратных матрицы в формате CRS
//      (3 массива, индексация с нуля)
// Возвращает  $\max|C_{ij}|$ , где  $C = A - B$ 
// Для сравнения использует функцию из MKL
// Возвращает признак успешности операции: 0 – OK,
//      1 – не совпадают размеры (N)
int SparseDiff(crsMatrix A, crsMatrix B, double &diff)
```



```
int SparseDiff(crsMatrix A, crsMatrix B, double &diff)
{
    if (A.N != B.N)
        return 1;
    int n = A.N;
    // Будем вычислять  $C = A - B$ , используя MKL
    // Структуры данных в стиле MKL
    double *c = 0; // Значения
    int *jc = 0;    // Номера столбцов (нумерация с единицы)
    int *ic;        // Индексы первых элементов строк
                    // (нумерация с единицы)

    // Настроим параметры для вызова функции MKL
    // Переиндексируем матрицы A и B с единицы
    int i, j;
    for (i = 0; i < A.NZ; i++)
        A.Col[i]++;
    for (i = 0; i < B.NZ; i++)
        B.Col[i]++;
    for (j = 0; j <= A.N; j++)
    {
        A.RowIndex[j]++;
        B.RowIndex[j]++;
    }
}
```

```
// Используется функция, вычисляющая  $C = A + \text{beta} * \text{op}(B)$ 
char trans = 'N'; // говорит о том,  $\text{op}(B) = B -$ 
                  // не нужно транспонировать B
// Параметр, влияющий на то, как будет выделяться память
// request = 0: память для результирующей матрицы должна
//             быть выделена заранее
// Если мы не знаем, сколько памяти необходимо для хранения
// результата, необходимо:
// 1) выделить память для массива индексов строк ic:
//     "Кол-во строк + 1" элементов;
// 2) вызвать функцию с параметром request = 1 -
//     в массиве ic будет заполнен последний элемент
// 3) выделить память для массивов c и jc:
//     кол-во элементов = ic[Кол-во строк] - 1
// 4) вызвать функцию с параметром request = 2
int request;
// Есть возможность настроить, нужно ли упорядочивать матрицы A, B, C.
// У нас предполагается, что все матрицы упорядочены,
// следовательно, выбираем вариант "No-No-Yes", который
// соответствует любому значению, кроме целых чисел от 1 по 7
int sort = 8;
```

```
// beta = -1 -> C = A + (-1.0) * B;
double beta = -1.0;
// Количество ненулевых элементов
// Используется только если request = 0
int nzmax = -1;
// Служебная информация
int info;
// Выделим память для индекса в матрице C
ic = new int[n + 1];
// Сосчитаем количество ненулевых элементов в матрице C
request = 1;
mkl_dcsradd(&trans, &request, &sort, &n, &n,
            A.Value, A.Col, A.RowIndex, &beta,
            B.Value, B.Col, B.RowIndex,
            c, jc, ic,
            &nzmax, &info);
int nzc = ic[n] - 1; c = new double[nzc]; jc = new int[nzc];
// Сосчитаем C = A - B
request = 2;
mkl_dcsradd(&trans, &request, &sort, &n, &n,
            A.Value, A.Col, A.RowIndex, &beta,
            B.Value, B.Col, B.RowIndex,
            c, jc, ic, &nzmax, &info);
```

```

// Сосчитаем max|Cij|
diff = 0.0;
for (i = 0; i < nzc; i++)
{
    double var = fabs(c[i]);
    if (var > diff)
        diff = var;
}
// Приведем к исходному виду матрицы A и B
for (i = 0; i < A.NZ; i++)
    A.Col[i]--;
for (i = 0; i < B.NZ; i++)
    B.Col[i]--;
for (j = 0; j <= n; j++)
{
    A.RowIndex[j]--;
    B.RowIndex[j]--;
}
// Освободим память
delete [] c;
delete [] ic;
delete [] jc;
return 0;
}

```


Вспомогательные функции...

- ❑ Умножение двух разреженных матриц в формате CRS – эталонная версия.
- ❑ Для проверки собственных реализаций алгоритма умножения разреженных матрицы необходимо обеспечить контроль за правильностью результата.
- ❑ Будем использовать функцию **`mkl_dcsrcmultcsr()`** в качестве эталона.
- ❑ В целом она весьма похожа по своему интерфейсу на функцию **`mkl_dcsrcadd()`** и имеет те же особенности (упорядоченный формат хранения в исходных матрицах, нумерация с единицы, двухуровневый механизм работы, связанный с необходимостью выделения памяти и др.).



Вспомогательные функции...

- Изучим код в среде MS Visual Studio.

```
// Принимает 2 квадратных матрицы в формате CRS
//      (3 массива, индексация с нуля)
// Возвращает C = A * B, C - в формате CRS
//      (3 массива, индексация с нуля)
// Память для C в начале считается не выделенной
// Возвращает признак успешности операции:
//      0 - ОК, 1 - не совпадают размеры (N)
// Возвращает время работы
int SparseMKLMult(crsMatrix A, crsMatrix B, crsMatrix &C,
                 double &time)
```

7. Программная реализация



Наивная последовательная версия...

- Попробуем выполнить умножение по определению.

A						B						C					
X				X		X		X		X		X		X		X	
			X	X				X					X				
							X										
				X												X	
										X							
	X	X			X						X		X	X			X

- Проблемы:

- Необходимо вычислять скалярные произведения строки одной матрицы и столбца другой матрицы.
- При вычислении скалярных произведений в строчном формате неудобно выделять столбец (в столбцовом – строку).

Наивная последовательная версия...

- ❑ Возможное решение проблемы – транспонировать матрицу B (или хранить дополнительно транспонированный портрет). Есть и другие варианты (см., например, алгоритм Густавсона).
- ❑ Далее в работе будем использовать транспонирование матрицы B .
- ❑ Дополнительная сложность:
 - Формирование разреженного представления матрицы C – необходимо избежать перепаковок.
- ❑ Решение: будем умножать каждую строку матрицы A на все столбцы матрицы B^T . Тогда матрица C заполняется последовательно.

Наивная последовательная версия...

- Подведем промежуточные итоги. Необходимо сделать следующее:
 - Реализовать транспонирование разреженной матрицы и применить его к матрице B .
 - Инициализировать структуру данных для матрицы C , обеспечить возможность пополнения ее элементами.
 - Последовательно перемножить каждую строку матрицы A на каждую из строк матрицы B^T , записывая в C полученные результаты и формируя ее структуру. При умножении строк реализовать сопоставление с целью выделения пар ненулевых элементов.



Наивная последовательная версия...

□ Важное замечание:

- В процессе вычисления скалярного произведения в точной арифметике может получиться нуль, не только в том случае, когда при сопоставлении векторов соответствующих друг другу пар не обнаружилось, но и просто как естественный результат.
- В арифметике с плавающей запятой нуль может получиться еще и в связи с ограничениями на представление чисел и погрешностью вычислений (см. стандарт IEEE-754).
- Нули, получившиеся в процессе вычислений, можно как хранить в матрице C , так и не хранить. Оба варианта применяются на практике.



Наивная последовательная версия...

□ Алгоритм транспонирования.

- Если создать нулевую матрицу, а далее добавлять туда по одному элементу, выбирая их из CRS-структуры исходной матрицы, получим перепакровки.
- Нужно формировать транспонированную матрицу построчно. Для этого можно брать столбцы исходной матрицы и создавать из них строки результирующей матрицы.
- Выделить из CRS-матрицы столбец i не так просто. Необходимо другое решение.

Наивная последовательная версия...

- ❑ **Алгоритм транспонирования (Густавсон)**
- ❑ Сформируем N одномерных векторов для хранения целых чисел, а также N векторов для хранения вещественных чисел.
- ❑ В цикле посмотрим все строки исходной матрицы, для каждой строки – все ее элементы. Пусть текущий элемент находится в строке i , столбце j , его значение равно v . Тогда добавим числа i и v в j -ые вектора для хранения целых и вещественных чисел (соответственно). Тем самым в векторах мы сформируем строки транспонированной матрицы.
- ❑ Последовательно скопируем данные из векторов в CRS-структуру транспонированной матрицы (**Col** и **Value**), попутно формируя массив **RowIndex**.

см. Писсанецки С. Технология разреженных матриц. — М.: Мир, 1988.



Наивная последовательная версия...

A	Структура хранения A	Вектора																																													
<table><tr><td></td><td>3</td><td></td><td>7</td></tr><tr><td></td><td></td><td>8</td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>9</td><td></td><td>15</td><td>16</td></tr></table>		3		7			8						9		15	16	<table><tr><td>3</td><td>7</td><td>8</td><td>9</td><td>15</td><td>16</td></tr></table> <i>Value</i> <table><tr><td>1</td><td>3</td><td>2</td><td>0</td><td>2</td><td>3</td></tr></table> <i>Col</i> <table><tr><td>0</td><td>2</td><td>3</td><td>3</td><td>6</td></tr></table> <i>RowIndex</i>	3	7	8	9	15	16	1	3	2	0	2	3	0	2	3	3	6	<table><tr><td>3</td></tr><tr><td>0</td></tr><tr><td>1</td><td>3</td></tr><tr><td>0</td><td>3</td></tr></table> <i>IntVectors</i> <table><tr><td>9</td></tr><tr><td>3</td></tr><tr><td>8</td><td>15</td></tr><tr><td>7</td><td>16</td></tr></table> <i>DoubleVectors</i>	3	0	1	3	0	3	9	3	8	15	7	16
	3		7																																												
		8																																													
9		15	16																																												
3	7	8	9	15	16																																										
1	3	2	0	2	3																																										
0	2	3	3	6																																											
3																																															
0																																															
1	3																																														
0	3																																														
9																																															
3																																															
8	15																																														
7	16																																														
A ^T	Структура хранения A ^T																																														
<table><tr><td></td><td></td><td></td><td>9</td></tr><tr><td>3</td><td></td><td></td><td></td></tr><tr><td></td><td>8</td><td></td><td>15</td></tr><tr><td>7</td><td></td><td></td><td>16</td></tr></table>				9	3					8		15	7			16	<table><tr><td>9</td><td>3</td><td>8</td><td>15</td><td>7</td><td>16</td></tr></table> <i>Value</i> <table><tr><td>3</td><td>0</td><td>1</td><td>3</td><td>0</td><td>3</td></tr></table> <i>Col</i> <table><tr><td>0</td><td>1</td><td>2</td><td>4</td><td>6</td></tr></table> <i>RowIndex</i>	9	3	8	15	7	16	3	0	1	3	0	3	0	1	2	4	6													
			9																																												
3																																															
	8		15																																												
7			16																																												
9	3	8	15	7	16																																										
3	0	1	3	0	3																																										
0	1	2	4	6																																											

- ❑ Недостаток – использование доп. памяти.
- ❑ Устранение перепакровок: использование структур данных матрицы A^T для промежуточных результатов вычислений (см., например, реализацию в книге С. Писсанецки).

Наивная последовательная версия...

Шаг 1

```
memset(AT.RowIndex, 0, (N+1) * sizeof(int));  
for (i = 0; i < A.NZ; i++)  
    AT.RowIndex[A.Col[i] + 1]++;
```

A	Структура хранения A	Структура хранения A^T																																																																																								
<table><tr><td></td><td>3</td><td></td><td>7</td></tr><tr><td></td><td></td><td>8</td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>9</td><td></td><td>15</td><td>16</td></tr></table>		3		7			8						9		15	16	<table><tr><td>3</td><td>7</td><td>8</td><td>9</td><td>15</td><td>16</td></tr><tr><td colspan="6">$A.Value$</td></tr><tr><td>1</td><td>3</td><td>2</td><td>0</td><td>2</td><td>3</td></tr><tr><td colspan="6">$A.Col$</td></tr><tr><td>0</td><td>2</td><td>3</td><td>3</td><td>6</td><td></td></tr><tr><td colspan="6">$A.RowIndex$</td></tr></table>	3	7	8	9	15	16	$A.Value$						1	3	2	0	2	3	$A.Col$						0	2	3	3	6		$A.RowIndex$						<table><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td colspan="6">$AT.Value$</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td colspan="6">$AT.Col$</td></tr><tr><td>0</td><td>1</td><td>1</td><td>2</td><td>2</td><td></td></tr><tr><td colspan="6">$AT.RowIndex$</td></tr></table>							$AT.Value$												$AT.Col$						0	1	1	2	2		$AT.RowIndex$					
	3		7																																																																																							
		8																																																																																								
9		15	16																																																																																							
3	7	8	9	15	16																																																																																					
$A.Value$																																																																																										
1	3	2	0	2	3																																																																																					
$A.Col$																																																																																										
0	2	3	3	6																																																																																						
$A.RowIndex$																																																																																										
$AT.Value$																																																																																										
$AT.Col$																																																																																										
0	1	1	2	2																																																																																						
$AT.RowIndex$																																																																																										

Наивная последовательная версия...

Шаг 2

```
S = 0;  
for (i = 1; i <= A.N; i++)  
{  
    tmp = AT.RowIndex[i];  
    AT.RowIndex[i] = S;  
    S = S + tmp;  
}
```

A	Структура хранения A	Структура хранения A ^T																																																																																								
<table><tr><td></td><td>3</td><td></td><td>7</td></tr><tr><td></td><td></td><td>8</td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td>9</td><td></td><td>15</td><td>16</td></tr></table>		3		7			8						9		15	16	<table><tr><td>3</td><td>7</td><td>8</td><td>9</td><td>15</td><td>16</td></tr><tr><td colspan="6">A.Value</td></tr><tr><td>1</td><td>3</td><td>2</td><td>0</td><td>2</td><td>3</td></tr><tr><td colspan="6">A.Col</td></tr><tr><td>0</td><td>2</td><td>3</td><td>3</td><td>6</td><td></td></tr><tr><td colspan="6">A.RowIndex</td></tr></table>	3	7	8	9	15	16	A.Value						1	3	2	0	2	3	A.Col						0	2	3	3	6		A.RowIndex						<table><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td colspan="6">AT.Value</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td colspan="6">AT.Col</td></tr><tr><td>0</td><td>0</td><td>1</td><td>2</td><td>4</td><td></td></tr><tr><td colspan="6">AT.RowIndex</td></tr></table>							AT.Value												AT.Col						0	0	1	2	4		AT.RowIndex					
	3		7																																																																																							
		8																																																																																								
9		15	16																																																																																							
3	7	8	9	15	16																																																																																					
A.Value																																																																																										
1	3	2	0	2	3																																																																																					
A.Col																																																																																										
0	2	3	3	6																																																																																						
A.RowIndex																																																																																										
AT.Value																																																																																										
AT.Col																																																																																										
0	0	1	2	4																																																																																						
AT.RowIndex																																																																																										

Наивная последовательная версия...

Шаг 3

```
for (i = 0; i < A.N; i++)
{
    j1 = A.RowIndex[i]; j2 = A.RowIndex[i+1];
    Col = i;           // Столбец в AT - строка в A
    for (j = j1; j < j2; j++)
    {
        V = A.Value[j];    // Значение
        RIndex = A.Col[j]; // Строка в AT
        IIndex = AT.RowIndex[RIndex + 1];
        AT.Value[IIndex] = V;
        AT.Col[IIndex] = Col;
        AT.RowIndex[RIndex + 1]++;
    }
}
```



Наивная последовательная версия...

- ❑ Создадим пустой проект.
- ❑ Добавим файлы `sparse.h`, `sparse.c` – основные алгоритмы.
- ❑ Добавим файлы `util.h`, `util.c` – вспомогательные алгоритмы.
- ❑ Добавим файл `main_n.c` – головная программа.
- ❑ Функция `main()`:
 - 2 параметра: `N` и кол-во ненулевых элементов в строке.
- ❑ **`const double EPSILON = 0.000001;`**
 - Для автоматизированного сравнения с эталоном.
- ❑ Подключим к проекту MKL:
 - Linker input: **`mkl_core.lib mkl_intel_c.lib mkl_sequential.lib`**
- ❑ Далее фрагмент функции `main()`:



Наивная последовательная версия...

```
...
crsMatrix A, B, BT, C;
GenerateRegularCRS(1, N, NZ, A);
GenerateSpecialCRS(2, N, NZ, B);
double timeT = Transpose2(B, BT);
double timeM, timeM1;
Multiply(A, BT, C, timeM);
crsMatrix CM;
double diff;
SparseMKLMult(A, B, CM, timeM1);
SparseDiff(C, CM, diff);
if (diff < EPSILON)
    printf("OK\n");
else
    printf("not OK\n");
printf("%d %d %d %d\n", A.N, A.NZ, B.NZ, C.NZ);
printf("%.3f %.3f %.3f\n", timeT, timeM, timeM1);
...
```



Наивная последовательная версия...

- Инфраструктура готова.
 - Инициализация/удаление матриц.
 - Генерация матриц.
 - Сравнение матриц.
 - Транспонирование матриц.
 - Сравнение с эталоном (корректность).
 - Сравнение с эталоном (производительность).
 - Замеры времени.
 - Вывод на консоль ключевых параметров.
- **Далее:** *приступаем к реализации разных вариантов алгоритма умножения $C = A * B^T$ и их распараллеливанию.*

Наивная последовательная версия...

□ Алгоритм умножения:

- Создайте 2 вектора (**Value**, **Col**) и массив **RowIndex** длины $N + 1$ для хранения матрицы **C**. Размер векторов **Value** и **Col** одинаков, но пока не известен. Можно использовать динамически распределяемые вектора из библиотеки STL.
- ...

Наивная последовательная версия...

□ Алгоритм умножения:

- Создайте 2 вектора (**Value**, **Col**) и массив **RowIndex** длины $N + 1$ для хранения матрицы **C**. Размер векторов **Value** и **Col** одинаков, но пока не известен. Можно использовать динамически распределяемые вектора из библиотеки STL.
- Транспонируйте матрицу **B** (в нашем случае матрица уже транспонирована, то есть этот пункт можно пропустить).
- ???

Наивная последовательная версия...

□ Алгоритм умножения:

- ...
- В цикле по i от 0 до $N - 1$ перебирайте все строки матрицы A .
 - Для каждого i в цикле по j от 0 до $N - 1$ перебирайте все строки матрицы B^T .
 - Вычислите скалярное произведение векторов-строк A_i и B_j , пусть оно равно V . Если V отлично от нуля, добавьте в вектор **Value** элемент V , в вектор **Col** – элемент j . При окончании цикла по j , скорректируйте значение **RowIndex[i+1]**, записав туда текущее значение числа ненулевых элементов V .
- Вспомним про дилемму о нулевых результатах вычислений и их включении/исключении из структуры матрицы.



```

int Multiply(crsMatrix A, crsMatrix B, crsMatrix &C, double &time) {
    if (A.N != B.N) return 1;
    int N = A.N;
    vector<int> columns; vector<double> values; vector<int> row_index;
    clock_t start = clock();
    int rowNZ; row_index.push_back(0);
    for (int i = 0; i < N; i++) {
        rowNZ = 0;
        for (int j = 0; j < N; j++) {
            double sum = 0;
            // Считаем скалярное произведение строк A и BT
            ...
            if (fabs(sum) > ZERO_IN_CRS) {
                columns.push_back(j); values.push_back(sum); rowNZ++;
            }
        }
        row_index.push_back(rowNZ + row_index[i]);
    }
    InitializeMatrix(N, columns.size(), C);
    for (int j = 0; j < columns.size(); j++) {
        C.Col[j] = columns[j]; C.Value[j] = values[j];
    }
    for(int i = 0; i <= N; i++) C.RowIndex[i] = row_index[i];
    clock_t finish = clock();
    time = (double)(finish - start) / CLOCKS_PER_SEC;
    return 0;
}

```

Наивная последовательная версия...

- ❑ Скалярное произведение разреженных векторов (строк матрицы) – ядро алгоритма.
- ❑ Простейший вариант:

```
for (int k = A.RowIndex[i]; k < A.RowIndex[i + 1]; k++)  
    for (int l = B.RowIndex[j]; l < B.RowIndex[j + 1]; l++)  
        if (A.Col[k] == B.Col[l])  
        {  
            sum += A.Value[k] * B.Value[l];  
            break;  
        }
```

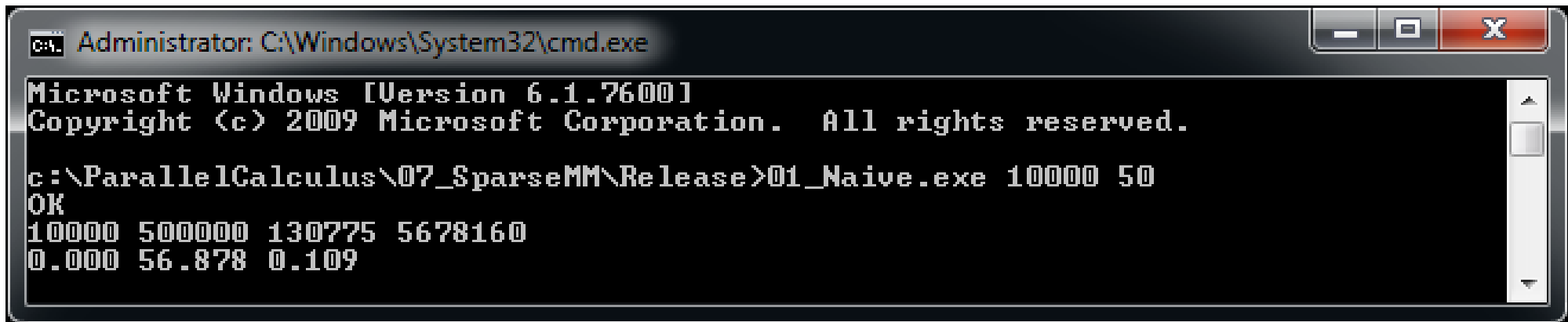
Наивная последовательная версия...

□ Задание

- Внесите все необходимые изменения в код.
- Перейдите к использованию Intel C++ Compiler из пакета инструментов Intel Parallel Studio XE.
- Добейтесь того, чтобы код компилировался, собирался и выдавал корректные результаты.

Наивная последовательная версия...

- Для сравнения представим результаты, полученные авторами на тестовой инфраструктуре, описанной ранее.



A screenshot of a Windows command prompt window titled "Administrator: C:\Windows\System32\cmd.exe". The window shows the output of a program execution. The text displayed is as follows:

```
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
c:\ParallelCalculus\07_SparseMM\Release>01_Naive.exe 10000 50  
OK  
10000 5000000 130775 5678160  
0.000 56.878 0.109
```

Наивная последовательная версия...

- ❑ Перемножаются матрицы 10000×10000 , матрица A имеет 50 ненулевых элементов в каждой строке, матрица B – от 1 в первых строках до 50 в последних.
- ❑ Программа печатает
 - размер матриц;
 - статистику по количеству ненулевых элементов в A , B и C ,
 - время транспонирования (0с – не хватило разрешающей способности датчика);
 - умножения текущей наивной реализации (56,878с) и эталонной версии из Intel MKL (0,109с);
 - «ОК» в первой строке вывода говорит о том, что результат прошел автоматизированный тест на корректность.

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 1...

- ❑ Мотивация для оптимизации: отставание от MKL – почти 3 порядка. **Надо что-то менять!!!**
- ❑ Попробуем понять, на что тратится время.
- ❑ Используем инструменты:
 - Профилировщик **Intel VTune Amplifier XE** из пакета **Intel Parallel Studio XE**.
- ❑ Запустим анализ. **Amplifier** позволяет получить первичные результаты профилировки в двух режимах – **Lightweight Hotspots** и **Hotspots**.
- ❑ В **Lightweight Hotspots** не собирается информация о стеке вызовов, выберем его (у нас одна вычислительная функция).



Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 1...

Lightweight Hotspots - Hotspots 🔧 ?

Analysis Target Analysis Type Summary Bottom-up sparse_n.cpp

/Function	CPU Time	Instructions Retired	CPI	Module
Multiply	71.981s	395,182,000,000	0.453	01_naive.exe
GenerateRegularCRS	0.111s	376,000,000	0.713	01_naive.exe
mkl_spblas_p4m3_dmcsr_notr	0.081s	232,000,000	0.991	01_naive.exe
mkl_spblas_p4m3_dmcsradd_notr	0.056s	302,000,000	0.450	01_naive.exe
SparseDiff	0.022s	44,000,000	1.273	01_naive.exe
mkl_spblas_p4m3_dsortrow	0.014s	80,000,000	0.250	01_naive.exe
SparseMKLMult	0.009s	30,000,000	0.667	01_naive.exe
GenerateSpecialCRS	0.002s	22,000,000	0.364	01_naive.exe
std::vector<int,class std::allocator<int> >::Insert_n	0.002s	4,000,000	1.500	01_naive.exe
Transpose2	0.002s	2,000,000	1.000	01_naive.exe
[Import thunk new]	0s	0	0.000	01_naive.exe
`eh vector constructor iterator'	0s	2,000,000	0.000	01_naive.exe
Selected 1 row(s):				
	71.981s	395,182,000,000		

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 1...

- ❑ **Function** – имена функций
- ❑ **CPU Time** – время работы каждой функции
- ❑ **Instructions retired** – количество выполненных инструкций
- ❑ **CPI** – количество тактов на инструкцию (оптимум = 0.25, «хорошее» для HPC значение – не более 1.0).
- ❑ **Module** – процесс, в котором работала функция.
- ❑ Действительно, все время работала одна функция.
- ❑ Обратим внимание на показатель **CPI**. У нас $CPI = 0.453$, что неплохо. Но инструкций слишком много.
- ❑ Обратим внимание: у **MKL** **CPI** больше, но инструкций гораздо меньше.



Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 1...

Lightweight Hotspots - Hotspots

Analysis Target Analysis Type Summary Bottom-up sparse_n....

Line	Source	CPU Time	Instructions Retired
117	vector<int> row_index;		
118			
119	clock_t start = clock();		
120	int rowNZ;		
121	row_index.push_back(0);		
122	for (int i = 0; i < N; i++)		
123	{		
124	rowNZ = 0;		
125	for (int j = 0; j < N; j++)	0.189	1,124,000,000
126	{		
127	double sum = 0;	0.026	172,000,000
128	int isFound = 0;		
129	for (int k = A.RowIndex[i]; k < A.RowIndex[i + 1]; k++)	2.332	14,110,000,000
130	{		
131	for (int l = B.RowIndex[j]; l < B.RowIndex[j + 1]; l++)	28.626	148,874,000,000
132	{		
133	if (A.Col[k] == B.Col[l])	40.402	229,338,000,000
134	{		
135	sum += A.Value[k] * B.Value[l];	0.138	386,000,000
136	isFound = 1;		
137	break;		
138	}		
139	}		
140	}		
141	if (fabs(sum) > ZERO_IN_CRS)	0.205	920,000,000
142	{		

Selected 1 row(s):

Address	Line	Assembly	CPU Time	Instructions Retired
0x2a79	129	cmp esi, ecx	0.003	10,000,000
0x2a7b	129	mov dword ptr [ebp-0x18], ecx	0.002	12,000,000
0x2a7e	129	jnl 0x2f7b <Block 120>		
0x2a84		Block 34:		
0x2a84	131	mov edx, dword ptr [ebx+0x2c]	0.039	164,000,000
0x2a87	131	mov edi, dword ptr [ebp-0x1c]		
0x2a8a	131	pxor xmm1, xmm1		6,000,000
0x2a8e	133	mov ecx, dword ptr [ebx+0x28]		
0x2a91	131	mov eax, dword ptr [edx+edi*4]	0.044	158,000,000
0x2a94	131	mov edx, dword ptr [edx+edi*4+0x4]	0.003	
0x2a98	131	mov dword ptr [ebp-0x14], eax	0.029	148,000,000
0x2a9b		Block 35:		
0x2a9b	131	mov eax, dword ptr [ebp-0x14]	2.140	10,482,000,000
0x2a9e	131	cmp eax, edx	0.152	870,000,000
0x2aa0	131	jnl 0x2aeb <Block 46>	0.792	4,146,000,000
0x2aa2		Block 36:		
0x2aa2	133	mov edi, dword ptr [ebx+0x14]	1.295	6,980,000,000
0x2aa5	133	mov edi, dword ptr [edi+esi*4]	0.323	1,196,000,000
0x2aa8		Block 37:		
0x2aa8	133	cmp edi, dword ptr [ecx+eax*4]	31.359	178,310,000,000
0x2aab	133	jz 0x2f55 <Block 118>	7.425	42,852,000,000
0x2ab1		Block 38:		
0x2ab1	131	inc eax	24.673	129,012,000,000
0x2ab2	131	cmp eax, edx	0.378	2,036,000,000
0x2ab4	131	j1 0x2aa8 <Block 37>	0.001	
0x2ab6		Block 39:		

Highlighted 5 row(s):

No filters are applied. Module: [All] Thread: [All] Process: [All]

Line	Source	CPU Time	Instructions Retired
117	vector<int> row_index;		
118			
119	clock_t start = clock();		
120	int rowNZ;		
121	row_index.push_back(0);		
122	for (int i = 0; i < N; i++)		
123	{		
124	rowNZ = 0;		
125	for (int j = 0; j < N; j++)	0.189	1,124,000,000
126	{		
127	double sum = 0;	0.026	172,000,000
128	int isFound = 0;		
129	for (int k = A.RowIndex[i]; k < A.RowIndex[i + 1]; k++)	2.332	14,110,000,000
130	{		
131	for (int l = B.RowIndex[j]; l < B.RowIndex[j + 1]; l++)	28.626	148,874,000,000
132	{		
133	if (A.Col[k] == B.Col[l])	40.402	229,338,000,000
134	{		
135	sum += A.Value[k] * B.Value[l];	0.138	386,000,000
136	isFound = 1;		
137	break;		
138	}		
139	}		
140	}		
141	if (fabs(sum) > ZERO_IN_CRS)	0.205	920,000,000
142	{		

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 1...

129	for (int k = A.RowIndex[i]; k < A.RowIndex[i + 1]; k++)	2.332	14,110,000,000
130	{		
131	for (int l = B.RowIndex[j]; l < B.RowIndex[j + 1]; l++)	28.626	148,874,000,000
132	{		
133	if (A.Col[k] == B.Col[l])	40.402	229,338,000,000
134	{		
135	sum += A.Value[k] * B.Value[l];	0.138	386,000,000
136	isFound = 1;		
137	break;		
138	}		
139	}		

- ❑ Гигантское количество сравнений в строке 133.
- ❑ Гигантское количество итераций цикла в строке 131.
- ❑ Что-то с этим нужно делать. Идеи?

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 1...

- ❑ Нужно воспользоваться тем фактом, что элементы в строке упорядочены по ординате.
 - ❑ Реализуем сопоставление за линейное время вместо старого варианта, который работает за квадратичное время.
 - ❑ Используем аналог алгоритма слияния отсортированных массивов.
1. Встать на начало обоих векторов (**ks = ...**, **ls = ...**).
 2. Сравнить текущие элементы **A.Col[ks]** и **B.Col[ls]**.
 - Если значения совпадают, накопить в сумму произведение **A.Value[ks] * B.Col[ls]** и увеличить оба индекса;
 - Иначе – увеличить один из индексов, в зависимости от того, какое значение больше (например, **A.Col[ks] > B.Col[ls] → ls++**).

Оптимизированная последовательная версия. Алгоритмическая оптимизация. Подход 1...

```
Administrator: C:\Windows\System32\cmd.exe  
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
c:\ParallelCalculus\07_SparseMM\Release>02_Optim1.exe 10000 50  
OK  
10000 500000 130775 5678160  
0.016 8.408 0.110
```

- ❑ Ускорение ~ в 7 раз.
- ❑ С ростом N оно будет только увеличиваться.

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 2...

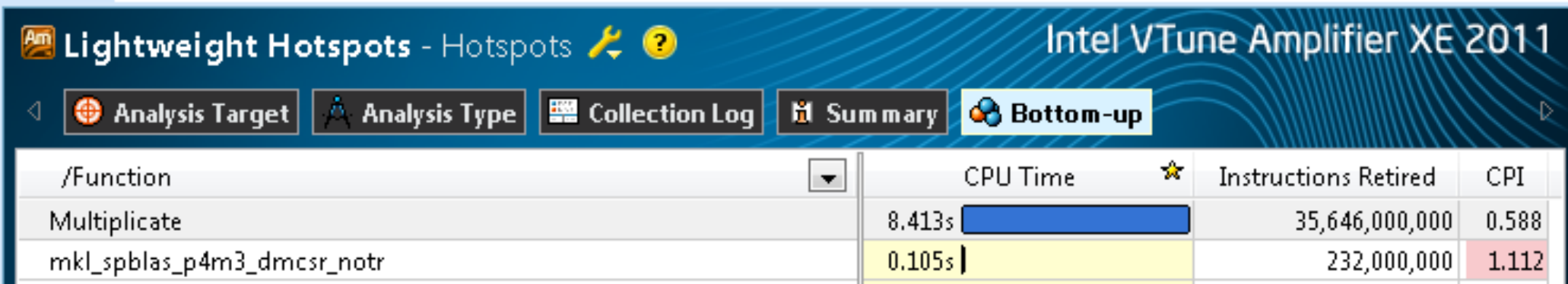
Am Lightweight Hotspots - Hotspots  

Intel VTune Amplifier XE 2011

 Analysis Target  Analysis Type  Collection Log  Summary  Bottom-up

/Function	CPU Time	Instructions Retired	CPI
Multiply	8.413s	35,646,000,000	0.588
mkl_spblas_p4m3_dmcsr_notr	0.105s	232,000,000	1.112
main	0.097s	298,000,000	0.799
mkl_spblas_p4m3_dmcsradd_notr	0.068s	294,000,000	0.565
std::push_back	0.026s	80,000,000	0.650
SparseDiff	0.023s	46,000,000	1.087
std::push_back	0.018s	66,000,000	1.091
mkl_spblas_p4m3_dsorow	0.011s	78,000,000	0.410
SparseMKLMult	0.008s	10,000,000	1.800
GenerateSpecialCRS	0.002s	8,000,000	0.500
std::Insert_n	0.001s	2,000,000	3.000
`eh vector constructor iterator'	0s	2,000,000	0.000
Selected 1 row(s):			
	8.413s	35,646,000,000	











Оптимизированная последовательная версия. Алгоритмическая оптимизация. Подход 2...



The screenshot shows the Intel VTune Amplifier XE 2011 interface. The title bar reads 'Lightweight Hotspots - Hotspots' and 'Intel VTune Amplifier XE 2011'. Below the title bar are five tabs: 'Analysis Target', 'Analysis Type', 'Collection Log', 'Summary', and 'Bottom-up'. The 'Bottom-up' tab is selected. Below the tabs is a table with four columns: '/Function', 'CPU Time', 'Instructions Retired', and 'CPI'. The table contains two rows of data.

/Function	CPU Time	Instructions Retired	CPI
Multiply	8.413s	35,646,000,000	0.588
mkl_spblas_p4m3_dmcscr_notr	0.105s	232,000,000	1.112

- ❑ Время работы уменьшилось.
- ❑ Количество инструкций уменьшилось.
- ❑ Но и время, и количество инструкций остаются слишком большими по сравнению с MKL-реализацией.

Line	Source	CPU Time 	Instructions Retired
121			
122	row_index.push_back(0);		
123	for (int i = 0; i < N; i++)		
124	{		
125	for (int j = 0; j < N; j++)	0.091s 	344,000,000
126	{		
127	// Умножаем строку i матрицы A и столбец j матрицы B		
128	double sum = 0;	0.001s	2,000,000
129	int ks = A.RowIndex[i]; int ls = B.RowIndex[j];	0.090s 	286,000,000
130	int kf = A.RowIndex[i + 1] - 1; int lf = B.RowIndex[j + 1] - 1	0.134s 	648,000,000
131	while ((ks <= kf) && (ls <= lf))	0.724s 	2,182,000,000
132	{		
133	if (A.Col[ks] < B.Col[ls])	2.366s 	9,088,000,000
134	ks++;	1.922s 	8,704,000,000
135	else		
136	if (A.Col[ks] > B.Col[ls])	1.788s 	8,110,000,000
137	ls++;	1.007s 	5,404,000,000
138	else		
139	{		
140	sum += A.Value[ks] * B.Value[ls];	0.032s	138,000,000
141	ks++;	0.006s	24,000,000
142	ls++;	0.047s	92,000,000
143	}		
144	}		
145	if (fabs(sum) > ZERO_IN_CRS)	0.092s 	352,000,000
146	{		

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 2...

- ❑ Основное время по-прежнему тратится на слияние векторов.
- ❑ Проверки во внутреннем цикле занимают практически все время.
- ❑ Реализация алгоритма не очень хорошо соответствует особенностям архитектуры: факт того, какой из элементов ($A.Col[k_s]$ или $B.Col[l_s]$) окажется больше, невозможно прогнозировать.
- ❑ Наличие ошибок в предсказании условных переходов приводит к неэффективной работе.

Оптимизированная последовательная версия. Алгоритмическая оптимизация. Подход 2...

□ Используем новый вид анализа: General Exploration

General Exploration - Ha							Intel V
Analysis Target Analysis							
/Function	CPI	Retire Stalls	LLC Miss	Instruction Starvation	Branch Mispredict	Execution Stalls	
Multiply	0.614	0.260	0.000	0.149	0.065	0.057	
mkl_spblas_p4m3_dmcsr_notr	1.088	1.691	0.106	0.000	0.000	0.000	
mkl_spblas_p4m3_dmcsradd_notr	0.515	0.000	0.000	0.000	0.000	0.000	
main	0.861	0.000	0.000	0.000	0.000	0.992	
SparseDiff	1.594	4.588	0.000	0.000	0.000	0.000	
mkl_spblas_p4m3_dsorow	0.490	0.000	0.000	0.000	0.000	0.000	
std::push_back	1.026	1.300	0.000	0.000	0.683	0.000	
std::push_back	1.059	0.000	0.000	0.000	0.054	0.000	
SparseMKLMult	1.800	8.667	0.000	0.000	0.000	0.000	
std::Insert_n	1.333	0.000	0.000	0.000	0.000	0.000	
GenerateSpecialCRS	0.750	0.000	0.000	0.000	0.000	0.000	
[Import thunk new]	0.000	0.000	0.000	0.000	0.000	0.000	

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 2...

- ❑ Итак, мы существенно ускорили приложение, но количество выполняемых инструкций остается слишком большим, а реализация плохо соответствует архитектуре.
 - ❑ Обе грани общей проблемы сводятся к поиску соответствующих элементов разреженных векторов.
 - ❑ Нельзя ли предложить более эффективный вариант?
-
- ❑ Обратимся к литературе за поиском идей для оптимизации. Так, например, в книге С. Писсанецки предлагается остроумный вариант вычисления **скалярного произведения разреженных векторов.**



Оптимизированная последовательная версия. Алгоритмическая оптимизация. Подход 2...

□ Шаг1:

- Создадим дополнительный целочисленный массив X длины N .
- Инициализируем его числом -1 .
- Обнулим вещественную переменную S .

□ Шаг2:

- Просмотрим в цикле все ненулевые элементы первого вектора $V1$:
 - Пусть такой элемент с порядковым номером i расположен в столбце с номером $j = V1.Col[i]$.
 - В этом случае запишем i в j -ю ячейку дополнительного массива.



Оптимизированная последовательная версия. Алгоритмическая оптимизация. Подход 2...

□ ...

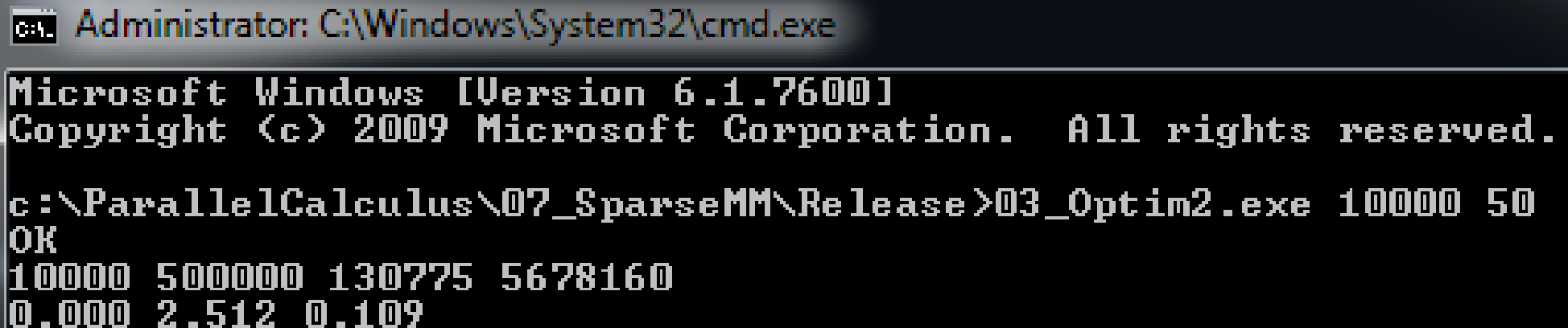
□ Шаг3:

- Просмотрим в цикле все ненулевые элементы второго вектора $V2$:
 - Пусть элемент с порядковым номером k расположен в столбце с номером $z = V2.Col[k]$.
 - Проверим значение $X[z]$:
 - Если оно равно -1, в первом векторе нет соответствующего элемента, т.е. умножение выполнять не нужно.
 - Иначе умножаем $V2.Value[k]$ и $V1.Value[X[z]]$ и накапливаем в S .

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 2...

- ❑ Создадим в рамках решения **07_SparseMM** новый проект с названием **03_Optim2**.
- ❑ Скопируем файлы.
- ❑ Изменим реализацию функции **Multiply()**.



```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.














c:\ParallelCalculus\07_SparseMM\Release>03_Optim2.exe 10000 50
OK
10000 500000 130775 5678160
0.000 2.512 0.109
```

- ❑ **Ускорение в 3 раза.**












Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 2...

/Function	CPU Time▼★	Instructions Retired	CPI
Multiply	2.444s 	11,428,000,000	0.532
mkl_spblas_p4m3_dmcscr_notr	0.102s 	234,000,000	1.137
main	0.096s 	294,000,000	0.830
mkl_spblas_p4m3_dmcscradd_notr	0.072s 	296,000,000	0.628
SparseDiff	0.032s 	46,000,000	1.783
std::push_back	0.026s 	80,000,000	0.925
std::push_back	0.023s 	76,000,000	0.947
mkl_spblas_p4m3_dsortrow	0.012s 	86,000,000	0.419
__intel_memset	0.011s 	36,000,000	1.000
SparseMKLMult	0.008s 	10,000,000	1.800
_intel_fast_memset	0.001s 	0	0.000
std::Insert_n	0.001s 	6,000,000	0.333
GenerateSpecialCRS	0.001s 	2,000,000	2.000
mkl_spblas_mkl_dcsradd	0s	2,000,000	0.000
[Import thunk new]	0s	2,000,000	0.000

Оптимизированная последовательная версия.

Алгоритмическая оптимизация. Подход 2...

line	Source	CPU Time 	Instructions Retired
140	// Построен индекс строки i матрицы A		
141	// Теперь необходимо умножить ее на каждую из строк матрицы BT		
142	for (j = 0; j < N; j++)	88.722ms 	290,000,000
143	{		
144	// j-я строка матрицы B		
145	volatile double sum = 0;	6.015ms	40,000,000
146	int ind3 = B.RowIndex[j], ind4 = B.RowIndex[j + 1];	133.835ms 	686,000,000
147	// Все ненулевые элементы строки j матрицы B		
148	for (k = ind3; k < ind4; k++)	561.654ms 	3,374,000,000
149	{		
150	int bcol = B.Col[k];	557.895ms 	2,040,000,000
151	int aind = temp[bcol];	96.241ms 	410,000,000
152	if (aind != -1)	751.880ms 	3,708,000,000
153	sum += A.Value[aind] * B.Value[k];	66.165ms 	298,000,000
154	}		
155	if (fabs(sum) > ZERO_IN_CRS)	109.023ms 	404,000,000
156	{		
157	columns.push_back(j);		
158	values.push_back(sum);		
159	NZ++;		
160	}		
161	}		
162	row_index.push_back(NZ);		2,000,000
163	}		

Оптимизированная последовательная версия. Алгоритмическая оптимизация. Подход 2...

- ❑ Время работы уменьшилось.
- ❑ Количество инструкций уменьшилось.
- ❑ Желающие могут продолжить оптимизацию кода при наличии идей по его ускорению.

Программная оптимизация. Последние штрихи...

- ❑ До сих пор все эксперименты, которые мы проводили, выполнялись в условиях, когда матрица A имеет «регулярную» структуру – в каждой ее строке по 50 ненулевых элементов.
- ❑ Проведем теперь эксперимент, когда «регулярной» является матрица B , то есть после генерации переставим их местами в функции умножения **Multiply()**.
- ❑ Проект **04_Optim3**.



Программная оптимизация. Последние штрихи...

```
Administrator: C:\Windows\System32\cmd.exe


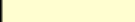
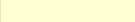
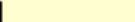


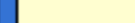
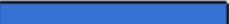
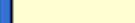
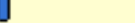
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\ParallelCalculus\07_SparseMM\Release>04_Optim3.exe 10000 50
OK
10000 500000 130775 6108209
0.016 8.517 3.838
```

- ❑ Вот это да!
- ❑ Время умножения у нас выросло в 4 раза.
- ❑ Время умножения у MKL выросло в 40 раз.
- ❑ MKL все еще обгоняет нашу реализацию.








Программная оптимизация. Последние штрихи...

line	Source	CPU Time 	Instructions Retired
140	// Построен индекс строки i матрицы A		
141	// Теперь необходимо умножить ее на каждую из строк матрицы BT		
142	for (j = 0; j < N; j++)	0.120s 	588,000,000
143	{		
144	// j-я строка матрицы B		
145	double sum = 0;	0.021s 	62,000,000
146	int ind3 = B.RowIndex[j], ind4 = B.RowIndex[j + 1];	0.104s 	570,000,000
147	// Все ненулевые элементы строки j матрицы B		
148	for (k = ind3; k < ind4; k++)	3.083s 	15,358,000,000
149	{		
150	int bcol = B.Col[k];	3.005s 	15,722,000,000
151	int aind = temp[bcol];	0.476s 	2,258,000,000
152	if (aind != -1)	5.114s 	25,980,000,000
153	sum += A.Value[aind] * B.Value[k];	0.347s 	664,000,000
154	}		
155	if (fabs(sum) > ZERO_IN_CRS)	0.278s 	1,176,000,000
156	{		
157	columns.push_back(j);		
158	values.push_back(sum);		
159	NZ++;		
160	}		
161	}		
162	row_index.push_back(NZ);		
163	}		

Программная оптимизация. Последние штрихи...

❑ Гигантское количество инструкций!

❑ В чем же дело?

147	// Все ненулевые элементы строки j матрицы B		
148	for (k = ind3; k < ind4; k++)	3.083s 	15,358,000,000
149	{		
150	int bcol = B.Col[k];	3.005s 	15,722,000,000
151	int aind = temp[bcol];	0.476s 	2,258,000,000
152	if (aind != -1)	5.114s 	25,980,000,000
153	sum += A.Value[aind] * B.Value[k];	0.347s 	664,000,000
154	}		

Программная оптимизация. Последние штрихи...

- ❑ Гигантское количество инструкций!
- ❑ В чем же дело?
- ❑ Не работает идея с построением индекса. Во внутреннем цикле становится слишком много итераций (50).
- ❑ Можно ли что-нибудь сделать?

147	// Все ненулевые элементы строки j матрицы B		
148	for (k = ind3; k < ind4; k++)	3.083s	15,358,000,000
149	{		
150	int bcol = B.Col[k];	3.005s	15,722,000,000
151	int aind = temp[bcol];	0.476s	2,258,000,000
152	if (aind != -1)	5.114s	25,980,000,000
153	sum += A.Value[aind] * B.Value[k];	0.347s	664,000,000
154	}		

Программная оптимизация. Последние штрихи...

- ❑ Гигантское количество инструкций!
- ❑ В чем же дело?
- ❑ Не работает идея с построением индекса. Во внутреннем цикле становится слишком много итераций (50).
- ❑ Можно ли что-нибудь сделать?
- ❑ Поменяем матрицы в нашей реализации местами.
 $(AB)^T = B^T A^T$
- ❑ Транспонирование работает очень быстро, а выигрыш от изменения порядка матриц будет весомым.



Программная оптимизация. Последние штрихи...

```
if (A.NZ > B.NZ)
{
    Multiply(A, B, C, time);
}
else
{
    crsMatrix CT;
    Multiply(B, A, C, time);
    Transpose2(C, CT);
    int i;
    for (i = 0; i < CT.NZ; i++)
    {
        C.Col[i] = CT.Col[i];
        C.Value[i] = CT.Value[i];
    }
    for(i = 0; i <= CT.N; i++)
        C.RowIndex[i] = CT.RowIndex[i];
    FreeMatrix(CT);
}
```

- ❑ Добавим эвристику в алгоритм умножения.
- ❑ В нашем случае это должно дать эффект.

Программная оптимизация. Последние штрихи...

```
C:\> Administrator: C:\Windows\System32\cmd.exe

Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\ParallelCalculus\07_SparseMM\Release>04_Optim3.exe 10000 50
OK
10000 500000 130775 6108209
0.016 2.777 3.837
```

- ❑ Время работы сократилось почти в 3 раза.
- ❑ Обогнали MKL 😊
- ❑ Разумеется, только на специфических данных.



Программная оптимизация. Последние штрихи...

C:\> Administrator: C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
c:\ParallelCalculus\07_SparseMM\Release>04_Optim3.exe 10000 50  
OK  
10000 500000 130775 6108209  
0.016 2.777 3.837
```

- ❑ Цели обогнать MKL в рамках лабораторной работы не ставилось.
- ❑ Мы хотели проиллюстрировать тот факт, что знание дополнительной информации о задаче часто позволяет обогнать реализацию, оптимизированную для общего случая.



Двухфазная последовательная реализация...

- В ряде задач требуется многократно выполнять однотипные операции над матрицами, имеющими одинаковые портреты, но разные значения элементов.
- Задача линейного программирования:

$$c^T x \rightarrow \min$$

$$Ax = b$$

В ряде задач матрица A является **разреженной**.

В некоторых методах локальной оптимизации (IPM и др.) необходимо на каждой итерации решать СЛАУ с разреженной матрицей.

При этом на каждой итерации портрет матрицы одинаков.



Двухфазная последовательная реализация...

- ❑ В ряде задач необходимо многократно выполнять однотипные операции над разреженными структурами, при этом портрет не меняется от итерации к итерации.
- ❑ **Классический подход к оптимизации:**
Разделение алгоритма на две фазы:
 - Символическая (получение портрета результата).
 - Численная (заполнение полученного портрета).Заметим, что в рамках символической части решается **задача определения объема необходимой памяти и ее выделение.**
- ❑ *Разделение на фазы часто не замедляет программу и в случае однократного выполнения операции.*

Двухфазная последовательная реализация...

Символическая фаза.

- ❑ Необходимо построить портрет результата.
 - ❑ Для этого требуется понять, в каких позициях матрицы C будут стоять элементы.
 - ❑ Заметим, что нули, полученные в результате вычислений, будут **храниться** в структуре матрицы.
-
- ❑ **Предложите алгоритм символической фазы.**

Двухфазная последовательная реализация...

Символическая фаза.

- ❑ После транспонирования матрицы В (достаточно символического транспонирования) используем предыдущую оптимизированную реализацию, убрав из нее расчетную часть.
- ❑ Проект **05_Two_phase**.

```
int SymbolicMult(crsMatrix A, crsMatrix B, crsMatrix &C,  
    double &time);  
int NumericMult(crsMatrix A, crsMatrix B, crsMatrix &C,  
    double &time);
```

Двухфазная последовательная реализация...

```
int main(int argc, char *argv[])
{
    ...
    double timeT = Transpose2(B, BT);
    double timeS, timeM, timeM1;
    SymbolicMult(A, BT, C, timeS);
    NumericMult(A, BT, C, timeM);
    crsMatrix CM; double diff;
    SparseMKLMult(A, B, CM, timeM1);
    SparseDiff(C, CM, diff);
    if (diff < EPSILON) printf("OK\n"); else printf("not OK\n");
    printf("%d %d %d %d\n", A.N, A.NZ, B.NZ, C.NZ);
    printf("%.3f %.3f %.3f\n", timeT, timeS, timeM, timeM1);
    FreeMatrix(A); FreeMatrix(B); FreeMatrix(BT); FreeMatrix(C);
    return 0;
}
```



```

for (j = 0; j < N; j++)    {
    //      double sum = 0;
    int IsFound = 0;
    int ind3 = B.RowIndex[j], ind4 = B.RowIndex[j + 1];
    for (k = ind3; k < ind4; k++)    {
        int bcol = B.Col[k];
        int aind = temp[bcol];
        if (aind != -1)
        {
            //      sum += A.Value[aind] * B.Value[k];
            IsFound = 1;
            break;
        }
    }
    //      if (fabs(sum) > ZERO_IN_CRS)
    if (IsFound) {
        columns.push_back(j); NZ++;
        //      values.push_back(sum);
    }
    ...
    for (j = 0; j < NZ; j++) C.Col[j] = columns[j];
    //      C.Value[j] = values[j];

```

Двухфазная последовательная реализация...

□ Лучше переписать так:

```
...  
int aind = -1; k = ind3;  
while ((aind == -1) && (k < ind4))  
{  
    int bcol = B.Col[k];  
    int aind = temp[bcol];  
    k++;  
}  
if (aind != -1)  
    ...
```

Двухфазная последовательная реализация...

- ❑ Численная фаза.

- ❑ Алгоритм:

Проход по матрице C и вычисление каждого ее элемента как скалярного произведения строки A на строку B^T .

«Дьявол кроется в деталях»

Весь вопрос в том, как считать скалярное произведение. Используем оптимизированный вариант, основанный на применении индексного массива.

Двухфазная последовательная реализация

```
Administrator: C:\Windows\System32\cmd.exe  
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
c:\ParallelCalculus\07_SparseMM\Release>05_Two-phase.exe 10000 50  
OK  
10000 500000 130775 5678427  
0.015 1.935 0.483 0.109
```

- ❑ Суммарное время работы примерно равно времени работы однофазной версии, построенной на тех же идеях.
- ❑ Численная фаза работает в несколько раз быстрее символической (полезно для случая многократного применения).

Алгоритм Густавсона...

- ❑ Один из лучших на практике.
- ❑ **Суть метода:** избежать проблемы выделения столбца в матрице B .
- ❑ Для этого предлагается **изменить порядок вычислений**:
 - вместо того, чтобы умножать строку на столбец, вычислять **произведения каждого элемента матрицы A на все элементы соответствующей строки матрицы B** , постепенно накапливая частичные суммы.

Алгоритм Густавсона...

- ❑ Рассмотрим i -ю строку матрицы A .
 - Для всех значений j умножим элемент $A[i, j]$ на все элементы j -ой строки матрицы B .
 - Все произведения будем накапливать в ячейки, соответствующие i -ой строке матрицы C .
 - По окончании обработки i -ой строки матрицы A оказывается полностью вычисленной i -я строка матрицы C .
- ❑ Могут быть реализованы символическая и численная фазы.
- ❑ Необходимо упорядочить результат – 2 транспонирования в конце.
- ❑ Реализуйте алгоритм в рамках дополнительных заданий к работе.



Алгоритм Густавсона

Порядок матриц (N)	Проект 03_Optim2 A * B	Алгоритм Густавсона A * B	MKL A * B	Проект 03_Optim2 B * A	Алгоритм Густавсона B * A	MKL B * A
10 000	2.512	0.468	0.110	3.339	0.515	3.837
15 000	5.445	0.764	0.171	6.989	0.827	6.068
20 000	9.454	0.842	0.234	12.309	1.185	8.268
25 000	14.617	1.185	0.328	18.658	1.529	10.498
30 000	20.982	1.747	0.406	26.504	1.904	12.823
35 000	34.429	1.872	0.452	35.849	2.278	15.116
40 000	44.460	1.529	0.375	46.503	2.667	17.269

A – регулярная (50 элементов в строке)

B – с нарастанием количества ненулевых элементов

Наблюдаются существенно лучшие результаты по времени работы.

Код разработан Филиппенко С. и Лебедевым С. (ВМК ННГУ)



Параллельная реализация...

- ❑ **OpenMP**
- ❑ Проект **06_OpenMP**
- ❑ Идея распараллеливания алгоритма (оптимизированная реализация обычного алгоритма) выглядит достаточно очевидной:
 - Во внешнем цикле мы перебираем строки матрицы A и далее работаем с ними независимо.
 - Таким образом, естественный вариант параллелизма – распределение строк между потоками.
- ❑ **Проблема:** обеспечить структуры данных для всех потоков (решить проблему доступа к векторам STL, `push_back()`).



Параллельная реализация...

- ❑ Продублируем структуры **columns**, **values** по числу строк. В конце объединим.
- ❑ Сделаем массив **temp** локальным внутри каждого потока.
- ❑ Массив **row_index** обрабатывается бесконфликтно – работаем с ячейками, соответствующими строкам. В конце – восстановим нормальное состояние.

```

int Multiply(crsMatrix A, crsMatrix B, crsMatrix &C,
    double &time) {
...
    vector<int>* columns = new vector<int>[N];
    vector<double> *values = new vector<double>[N];
    int* row_index = new int[N + 1];
    memset(row_index, 0, sizeof(int) * N);
#pragma omp parallel {
    int *temp = new int[N];
#pragma omp for private(j, k)
    for (i = 0; i < N; i++) {
        ...
        if (fabs(sum) > ZERO_IN_CRS) {
            columns[i].push_back(j);
            values[i].push_back(sum);
            row_index[i]++;
        }
    }
}
    delete [] temp;
} // omp parallel

```

```

...
int NZ = 0;
for(i = 0; i < N; i++) {
    int tmp = row_index[i];
    row_index[i] = NZ;
    NZ += tmp;
}
row_index[N] = NZ;
InitializeMatrix(N, NZ, C);
int count = 0;
for (i = 0; i < N; i++) {
    int size = columns[i].size();
    memcpy(&C.Col[count], &columns[i][0], size*sizeof(int));
    memcpy(&C.Value[count], &values[i][0], size*sizeof(double));
    count += size;
}
memcpy(C.RowIndex, &row_index[0], (N + 1) * sizeof(int));
delete [] row_index; delete [] columns; delete [] values;
...
}

```

Параллельная реализация...

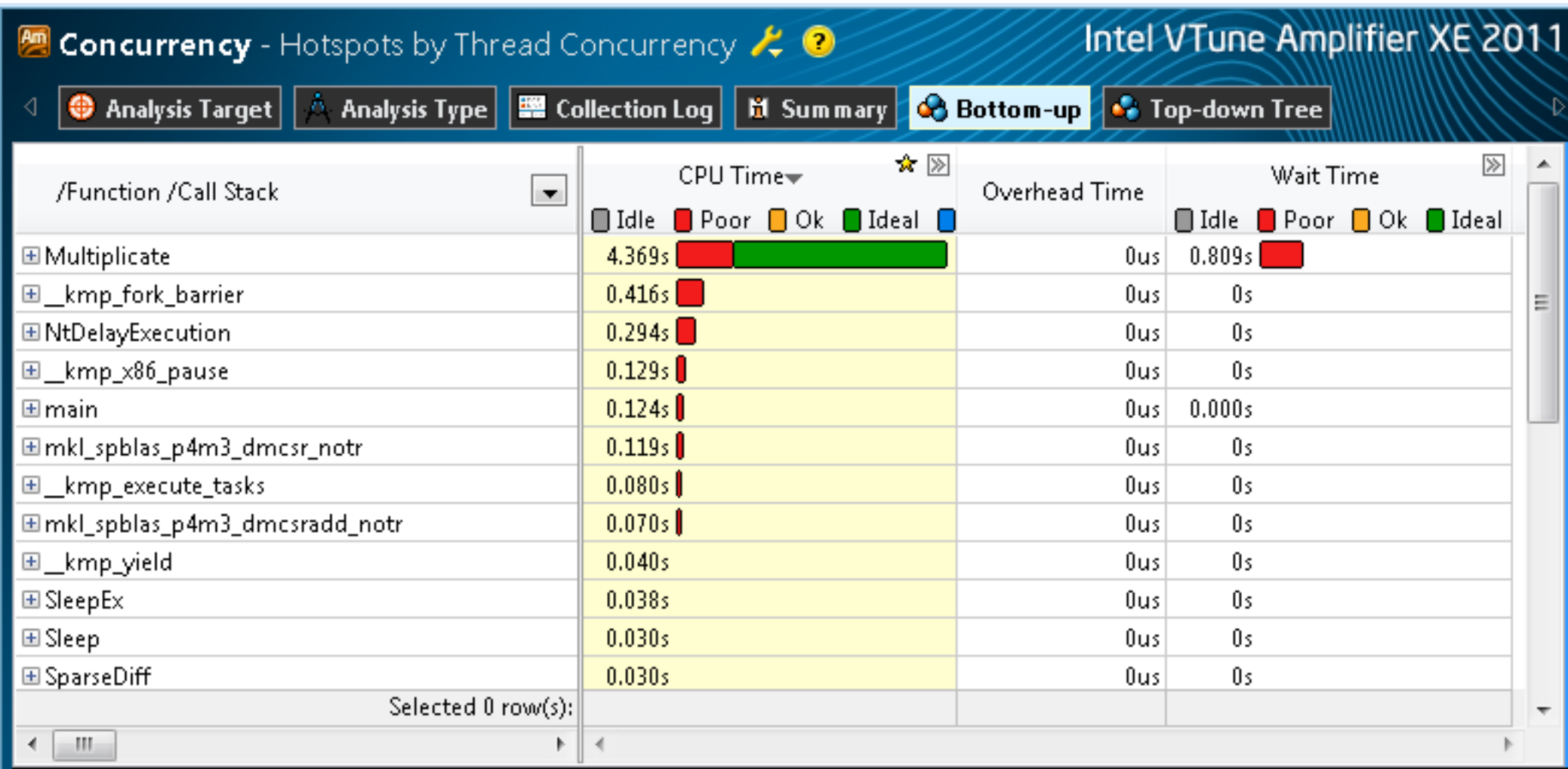
```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\ParallelCalculus\07_SparseMM\Release>06_OpenMP.exe 10000 50
OK
10000 500000 130775 5678160
0.000 1.029 0.110
```

- ❑ Ускорение по сравнению с версией из проекта **03_Optim2** составляет $2.512 / 1.029 = 2.441$ и довольно далеко от ВОСЬМИ.
- ❑ Используем профилировщик.

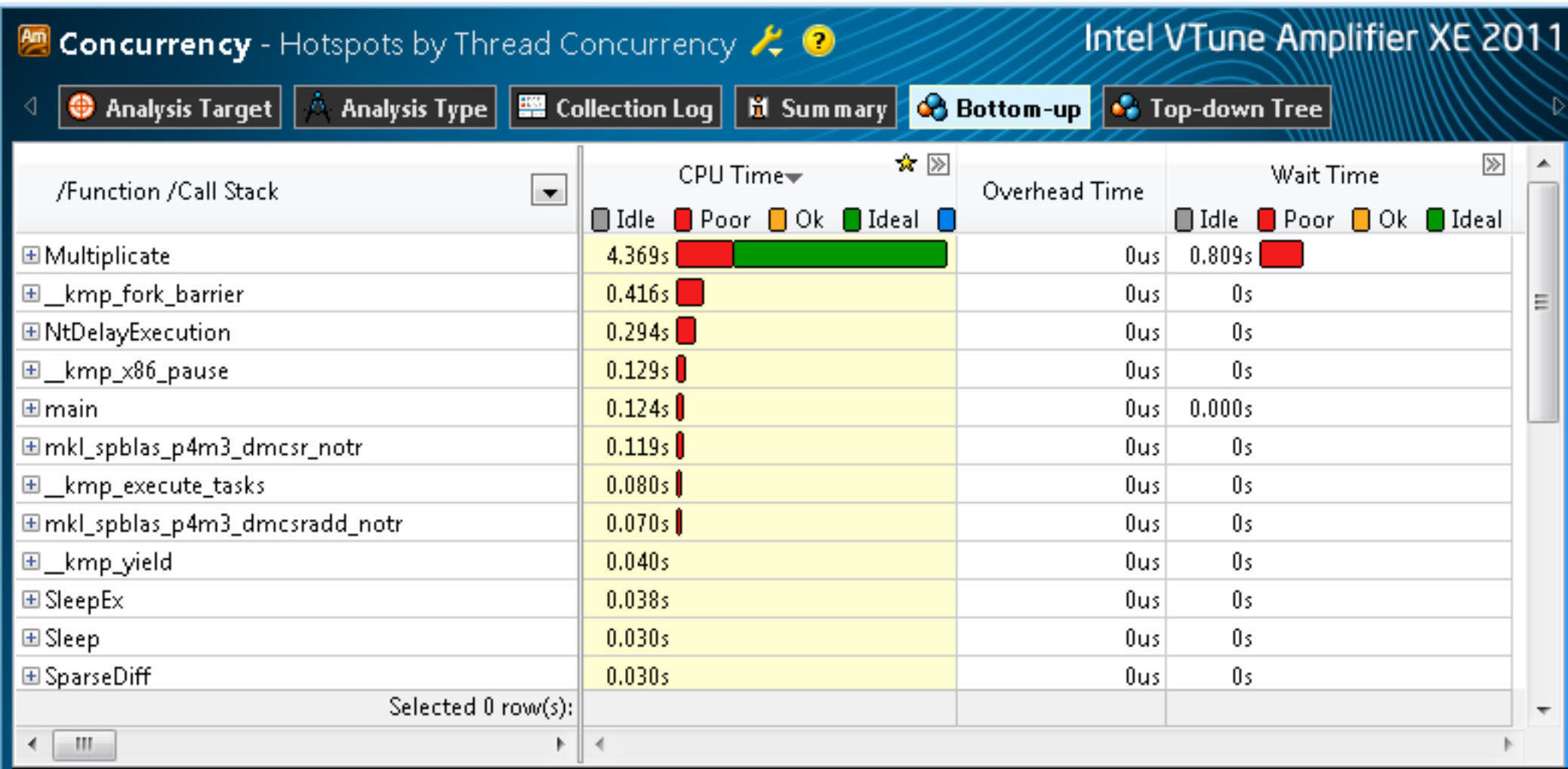
Параллельная реализация...

- ❑ Amplifier, режим Concurrency.



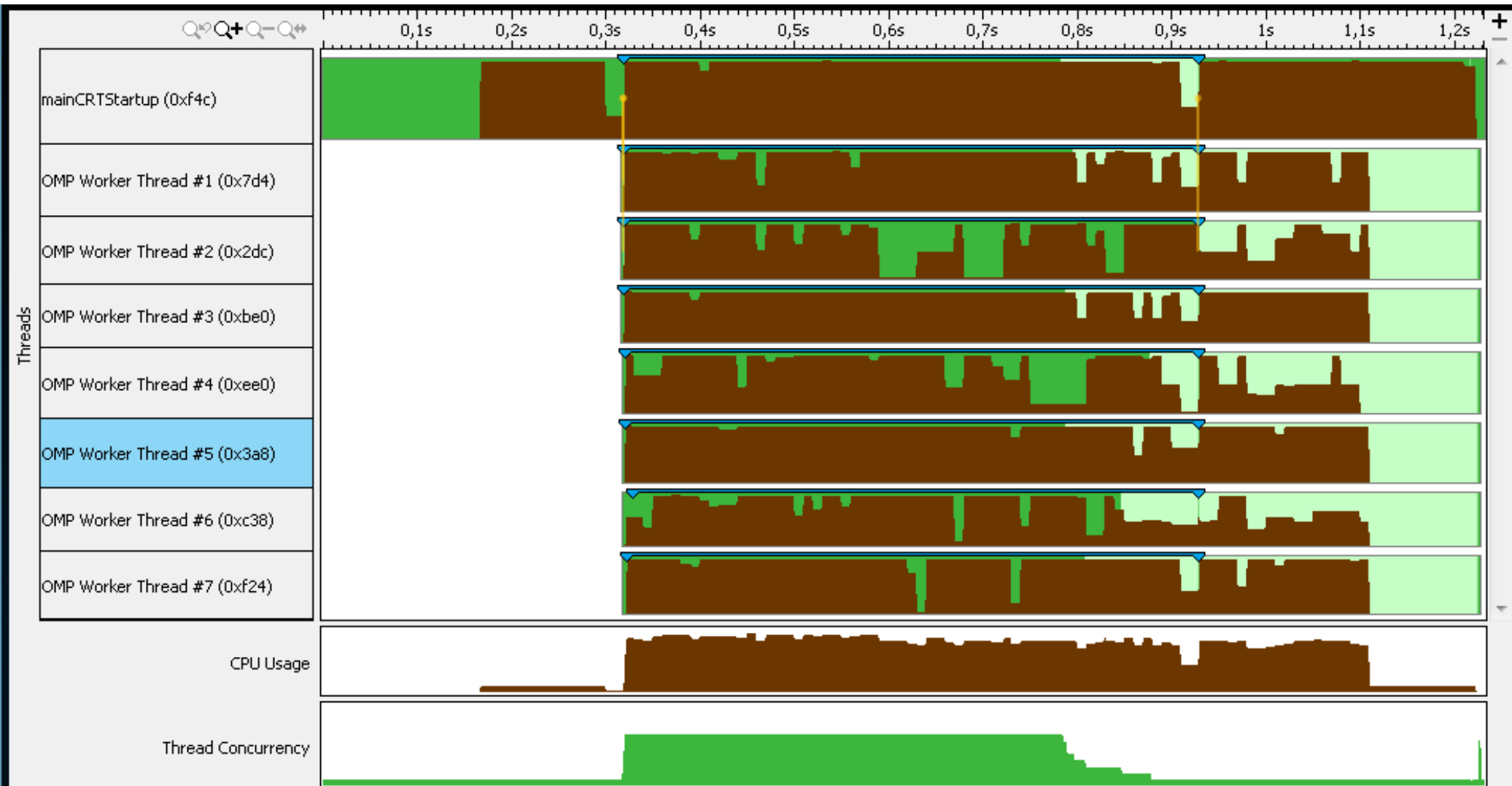
Параллельная реализация...

- ❑ Существенное время ожидания.



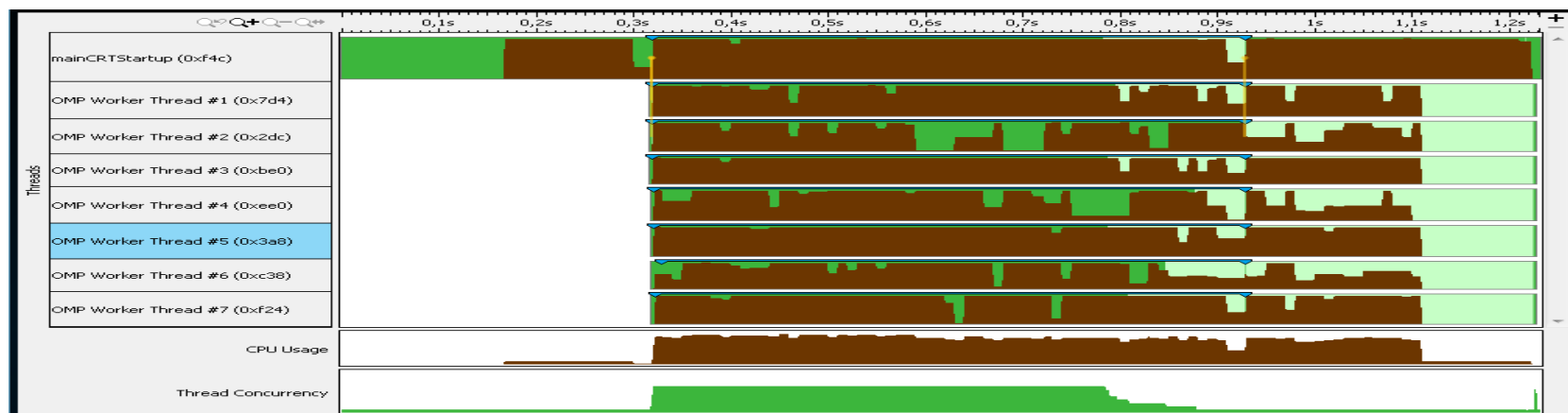
Параллельная реализация...

❑ Amplifier, режим Concurrency.

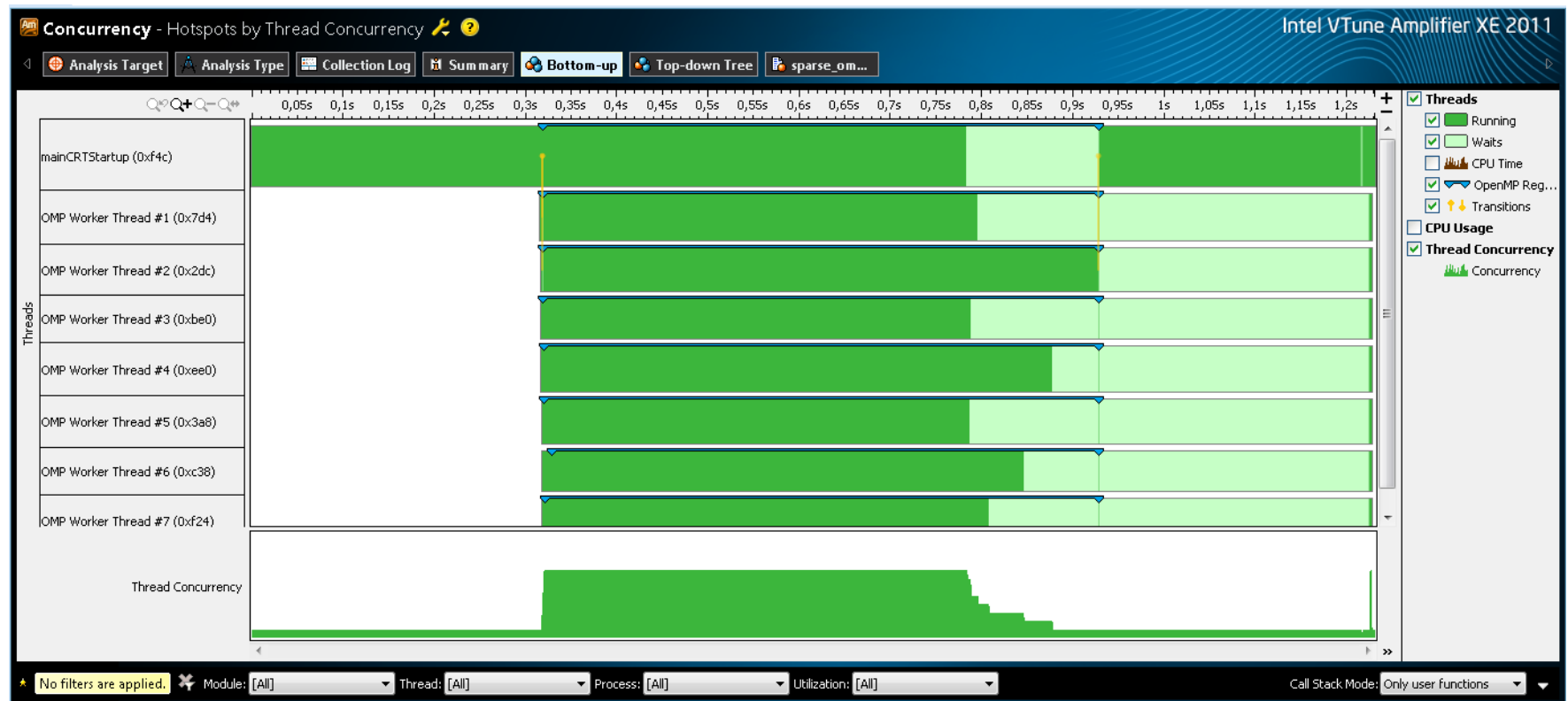


Параллельная реализация...

- ❑ Главный поток работал последовательно, далее создал 7 доп. потоков.
- ❑ Работа шла в параллельной секции (синие скобки).
 - Коричневая часть – использование CPU.
 - Темно-зеленая часть – поток работал.
 - Светло-зеленая часть – поток ждал.
- ❑ Очевиден дисбаланс → большое время ожидания.
- ❑ Параллельность сошла на нет значительно раньше завершения параллельной секции (см. нижнюю полосу).



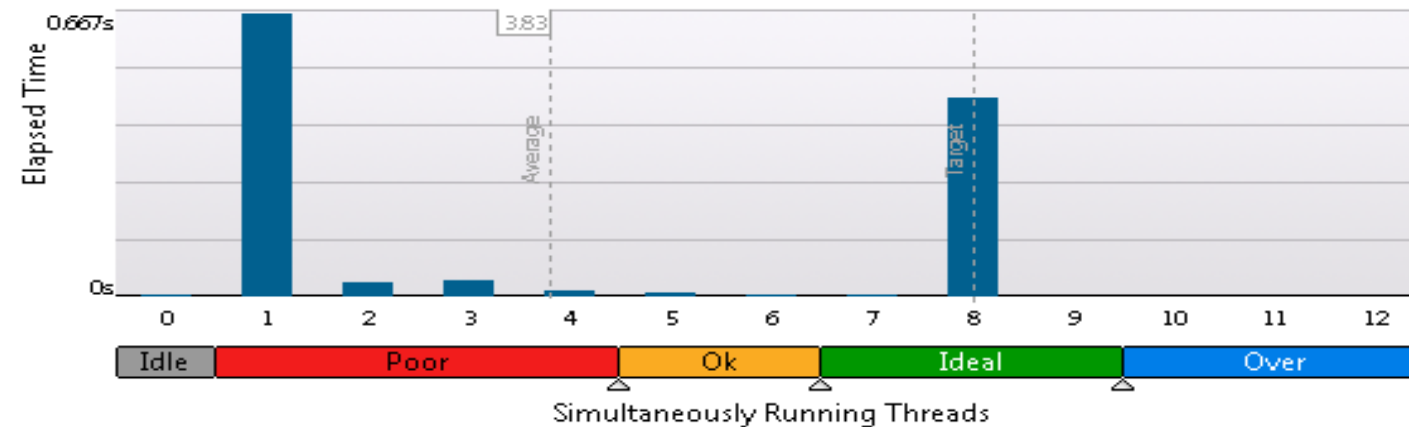
Параллельная реализация...



- ❑ Еще одна полезная диаграмма: потоки завершают расчеты в разное время и ждут.
- ❑ Налицо дисбаланс нагрузки.

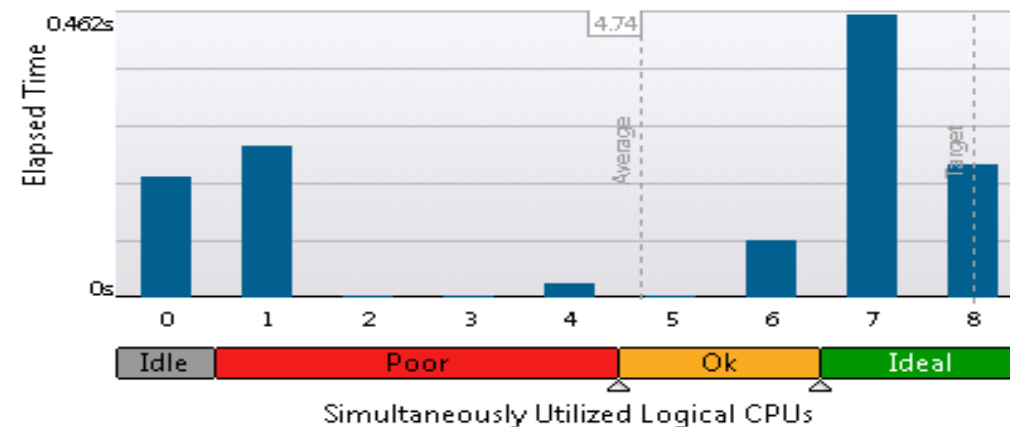
Thread Concurrency Histogram

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running on a CPU or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may be higher than CPU usage if threads are in the runnable state and not consuming CPU time.



CPU Usage Histogram

This histogram represents a breakdown of the Elapsed Time. It visualizes what percentage of the wall time the specific number of CPUs were running simultaneously. CPU Usage may be higher than the thread concurrency if a thread is spinning or executing code on a CPU while it is logically waiting.



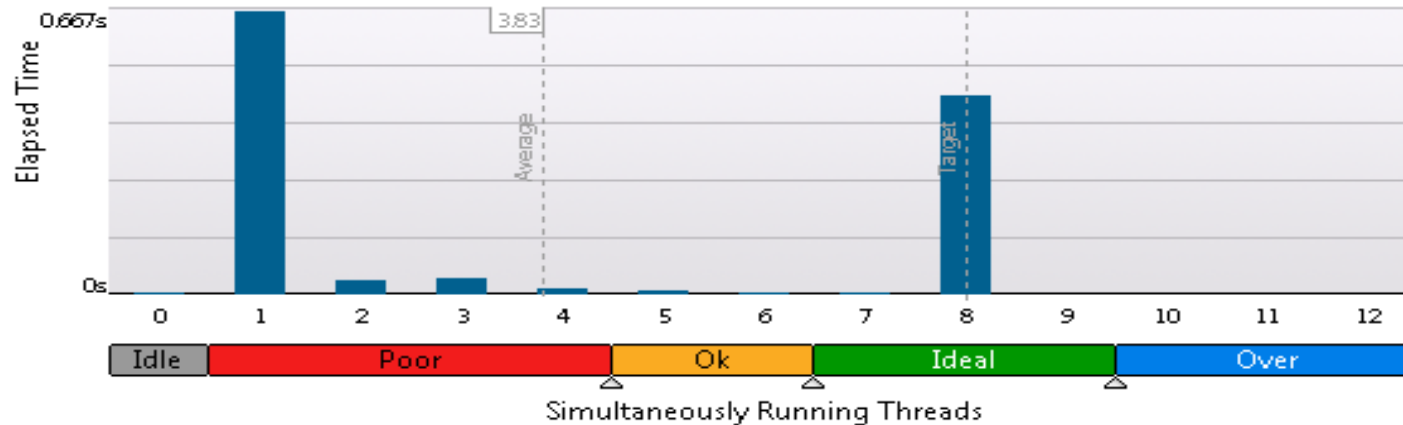
Параллельная реализация...

Данная вкладка предоставляет интегральную информацию о степени параллелизма. Из первой диаграммы видно, что в основном приложение работало в 1 и в 8 потоков. При этом если поток находится в режиме ожидания, считается, что он «работает».



Thread Concurrency Histogram

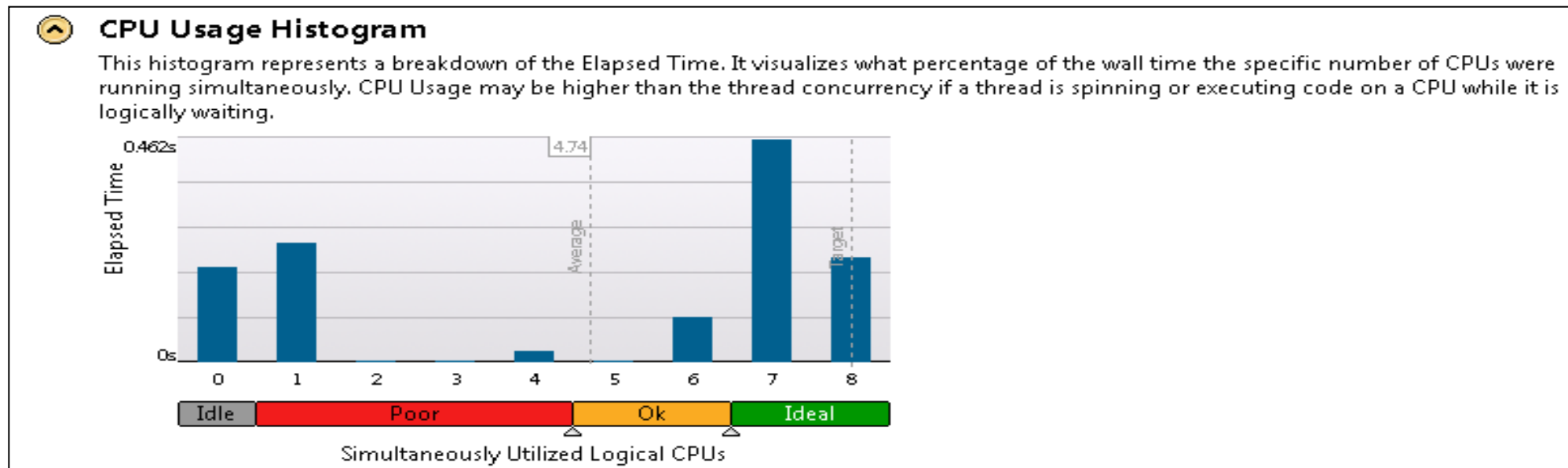
This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running on a CPU or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may be higher than CPU usage if threads are in the runnable state and not consuming CPU time.



Параллельная реализация...

Данная вкладка предоставляет интегральную информацию о степени параллелизма. Вторая диаграмма дает информацию о степени использования CPU.

Обе вкладки показывают информацию по всему приложению в целом, а не только по функции `Multiply()`.

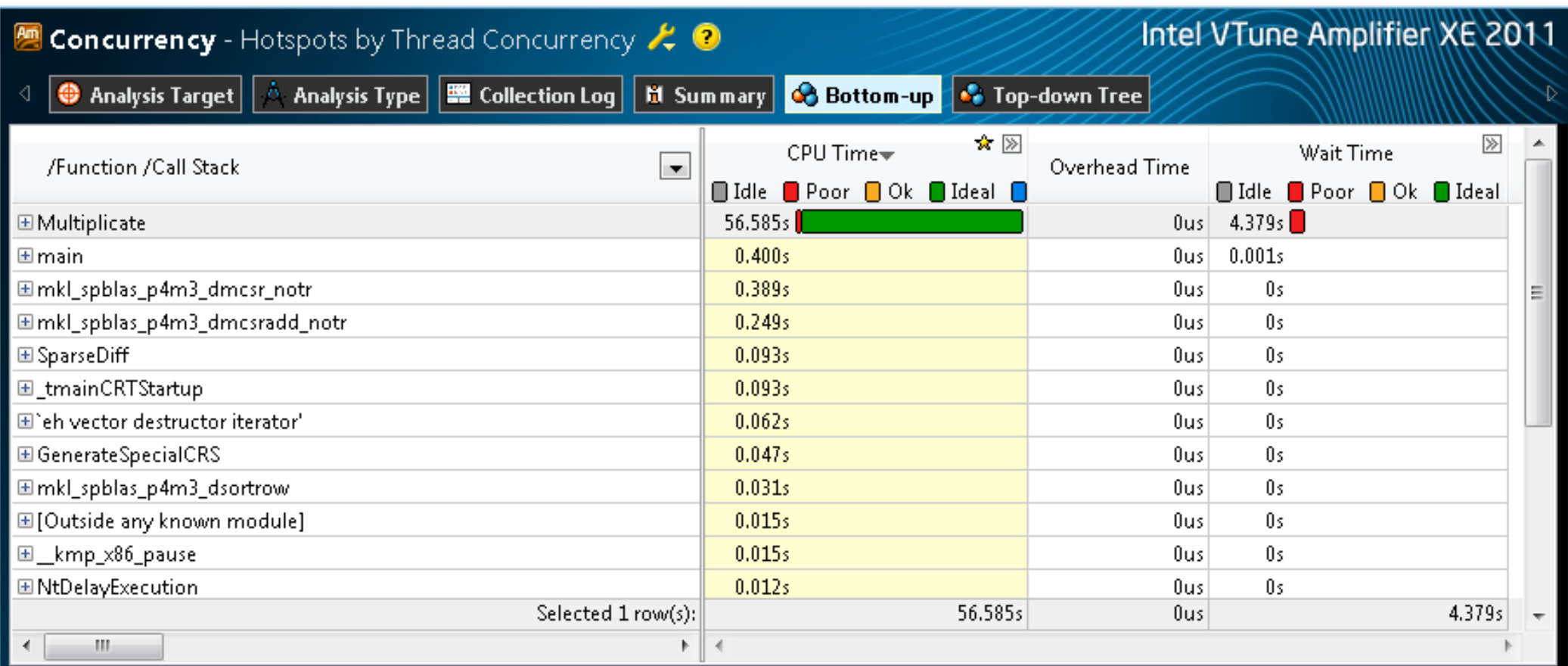


Параллельная реализация...

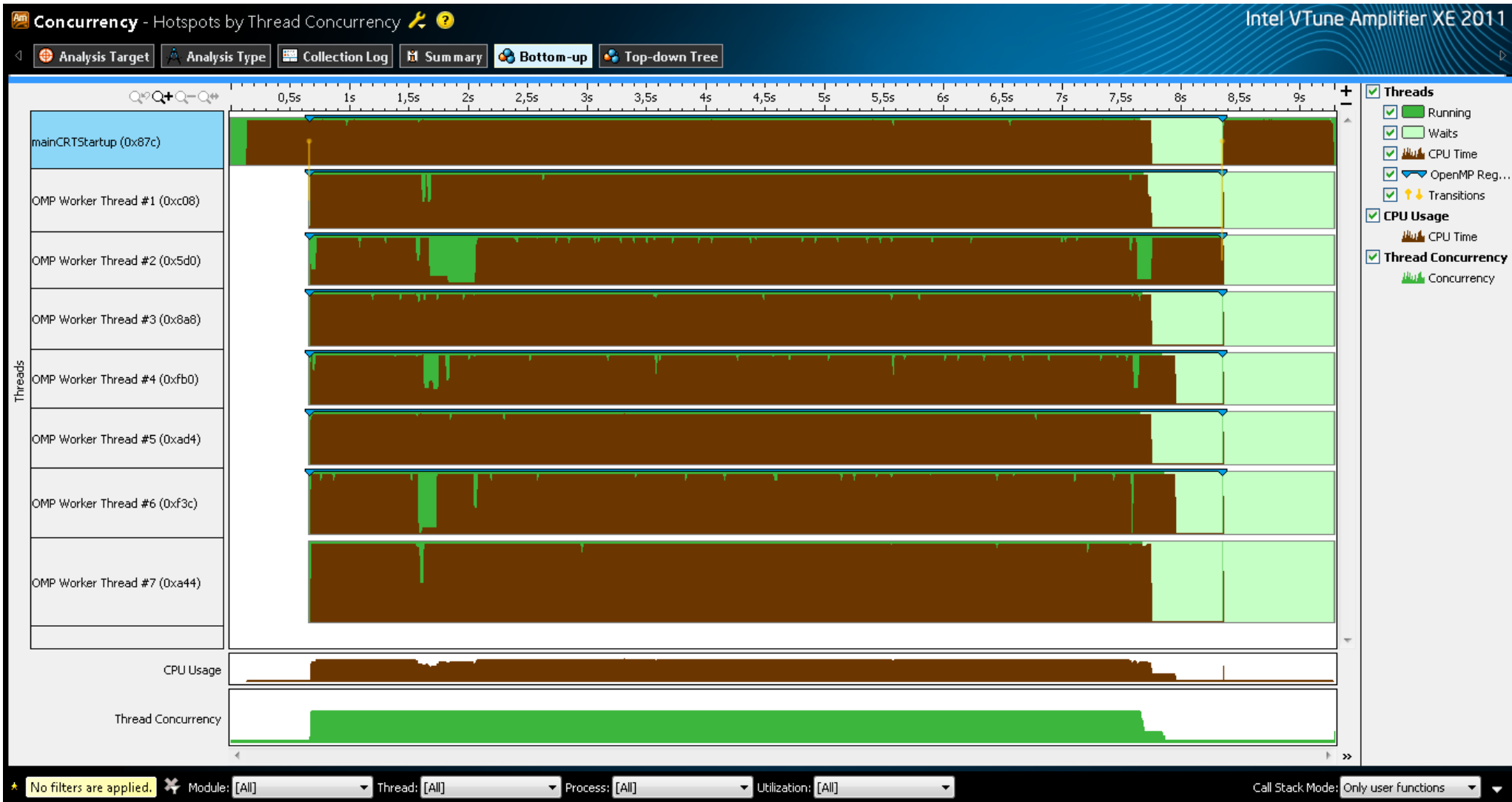
- ❑ Может быть потоки «не догружены» работой?
- ❑ Увеличим N и соберем результаты экспериментов.

Порядок матриц (N)	1 поток, t (сек)	8 потоков, t (сек)	Ускорение
10 000	2.512	1.029	2.441
15 000	5.445	1.841	2.958
20 000	9.454	2.667	3.545
25 000	14.617	3.931	3.718
30 000	20.982	5.538	3.789
35 000	34.429	7.067	4.872
40 000	44.460	8.408	5.288

Параллельная реализация...



Параллельная реализация...



Параллельная реализация...

- ❑ Действительно, с увеличением объема работы растет ускорение и снижается эффект дисбаланса нагрузки.
- ❑ Тем не менее, нужно подумать о балансировке.
- ❑ Используем возможности OpenMP.

```
#pragma omp for private(j, k) schedule(static, chunk)
```

- ❑ Выберем `chunk = 10`, запустим эксперимент.
- ❑ Предлагаем подобрать оптимальное значение `chunk` экспериментально.



Параллельная реализация...

Порядок матриц (N)	1 поток, t (сек)	8 потоков, t (сек)	Ускорение
10 000	2.512	0.920	2.730
15 000	5.445	1.591	3.422
20 000	9.454	2.215	4.268
25 000	14.617	3.198	4.571
30 000	20.982	4.444	4.721
35 000	34.429	5.568	6.183
40 000	44.460	6.394	6.953

- ❑ Ситуация существенно улучшилась.
- ❑ Спрофилируйте приложение и изучите результаты.
- ❑ Попробуйте динамическое расписание.

Параллельная реализация...

Реализация с использованием Intel Threading Building Blocks.

- ❑ Проект **07_TBB**.
- ❑ Используемая функциональность TBB:
 - Инициализация библиотеки с использованием объекта класса **task_scheduler_init**.
 - Распараллеливание цикла с помощью шаблонной функции **parallel_for()**.
 - Одномерное итерационное пространство **blocked_range**.
 - Класс-функтор, который нам предстоит разработать и который, собственно, и будет реализовывать основную часть умножения в методе **operator()**.



Параллельная реализация...

- Разработка параллельной версии на основе TBV будет очень похожа на OpenMP:
 - Продублируем по числу строк служебные вектора **columns** и **values**.
 - Создадим массив **row_index** длины «число строк + 1» и точно также будем запоминать в каждом его элементе, сколько не нулей будет содержать соответствующая строка матрицы C.
 - Фактически отличия в реализации функции **Multiply()** будут состоять в использовании шаблонной функции **parallel_for()**, которая «скроет» в себе весь содержательный код – то, что в OpenMP-версии составляло содержимое параллельной секции.



Параллельная реализация...

```
int Multiply(crsMatrix A, crsMatrix B, crsMatrix &C,
double &time) {
...
task_scheduler_init init();
vector<int>* columns = new vector<int>[N];
vector<double> *values = new vector<double>[N];
int* row_index = new int[N + 1];
memset(row_index, 0, sizeof(int) * N);
int grainsize = 10;
parallel_for(blocked_range<int>(0, A.N, grainsize),
    Multiplicator(A, B, columns, values, row_index));
int NZ = 0;
for(i = 0; i < N; i++) {
    int tmp = row_index[i]; row_index[i] = NZ; NZ += tmp;
}
row_index[N] = NZ;
InitializeMatrix(N, NZ, C);
...
}
```

Параллельная реализация...

```
class Multiplier
{
    crsMatrix A, B;
    vector<int>* columns;
    vector<double>* values;
    int *row_index;
public:
    Multiplier(crsMatrix& _A, crsMatrix& _B,
        vector<int>* &_columns, vector<double>* &_values,
        int *_row_index) : A(_A), B(_B), columns(_columns),
        values(_values), row_index(_row_index)
    {}
    void operator()(const blocked_range<int>& r) const
    {
        ...
    }
};
```

Параллельная реализация...

```
void operator() (const blocked_range<int>& r) const
{
    int begin = r.begin();
    int end = r.end();
    int N = A.N;
    int i, j, k;
    int *temp = new int[N];
    for (i = begin; i < end; i++)
    {
        ...
    }
    delete [] temp;
}
```


Параллельная реализация...

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

c:\ParallelCalculus\07_SparseMM\Release>07_TBB.exe 10000 50
OK
10000 500000 130775 5678160
0.000 0.624 0.109
```

Порядок матриц (N)	1 поток, t (сек)	8 потоков, t (сек)	Ускорение
10 000	2.512	0.624	4.026
15 000	5.445	1.139	4.781
20 000	9.454	1.731	5.462
25 000	14.617	2.589	5.646
30 000	20.982	3.620	5.796
35 000	34.429	4.758	7.236
40 000	44.460	5.850	7.600



Параллельная реализация...

- ❑ Даже без настройки гранулярности (`grainsize = 10` выбрано «наобум») мы получаем очень хорошие результаты с точки зрения масштабируемости.
- ❑ Далее рекомендуется провести эксперименты по подбору оптимального значения `grainsize`.

Задания для самостоятельной работы...

- ❑ Дополнительные задания имеют разный уровень сложности. Некоторые из них являются достаточно трудоемкими и подходят в качестве тем зачетных тем для студентов, изучающих параллельные численные методы.
- ❑ Все задания предполагают выполнение параллельных реализаций для систем с общей памятью. Сравнение производительности и точности выполняется для параллельных версий.

Задания для самостоятельной работы...

- ❑ Провести эксперименты с умножением матриц в столбцовом формате (CCS). Выявить и объяснить эффекты, связанные с соотношением времен работы разных последовательных алгоритмов. Выполнить сравнение с базовой версией, представленной в работе (матрицы в формате CRS). Разработать и настроить параллельную реализацию.
- ❑ Провести эксперименты с умножением матриц в координатном формате. Выявить и объяснить эффекты, связанные с соотношением времен работы разных последовательных алгоритмов. Выполнить сравнение с базовой версией, представленной в работе (матрицы в формате CRS). Разработать и настроить параллельную реализацию.

Задания для самостоятельной работы...

- ❑ Провести эксперименты с умножением матриц другой структуры. Рассмотреть матрицы с равным числом элементов в строках. Выявить и объяснить эффекты, связанные с соотношением времен работы разных последовательных алгоритмов. Разработать и настроить параллельную реализацию.
- ❑ Провести эксперименты с умножением матриц другой структуры. Рассмотреть матрицы с нарастающим числом элементов в строках. Выявить и объяснить эффекты, связанные с соотношением времен работы разных последовательных алгоритмов. Разработать и настроить параллельную реализацию.

Задания для самостоятельной работы...

- ❑ Провести эксперименты с умножением матриц с сохранением результата в плотную матрицу C . Рассмотреть матрицы с равным числом элементов в строках. Выявить и объяснить эффекты, связанные с соотношением времен работы разных последовательных алгоритмов. Разработать и настроить параллельную реализацию.
- ❑ Провести эксперименты с умножением матриц с сохранением результата в плотную матрицу C . Рассмотреть матрицы с нарастающим числом элементов в строках. Выявить и объяснить эффекты, связанные с соотношением времен работы разных последовательных алгоритмов. Разработать и настроить параллельную реализацию.

Задания для самостоятельной работы...

- ❑ Адаптировать использованные в работе алгоритмы для прямоугольных матриц. Выполнить программную реализацию. Провести вычислительные эксперименты.
- ❑ В наивной версии алгоритма умножения проверить, будет ли выигрыш от использования одного вектора вместо двух (см. сноску в соответствующем разделе). Исследовать вопрос о возможности получения выигрыша в производительности при отказе от использования STL.

Указание: для эксперимента инициализируйте матрицу C , в качестве NZ используйте заведомо большое число, чтобы изначально выделилось достаточно памяти. Это позволит оценить возможный выигрыш сверху.

Задания для самостоятельной работы...

- ❑ Реализовать алгоритм Густавсона для умножения разреженных матриц. Разработать параллельную реализацию. Провести вычислительные эксперименты.
- ❑ Ознакомьтесь с научной литературой по рассматриваемой теме. Изучите/разработайте другие алгоритмы умножения. Опробуйте их на практике, проведите вычислительные эксперименты. Убедитесь в корректности результатов. Сравните время работы.

Задания для самостоятельной работы

- ❑ Выполните распараллеливание символической и численной фазы матричного умножения. Проведите вычислительные эксперименты. Убедитесь в корректности результатов. Проведите анализ масштабируемости.
- ❑ Разработайте другой, более быстрый алгоритм для выполнения численной фазы. Учитывая тот факт, что приведенный вариант численной фазы уступает по скорости общему времени работы функции умножения матриц из библиотеки MKL, есть потенциал для оптимизации.
- ❑ Выясните оптимальный размер порции данных для OpenMP- и TBB-реализаций.



Использованные источники информации

- Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений. – М.: Мир, 1984.
- Писсанецки С. Технология разреженных матриц. — М.: Мир, 1988.

Дополнительная литература...

- ❑ Голуб Дж., Ван Лоун Ч. Матричные вычисления. – М.: Мир, 1999.
- ❑ Тьюарсон Р. Разреженные матрицы. – М.: Мир, 1977.
- ❑ Bik A.J.C. The software vectorization handbook. Applying Multimedia Extensions for Maximum Performance. – IntelPress, 2004.
- ❑ Gerber R., Bik A.J.C., Smith K.B., Tian X. The Software Optimization Cookbook. High-Performance Recipes for the Intel® Architecture. – Intel Press, 2006.
- ❑ Касперски К. Техника оптимизации программ. Эффективное использование памяти. – BHV, 2003.
- ❑ Stathis P., Cheresiz D., Vassiliadis S., Juurlink B. Sparse Matrix Transpose Unit // 18th International Parallel and Distributed Processing Symposium (IPDPS'04) – Papers, vol. 1, 2004.
[\[http://ce.et.tudelft.nl/publicationfiles/869_1_IPDPS2004paper.pdf\]](http://ce.et.tudelft.nl/publicationfiles/869_1_IPDPS2004paper.pdf)

Дополнительная литература

- ❑ Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ. – М.: МЦНМО, 2001.
- ❑ Gustavson F. Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition // ACM Transactions on Mathematical Software (TOMS), Volume 4 Issue 3, Sept. 1978. – Pp. 250-269.
- ❑ Корняков К.В., Мееров И.Б., Сиднев А.А., Сысоев А.В., Шишков А.В. Инструменты параллельного программирования в системах с общей памятью. – Учебное пособие / Под ред. проф. В.П. Гергеля. – Н. Новгород: Изд-во Нижегородского госуниверситета, 2010. – 201 с.
- ❑ Белов С.А., Золотых Н.Ю. Численные методы линейной алгебры. Лабораторный практикум. – Н. Новгород: Изд-во Нижегородского госуниверситета, 2005. – 264 с.



Ресурсы сети Интернет

- ❑ Buttari A. Software Tools for Sparse Linear Algebra Computations.
[<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.7162&rep=rep1&type=pdf>]
- ❑ Официальная страница Джеймса Деммеля (James Demmel Home Page) [<http://www.cs.berkeley.edu/~demmel/>]
- ❑ Demmel J. U.C. Berkeley Math 221 Home Page: Matrix Computations / Numerical Linear Algebra
[<http://www.cs.berkeley.edu/~demmel/ma221>]
- ❑ Demmel J. U.C. Berkeley CS267/EngC233 Home Page: Applications of Parallel Computers
[http://www.cs.berkeley.edu/~demmel/cs267_Spr10/]
- ❑ Demmel J. CS170: Efficient Algorithms and Intractable Problems
[http://www.cs.berkeley.edu/~demmel/cs170_Spr10/#starthere]
- ❑ Справочная система к библиотеке PETSc.
[<http://www.geo.uu.nl/~govers/petsc/node36.html#Node36>]



Авторский коллектив

- ❑ 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

Авторы выражают благодарность Сафоновой Я., Филиппенко С.,
Лебедеву С. за помощь в проведении экспериментов с различными
реализациями алгоритмов.

