« Параллельные вычисления »

Н.Е.Богданов

23 марта 2016 г.

Содержание

1 Задание

Вычислить пересечение трех множеств.

2 Программа работы

1. Для алгоритма из полученного задания написать последовательную программу на языке C или C++, реализующую этот алгоритм.

2. Для созданной последовательной программы необходимо написать 3-5 тестов, которые покрывают основные варианты функционирования программы. Для создания тестов можно воспользоваться механизмом Unit-тестов среды NetBeans, или описать входные тестовые данные в файлах. При использовании NetBeans необходимо в свойствах проекта установить ключ компилятора -pthread.

3. Проанализировать полученный алгоритм, выделить части, которые могут быть распараллеленны, разработать структуру параллельной программы. Определить количество используемых потоков, а также правила и используемые объекты синхронизации.

4. Согласовать разработанную структуру и детали реализации параллельной программы с преподавателем.

5. Написать код параллельной программы и проверить ее корректность на созданном ранее наборе тестов. При необходимости найти и исправить ошибки.

6. Провести эксперименты для оценки времени выполнения последовательной и параллельной программ. Проанализировать полученные результаты.

7. Сделать общие выводы по результатам проделанной работы: Различия между способами проектирования последовательной и параллельной реализаций алгоритма, Возможные способы выделения параллельно выполняющихся частей, Возможные правила синхронизации потоков, Сравнение времени выполнения последовательной и параллельной программ, Принципиальные ограничения повышения эффективности параллельной реализации по сравнению с последовательной.

3 Решение

Из решения представленной последовательной программы(3.1), видно что для распараллеливания оптимально подходит разделение циклов на блоки итераций которые выполняет каждый поток независимо от других(3.2).

3.1 Последовательная программа

Листинг 1: Алгоритм последовательной программы

```
//структура для получения размера получаемого
//пересечения множеств - size
//и времени за которое отработал алгоритм - time
struct data{
        int size;
        LONGLONG time;
};
/** T - тип данных
T *A, *B, *C - входные масссивы для пересечения.
T *M - результат пересечения.
массивы проинициализированны заранее
и имеют одинаковый размер
int length - длинна массивов множества*/
data algorithm0(T *A,T *B,T *C,T *M,int length){
start_time();
int index=0;
for(int i = 0; i < length; i++){
        for(int j = 0; j < length; j++){
                if(A[i] == B[j]){
                        for(int e = 0; e < length;e++){
                                if( B[j]==C[e]){
                                        M[index]= C[e];
                                        index++;
                                }
                        }
                }
        }
}
data d;
d.time = end_time();
```

```
d.size = index;
return d;
```

3.2 Многопоточная программа Windows API

В качестве объекта синхронизации была выбрана критическая секция.

Листинг 2: Многопоточная программа средствами Windows API

```
//функция для распаралеливания цикла
//при помощи CreateThread
DWORD WINAPI function1(LPVOID lpParam){
//номер потока от 0 до MAX_THREADS-1
int* pDataArray = (int*)lpParam;
unsigned int start =
 ((unsigned int)*pDataArray)*(length/MAX_THREADS);
unsigned int end =
 (((unsigned int)*pDataArray) + 1)*(length/MAX_THREADS);

for(unsigned int i = start; i < end; i++){
        for(unsigned int j = 0; j < length; j++){
        if(A[i] == B[j]){
                for(unsigned int e = 0; e < length;e++){
                if( B[j]==C[e]){
                //вызов критической секции
                        EnterCriticalSection(&cr_s);
                        M[ThradArrayIndex]= C[e];
                        ThradArrayIndex++;
                //освобождение критической секции
                        LeaveCriticalSection(&cr_s);
                }
                }
        }
        }
}
return 0;
}
```

3.3 Многопоточная программа с использованием OpenMP

Распараллеливаем только верхний цикл.

Листинг 3: Многопоточная программа с использованием OpenMP

```
data algorithm2(T *A,T *B,T *C,T *M,int length){
start_time();
int index=0;
#pragma omp parallel for
for(int i = 0; i < length; i++){
        for(int j = 0; j < length; j++){
                if(A[i] == B[j]){
                        for(int e = 0; e< length;e++){
                                if( B[j]==C[e]){
                                #pragma omp critical
                                {
                                        M[index] = C[e];
                                        index++;
                                }
                                }
                        }
                }
        }
}
data d;
```

Распараллеливаем каждый цикл.

Листинг 4: Многопоточная программа с использованием OpenMP

```
data algorithm1(T *A,T *B,T *C,T *M,int length){
start_time();
int index=0;
#pragma omp parallel for
for(int i = 0; i < length; i++){
        #pragma omp parallel for
        for(int j = 0; j < length; j++){
                if(A[i] == B[j]){
                        #pragma omp parallel for
                        for(int e = 0; e< length;e++){
                                if( B[j]==C[e]){
                                #pragma omp critical
                                {
```

```
                                                M[index] = C[e];
                                                index++;
                                            }
                                            }
                                    }
                                }
                            }
}
data d;
d.time = end_time();
d.size = index;
return d;
}
```

4 Тестирование

Таблица замеров времени выполнения тестов для последовательной и параллельной программы, выполнены на компьютерах с процессорами:

- Intel Core i3 M330 2.13GHz(2 физических ядра с технологией Hyper-Threading)
- Intel Core i5 2300 2.8GHz (4 физических ядра без технологии Hyper-Threading)

Под 64 разрядной ОС Windows 7. Для замеров времени использован API интерфейс счётчика с большой разрядной сеткой (https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408(v=vs.85).aspx) и поддерживающий высокую точность, основанной на аппаратном счётчике TSC.

Тип элементов массива - **int**, размер множества в тестах от 2^{10} до 2^{17}, время замеров указанно в микросекундах.

Выходное множество полученное работой последовательной программы, соответствует множеству полученному от многопоточной программы при многократных запусках. Входные данные для множеств A, B и C для различных типов тестов:

- Тест 1. A - последовательность чисел $i \times 2 + 1$, где - i от 0 до N-1, B - от 0 до N-1, C - $i \times 2$ i от 0 до N-1, пересечение множеств $\emptyset$.
- Тест 2. Для каждого множества задаются не повторяющиеся псевдослучайные числа, пресечение множеств $\emptyset$.
- Тест 3. A - последовательность чисел от N до 1, B - последовательность первых N-1 чисел Фибоначчи, C - от 0 до N-1, пересечение множеств, множество первых N-1 чисел Фибоначчи.
- Тест 4. A - i*2, B - i*3, C - i*5, пересечение множеств, множество чисел кратных 30.
- Тест 5. A, B и C одинаковы, последовательность чисел от 0 до N-1, т.е. пересечением множеств является множество элементов от 0 до N.

Где N - количество элементов в множестве. Тесты подобраны специально чтобы от теста к тесту увеличить количество заходов в конкурентную область кода окружённую блоками синхронизации.

5 Анализ результатов

Проводился запуск как при различном размере множества для Core i3 (5.2), так и при различном количестве потоков для Intel Core i3 и Intel Core i5 (5.1).

5.1 Производительность при разном количестве потоков

Ниже представлены графики средних значений выполнения тестов при разном количестве потоков для OpenMP на Intel Core i3 и Intel Core i5 и графики дисперсий. Программы были запущенны с количеством потоков от 2 до 16. Результаты от запуска к запуску меняются, в таблицах представлены средние значения и дисперсия при количестве измерений 50, данные представлены для размера множества 2^{14} элементов и компиляции программы с ключом Release. Графики сравнения результатов для разных тестов при различном количестве потоков можно увидеть в разделе Дополнительные графики(8).

Таблица 1: Сравнения для реализации многопоточности через Open MP(Листинг 3) и Windows API(Листинг 2) для Intel Core i3 с указанием дисперсии для 3 теста.[1]

количество потоков	OpenMP	Windows API
2	1,894362934 ± 0,01552658	2,959281532 ± 0,042425957
3	2,367448031 ± 0,009361491	3,335753197 ± 0,080353874
4	3,031010127 ± 0,008447384	3,151030799 ± 0,06274519
5	2,193511257 ± 0,002808351	3,238055394 ± 0,062430442
6	2,447983599 ± 0,030279963	3,303484226 ± 0,046896102
7	2,520745152 ± 0,056614046	3,275392987 ± 0,069842821
8	2,457159773 ± 0,074375152	3,318793811 ± 0,041226487
9	2,397709762 ± 0,010224168	3,299328912 ± 0,061658737
10	2,592496716 ± 0,026949706	3,125160578 ± 0,070093624
11	2,466092001 ± 0,034374156	3,32126876 ± 0,037001564
12	2,507973435 ± 0,048375805	3,219525723 ± 0,048954077
13	2,487121008 ± 0,033782739	3,140928692 ± 0,032430239
14	2,54280867 ± 0,03522073	3,254906856 ± 0,076830226
15	2,402674285 ± 0,047772088	3,2264896 ± 0,056638749
16	2,432340369 ± 0,036996234	3,180035529 ± 0,04337758

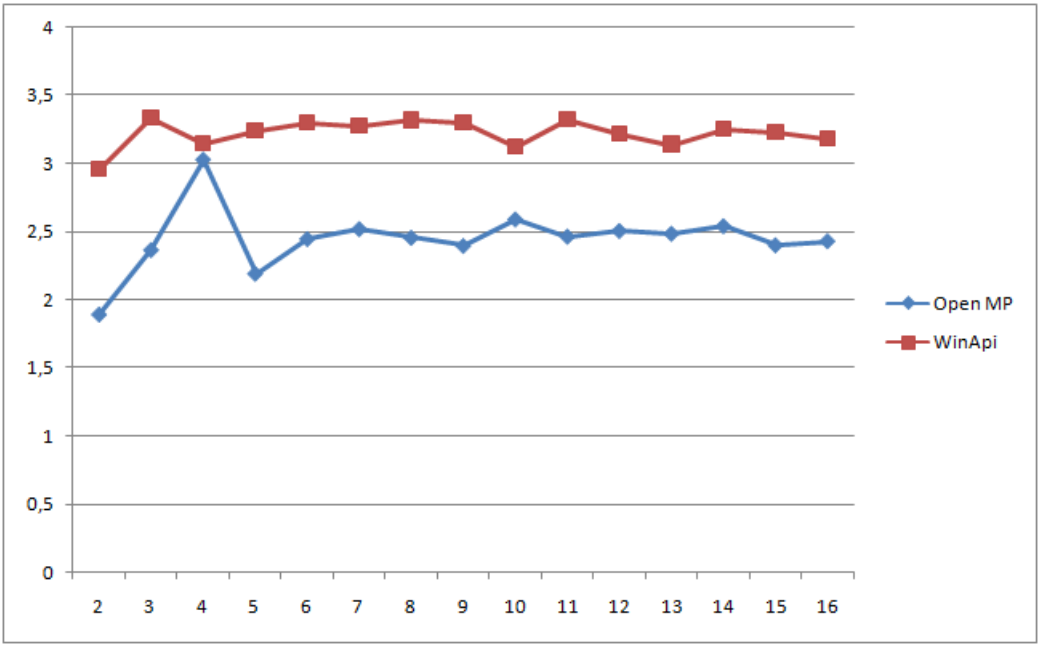

Рис. 1: по оси Y - коэффициент отношения последовательной к параллельной, по оси X - количество потоков,результаты для Core i3

Таблица 2: Сравнения для реализации многопоточности через Open MP(Листинг 3)на процессорах Intel Core i3 и Inel Core i5, для 3 теста.[1]

количество потоков	Core i3	Core i5
2	1,894362934 ± 0,01552658	2,271356258 ± 0,000105818
3	2,367448031 ± 0,009361491	3,407000946 ± 0,00044838
4	3,031010127 ± 0,008447384	4,397038821 ± 0,019930802
5	2,193511257 ± 0,002808351	2,807566235 ± 0,000787844
6	2,447983599 ± 0,030279963	3,255586011 ± 0,052119712
7	2,520745152 ± 0,056614046	3,751570311 ± 0,133561913
8	2,457159773 ± 0,074375152	3,249020164 ± 0,275875862
9	2,397709762 ± 0,010224168	3,365004825 ± 0,001080306
10	2,592496716 ± 0,026949706	3,557202463 ± 0,113951497
11	2,466092001 ± 0,034374156	3,346508033 ± 0,176471681
12	2,507973435 ± 0,048375805	3,452180501 ± 0,095178175
13	2,487121008 ± 0,033782739	3,6097189 ± 0,00420886
14	2,54280867 ± 0,03522073	3,74479593 ± 0,090539201
15	2,402674285 ± 0,047772088	3,584766547 ± 0,136796063
16	2,432340369 ± 0,036996234	3,692476817 ± 0,099739954

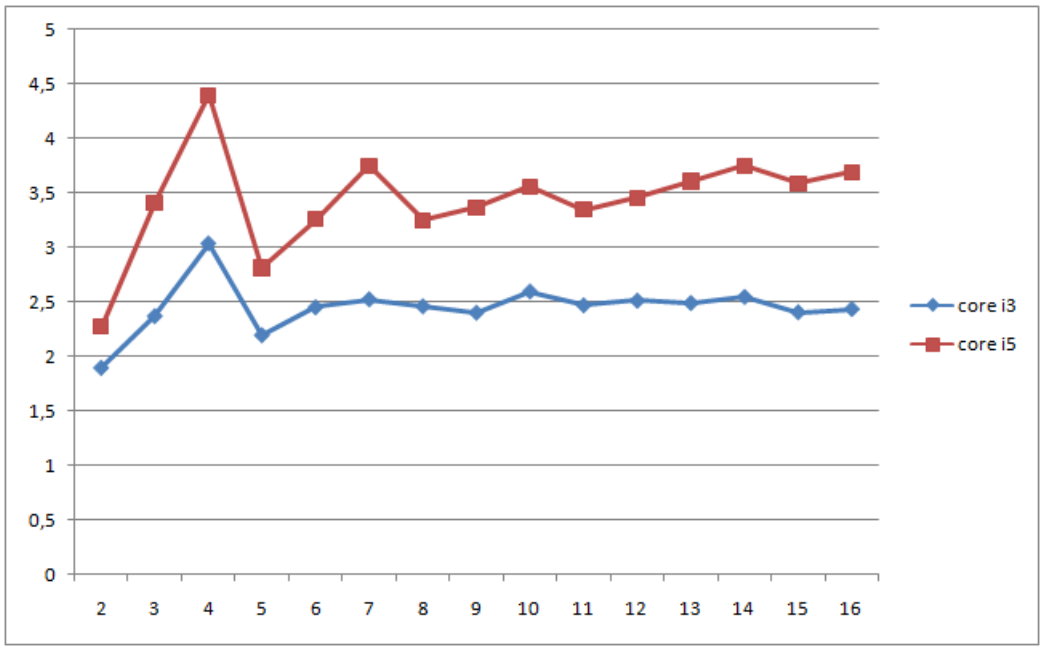

Рис. 2: по оси Y - коэффицент отношения последовательной к параллельной, по оси X - количество потоков,результаты для Open MP

[1] По полученным результатам можно заключить, что производительность четного количества потоков выше чем нечётного, а оптимальное количество потоков соответствует количеству логических ядер процессора. Т.к. данный процессор использует технологию Hyper-threading то каждое физическое ядро процессора определяется операционной системой как два логических ядра,если же нет технологии Hyper-threading, то количество потоков должно соответствовать количеству физических ядер процессора.

Разница производительности для Open MP и Windows API связанна с накладными ресурсами работы фреймворка Open MP.

[1]значения для других тестов содержатся в прикреплённом файле см. пункт Приложения

5.2 Временные характеристики для Windows API

Ниже приведены результаты и графики для теста с размером множества 2^{17} элементов.

Таблица 3: Временные характеристики для программ с ключом Debug.

№ теста	последовательная,μs	параллельная,μs	коэффициент отношения
1	$1{,}23406106\times10^8$	$6{,}0595134\times10^7$	2,03656792
2	$8{,}2243323\times10^7$	$3{,}7307467\times10^7$	2,204473517
3	$8{,}2194345\times10^7$	$3{,}7353587\times10^7$	2,200440482
4	109567826×10^8	$4{,}9822935\times10^7$	2,19914435
5	$1{,}64433976\times10^8$	$7{,}4546488\times10^7$	2,205791049

Таблица 4: Временные характеристики для программ с ключом Release.

№ теста	последовательная,μs	параллельная,μs	коэффициент отношения
1	$5{,}4171361\times10^7$	$1{,}9725392\times10^7$	2,746275511
2	$4{,}1499630\times10^7$	$1{,}2535936\times10^7$	3,310453244
3	$4{,}1524151\times10^7$	$1{,}2545656\times10^7$	3,309842945
4	$5{,}0035276\times10^7$	$1{,}6777756\times10^7$	2,982238864
5	$6{,}6853514\times10^7$	$2{,}5395378\times10^7$	2,632507144

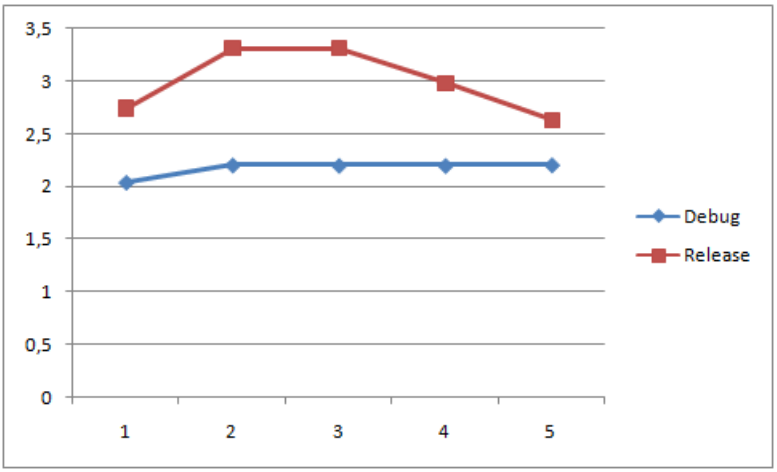

Рис. 3: График отношения времени выполнения,по оси Y - отношение последовательной программы к параллельной, по оси X - номер теста.(Листинг 2)

5.3 Временные характеристики для OpenMP

Замеры аналогичные пункту (5.2), для OpenMP, но рассмотрены 2 случая распараллеливания: только верхнего цикла (Листинг 3) и распараллеливания каждого цикла (Листинг 4). Результаты при распараллеливании каждого цикла:

Таблица 5: Временные характеристики для программы (Листинг 4) с ключом Debug.

№ теста	последовательная,μs	параллельная,μs	коэффициент отношения
1	$1{,}23014345\times10^8$	$7{,}0256500\times10^7$	1,750931871
2	$8{,}1876900\times10^7$	$4{,}3163156\times10^7$	1,896916435
3	$8{,}1846762\times10^7$	$4{,}3163749\times10^7$	1,89619215
4	$1{,}09335318\times10^8$	$5{,}7464682\times10^7$	1,90265245
5	$1{,}63664568\times10^8$	$8{,}6275258\times10^7$	1,897004678

Таблица 6: Временные характеристики для программы (Листинг 4) с ключом Release.

№ теста	последовательная,μs	параллельная,μs	коэффициент отношения
1	66290098×10^7	$3{,}3783783\times10^7$	1,962186946
2	49701756×10^7	$2{,}0704824\times10^7$	2,400491596
3	49694791×10^7	$2{,}0633515\times10^7$	2,408450087
4	60753979×10^7	$2{,}7556317\times10^7$	2,204720573
5	82870378×10^7	$4{,}1251537\times10^7$	2,008904008

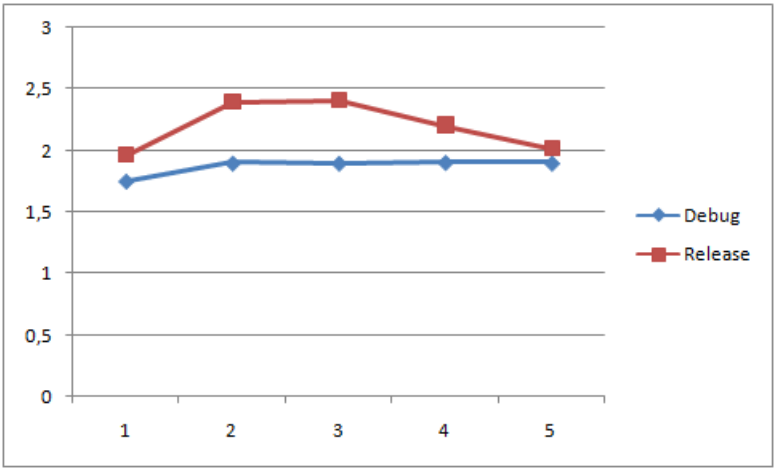

Рис. 4: График отношения времени выполнения тестов последовательной к параллельной программе для OpenMP(Листинг 4)

Результаты при распараллеливании только верхнего цикла:

Таблица 7: Временные характеристики для программы (Листинг 3) с ключом Debug.

№ теста	последовательная,μs	параллельная,μs	коэффициент отношения
1	$1{,}23014345\times10^8$	$6{,}2267420\times10^7$	1,975581211
2	$8{,}1876900\times10^7$	$3{,}7624727\times10^7$	2,176146022
3	$8{,}1846762\times10^7$	$3{,}7152256\times10^7$	2,203009206
4	$1{,}09335318\times10^8$	$4{,}9411584\times10^7$	2,212746671
5	$1{,}63664568\times10^8$	$7{,}4292714\times10^7$	2,202969298

Таблица 8: Временные характеристики для программы (Листинг 3) с ключом Release.

№ теста	последовательная,μs	параллельная,μs	коэффициент отношения
1	$6{,}6290098\times10^7$	$2{,}6110405\times10^7$	2,538838367
2	$4{,}9701756\times10^7$	$1{,}6771260\times10^7$	2,963507572
3	$4{,}9694791\times10^7$	$1{,}6823851\times10^7$	2,953829715
4	$6{,}0753979\times10^7$	$2{,}0932438\times10^7$	2,90238428
5	$8{,}2870378\times10^7$	$2{,}9334040\times10^7$	2,825058465

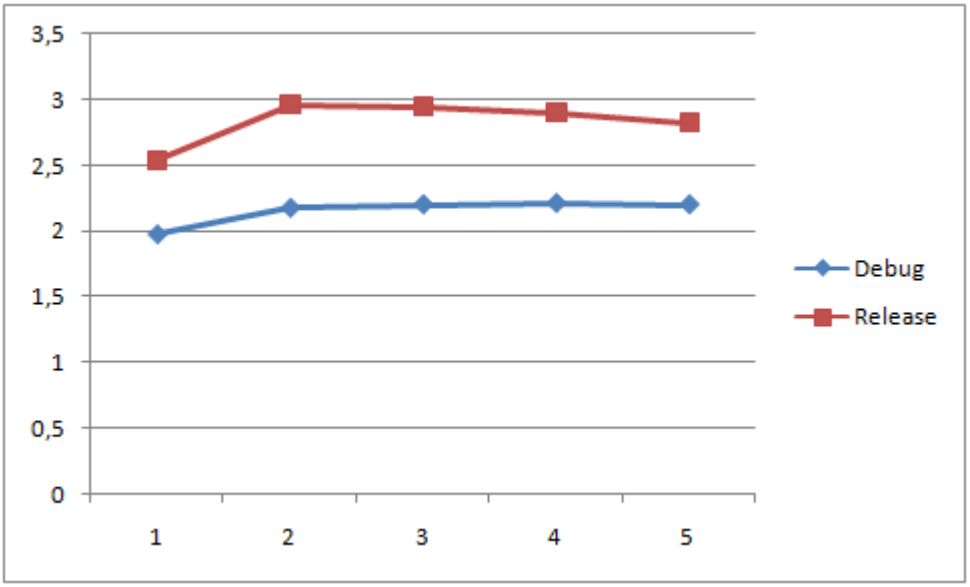

Рис. 5: График отношения времени выполнения тестов последовательной к параллельной программе для Open MP(Листинг 3)

6 Вывод

Фреймворк Open MP требует небольших накладных расходов от 5 до 25% в отличии от программы использующей напрямую Windows API, но зато значительно облегчает процесс разработки многопоточной программы,за счёт уменьшения рутинных операций инициализации и запуска потоков, так же облегчает работу с объектами синхронизации конкурентных блоков кода.

Оптимальное количество потоков для программы соответствует количеству логических ядер процессора.

В зависимости от входных данных может меняться эффективность многопоточности.

Чем дольше время выполнения последовательной программы тем более значимо видна разница при распараллеливании, но стоит понимать что прибавка к производительности изменяется от приложения к приложению. Скорость выполнения некоторых программ может даже уменьшиться. Это, в первую очередь, связано с « системой повторения » , занимающей необходимые вычислительные ресурсы, отчего и начинают « голодать » другие потоки.

7 Приложения

- Исходные коды многопоточной программы использующей Windows API находится в папке папке Midle3MassWinAPI.
- Исходные коды многопоточной программы использующей Open MP находится в папке папке Midle3MassOMP.
- Результаты тестов при различном размере множеств в файле "результаты тестов при разном размере множеств.xlsx".
- Результаты тестов при различном количестве потоков в файле "результаты тестов при разном количестве потоков.xlsx".

8 Дополнительные графики

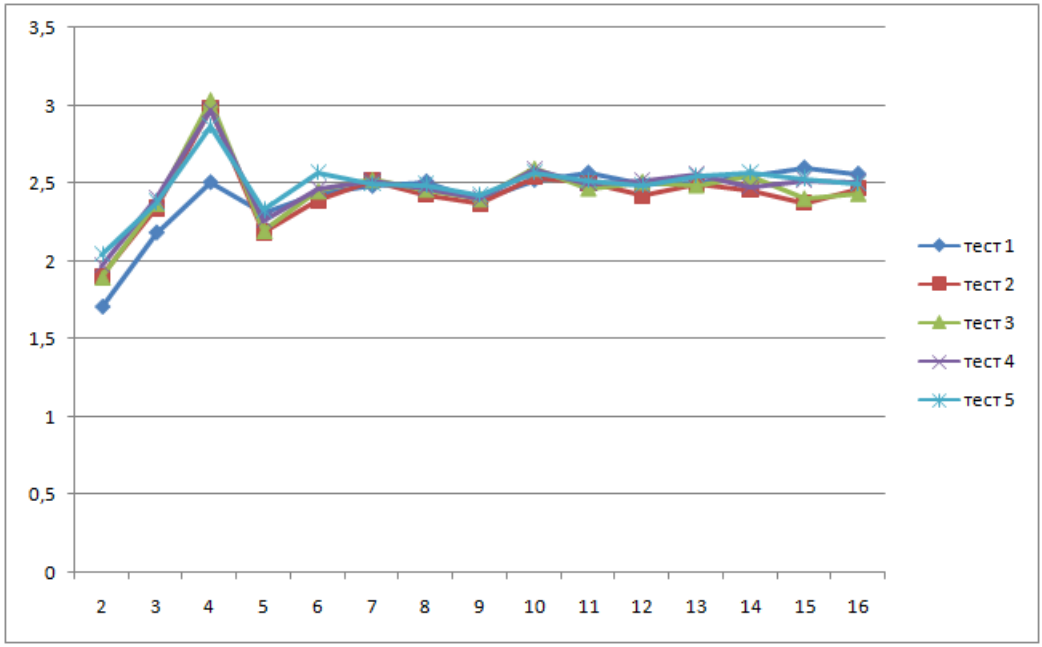

Рис. 6: по оси Y - коэффициент отношения последовательной к параллельной, по оси X - количество потоков,средние значения для Core i3

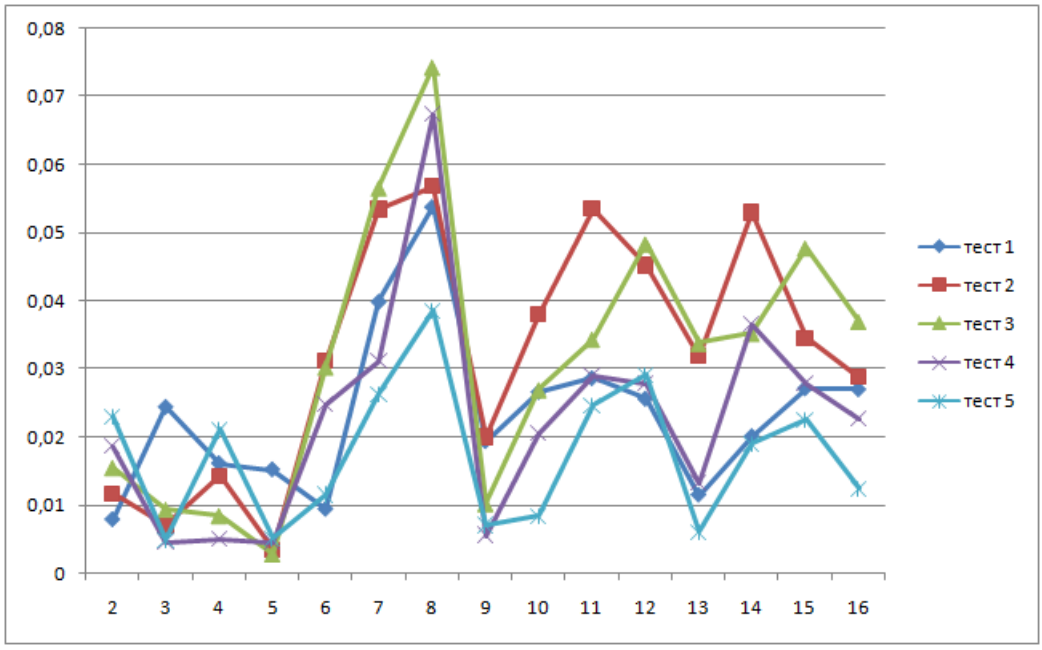

Рис. 7: по оси Y - коэффициент отношения последовательной к параллельной, по оси X - количество потоков,значение дисперсий для Core i3

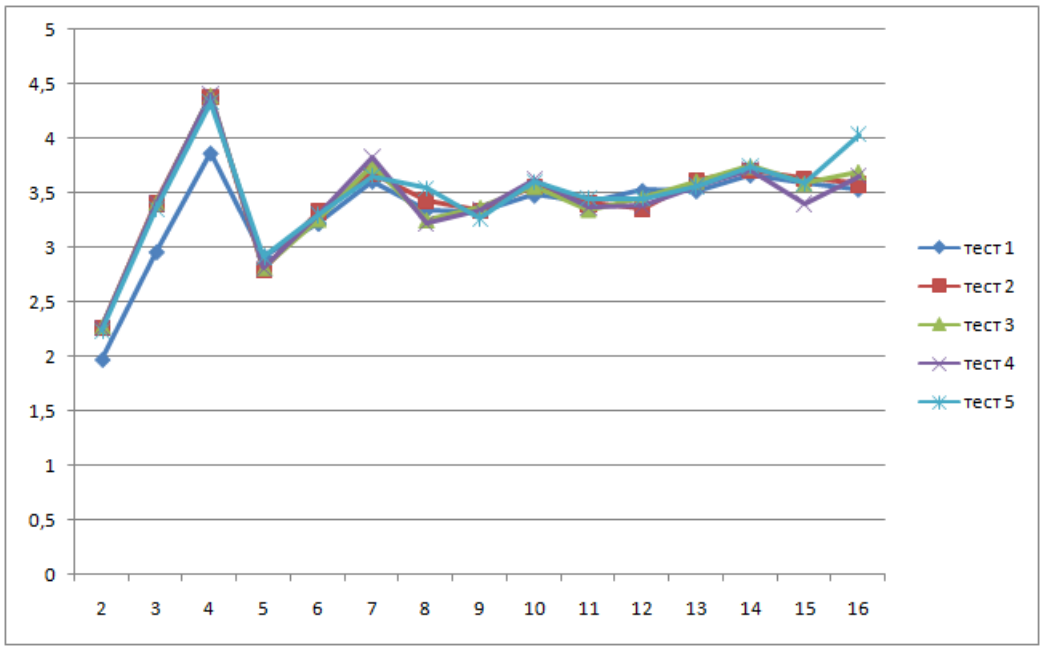

Рис. 8: по оси Y - коэффициент отношения последовательной к параллельной, по оси X - количество потоков,средние значения для Core i5

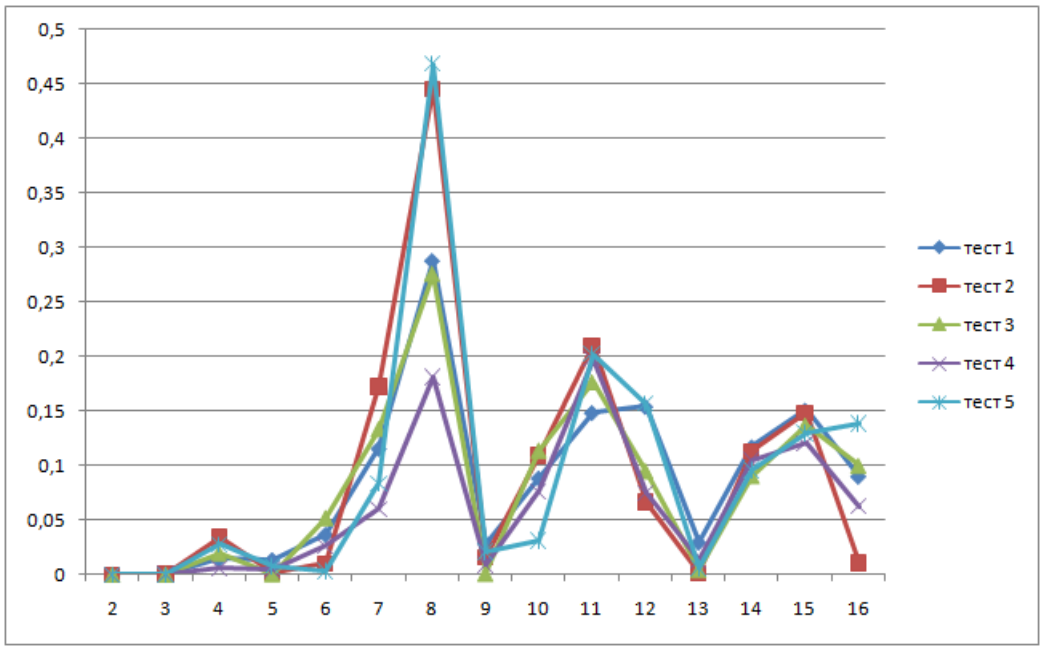

Рис. 9: по оси Y - коэффициент отношения последовательной к параллельной, по оси X - количество потоков,значение дисперсий для Core i5