Нижегородский государственный университет им. Н.И. Лобачевского

Параллельные численные методы

Лабораторная работа 1: Поиск простых чисел

При поддержке компании Intel

Кустикова В.Д., Сиднев А.А., Сысоев А.В.
Кафедра математического обеспечения ЭВМ

Содержание

- Описание приложения
- Краткое описание инструментов пакета Intel Parallel Studio
  - Возможности Intel Parallel Composer
  - Возможности Intel Parallel Inspector
  - Возможности Intel Parallel Amplifier
- Подходы к распределению вычислительной нагрузки:
  - Подход #1: разделение множества чисел на одинаковые части по числу потоков
  - Подход #2: разделение множества чисел на четные и нечетные
  - Подход #3: разделение множества чисел на небольшие группы

Авторы выражают благодарность за идею и предоставленные фрагменты текста Кириллу Корнякову и Александру Шишкову

Цели

- Рассмотрение на примере задачи разложения чисел на простые множители некоторых вопросов, возникающих при распараллеливании алгоритмов на системах с общей памятью.
- Приобретение навыков анализа и сравнения различных подходов к распараллеливанию с помощью инструментов Intel Parallel Studio.

Разложение множества чисел на простые сомножители

- Задача: разложить на простые множители (факторизовать) числа из диапазона от 1 до N.
- Используется алгоритм, который основан на попытке деления факторизуемого числа на каждое из меньших его чисел:
  - Если остаток от деления равен нулю, то очередной множитель запоминается, после чего производится повторная попытка деления на это же число.
  - При нахождении каждого множителя, факторизуемое число делится на него, и алгоритм завершает работу, когда частное от очередного деления становится равным единице.

Пример разложения на простые множители

- Пример: 12 = ?*?*...*?

  12 / 2 = 6; // пробуем разделить на 2 раз

  6 / 2 = 3; // пробуем разделить на 2 еще раз

  3 / 2 = 1.5; // берем следующий делитель

  3 / 3 = 1; // СТОП! – получили единицу

Пример разложения на простые множители

- Пример: 12 = ?*?*...*?

  12 / **2** = 6; // пробуем разделить на 2 раз

  6 / **2** = 3; // пробуем разделить на 2 еще раз

  3 / 2 = 1.5; // берем следующий делитель

  3 / **3** = 1; // СТОП! – получили единицу

- Результат: 12 = 2 * 2 * 3
- Замечание: данный алгоритм факторизации использован только для демонстрации возможностей пакета Intel Parallel Studio
- На практике используют алгоритмы Полларда, Диксона и др.

Intel Parallel Studio

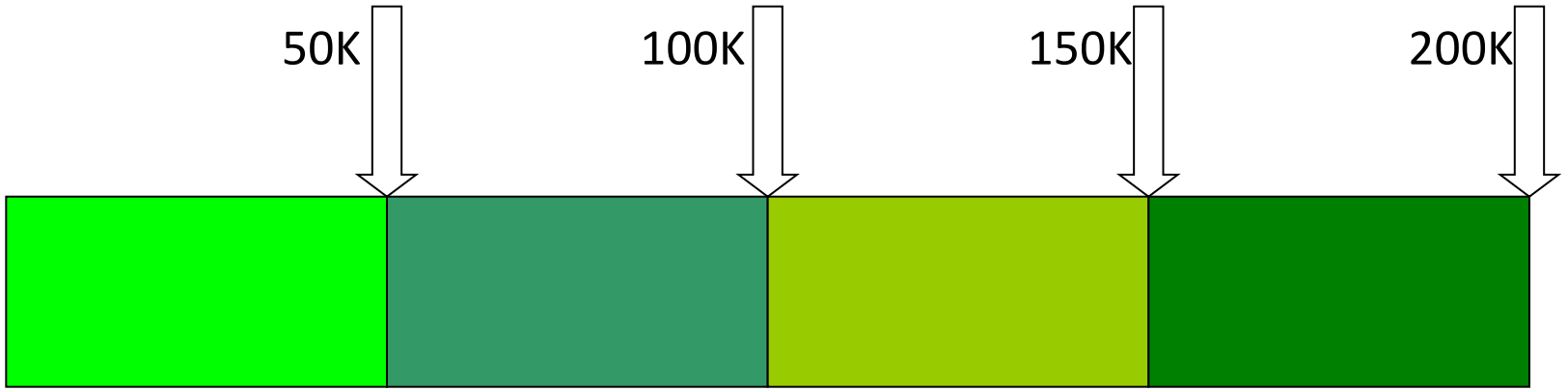
- Intel Parallel **Composer** предназначен для быстрого подключения средств параллельной обработки к компилятору языка C/C++ с использованием обширного набора библиотек с функциями многопоточной обработки.
- Intel Parallel **Inspector** обеспечивает поиск ошибок в коде приложений для любых моделей параллельного программирования.
- Intel Parallel **Amplifier** представляет собой анализатор производительности последовательных и параллельных приложений для поиска узких мест в алгоритме.

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 SP1

Подход #1: разделение множества чисел на одинаковые части по числу потоков

- Стратегия распределения нагрузки между потоками:




```
// tid – идентификатор потока
1. start ← (NUM_NUMBERS / NUM_THREADS) * tid + 1;
2. end   ← (NUM_NUMBERS / NUM_THREADS) * (tid + 1) + 1;
```

Открытие проекта

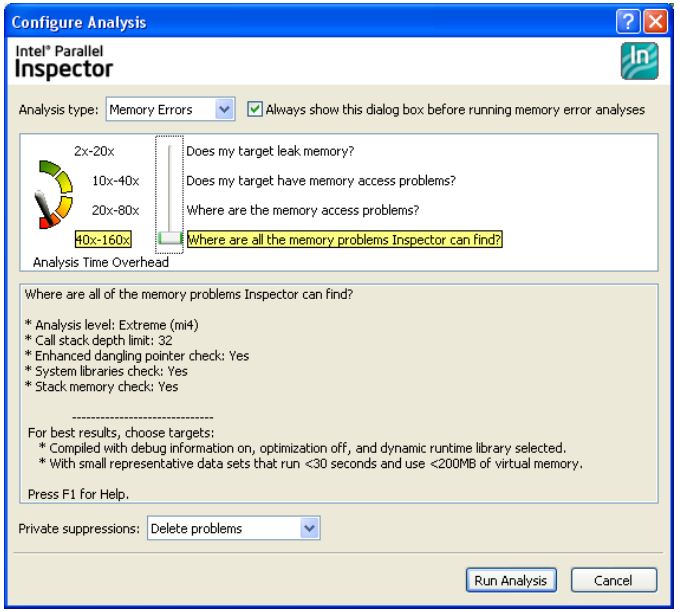
- Откройте решение “.\01_PrimeNumbers.sln”.
- Сделайте проект **01_EqualPartition** рабочим.
- Измените количество создаваемых потоков по количеству ядер.
- Соберите Debug версию проекта.
- Запустите Debug версию приложения, убедитесь, что приложение упало.

Intel Parallel Inspector (IPI)

- Инструмент поиска ошибок для приложений разрабатываемых на C/C++, функционирующих на базе операционной системы Microsoft Windows, использующих преимущества многопоточности.
- Memory Errors:
  - поиск ошибок работы с памятью.
- Threading Errors:
  - поиск ошибок многопоточности.

IPI – поиск ошибок многопоточности

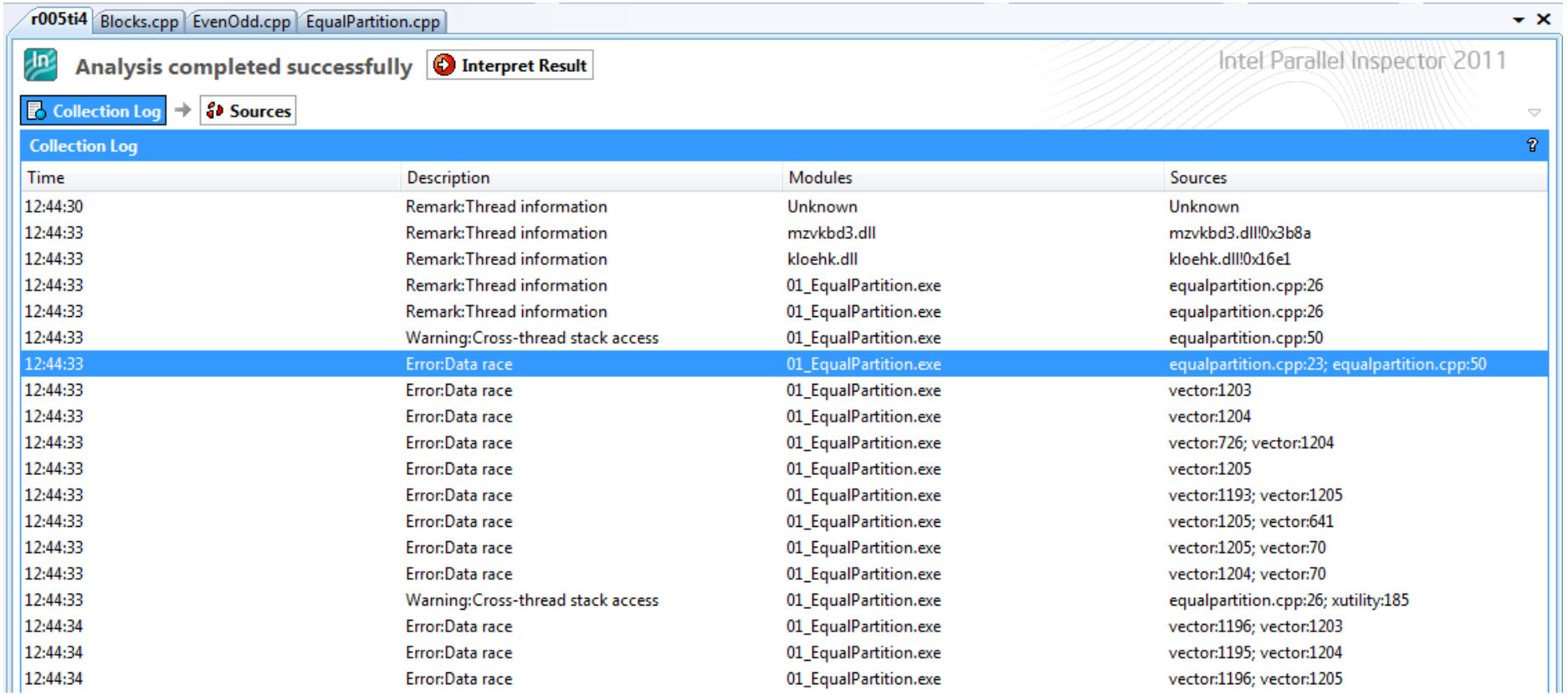
- Схема поиска ошибок многопоточности с использованием Intel Parallel Inspector:
  - собрать “Debug” версия программы;
  - выбрать в IPI режим работы “Threading errors”;
  - выбрать режим анализа “Analysis level: Extreme”;
  - нажать Run Analysis.



- Найдите ошибки многопоточности в проекте **01_EqualPartition**

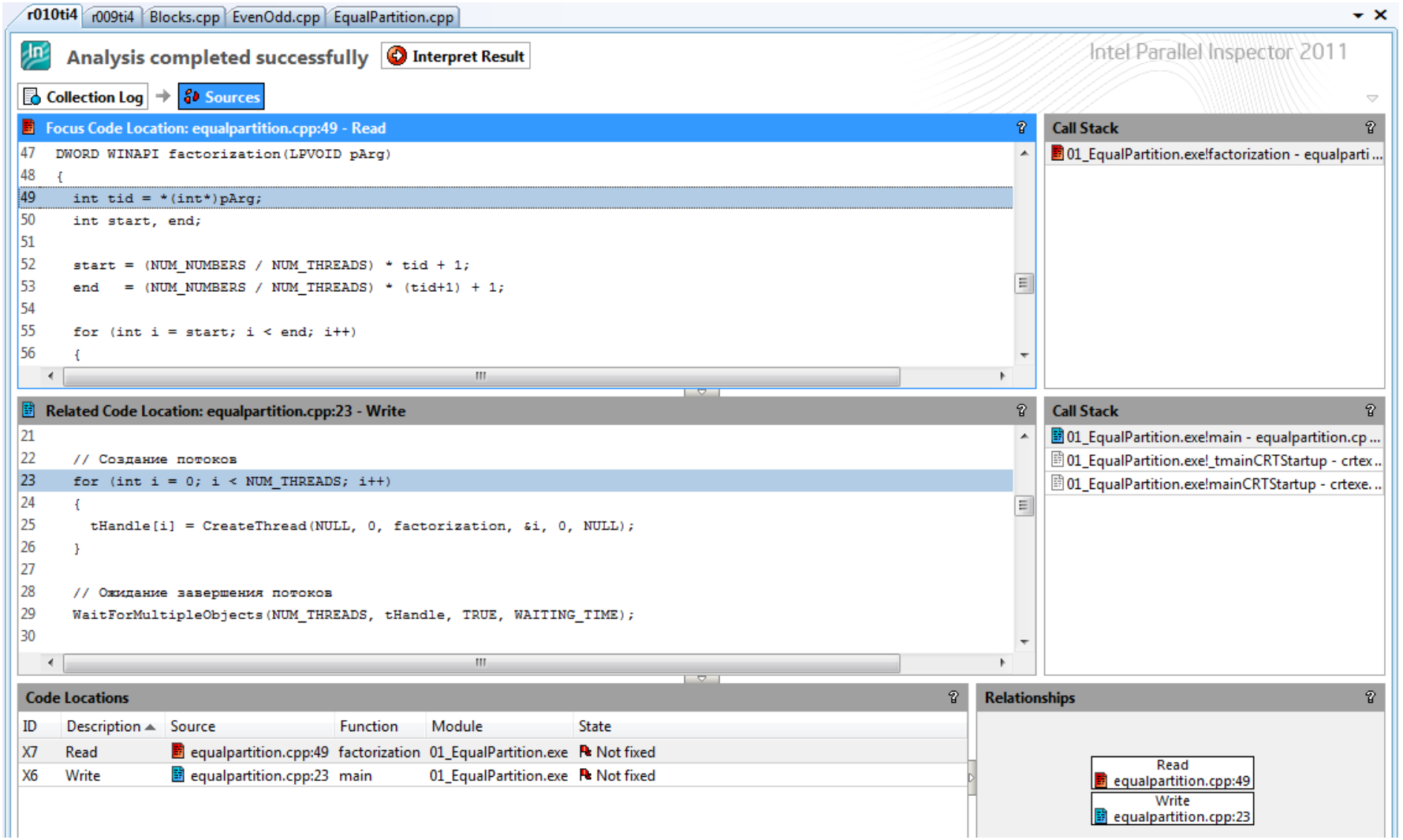
Гонка данных (data race) (1)

- В программе найдена гонка данных:



Time	Description	Modules	Sources
12:44:30	Remark:Thread information	Unknown	Unknown
12:44:33	Remark:Thread information	mzvkbd3.dll	mzvkbd3.dll!0x3b8a
12:44:33	Remark:Thread information	kloehk.dll	kloehk.dll!0x16e1
12:44:33	Remark:Thread information	01_EqualPartition.exe	equalpartition.cpp:26
12:44:33	Remark:Thread information	01_EqualPartition.exe	equalpartition.cpp:26
12:44:33	Warning:Cross-thread stack access	01_EqualPartition.exe	equalpartition.cpp:50
12:44:33	Error:Data race	01_EqualPartition.exe	equalpartition.cpp:23; equalpartition.cpp:50
12:44:33	Error:Data race	01_EqualPartition.exe	vector:1203
12:44:33	Error:Data race	01_EqualPartition.exe	vector:1204
12:44:33	Error:Data race	01_EqualPartition.exe	vector:726; vector:1204
12:44:33	Error:Data race	01_EqualPartition.exe	vector:1205
12:44:33	Error:Data race	01_EqualPartition.exe	vector:1193; vector:1205
12:44:33	Error:Data race	01_EqualPartition.exe	vector:1205; vector:641
12:44:33	Error:Data race	01_EqualPartition.exe	vector:1205; vector:70
12:44:33	Error:Data race	01_EqualPartition.exe	vector:1204; vector:70
12:44:33	Warning:Cross-thread stack access	01_EqualPartition.exe	equalpartition.cpp:26; xutility:185
12:44:34	Error:Data race	01_EqualPartition.exe	vector:1196; vector:1203
12:44:34	Error:Data race	01_EqualPartition.exe	vector:1195; vector:1204
12:44:34	Error:Data race	01_EqualPartition.exe	vector:1196; vector:1205

Гонка данных (data race) (2)



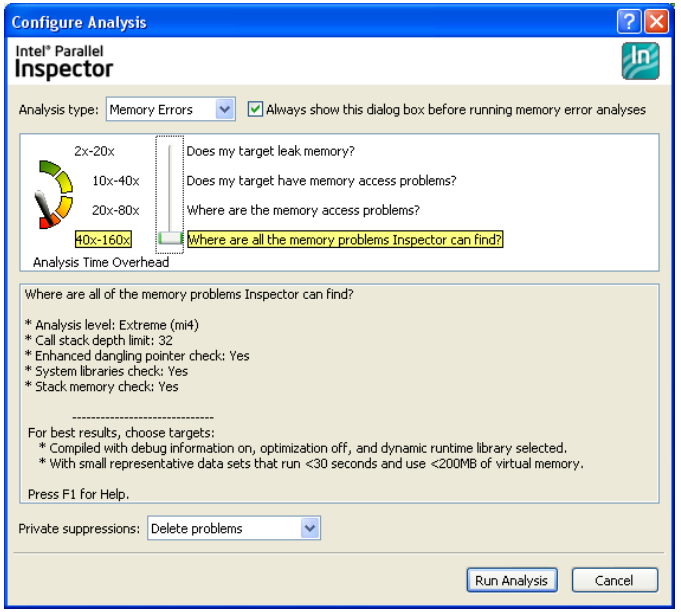
Правильная реализация программы

```
int *tNum = new int[NUM_THREADS];
// создание потоков
for (int i = 0; i < NUM_THREADS; i++)
{
  tNum[i] = i;
  tHandle[i] = CreateThread(NULL, 0, factorization,
                            &tNum[i], 0, NULL);
}
```

Убедитесь, что программа отрабатывает корректно!

IPI – поиск ошибок работы с памятью

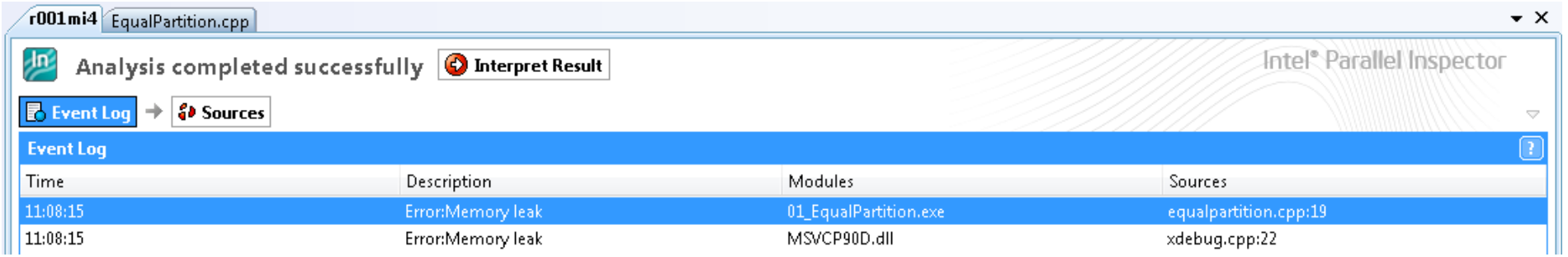
- Схема поиска ошибок работы с памятью с использованием Intel Parallel Inspector:
  - собрать “Debug” версия программы;
  - выбрать в IPI режим работы “Memory errors”;
  - выбрать режим анализа “Analysis level: Extreme”;
  - нажать Run Analysis.



- Найдите ошибки работы с памятью в проекте **01_EqualPartition**

Утечка памяти

- В программе найдена утечка памяти:



- Одна из типичных ошибок в программах: забыли освободить используемую память (сделать *delete*).

- Исправьте ошибку!

Intel Parallel Amplifier (1)

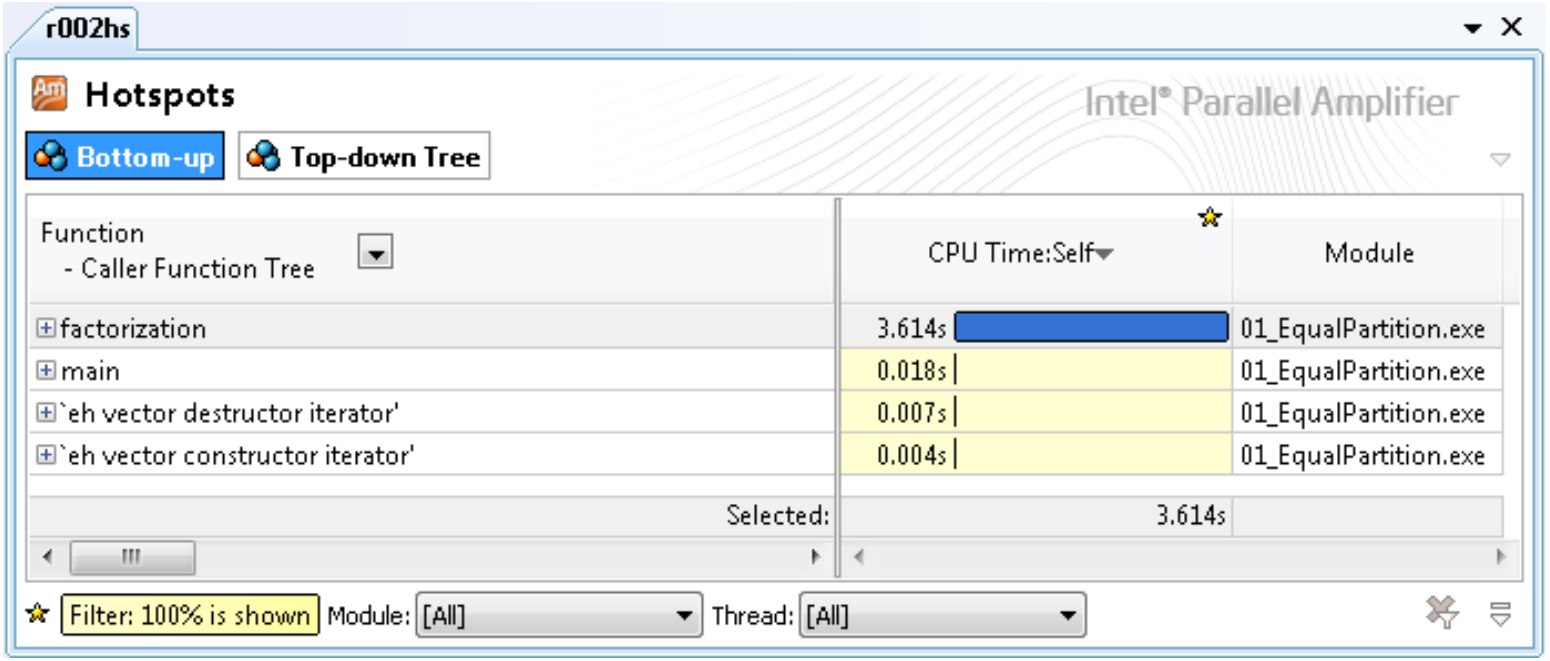
- **Hotspots**. «На что программа тратит вычислительное время процессора?» Необходимо знать те места в программе, Hotspot-функции, где больше всего тратится вычислительных ресурсов при исполнении, а также тот путь, по которому программа в эти места попала, т.е. стэк вызовов.
- **Concurrency**. «Какова степень параллелизации кода?» Часто бывает, что ожидаемый прирост производительности, например, при переходе от 4-ядерной системы к 8-ядерной, так и не достигается. Поэтому необходима оценка эффективности параллельного кода, которая дала бы представление о том, насколько полно используются ресурсы микропроцессора.

Intel Parallel Amplifier (2)

- **Locks and Waits**. «Где программа простаивает в ожидании синхронизации или операции ввода-вывода?» Поняв, что программа плохо масштабируется, можно найти, где именно и какие именно объекты синхронизации стали на пути к хорошей параллельности. Возможно необходимо пересмотреть реализацию алгоритмов, а может, и всю параллельную инфраструктуру приложения.

Анализ эффективности («Горячие» точки)

- Соберите Release версию проекта.
- Запустите Intel Parallel Amplifier в режиме Hotspots и найдите самую “медленную” функцию.



- Время, потраченное на обработку: 3.643 с

«Горячая» точка

- Основное время программа тратит на выполнение операции получения остатка от деления и сравнение:

```
60      for (int j = 2; j < number; j++)                    0.689s
61      {
62        if (number == 1) break;
63
64        int r;
65        r = number % j;                                   2.246s
66        if (r == 0)                                       1.553s
67        {
68          number /= j;
69          divisors[idx].push_back(j);                     0.187s
70          j--;
71        }
72      }                                                   0.396s
73    }
```

Необходимо минимизировать количество операций сравнения!

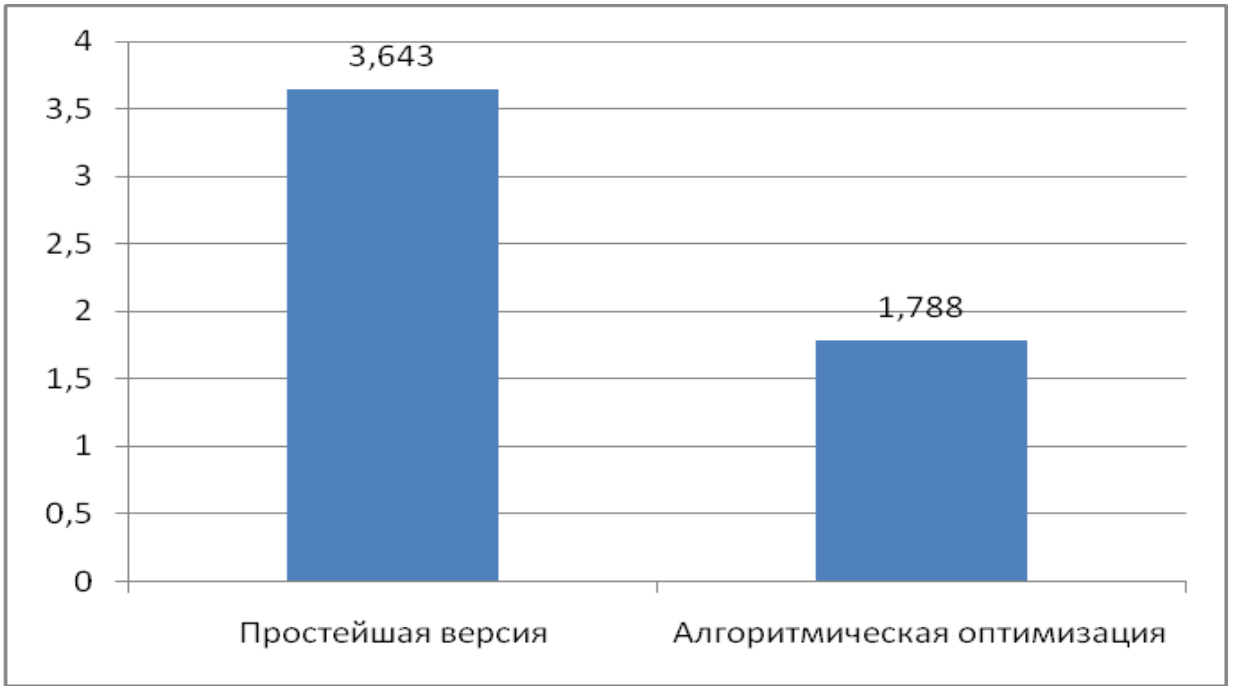
Алгоритмическая оптимизация

- Используемый алгоритм основан на попытке деления факторизуемого числа на каждое из меньших его чисел. Это избыточный алгоритм!
- Достаточно выполнять деления до величины, равной половине факторизуемого числа.
- Внесите соответствующие изменения в потоковую функцию.

Алгоритмическая оптимизация

- Запустите Intel Parallel Amplifier в режиме Hotspots и оцените полученное ускорение.

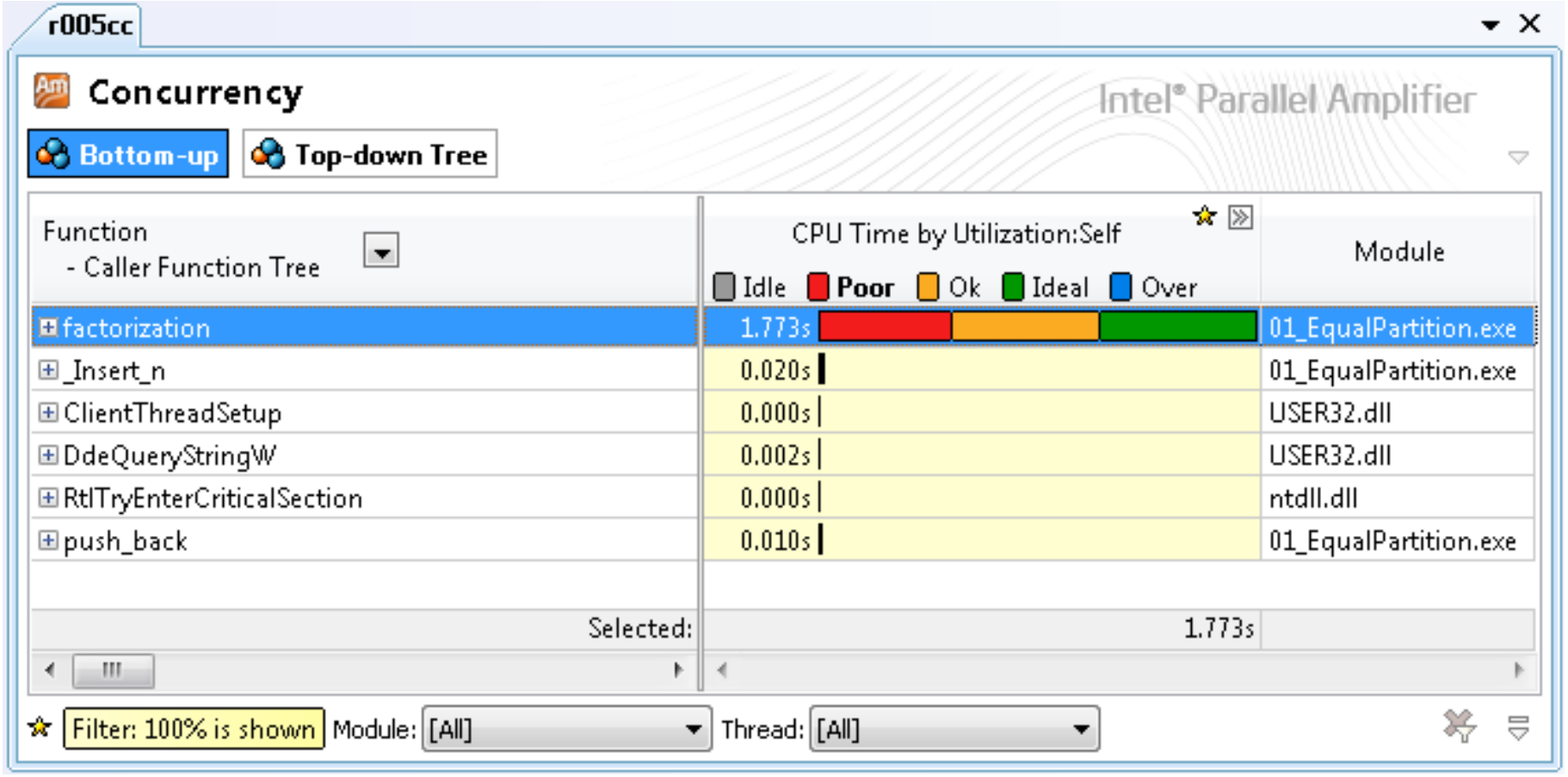
- Время, потраченное на обработку: 1.788 с



-49 %

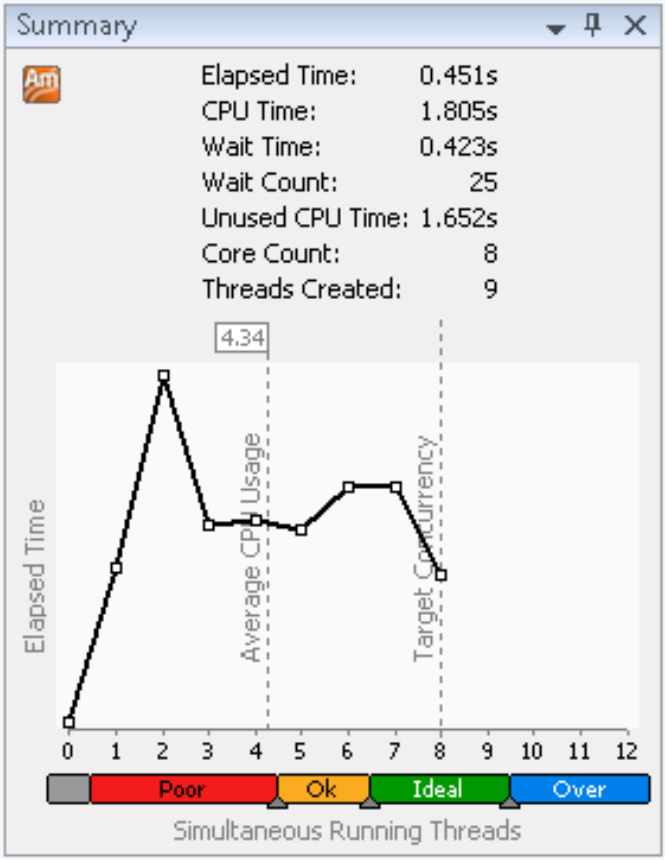
Анализ эффективности (Степень параллелизма)

- Запустите Intel Parallel Amplifier в режиме Concurrency и оцените степень параллелизма.



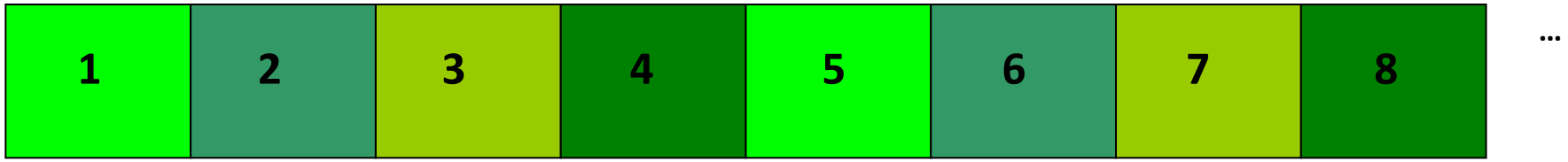
Анализ эффективности

- Видно, что программа достаточно длительное время работает в 2 потока



Подход #2: разделение множества чисел на четные и нечетные

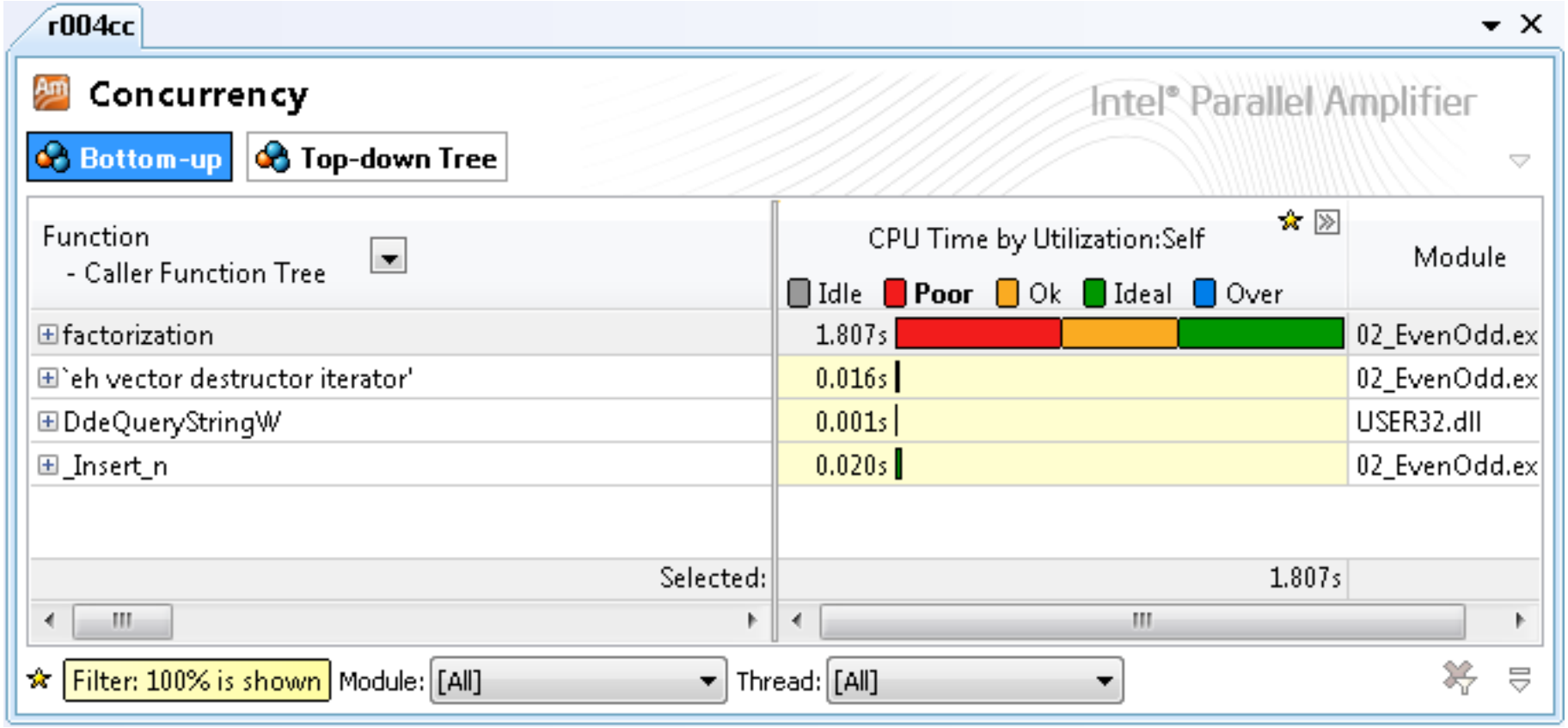
- Стратегия распределения нагрузки между потоками (откройте проект 02_EvenOdd):




```
// tid – идентификатор потока
// i – индекс числа в наборе чисел, факторизуемых потоком
1. number = i * NUM_THREADS + tid + 1;
```

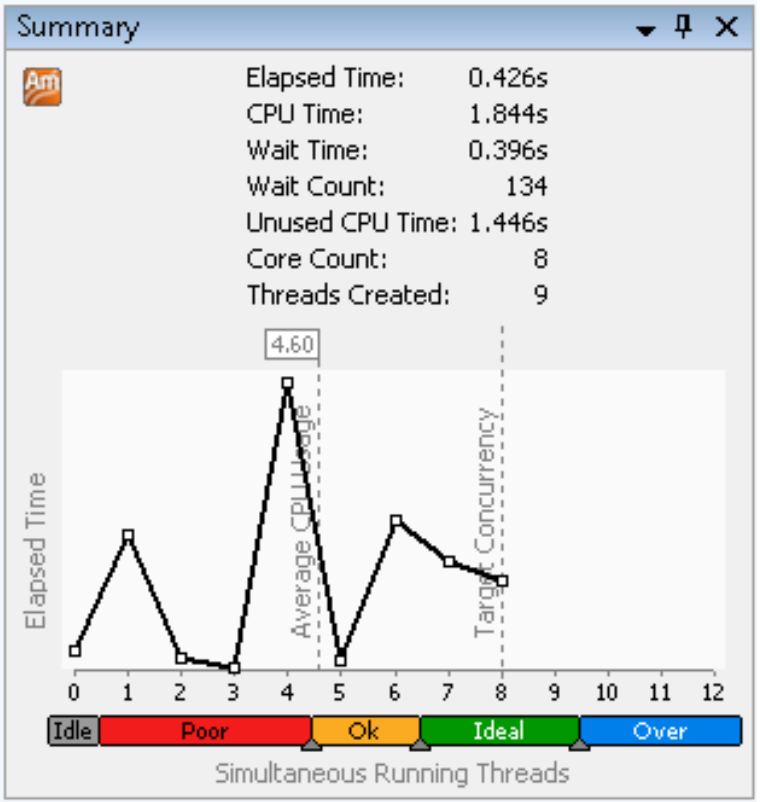
Подход #2. Анализ эффективности

- Запустите Intel Parallel Amplifier в режиме Concurrency и оцените степень параллелизма.



Подход #2. Оценка степени параллелизма

- Большую часть времени программа работает в 4 потока



Но! Четные числа факторизуется проще, чем нечетные

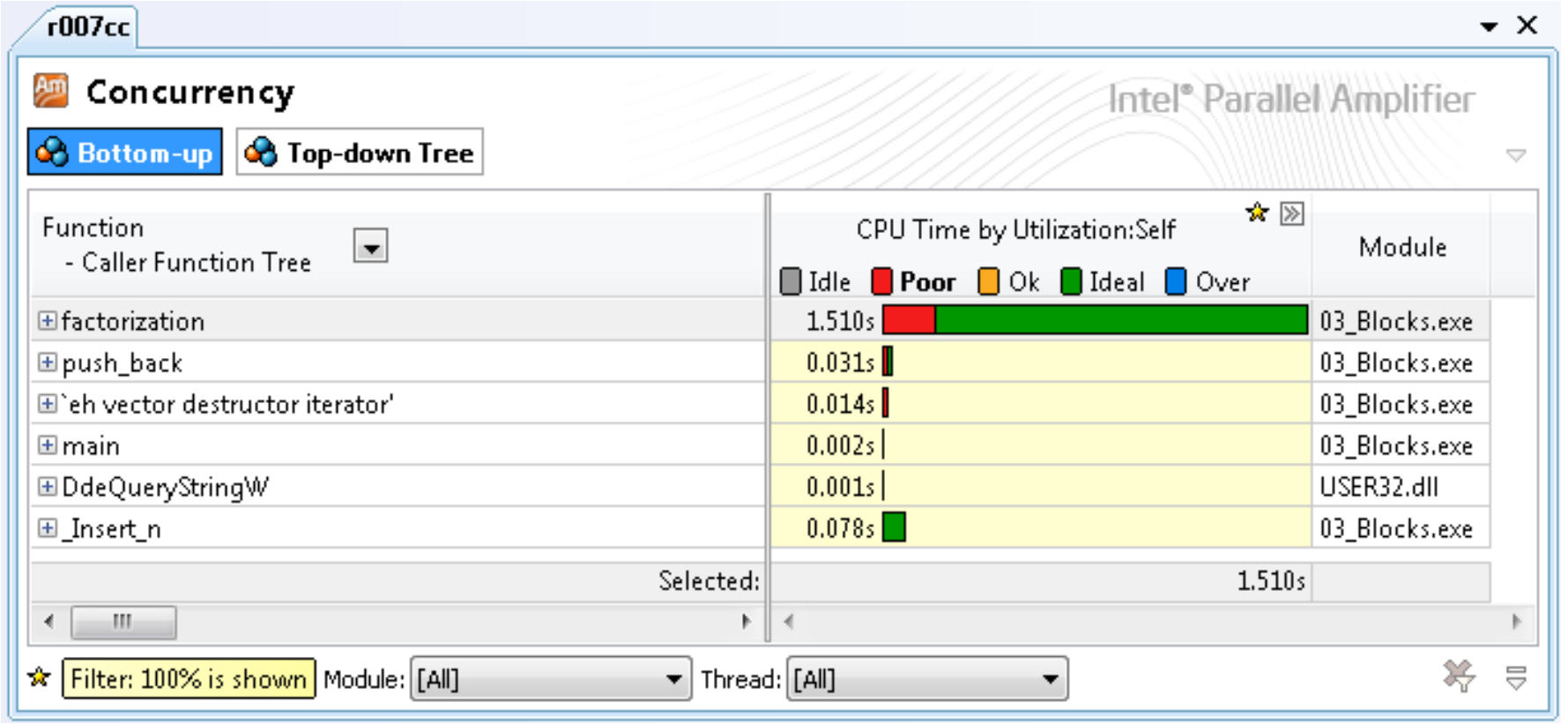
Подход #3: разделение множества чисел на небольшие группы

- Стратегия распределения нагрузки между потоками (откройте проект 03_Blocks):


```
// tid – идентификатор потока
1. numberOfGrains = NUM_NUMBERS / NUM_THREADS / GRAIN_SIZE;
2. for i = 0 to numberOfGrains
3.   begin = (NUM_THREADS * i + tid) * GRAIN_SIZE + 1;
4.   end   = (NUM_THREADS * i + tid + 1) * GRAIN_SIZE + 1;
```

Подход #3. Анализ эффективности

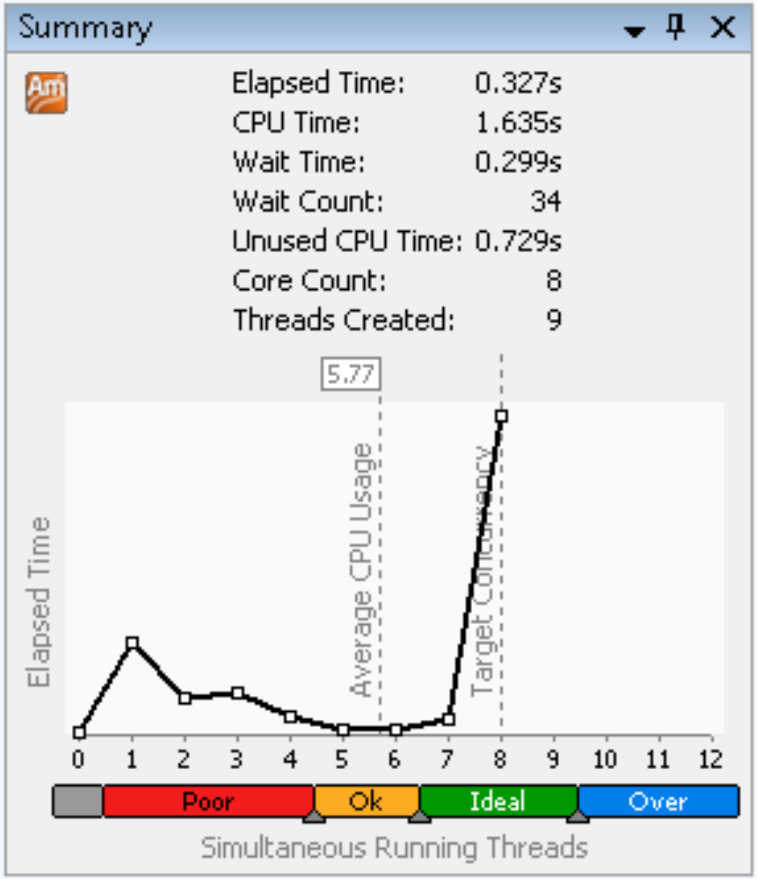
- Запустите Intel Parallel Amplifier в режиме Concurrency и оцените степень параллелизма.



Но! Как определить оптимальный размер блоков?

Подход #3. Оценка степени параллелизма

- Большую часть времени программа работает в 8 потоков



Задания для самостоятельной работы

- Подберите оптимальный размер блока при реализации подхода #3.
- Реализуйте алгоритм Полларда или Диксона для определения простых множителей числа.
- Проанализируйте эффективность описанных подходов к распределению нагрузки между потоками при условии, что для получения простых множителей числа используется алгоритм, реализованный в процессе выполнения предыдущего самостоятельного задания. Используйте инструменты пакета Intel Parallel Studio.
- Сравните эффективность и степень параллелизма реализаций, полученных при выполнении основного и дополнительных заданий лабораторной работы.

Авторский коллектив

- Кустикова Валентина Дмитриевна,
  ассистент кафедры
  Математического обеспечения ЭВМ факультета ВМК ННГУ
  valentina.kustikova@gmail.com
- Сиднев Алексей Александрович,
  ассистент кафедры
  Математического обеспечения ЭВМ факультета ВМК ННГУ
  alexey.sidnev@gmail.com
- Сысоев Александр Владимирович,
  ассистент кафедры
  Математического обеспечения ЭВМ факультета ВМК ННГУ
  sysoyev@vmk.unn.ru