Лекция 5. Анализ быстродействия программ

Инструменты разработки быстрых программ

11 декабря 2017 г.

Виды инструментов

Принцип работы

- Перехват событий (JVM, .NET, Python, Ruby).
- Сбор статистики о счётчике команд во время прерываний ОС.
- Инструментирование программы.
  - вручную;
  - автоматически на уровне исходного кода;
  - промежуточного представления;
  - с помощью компилятора;
  - двоичная трансляция;
  - перед выполнением программы.
- Инструментирование интерпретатора.
- Симулятор/гипервизор.

Виды анализа

Название	Пользователя	Аппаратный
Алгоритмические		
Basic hotspots	✓	
Advanced hotspots		✓
Concurrency	✓	
Locks and waits	✓	
Микроархитектурные		
General exploration		✓
Memory access		✓
...		

Таблица 1: основные виды анализа в VTune

Микроархитектура процессора

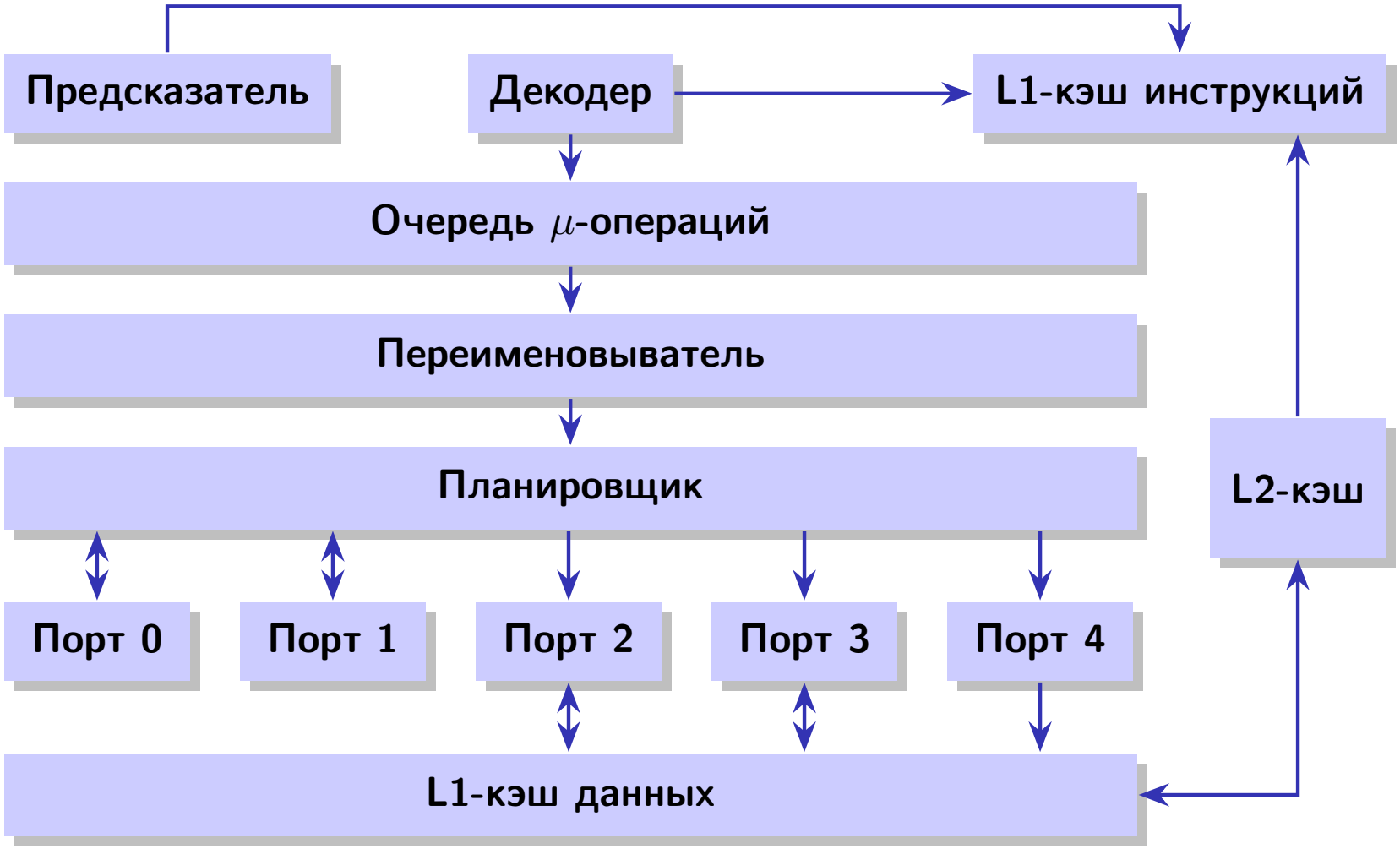

Рис. 1: микроархитектура процессорного ядра

Предсказатели ветвлений и целей ветвлений

Виды предсказателей ветвления

- Статический.
- Счётчик с насыщением:
  - 1 бит
  - 2 бит (93,5 % на SPEC'89);
- Двухуровневый адаптивный:
  - локальный (97,1 % на SPEC'89);
  - глобальный;
  - сплавленный;
  - согласованный;

Виды предсказателей ветвления

- Комбинированные;
- Специальные:
  - для циклов;
  - для косвенных переходов;
  - для возвратов из функций;
- Переключаемые;
- На основе нейронных сетей.

Переименовыватель

Функции переименовывателя

- Выделение ресурсов микрооперациям (порт, буферы чтения, записи, ...) (allocating).
- Переименование ресурсов (renaming).
- Удаление ложных зависимостей.
- Исполнение и удаление некоторых операций:
  - обнуление (`XOR reg1, reg1, ...`);
  - установка 1 (`CMPEQ reg1, reg1, reg2`);
  - перемещение регистров;
  - `NOP`.

Простои конвейера

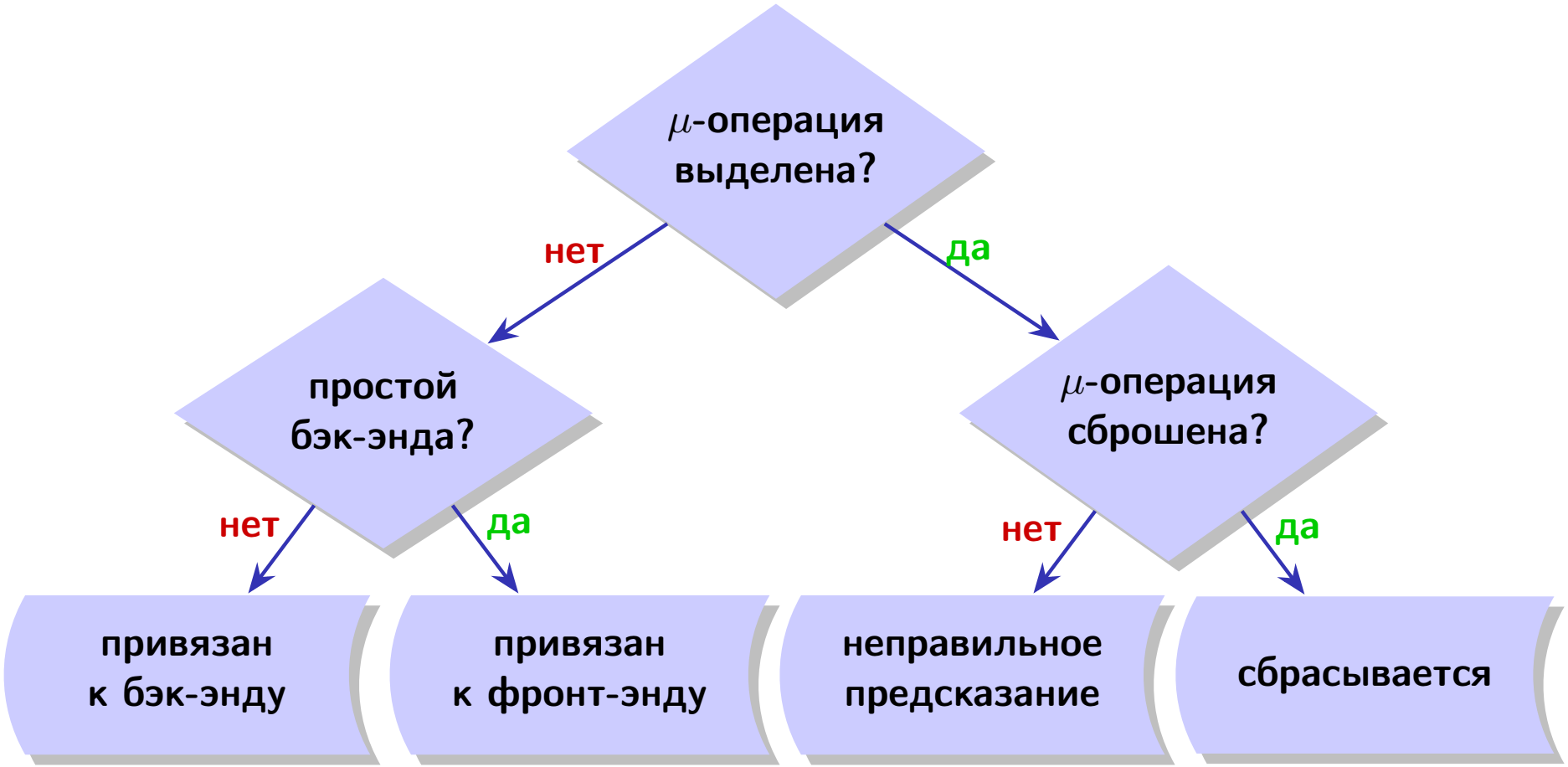

Рис. 2: диаграмма классификации простоев конвейера

Метрики загруженности

$$
\begin{aligned}
N &= 4 \times \texttt{CPU\_CLK\_UNHALTED.THREAD} \\
FE\_Bound &= \texttt{IDQ\_UOPS\_NOT\_DELIVERED.CORE}/N \\
Bad\_Speculation &= \big(\texttt{UOPS\_ISSUED.ANY} - \texttt{UOPS\_RETIED.RETIRE\_SLOTS} + \\
&\quad + 4 \times \texttt{INT\_MISC.RECOVERY\_CYCLES}\big)/N \\
Retiring &= \texttt{UOPS\_RETIED.RETIRE\_SLOTS}/N \\
BE\_Bound &= 1 - FE\_Bound - Bad\_Speculation - Retiring
\end{aligned}
$$

Секция конвейера

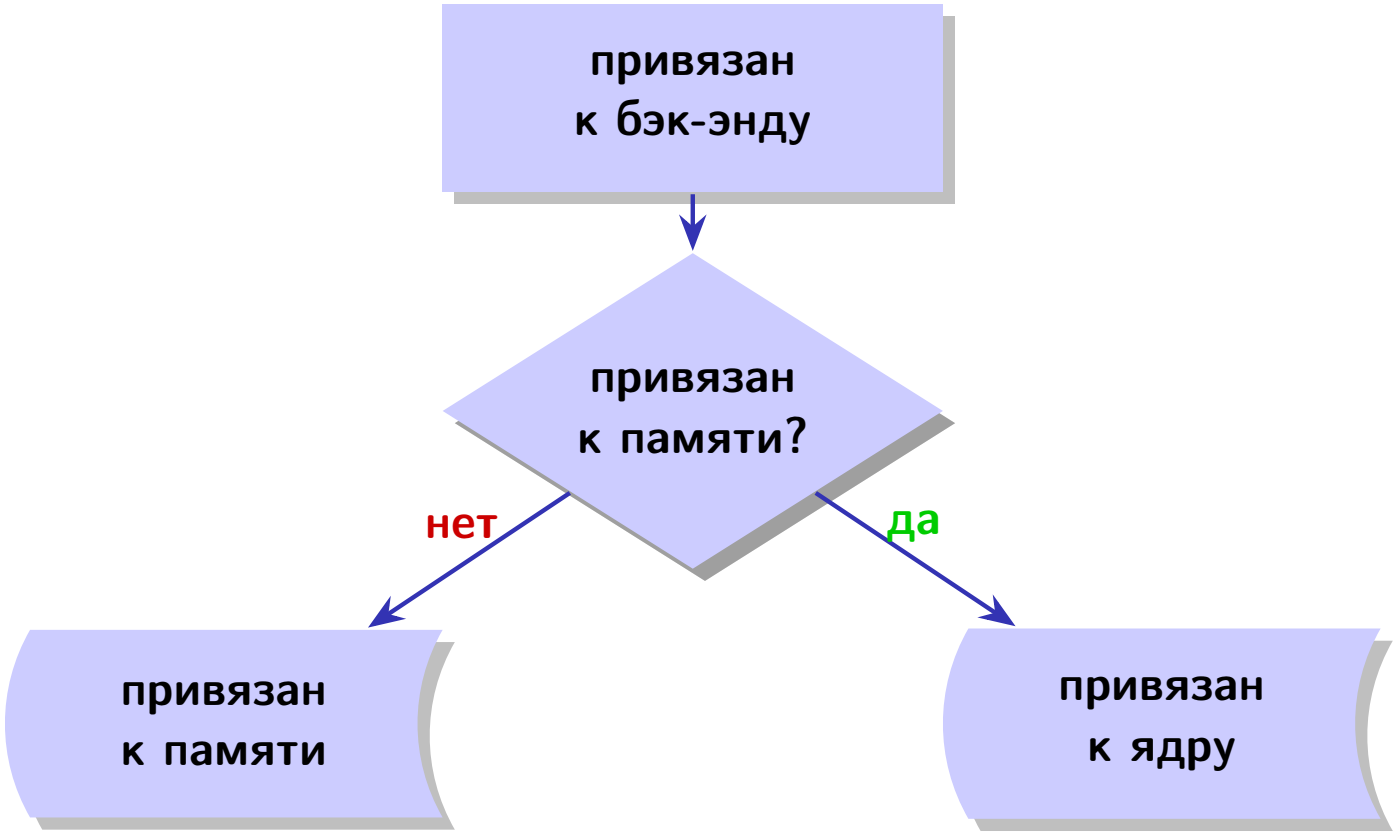

Рис. 3: диаграмма классификации простоев конвейера

Состояния слотов конвейера

Категория приложений	Сбрасываемые			К бэкенду			К фронтэнду			Непредсказанные		
Клиентские Настольные	20	–	50	20	–	40	5	–	10	5	–	10
Серверные СУБД Распределённые	10	–	30	20	–	60	10	–	25	5	–	10
Быстрые вычисления	30	–	70	20	–	40	5	–	10	1	–	5

Таблица 2: ожидаемые доли слотов в горячих участках

Оптимизации ветвлений

Оптимизации ветвлений

- Избегать ветвлений:
  - упорядочивать код, делая базовые блоки непрерывными;
  - выполнять развёртку циклов, если при этом размер кода сильно не увеличивается;
  - использовать условные инструкции (`CMOV`, `SETCC`, ...)
- Помещать код и данные в разных страницах.
- Упорядочить код в соответствии со статическим алгоритмом предсказания ветвлений.
- Поддерживать соответствие `CALL`/`RET`.
- Выполнять подстановку, где имеет смысл.

Пример создания потоков

Пример

```
for (i = 0; i < g_cuNumThreads; ++ i)
{
  // ...
  //
  aParams[i].m_uStart = uPrev;
  aParams[i].m_uEnd = uIndex;
  aParams[i].m_pdData = pdData;
  uPrev = uIndex;
  ahThreads[i] = CreateThread(
    NULL, 0, &ThreadProc, &aParams[i], 0, NULL);
  WaitForSingleObject(ahThreads[i], INFINITE);
}
```

Пример создания потоков

Пример

```
for (i = 0; i < g_cuNumThreads; ++ i)
{
  // ...
  //
  aParams[i].m_uStart = uPrev;
  aParams[i].m_uEnd = uIndex;
  aParams[i].m_pdData = pdData;
  uPrev = uIndex;
  ahThreads[i] = CreateThread(
    NULL, 0, &ThreadProc, &aParams[i], 0, NULL);
}
//
WaitForMultipleObjects(g_cuNumThreads, ahThreads, TRUE, INFINITE);
```

Анализ многопоточных программ

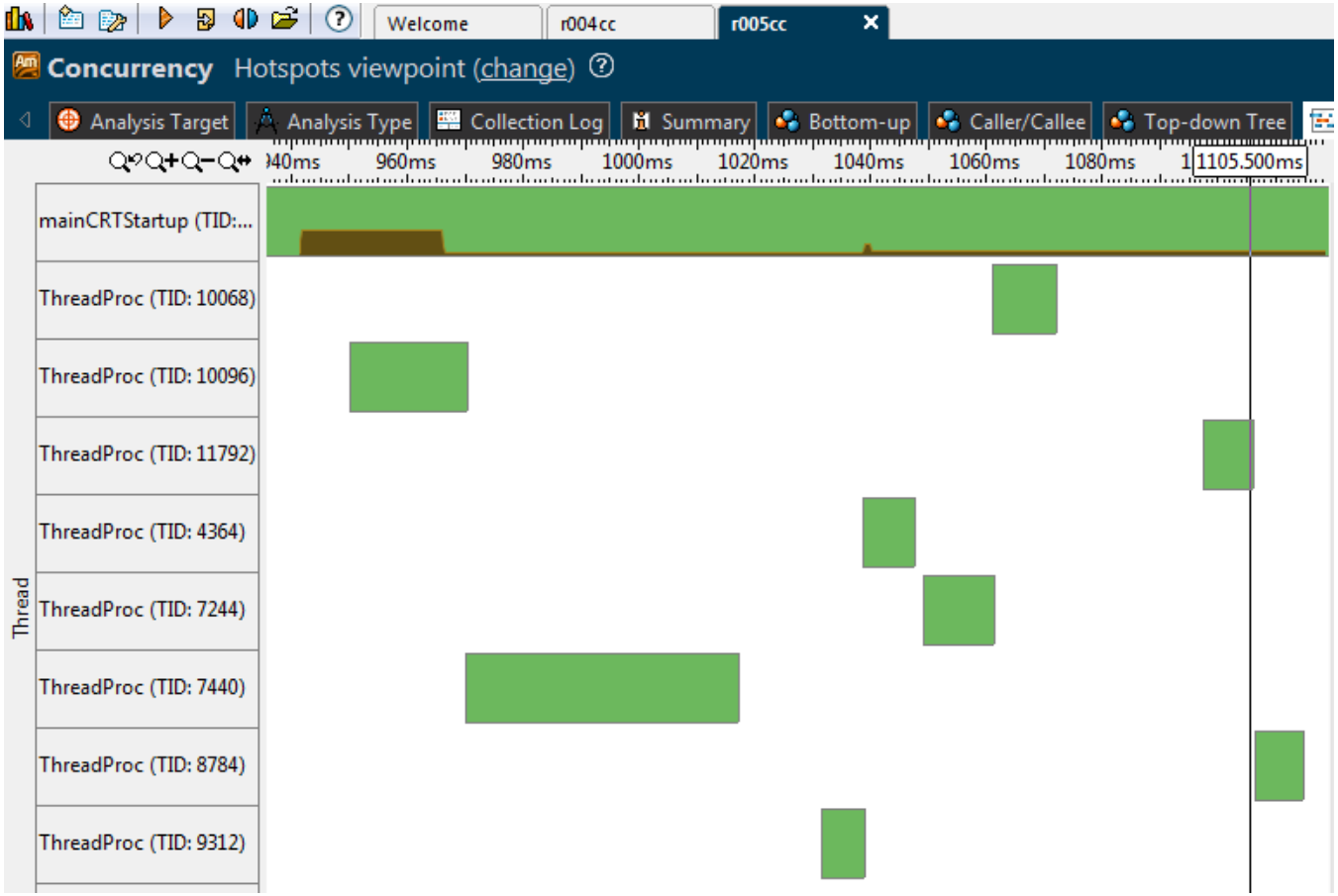

Рис. 4: часть окна программы Intel VTune Amplifier XE 2016

Анализ многопоточных программ (окончание)

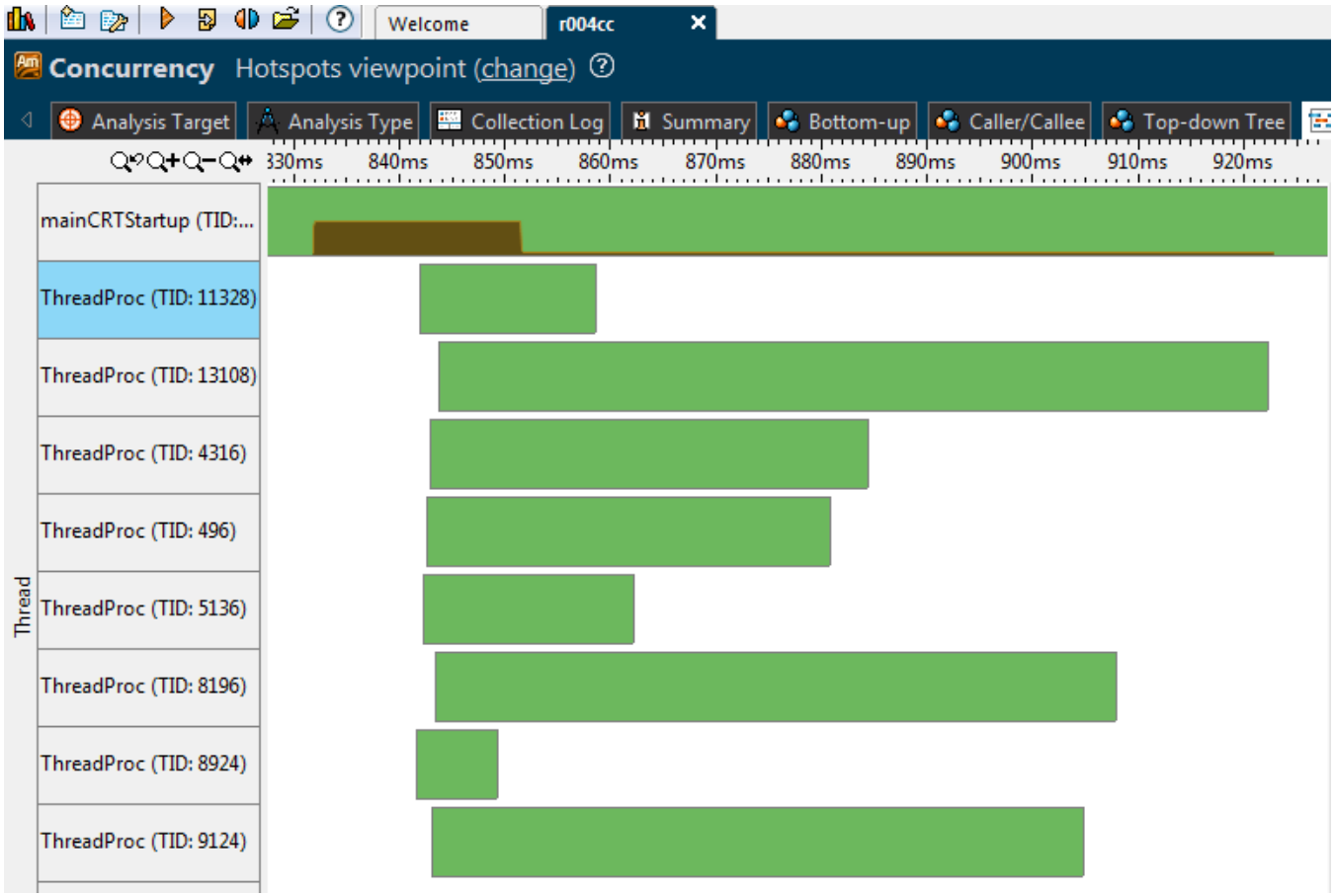

Рис. 5: часть окна программы Intel VTune Amplifier XE 2016

Параллельные программы

Проблемы быстродействия параллельных программ

- Несбалансированность нагрузки
- Избыточное использование глобальных данных с синхронизацией
- Расходы на создание и работу объектов.
- Недостаточное количество параллельной работы.

Улучшение производительности параллельных программ

Уменьшение расходов на синхронизации

- Использовать локальные данные (локальные переменные для частичных результатов, TLS)
- Уменьшать размеры критических областей до минимальных, где происходит доступ к общим переменным
- Использовать атомарные операции вместо блокировок везде, где возможно
- Использовать критические секции вместо мьютексов везде, где возможно

Пример блокировки

Пример

```
DWORD WINAPI PiThreadFunc(LPVOID pvArg)
{
  int myThreadNum = *((int *) pvArg);
  int start = myThreadNum + 1;
  for (int i = start; i < MaxIterations; i += globalNumThreads)
  {
    double dx = (i - 0.5) * globalDInterval;
    EnterCriticalSection(&globalCS);
      globalDSum = globalDSum + f(dx);
    LeaveCriticalSection(&globalCS);
  }
  return myThreadNum;
}
```

Анализ многопоточных программ

Рис. 6: часть окна программы Intel VTune Amplifier XE 2016

Анализ многопоточных программ (продолжение)

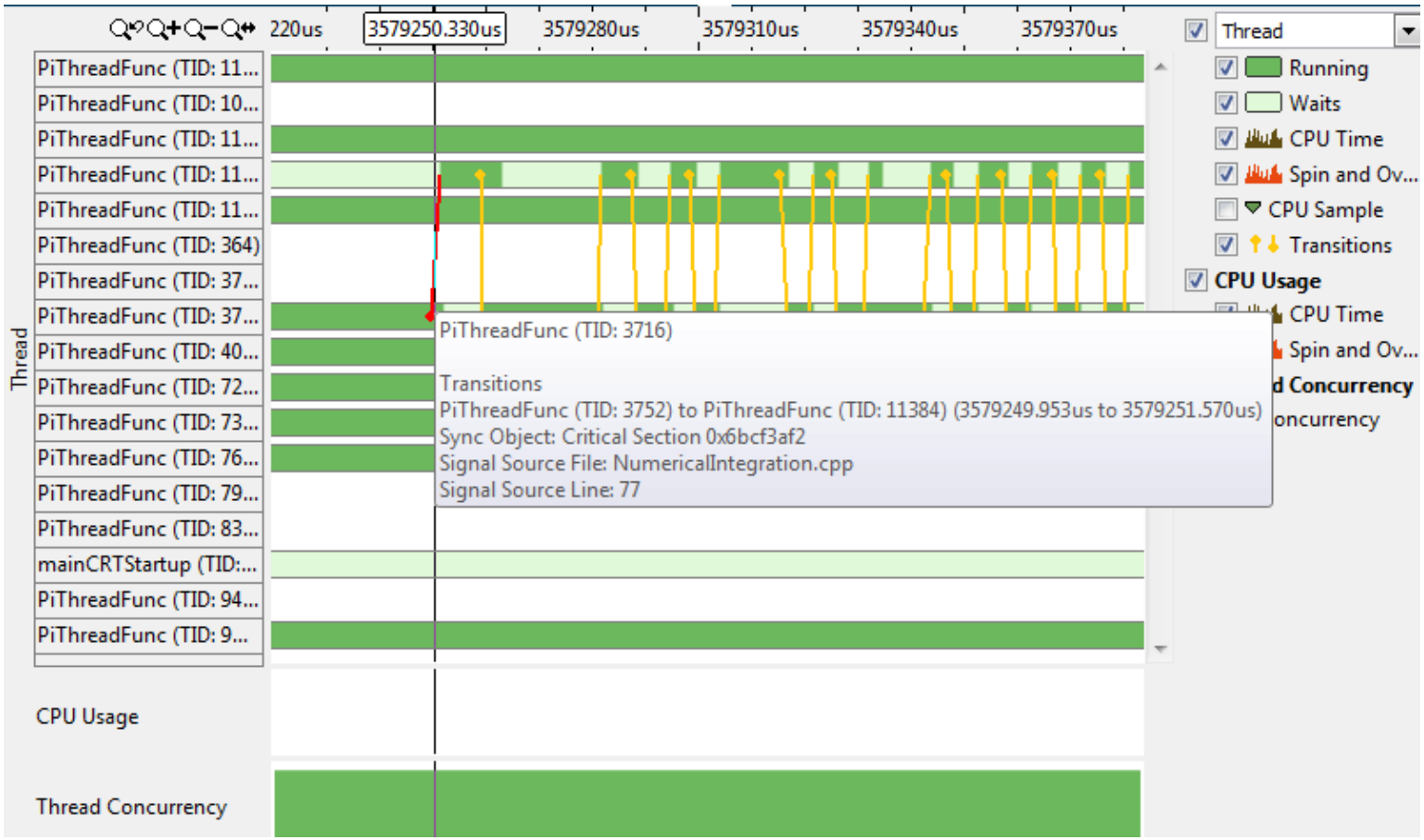

Рис. 7: часть окна программы Intel VTune Amplifier XE 2016

Анализ многопоточных программ (продолжение)

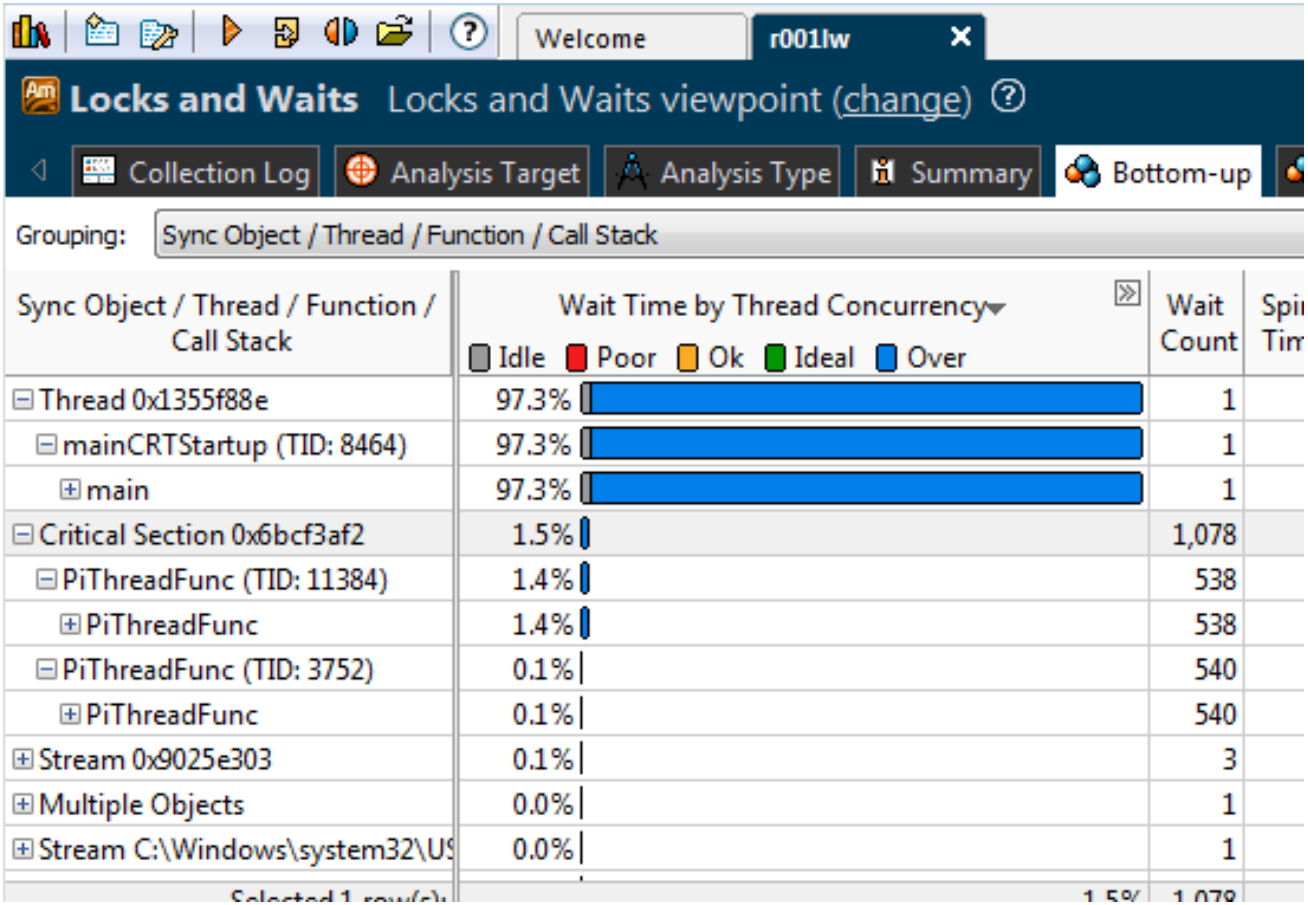

Рис. 8: часть окна программы Intel VTune Amplifier XE 2016

Анализ многопоточных программ (окончание)

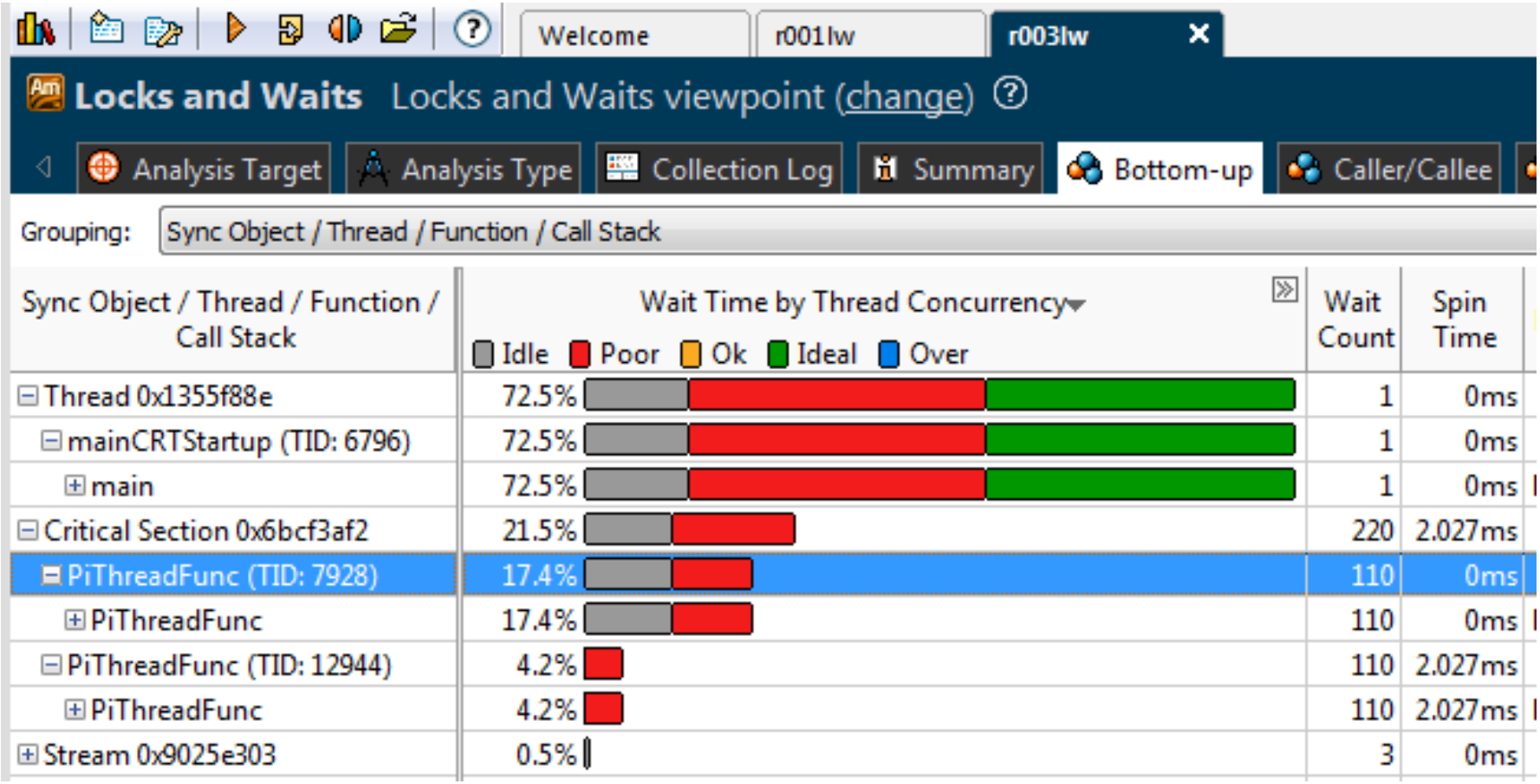

Рис. 9: часть окна программы Intel VTune Amplifier XE 2016