Введение в Concurrency Visualizer

Concurrency Visualizer - новое средство профилирования, которое было добавлено в *Visual Studio* 2010, позволяющее значительно облегчить *анализ* производительности параллельных программ, и позволяет разработчикам анализировать последовательные приложения на предмет возможности их распараллеливания.

Concurrency Visualizer включает несколько инструментов для визуализации и создания отчетов. Поддерживается три основных представления (режима просмотра) для просмотра отчетов:

- CPU Utilization (использование процессора);
- Threads (потоки);
- Cores (ядра процессора).

Представление использования центрального процессора (CPU Utilization)

Представление CPU Utilization (Рис.1) показывает среднюю интенсивность использования ядра во времени, анализируемым процессом, системным процессом и другими процессами, запущенными в системе. Данное *представление* не показывает, какое из ядер активно в заданный момент времени. Например, если два ядра были загружены на 50% каждое в течение заданного периода времени, *график* покажет, что было утилизировано одно логическое *ядро*.

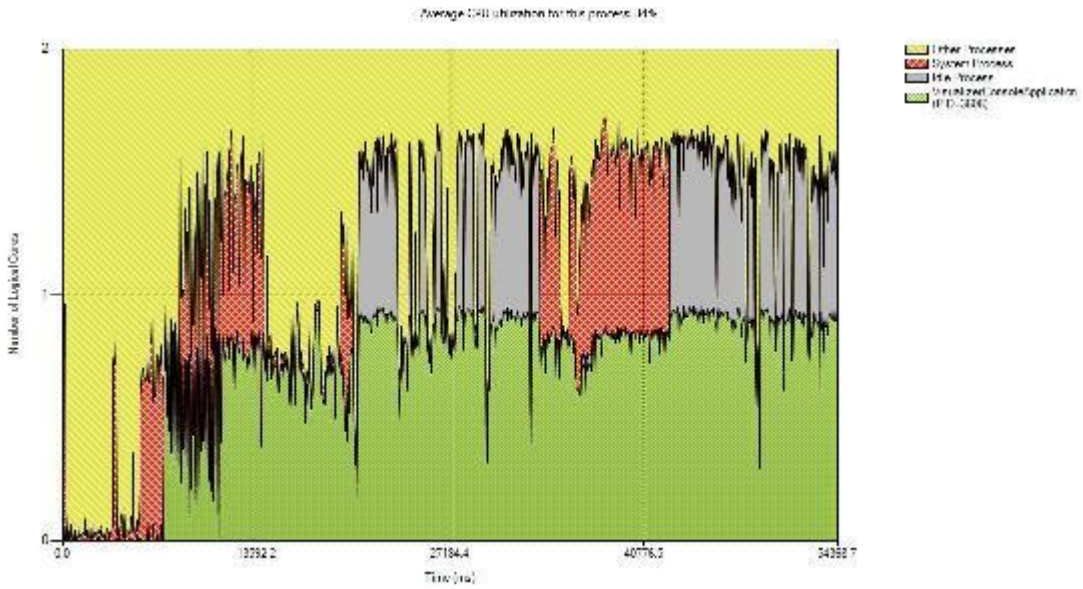

Рисунок 1 – Представление CPU Utilization

По оси X показывается время, истекшее с начала трассировки и до конца работы приложения. *По* оси Y показывается число логических ядер процессора в системе.

Как видно в примере (Рис.1), на графике отображаются четыре области, обозначенные в легенде. Зеленая область отражает среднее количество логических ядер, которое анализируемое *приложение* использует во время *цикла* профилирования. Остальные логические ядра либо простаивают (закрашиваются серым цветом), либо задействованы процессом *System* (красный цвет), либо используются другими процессами в системе (желтый).

Представление CPU Utilization служит трем основным целям:

1. Если необходимо распараллеливание приложения, то на графике нужно искать области выполнения, которые обнаруживают значительные объемы последовательной работы с процессором (они показываются как длинные зеленые области на уровне одного ядра по оси Y), или области, где процессор используется довольно слабо (области зеленого цвета нет или высота этой области существенно меньше единицы в среднем).
2. Если необходимо оптимизировать распараллеленное приложение, это представление позволяет увидеть реальную степень его параллелизма при выполнении. Простое изучение этого графика обычно делает явными многие распространенные ошибки, связанные с параллельной работой. Например: неправильное распределение нагрузки в виде ступенчатых областей на графике или конкуренцию за синхронизирующие объекты, которая проявляется как последовательное выполнение вместо параллельного.
3. Поскольку приложение «живет» в системе, которая, скорее всего, выполняет массу других программ, конкурирующих за ее ресурсы, важно знать, влияют ли другие приложения на производительность нашего приложения, поэтому перед профилированием стоит отключать посторонние приложения и сервисы, чтобы повысить точность данных.

Представление потоков (Threads)

Представление Threads (Рис.2) содержит множество средств детального анализа и отчетов в *Concurrency* Visualizer. Именно с помощью этого представления можно получать дополнительную информацию, проясняющую поведение работы приложения, которую можно получить в представлениях *CPU Utilization* или Cores. Кроме того, здесь есть данные, позволяющие *по* возможности связывать *поведение приложения* с его исходным кодом. В этом представлении есть три основных компонента:

- временная шкала (timeline);
- активная легенда (active legend);
- элемент управления "вкладка" для получения отчета/детальных сведений.

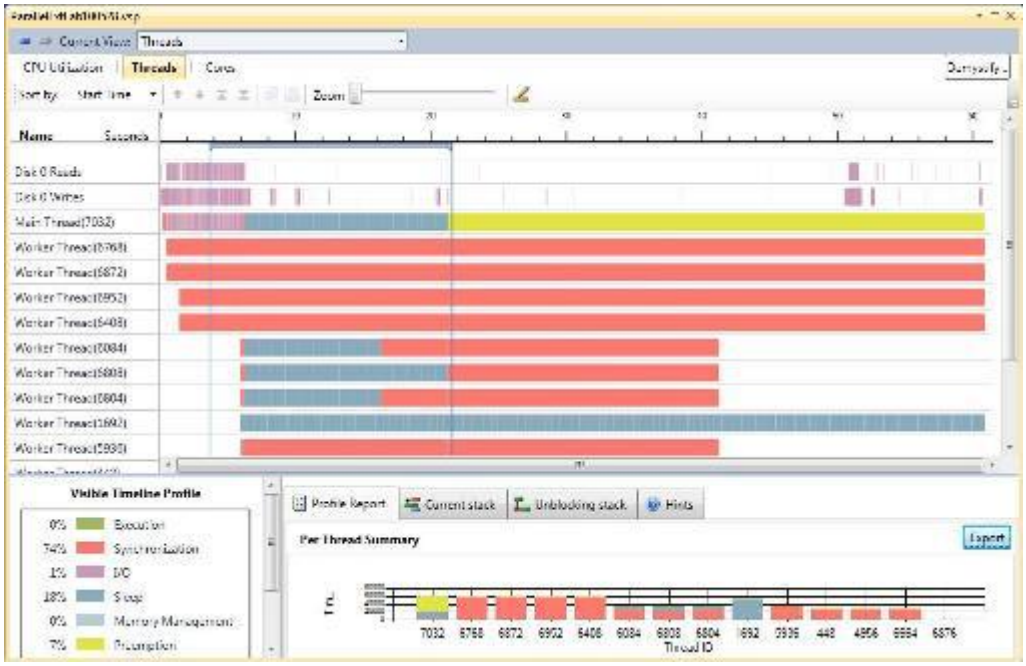

Рисунок 2 – Представление потоков (Threads)

Как и в *CPU Utilization*, в представлении потоки (Threads) *по* оси X показывается время. (При переключении между представлениями в *Concurrency* Visualizer *диапазон* времени, показываемый *по* оси X, сохраняется.) Однако в представлении Threads *по* оси Y размещаются два типа горизонтальных каналов.

Верхние каналы обычно отводятся под физические диски в системе, если они активны в профиле приложения. На каждый *диск* создается два канала: один для операций чтения, а другой для *операции* записи. Эти каналы отображают обращения к диску из потоков вашего приложения или процесса *System*. Каждая операция чтения или записи рисуется как *прямоугольник*. *Длина* прямоугольника обозначает задержку доступа, связанную, в том числе с помещением в *очередь*; из-за этого прямоугольники могут перекрываться. Остальные каналы на временном графике перечисляют все потоки, существовавшие в приложении за период профилирования. Для каждого потока, если инструмент обнаруживает любую *активность* в ходе профилирования, показывается его состояние вплоть до завершения.

Представление Cores

И последнее *представление* доступное программисту – это *представление* «Cores». *Представление* «Cores» состоит из двух частей, *графика* и легенды, которые показывают, как выполнение потоков сопоставлено с логическими ядрами процессора. При написании серверных приложений это *представление* может помочь оптимизировать *производительность* кэша, используя сходство потоков или управление пулом потоков. *Представление* «Cores» также может помочь визуализировать случаи, когда использование сходства потоков могло фактически ухудшить проблему переходов между ядрами.

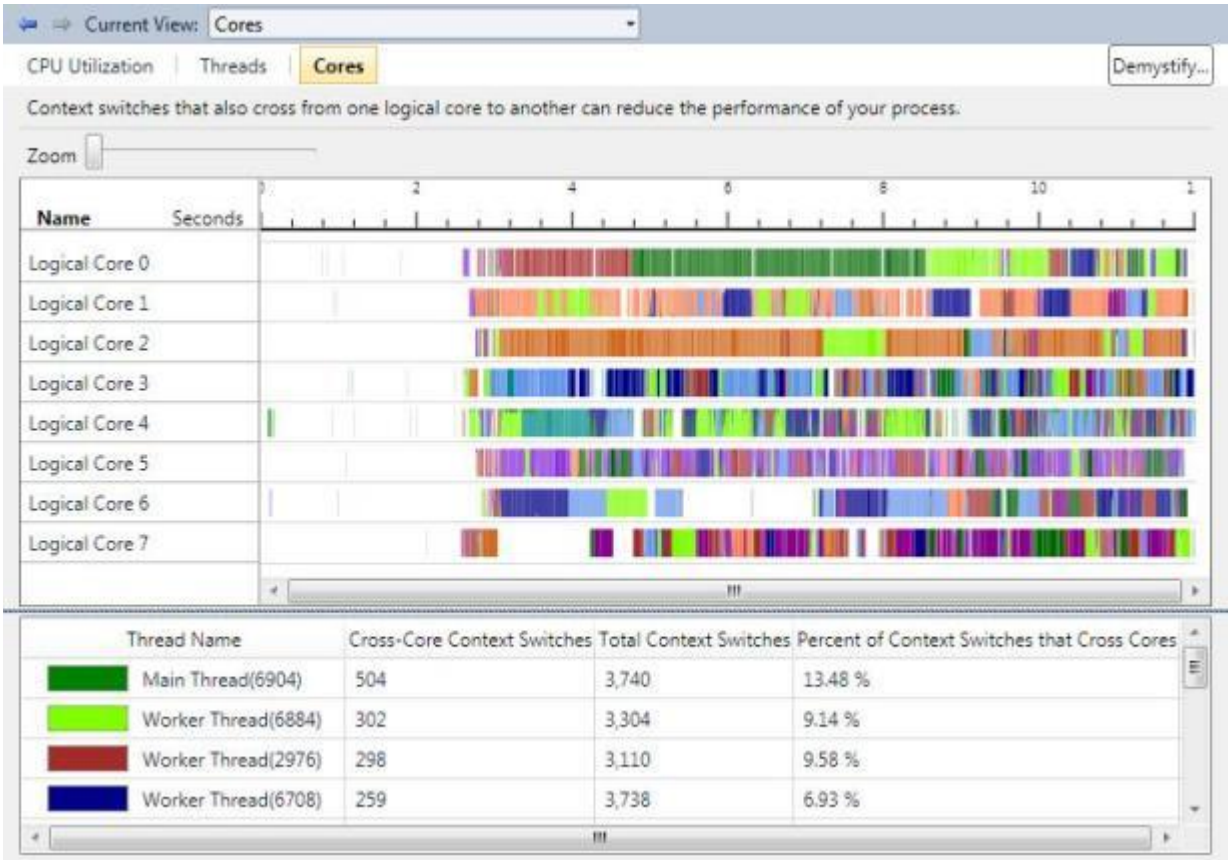

Рисунок 3 – Представление Cores

На Рис.3 показаны логические ядра *по* оси Y и время *по* оси X. Каждый *поток* в графике имеет уникальный цвет, что позволяет видеть перемещения каждого потока между ядрами с течением времени. Можно, также, увеличить или уменьшить масштаб для интересующего участка временной шкалы: для этого нужно выделить конкретную зону и, зажав клавишу CTRL, прокрутить колесико мыши.

В легенде представления Threads приведены записи для каждого цвета на верхнем графике. В них указаны цвет и имя потока, количество межъядерных переключений контекста, общее количество переключений контекста и процентное соотношение переключений контекста между ядрами. Легенда сортируется *по* убыванию количества межъядерных переключений контекста.

Статистика, получаемая с помощью представления Cores, помогает определять потоки со слишком частым переключением контекста и миграцией между ядрами.

Секция отчетов

Отчеты профилей позволяют легко находить наиболее значимые факторы, влияющие на *производительность* приложения. В представлении Threads предлагаются четыре типа отчетов:

- профили выборки выполнения (execution sampling profiles);
- профили блокировки (blocking profiles);
- отчеты по файловым операциям;
- сводки по каждому потоку.

Просмотр отчетов доступен через легенду, изображенную на Рис.4.

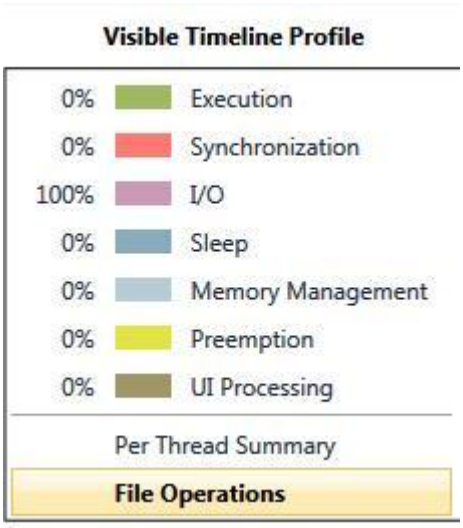

Рисунок 4 – Профили отчетов представления Threads

В Табл.1 представлены все виды отчетов, доступные для просмотра.

Таблица 1. Виды отчетов

Тип отчета	**Описание**
Execution (Выполнение)	В отчете профиля выполнения отображается подробная таблица с указанием процента времени, затраченного каждым

	потоком в различных состояниях, например в процессе выполнения, ввода-вывода и управления памятью.
Synchronization (Синхронизация)	В отчете о синхронизации отображаются вызовы, отвечающие за блокировки синхронизации, с указанием совокупного времени блокировки каждого стека вызова. Эти сведения можно использовать для определения интересующих областей и их анализа.
I/O (Ввод-вывод)	В отчете о вводе-выводе отображаются вызовы, отвечающие за блокировки ввода-вывода, с указанием совокупного времени блокировки каждого стека вызова. Эти сведения можно использовать для определения интересующих областей и их анализа.
Sleep	В отчете о спящем режиме отображаются вызовы, отвечающие за блокировки спящего режима, с указанием совокупного времени блокировки каждого стека вызова. Эти сведения можно использовать для определения интересующих областей и их анализа.
Memory management (Разбиение по страницам)	В отчете о разбиении по страницам отображаются вызовы, отвечающие за блокировки вытеснения, с указанием совокупного времени блокировки каждого стека вызова. Эти сведения можно использовать для определения интересующих областей и их анализа. Этот отчет о блокировках имеет меньшее практическое значение, чем другие, поскольку вытеснение чаще задается для процесса операционной системой, чем вызывается кодом пользователя. Он показывает, какие виды вытеснения созданы, где они произошли, и как долго процесс находился в определенном состоянии вытеснения.
Preemption (Вне очереди)	В отчете о разбиении по страницам отображаются вызовы, отвечающие за блокировки вытеснения, с указанием совокупного времени блокировки каждого стека вызова. Эти сведения можно использовать для определения интересующих областей и их анализа. Этот отчет о блокировках имеет меньшее практическое значение, чем другие, поскольку вытеснение чаще задается для процесса операционной системой, чем вызывается кодом пользователя. Он показывает, какие виды вытеснения созданы, где они произошли, и как долго процесс находился в определенном состоянии вытеснения.
UI-processing (Обработка пользовательского интерфейса)	В отчете об обработке пользовательского интерфейса отображаются вызовы, отвечающие за блокировки обработки пользовательского интерфейса, с указанием совокупного времени блокировки каждого стека вызова. Эти сведения можно использовать для определения интересующих областей и их анализа.
Per Thread Summary (Сводный отчет по каждому потоку)	Данный отчет строит гистограмму, на которой показано количество времени, затраченное каждым не скрытым потоком в каждой категории действия на отображаемом в данный момент промежутке времени. Категория " Выполнение" означает, что поток выполняется; все остальные категории означает, что поток находится в состоянии ожидания какого-либо события.
File Operations	Отчет File Operations включает сводку по всем операциям файлового чтения и записи, видимым в текущем временном

	диапазоне. Для каждого файла Concurrency Visualizer перечисляет поток приложения, который обращается к нему, число операций чтения и записи, общее количество считанных или записанных байтов, а также общую задержку чтения или записи. Кроме отображения файловых операций, прямо связанных с приложением, Concurrency Visualizer показывает и те, которые выполняются процессом System. Это сделано из-за того, что они могут выполняться от имени вашего приложения. Экспорт отчета позволяет проводить сравнение между профилями.

Отчеты *Concurrency* Visualizer - удобные средство, позволяющее расставить приоритеты в оптимизации производительности и выявить части приложения, ответственные за наиболее значимые задержки. Отчет *по* вытеснению носит чисто информационный характер и обычно не предоставляет никаких данных для дальнейших действий из-за самой природы этой категории.

Отчеты позволяют переключаться в соответствующие места исходного кода. Это можно сделать, щелкнув правой кнопкой мыши нужный *фрейм* стека. Появляющееся после этого контекстного *меню* позволяет перейти либо к определению функции (*команда* View Source) или к участку кода в приложении, где эта *функция* была вызвана (*команда* View *Call* Sites). Если вызовы исходили от нескольких блоков кода, можно будет выбрать несколько команд. Это обеспечивает бесшовную интеграцию диагностических данных в процесс разработки, помогающую настраивать *поведение приложения*. Отчеты также можно экспортировать для сравнений между профилями.