Введение в параллельное программирование и высокопроизводительные вычисления

С.А.Немнюгин

OpenMP – дополнительные материалы

Анализатор производительности Intel ® VTune Analyzer

MPI

(конспект)

Санкт-Петербург
2008

OpenMP. Накладные расходы на создание различных конструкций

Данные приведены для процессора Intel ® Xeon 3Ггц

Конструкция	Длительность инициализации, мкс
parallel	1.5
barrier	1.0
schedule(static)	1.l0
schedule(guided)	6.0
schedule(dynamic)	50.0
schedule(dynamic, 16)	5.0
ordered	0.5
single	1.0
reduction	2.5

Intel ® VtuneTM Analyzer

Intel ® Vtune™ Analyzer (IVA) – программный инструмент, позволяющий выявить и локализовать проблемы производительности ПО.
Возможности:

- сбор различных показателей производительности;
- отображение данных в различных режимах (system-wide, исходный код и процессорные инструкции);
- выявление потенциальных проблем производительности и рекомендации по их разрешению.

IVA поддерживает:

- операционные системы: *Microsoft Windows*, *Linux*;
- приложения: *C/C++/Fortran*, *Java*, *.NET*;
- интерфейсы графический (в Linux на основе Eclipse) и командной строки;
- локальный и удаленный сбор данных (агент и коллекторы данных на удалённой машине, управление процессом, анализ и отображение результатов на host-машине).

Администрирование

Пользователь должен входить в дополнительную группу **vtune**.

Сбор статистики (sampling activity):

- по времени (TBS – Time Based Statistic);
- по заданному количеству событий (например, по заданному количеству ошибочно предсказанных ветвлений, EBS – Event Based Statistic);
- привязка к исходному коду или дизассемблирование.

Построение графа вызовов (callgraph activity)

- граф вызовов с подробной информацией о временных затратах;
- критический путь исполнения.

Tuning Assistant

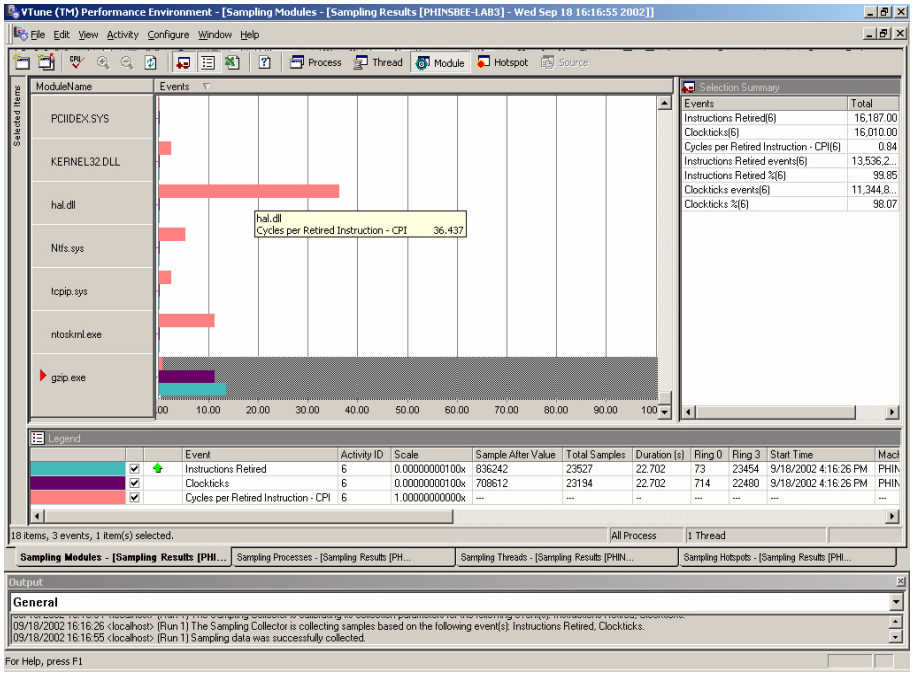
- комментарии по проблемам;
- подсказки по модификации кода.

Обработка данных

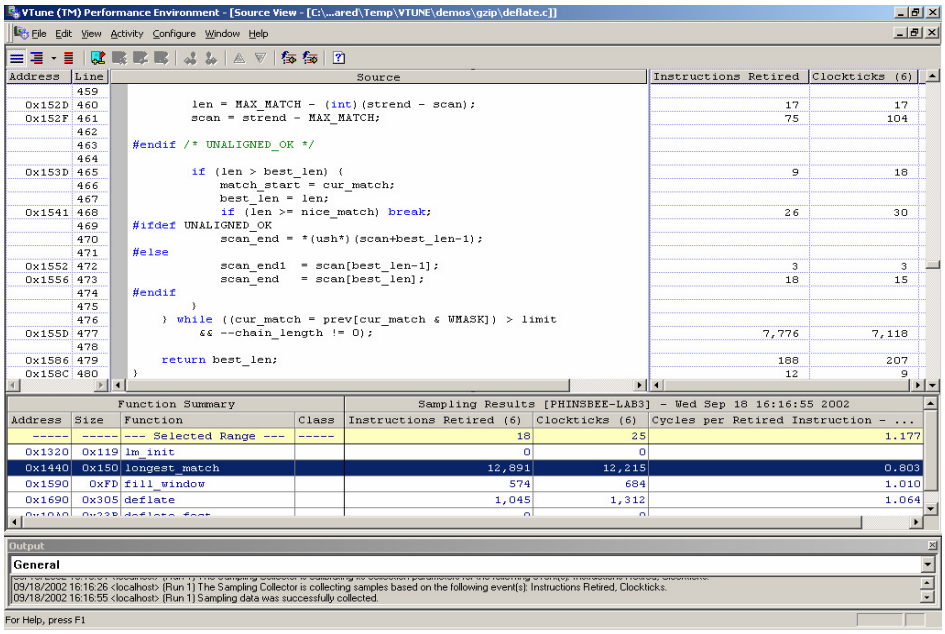
- «мастера» для конфигурирования коллекторов;
- упаковка и перенос проектов на другую машину.

Окна отображения результатов

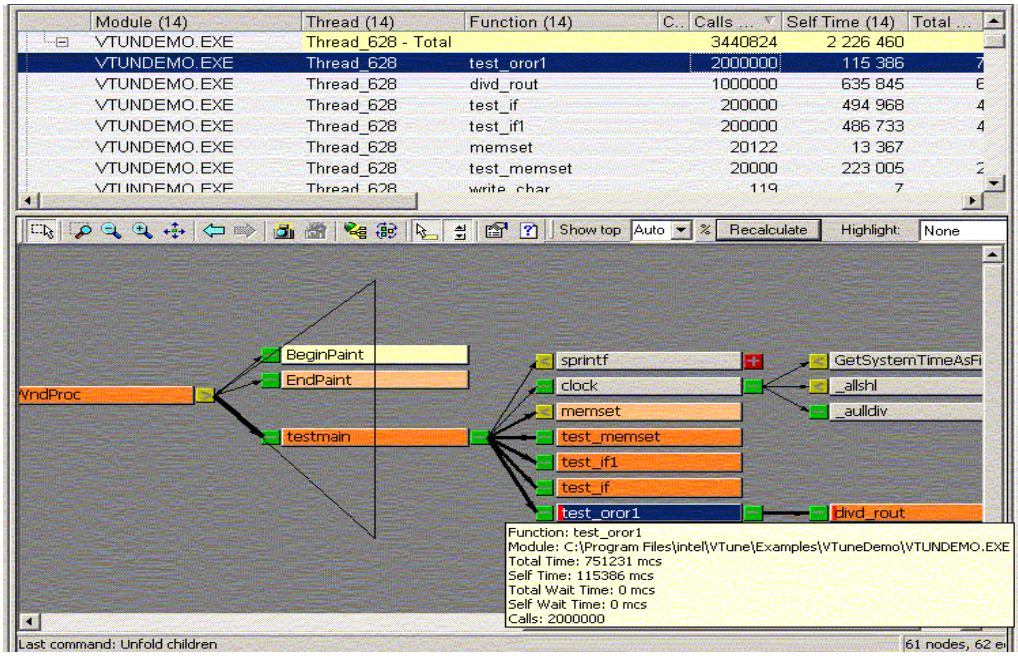
Module view



Sampling и source view



Callgraph view



При оптимизации и поиске «горячих пятен» в первую очередь следует обратить внимание на следующие факторы:

- промахи кэша (cache misses);
- ошибки предсказания ветвлений (branch mispredictions);
- промахи при работе с буфером трассировки (trace buffer misses);
- вычисления с плавающей точкой;
- конфликты по доступу к ресурсам (resource related stalls);
- выравнивание на границу 64k aliasing (P4).

В справочнике Help for VTune performance analyzer имеется информация о регистрируемых событиях и о том, как оптимизировать приложение с точки зрения этих событий.

Интерфейс командной строки

Сначала необходимо подготовить программу для исследования. Для этого необходимо выполнить трансляцию с построением таблицы символов. Оптимизацию при этом можно отключить:

```
icc -g -O0 -o exa example_test.c
```

При работе с IVA сначала следует создать конфигурационный файл:

```
vtl query -c тип_активности -dof event.cfg
```

Пример:

```
vtl query -c sampling -dof event.cfg
```

Текстовый форматный файл **event.cfg** имеет примерно такой вид:

```
-avail-cpu-mask all
-si 1
-sb 2000
-sd 0
-term yes
-ttc yes
-sm ebs
-sterm yes
-msc 0
-ec
en='CPU_CLK_UNHALTED.CORE':sa=1861000,en='INST_RETIRED.ANY':sa=1
861000
en='L1I_MISSES':sa=1
```

Последняя строка этого файла содержит список событий, по которым будет производиться сбор статистики. Добавить новое событие можно, добавив запись в конец последней строки (`-ec`) конфигурационного файла через запятую:

```
en='2nd Level Cache Read Misses ',sa=1
```

Здесь sa – “sample after”

Список всех доступных событий можно получить командой:

```
vtl query -c sampling > events.lst
```

Затем следует создать новую «активность», в рамках которой будет запущено указанное приложение и проведен сбор статистики на основе подсчета тактов или построен граф вызовов:

```
vtl activity имя -c тип_активности -app приложение “опции
приложения”
```

Примеры:

```
vtl activity exa -c sampling -app exa

vtl activity exa -c callgraph -app exa

vtl activity exa -d 20 -c sampling -app exa
```

В последнем примере сбор статистики производится в течение заданного времени (20 сек)

Следующий шаг - запуск активности:

```
vtl run
```

Запустить активность можно и, добавив `run` к команде формирования активности

Просмотр результатов выполняется командой:

```
vtl view –modules
```

На экран выводится таблица, в которой содержится информация по всей системе. Следует найти интересующий Вас модуль.

Просмотр результатов на уровне функций выбранного приложения («модуля») производится командой вида:

```
vtl view –hf –mn exa
```

На экран выводится таблица, в которой содержится информация о приложении. В данном примере это приложение `exa`.
В таблице обратите внимание на колонку с номером 12 (`ADDRESS`). Здесь находится адрес функции, используя который можно привязать результаты сбора статистики к исходному тексту (если он есть)

Просмотр результатов на уровне функций выбранного приложения («модуля») возможен командой:

```
vtl view –code –rva 0x968 –hf –mn exa
```

На экран выводится таблица, в которой содержится информация в привязке к исходному тексту.

Помощник настройки (Tuning Assistant) может быть включен с помощью опции `–ta`.

Другие инструменты анализа
Sun Studio

Sun Studio - интегрированная среда разработки программ для языков программирования C, C++ и Фортран, разработанная компанией Sun Microsystems Inc. Распространяется бесплатно. В Sun Studio включены средства сборки, отладки, профилирования и анализа многопоточных приложений.
Компоненты Sun Studio:

- компиляторы C, C++ и Fortran;
- графическая среда разработки, базирующаяся на NetBeans;
- отладчик dbx, интегрированный со средой разработки;
- статические верификаторы кода lint и lock_lint;
- инструмент для распределенной или параллельной сборки приложений dmake;
- профилировщик Performance Analyzer;
- инструмент для поиска ситуаций "data race" - Thread Analyzer;
- инструмент для поиска утечек памяти и ошибок, связанных с неправильным доступом памяти - RTC (Run-Time Checking); является частью dbx.

AMD

Список ПО для разработчиков, находящегося в стадии создания для 64-бит Linux:

- анализатор кода AMD CodeAnalyst 2.0;
- библиотеки AMD Core Math Libraries (ACML) – от NAG;
- оптимизированные компиляторы C, C++ Fortran 77/90 – от Portland Group;
- язык Java - от Blackdown;
- кластерная утилита MPICH x86-64;
- оптимизированные математические библиотеки – от NAG.

Intel ® Cluster OpenMP

Это OpenMP для вычислительных систем с распределенной памятью.

Message Passing Interface – MPICH

MPICH-2. Настройка

Пример конфигурационного файла **.mpd.conf**

```
MPD_SECRETWORD=kalosha
MPD_PORT_RANGE=56000:57000
```

Пример файла **mpd.hosts**:

```
pd00
pd05
pd06
pd07
```

Команды запуска и управления демоном mpd

```
mpdboot –n 4
mpdallexit
mpdtrace –l
mpdcheck
```

Запуск демонов mpd в необходимом количестве и на хостах, заданных в файле **mpd.hosts** выполняется командой:

```
mpdboot –n  число_демонов
```

Завершение работы всех демонов:

```
mpdallexit
```

Проверка правильности установки пакета:

```
mpdcheck
```

Трассировка демонов mpd:

```
mpdtrace -l
```

Трансляция программ MPI для разных языков:

`mpicc`	C
`mpicxx`	C++
`mpif77`	Fortran
`mpif90`	

Запуск программ MPI:

```
mpiexec -n 4 -machinefile file ./a.out
```

```
mpirun -np 8 ./a.out
```