Профилирование программ

Алексей А. Романенко

arom@ccfit.nsu.ru

Профилирование

Сбор характеристик работы программы или системы с целью их дальнейшей оптимизации.

Характеристики

- Время выполнения функций\отдельных строк кода
- Количество вызовов функций
- Дерево вызовов
- Hotspots
- Доля параллелизма
- Загрузка CPU\сети\шины доступа к памяти
- Количество промахов в кэше
- пр.

Время выполнения

```
Bash# date; wc -l /etc/passwd; date
Срд Июл 28 17:43:29 NOVST 2010
     564 /etc/passwd
Срд Июл 28 17:43:29 NOVST 2010

Bash# time wc /etc/passwd
     564    1348   42335 /etc/passwd

real     0m0.011s
user     0m0.000s
sys      0m0.010s
```

Подходы

- Ручной
- Инструментальный
  - Инструмент — профилировщик

Недостатки ручного способа профилирования

- Вставка дополнительного кода в программу → новые ошибки и необходимость чистки кода
- Анализ результата в текстовом виде
- Большой объем выходных файлов
- Отсутствие дерева вызовов

Макросы для ручного профилирования

```
#define newT(name)   int name ## _timer_linef; \
                     struct timeval name ## _tv_1; \
                     struct timeval name ## _tv_2;

#define startT(name) gettimeofday(&name ## _tv_1, NULL); \
                     name ## _timer_linef = __LINE__;

#define printT(name) gettimeofday(&name ## _tv_2, NULL); \
                printf("%s:%d through %d takes %f usec \n", \
                    __FILE__, name ## _timer_linef ,__LINE__, \
   (name ## _tv_2.tv_sec - name ## _tv_1.tv_sec)*1e6 + \
   (name ## _tv_2.tv_usec - name ## _tv_2.tv_usec));
```

Пример использования

```
int main(...){
    ...
    newT(my_work);
    startT(my_work);
    do_something();
    printT(my_work);
    ...
}
```

```
test.c:23 through 31 takes 20.386 usec
```

Инструменты

какая часть программы нуждается в оптимизации?

- Профилировщик — инструмент анализа производительности программы. Сбор информации о поведение программы во время ее выполнения.
- Первые инструменты - начало 1970s
  - Использование прерываний по времени, сохоранение регистра PSW для поиска “hot spots” на IBM/360, IBM/370
- Вывод:
  - Trace
  - Sampling

Методы сбора данных

http://en.wikipedia.org/wiki/List_of_performance_analysis_tools

- Event based profilers
  - .NET, Java, Python, Ruby
- Statistical profilers
  - gprof
  - Oprofile
  - CodeAnalyst
  - VTune
  - Valgring
  - ...
- Hypervisor/Simulator
  - SIMMON, OLIVER

DTrace

- Разработан Sun Microsystems
- Трассировка в режиме реального времени
- Более 50 000 точек проверки
- Язык программирования «D»

- http://en.wikipedia.org/wiki/DTrace

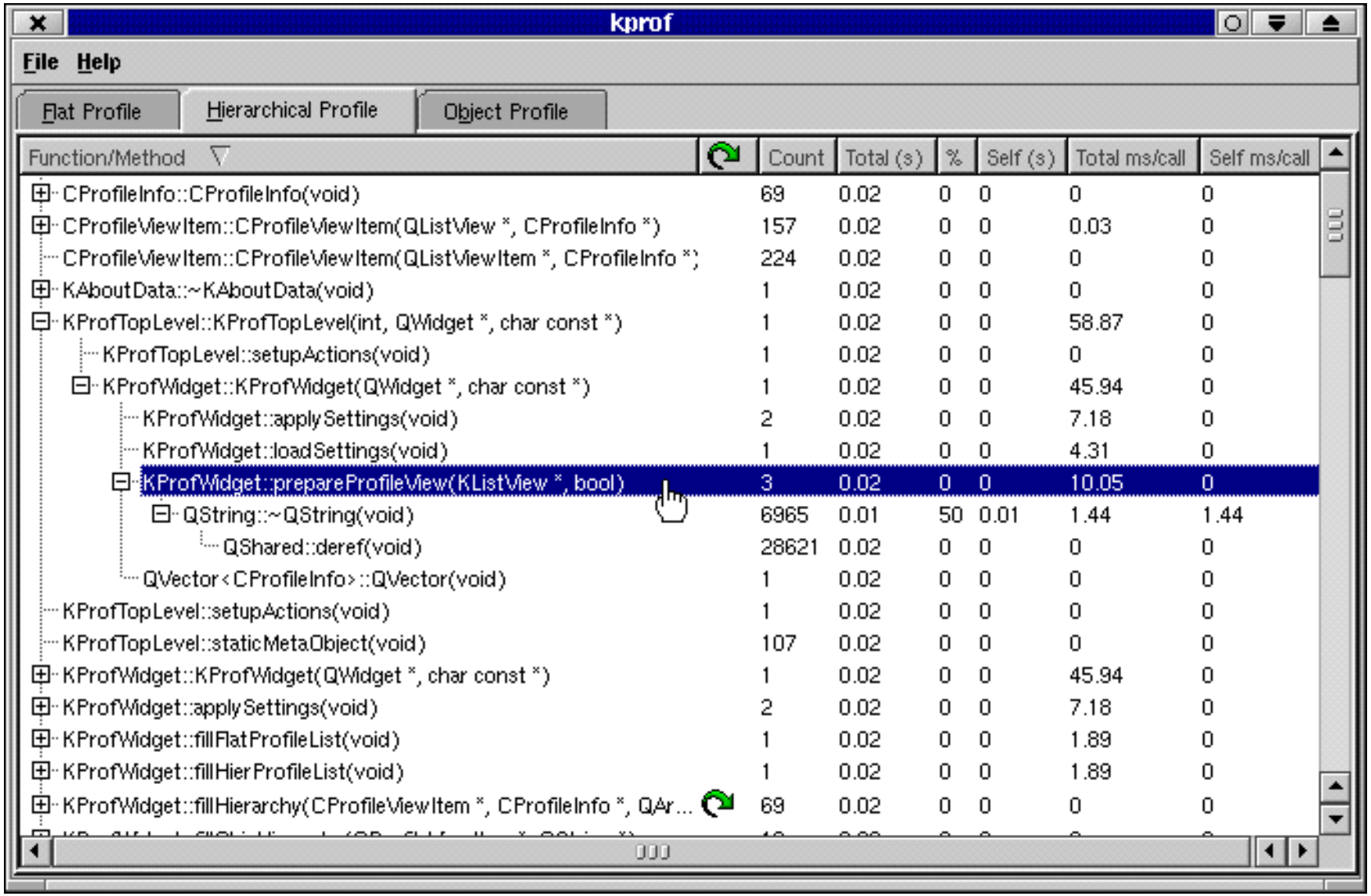
GProf

- GNU profiler
- Информация
  - Вызовы функций (количество/время)
  - Дерево вызовов (Call graph)
- Сборка программ
  - gcc -g -pg <прочие опции>
- Запуск программы → на выходе файл gmon.out

GProf. Пример

```
bash # gcc -O3 -g -pg test.c
bash # ./a.out
bash # ls
a.out    gmon.out    test.c
bash # gprof
...
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00      7.93     7.93        1     7.93     7.93  calculate
  0.00      7.93     0.00        1     0.00     0.00  init_depth
  0.00      7.93     0.00        1     0.00     0.00  init_h1_and_h2
  0.00      7.93     0.00        1     0.00     0.00  init_mareographs
...
index % time    self  children    called     name
                7.93    0.00       1/1           main [2]
[1]    100.0    7.93    0.00       1          calculate [1]
-----------------------------------------------
                0.00    0.00       1/1           init_params [6]
[3]      0.0    0.00    0.00       1          init_depth [3]
...
```

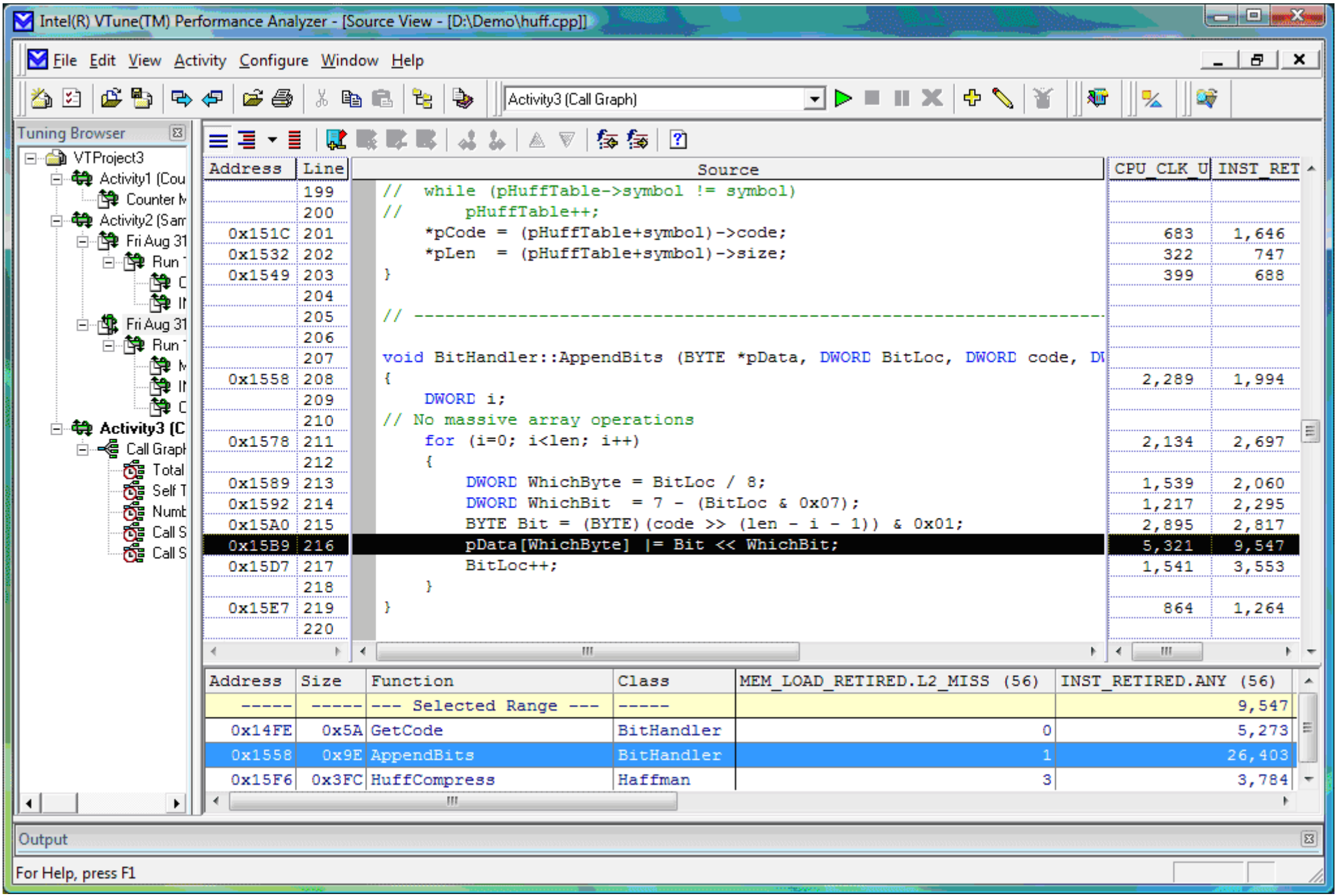
Kprof - GUI обертка для gprof

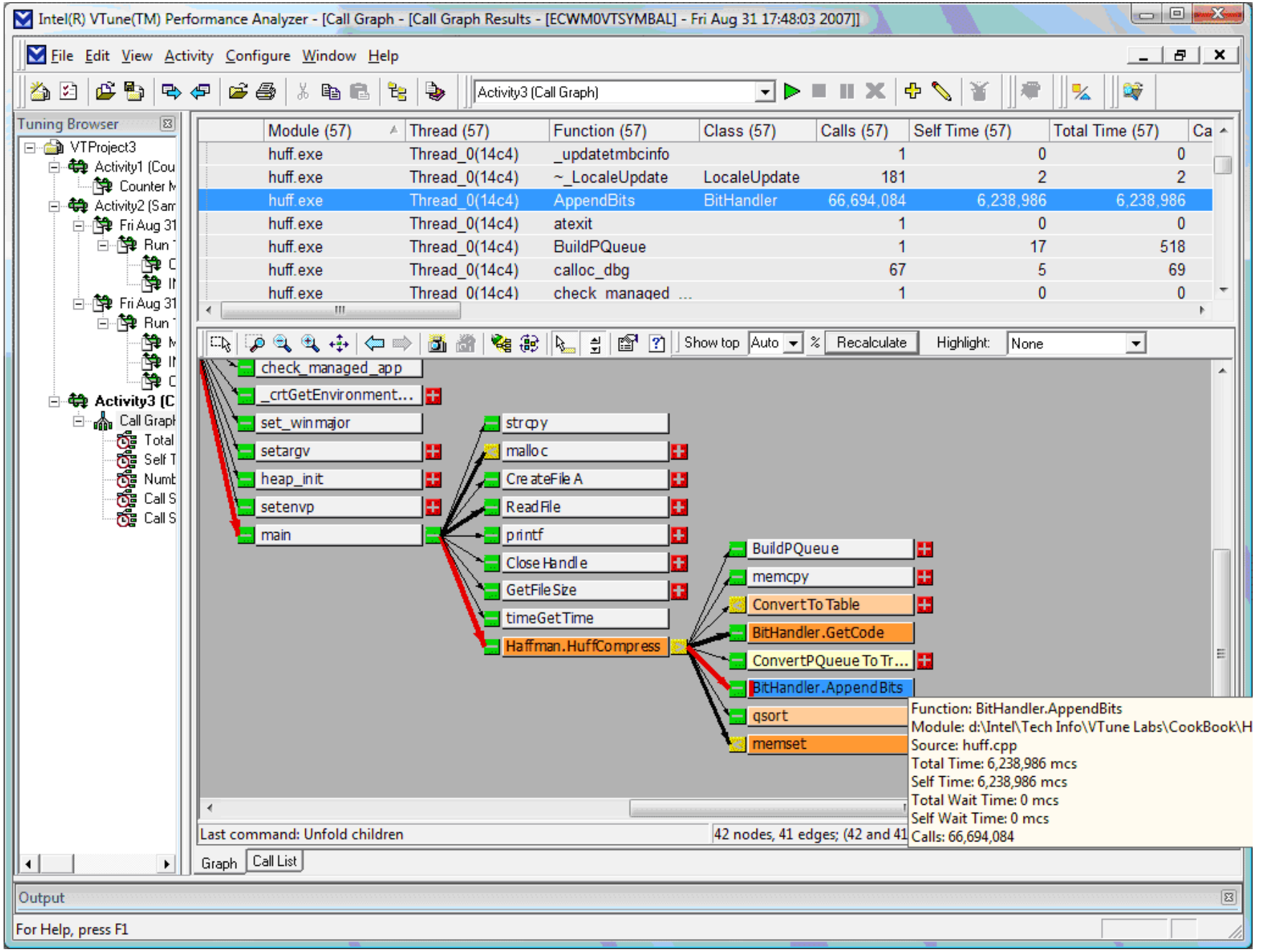


VTune

- Коммерческий продукт от Intel
- Информация
  - Дерево вызовов
  - Семплирование
  - Просмотр исходного кода
  - Мониторинг показаний
  - Профилирование потоков

- http://software.intel.com/en-us/intel-vtune/


Intel(R) VTune(TM) Performance Analyzer - [Source View - [D:\Demo\huff.cpp]]
File Edit View Activity Configure Window Help
Activity3 (Call Graph)
Tuning Browser
VTProject3
Activity1 (Cou
Counter M
Activity2 (Sam
Fri Aug 31
Run
Fri Aug 31
Run
Activity3 (C
Call Graph
Total
Self T
Numb
Call S
Call S
Address Line Source CPU_CLK_U INST_RET
199 // while (pHuffTable->symbol != symbol)
200 // pHuffTable++;
0x151C 201 *pCode = (pHuffTable+symbol)->code; 683 1,646
0x1532 202 *pLen = (pHuffTable+symbol)->size; 322 747
0x1549 203 } 399 688
204
205 // ----------
206
207 void BitHandler::AppendBits (BYTE *pData, DWORD BitLoc, DWORD code, D
0x1558 208 { 2,289 1,994
209 DWORD i;
210 // No massive array operations
0x1578 211 for (i=0; i<len; i++) 2,134 2,697
212 {
0x1589 213 DWORD WhichByte = BitLoc / 8; 1,539 2,060
0x1592 214 DWORD WhichBit = 7 - (BitLoc & 0x07); 1,217 2,295
0x15A0 215 BYTE Bit = (BYTE)(code >> (len - i - 1)) & 0x01; 2,895 2,817
0x15B9 216 pData[WhichByte] |= Bit << WhichBit; 5,321 9,547
0x15D7 217 BitLoc++; 1,541 3,553
218 }
0x15E7 219 } 864 1,264
220
Address Size Function Class MEM_LOAD_RETIRED.L2_MISS (56) INST_RETIRED.ANY (56)
----- ----- --- Selected Range --- ----- 9,547
0x14FE 0x5A GetCode BitHandler 0 5,273
0x1558 0x9E AppendBits BitHandler 1 26,403
0x15F6 0x3FC HuffCompress Haffman 3 3,784
Output
For Help, press F1


Intel(R) VTune(TM) Performance Analyzer - [Call Graph - [Call Graph Results - [ECWM0VTSYMBAL] - Fri Aug 31 17:48:03 2007]]
File Edit View Activity Configure Window Help
Activity3 (Call Graph)
Tuning Browser
VTProject3
Activity1 (Cou
Counter M
Activity2 (Sam
Fri Aug 31
Run
Fri Aug 31
Run
Activity3 (C
Call Graph
Total
Self T
Numb
Call S
Call S
Module (57) Thread (57) Function (57) Class (57) Calls (57) Self Time (57) Total Time (57) Ca
huff.exe Thread_0(14c4) _updatetmbcinfo 1 0 0
huff.exe Thread_0(14c4) ~_LocaleUpdate LocaleUpdate 181 2 2
huff.exe Thread_0(14c4) AppendBits BitHandler 66,694,084 6,238,986 6,238,986
huff.exe Thread_0(14c4) atexit 1 0 0
huff.exe Thread_0(14c4) BuildPQueue 1 17 518
huff.exe Thread_0(14c4) calloc_dbg 67 5 69
huff.exe Thread_0(14c4) check_managed ... 1 0 0
Show top Auto % Recalculate Highlight: None
check_managed_app
_crtGetEnvironment...
set_winmajor
setargv
heap_init
setenvp
main
strcpy
malloc
CreateFileA
ReadFile
printf
CloseHandle
GetFileSize
timeGetTime
Haffman.HuffCompress
BuildPQueue
memcpy
ConvertToTable
BitHandler.GetCode
ConvertPQueueToTr...
BitHandler.AppendBits
qsort
memset
Function: BitHandler.AppendBits
Module: d:\Intel\Tech Info\VTune Labs\CookBook\H
Source: huff.cpp
Total Time: 6,238,986 mcs
Self Time: 6,238,986 mcs
Total Wait Time: 0 mcs
Self Wait Time: 0 mcs
Calls: 66,694,084
Last command: Unfold children
42 nodes, 41 edges; (42 and 41
Graph Call List
Output
For Help, press F1

Профилирование параллельных программ

- Профилирование взаимодействия процессов\потоков
  - Создание\уничтожение потоков
  - Синхронизация (примитивы синхронизации)
  - Неравномерное распределение работ
- Специальные профилировщики

Инструменты профилирования и отладки параллельных программ (MPI)

- HeNCE
- TRAPPER
- EDPEPPS
- GRADE
- AIMS (An Automated Instrumentation and Monitoring System)
- Vampir, VampirTrace
- Pablo Performance Analysis Toolkit Software
- Paradyn
- Jumpshot, Nupshot
- Puma
- CXperf

Профилировщики параллельных программ (SMP)

- Intel® Thread Profiler
  www.intel.com/cd/software/products/asmo-na/eng/286749.htm
- Intel® VTune Performance Analyzer
  www.intel.com/cd/software/products/asmo-na/eng/vtune/239144.htm
- Intel® Threading Analysis Tools
  www.intel.com/cd/software/products/asmo-na/eng/threading/219785.htm
- Intel® Trace Analyzer and Collector 7.1
  http://www.intel.com/cd/software/products/asmo-na/eng/306321.htm