PERFORMANCE TOOLS DEVELOPMENTS

Roberto A. Vitillo
presented by Paolo Calafiura & Wim Lavrijsen
Lawrence Berkeley National Laboratory

Future computing in particle physics, 16 June 2011

LINUX PERFORMANCE EVENTS SUBSYSTEM

- The perf events subsystem was merged into the Linux kernel in version 2.6.31 and introduced the *sys_perf_event_open* system call
- Uses special purpose registers on the CPU to count the number of "events"
- An HW event can be, for example, the number of cache miss suffered or mispredicted branches
- SW events, like page misses, are also supported
- Performance counters are accessed via file descriptors using the above mentioned system call

LINUX PERFORMANCE EVENTS SUBSYSTEM (2)

- perf is an user space utility that is part of the kernel repository
- Available in Scientific Linux 6
- Basic usage: data is collected by using the perf-record tool and displayed with perf-report

THE PERF TOOL: EXAMPLE USAGE

vitillo@mobile-eniac: ~/sandbox/gcc

File Edit View Search Terminal Help

```
vitillo@mobile-eniac:~/sandbox/gcc$ perf record -c 1000 g++ -O3 helloworld.cpp
[ perf record: Woken up 3 times to write data ]
[ perf record: Captured and wrote 0.671 MB perf.data (~29314 samples) ]
vitillo@mobile-eniac:~/sandbox/gcc$ perf report -U --stdio | head -n 20
# Events: 3K cycles
#
# Overhead  Command                      Shared Object                               Symbol
# ........  .......  ..................................  .................................
#
    27.28%       ld  libbfd-2.20.51-system.20100908.so  [.] bfd_hash_lookup
    17.03%       ld  libbfd-2.20.51-system.20100908.so  [.] bfd_elf_link_add_symbols
    13.35%       ld  libbfd-2.20.51-system.20100908.so  [.] bfd_hash_traverse
     2.97%       ld  libbfd-2.20.51-system.20100908.so  [.] _bfd_elf_archive_symbol_lookup
     2.05%      g++  libc-2.12.1.so                     [.] __GI___strncmp_ssse3
     1.69%       ld  libc-2.12.1.so                     [.] __strchr_sse2
     1.38%       ld  libc-2.12.1.so                     [.] __GI___strcmp_ssse3
     1.36%      g++  libc-2.12.1.so                     [.] __strlen_sse2
     1.26%      g++  libc-2.12.1.so                     [.] __strncmp_sse2
     1.00%      g++  libc-2.12.1.so                     [.] _int_malloc
     0.97%       ld  libc-2.12.1.so                     [.] __GI_memset
     0.97%       ld  libbfd-2.20.51-system.20100908.so  [.] bfd_link_hash_lookup
     0.95%      g++  libc-2.12.1.so                     [.] memcpy
     0.85%      g++  libc-2.12.1.so                     [.] __malloc
     0.82%       ld  libbfd-2.20.51-system.20100908.so  [.] _bfd_generic_link_add_one_symbol
vitillo@mobile-eniac:~/sandbox/gcc$ 
```

WHY DO WE CARE?

- The Linux Performance Events Subsystem provides a **low overhead** way to measure the workloads of a single application or the full system
- It's at least an order of magnitude faster than an instrumenting profiler
- It provides far more information compared to statistical profiler

WHAT IS MISSING

- Annotating the objdump output one event at a time is not enough for efficiently finding bottlenecks
- A real GUI that can display multiple events and their relations is missing
- New CPU's have a buffer that records the last taken branches but a support to exploit it is missing

PERF EVENTS CONVERTER

- As a first step a converter tool for the perf-tools data format has been introduced
- The tool is capable to convert a perf data file to a **callgrind** one that can be displayed with **kcachegrind**:
  - multiple events are supported
  - annotated source code, assembly and function list view
  - complete inline chain

PERF EVENTS CONVERTER (2)

callgrind.out

File View Go Settings Help

Open | Back | Forward | Up | % Relative | Cycle Detection | Relative to Parent | <> Shorten Templates | cycles

Flat Profile

Search: (No Grouping)

Self	Function	Location
99.81	fastfib(unsigned int)	a.out: fib.cpp
0.06	_dl_map_object	ld-2.12.1.so: dl-load.c
0.04	main	a.out: fib.cpp
0.02	_dl_map_object_deps	ld-2.12.1.so: dl-deps.c
0.01	_dl_load_cache_lookup	ld-2.12.1.so: dl-cache.c
0.01	_dl_name_match_p	ld-2.12.1.so: dl-misc.c
0.01	__sigsetjmp	ld-2.12.1.so: setjmp.S
0.01	_dl_catch_error	ld-2.12.1.so: dl-error.c
0.01	index	ld-2.12.1.so: strchr.S
0.01	openaux	ld-2.12.1.so: dl-deps.c
0.01	strcmp	ld-2.12.1.so: strcmp.S
0.00	__GI_strlen	ld-2.12.1.so: rtld-strlen.S
0.00	__libc_memalign	ld-2.12.1.so: dl-minimal.c
0.00	_dl_cache_libcmp	ld-2.12.1.so: dl-cache.c
0.00	_dl_important_hwcaps	ld-2.12.1.so: dl-sysdep.c
0.00	_dl_init_paths	ld-2.12.1.so: dl-load.c
0.00	_dl_map_object_from_fd	ld-2.12.1.so: dl-load.c, dynamic-link.h
0.00	_dl_new_object	ld-2.12.1.so: dl-object.c
0.00	_dl_start	ld-2.12.1.so: rtld.c
0.00	access	ld-2.12.1.so: syscall-template.S
0.00	dl_main	ld-2.12.1.so: rtld.c, dynamic-link.h
0.00	memcpy	ld-2.12.1.so: memcpy.S
0.00	memset	ld-2.12.1.so: memset.c

fastfib(unsigned int)

Types | Callers | All Callers | Callee Map | Source Code

#	cycles	instruct	Source ('/home/vitillo/sandbox/libpfm4/fib.cpp')
8	7.23	10.35	if(i<2) *p=i;
9			else {
10	13.86	14.40	if(p==a) *p=*(a+1)+*(a+2);
11	14.32	12.80	else if(p==a+1) *p=*a+*(a+2);
12	4.99	6.85	else *p=*a+*(a+1);
13			}
14	42.06	39.09	if(++p>a+2) p=a;
15			}
16			
17	0.02	0.03	return p==a?*(p+2):*(p-1);
18	0.01	0.00	}
19			
20			
21			

#	cycles	instruc	Hex		Assembly Instructions	Source Position
40 0622	9.38	8.54	48 83 45 f0 04	addq	$0x4,-0x10(%rbp)	fib.cpp:14
40 0627	2.87	3.07	48 8d 45 e0	lea	-0x20(%rbp),%rax	fib.cpp:14
40 062B			48 83 c0 08	add	$0x8,%rax	
40 062F	5.07	4.88	48 3b 45 f0	cmp	-0x10(%rbp),%rax	fib.cpp:14
40 0633	9.24	8.50	0f 92 c0	setb	%al	fib.cpp:14
40 0636	4.67	2.53	84 c0	test	%al,%al	fib.cpp:14
40 0638	9.57	9.01	74 08	je	400642 <fastfib(unsigned int)+0xae>	fib.cpp:14
40 063A	1.26	2.56	48 8d 45 e0	lea	-0x20(%rbp),%rax	fib.cpp:14
40 063E			48 89 45 f0	mov	%rax,-0x10(%rbp)	
40 0642	4.45	3.89	83 45 fc 01	addl	$0x1,-0x4(%rbp)	fib.cpp:7
40 0646	2.54	2.05	8b 45 fc	mov	-0x4(%rbp),%eax	fib.cpp:7
40 0649	0.22	0.53	3b 45 dc	cmp	-0x24(%rbp),%eax	fib.cpp:7
40 064C	6.77	5.30	0f 96 c0	setbe	%al	fib.cpp:7

Parts | Callees | Call Graph | All Callees | Caller Map | Machine Code

callgrind.out [1] - Total cycles Cost: 24 646

PERF EVENTS VISUALIZER

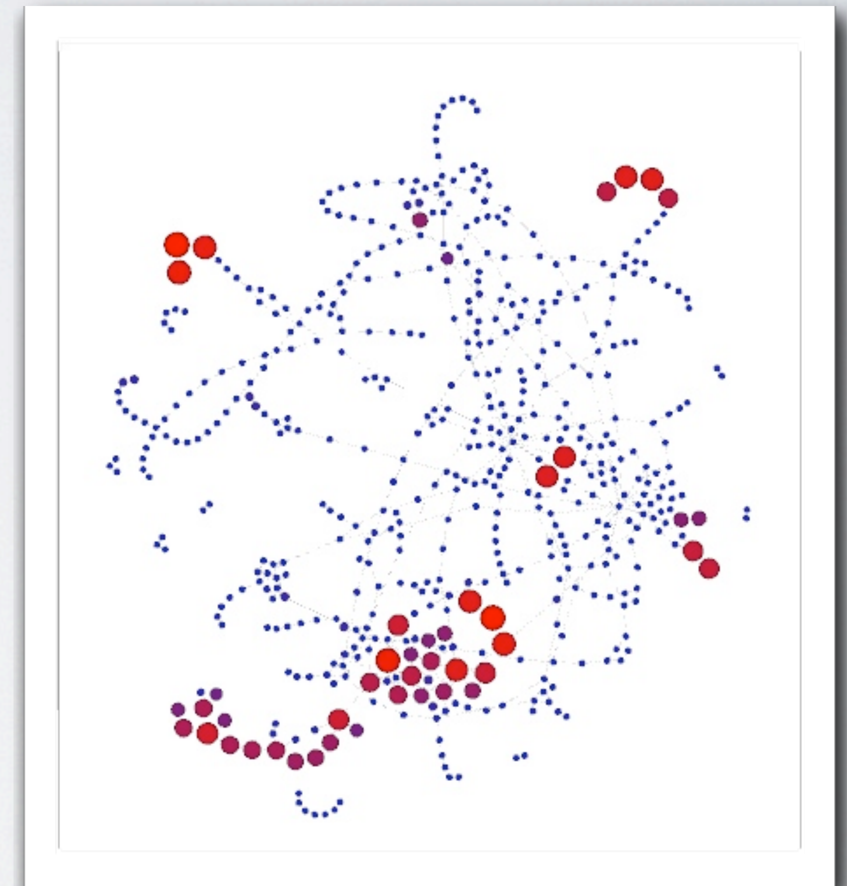
- KCachegrind doesn't permit to show an arbitrary number of events at the same time
- A new converter and a web-based GUI is under development
- The converter reads the a raw perf data file and produces spreadsheets, cycle accounting trees and call graphs
- The GUI will be able to:
  - present the available data in spreadsheets, cycle accounting trees and callgraphs
  - **offer insights on the callgraph**, e.g. mark as hot virtual methods with high call counts
  - **correlate different HW/SW events to gain a deeper understanding of the performance bottlenecks**

LAST BRANCH RECORD SUPPORT

- New Intel processors have a cyclic buffer that can record taken branches
- Each recorded branch is composed of a pair of registers for source and destination
- Last Branch Records (LBR) **sampling** can be used to, e.g.
  - evaluate the frequency of function calls and perform inline decisions
  - yield the partial path of an event
  - building a partial callgraph

IMPORTANCE OF LBR

- Atlas Software Issues:
  - low instruction retired / call retired ratio
  - high call retired / branch retired ratio
- Inlining functions called millions of times per event can indeed bring considerable benefits
- David Levinthal's proposal:
  - "Use LBR and static analysis to evaluate frequency and cost of function calls"
  - "Use social network analysis / network theory to identify clusters of active, costly function call activity"
  - "Order cluster by total cost and inline"



LBR DEVELOPMENTS

- Kernel patch for filtering and dumping of the LBR is completed; After validation the patch will be integrated in the kernel trunk
- The perf report user space utility has a new feature to display statistics about the taken branches

EXPLOITING THE LBR IN PERF

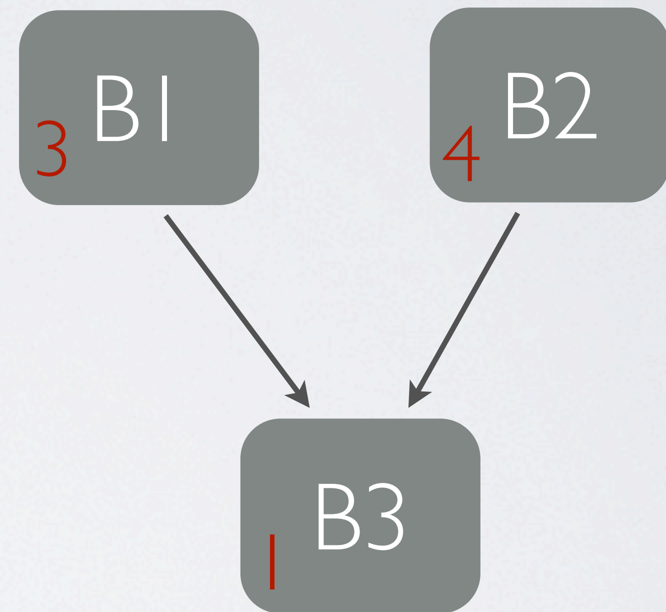
- Statistics about DSO to DSO and Symbol to Symbol supported
- Optionally distinguish between predicted and mispredicted branches
- **Filtering support**

```
vitillo@mobile-eniac: ~/sandbox/gcc
File  Edit  View  Search  Terminal  Help
vitillo@mobile-eniac:~/sandbox/gcc$ perf report -U -b --sort dso --column-widths=10,35
Warning: TUI interface not supported in branch mode
# Events: 7K cycles
#
# Overhead  Source Shared Object                 Target Shared Object
# ........  ...................................  ....................................
#
    74.06%  libbfd-2.20.51-system.20100908.so    libbfd-2.20.51-system.20100908.so
    17.80%  libc-2.12.1.so                       libc-2.12.1.so
     3.90%  ld-2.12.1.so                         ld-2.12.1.so
     2.04%  libbfd-2.20.51-system.20100908.so    libc-2.12.1.so
     0.62%  libc-2.12.1.so                       libbfd-2.20.51-system.20100908.so
     0.57%  g++-4.4                              libc-2.12.1.so
     0.19%  ld.bfd                               libbfd-2.20.51-system.20100908.so
     0.18%  ld.bfd                               libc-2.12.1.so
     0.17%  ld-2.12.1.so                         libc-2.12.1.so
     0.14%  cc1plus                              libc-2.12.1.so
     0.14%  collect2                             libc-2.12.1.so
     0.06%  collect2                             collect2
     0.04%  g++-4.4                              g++-4.4
     0.03%  cc1plus                              cc1plus
     0.03%  libc-2.12.1.so                       ld-2.12.1.so
     0.01%  cc1plus                              ld-2.12.1.so
     0.01%  libc-2.12.1.so                       ld.bfd

vitillo@mobile-eniac:~/sandbox/gcc$
```

TODO

- Use a recursive disassembler instead of a linear one?
- Disassemble a module/function on the fly?
- Improve basic block counts by:
  - using LBR to generate software instruction retired event
  - adhering to flow conservation rules while limiting the amount of changes to sample counts to a minimum



In general with sampling
#B1 + #B2 != #B3

CONCLUSIONS

- The callgrind converter and the new GUI under development will offer an easy way to non experts to navigate and understand the profiled application

- The LBR support adds important profiling possibilities, vital for OO SW, to the Linux Performance Events Subsystem