

COMP3320/COMP6464: Hardware Performance Counters

Alistair Rendell

See various articles linked to lecture note web site

7.1 Why

- We've seen that modern machines are complex
 - pipelining
 - superscalar
 - load/store architectures
 - memory hierarchy
- Understanding observed performance is not easy
- Performance counters enable us to count critical events and provide a very accurate means of assessing how well we are using the computer system

Can you suggest some quantities that we might want to count?

7.2 Hardware Performance Counters

- All (nearly?) modern microprocessors have them
- Typically a group of registers that keep track of programmable events
- Provide high resolution data on many performance related variables, e.g.
 - cycles
 - instruction count
 - floating point operations
 - cache references
 - TLB misses
 - data/instruction stalls
- Enable vendors to better understand the performance of existing code on their hardware
- Enable users to build better (faster) software

7.3 Accessing Hardware Counters

- Cray YMP provided HPM that gave info on vector lengths and flops
 - enabled users to quote macho flops!
- Used to build metrics
- Used in system wide tools (Solaris provides: cputrack, cpustat, har)
- Accessed via libraries
 - vendor specific: (libcpc, libpctx)
 - portable (PCL, PAPI)
 - futher GUI often provided to provide higher level analysis
- There are no standard counters
 - different vendors have different counters
 - different generations of the same processor may have different counters (eg UII and UIII)

7.4 Simple Metrics

$$\text{MFLOPS} = \frac{\text{Flp_Instr_Exec}}{\text{Cycles}} \times \text{clock(MHz)}$$

$$\text{MIPS} = \frac{\text{Int_Instr_Exec}}{\text{Cycles}} \times \text{clock(MHz)}$$

$$\text{IPC} = \frac{\text{All_Instr_Exec}}{\text{Cycles}}$$

$$\text{L1Hits} = 1 - \frac{\text{L1_Misses}}{\text{Loads} + \text{Stores}}$$

$$\text{L2Hits} = 1 - \frac{\text{L2_Misses}}{\text{L1_Misses}}$$

$$\text{Branch_rate} = \frac{\text{Decoded_Branches}}{\text{Total_Instruction_Exec.}}$$

7.5 More Complex Metrics

$$\text{L1 - L2_bandwidth} = \frac{L1_Misses \times L1_Line_Size}{Cycles} \times \text{clock}(MHz)$$

$$\text{L2 - RAM_bandwidth} = \frac{L2_Misses \times L2_Line_Size}{Cycles} \times \text{clock}(MHz)$$

$$\text{Data_Stall} = \frac{Load_Use + Load_Use_Raw + Store_Buf_Full}{Cycles}$$

7.6 Sun Performance Counters

- Two 32 bit counters (PIC0 and PIC1)
 - max 4 billion events - so will overflow
 - programmed to record different events (ultrall)

Code	Register	Meaning
Cycle_cnt	0+1	Accumulated cycles
Instr_cnt	0+1	Completed instructions (not annulled ones)
IC_ref	0	I-cache references
IC_hits	1	I-cache hits
DC_rd	0	D-cache read reference
DC_rd_hit	1	D-cache read hit
DC_wr	0	D-cache write reference
DC_wr_hit	1	D-cache write hit
EC_ref	0	Total e-cache references
EC_hit	1	Total e-cache hits
EC_wb	1	e-cache misses that do writebacks

7.7 UltraSPARC III Counters

event0: Cycle_cnt Instr_cnt Dispatch0_IC_miss IC_ref DC_rd DC_wr
EC_ref EC_snoop_inv Dispatch0_br_target Dispatch0_2nd_br
Rstall_storeQ Rstall_IU_use EC_write_hit_RTO EC_rd_miss
PC_port0_rd SI_snoop SI_ciq_flow SI_owned SW_count_0
IU_Stat_Br_miss_taken IU_Stat_Br_count_taken
Dispatch_rs_mispred FA_pipe_completion MC_reads_0
MC_reads_1 MC_reads_2 MC_reads_3 MC_stalls_0 MC_stalls_2

event1: Cycle_cnt Instr_cnt Dispatch0_mispred EC_wb EC_snoop_cb
IC_miss_cancelled Re_FPU_bypass Re_DC_miss Re_EC_miss
IC_miss DC_rd_miss DC_wr_miss Rstall_FP_use EC_misses
EC_ic_miss Re_PC_miss ITLB_miss DTLB_miss WC_miss
WC_snoop_cb WC_scrubbed WC_wb_wo_read PC_soft_hit
PC_snoop_inv PC_hard_hit PC_port1_rd SW_count_1
IU_Stat_Br_miss_untaken IU_Stat_Br_count_untaken
PC_MS_misses MC_writes_0 MC_writes_1 MC_writes_2
MC_writes_3 MC_stalls_1 MC_stalls_3 Re_RAW_miss
FM_pipe_completion Re_endian_miss

7.8 Metrics for Ultra II and Ultra III

Metric	Ultra II	Ultra III
mips	$\text{instr_cnt}/\text{tick_cnt}*\text{clock}$	$\text{instr_cnt}/\text{tick_cnt}*\text{clock}$
cpi	$\text{cycle_cnt}/\text{instr_cnt}$	$\text{cycle_cnt}/\text{instr_cnt}$
flops		$(\text{fa_pipe_completion} + \text{fm_pipe_completion}) / \text{tick_cnt}*\text{clock}$
address bus utilization	$2 * (\text{ec_ref} - \text{ec_hit} + \text{ec_wb}) / (\text{tick_cnt} * \text{bus_clock}/\text{cpu_clock})$	$(\text{ec_misses} + \text{ec_wb}) / (\text{tick_cnt} + \text{bus_clock}/\text{cpu_clock})$
D-cache miss rate	$1 - (\text{dc_rd_hit} + \text{dc_wr_hit})/(\text{dc_rd} + \text{dc_wr})$	$(\text{dc_rd_miss} + \text{dc_wr_miss})/(\text{dc_rd} + \text{dc_wr})$
I-cache miss rate	$1 - \text{ic_hit}/\text{ic_ref}$	$(\text{ic_miss} - \text{ic_miss_canceled}) / (\text{ic_ref} + \text{ic_miss} - \text{ic_miss_canceled})$
L2 miss rate	$1 - \text{ec_hit}/\text{ec_ref}$	$\text{ec_misses}/\text{ec_ref}$
I-TLB miss rate		$\text{itlb_miss}/\text{ic_ref}$
D-tlb miss rate		$\text{dtlb_miss}/(\text{dc_rd} + \text{dc_wr})$
branch rate		$(\text{i_u_stat_br_count_taken} + \text{i_u_stat_br_count_untaken}) / \text{instr_cnt}$
Data stall rate	$(\text{load_use} + \text{load_use_raw} + \text{dispatch0_storebuf})/\text{cycle_cnt}$	$(\text{re_dc_miss} + \text{rstall_storeq} + \text{rstall_iu_use} + \text{re_raw_miss} + 12 * (\text{re_pc_miss} + \text{re_fpu_bypass} + \text{re_endian_miss})) / \text{cycle_cnt}$

7.9 Sun Tools: CPUSTAT

- Provides a system wide report on CPU counters

- need to be root or have suid set to root

```
cpustat -c eventspec [-c eventspec ] ... [-ntD] [interval[count]]
```

```
cpustat -c Cycle_cnt,Instr_cnt -c EC_ref,EC_misses 1 5
```

time	cpu	event	pic0	pic1	
1.007	1	tick	48959	4640	# pic0=Cycle_cnt,pic1=Instr_cnt
1.007	0	tick	271729	39874	# pic0=Cycle_cnt,pic1=Instr_cnt
2.007	0	tick	8768	390	# pic0=EC_ref,pic1=EC_misses
2.007	1	tick	11071	1410	# pic0=EC_ref,pic1=EC_misses
3.007	0	tick	118110	47512	# pic0=Cycle_cnt,pic1=Instr_cnt
3.007	1	tick	238131	57790	# pic0=Cycle_cnt,pic1=Instr_cnt
4.007	0	tick	6215	116	# pic0=EC_ref,pic1=EC_misses
4.007	1	tick	2214	197	# pic0=EC_ref,pic1=EC_misses
5.007	0	tick	139530	76730	# pic0=Cycle_cnt,pic1=Instr_cnt
5.007	1	tick	607836	228673	# pic0=Cycle_cnt,pic1=Instr_cnt
5.007	2	total	1424295	455219	# pic0=Cycle_cnt,pic1=Instr_cnt
4.007	2	total	28268	2113	# pic0=EC_ref,pic1=EC_misses

7.10 Sun Tools: CPUTRACK

- Provides counter information on a specific process

- does not require root access

```
cpustrack -c eventspec [-c eventspec ] ... [-efntvD] [-N count] [-o  
pathname] [-T interval] command [args]
```

```
# cputrack -T 0 -fve -c Cycle_cnt,Instr_cnt sh -c date
```

time	pid	lwp	event	pic0	pic1
0.008	7193	1	init_lwp	0	0
0.024	7193	1	fork		# 7194
0.028	7194	1	init_lwp	0	0
0.031	7194	1	fini_lwp	85848	32003
0.031	7194	1	exec	85848	32003
0.000	7194	1	exec		# 'date'
0.043	7194	1	init_lwp	0	0
Tue Apr 1 23:23:10 EST 2003					
0.053	7194	1	fini_lwp	824313	366668
0.053	7194	1	exit	824313	366668
0.058	7193	1	fini_lwp	1058364	411754
0.058	7193	1	exit	1058364	411754

7.11 Hard Coded Instrumentation: libcpc

```
/* example_libcpc.c */
/* cc -fast -fsimple=2 -xchip=ultra3 -xarch=v8plusa \
   example_libcpc.c -lcpc -o example_libcpc */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <inttypes.h>
#include <libcpc.h>
#include <errno.h>

#define N (500019+1)
#define TIMES 100

int main(int argc, char **argv) {

    int i,j,k,nl,nll;
    double sum = 0.0, b[N], a[N];
    double t1=0.0,t2=0.0;
    int cpuver, iter;
    char *setting = NULL;
    cpc_event_t event;
    long long fltadd=0, fltmult=0, fltaddtheo=0, fltmulttheo=0;
```

```

if ((cpuver = cpc_getcpuver()) == -1) {
    printf("no performance counter hardware \n");
} else {
    printf("hardware identifier %d\n",cpuver);
}

if ((setting = getenv("PERFEVENTS")) == NULL)
    setting = "pic0=FA_pipe_completion,pic1=FM_pipe_completion";

if (cpc_strtoevent(cpuver,setting,&event) != 0)
    printf("Cannot measure %s on this processor\n",setting);
setting = cpc_eventtostr(&event);

if (cpc_bind_event(&event, 0) == -1)
    printf("cannot bind lwp %d %s\n",_lwp_self(),
           strerror(errno));

nl = N-1;
fltaddtheo = ((long long)TIMES)*(N-1);
fltmulttheo = (((long long)TIMES)*(N-1));

{cpc_event_t before, after;
sum = 0.0;
for (j = 0; j < TIMES; j++) {
    for (i = 0; i < N; i++)

```

```

        b[i] = 0.3 + 0.001*j + 0.00001*i;
for (i = 0; i < N; i++)
    a[i] = 0.1 + 0.003*(j-1) + 0.0004*i;
if (cpc_take_sample(&before) == -1) exit(-1);
    for (k = 0; k < N-1; k++)
        sum = sum + a[k]*b[k];
if (cpc_take_sample(&after) == -1) exit(-1);
fltadd += (after.ce_pic[0] - before.ce_pic[0]);
fltmult += (after.ce_pic[1] - before.ce_pic[1]);
}
}

printf("Measured Flt Adds %lld\n",fltadd);
printf("Measured Flt Mults %lld\n",fltmult);
printf("Theoretical Flt Adds %lld\n",fltaddtheo);
printf("Theoretical Flt Mults %lld\n",fltmulttheo);
printf("sum = %14.7e\n", sum);
}
apr900@alto example_libcpc
hardware identifier 1002
Measured Flt Adds 50002100
Measured Flt Mults 50001900
Theoretical Flt Adds 50001900
Theoretical Flt Mults 50001900
sum = 1.8451251e+10

```

7.12 Problems and Alternatives?

- Counters provide lots of nice information, but care required
 - use of correct counter
 - counter working correctly (lmbench)
 - hard coded correctly
 - special consideration missed (madd)
 - scripts correct and working
 - counter overflow
- What are the alternatives?