Speeding up system calls on the Power Architecture

Open Source Summit Japan 2018

Nicholas Piggin
21 June 2018

IBM

Agenda

What is a system call?

Power ISA system call

Linux/powerpc code

Performance analysis

Problems and solutions

New POWER9 system call

What is a system call?

- System call is like a function call: saves where it came from so it can return.
- Difference is it sets Machine State Register (MSR) to privileged mode.
- MSR controls privilege level, facilities used (e.g., fp, vector), interrupts, etc.

PC	INSN	
0x0100	ld	r3, str1
0x0104	ld	r4, str2
0x0108	bl	strcmp
0x0250	lbz	r9, 0(r3)
0x0254	lbz	r10, 0(r4)
0x0258	cmp	r9, r10

MSR	PC	INSN	
USER	0x0100	li	r0, SYS_open
USER	0x0104	ld	r3, filename
USER	0x0108	li	r4, O_RDWR
USER	0x010c	sc	
KERN	0xcc00	mr	r9,r13
KERN	0xcc04	mfspr	r13,SPRG1
KERN	0xcc08	mfspr	r11,SRR0
KERN	0xcc0c	mfspr	r12,SRR1

Agenda

What is a system call?

Power ISA system call

Linux/powerpc code

Performance analysis

Problems and solutions

New POWER9 system call

Power ISA system call

- **sc** instruction (system call):
  - Address of next instruction will be the system call handler address (0xc000000000004c00).
  - MSR of next instruction will be kernel privileged, interrupts off, RI off.
  - SRR0 and SRR1 “Save Restore Registers” contain instruction address after the **sc**, and the previous MSR respectively.
- **rfi** class of instructions (return from interrupt):
  - Next instruction address and MSR is loaded from SRR0 and SRR1.

Power ISA system call – SRR registers

- SRR0/1 are also used by other interrupts, including non-maskable interrupts (NMI).

MSR	PC	INSN	
USER	0x0100	li	r0, SYS_open
USER	0x0104	ld	r3, filename
USER	0x0108	li	r4, O_RDWR
USER	0x010c	sc	
KERN	0xcc00	mr	r9,r13
KERN	0xcc04	mfspr	r13,SPRG1
KERN	0xcc08	mfspr	r11,SRR0 (0x0110)
KERN	0xcc0c	mfspr	r12,SRR1 (USER)

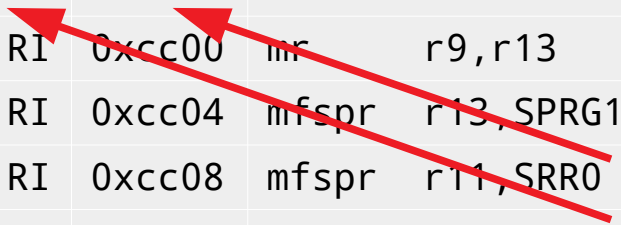
Power ISA system call – Non-Maskable Interrupt

NMI here will overwrite SRR0/1 and would cause the syscall to return to here when it is finished. Chaos ensues.

MSR	PC	INSN	
USER	0x0100	li	r0, SYS_open
USER	0x0104	ld	r3, filename
USER	0x0108	li	r4, O_RDWR
USER	0x010c	sc	
KERN	0xcc00	mr	r9,r13
KERN	0xcc04	mfspr	r13,SPRG1
KERN	0xcc08	mfspr	r11,SRR0 (0xcc04)
KERN	0xcc0c	mfspr	r12,SRR1 (KERN)

Power ISA system call – NMI and MSR RI bit

MSR	PC	INSN	
USER	0x0100	li	r0, SYS_open
USER	0x0104	ld	r3, filename
USER	0x0108	li	r4, O_RDWR
USER	0x010c	sc	
KERN !RI	0xcc00	mr	r9,r13
KERN !RI	0xcc04	mfspr	r13,SPRG1
KERN !RI	0xcc08	mfspr	r11,SRR0 (0x0110)
KERN !RI	0xcc0c	mfspr	r12,SRR1 (USER)
KERN !RI	0xcc10	mtmsr	RI
KERN	0xcc14	...	



Power ISA system call – NMI and MSR RI bit

If MSR RI bit is clear, that indicates NMI can not recover. The program is terminated instead.

MSR	PC	INSN	
USER	0x0100	li	r0, SYS_open
USER	0x0104	ld	r3, filename
USER	0x0108	li	r4, O_RDWR
USER	0x010c	sc	
KERN !RI	0xcc00	mr	r9,r13
KERN !RI	0xcc04	mfspr	r13,SPRG1
KERN !RI	0xcc08	mfspr	r11,SRR0
KERN !RI	0xcc0c	mfspr	r12,SRR1
KERN !RI	0xcc10	mtmsr	RI
KERN	0xcc14	...	

Agenda

What is a system call?

Power ISA system call

Linux/powerpc code

Performance analysis

Problems and solutions

New POWER9 system call

Linux/powerpc code

- arch/powerpc/kernel/exceptions-64s.S
- arch/powerpc/kernel/entry_64.h
- Documentation/powerpc/syscall64-abi.txt

Linux/powerpc code – getppid execution trace

```
sc
mr      r2,r2
mr      r9,r13
mfspr   r13,304
mfsrr0  r11
mfsrr1  r12
li      r10,2
mtmsrd  r10,1
b       c00000000000b000
nop
nop
andi.   r10,r12,16384
mr      r10,r1
addi    r1,r1,-752
beq-    c00000000000b01c
ld      r1,2392(r13)
std     r10,0(r1)
std     r11,368(r1)
std     r12,376(r1)
std     r0,112(r1)
std     r10,120(r1)
beq     c00000000000b034
std     r2,128(r1)
std     r3,136(r1)
mfcr    r2
std     r4,144(r1)
std     r5,152(r1)
std     r6,160(r1)
std     r7,168(r1)
std     r8,176(r1)
li      r11,0
std     r11,184(r1)
std     r11,192(r1)
std     r11,200(r1)
std     r11,208(r1)
std     r11,408(r1)
std     r11,392(r1)
std     r9,216(r1)
mflr    r10
rldimi  r2,r11,28,35
li      r11,3073
std     r10,400(r1)
std     r11,432(r1)
std     r3,384(r1)
std     r2,416(r1)
ld      r2,8(r13)
addi    r9,r1,112
ld      r11,-32560(r2)
std     r11,-16(r9)
li      r11,2
ori     r11,r11,32768
mtmsrd  r11,1
li      r10,0
std     r10,424(r1)
rldicr  r11,r1,0,49
ld      r10,128(r11)
andi.   r11,r10,34433
bne     c00000000000b1ec
cmpldi  r0,387
bge-    c00000000000b22c
ld      r11,-32568(r2)
andi.   r10,r10,16
beq     c00000000000b0f4
rlwinm  r0,r0,4,0,27
ldx     r12,r11,r0
mtctr   r12
bctrl
addis   r2,r12,103
addi    r2,r2,-10944
mflr    r0
std     r0,16(r1)
stdu    r1,-32(r1)
ld      r9,2384(r13)
ld      r3,928(r9)
li      r5,0
li      r4,4
bl      c0000000000b91c0
cmpdi   r5,0
beq     c0000000000b9230
ld      r9,2384(r13)
ld      r9,1032(r9)
cmpdi   cr7,r9,0
beq
cr7,c0000000000b91c8
lwz     r10,4(r9)
addi    r10,r10,3
rldicr  r10,r10,4,59
add     r9,r9,r10
ld      r5,8(r9)
ld      r9,1032(r3)
cmpdi   cr7,r9,0
bne
cr7,c0000000000b91d4
cmpdi   cr7,r4,0
li      r9,1032
beq
cr7,c0000000000b91f4
cmplwi  cr7,r4,4
beq
cr7,c0000000000b91f0
ld      r3,976(r3)
ldx     r9,r3,r9
li      r3,0
cmpdi   cr7,r9,0
beqlr   cr7
lwz     r7,72(r5)
lwz     r8,4(r9)
cmplw   cr7,r8,r7
bltlr   cr7
rldicr  r7,r7,4,59
add     r9,r9,r7
ld      r10,56(r9)
cmpd    cr7,r5,r10
bnelr   cr7
lwa     r3,48(r9)
blr
nop
addi    r1,r1,32
ld      r0,16(r1)
mtlr    r0
blr
std     r3,456(r1)
rldicr  r12,r1,0,49
ld      r8,376(r1)
andi.   r10,r8,2
beq-    c00000000000b944
li      r11,0
mtmsrd  r11,1
ld      r9,128(r12)
li      r11,-4095
andi.   r0,r9,65511
bne-    c00000000000b234
andi.   r0,r8,8192
beq     c00000000000b140
andis.  r0,r8,512
bne     c00000000000b164
cmpld   r3,r11
ld      r5,416(r1)
bge-    c00000000000b1dc
ld      r7,368(r1)
nop
andi.   r6,r8,16384
ld      r4,400(r1)
beq-    c00000000000b1bc
mr      r6,r6
ld      r13,216(r1)
ld      r2,128(r1)
ld      r1,120(r1)
mtlr    r4
mtcr    r5
mtsrr0  r7
mtsrr1  r8
nop
nop
nop
rfid

getppid(2)
151 kernel instructions!
```


Linux/powerpc code – getppid execution trace

```
sc
mr      r2,r2
mr      r9,r13
mfspr   r13,304
mfsrr0  r11
mfsrr1  r12
li      r10,2
mtmsrd  r10,1
b       c00000000000b000
nop
nop
andi.   r10,r12,16384
mr      r10,r1
addi    r1,r1,-752
beq-    c00000000000b01c
ld      r1,2392(r13)
std     r10,0(r1)
std     r11,368(r1)
std     r12,376(r1)
std     r0,112(r1)
std     r10,120(r1)
beq     c00000000000b034
std     r2,128(r1)
std     r3,136(r1)
mfcr    r2
std     r4,144(r1)
std     r5,152(r1)
std     r6,160(r1)
std     r7,168(r1)
std     r8,176(r1)
li      r11,0
std     r11,184(r1)
std     r11,192(r1)
std     r11,200(r1)
```

```
std     r11,208(r1)
std     r11,408(r1)
std     r11,392(r1)
std     r9,216(r1)
mflr    r10
rldimi  r2,r11,28,35
li      r11,3073
std     r10,400(r1)
std     r11,432(r1)
std     r3,384(r1)
std     r2,416(r1)
ld      r2,8(r13)
addi    r9,r1,112
ld      r11,-32560(r2)
std     r11,-16(r9)
li      r11,2
ori     r11,r11,32768
mtmsrd  r11,1
li      r10,0
std     r10,424(r1)
rldicr  r11,r1,0,49
ld      r10,128(r11)
andi.   r11,r10,34433
bne     c00000000000b1ec
cmpldi  r0,387
bge-    c00000000000b22c
ld      r11,-32568(r2)
andi.   r10,r10,16
beq     c00000000000b0f4
rlwinm  r0,r0,4,0,27
ldx     r12,r11,r0
mtctr   r12
bctrl
```

```
kernel/sys.c:903

SYSCALL_DEFINE0(getppid)
{
        int pid;

        rcu_read_lock();
        pid = task_tgid_vnr(
                 rcu_dereference(current->real_parent));
        rcu_read_unlock();

        return pid;
}
```

```
std     r3,456(r1)
rldicr  r12,r1,0,49
ld      r8,376(r1)
andi.   r10,r8,2
beq-    c00000000000b944
li      r11,0
mtmsrd  r11,1
ld      r9,128(r12)
li      r11,-4095
andi.   r0,r9,65511
bne-    c00000000000b234
andi.   r0,r8,8192
beq     c00000000000b140
andis.  r0,r8,512
bne     c00000000000b164
cmpld   r3,r11
ld      r5,416(r1)
bge-    c00000000000b1dc
ld      r7,368(r1)
nop
andi.   r6,r8,16384
ld      r4,400(r1)
beq-    c00000000000b1bc
mr      r6,r6
ld      r13,216(r1)
ld      r2,128(r1)
ld      r1,120(r1)
mtlr    r4
mtcr    r5
mtsrr0  r7
mtsrr1  r8
nop
nop
nop
rfid
```

Linux/powerpc code – getppid entry simplified

```
sc
mr      r9,r13
mfspr   r13,304
mfsrr0  r11
mfsrr1  r12
mtmsr   MSR[RI]=1
mr      r10,r1
ld      r1,2392(r13)
std     r10,0(r1)
std     r11,368(r1)
std     r12,376(r1)
std     r0,112(r1)
std     r10,120(r1)
std     r2,128(r1)
std     r3,136(r1)
std     r4,144(r1)
std     r5,152(r1)
std     r6,160(r1)
std     r7,168(r1)
std     r8,176(r1)
li      r11,0
std     r11,184(r1)
std     r11,192(r1)
std     r11,200(r1)
std     r11,208(r1)
std     r11,408(r1)
std     r11,392(r1)
std     r9,216(r1)
mfcr    r2
rldimi  r2,r11,28,35
std     r2,416(r1)
mflr    r10
std     r10,400(r1)
li      r11,3073
std     r11,432(r1)
ld      r2,8(r13)
addi    r9,r1,112
ld      r11,-32560(r2)
std     r11,-16(r9)
mtmsr   MSR[EE]=1
cmpldi  r0,387
bge-    c00000000000b22c
ld      r11,-32568(r2)
rlwinm  r0,r0,4,0,27
ldx     r12,r11,r0
mtctr   r12
bctrl
```

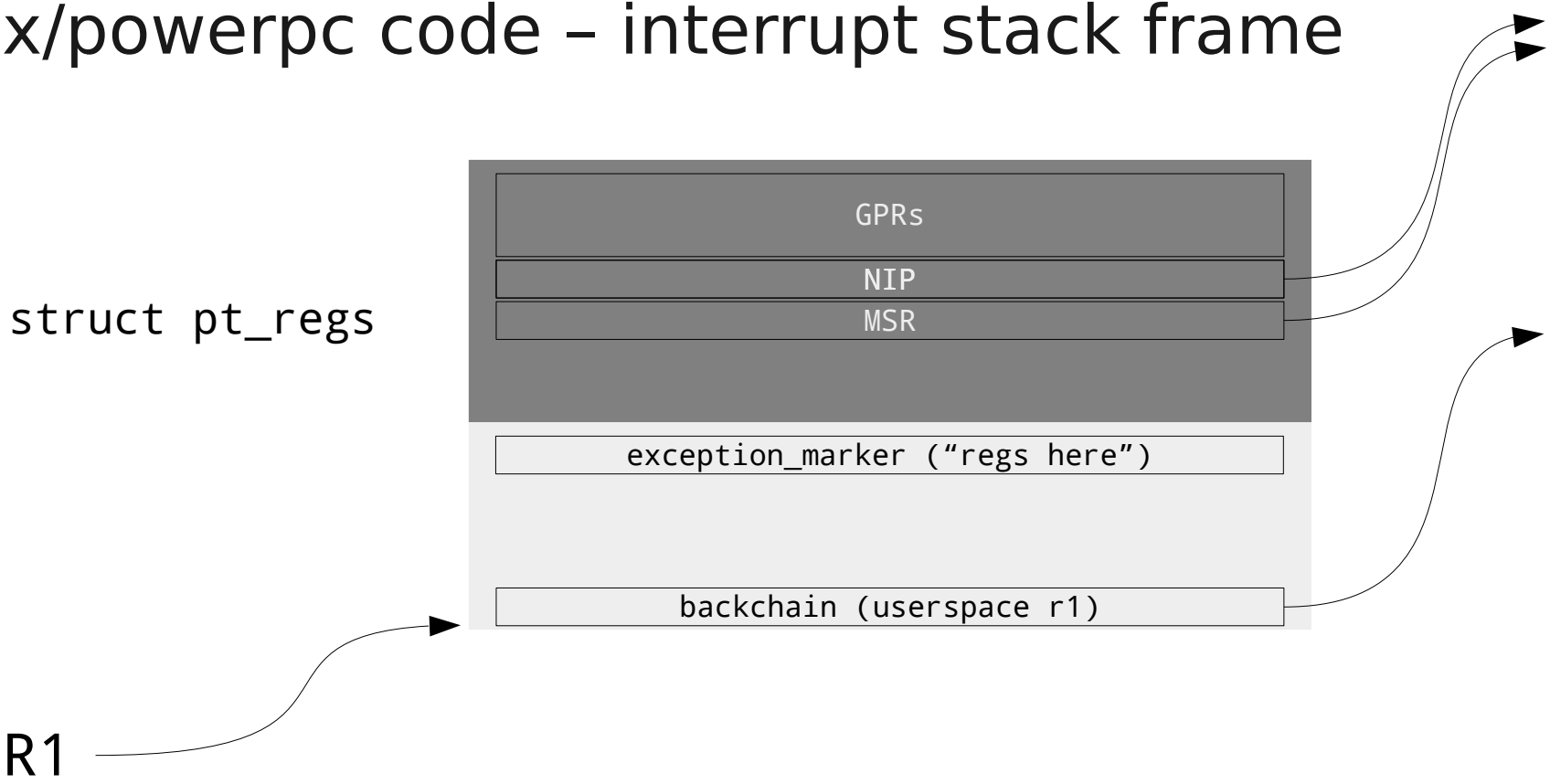

Linux/powerpc code – getppid entry simplified

```
sc
mr      r9,r13
mfspr   r13,304
mfsrr0  r11
mfsrr1  r12
mtmsr   MSR[RI]=1
mr      r10,r1
ld      r1,2392(r13)
std     r10,0(r1)
std     r11,368(r1)
std     r12,376(r1)
[ Save GPRs 0-8, 13, CR, LR ]
li      r11,3073
std     r11,432(r1)
ld      r2,8(r13)
addi    r9,r1,112
ld      r11,-32560(r2)
std     r11,-16(r9)
mtmsr   MSR[EE]=1
cmpldi  r0,387
bge-    c00000000000b22c
ld      r11,-32568(r2)
rlwinm  r0,r0,4,0,27
ldx     r12,r11,r0
mtctr   r12
bctrl
```

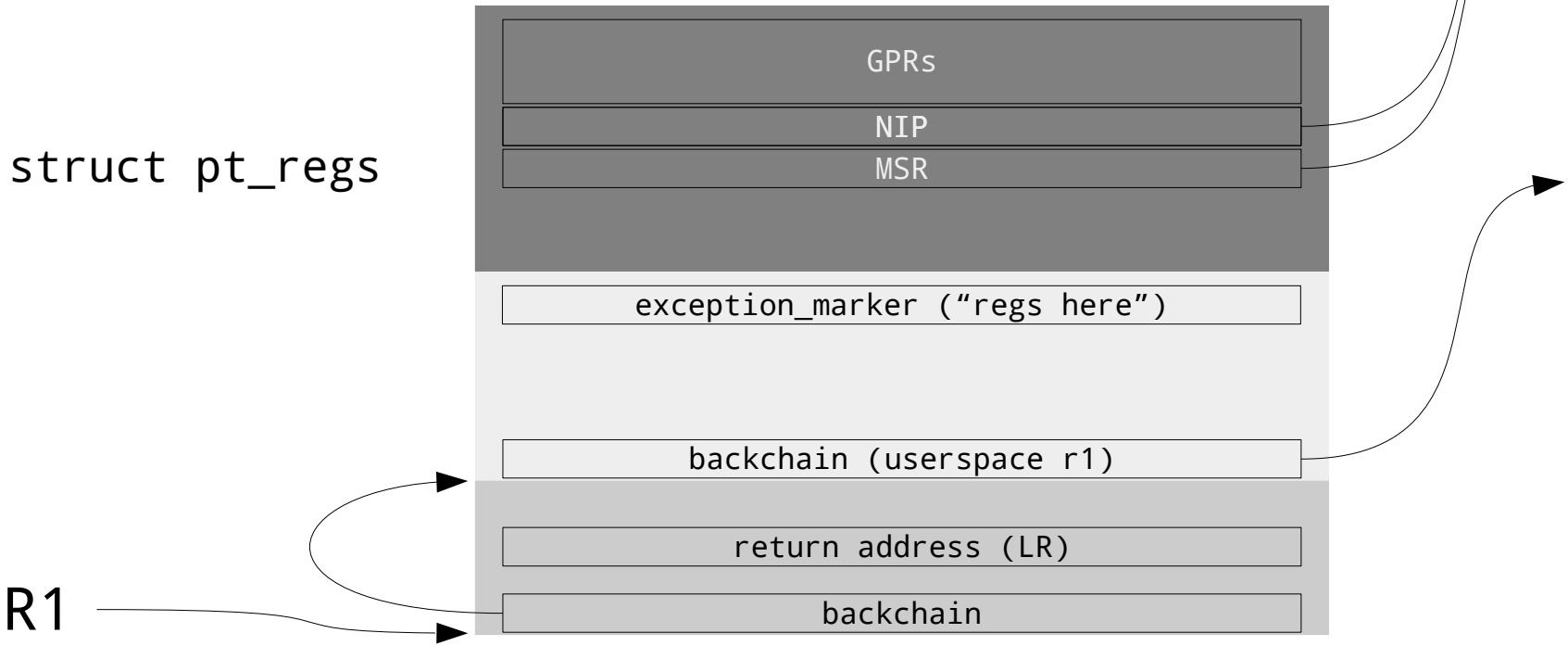

Linux/powerpc code – getppid entry simplified

```
mr      r9,r13                          # Put user r13 in r9
mfspr   r13,SPRN_SPRG_PACA              # PACA pointer to r13
mfspr   r11,SPRN_SRR0                   # Get caller NIA
mfspr   r12,SPRN_SRR1                   # Get caller MSR
mtmsr   MSR[RI]=1                       # Enable MSR RI bit
mr      r10,r1                          # Put user r1 in r10
ld      r1,PACA_KSAVE(r13)              # Load the kernel stack to r1
std     r10,0(r1)                       # Store user r1 as backchain
std     r11,_NIP(r1)                    # Save NIA and...
std     r12,_MSR(r1)                    # ...MSR on stack
[ Save GPRs 0-8, 13, CR, LR ]           # Save args and non-volatile regs
li      r11,SYSCALL_TRAP
std     r11,_TRAP(r1)                   # Store trap type to stack
ld      r2,PACA_TOC(r13)                # Load “TOC” pointer to r2
addi    r9,r1,STACK_FRAME_OVERHEAD      # r9 is “struct pt_regs *regs”
ld      r11,exception_marker@toc(r2)    # Load exception_marker global
std     r11,-16(r9)                     # Put a marker in the stack
mtmsr   MSR[EE]=1                       # Enable interrupts (MSR[EE])
cmpldi  r0,NR_syscalls                  # Test for r0 >= NR_syscalls
bge-    .Lsyscall_enosys
ld      r11,SYS_CALL_TABLE@toc(r2)      # Load SYS_CALL_TABLE global
slwi    r0,r0,4                         # Each table entry is 16 bytes
ldx     r12,r11,r0                      # Load SYS_CALL_TABLE[r0]
mtctr   r12                             # Indirect call
bctrl
```

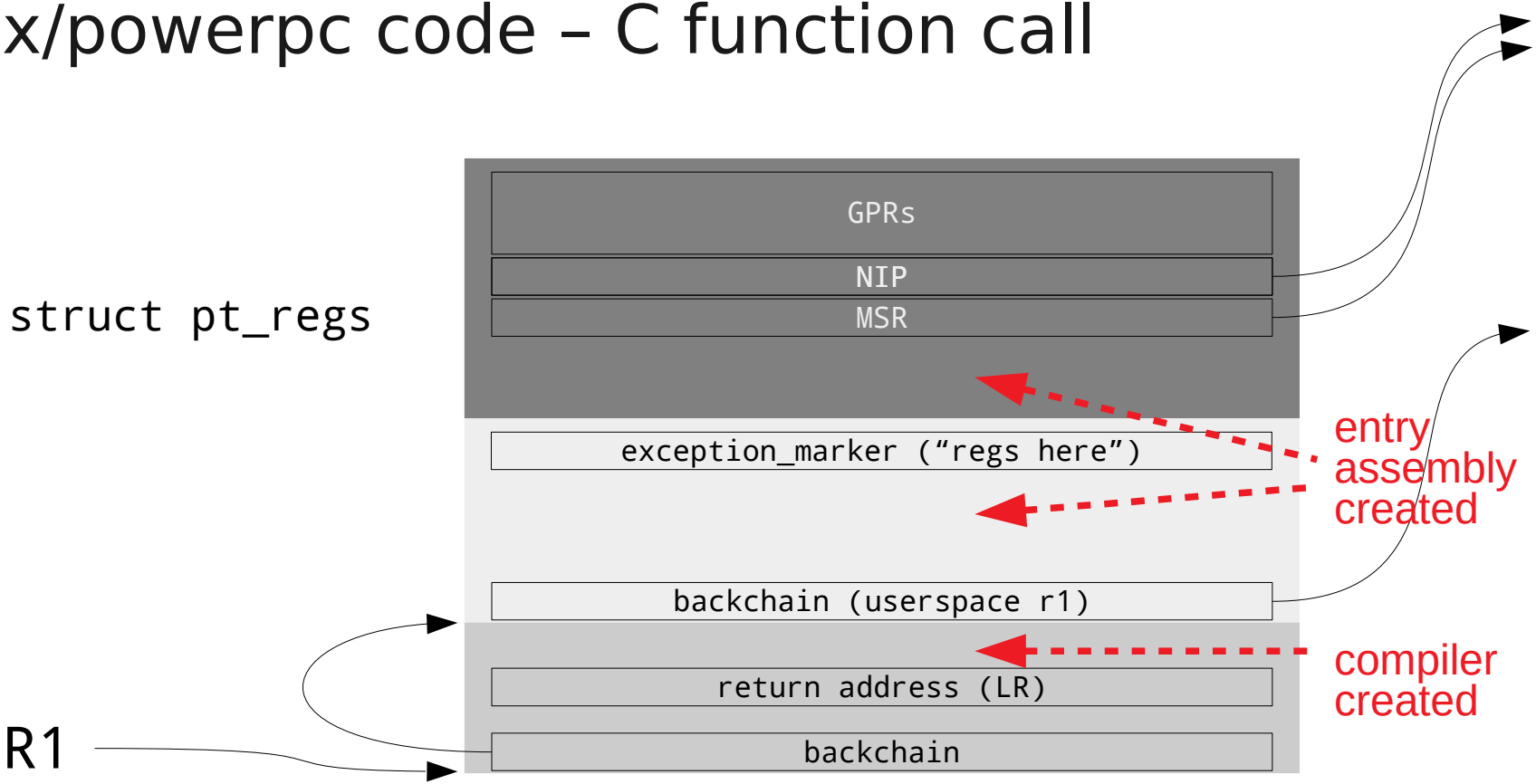

Linux/powerpc code – interrupt stack frame



Linux/powerpc code – C function call



Linux/powerpc code – C function call



Agenda

What is a system call?

Power ISA system call

Linux/powerpc code

Performance analysis

Problems and solutions

New POWER9 system call

Performance analysis

- Workload to test and optimise **{ while (true) getppid(); }**.
- Tools:
  - perf.
  - Execution traces (IBM SDK systemsim, valgrind itrace).
    - Your own analysis scripts, models, visualisation, etc.
    - Performance simulation / analysis tools (sim_ppc, pipestat).
- https://openpowerfoundation.org/wp-content/uploads/2018/04/Paul-Clarke.pdf
- https://github.com/antonblanchard/qtrace-tools

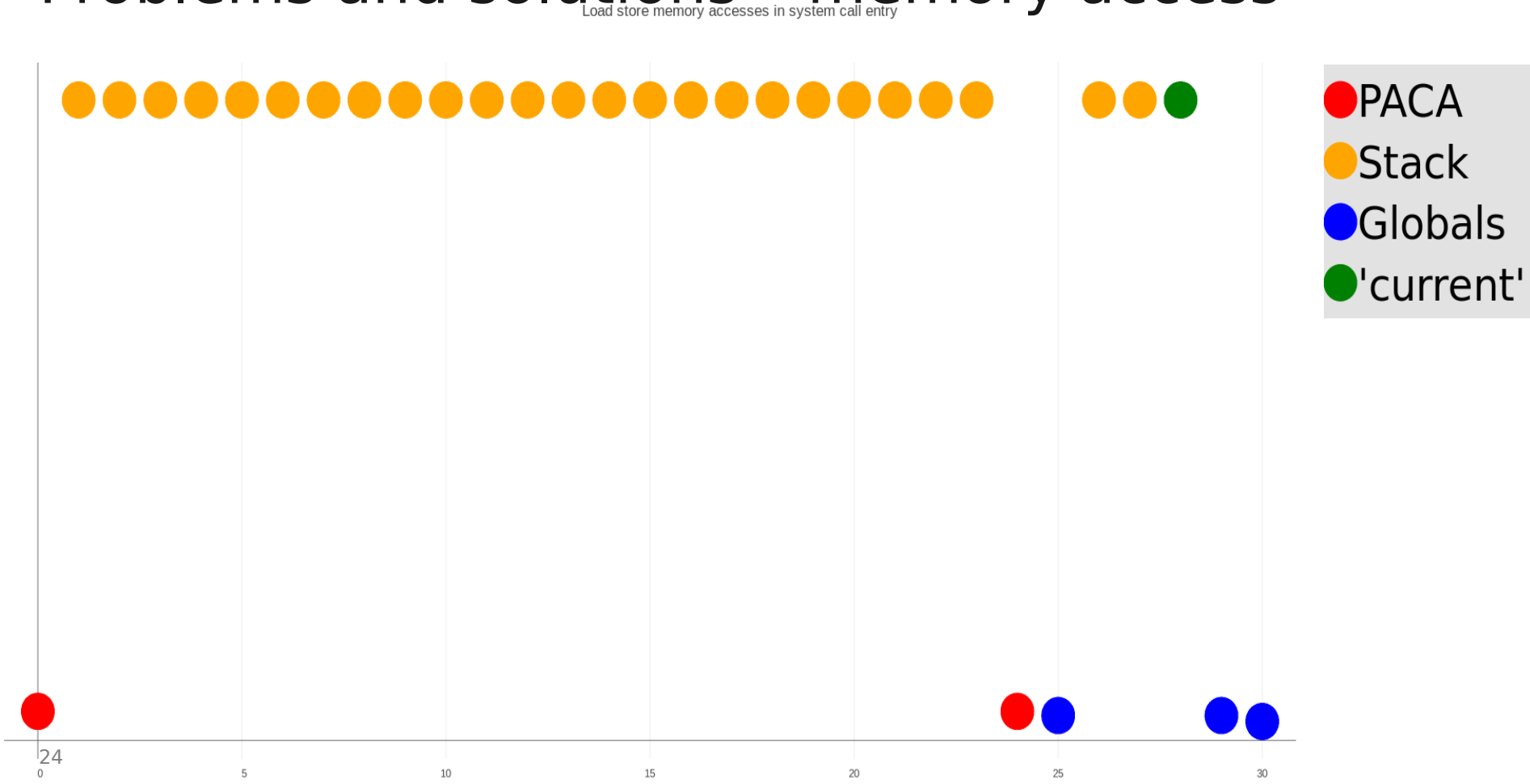
Performance analysis - perf

```
 4.94% | mr      r9,r13                          # Put r13 in r9
       | mfspr   r13,SPRN_SPRG_PACA              # PACA pointer to r13
       | mfspr   r11,SPRN_SRR0                  # Get caller NIA
       | mfspr   r12,SPRN_SRR1                  # Get caller MSR
       | mtmsr   MSR[RI]=1                      # Enable MSR RI bit
       | mr      r10,r1                          # Put r1 in r10
       | ld      r1,PACA_KSAVE(r13)              # Load the kernel stack to r1
       | std     r10,0(r1)                       # Store previous r1 as backchain
       | std     r11,_NIP(r1)                    # Save NIA and
       | std     r12,_MSR(r1)                    # MSR in stack
 0.01% | [ Save GPRs 0-8, 13, CR, LR ]
 0.37% | li      r11,SYSCALL_TRAP_NR
       | std     r11,_TRAP(r1)                   # Store trap type
       | ld      r2,PACA_TOC(r13)                # Load “TOC” pointer to r2
       | addi    r9,r1,STACK_FRAME_OVERHEAD      # r9 is “struct pt_regs *regs”
       | ld      r11,exception_marker@toc(r2)    # Load exception_marker global
       | std     r11,-16(r9)                     # Put a marker in the stack
 6.38% | mtmsr   MSR[EE]=1                       # Enable interrupts (MSR[EE])
 8.66% | cmpldi  r0,NR_syscalls                  # Test for r0 >= NR_syscalls
 0.35% | bge-    .Lsyscall_enosys
       | ld      r11,SYS_CALL_TABLE@toc(r2)      # Load SYS_CALL_TABLE global
       | slwi    r0,r0,4                         # Each table entry is 16 bytes
       | ldx     r12,r11,r0                      # Load SYS_CALL_TABLE[r0]
 0.17% | mtctr   r12                             # Indirect call
 0.15% | bctrl
```

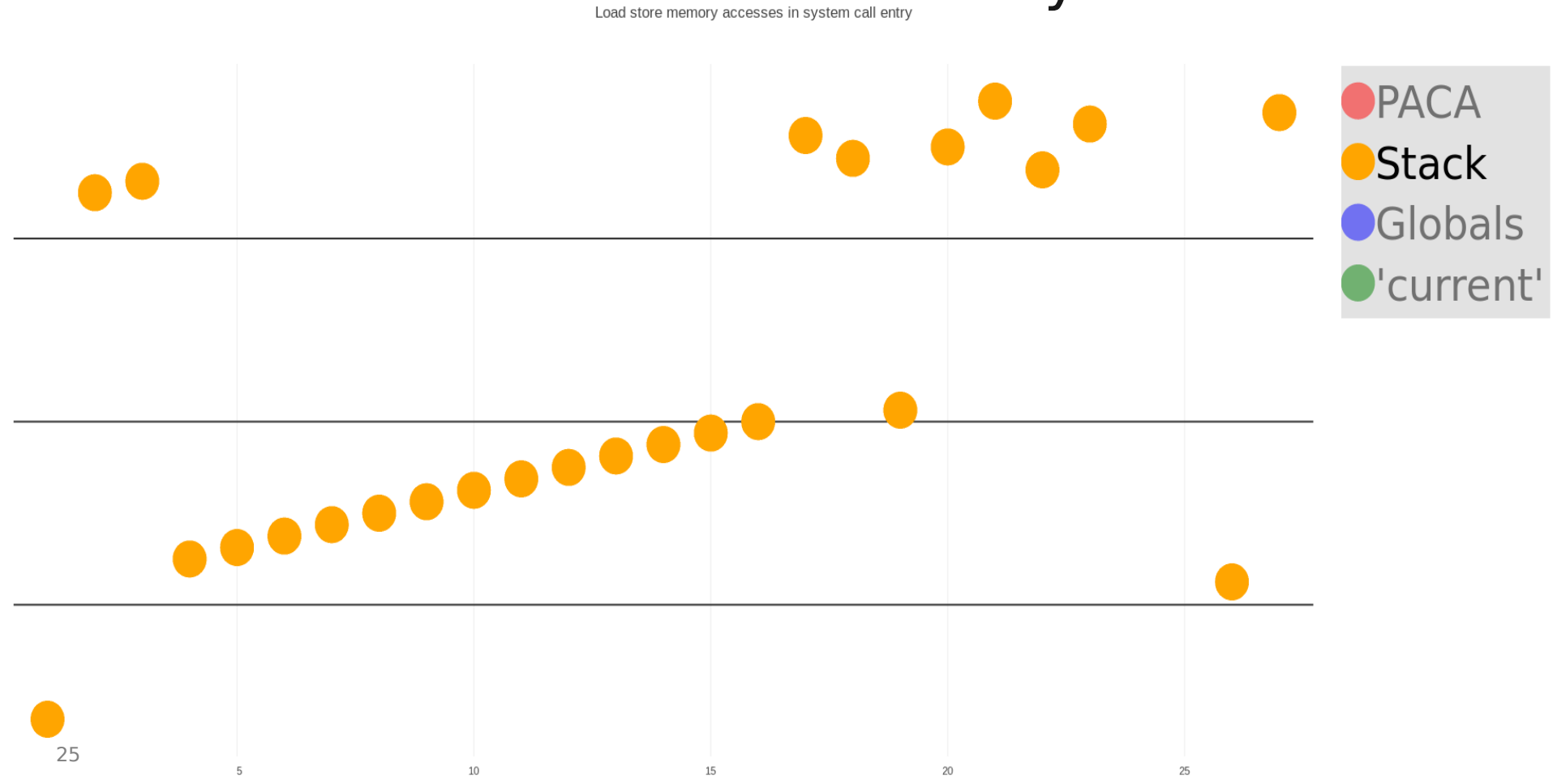

Problems and solutions – pain points

- General problems:
  - Locality of reference, cache footprint of data/instructions (cache misses).
  - Control dependencies if they are unpredictable (branch mispredicts).
  - Data dependencies.
  - ~~Locks, barriers, atomics~~ (none in basic syscall path).
- Problems for system and low level code:
  - Serialising operations.
  - Changing execution context, flushes.
  - High latency operations, microcode, etc.

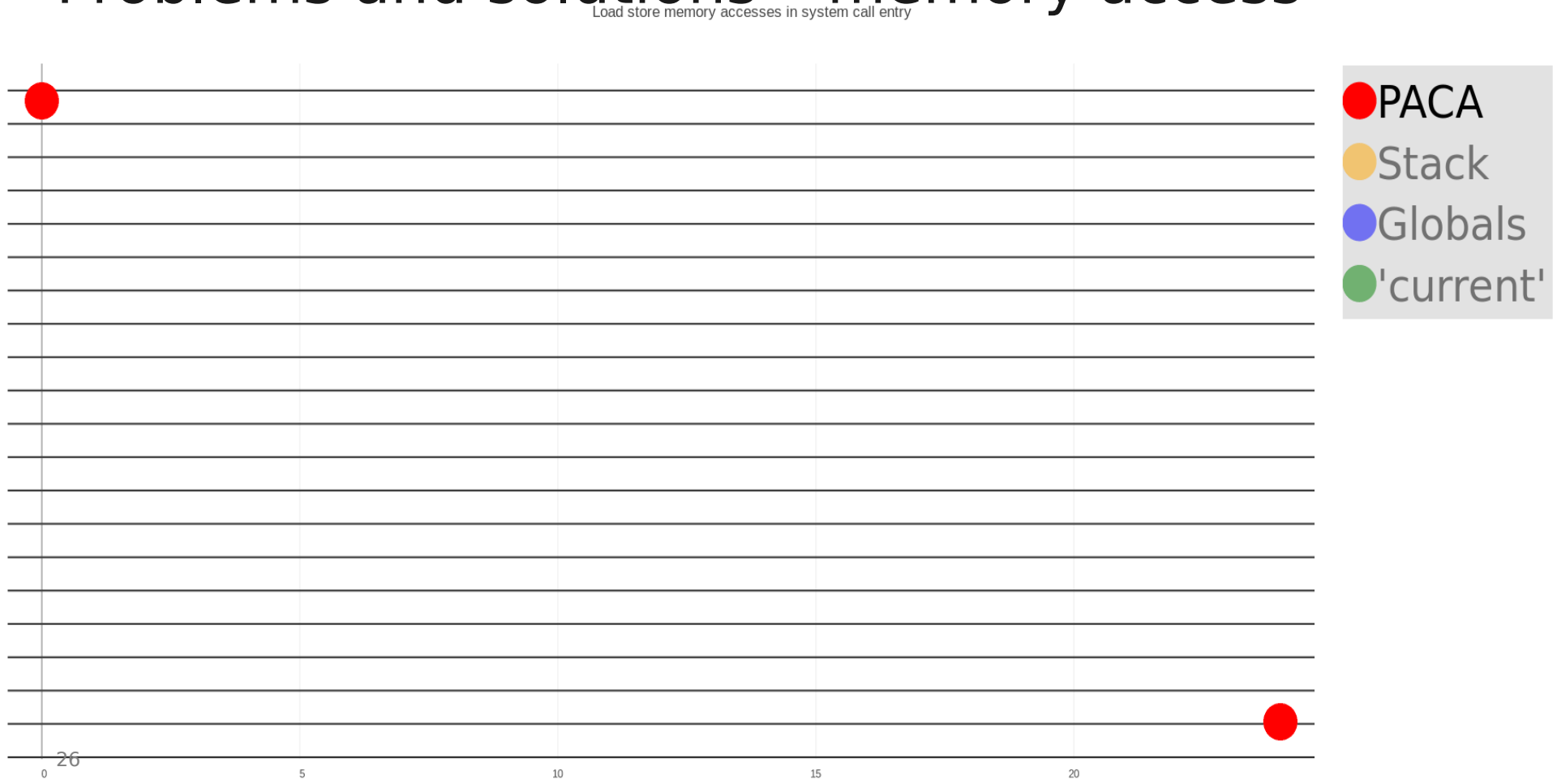
Problems and solutions – memory access



Problems and solutions – memory access



Problems and solutions – memory access



Problems and solutions – memory access

Move data into same cacheline

```
mr      r9,r13                          # Put user r13 in r9
mfspr   r13,SPRN_SPRG_PACA              # PACA pointer to r13
mfspr   r11,SPRN_SRR0                   # Get caller NIA
mfspr   r12,SPRN_SRR1                   # Get caller MSR
mtmsr   MSR[RI]=1                       # Enable MSR RI bit
mr      r10,r1                          # Put user r1 in r10
ld      r1,PACA_KSAVE(r13)              # Load the kernel stack to r1
std     r10,0(r1)                       # Store user r1 as backchain
std     r11,_NIP(r1)                    # Save NIA and...
std     r12,_MSR(r1)                    # ...MSR on stack
[ Save GPRs 0-8, 13, CR, LR ]           # Save args and nonvolatile regs
li      r11,SYSCALL_TRAP
std     r11,_TRAP(r1)                   # Store trap type to stack
ld      r2,PACA_TOC(r13)                # Load "TOC" pointer to r2
addi    r9,r1,STACK_FRAME_OVERHEAD      # r9 is "struct pt_regs *regs"
ld      r11,exception_marker@toc(r2)    # Load exception_marker global
std     r11,-16(r9)                     # Put a marker in the stack
mtmsr   MSR[EE]=1                       # Enable interrupts (MSR[EE])
cmpldi  r0,NR_syscalls                  # Test for r0 >= NR_syscalls
bge-    .Lsyscall_enosys
ld      r11,SYS_CALL_TABLE@toc(r2)      # Load SYS_CALL_TABLE global
slwi    r0,r0,4                         # Each table entry is 16 bytes
ldx     r12,r11,r0                      # Load SYS_CALL_TABLE[r0]
mtctr   r12                             # Indirect call
bctrl
```


Problems and solutions – branch prediction

- Return predictor: important to get right when writing assembly!
- Mispredicted branch can not be corrected until dependencies are met and it is executed. So hard to predict branches require timely inputs. Optimizing mispredicted branch performance can be useful if no option to avoid them.

Problems and solutions – return prediction

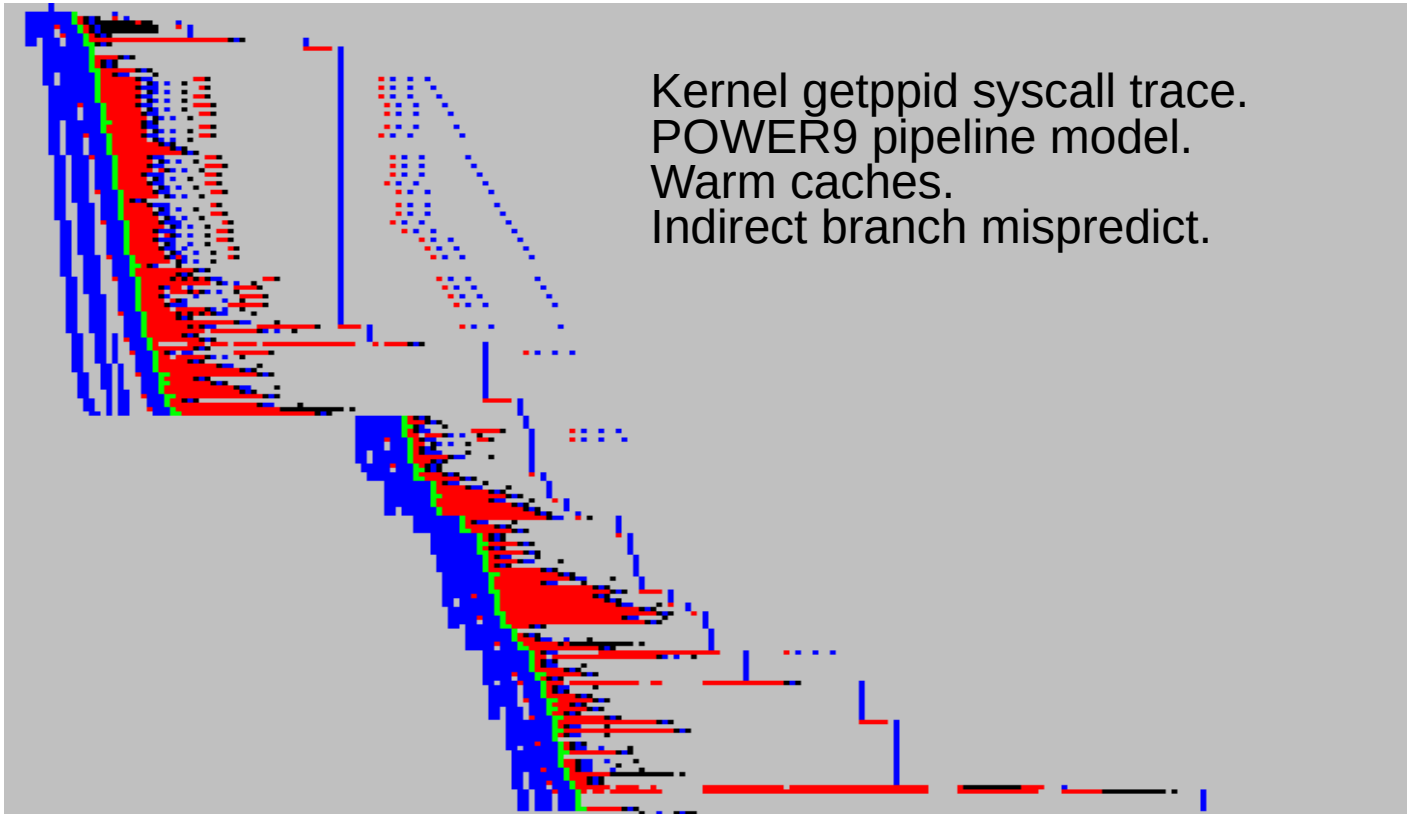
```
commit 6a404806dfceba9b154015705a9e647fa7749fd8
Author: Michael Neuling <mikey@neuling.org>
Date:   Wed Feb 27 10:45:52 2013 +0000

    powerpc: Avoid link stack corruption in MMU on syscall entry path

    Currently we use the link register to branch up high in the early MMU on
    syscall entry path.  Unfortunately, this trashes the link stack as the
    address we are going to is not associated with the earlier mflr.

    This patch simply converts us to used the count register (volatile over
    syscalls anyway) instead.  This is much better at predicting in this
    scenario and doesn't trash link stack causing a bunch of additional
    branch mispredicts later.  Benchmarking this on POWER8 saves a bunch of
    cycles on Anton's null syscall benchmark here:
       http://ozlabs.org/~anton/junkcode/null_syscall.c

    Signed-off-by: Michael Neuling <mikey@neuling.org>
    Signed-off-by: Benjamin Herrenschmidt <benh@kernel.crashing.org>
```


Problems and solutions – return prediction

- getppid ~50(?) cycles improvement

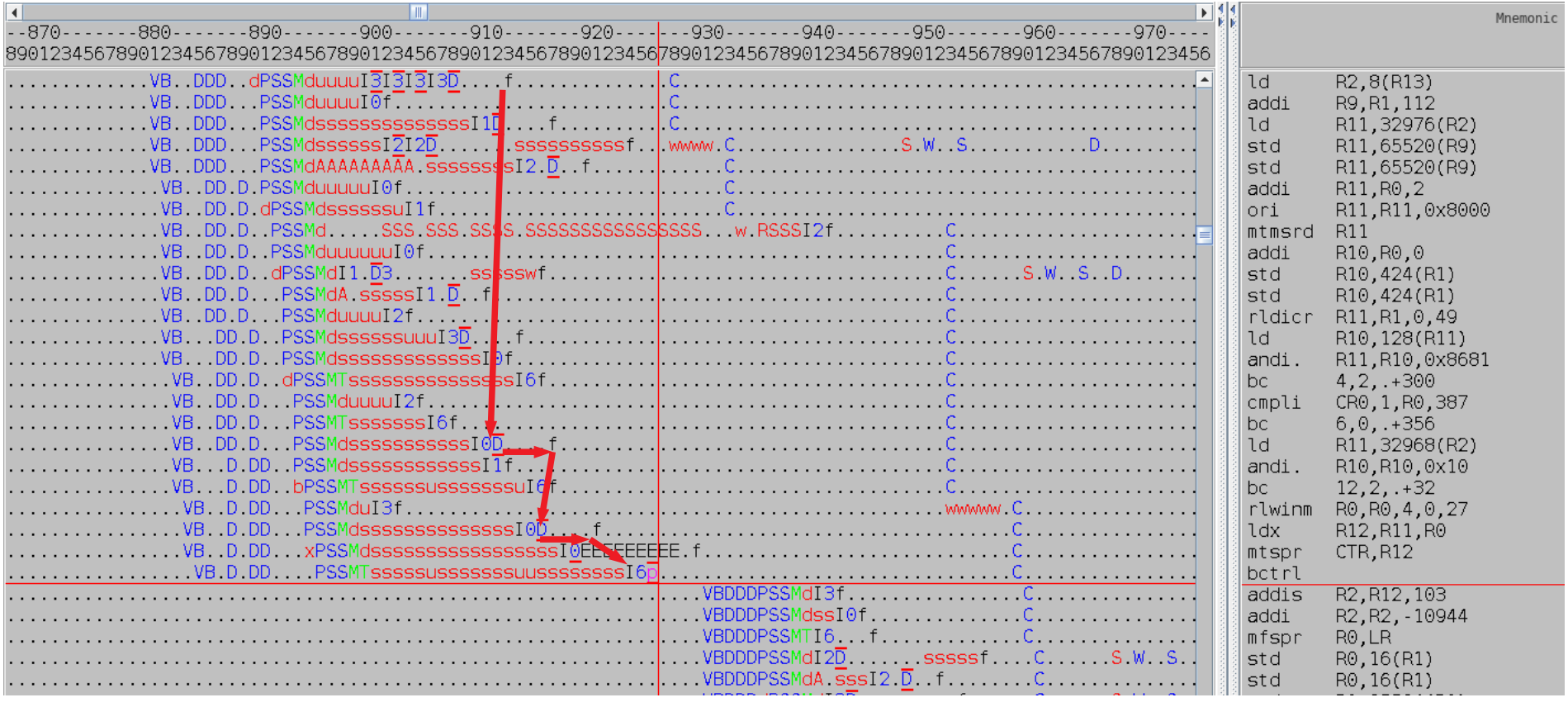
```
diff --git a/arch/powerpc/kernel/exceptions-64s.S
b/arch/powerpc/kernel/exceptions-64s.S
index a8a5361fb70c..87ef8f5ee5bc 100644
--- a/arch/powerpc/kernel/exceptions-64s.S
+++ b/arch/powerpc/kernel/exceptions-64s.S
@@ -74,13 +74,13 @@ END_FTR_SECTION_IFSET(CPU_FTR_REAL_LE)
         \
        mflr    r10 ;                                           \
        ld      r12,PACAKBASE(r13) ;                            \
        LOAD_HANDLER(r12, system_call_entry_direct) ;           \
-       mtlr    r12 ;                                           \
+       mtctr   r12 ;                                           \
        mfspr   r12,SPRN_SRR1 ;                                 \
        /* Re-use of r13... No spare regs to do this */ \
        li      r13,MSR_RI ;                                    \
        mtmsrd  r13,1 ;                                         \
        GET_PACA(r13) ; /* get r13 back */                      \
-       blr ;
+       bctr ;
 #else
        /* We can branch directly */
 #define SYSCALL_PSERIES_2_DIRECT                               \
```


Problems and solutions – simulator (sim_ppc)



Problems and solutions – dependent operations

- load R2 (TOC) -> load syscall table -> load function pointer -> branch indirect



Problems and solutions – dependent operations

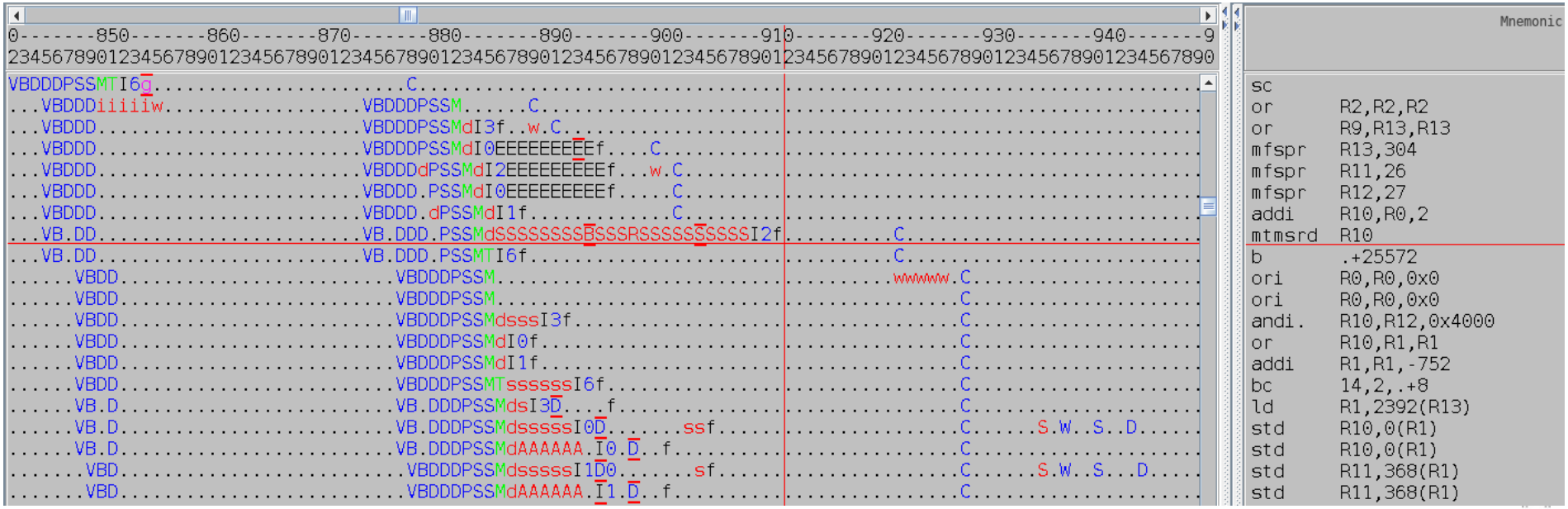
Duplicate in the PACA.

Move this load earlier.

```
mr      r9,r13                          # Put user r13 in r9
mfspr   r13,SPRN_SPRG_PACA              # PACA pointer to r13
mfspr   r11,SPRN_SRR0                   # Get caller NIA
mfspr   r12,SPRN_SRR1                   # Get caller MSR
mtmsr   MSR[RI]=1                       # Enable MSR RI bit
mr      r10,r1                          # Put user r1 in r10
ld      r1,PACA_KSAVE(r13)              # Load the kernel stack to r1
std     r10,0(r1)                       # Store user r1 as backchain
std     r11,_NIP(r1)                    # Save NIA and...
std     r12,_MSR(r1)                    # ...MSR on stack
[ Save GPRs 0-8, 13, CR, LR ]           # Save args and nonvolatile regs
li      r11,SYSCALL_TRAP
std     r11,_TRAP(r1)                   # Store trap type to stack
ld      r2,PACA_TOC(r13)                # Load "TOC" pointer to r2
addi    r9,r1,STACK_FRAME_OVERHEAD      # r9 is "struct pt_regs *regs"
ld      r11,exception_marker@toc(r2)    # Load exception_marker global
std     r11,-16(r9)                     # Put a marker in the stack
mtmsr   MSR[EE]=1                       # Enable interrupts (MSR[EE])
cmpldi  r0,NR_syscalls                  # Test for r0 >= NR_syscalls
bge-    .Lsyscall_enosys
ld      r11,SYS_CALL_TABLE@toc(r2)      # Load SYS_CALL_TABLE global
slwi    r0,r0,4                         # Each table entry is 16 bytes
ldx     r12,r11,r0                      # Load SYS_CALL_TABLE[r0]
mtctr   r12                             # Indirect call
bctrl
```


Problems and solutions – changing MSR at entry

- **mtmsrd** to change MSR RI does not issue until after previous instructions complete. Then it is a 12 cycle latency.
- Pushes out completion, which adds to latency of subsequent serialising instructions, ties up out of order resources, delays store queue draining, etc.



Problems and solutions – changing MSR at entry

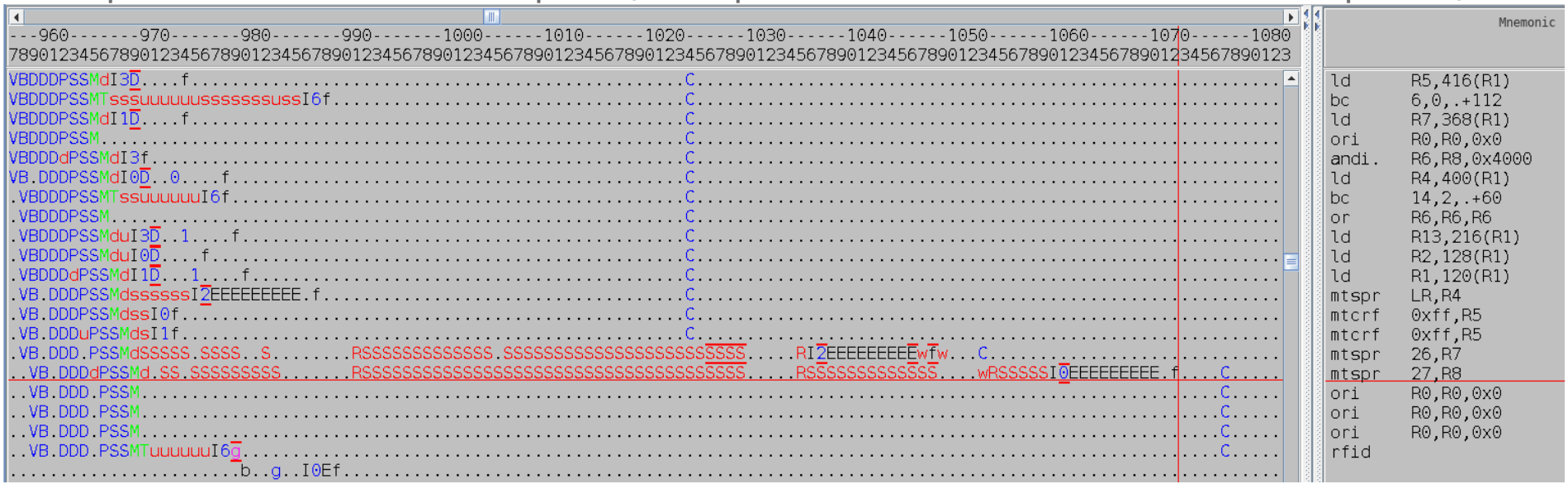
Remove this mtmsr

Enable EE and RI with one mtmsr

```
mr      r9,r13                          # Put user r13 in r9
mfspr   r13,SPRN_SPRG_PACA              # PACA pointer to r13
mfspr   r11,SPRN_SRR0                   # Get caller NIA
mfspr   r12,SPRN_SRR1                   # Get caller MSR
mtmsr   MSR[RI]=1                       # Enable MSR RI bit
mr      r10,r1                          # Put user r1 in r10
ld      r1,PACA_KSAVE(r13)              # Load the kernel stack to r1
std     r10,0(r1)                       # Store user r1 as backchain
std     r11,_NIP(r1)                    # Save NIA and...
std     r12,_MSR(r1)                    # ...MSR on stack
[ Save GPRs 0-8, 13, CR, LR ]           # Save args and nonvolatile regs
li      r11,SYSCALL_TRAP
std     r11,_TRAP(r1)                   # Store trap type to stack
ld      r2,PACA_TOC(r13)                # Load "TOC" pointer to r2
addi    r9,r1,STACK_FRAME_OVERHEAD      # r9 is "struct pt_regs *regs"
ld      r11,exception_marker@toc(r2)    # Load exception_marker global
std     r11,-16(r9)                     # Put a marker in the stack
mtmsr   MSR[EE]=1                       # Enable interrupts (MSR[EE])
cmpldi  r0,NR_syscalls                  # Test for r0 >= NR_syscalls
bge-    .Lsyscall_enosys
ld      r11,SYS_CALL_TABLE@toc(r2)      # Load SYS_CALL_TABLE global
slwi    r0,r0,4                         # Each table entry is 16 bytes
ldx     r12,r11,r0                      # Load SYS_CALL_TABLE[r0]
mtctr   r12                             # Indirect call
bctrl
```


Problems and solutions – mtspr SRR0/1 at exit

- **mtspr** to set SRR0 and SRR1 for **rfid** (return from interrupt).
- These instructions are serialised and non-pipelined, long latency.
- Operations on the critical path (userspace insns can’t execute until completed).

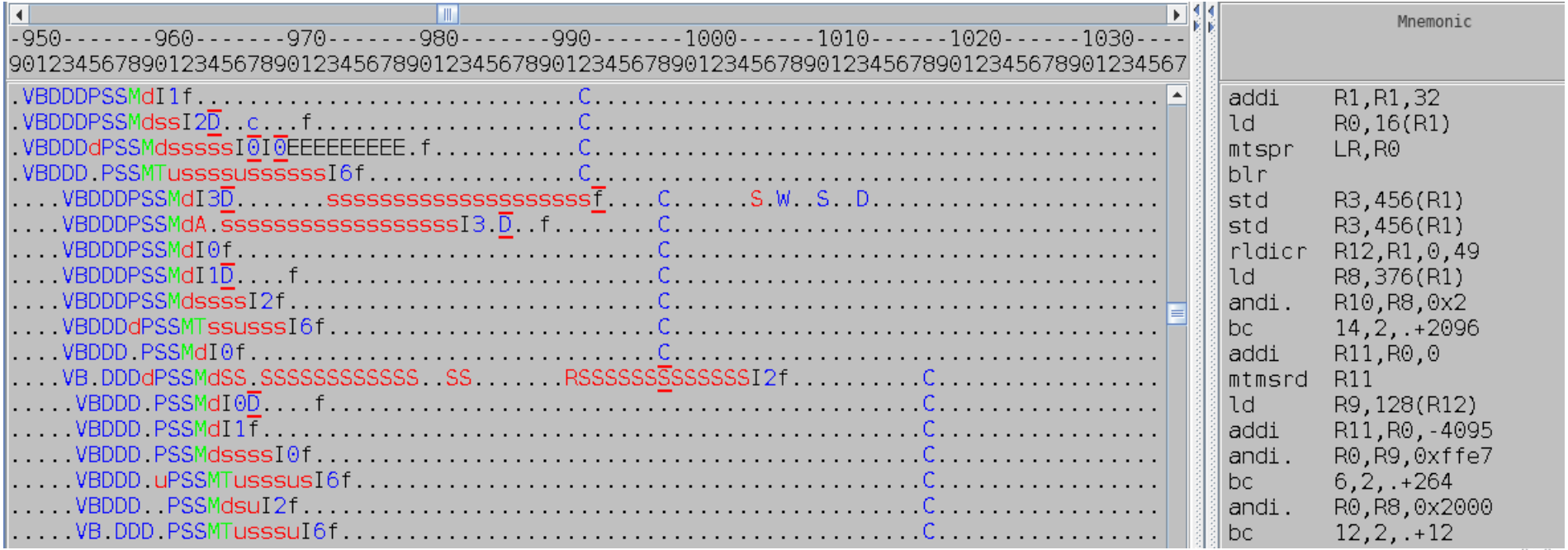


Problems and solutions – mtspr SRR0/1 at exit

- SRR0/1 will not change unless a new interrupt comes in, or the kernel sets them.
- **sc** sets up SRR0/1 values for **rfid** return.
- So maintain a flag that says SRRs are still valid for our **rfid**.
- Test that flag after clearing MSR[RI] to avoid re-entrancy problem.
- Avoid mtspr SRR0/1 entirely.

Problems and solutions – changing MSR at exit

- MSR is changed to disable interrupts and RI at exit.
- This is done so we can ensure there is no work left to do before exit.



Problems and solutions – changing MSR at exit

- Don’t change the MSR, interrupts and RI remain set.
- If the exit region is interrupted, have the interrupt return to a “fixup” handler instead.
- The fixup handler will double check to ensure no work remains, then goes on to exit, including setting SRR0/1.

Problems and solutions – summary

- Reduction in cache line footprint.
- Fewer dependent loads, loads earlier on critical branch mispredict path.
- 2 fewer **mtmsr**, 2 fewer **mtspr**.
- 482 -> 346 cycles per getppid (39% improvement).
- Most improvements can also apply to other interrupts.

Agenda

What is a system call?

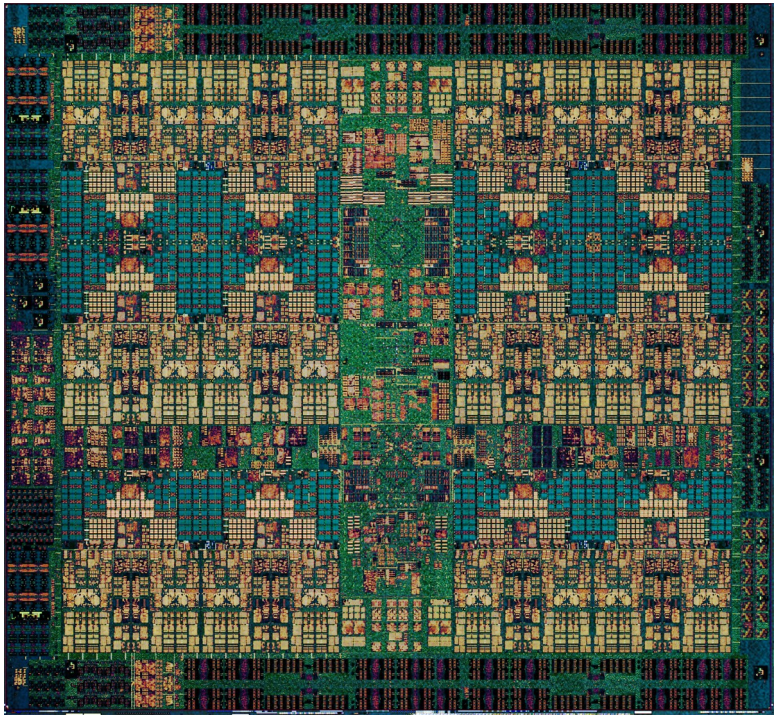
Power ISA system call

Linux/powerpc code

Performance analysis

Problems and solutions

New POWER9 system call



New POWER9 system call

- **scv** instruction (system call vectored):
  - Next instruction address will be the system call handler address (0xc000000000003000 + 0x20 * lev, 0 <= lev < 128).
  - Next instruction MSR will be privileged, interrupts **on**, RI bit **on**.
  - **LR** and **CTR** branch registers contain next sequential address and current MSR.
- **rfcsv** instruction (return from system call vectored):
  - Next instruction address and MSR is loaded from **LR** and **CTR**.

New POWER9 system call

- 128 entry points.
- Does not use SRR0, SRR1.
- No unrecoverable window, does not clear RI.
- Not shared with hypercalls.
- Does not disable interrupts.
- Can define a new user syscall ABI. E.g., provide more scratch registers to kernel.
- Can specify that kernel syscalls are disallowed, 32-bit syscalls, transactions must not be active, etc., - remove “cruft”.

New POWER9 system call

- Kernel entry with interrupts enabled. This is the main difficulty.
- When an interrupt comes in, must check if it came from kernel or user.
  - If from user, begin with the bottom of the kernel stack for r1.
  - If from kernel, take existing r1 and allocate a new interrupt frame.
  - This works because kernel entry does not enable interrupts until stack frame is set up (NMI use different stack).
- Solution: consider interrupts soft disabled if they came from interrupt entry code. Soft disabled handler does not touch kernel stack or clobber registers.
- Soft disable is an existing Linux/powerpc concept where interrupts can be disabled in software by storing a flag. The hardware interrupt happens, but a disabled interrupt is recorded and later “replayed” when soft enabled.

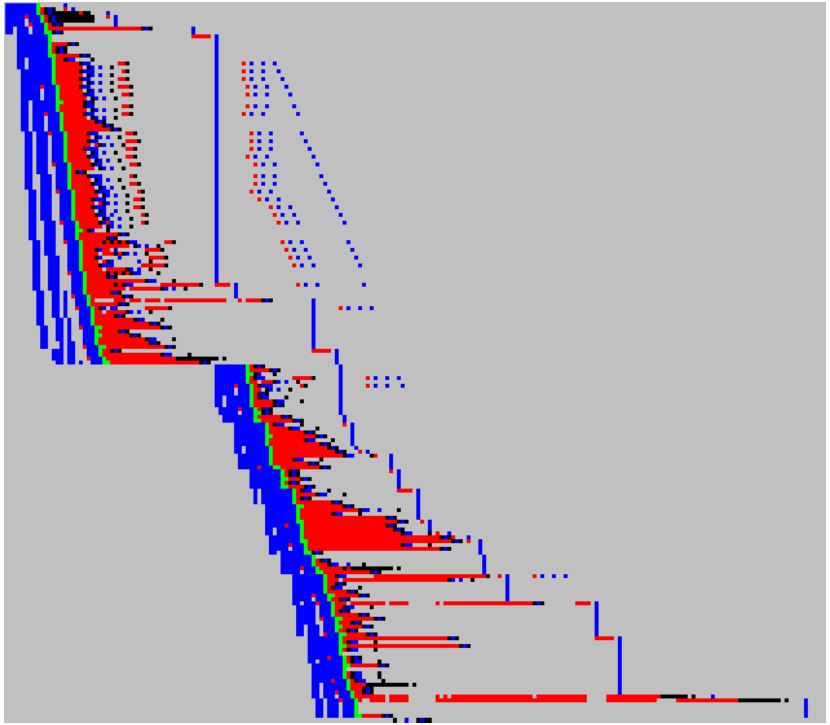
New POWER9 system call - results

- Small improvement in micro-benchmark after previous system call speedups.
- Larger benefit in real world from avoiding SRRs, no !RI window, removing cruft, not sharing system call handler with hypercall, new ABIs, etc.
- Code: Prototype only, proof of concept and performance measurement.
- Not yet completed.

End

- Thank you.

Problems and solutions – summary



Problems and solutions – summary