GDC 2012 March 5-9

Runtime CPU Performance Spike Detection using Manual and Compiler Automated Instrumentation

Adisak Pochanayon : adisak@wbgames.com
Principal Software Engineer, Mortal Kombat Team
Netherrealm Studios (WB Games Chicago)

This is a handout for source details that are would be impossible to read off a slide. Also, it should prove handy for later reference.

PC (32-bit X86) _penter() and _pexit()
(only one underscore precedes function names)

```
extern "C" void __declspec(naked) _cdecl _penter( void )
{
	_asm
	{
		push eax
		push ebx
		push ecx
		push edx
		push ebp
		push edi
		push esi
	}

	if(0==tls_PET_bIsInProcessing)
	{
		tls_PET_bIsInProcessing=true;
		_internal_PET_LogEntry(0);
		tls_PET_bIsInProcessing=false;
	}

	_asm
	{
		pop esi
		pop edi
		pop ebp
		pop edx
		pop ecx
		pop ebx
		pop eax
		ret
	}
}
```

NOTE: _pexit() function is identical except for call to handling routine is “LogExit” instead of “LogEntry”.

XBOX 360 (PowerPC assembler) __penter() and __pexit()
(two underscore precede function names)

NOTE: again, two functions nearly identical except for call to handling routine.

```
void __declspec(naked) _cdecl __penter( void )
{
  __asm
  {
      // Tiny Prolog
      //  - Set link register (r12) & return address (two steps)
      std               r12,-20h(r1)      // Saving LR here is extra step !
      mflr              r12
      stw               r12,-8h(r1)       // Return Address

      bl                PET_prolog
      bl                _internal_PET_LogEntry
      b                 PET_epilog
  }
}


void __declspec(naked) _cdecl __pexit( void )
{
  __asm
  {
      // Tiny Prolog
      //  - Set link register (r12) & return address (two steps)
      std               r12,-20h(r1)      // Saving LR here is extra step !
      mflr              r12
      stw               r12,-8h(r1)       // Return Address

      bl                PET_prolog
      bl                _internal_PET_LogExit
      b                 PET_epilog
  }
}
```

XBOX 360 (PowerPC assembler) Prolog and Epilog for __penter() and __pexit()

```
static void __declspec(naked) _cdecl PET_prolog( void )
{
  __asm
  {
      // Tiny Prolog
      //  - r12 is free to use (saved by caller)
      //  - Save extra registers (r11,r14)
      //  - Set stack frame (r1)
      //  - Compute TLS offset pointer(s)
      std               r11,-30h(r1)
      std               r14,-28h(r1)

      // Old Stack Pointer (r1) is at 0(r1) after this instruction
      stwu              r1,-100h(r1)


      // Get the TLS thread-specific base
      lwz               r11,0(r13)
      // Do not try to run in DPC!
      // In DPC { 0(r13) == 0 }
      cmplwi            cr6,r11,0
      beq               cr6,label__early_exit_prolog

      lau               r14,_tls_start    // Get the TLS global base
      lau               r12,tls_PET_bIsInProcessing
      lal               r14,r14,_tls_start
      lal               r12,r12,tls_PET_bIsInProcessing
      sub               r11,r11,r14       // TLS Base Offset (r11)
      add               r14,r11,r12       // r14 == &tls_PET_bIsInProcessing


      // Avoid recursion using thread variable tls_PET_bIsInProcessing
      lwzx              r12,r11,r12
      cmplwi            cr6,r12,0
      bne               cr6,label__early_exit_prolog
      li                r12,1
      stw               r12,0(r14)  // Set tls_PET_bIsInProcessing


      // Save r0/r2-r10 (temporaries)
      std               r0,8h(r1)
      std               r2,10h(r1)  // (r2 is reserved on XBOX 360)
      std               r3,18h(r1)
      std               r4,20h(r1)
      std               r5,28h(r1)
      std               r6,30h(r1)
      std               r7,38h(r1)
      std               r8,40h(r1)
      std               r9,48h(r1)
      std               r10,50h(r1)

      blr               // Return To Caller
```

```
label__early_exit_prolog:
      // Tiny Epilog -- adjust stack (r1) & restore r12/r14/r11
      addi              r1,r1,100h
      lwz               r12,-8h(r1)
      mtlr              r12
      ld                r12,-20h(r1)
      ld                r14,-28h(r1)
      ld                r11,-30h(r1)
      blr
  } // __asm
}

void __declspec(naked) _cdecl PET_epilog( void )
{
  __asm
  {
      // Clear tls_PET_bIsInProcessing
      li                r12,0
      stw               r12,0(r14)

      // Restore r0/r2-r10 (temporaries)
      ld                r0,8h(r1)
      ld                r2,10h(r1)  // (r2 is reserved on XBOX 360)
      ld                r3,18h(r1)
      ld                r4,20h(r1)
      ld                r5,28h(r1)
      ld                r6,30h(r1)
      ld                r7,38h(r1)
      ld                r8,40h(r1)
      ld                r9,48h(r1)
      ld                r10,50h(r1)

      // Tiny Epilog -- adjust stack (r1) & restore r12/r14/r11
      addi              r1,r1,100h
      lwz               r12,-8h(r1)
      mtlr              r12
      ld                r12,-20h(r1)
      ld                r14,-28h(r1)
      ld                r11,-30h(r1)
      blr
  }
}
```

PowerPC ABI (required knowledge when writing NAKED ASM functions on XBOX 360)

```
Misc Registers:

       lr              link register (HW)
       r0              general purpose (special: no-index ops for 'x')

       r1              stack register & gpr register save/restore base
       r12             link register (gpr save)
                       & fpr register save/restore base

       r13             thread specific storage
                       & special register (mtmsree r13)

       r2              Reserved for OS Use on XBOX360
                       (temporary on standard PowerPC ABI)

Temporary Registers
       r2              temporary except on XBOX360
       r3 .. r10       (also passed regs)
       r11             temporary - preserved in nested function
                       but not by leaf functions

Saved Registers:
       r14 .. r31      (saved/restored off r1)
                               __savegprlr / __restgprlr
       fr14 .. fr31    (saved/restored off r12)
                               __savefpr / __restfpr
       NOTE: use higher saved regs first ( r31 / fr31 )

Stack Frame Building:
       __restgprlr     auto-return to restored link-register r12

       Stwu            STore Word with Update
                       push & update stack pointer

               stwu    rS,d(rA)
                       EA <- (rA) + EXTS(d)
                       MEM(EA,4) <- rS[32..63]
                       rA <- EA

Passed Registers:
       r3              first parameter
                       & "this" pointer (C++ member functions)
       r4..r10         additional register-passed parameters
       [on stack]      stack-passed parameters (additional)

       fr1..fr13       floating registers

Return Registers:
       fr1             floating point return value (thru fr4)
       r3              integer return value (thru r10)
```

Complete PS3 PPU ABI can be found online – Search https://ps3.scedev.net for "PPU ABI Specifications" in "SDK Docs" sections

GDC 2012 March 5-9

Runtime CPU Performance Spike Detection using Manual and Compiler Automated Instrumentation

Adisak Pochanayon : adisak@wbgames.com
Principal Software Engineer, Mortal Kombat Team
Netherrealm Studios (WB Games Chicago)

This is a handout for source details that are would be impossible to read off a slide. Also, it should prove handy for later reference.

Implementing a Detour using Trampoline Functions – MIPS version

```
//------------------------------------------------------------
// HEADER - MIPS DETOUR / TRAMPOLINE

#define MIPS_InstrumentedFunction_NumCodeWords  (6)
typedef UINT32 MIPS_InstFunc_CodeByte;

// This structure should be "private" in that it shouldn't be used by
// functions that don't know what they're doing.
typedef struct _MIPS_InstrumentedFunction_Desc
{
      void  *m_fnPtrInstrumentedTarget;
      void  *m_fnPtrDetourRoutine;
      int   m_nReplicatedCodeWords;
      MIPS_InstFunc_CodeByte m_nTrampolineBuffer
            [MIPS_InstrumentedFunction_NumCodeWords];
} MIPS_InstrumentedFunction_Desc;

void *MIPS_GetTrampoline(MIPS_InstrumentedFunction_Desc *pIFDesc)
{ return(pIFDesc->m_nTrampolineBuffer); }

int MIPS_InstrumentFunction(MIPS_InstrumentedFunction_Desc *pIF,
      void *pFnInstrumentedTarget,void *pFnDetour);

#define MIPS_TRAMPOLINE_CALL(FN_TYPE,GPIF_DESC) \
      ((FN_TYPE)MIPS_GetTrampoline(GPIF_DESC))

//------------------------------------------------------------
// LIBRARY - MIPS DETOUR / TRAMPOLINE

// MIPS_DoesInstructionRequireDelaySlot()
//    returns true if Branch Delay Slot required:
//          j, jal, jr, jalr, branch-conditional (bXX), syscall,
//          break, trap (tXXX), branch-coprocessor (bcNXX), etc.
extern bool MIPS_DoesInstructionRequireDelaySlot(
      MIPS_InstFunc_CodeByte Code);

static unsigned int GetMastBits(unsigned int value,unsigned int
topbit,unsigned int bottombit)
{
```

```
	return ( (value>>bottombit)&((1<<(topbit-bottombit+1))-1) );
}

static int MIPS_InstFunc_SetJump(MIPS_InstFunc_CodeByte *pCode,void *pTarget)
{
	MIPS_InstFunc_CodeByte  nCode,nTarget;
	nCode = (MIPS_InstFunc_CodeByte) pCode;
	nTarget     = (MIPS_InstFunc_CodeByte) pTarget;
	// Test to see that top 4-bits match
	if(GetMastBits(nCode,31,28)==GetMastBits(nTarget,31,28))
	{
		// Insert two instructions: j (jump) + nop
		// j Target -- NOTE: uses 26 bits for Target
		// (4 top bits from PC + 2 implicit lowbits of zero)
		pCode[0]=(0x02<<26) | GetMastBits(nTarget,27,2);
		// nop (in branch delay slot)
		pCode[1]=0;
		return(1);
	}
	return(0);
}

int MIPS_InstrumentFunction(MIPS_InstrumentedFunction_Desc *pIF,
	void *pFnTargetToInstrument,void *pFnDetour)
{
	int   nCopiedCodeWords;
	MIPS_InstFunc_CodeByte  *pInstTarget;

	// Set the Instrumented Target and Detour functions
	pIF->m_fnPtrInstrumentedTarget=pFnTargetToInstrument;
	pIF->m_fnPtrDetourRoutine=pFnDetour;

	// Actually copy over the Instrumented Target to code buffer
	pInstTarget=(MIPS_InstFunc_CodeByte*)pFnTargetToInstrument;
	pIF->m_nTrampolineBuffer[0]=pInstTarget[0];
	pIF->m_nTrampolineBuffer[1]=pInstTarget[1];
	// Make sure that branch delay slots are handled.
	if(MIPS_DoesInstructionRequireDelaySlot(pInstTarget[1])) {
		pIF->m_nTrampolineBuffer[2]=pInstTarget[2];
		nCopiedCodeWords=3;
	} else {
		nCopiedCodeWords=2;
	}
	pIF->m_nReplicatedCodeWords=nCopiedCodeWords;

	// TODO : Add check for code we cannot instrument properly
	//          Bad code includes forward branches out of
	//          copied code or backwards branches into
	//          absolute jump/nop used to instrument.

	// Now that base code is copied to the Trampoline Buffer
	// actually instrument the Instrumented Target
	MIPS_InstFunc_SetJump(pInstTarget,pFnDetour);
	// Make the trampoline jump to Instrumented Target continuation point
	MIPS_InstFunc_SetJump(&(pIF->m_nTrampolineBuffer[nCopiedCodeWords]),
		&(pInstTarget[nCopiedCodeWords]));
```

```
	return(1);  // SUCCESS !!!
}


//-------------------------------------------------------
// EXAMPLE - MIPS DETOUR / TRAMPOLINE

typedef void (*pFN_MIPS_Intrument_Test)(const char *message);

void TestOriginal(const char *message)
{
	printf("Original Message Printer:%s\n",message);
}

MIPS_InstrumentedFunction_Desc TestDetourTrampoline;
void TestDetour(const char *message)
{
	MIPS_InstrumentedFunction_Desc *pTrampoline;

	printf("Detour BEFORE with message:%s\n",message);

	// Call the original version of the instrumented function
	//    through the trampoline funtion
	pTrampoline=&TestDetourTrampoline;
	MIPS_TRAMPOLINE_CALL(pFN_MIPS_Intrument_Test,pTrampoline)(message);

	printf("Detour After with message:%s\n",message);
}

// Use a function pointer for this test to make sure that TestOriginal()
// doesn't get inlined even in an optimized build
pFN_MIPS_Intrument_Test MIPS_Intrument_Test_Function=TestOriginal;

void MIPS_Intrument_Test()
{
	MIPS_InstrumentFunction(&TestDetourTrampoline,TestOriginal,TestDetour);
	MIPS_Intrument_Test_Function("Test Detour and Trampoline Functions");
}
//-------------------------------------------------------
// END OF EXAMPLE CODE
```

Branch Delay Slots: j, jal, jr, jalr, branch-conditional (bXX), syscall, break, trap (tXXX), branch-coprocessor (bcNXX), etc.

Note Trampoline functions are trivial to implement on RISC due to uniform instruction size. Only caveats are checking for branch delay slots or jumps (source or target) within the trampoline.

Microsoft 1999 Paper on Detours and Trampolines:
http://research.microsoft.com/pubs/68568/huntusenixnt99.pdf
Microsoft Detours Software ($9,999.95 commercial use / Express == free for Non-commercial)
http://research.microsoft.com/en-us/projects/detours/

If you use any of these techniques in your games, I'd be interested in hearing the results at adisak@wbgames.com or adisak@gmail.com