Instruction Tracing and dynamic codegen analysis to identify unique llvm performance issues.

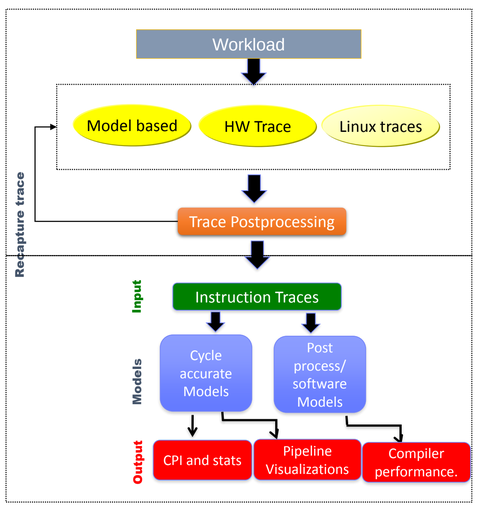
Biplob Mishra

INTRODUCTION

Performance analysis of the machine code generated by a compiler can be carried out in different ways and can also be based on application in question. Common methods use some form of profiling on a running program which generally provides the statistical information about certain data and events. While this method does give important insights to a performance problem, some of the issues are more clearly understood when the compiled applications is actually run and the dynamic instructions of hot code execution paths are traced and analyzed in a small execution window.

Trace records contain instructions and data, memory addresses and other information which provide complete visibility into the workings of an application.

TRACE OVERVIEW



Trace collection

```
68  0000000010136e00  0          38B50050 addi     r5,r21,80
69  0000000010136e04  0          3B35005E addi     r25,r21,94
70  0000000010136e08  0          7ED17378 mr       r17,r14
71  0000000010136e0c  0          F86100D8 std      r3,216(r1)    IEA=00007fffc96f4508
72  0000000010136e10  0          3875001C addi     r3,r21,28
73  0000000010136e14  0          F86100D0 std      r3,208(r1)    IEA=00007fffc96f4500
74  0000000010136e18  0          E86101E8 ld       r3,488(r1)    IEA=00007fffc96f4618

75  0000000010136e1c  0          2DA30000 cmpdi    cr3,r3,0
76  0000000010136e20  0          E86101F8 ld       r3,504(r1)    IEA=00007fffc96f4628
77  0000000010136e24  0          E8C30000 ld       r6,0(r3)      IEA=00007fffc96f49a0
78  0000000010136e28  0          7CC13040 cmpld    r17,r6
79  0000000010136e2c  0 00 80 00 40800688 bge
80  0000000010136e30  0          88110000 lbz      r0,0(r17)     IEA=00000000339f0675
81  0000000010136e34  0          28840085 cmplwi   cr1,r4,133
82  0000000010136e38  0 00 80 00 41850688 bgt      cr1,    exceeded branch depth
83  0000000010136e3c  0          78830020 clrldi   r3,r4,32
84  0000000010136e40  0          78631764 rldicr   r3,r3,2,61
85  0000000010136e44  0          7C63249A lwax     r3,r3,r8      IEA=00000000102339 1c
86  0000000010136e48  0          7C634214 add      r3,r3,r8
87  0000000010136e4c  0          7C690346 mtctr    r3
88  0000000010136e50  1 00 80 00 4E800420 bctr     IEA=0000000010136e8c,
89  0000000010136e8c  0          8082C2D8 lwz      r4,-15656(r2)     IEA=0000000010274 1d8
90  0000000010136e90  0          806F0000 lwz      r3,0(r15)     IEA=00000000 2d7c5b20
91  0000000010136e94  0          7C041800 cmpw     r4,r3
```

Sample trace with data addresses

```
0000000010136e04 addi    r25,r21,94
0000000010136e08 mr      r17,r14
0000000010136e0c std     r3,216(r1)
0000000010136e10 addi    r3,r21,28
0000000010136e14 std     r3,208(r1)
0000000010136e18 ld      r3,488(r1)
0000000010136e1c cmpdi   cr3,r3,0
0000000010136e20 ld      r3,504(r1)
0000000010136e24 ld      r6,0(r3)
0000000010136e28 cmpld   r17,r6
0000000010136e2c bge     00000000101374b4
0000000010136e30 lbz     r0,0(r17)
0000000010136e34 cmplwi  cr1,r4,133
0000000010136e38 bgt     cr1,00000000101374c0
0000000010136e3c clrldi  r3,r4,32
0000000010136e40 rldicr  r3,r3,2,61
0000000010136e44 lwax    r3,r3,r8
0000000010136e48 add     r3,r3,r8
0000000010136e4c mtctr   r3
0000000010136e50 bctr
0000000010136e8c lwz     r4,-15656(r2)
0000000010136e90 lwz     r3,0(r15)
0000000010136e94 cmpw    r4,r3
0000000010136e98 bgt-    000000001013a814
0000000010136e9c ld      r3,216(r1)
0000000010136ea0 lwz     r4,0(r10)
0000000010136ea4 lwz     r3,0(r3)
0000000010136ea8 cmplw   r3,r4
0000000010136eac bgt-    0000000010138280
0000000010136eb0 stw     r3,112(r19)
0000000010136eb4 lwz     r3,28(r21)
0000000010136eb8 stw     r3,116(r19)
0000000010136ebc mfocrf  r3,4
0000000010136ec0 ld      r16,40(r21)
0000000010136ec4 lwz     r4,484(r1)
0000000010136ec8 rlwimi  r3,r4,12,20,20
0000000010136ecc cmpldi  r16,0
0000000010136ed0 mtocrf  4,r3
0000000010136ed4 crandc  4*cr4+gt,4*cr4+gt,4*cr5+lt
0000000010136ed8 beq-    00000000101 3a7f8
0000000010136edc lbz     r3,1(r16)
0000000010136ee0 crclr   4*cr5+lt
0000000010136ee4 mr      r14,r17
0000000010136ee8 mfocrf  r4,4
0000000010136eec rlwinm  r4,r4,20,0,0
0000000010136ef0 cmplwi  r3,30
0000000010136ef4 stw     r4,484(r1)
0000000010136ef8 beq     0000000010136dc8
0000000010136efc cmplwi  r3,62
```

->Trace is the dynamic sequence of executed instructions as it occurred when the program was executed.

-> Trace provides complete history of what happens in a processor when the an application runs.

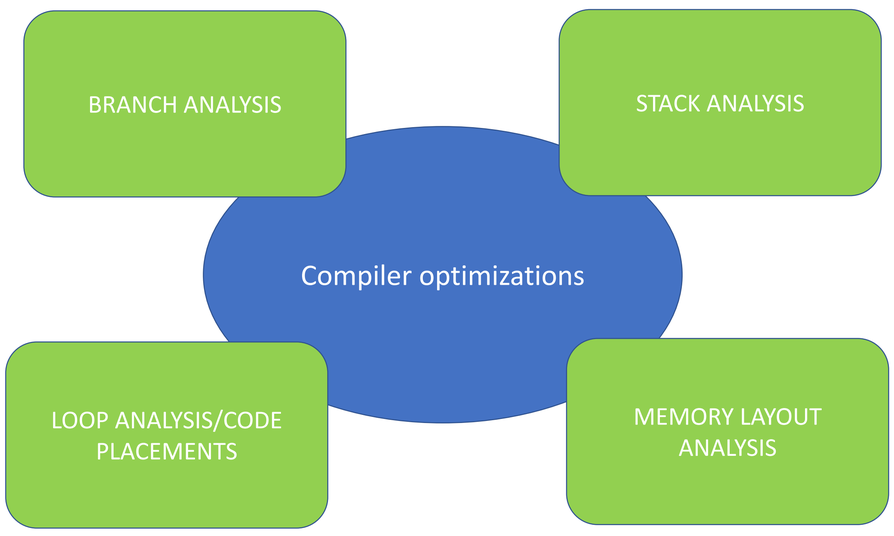
-> Multiple events as occurred on Instruction/code block execution can also be Superimposed on the dynamic Execution sequence.

-> On right we have a “perf record” Profile of the same which contains cycle event imposed on a static profile.

-> The static profile though useful does not give complete detail of running program.

```
0.05 10136e04:   addi    r25,r21,94
0.02 10136e08:   mr      r17,r14
0.75 10136e0c:   std     r3,216(r1)
0.02 10136e10:   addi    r3,r21,28
0.94 10136e14:   std     r3,208(r1)
0.98 10136e18:   ld      r3,488(r1)
0.04 10136e1c:   cmpdi   cr3,r3,0
0.08 10136e20:   ld      r3,504(r1)
0.54 10136e24:   ld      r6,0(r3)
0.04 10136e28:   cmpld   r17,r6
0.39 10136e2c: ↓ bge     101374b4
0.34 10136e30:   lbz     r0,0(r17)
0.04 10136e34:   cmplwi  cr1,r4,133
0.07 10136e38: ↓ bgt     cr1,101374c0
0.10 10136e3c:   clrldi  r3,r4,32
0.11 10136e40:   rldicr  r3,r3,2,61
3.09 10136e44:   lwax    r3,r3,r8
0.19 10136e48:   add     r3,r3,r8
20.29 10136e4c:  mtctr   r3
0.02 10136e50: ↑ bctr
0.26 10136e54:   lwz     r3,112(r19)
0.04 10136e58:   li      r5,110
         10136e5c:   addi    r16,r16,4
0.01 10136e60:   mr      r4,r17
0.31 10136e64:   stw     r3,24(r21)
         10136e68:   lwz     r3,116(r19)
         10136e6c:   std     r28,40(r21)
0.04 10136e70:   stw     r3,28(r21)
0.01 10136e74:   lwz     r3,-15656(r2)
0.01 10136e78:   stw     r3,32(r21)
0.01 10136e7c:   stw     r5,0(r21)
         10136e80: ↓ bgt     cr5,101372a0
0.01 10136e84:   mr      r14,r17
         10136e88: ↓ b       1013cb2c
0.22 10136e8c:   lwz     r4,-15656(r2)
0.02 10136e90:   lwz     r3,0(r15)
0.02 10136e94:   cmpw    r4,r3
         10136e98: ↓ bgt-    1013a814
0.02 10136e9c:   ld      r3,216(r1)
0.02 10136ea0:   lwz     r4,0(r10)
0.15 10136ea4:   lwz     r3,0(r3)
         10136ea8:   cmplw   r3,r4
```

WHERE CAN TRACES PLAY A ROLE



BRANCH ANALYSIS

```
0000000010136e04 addi    r25,r21,94
0000000010136e08 mr      r17,r14
0000000010136e0c std     r3,216(r1)
0000000010136e10 addi    r3,r21,28
0000000010136e14 std     r3,208(r1)
0000000010136e18 ld      r3,488(r1)
0000000010136e1c cmpdi   cr3,r3,0
0000000010136e20 ld      r3,504(r1)
0000000010136e24 ld      r6,0(r3)
0000000010136e28 cmpld   r17,r6
0000000010136e2c bge     00000000101374b4 <S_regmatch+0x9f4>
0000000010136e30 lbz     r0,0(r17)
0000000010136e34 cmplwi  cr1,r4,133
0000000010136e38 bgt     cr1,00000000101374c0 <S_regmatch+0xa00>
0000000010136e3c clrldi  r3,r4,32
0000000010136e40 rldicr  r3,r3,2,61
0000000010136e44 lwax    r3,r3,r8
0000000010136e48 add     r3,r3,r8
0000000010136e4c mtctr   r3
0000000010136e50 bctr
0000000010136e8c lwz     r4,-15656(r2)
0000000010136e90 lwz     r3,0(r15)
0000000010136e94 cmpw    r4,r3
0000000010136e98 bgt-    000000001013a814 <S_regmatch+0x3d54>
0000000010136e9c ld      r3,216(r1)
0000000010136ea0 lwz     r4,0(r10)
0000000010136ea4 lwz     r3,0(r3)
0000000010136ea8 cmplw   r3,r4
0000000010136eac bgt-    0000000010138280 <S_regmatch+0x17c0>
0000000010136eb0 stw     r3,112(r19)
0000000010136eb4 lwz     r3,28(r21)
0000000010136eb8 stw     r3,116(r19)
0000000010136ebc mfocrf  r3,4
0000000010136ec0 ld      r16,40(r21)
0000000010136ec4 lwz     r4,484(r1)
0000000010136ec8 rlwimi  r3,r4,12,20,20
0000000010136ecc cmpldi  r16,0
0000000010136ed0 mtocrf  4,r3
0000000010136ed4 crandc  4*cr4+gt,4*cr4+gt,4*cr5+lt
0000000010136ed8 beq-    00000000101 3a7f8 <S_regmatch+0x3d38>
0000000010136ec0 cmpldi  r3,255
0000000010136ec4 ble+    000000001013b0a4 <S_regmatch+0x45e4>
0000000010136ec8 ld      r4,8(r29)
0000000010136ecc rldicr  r3,r3,1,62
0000000010136ed0 lhzx    r5,r4,r3
0000000010136ed4 ld      r3,544(r1)
0000000010136ed8 cmpldi  r3,0
0000000010136edc bne-    000000001013c23c <S_regmatch+0x577c>
```

not taken

indirect branches

taken branch

-> Trace can be used to derive critical information on branch intensive codes.

-> determine which branches are critical to performance.

-> for direct branches determine the % times it was taken vs non-taken.

-> above can be useful in optimal branch placements.

-> also evaluate the indirect branch call destination distribution.

CODE ANALYSIS

QUICK SORT

```
void quick_sort_r(int *array, int l, int r)
{
    int partition_key, i, j;
    if(r <= l)
    return;

    i = l-1;
    j = r;
    partition_key = array[r];
    while(1) {
    while(array[++i] < partition_key)
        ;
    while(partition_key < array[--j]) {
        if( j == l )
        break;
    }
    if( i >= j )
        break;
    swap(array[i], array[j]);
    }
    swap(array[i], array[r]);
    quick_sort_r(array, l, i-1);
    quick_sort_r(array, i+1, r);
}
```

CALL NO	NUM_INSTR
1	X
2	0.92X
3	X
4	0.90X
5	0.97X
6	0.93X
7	3X
8	0.92X
9	X
10	0.93X

DRIVER

```
    int tmp_array[N];
    int size;
    init();
    for(i=0; i<10; i++) {
    // Let's read in an array and sort it
    read_array(tmp_array, &size, i);
    quick_sort(tmp_array, size);
```

-> Sample quicksort Program and the driver which calls it 10 times to sort an array.

-> Using statistical profiler it would be difficult to determine if a single iteration was slow.

-> Table derived from trace data tells us that call7 is the one of interest.

-> Events(Instruction) collected are as a ratio of event from 1st call.

STACK ANALYSIS

```
0000000010136e04 <S_regmatch+0x344>  addi    r25,r21,94
0000000010136e08 <S_regmatch+0x348>  mr      r17,r14
0000000010136e0c <S_regmatch+0x34c>  std     r3,216(r1)
0000000010136e10 <S_regmatch+0x350>  addi    r3,r21,28
0000000010136e14 <S_regmatch+0x354>  std     r3,208(r1)
0000000010136e18 <S_regmatch+0x358>  ld      r3,488(r1)
0000000010136e1c <S_regmatch+0x35c>  cmpdi   cr3,r3,0
0000000010136e20 <S_regmatch+0x360>  ld      r3,504(r1)
0000000010136e24 <S_regmatch+0x364>  ld      r6,0(r3)
0000000010136e28 <S_regmatch+0x368>  cmpld   r17,r6
0000000010136e2c <S_regmatch+0x36c>  bge     00000000101374b4 <S_regmatch+0x9f4>
0000000010136e30 <S_regmatch+0x370>  lbz     r0,0(r17)
0000000010136e34 <S_regmatch+0x374>  cmplwi  cr1,r4,133
0000000010136e38 <S_regmatch+0x378>  bgt     cr1,00000000101374c0 <S_regmatch+0xa00>
0000000010136e3c <S_regmatch+0x37c>  clrldi  r3,r4,32
```

stack usage

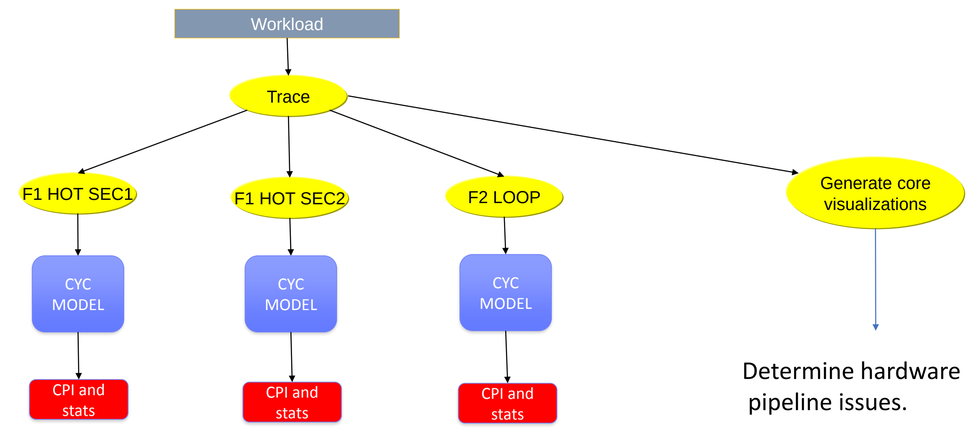
-> the image shows a push to and pop from the stack.

-> In a trace keep a count of stack accesses at different code levels.

-> with help of the instruction addresses which map to a particular function total stack access for a function can be determined.

-> Generally useful in comparing two different compiler register usage and stack accesses.
-> Analyze what percentage of instructions and cycles are used as stack operations.
-> Stack usage in prologue/epilogue vs hot path.
-> Register spill analysis in critical paths.
*as in above example it can be useful to determine if stack usage shoots up in a particular call to the function/or a different execution path is taken.

BEYOND



Enable cycle and other event analysis at lower code levels, particularly the hot loops or code sections within a function.
So the above quicksort example cycle impact can be evaluated per function call level.
The traces can further be used to evaluate core pipeline visualizations and determine hardware issues which cannot be explained by just disassembly analysis.