Scalable Tools Workshop 2023

Binary Visualization: Needs and Directions

Matt Legendre and Shadmaan Hye

http://bit.ly/BinaryVis

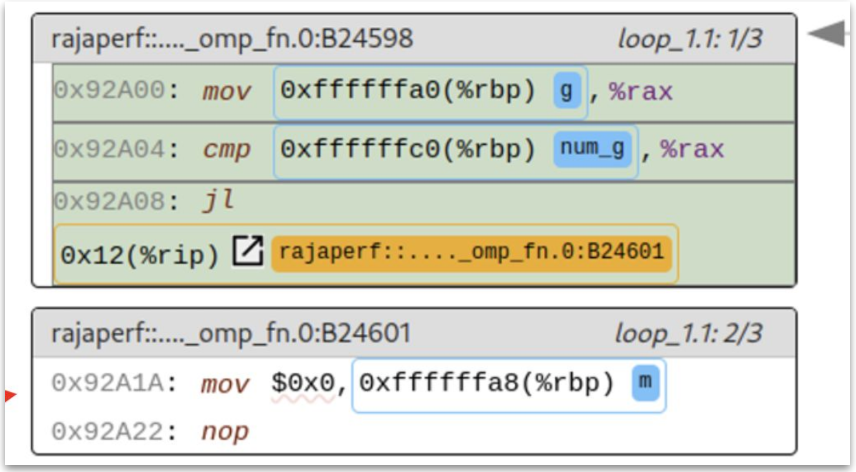
Participants

- Shadmaan Hye (The University of Utah)
- Matt Legendre (LLNL)
- Connor Scully-Allison (The University of Utah)
- Kate Isaacs (The University of Utah)
- Kathleen Shoga (LLNL)
- Wileam Phan (Rice)
- Jim Kupsch (Wisconsin)
- Tim Haines (Wisconsin)
- William Jalby (University of Versailles Paris Saclay)
- Cédric Valensi (University of Versailles Paris Saclay)
- Rahat Zaman (University of Utah)

Feature Possibilities & Requests

What are people trying to analyze in the binary file (so we can figure out what features to add)?

- [Upcoming Dyninst Feature!] Data flow analysis by propagating registers (we can associate with variables and/or know type, know which registers are live/dead) ← Associate this info with instruction/operand. Users who wrote the code can help understand what's going on.
  - In the added image, there would be more info in the **blue boxes**
  - Useful to annotate NOT replace
  - Timeline? - Next few weeks
  - API: Data flow object
  - Processing Cost: Fine as long as functions not 100s of MBs :)



rajaperf::...._omp_fn.0:B24598 *loop_1.1: 1/3*

```
0x92A00:  mov  0xffffffa0(%rbp) g , %rax
0x92A04:  cmp  0xffffffc0(%rbp) num_g , %rax
0x92A08:  jl
0x12(%rip) rajaperf::...._omp_fn.0:B24601
```

rajaperf::...._omp_fn.0:B24601 *loop_1.1: 2/3*

```
0x92A1A:  mov  $0x0, 0xffffffa8(%rbp) m
0x92A22:  nop
```

http://bit.ly/BinaryVis

Displaying an instruction

- Dyninst pretty-printers are designed for the terminal. How can we get better strings?
    - **[Converged idea] Would be good if we could access piecemeal an instruction knowing *what* each piece is**
        - **Possibly vector of structs that says what is an operand or opcode, which operand is it, which opcode is it, etc.**
        - **This gives us the ability on the front-end to provide multiple style options to users**
    - Other ideas
        - Dyninst could add them to translate operands
        - Should we the consumer give the operand annotations so we can get them back?
        - Maybe ask Dyninst for the opcode of operand 1, but they might be in different orders depending on architecture
            - Is Operand 1 the leftmost or rightmost operand? Depends on architecture.

http://bit.ly/BinaryVis

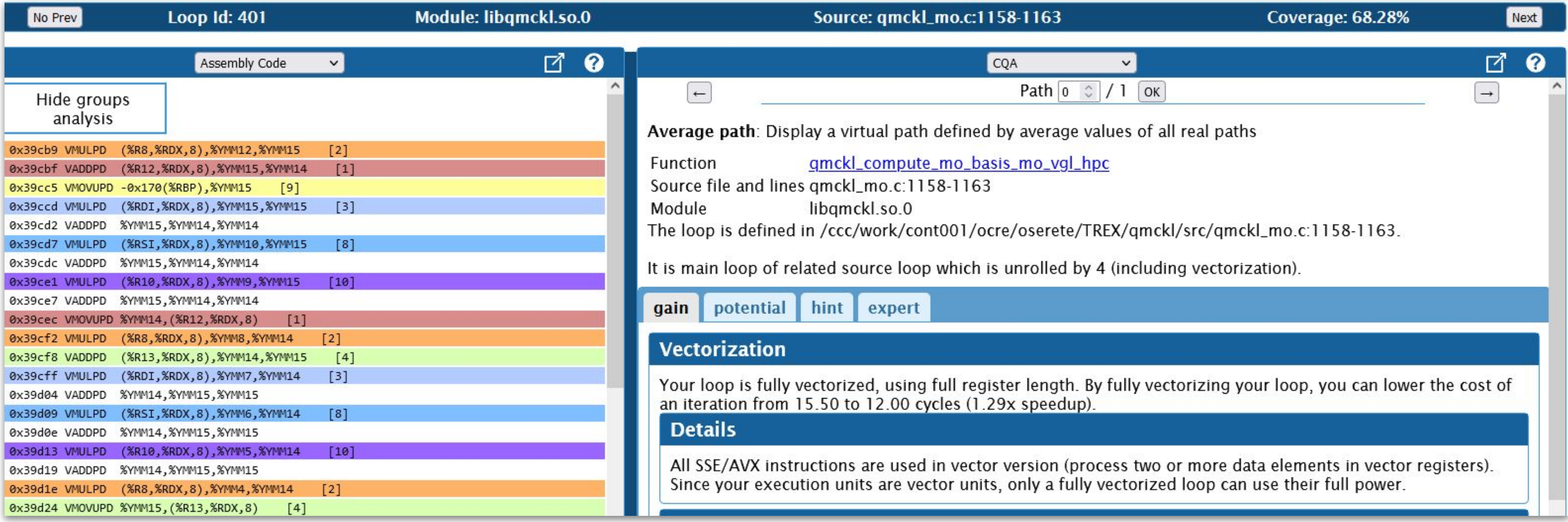
Instruction Re-ordering/Interleaving & Detecting Optimizations

- We want some way to make it more visually salient, concerns with if different instructions interleaved talk about different variables.
- We want to highlight things like hoisting. It would be good to have first class saliency in marking whether things get hoisted.
  - Want to be able to mark hoisting and unrolling, needs to be written on top of Dyninst to do it. Is this possible?
    - Code duplication could be quick if two blocks far away point to the same code
      - Or is it rough because DWARF doesn't break them up?
    - Loop unrolling similar? (See next slide)

Detecting Loop Unrolling

- Loop unrolling similar to code duplication? But what if vectorization?
  - HPCStruct does this with heuristics
  - Look for same source line + repeated operations (instruction mix)
  - Does detecting loop increments (induction variable) help? Can we do this in the binary?
    - Look for the conditional branch near the back edge?
      - Stencils will have common patterns
    - MAQAO does this, check out how they do it (example: https://tinyurl.com/yt9pjwpz), will send source file link
      - 90-95% success depending on degree of optimization and compiler tricks
  - Would “loop widths” (increments) be enough to understand that unrolling occur?

MAQAO Example



Loop Id: 401 Module: libqmckl.so.0 Source: qmckl_mo.c:1158-1163 Coverage: 68.28%

```
0x39cb9 VMULPD  (%R8,%RDX,8),%YMM12,%YMM15     [2]
0x39cbf VADDPD  (%R12,%RDX,8),%YMM15,%YMM14     [1]
0x39cc5 VMOVUPD -0x170(%RBP),%YMM15     [9]
0x39ccd VMULPD  (%RDI,%RDX,8),%YMM15,%YMM15     [3]
0x39cd2 VADDPD  %YMM15,%YMM14,%YMM14
0x39cd7 VMULPD  (%RSI,%RDX,8),%YMM10,%YMM15     [8]
0x39cdc VADDPD  %YMM15,%YMM14,%YMM14
0x39ce1 VMULPD  (%R10,%RDX,8),%YMM9,%YMM15     [10]
0x39ce7 VADDPD  %YMM15,%YMM14,%YMM14
0x39cec VMOVUPD %YMM14,(%R12,%RDX,8)     [1]
0x39cf2 VMULPD  (%R8,%RDX,8),%YMM8,%YMM14     [2]
0x39cf8 VADDPD  (%R13,%RDX,8),%YMM14,%YMM15     [4]
0x39cff VMULPD  (%RDI,%RDX,8),%YMM7,%YMM14     [3]
0x39d04 VADDPD  %YMM14,%YMM15,%YMM15
0x39d09 VMULPD  (%RSI,%RDX,8),%YMM6,%YMM14     [8]
0x39d0e VADDPD  %YMM14,%YMM15,%YMM15
0x39d13 VMULPD  (%R10,%RDX,8),%YMM5,%YMM14     [10]
0x39d19 VADDPD  %YMM14,%YMM15,%YMM15
0x39d1e VMULPD  (%R8,%RDX,8),%YMM4,%YMM14     [2]
0x39d24 VMOVUPD %YMM15,(%R13,%RDX,8)     [4]
```

Path 0 / 1

Average path: Display a virtual path defined by average values of all real paths

Function	qmckl_compute_mo_basis_mo_vgl_hpc
Source file and lines	qmckl_mo.c:1158-1163
Module	libqmckl.so.0

The loop is defined in /ccc/work/cont001/ocre/oserete/TREX/qmckl/src/qmckl_mo.c:1158-1163.

It is main loop of related source loop which is unrolled by 4 (including vectorization).

Vectorization

Your loop is fully vectorized, using full register length. By fully vectorizing your loop, you can lower the cost of an iteration from 15.50 to 12.00 cycles (1.29x speedup).

Details

All SSE/AVX instructions are used in vector version (process two or more data elements in vector registers). Since your execution units are vector units, only a fully vectorized loop can use their full power.

Colors show grouping of memory addresses (often a given array) like variable renaming. The vectorization (loop width) is shown on the right panel.

http://bit.ly/BinaryVis

Other optimizations we want to make salient?

- What other optimizations should we try to flag for our scientific users?
  - Loop Un-Switching (For-If becomes If-For/Else-For) as named in GCC aka “If-hoisting” as a way to think of it
    - Even more generically, other variants?
- Can we make conditionals more obvious some way visually? Would that make it easier to see the variants? Would users understand that?
  - Note if-statements don’t necessarily come back together.
  - What if there’s a no-return call in the middle?
- Aligned memory accesses – which instructions are aligned

Biggest Limitations

- On-demand piecemeal binary analysis and vis
  - Big binaries may take a while
  - We already have client-server so not all data is on the client
- Not yet packaged - coming soon!

http://bit.ly/BinaryVis

Can we obtain more on line mappings with instrumented rewritten binaries?

- Symtab API has it available but not available easily. HPCStruct using it.
- Right now can map something back to where it derived from, but a nicer interface to retrieve that.
- This is hairy enough we decide not to challenge it from the vis side now

We assume the user is fairly advanced.

CcNav is for compiler optimizations, not testing on user codes that aren't as optimized.

- Artifact of RAJAPerf