The Orc Quest for Better Multimedia Performance

Adding Mips support to liborc

Guillaume Emont - Software Engineer
Igalia
ELCE 2014 - Monday, October 13 2014

Some Context

What is SISD?

SISD: **S**ingle **I**nstruction **S**ingle Data

ASM

```
lbu  $2, variable1      ; $2 <- variable1
lbu  $3, variable2      ; $3 <- variable2
addu $2, $2, $3         ; $2 <- $2 + $3
sb   $2, variable3      ; variable3 <- $2
```

PSEUDO-CODE

```
variable3 <- variable1 + variable2
```

Some Context

What is SIMD?

SIMD: **S**ingle **I**nstruction **M**ultiple **D**ata

ASM

```
lw      $2, array1      ; $2 <- array1[0..3]
lw      $3, array2      ; $3 <- array2[0..3]
addu.qb $2, $2, $3      ; $2[i] <- $2[i] + $3[i] for i in 0..3
sw      $2, array3      ; array3[0..3] <- $2
```

PSEUDO-CODE

```
array3 <- array1 + array2
```

Some Context

SIMD use cases

- colorspace conversion (e.g. YUV -> RGB, etc)
- image scaling
- blending two videos feeds together
- various audio processing (volume, combine 2 sources, etc)

Some Context

Orc

The **O**il **R**untime **C**ompiler

OIL: **O**ptimized **I**nner **L**oops

"Portable SIMD"

Some Context

Orc

This ORC code

```ORC
.function simple_addb
.dest 1 d1
.source 1 s1
.source 1 s2
addb d1, s1, s2
```

Will give you this C function

```C
void simple_addb (orc_uint8 * ORC_RESTRICT d1,
                  const orc_uint8 * ORC_RESTRICT s1
                  const orc_uint8 * ORC_RESTRICT s2,
                  int n);
```

Some Context

Orc

```sh
$ orcc addb.orc --assembly --target mips -o addb-mips.s
$ wc -l addb-mips.s
      414 addb-mips.s
```

MIPS port

MIPS DSPv2 ASE

DSPv2 ASE provides:

- saturating arithmetics
- simple SIMD instructions (32 bits registers)
- fixed point arithmetic

MIPS port

MIPS DSPv2 ASE

Port is only for the DSPv2 ASE, and not for:

- MIPS SIMD Architecture (MSA)
- MIPS-3D
- MDMX

MIPS port

MIPS DSPv2 ASE

I used a MIPS32 74Kc

MIPS port

What's in a port?

- simple instruction builder
- rules
- program manager

MIPS port

What's in a port?

Simple instruction builder aka orcmips.{h,c}

- enum of available registers
- orc_mips_emit_<instruction>()
- some logic to handle labels and jumps

Can generate assembly source code or binary code.

MIPS port

What's in a port?

Rules (orcrules-mips.c): convert ORC opcode to binary code.

Example:

```C
void
mips_rule_addb (OrcCompiler compiler, void user, OrcInstruction *insn)
{
  int src1 = ORC_SRC_ARG (compiler, insn, 0);
  int src2 = ORC_SRC_ARG (compiler, insn, 1);
  int dest = ORC_DEST_ARG (compiler, insn, 0);
  orc_mips_emit_addu_qb (compiler, dest, src1, src2);
}
```

MIPS port

What's in a port?

Program manager

Constructs the whole "program" (= binary function)

- push registers on stack
- load constants and parameters into registers
- try to do some simple optimisations (load ordering)
- stride handling
- creates the many loops that are needed

MIPS port

Many loops

Many loops:

- 2^(n-1) alignment cases with (n=number of arrays)
- in case array is big: loop unrolling
- array end might not be aligned

MIPS port

Many loops

Alignment issues

This is only fine if $a1 is a multiple of 4

```asm
    lw  $t0, 0($a1)
```

If not:

```asm
    lwr  $t0, 0($a1)
    lwl  $t0, 3($a1)
```

We want to be in the first case as much as possible.

MIPS port

Many loops

ORC

```
.function simple_addb
.dest 1 d1
.source 1 s1
.source 1 s2
addb d1, s1, s2
```

MIPS port

Many loops

Really:

ORC

```
.function simple_addb
.dest 1 d1
.source 1 s1
.source 1 s2
.temp 1 t1
.temp 1 t2
loadb t1, s1
loadb t2, s2
addb t1, t1, t2
storeb s1, t1
```

MIPS port

Many loops

Everything aligned:

ASM

```
    / 0: loadb /
      lw      $t7, 0($a3)
    / 1: loadb /
      lw      $t6, 0($a2)
    / 2: addb /
      addu.qb $t6, $t6, $t7
    / 3: storeb /
      sw      $t6, 0($a1)
```

MIPS port

Many loops

If one is not aligned:

ASM

```
/ 0: loadb /
  lwr     $t7, 0($a3)
  lwl     $t7, 3($a3)
/ 1: loadb /
  lw      $t6, 0($a2)
/ 2: addb /
  addu.qb $t6, $t6, $t7
/ 3: storeb /
  sw      $t6, 0($a1)
```

MIPS port

Many loops

Strategy:

- Make sure we are aligned for at least d1
- One loop for each possible alignment configuration for (s1, s2)

MIPS port

Many loops

Loops we generate:

- byte by byte until d1 is aligned.

Then, loops for d1 aligned:

- s1 aligned, s2 not aligned.
- s1 not aligned, s2 aligned.
- everything aligned.
- neither s1 nor s2 aligned.

Finally:

- another byte by byte loop to finish processing if the arrays did not finish on an alignment border for d1.

MIPS port

Many loops

Loops we generate:

- byte by byte until d1 is aligned.

Then, loops for d1 aligned:

- s1 aligned, s2 not aligned. **Unrolled 8 times**
- s1 not aligned, s2 aligned. **Unrolled 8 times**
- everything aligned. **Unrolled 8 times**
- neither s1 nor s2 aligned. **Unrolled 8 times**

Finally:

- another byte by byte loop to finish processing if the arrays did not finish on an alignment border for d1.

MIPS port

Many loops

Overall algorithm of generated code

```
1. Load parameters
2. Calculate number of iterations needed to have d1 aligned
3. Loop until d1 is aligned
4. Check alignment of the other array pointers
5. Go to loop corresponding to our alignment configuration
6. Iterate in said loop
7. Iterate in "left-over" loop
```

MIPS port

Many loops

Overall algorithm of generated code

```
1. Load parameters -> 5 lines
2. Calculate number of iterations needed to have d1 aligned -> 15 lines
3. Loop until d1 is aligned -> 15 lines
4. Check alignment of the other array pointers  }
5. Go to loop corresponding to our alignment configuration  } 22 lines
6. Iterate in said loop -> ~80 lines x 4 -> 320 lines
7. Iterate in "left-over" loop -> 15 lines
```

Total: ~ 392

Add 22 lines of boilerplate and you know why it takes 414 lines of MIPS assembly to efficiently add two arrays byte by byte.

Some conclusions

```sh
$ wc -l orcmips.{h,c} orcprogram-mips.c orcrules-mips.c
  209 orcmips.h
  961 orcmips.c
  872 orcprogram-mips.c
  733 orcrules-mips.c
 2775 total
```

Some conclusions

A lot is already handled by the core of Orc:

- register allocation
- (part of) label management
- parameter passing convention
- generation of wrapping function

<Thank You!>

http://www.igalia.com/

twitter @guijemont
www emont.org/blog
github github.com/guijemont