Achieving Software Speed Records with `qhasm`

Peter Schwabe

Eindhoven University of Technology

12.10.2008

EIPSI Seminar

Overview

What is `qhasm`?

What does a `qhasm` program look like?

AES on the UltraSPARC – a CACE study

What is qhasm?

...as opposed to Assembly and C.

What is qhasm?

...as opposed to Assembly and C.

- Assembly:
  - Programmer has full control (choice of instructions, scheduling, usage of memory/registers)
  - Different instruction set for different architectures ⇒ different implementation for each architecture
  - Different syntax for different architectures

What is qhasm?

...as opposed to Assembly and C.

- Assembly:
  - Programmer has full control (choice of instructions, scheduling, usage of memory/registers)
  - Different instruction set for different architectures ⇒ different implementation for each architecture
  - Different syntax for different architectures
- C:
  - Choice of instructions, scheduling etc. left to compiler, programmer can only give hints (`register`)
  - Unified "instruction set" and unified syntax ⇒ just one implementation on all architectures

What is qhasm?

...as opposed to Assembly and C.

- Assembly:
  - Programmer has full control (choice of instructions, scheduling, usage of memory/registers)
  - Different instruction set for different architectures ⇒ different implementation for each architecture
  - Different syntax for different architectures
  - Programmer has to keep track of which "variable" is in which register
- C:
  - Choice of instructions, scheduling etc. left to compiler, programmer can only give hints (`register`)
  - Unified "instruction set" and unified syntax ⇒ just one implementation on all architectures

What is qhasm?

...as opposed to Assembly and C.

- Assembly:
  - Programmer has full control (choice of instructions, scheduling, usage of memory/registers)
  - Different instruction set for different architectures ⇒ different implementation for each architecture
  - Different syntax for different architectures
  - Programmer has to keep track of which "variable" is in which register
- C:
  - Choice of instructions, scheduling etc. left to compiler, programmer can only give hints (`register`)
  - Unified "instruction set" and unified syntax ⇒ just one implementation on all architectures
- qhasm assigns registers to register variables
- qhasm assigns stack space to stack variables automatically

Why would anyone want qhasm?

Why would anyone want qhasm?

Consider AES implementation for UltraSPARC

Why would anyone want qhasm?

Consider AES implementation for UltraSPARC

- 25.08 cycles/byte with gcc

Why would anyone want qhasm?

Consider AES implementation for UltraSPARC

- 25.08 cycles/byte with gcc
- 20.75 cycles/byte with Sun C compiler

Why would anyone want qhasm?

Consider AES implementation for UltraSPARC

- 25.08 cycles/byte with gcc
- 20.75 cycles/byte with Sun C compiler
- 15.98 cycles/byte with qhasm implementation

What does a qhasm program look like?

- No function calls
- One instruction (line) in qhasm translates into one CPU instrution
- Which instructions are available: Check documentation

The Baseline

- Consider 128 bit AES (10 Rounds) in Counter mode

The Baseline

- Consider 128 bit AES (10 Rounds) in Counter mode
- Each round has 20 loads, 16 shifts, 16 masks and 16 xors

The Baseline

- Consider 128 bit AES (10 Rounds) in Counter mode
- Each round has 20 loads, 16 shifts, 16 masks and 16 xors
- Last round is slightly different: Needs 16 more mask instructions
- Four load instructions to load input, four xors with key stream, four stores for output
- . . . some more overhead
- Results in 720 instructions needed to encrypt a block of 16 bytes
- Specifically: 208 loads, 4 stores, 508 integer instructions

How can the UltraSPARC handle these instructions?

Reminder: 208 loads, 4 stores, 508 integer instructions

How can the UltraSPARC handle these instructions?

Reminder: 208 loads, 4 stores, 508 integer instructions

- Can dispatch several (up to 4) instructions per cycle

How can the UltraSPARC handle these instructions?

Reminder: 208 loads, 4 stores, 508 integer instructions

- Can dispatch several (up to 4) instructions per cycle
- Only one load or store per cycle ($\Rightarrow$ at least 212 cycles)

How can the UltraSPARC handle these instructions?

Reminder: 208 loads, 4 stores, 508 integer instructions

- Can dispatch several (up to 4) instructions per cycle
- Only one load or store per cycle ($\Rightarrow$ at least 212 cycles)
- Only 2 integer instructions per cycle ($\Rightarrow$ at least 254 cycles)

How can the UltraSPARC handle these instructions?

Reminder: 208 loads, 4 stores, 508 integer instructions

- Can dispatch several (up to 4) instructions per cycle
- Only one load or store per cycle ($\Rightarrow$ at least 212 cycles)
- Only 2 integer instructions per cycle ($\Rightarrow$ at least 254 cycles)
- Idea: "Hide" load/store instructions between integer instructions (needs more registers!)

How can the UltraSPARC handle these instructions?

Reminder: 208 loads, 4 stores, 508 integer instructions

- Can dispatch several (up to 4) instructions per cycle
- Only one load or store per cycle ($\Rightarrow$ at least 212 cycles)
- Only 2 integer instructions per cycle ($\Rightarrow$ at least 254 cycles)
- Idea: "Hide" load/store instructions between integer instructions (needs more registers!)
- Result: 254 cycles/block, 15.98 cycles/byte in the eSTREAM benchmarking framework for encryption of 4096 bytes

Some more results (joint work with D.J. Bernstein)

- 12.08 cycles/byte for UltraSPARC III
- 14.57 cycles/byte for PowerPC G4 7410
- 14.15 cycles/byte for Pentium 4 f12
- 10.57 cycles/byte for Core 2
- 10.43 cycles/byte for Athlon64

Some more results (joint work with D.J. Bernstein)

- 12.08 cycles/byte for UltraSPARC III
- 14.57 cycles/byte for PowerPC G4 7410
- 14.15 cycles/byte for Pentium 4 f12
- 10.57 cycles/byte for Core 2
- 10.43 cycles/byte for Athlon64
- All these implementations improve upon previously fastest code.

Some more results (joint work with D.J. Bernstein)

- 12.08 cycles/byte for UltraSPARC III
- 14.57 cycles/byte for PowerPC G4 7410
- 14.15 cycles/byte for Pentium 4 f12
- 10.57 cycles/byte for Core 2
- 10.43 cycles/byte for Athlon64
- All these implementations improve upon previously fastest code.
- All these implementations are in the public domain