A Gate-Level Approach To Compiling For Quantum Computers

Keeping Current, 4:30PM Feb. 13, 2019

Hank Dietz
Professor and Hardymon Chair, Electrical & Computer Engineering
What Is A Quantum Computer?

Parallel processing *without* parallel hardware.

- **Qubits** instead of bits
  - Each qubit can be 0, 1, or *superposed*
  - *Entangled* qubits maintain values together
  - Measuring a qubit’s value picks 0 or 1
- Quantum computers are *not state machines*;
  all they implement is *combinatorial logic*
- Gates implemented *in sequence*
Kentucky’s Rotationally Emulated Quantum Computer

- 6 qubits encode up to $2^6$ 6-bit values
Optimizing / Parallelizing Compilers

- Programming languages like C and Fortran
- Lots of analysis and transformations!
- Speedup-oriented automatic parallelization
  - Recognize parallelizable loops, etc.
  - Rewrite for as parfor, etc.
- Many optimizations, mostly at the word level: Common subexpression elimination, folding, register allocation, code scheduling, ...

... do this at the bit level!
True Bit-Level Optimization

- Bit-slice systems were generally microcoded to implement a simple word-level ISA
- **Word-level operations can imply useless work**
  - E.g., using an **Add** to add 4 to a register:
True Bit-Level Optimization

```c
int:8 a, b, c;
a = (c * c) ^ 70;
a = ((a >> 1) & 1);
a = b + (c * b) + a;
a = a + ~(b * (c + 1));
```
True Bit-Level Optimization

```c
int:8 a, b, c;
a = (c * c) ^ 70;
a = ((a >> 1) & 1);
a = b + (c * b) + a;
a = a + ~(b * (c + 1));
```

Total of 206669 ITEs created, 8 kept
Language Support For Bit-Level Specification

- How big is an int?
  - C has types like `int_fast8_t`
  - Only supports 8, 16, 32, or 64 bits
  - PCC: 2,882 `int`, 174 `unsigned`, but just 44 specifying 8, 16, 32, or 64 bits!
- Allow syntax like `int:10`
- Can also use for floats, although we prefer specifying accuracy rather than precision
Language Support For Explicit Quantum Algorithms

• Allowing quantum values has very little impact on gate-level logic design optimization
• Could allow a q attribute for quantum bits
  – q int:5 a; would be a 5-qubit integer
  – int:5 *q p; would be a qubit pointer to a randomly selected 5-bit signed integer
• Could allow ? to be superpositioned bits
  – a=?; sets a to all possible 5-bit values
Issues In The Prototype “Hardly Software” Compiler

- No range nor precision analysis
- No code generation for array references – perhaps a conventional memory interface?
- Seamless handling of function calls, including recursion, not yet implemented (needs arrays)
- No support for cracking basic blocks – a single very complex basic block can increase the size of the combinatorial logic for all states
Basic Compilation Example

- Consider a trivial (8-bit default `int`) program:

```c
int a, b, c;

main()
{
    b = 42; a = 100;
    while (a > b) a = a - 1;
    c = a - b;
}
```
CSWAP (Fredkin) Logic

- “Billiard-ball model” adiabatic gate
- All signals must be unit-fanout
- Efficient quantum implementation (2016)
// 1-bit full adder
p=1;
q=1;
carry=0;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);

QUBIT

CSWAP

CSWAP

CSWAP

CSWAP

CSWAP

0  64  64
x  x  64
@  @  @

64  0  0
x  x  64
@  @  @

64  0  0
x  x  64
@  @  @

0  64  64
x  x  64
@  @  @

64  64  64
64  64  64
64  64  64
64  64  64

0  0  1  1  1
KREQC Program

// 1-bit full adder
p=1;
q=0;
carry=?;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);

Simulation Output

QUBIT

CSWAP

CSWAP

CSWAP

CSWAP

CSWAP

CSWAP

32/64

32/64
// 1-bit full adder
p=?;
qu=?;
carry=?;
parity=0;
g=1;
CSWAP(p, parity, g);
CSWAP(q, parity, g);
CSWAP(carry, parity, g);
CSWAP(parity, carry, g);
CSWAP(q, carry, g);

QUBIT

<table>
<thead>
<tr>
<th>g</th>
<th>parity</th>
<th>carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>48</td>
<td>16</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Simulation Output

<table>
<thead>
<tr>
<th>g</th>
<th>parity</th>
<th>carry</th>
<th>q</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
CSWAP Output From Prototype “Hardly Software” Compiler

- Unit-fanout CSWAP generation:
  1. AND/OR/NOT/XOR \( \Rightarrow \) multiplexors (MUX)
  2. MUX \( \Rightarrow \) CSWAP, inserting duplication gates wherever there is fanout
  3. **Search to use alternate CSWAP outputs**
  4. Order CSWAPs to sequence use of control pass-thru outputs, remove duplicate gates

- Considering **Genetic Algorithm** restructuring to minimize CSWAP complexity...
Second Prototype Compiler

- Reimplementation using code from BitC
- New SITE ⇒ CSWAP algorithm
  - Incrementally creates duplicates as needed
  - Tracks “lanes” and routes new values to same lane the target variable began in
- Output as Verilog code, text “lane” diagram, gate list, and circuit diagram
int:4  a;  a=a*a;
Use Of Entangled Qubit Quantum Computation?

- Could express quantum algorithms using Hadamard values... by writing new code
- Compiling ordinary C code results in CSWAP logic that never uses entangled qubits?
  - Could substitute quantum operations for basic math functions, e.g., `sqrt()`
  - Could recognize parallelizable loops that produce a single result and “parallelize” them using Hadamard inputs
Conclusions

- Reduce power by using fewer gate-level ops
- Complete state machines can be implemented with minimal (if any) reconfiguration
- Gate-level compiler optimization of whole C programs to unit-fanout CSWAPs is feasible
- More to do to make use of entangled qubits, improve optimization