Hardware design space exploration using HercuLeS HLS

Nikolaos Kavvadias
nkavvadias@ajaxcompilers.com

CEO, Ajax Compilers, Athens, Greece
www.ajaxcompilers.com
The need for high-level synthesis (HLS)

- Moore’s law anticipates an annual increase in chip complexity by 58%
- At the same time, human designer’s productivity increase is limited to 21% per annum
- This designer-productivity gap is a major problem in achieving time-to-market of hardware products

Solution
Adoption of a high-level design and synthesis methodology imposing user entry from a raised level of abstraction
- Hide low-level, time-consuming, error-prone details
- Drastically reduce human effort
- End-to-end automation from concept to production

HLS
An algorithmic description is automatically synthesized to a customized digital embedded system
The HercuLeS environment

- HercuLeS is an easy to use, extensible, high-level synthesis environment for whole-program hardware compilation
- In development since 2009
- Marketed by Ajax Compilers
- HercuLeS targets both hardware and software engineers/developers
  1. ASIC/SoC developers, FPGA-based/prototype/reference system engineers
  2. Algorithm developers (custom hardware algorithm implementations)
  3. Application engineers (application acceleration)
- Terminology
  - ASIC: Application-Specific Integrated Circuit
  - SoC: System-on-a-Chip
  - FPGA: Field-Programmable Gate Array (post-fabrication reconfigurable chip)
The HercuLeS flow

- Optimized C code is passed to the host compiler for compiler intermediate representation (IR) generation
- *ir2nac* translates to N-Address Code (NAC)
- IP components are automatically inserted using black box function calls
- HercuLeS = *nac2cdfg + cdfg2hdl*
  - *nac2cdfg*: CDFG (Control-Data Flow Graph) extraction
  - *cdfg2hdl*: hardware and self-checking testbench generation
- ANSI C backend for rapid algorithm verification
- Optimizations at the source, NAC, Graphviz, and VHDL levels
The optimization space of Herculēs HLS

source language

N-Address Code

Graphviz CDFGs

VHDL

Hardware design space exploration using Herculēs HLS

Arithmetic

constant multiplication optimization
constant division optimization
optimization of polynomial expressions
low-level superoptimizations

Loop-oriented

circular scheduling
ASAP scheduling
dead node elimination

Restructuring

SSA construction
if-conversion
peephole transformations
basic block partitioning
black box function insertion
GNU MP extensions

level

constant division optimization
optimization of polynomial expressions
low-level superoptimizations

partial loop unrolling
full loop unrolling
loop coalescing
strip mining

array flattening
statement vectorizer

SSA construction
if-conversion
peephole transformations
basic block partitioning
black box function insertion
GNU MP extensions

sequential scheduling
ASAP scheduling
dead node elimination

hardware interpretation of SSA form
synchronous embedded memories (block RAMs)
operation chaining

 Nikoλαoς Kavvadias nkavvadias@ajaxcompilers.com
SSA (Single Static Assignment) form construction

- Pseudo-statements called $\phi$-functions join variable definitions from different control-flow paths
- Enforce a single definition site for each variable
- False dependencies are naturally removed
- Many analyses and optimizations are simplified
- HercuLeS supports minimal SSA (in the number of $\phi$s) and intrablock-only (pseudo) SSA

Prior SSA

Minimal SSA

Pseudo SSA

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

Hardware design space exploration using HercuLeS HLS
Representing hardware as FSMDs

<table>
<thead>
<tr>
<th>Port</th>
<th>Dir.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>I</td>
<td>external clocking source</td>
</tr>
<tr>
<td>reset</td>
<td>I</td>
<td>asynchronous (or synchronous) reset</td>
</tr>
<tr>
<td>start</td>
<td>I</td>
<td>enable computation</td>
</tr>
<tr>
<td>din</td>
<td>I</td>
<td>data inputs</td>
</tr>
<tr>
<td>dout</td>
<td>O</td>
<td>data outputs</td>
</tr>
<tr>
<td>ready</td>
<td>O</td>
<td>the block is ready to accept new input</td>
</tr>
<tr>
<td>valid</td>
<td>O</td>
<td>a data output port is streamed out</td>
</tr>
<tr>
<td>done</td>
<td>O</td>
<td>end of computation for the block</td>
</tr>
</tbody>
</table>

- FSMD (Finite-State Machine with Datapath) as a MoC is universal, well-defined and suitable for either data- or control-dominated applications
- FSMDs = FSMs with embedded datapath actions
- HercuLeS supports extended FSMDs (hierarchical calls, communication with on-chip memories, IP integration)
Hierarchical calls between FSMDs

- Supported to arbitrary depth and complexity (apart from recursion)
- Example of a caller FSMD, handing over computation to callee superstate (a square root computation)
- Variable-related quantities are represented by three signals: *next (value to-be-written), *reg (value currently read from register), *eval (callee output)
load Requires a wait-state register for devising a dual-cycle substate (address + data cycles)

store Raises block RAM write. Stored data are made available in the subsequent machine cycle
Automatic IP integration

**IPs** Third-party components used in hardware systems (e.g. dividers, floating-point operators)

- How to import and use your own IP
  1. Implement IP and place in proper subdirectory
  2. Add entry in text database
  3. Replace operator uses by black-box function calls
  4. HercuLeS creates a hierarchical FSMD with the requested callee(s)
A streaming-output implementation of prime factorization

```c
void pfactor(unsigned int x, unsigned int *outp) {
    unsigned int i = 2, n = x;
    while (i <= n) {
        while ((n % i) == 0) {
            n = n / i;
            *outp = i;
        }
        i = i + 1;
    }
}
```

NAC

```c
procedure pfactor(in u32 x, out u32 outp) {
    localvar u32 D_1369, i, n;
    L0005:
        n <= mov x;
        i <= ldc 2;
        D_1366 <= jmpun;
    D_1365:
        D_1363 <= jmpun;
    D_1362:
        (n) <= divu(n, i);
        outp <= mov i;
        D_1363 <= jmpun;
    D_1363:
        (D_1369) <= modu(n, i);
        D_1362, D_1364 <= jmpeq D_1369, 0;
    D_1364:
        i <= add i, 1;
        D_1366 <= jmpun;
    D_1366:
        D_1365, D_1367 <= jmple i, n;
    D_1367:
        nop;
}
```
Example: Prime factorization (2/4)
Example: Prime factorization (3/4)

```
when S_002_001 =>
  modu_10_start <= '1';
  next_state <= S_004_001;
when S_003_001 =>
  if (divu_6_ready='1' and
       divu_6_start='0') then
    n_1_next <= n_1_eval;
    next_state <= S_003_002;
  else
    next_state <= S_003_001;
  end if;
when S_004_001 =>
  if (modu_10_ready='1' and
       modu_10_start='0') then
    D_1369_1_next <= D_1369_1_eval;
    next_state <= S_004_002;
  else
    next_state <= S_004_001;
  end if;
when S_004_002 =>
  if (D_1369_1_reg = CNST_0) then
    divu_6_start <= '1';
    next_state <= S_003_001;
  else
    next_state <= S_005_001;
  end if;
```

```
VHDL

VHDL (cont.)
```
Example: Prime factorization (4/4)

Input vector data

Diagnostic output

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

Hardware design space exploration using HercuLeS HLS
Design space exploration configurations

- Explore different choices both in frontend translation (e.g. SSA construction) and hardware optimization
- Numerous configurations are possible
- The following configuration sets will be used
  - O1 sequential scheduling
  - O2 ASAP scheduling using SSA
  - O3 O2 with operation chaining (collapsing dependent operations to a single state)
  - O4 O3 with pseudo-SSA construction
  - O5 O3 with preserving $\phi$ functions
  - O6 O3 with block RAM inference
Fibonacci series example: Introduction

- Fibonacci series computation is defined as

\[
F(n) = \begin{cases} 
0 & n = 0 \\
1 & n = 1 \\
F(n - 1) + F(n - 2) & n > 1 
\end{cases}
\]

- Three iterative variants
  - **A** Addition and subtraction in the main loop
  - **B** Addition with one more temporary
  - **C** Addition with an in-situ register swap

```c
uint32 fibo(uint32 x) {
    uint32 f0=0, f1=1, k=2;
    #ifdef B
    uint32 f;
    #endif
    do {
        k = k + 1;
        #if A
        f1 = f1 + f0;
        f0 = f1 - f0;
        #elif B
        f = f1 + f0;
        f0 = f1;
        f1 = f;
        #elif C
        f0 = f1 + f0;
        SWAP(f0, f1);
        #endif
    } while (k <= x);
    #if A || B
    return (f1);
    #else
    return (f0);
    #endif
}
```

C code

Nikolaos Kavvadias nkavvadias@ajaxcompilers.com

Hardware design space exploration using HercuLeS HLS
Fibonacci series example: Machine cycles

<table>
<thead>
<tr>
<th>Design</th>
<th>Cycles</th>
<th>Design</th>
<th>Cycles</th>
<th>Design</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>( n )</td>
<td>B0</td>
<td>( n )</td>
<td>C0</td>
<td>( 2n - 1 )</td>
</tr>
<tr>
<td>A1</td>
<td>( 4n + 3 )</td>
<td>B1</td>
<td>( 5n + 2 )</td>
<td>C1</td>
<td>( 7n + 1 )</td>
</tr>
<tr>
<td>A2</td>
<td>( 4n + 2 )</td>
<td>B2</td>
<td>( 4n + 2 )</td>
<td>C2</td>
<td>( 7n )</td>
</tr>
<tr>
<td>A3-A5</td>
<td>( n + 2 )</td>
<td>B3-B5</td>
<td>( n + 2 )</td>
<td>C3-C5</td>
<td>( 2(n + 1) )</td>
</tr>
</tbody>
</table>

- Hand-optimized designs are A0, B0, C0
- The benefit of ASAP is not significant due to the data dependencies in the algorithm
- Cycle reduction is achieved through operation chaining
- HercuLeS can closely match the result of a human expert for optimization schemes O3-O5
- The slight differences in cycle performance are due to specific design choices of the human expert
  - initializing \( f_0, f_1 \) and \( k \) in the FSMD entry state
  - passing the output data argument without use of an intermediate register
Fibonacci series example: Execution time vs LUTs

- Better results are placed near the bottom-left corner
- Pareto-optimal designs by human expert: B0 and C0
- Pareto-optimal designs by HercuLeS: B2, B3, B4
Fibonacci series example: Execution time vs Registers

- Better results are placed near the bottom-left corner
- Pareto-optimal designs by human expert: A0
- Pareto-optimal designs by HercuLeS: A1 and B3
Benchmark results: Speed

- Average computation time is reduced by 44.3% when comparing O1 to O3.
- This gain is limited to 37.3% for O6 due to block RAM timing.
- `float2half` and `half2float` achieve up to 4x execution time reduction.
- Maximum operating frequencies in the range of 119-450MHz.
Benchmark results: Area

- O1 generates (slower and) smaller hardware in terms of LUTs and registers.
- O3 introduces the highest LUT requirements; O2 the highest register demand.
- Registers are reduced by 17.5% among O2 and O6; LUTs by 14% among O3 and O6.
- Block RAM inference leads to significant LUT/register area reduction (*smwat*).
QoR against commercial competition

- New frontends, analyses and optimizations are easy to add
- Maps character I/O and `malloc/free` to efficient hardware
- Uses open specifications and formats (NAC, Graphviz)
- Vendor and technology-independent HDL code generation
- Preliminary results against Vivado HLS 2013.1

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Vivado HLS</th>
<th>HercuLeS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUTs</td>
<td>Regs</td>
</tr>
<tr>
<td>Array sum</td>
<td>102</td>
<td>132</td>
</tr>
<tr>
<td>Bit reversal</td>
<td>67</td>
<td>39</td>
</tr>
<tr>
<td>Edge detection*</td>
<td>246</td>
<td>130</td>
</tr>
<tr>
<td>Fibonacci series</td>
<td>138</td>
<td>131</td>
</tr>
<tr>
<td>FIR filter</td>
<td>102</td>
<td>52</td>
</tr>
<tr>
<td>Greatest common divisor</td>
<td>210</td>
<td>98</td>
</tr>
<tr>
<td>Cubic root approximation</td>
<td>239</td>
<td>207</td>
</tr>
<tr>
<td>Population count</td>
<td>45</td>
<td>65</td>
</tr>
<tr>
<td>Prime sieve*</td>
<td>525</td>
<td>595</td>
</tr>
<tr>
<td>Sierpinski triangle</td>
<td>88</td>
<td>163</td>
</tr>
</tbody>
</table>
Summary

HercuLeS is an extensible HLS environment for hardware and software engineers

- Straightforward to use from ANSI C, generic assembly (NAC) or custom DSLs (Domain Specific Languages) to producing compact VHDL designs with competitive QoR
- Users can investigate different optimization scenarios regarding speed and area
- Product information
  - HercuLeS GUI for specifying code generation, simulation and synthesis options
  - Commercial distribution: http://www.ajaxcompilers.com
  - Technical details: http://www.nkavvadias.com/hercules
  - FREE, BASIC, and FULL software licensing schemes (2013.a version)
Thank you

for your interest