Domain-specific Optimisation for the High-level Synthesis of CellML-based Simulation Accelerators

Julian Oppermann
Andreas Koch

Ting Yu
Oliver Sinnen
CellML

• Standard to model biomedical problems

• Differential equations describe interaction between components
CellML

- Standard to model biomedical problems
- Differential equations describe interaction between components

\[
alpha_m = \frac{-(V + 47)}{e^{\frac{V+47}{10}} - 1}
\]

\[
beta_m = 40 \times e^{-0.056 \times (V+72)}
\]

\[
\frac{dm}{dt} = alpha_m \times (1 - m) - (beta_m \times m)
\]
• Standard to model biomedical problems

• Differential equations describe interaction between components

\[ \alpha_m = \frac{-(V + 47)}{e^{-\frac{V+47}{10}} - 1} \]

\[ \beta_m = 40 \times e^{-0.056 \times (V+72)} \]

\[ \frac{dm}{dt} = \alpha_m \times (1 - m) - (\beta_m \times m) \]

\[
\begin{align*}
\text{ALGEBRAIC}[1] &= -1.0 \times (\text{STATES}[0] + 47.0) \\
&\quad \div (\exp(-0.1 \times (\text{STATES}[0] + 47.0)) - 1.0); \\
\text{ALGEBRAIC}[8] &= 40.0 \times \exp(-0.056 \times (\text{STATES}[0] + 72.0)); \\
\text{RATES}[1] &= \text{ALGEBRAIC}[1] \times (1.0 - \text{STATES}[1]) \\
&\quad - \text{ALGEBRAIC}[8] \times \text{STATES}[1];
\end{align*}
\]
Hardware-accelerated cell simulation

- Numerical integration
- Cells can be treated independently for some time
- ODoST (Yu et al., 2015): fully-spatial, fully-pipelined FPGA accelerators from a model’s equation system
- Instantiate as many pipelines as fit on the FPGA
Hardware-accelerated cell simulation

- Numerical integration
- Cells can be treated independently for some time
- ODoST (Yu et al., 2015): fully-spatial, fully-pipelined FPGA accelerators from a model’s equation system
- Instantiate as many pipelines as fit on the FPGA

Fewer resources per pipeline  More throughput
Approach

• Fully-spatial computation
  = every SW instruction becomes HW operator

• SW compiler’s architecture independent optimisations
  • eliminate redundant operations, or
  • replace “expensive” ops by “cheaper” ones

• Try unsafe floating-point transformations
Cost model

• Estimation of resource demand → guide opts

• Based on relative, per operator ALM and DSP usage on Stratix IV

\[ c(op) = \left| \left( \frac{n_{ALM}(op)}{212480} \right) \left( \frac{n_{DSP}(op)}{1024} \right) \right| \]

• Allows transformation with a Pareto improvement

• Resulting order of operation costs

Add < Exp < Mul < Div < Log < Pow
Adding LLVM to the mix

- Sequential computation in C generated from CellML equations → idiomatic DSL-like structure
- Use clang/LLVM as frontend
- Optimise on LLVM-IR (existing and custom opts)
- Reconstruct C code for ODoST input
Identifying redundancies

• Array accesses, function calls hinder optimisation

\[ ... = -0.1 \times (\text{STATES}[0]+50.0) / (\exp(-(\text{STATES}[0]+50.0)/10.0) - 1.0); \]

same value?
Identifying redundancies

- Array accesses, function calls hinder optimisation

\[ ... = \frac{-0.1 \times (\text{STATES}[0] + 50.0)}{(\exp(-\frac{\text{STATES}[0] + 50.0}{10.0}) - 1.0)}; \]

- But we know:
  - Input arrays do not alias or overlap
  - Function calls are mathematical operators, side effect-free
Identifying redundancies

• Augment the IR with this domain knowledge to help alias analysis

• Mark input pointers as `noalias`

• Map function calls to LLVM intrinsics

• LLVM’s global value numbering can now identify expressions across the whole equation system

• Equation system ≅ Dataflow graph
Existing optimisation patterns in LLVM

• -instcombine pass
  • Constant folding & algebraic identities
  • Add $<$ Mul $<$ Div in software compiler as well
Existing optimisation patterns in LLVM

• -instcombine pass

• Constant folding & algebraic identities

• Add < Mul < Div in software compiler as well

• Some transformations only if unsafe FP transformations are allowed
  e.g. \( x/c = x \cdot 1/c \) only safe if reciprocal is exact
Domain-specific optimisations
Higher-order powers

- Equations contain $x^p$ with an integer constant
- $8 \cdot c(\text{Mul}) < 1 \cdot c(\text{Pow})$
- Use Knuth’s binary exponentiation method
- lower generic power operator to short sequence of multiplications
- Example: $x^6 = (((x \cdot x) \cdot y) \cdot y) \cdot y$

<table>
<thead>
<tr>
<th>Op</th>
<th>ALM</th>
<th>DSP</th>
<th>$c(\bullet)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul</td>
<td>132</td>
<td>4</td>
<td>0.39</td>
</tr>
<tr>
<td>Pow</td>
<td>2058</td>
<td>31</td>
<td>3.18</td>
</tr>
</tbody>
</table>
A closer look at the exponential function

\[ e^x + c \cdot d \]

Common pattern!

Add < Exp < Mul < …
A closer look at the exponential function

\[ e^x + c \cdot d \]

Add < Exp < Mul < …
A closer look at the exponential function

\[ e^x + c \cdot d \]

- 1 add

- 1 mul

\[ e^x \cdot e^{c \cdot d} \]

- 1 mul, +1 add

\[ e^x + c + \ln(d) \]

- 1 add, +1 mul

Add < Exp < Mul < …
A closer look at the exponential function

\[ e^x + c \cdot d \]

**Add** < **Exp** < **Mul** < …

Best when \( e^x \) can be reused.

Constants underlined.
A closer look at the exponential function

\[ e^x + c \cdot d \]

- 1 add

\[ e^x \cdot e^{c \cdot d} \]

- 1 mul

\[ e^x + c + \ln(d) \]

- 1 mul, +1 add

- 1 add, +1 mul

Best when \( e^x \) can be reused

Add < Exp < Mul < ...

Best otherwise

Best when \( e^x \) can be reused

Add < Exp < Mul < ...

Best otherwise
Multiple Constant Multiplication

- x is multiplied with a set of constants $c_i$
- Can trade 1 multiplication for 1 addition if:
  - $c_2 = 2 \cdot c_1 \quad \rightarrow \quad x \cdot c_2 = (x \cdot c_1) + (x \cdot c_1)$
  - $c_4 = c_3 + 1 \quad \rightarrow \quad x \cdot c_4 = (x \cdot c_3) + x$

- Handle factors in ascending order of absolute values
- Works also for chains of constants, e.g. 2, 3, 4, 6
Results
Compilation flows
Compilation flows

NoOpt: clang → IR → C
Compilation flows

NoOpt:

- Compile C to LLVM-IR
- Reconstruct equations

clang → IR → C
Compilation flows

NoOpt: clang \[\text{IR} \rightarrow \text{C}\]
Compilation flows

NoOpt: clang → IR → C

SWSIZE: clang → opt -Oz → IR → C
Compilation flows

NoOpt:

```
clang → IR → C
```

LLVM's aggressive size optimisation preset

SWSize:

```
clang → opt -Oz → IR → C
```
Compilation flows

NoOpt: \[ \text{clang} \rightarrow \text{IR} \rightarrow \text{C} \]

SWSize: \[ \text{clang} \rightarrow \text{opt -Oz} \rightarrow \text{IR} \rightarrow \text{C} \]
Compilation flows

NoOpt: \[ \text{clang} \rightarrow \text{IR} \rightarrow \text{C} \]

SWSize: \[ \text{clang} \rightarrow \text{opt -Oz} \rightarrow \text{IR} \rightarrow \text{C} \]

SWSizeU: \[ \text{clang -ffast-math} \rightarrow \text{opt -Oz} \rightarrow \text{IR} \rightarrow \text{C} \]
Compilation flows

NoOpt: clang → IR → C

SWSize: clang → opt -Oz → IR → C

SWSizeU: clang -ffast-math → opt -Oz → IR → C

Enable unsafe FP transformations
Compilation flows

NoOpt:  \[ \text{clang} \rightarrow \text{IR} \rightarrow \text{C} \]

SWSize:  \[ \text{clang} \rightarrow \text{opt -Oz} \rightarrow \text{IR} \rightarrow \text{C} \]

SWSizeU:  \[ \text{clang -ffast-math} \rightarrow \text{opt -Oz} \rightarrow \text{IR} \rightarrow \text{C} \]
Compilation flows

NoOpt: 
\[
\text{clang} \rightarrow \text{IR} \rightarrow \text{C}
\]

SWSize: 
\[
\text{clang} \rightarrow \text{opt -Oz} \rightarrow \text{IR} \rightarrow \text{C}
\]

SWSizeU: 
\[
\text{clang -ffast-math} \rightarrow \text{opt -Oz} \rightarrow \text{IR} \rightarrow \text{C}
\]

Ours: 
\[
\text{clang -ffast-math} \rightarrow \text{cellml-opt} \rightarrow \text{IR} \rightarrow \text{C}
\]
Compilation flows

NoOpt: 
- clang 
- IR → C

SWSize: 
- clang 
- opt -Oz 
- IR → C

SWSizeU: 
- clang -ffast-math 
- opt -Oz 
- IR → C

Ours: 
- clang -ffast-math 
- cellml-opt 
- IR → C
Error measurement

• Generic driver, 1000 integration steps of 1 μs each, starting at $t = 1.0$ s

• Compare computed values *before / after optimisation*, calculate relative error

• Certain, model specific deviation is acceptable
  • e.g. precision of “wet biology experiments” $\sim 0.01\%$
Example model

# Operations

- NoOpt
- SWSIZE
- SWSIZEU
- Ours

Operations:
- Add/Sub
- Exp
- Mul
- Div
- Pow
- Other

Example model

- Add/Sub
- Exp
- Mul
- Pow
- Div
- Other

Least total number of operations

<table>
<thead>
<tr>
<th>NoOpt</th>
<th>SWSize</th>
<th>SWSizeU</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example model

- Least total number of operations
- Many generic power operators eliminated

Example model

- **Add/Sub**: Least total number of operations
- **Exp**: Many generic power operators eliminated
- **Mul**: Transform many div to mul when unsafe FP is allowed

## Operations

- **NoOpt**
- **SWSIZE**
- **SWSIZEU**
- **Ours**

Example model

- Add/Sub
- Exp
- Mul
- Div
- Pow
- Other

Operations:
- NoOpt
- SWSize
- SWSizeU
- Ours

Example model

- Add/Sub
- Exp
- Mul
- Div
- Pow
- Other

# Operations

NoOpt | SWSIZE | SWSIZEU | Ours

Usage

NoOpt | SWSIZE | SWSIZEU | Ours

Example model

J. Oppermann: Domain-specific Optimisation for the High-level Synthesis of CellML-based Simulation Accelerators
Example model

Example model

Example model

<table>
<thead>
<tr>
<th>Operations</th>
<th>NoOpt</th>
<th>SWSize</th>
<th>SWSizeU</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add/Sub</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mul</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
<th>NoOpt</th>
<th>SWSize</th>
<th>SWSizeU</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM (est.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALM (syn.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP (est.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP (syn.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SWSIZEU | Ours
---------|-----
Rel. Err [%] | 0.00054 | 0.0012
F_max [MHz] | - | 111

Example model

<table>
<thead>
<tr>
<th># Operations</th>
<th>Add/Sub</th>
<th>Mul</th>
<th>Div</th>
<th>Exp</th>
<th>Pow</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoOpt</td>
<td>550</td>
<td>500</td>
<td>450</td>
<td>400</td>
<td>350</td>
<td>300</td>
</tr>
<tr>
<td>SWSIZE</td>
<td>500</td>
<td>450</td>
<td>400</td>
<td>350</td>
<td>300</td>
<td>250</td>
</tr>
<tr>
<td>SWSIZEU</td>
<td>500</td>
<td>450</td>
<td>400</td>
<td>350</td>
<td>300</td>
<td>250</td>
</tr>
<tr>
<td>Ours</td>
<td>500</td>
<td>450</td>
<td>400</td>
<td>350</td>
<td>300</td>
<td>250</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage (Rel. Err [%])</th>
<th>NoOpt</th>
<th>SWSIZE</th>
<th>SWSIZEU</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALM (est.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALM (syn.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP (est.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSP (syn.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Slightly larger error
General applicability

• 146 models from the CellML repository
  (> 20 equations, operators available as intrinsics, converge in input interval, 2+ curation stars)

• 4 thresholds for maximum relative error per model

• Use the cost model to estimate impact of transformations
Least cost per flow

- Count models with least cost after optimisation with a given flow
- If error > threshold, fall-back to SWSafe
Least cost per flow

- Count models with least cost after optimisation with a given flow
- The larger the acceptable deviation, the more models benefit from our flow
- If error > threshold, fall-back to SWSafe
Summary

• Size reduction after synthesis in 4 example models
  • Our recipe: up to 25 % less ALM, 20 % less DSP
  • Never worse than unoptimised (c.f. other flows)
• Broad applicability for domain-specific optimisations across 146 models
Future work

• Cost model served us well as quantitative instrument
  • Estimation of DSP usage ok
  • More accurate estimation of ALM demand needed
• A priori error analysis instead of empirical study
Thank you!