A Fast and Accurate Cost Model for FPGA Design Space Exploration in HPC Applications

Developing an optimizing compiler for running scientific code on FPGAs

S Waqar Nabi and Wim Vanderbauwhede
School of Computing Science,
University of Glasgow
Motivation
FPGAs for High Performance Computing (HPC)?

- FPGAs *now* beginning to be used in mainstream computing
  - big-data and big-compute (HPC)

- *Can* provide better FLOPS/Watt for some types of applications

- Difficult to tune even with today’s HLS tools like OpenCL, Maxeler, etc
  - E.g. 3 days vs 3 weeks

- **A still higher-level approach** is needed
  - automatic architectural exploration of the FPGA design-space
  - make FPGAs more accessible to scientists/HPC users

- The argument can be generalized to **heterogeneous** computing targets

*Lots of promise, encouraging recent developments, but miles to go still...*
• FPGAs now beginning to be used in mainstream computing
  – big-data and big-compute (HPC)

• Can provide better FLOPS/Watt for some types of applications

Today’s high-level language is tomorrow’s compiler target

• A still higher-level approach is needed
  – automatic architectural exploration of the FPGA design-space
  – make FPGAs more accessible to scientists/HPC users

• The argument can be generalized to heterogeneous computing targets

Lots of promise, encouraging recent developments, but miles to go still...
Why are we working on an FPGA cost-model?

- Our proposed TyTra compiler flow requires evaluation of multiple design-variants, in order to converge on the best one.

- It requires a light-weight, reasonably accurate cost-model.

A light-weight cost-model is the linchpin of our proposed FPGA optimizing compiler flow.
OUTLINE OF TALK
Towards an optimizing compiler for running scientific code on FPGAs

1. The TyTra Framework
   - Compiler flow
   - The Intermediate Representation (IR)
   - Need for a cost-model

2. Developing a Cost-Model
   - Models of abstraction
   - Cost-model for resource-utilization and performance

3. Observations
   - Results: design-space exploration potential, accuracy of cost-model, potential for improved performance
   - Limitations
   - The way forward

An auto-tuning programming approach, for scientific computing, requiring a fresh approach to cost-modelling
THE TYTRA FRAMEWORK
Blue Sky Target

Make HPC on “exotic” architectures more accessible to scientists

Optimized HPC solution!

- Legacy Scientific Code
- Cost Model
- Heterogeneous HPC Target
I have a cunning plan.
1. Use the **functional programming paradigm**
   - *type-transformations* to create design-variants

2. Have an **Intermediate Language** that can:
   - express the design-space
   - be costed directly and quickly

3. Create a **light-weight cost-model** that can estimate:
   - performance
   - resource-utilization

*Exploit the “elegance” of functional abstraction to generate equivalent design variants, then lower it to an IR that can be costed.*
Key contributions

Following on from the cunning plan

1. Type transformations for generating program variants

2. A new(ish) intermediate language based on LLVM, and

3. A light-weight cost-model

Generating variants, and connecting them to a cost-model (and generator) via an appropriate Intermediate Representation
The TyTra Flow

What’s keeping us busy these days

- Refactored Fortran Code
- Legacy Fortran Scientific Code

this work
Key contributions

1. Type transformations for generating program variants

2. A new(ish) intermediate language based on LLVM, and

3. A light-weight cost-model

Generating variants, and connecting them to a cost-model (and code-generator) via an appropriate Intermediate Representation
Type Transformations

Why Functional Programming

- Describe **what something is**, not **what to do**
  - *not imperative*
  - *no “side-effects”*

- High-level **types** that describe functions as well as variables

- Transformation of **vector-types** can be done in a provably correct manner

- **Type-transformations** translate to **design-variants** on the FPGA

*A functional paradigm with high-level functions allows creation of design-variants that are correct-by-construction.*
Vector Types

- typeA : Vect im int  -- 1D integer vector sized im

- typeB : Vect km (Vect im int)  -- transformed 2D data

Program Variants

- output = map\_pipe kernel\_func input  -- original program

- inputTra = reshapeTo km input  -- reshaping data

- output = map\_par (map\_pipe kernel\_func) inputTra  -- new program

Simple and provably correct vector transformations in the functional paradigm enable generation of "program-variants"
The program-variants from high-level transformation translate into design-variants on the FPGA.
Key contributions

1. Type transformations for generating program variants

2. A new(ish) intermediate language based on LLVM, and

3. A light-weight cost-model

Generating variants, and connecting them to a cost-model (and generator) via an appropriate Intermediate Representation
Design-variants are lowered into an Intermediate-Representation, making it easier to estimate cost, performance, and then generate HDL code.
Nesting functions of types pipe, par, seq and comb in different combinations enables expression of different design configurations.
Now that we have design variants…

How do we know which variants are valid (fit on the FPGA)?

How do we know which one performs the best?
DEVELOPING THE COST-MODEL
A set of standardized experiments for each new target feeds empirical data to the cost model, and the rest comes from the IR description.
Pre-requisite: Models Of Abstraction

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space and cost-space model
5. Memory execution model
6. Data access pattern model

*Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space*
Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space.
Platform And Memory Model
Pre-requisite: Models Of Abstraction

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. Data access pattern model

*Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space*
A way to look at the design-space for FPGA implementation. This still does not
Pre-requisite: Models Of Abstraction

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. Data access pattern model

Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space
The manner in which the FPGA memory-hierarchy is accessed across the execution of an application has a huge impact on performance.
Pre-requisite: Models Of Abstraction

1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. Data access pattern model

Models of Abstraction needed to have a systematic way to reason about the complex FPGA-design space
1. Platform model
2. Memory hierarchy model
3. Execution model
4. Design-space model
5. Memory execution model
6. Data access pattern model
   - Contiguous access
   - (Fixed) Strided access

The data-access pattern has significant impact on performance of memory-bound applications
Two Types of Estimates

- **Resource-Utilization Estimates**
  - ALUTs, REGs, DSPs

- **Performance Estimates**
  - Memory-bound or compute-bound?
    - **Memory-bound**: The sustained memory bandwidth
    - **Compute-bound**: FPGA pipeline throughput

Both estimates needed to allow compiler to choose the best design variant.
• Estimate cost of primitive instructions
  – Instructions should be cost-able across valid data types

• Accumulate costs based on parallelism configuration
  – which is expressed by nesting of functions of types par, pipe, seq
Light-weight cost expressions associated with every legal SSA instruction in the TyTra-IR, e.g. integer division
Light-weight cost expressions associated with every legal SSA instruction in the TyTra-IR, e.g. integer multiplication
• **Effective Work-Instance Throughput (EWIT)**
  - Work-Instance = Executing the kernel over the entire index-space

• **Key Determinants**
  - Memory execution model
  - Sustained memory bandwidth for the target architecture and design-variant
  - Data-access pattern
  - Design configuration of the FPGA
  - Operating frequency of the FPGA
  - Compute-bound or IO-bound?

*The performance estimate requires design to be classified based on the abstractions we developed earlier*
Performance Estimates

The Expressions

\[ EWUT_A = \frac{N_{GS} \cdot N_{WPT}}{H_{PB} \cdot \rho_H} + \frac{N_{off}}{G_{PB} \cdot \rho_G} + \frac{K_{PD}}{F_D} + \max\left(\frac{N_{GS} \cdot N_{WPT}}{G_{PB} \cdot \rho_G}, \frac{N_{GS} \cdot N_{WPT} \cdot N_{TO} \cdot N_{I}}{F_D \cdot K_{NL} \cdot D_{V}}\right) \]

\[ EWUT_B = \frac{N_{GS} \cdot N_{WPT}}{N_{WU} \cdot H_{PB} \cdot \rho_H} + \frac{N_{off}}{G_{PB} \cdot \rho_G} + \frac{K_{PD}}{F_D} + \max\left(\frac{N_{GS} \cdot N_{WPT}}{G_{PB} \cdot \rho_G}, \frac{N_{GS} \cdot N_{WPT} \cdot N_{TO} \cdot N_{I}}{F_D \cdot K_{NL} \cdot D_{V}}\right) \]

\[ EWUT_C = \frac{N_{GS} \cdot N_{WPT}}{N_{WU} \cdot H_{PB} \cdot \rho_H} + \frac{N_{off}}{G_{PB} \cdot \rho_G} + \frac{K_{PD}}{F_D} + \max\left(\frac{N_{GS} \cdot N_{WPT} \cdot N_{TO} \cdot N_{I}}{F_D \cdot K_{NL} \cdot D_{V}}\right) \]
Parameters that Make up the Expression

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Key Dependence</th>
<th>Short Description</th>
<th>Evaluation Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{PB}$</td>
<td>Node architecture</td>
<td>The host-device peak bandwidth (typically PCI Express).</td>
<td>Architecture description fed to compiler</td>
</tr>
<tr>
<td>$\rho_H$</td>
<td>Node architecture &amp; design-variant</td>
<td>Scaling factor, host-device bandwidth</td>
<td>Experiments with different data-patterns on the target node.</td>
</tr>
<tr>
<td>$G_{PB}$</td>
<td>Node architecture</td>
<td>The device DRAM peak bandwidth.</td>
<td>Architecture description fed to compiler</td>
</tr>
<tr>
<td>$\rho_H$</td>
<td>Device architecture &amp; design-variant</td>
<td>Scaling factor, host-device bandwidth</td>
<td>Experiments with different data-patterns on the target node.</td>
</tr>
<tr>
<td>$N_{GS}$</td>
<td>Program</td>
<td>Global-size of work-items in NDRange</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$N_{WPT}$</td>
<td>Program</td>
<td>Words per tuple per work-item</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$N_{NDR}$</td>
<td>Program</td>
<td>Repetition of kernel over NDRange</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$N_{off}$</td>
<td>Program</td>
<td>Maximum offset in a stream</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$K_{PD}$</td>
<td>Program &amp; design-variant</td>
<td>Pipeline depth of kernel</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$F_D$</td>
<td>Program &amp; design-variant</td>
<td>Device’s operating frequency</td>
<td>Compiler costing of IR</td>
</tr>
<tr>
<td>$N_{TO}$</td>
<td>Program &amp; design-variant</td>
<td>Cycles per instruction</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$N_I$</td>
<td>Program &amp; design-variant</td>
<td>Instructions per PE</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$K_{NL}$</td>
<td>Program &amp; design-variant</td>
<td>Number of parallel kernel lanes</td>
<td>Compiler parse of IR</td>
</tr>
<tr>
<td>$D_V$</td>
<td>Program &amp; design-variant</td>
<td>Degree of vectorization per lane</td>
<td>Compiler parse of IR</td>
</tr>
</tbody>
</table>

The variables that make up the expression for estimating performance (throughput) are either directly available from IR, or require an empirical model.
Effect of Access Pattern with Different Array Sizes

An illustration showing impact of data-access pattern on an sdaccel programmed Alpha-Data FPGA board with Virtex 7 device.
Observations and Results
Preliminary results show estimated vs actual values are quite close. Frequency estimate is still a tricky (hence Clocks Per Work-Instance)

<table>
<thead>
<tr>
<th>Kernel</th>
<th>ALUT</th>
<th>REG</th>
<th>BRAM</th>
<th>DSP</th>
<th>CPWI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotspot (Rodinia)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated</td>
<td>391</td>
<td>1305</td>
<td>32.8K</td>
<td>12</td>
<td>262.3K</td>
</tr>
<tr>
<td>Actual</td>
<td>408</td>
<td>1363</td>
<td>32.7K</td>
<td>12</td>
<td>262.1K</td>
</tr>
<tr>
<td>% error</td>
<td>4</td>
<td>4.2</td>
<td>0.3</td>
<td>0</td>
<td>0.07</td>
</tr>
<tr>
<td>LavaMD (Rodinia)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated</td>
<td>408</td>
<td>1496</td>
<td>0</td>
<td>26</td>
<td>111</td>
</tr>
<tr>
<td>Actual</td>
<td>385</td>
<td>1557</td>
<td>0</td>
<td>23</td>
<td>115</td>
</tr>
<tr>
<td>% error</td>
<td>6</td>
<td>3.9</td>
<td>0</td>
<td>13</td>
<td>3.4</td>
</tr>
<tr>
<td>SOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estimated</td>
<td>528</td>
<td>534</td>
<td>5418</td>
<td>0</td>
<td>292</td>
</tr>
<tr>
<td>Actual</td>
<td>534</td>
<td>575</td>
<td>5400</td>
<td>0</td>
<td>308</td>
</tr>
<tr>
<td>% error</td>
<td>1.1</td>
<td>7.1</td>
<td>0.3</td>
<td>0</td>
<td>5.2</td>
</tr>
</tbody>
</table>
Does the TyTra approach work?
The requirement of the cost-model to be light-weight is very important if we want to evaluate many design-variants.
Estimates for multiple variants allows us to converge on the best option, and can also give optimization hint back to the compiler/programmer.
On-going Challenges

Memory Bandwidth Estimates

- Estimating the memory bandwidth for:
  - a particular application
  - configured as a particular design-variant
  - being compiled for a particular HPC target platform

- If estimate show application will be memory-bound, then:
  - Can we optimize memory access to get better overall performance?

- We are currently working on a memory-bandwidth benchmark for FPGAs
Limitations → Future Work

- Experiment with simple kernels
- Cost-model currently for integers only
- (Lack of) Re-usable, user-friendly and publicly available benchmarks
- Non-optimized number representations
- No automated integration of generated HDL code with HLS tools
  - Manually we have integrated our generated code with Maxeler HLS
- Estimating resources for memory controllers/base platform
  - Also, more accurate estimates of frequency

Stay tuned…
Auto-tuning scientific computing for FPGAs - A fresh approach to cost-modelling

1. A functional language paradigm based TyTra Framework
   - Type transformations, variants, IR and need for a cost model

2. Making a light-weight cost-model
   - Models of Abstractions, the cost model for resources and performance, the key variables

3. Experimental results and observations
   - Accuracy, exploration, potential for improvement
   - Limitations and the way forward

Towards an optimizing compiler for running scientific code on FPGAs
The woods are lovely, dark and deep,
But I have promises to keep,
And lines to code before I sleep,
And lines to code before I sleep.

syed.nabi@glagow.ac.uk
http://dcs.gla.ac.uk/~waqar/