Generation of Multigrid-based Numerical Solvers for FPGA Accelerators

Christian Schmitt‡, Moritz Schmid‡, Frank Hannig‡, Jürgen Teich‡, Sebastian Kuckuk†, Harald Köstler†
‡Hardware/Software Co-Design, †System Simulation,
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
HiStencils, Amsterdam, The Netherlands; January 20, 2015
Motivation

Multigrid methods are widely used

- Solution of discretized PDEs
- Preconditioners for other iterative solvers

and highly scalable

- $O(N)$ operations
- but also on a device scale: from embedded hardware to supercomputers!

But: Most efficient implementation varies greatly on numerical problem and target architecture!

Our research

Generation of multigrid-based solvers and automatic application of domain- and target-specific optimizations
Motivation

Code generator works for supercomputers, e.g., on JUQUEEN (TOP500 #8)\(^1\):

![Graph showing total runtime vs. number of cores](image)

but can it also work on the other end of the scale, i.e., for energy-efficient embedded devices such as FPGAs?

Basic Multigrid Ideas

Multigrid method

1. Pre-smoothing
2. Calculation of residual
3. Restriction
4. Recursive call(s) or solve (at coarsest level)
5. Prolongation
6. Correction
7. Post-smoothing
Basic Multigrid Ideas

Residual on fine grid

Residual on coarse grid

Smother applied
FPGA Basics

- Field Programmable Gate Arrays
- Array of lookup tables and registers → configurable logic blocks (CLBs)
- Switch matrices to connect CLBs
- Trade-off between performance of hardware (ASIC) and flexibility of software
- Programming via Hardware Description Language, e.g., VHDL, Verilog
FPGA Basics

Spread across chip: Hard-IP cores

- **Block RAM**
  - Distributed memory
  - ~1000, 1-2 kB each
  → Very high memory bandwidth

- **DSP Blocks**
  - Dedicated multiplier/adder units
  - Typically $16 \times 16$ bit
  - Clock: ~500 MHz

- **High-speed serial I/O**
  - PCIe hard-IP for communication with host PC
  - Off-chip DDR3
  - Soft IP support for various protocols (Infiniband, Ethernet, . . .)
FPGA Basics

Basic principle: Spatial Computing, e. g., stream processing

Temporal Computing: Sequential execution

Spatial Computing: Parallel execution

(time)

(time)
FPGA Basics

Traditional Workflow

- Hand-coding HDL
- Register Transfer Level (RTL)
- Post Place & Route (PPnR)
- Upload configuration file to FPGA

High-level Synthesis (HLS)

- Behavioral description (algorithm, math. model), often in a subset of C/C++
- Conversion to structured description
  - Register-transfer level (RTL)
  - Connected blocks

```
struct Smoother_1Kernel {
    double operator()(double rhsData_1[3][3], double solutionData_1[3][3][3]) {
        double temp1 = 4.0f*solutionData_1[1][1] - solutionData_1[2][1] - solutionData_1[0][1] - solutionData_1[1][2];
        double temp2 = rhsData_1[1][1] - temp1;
        double temp3 = solutionData_1[1][1] + temp2 / 2;
        return temp3;
    }
};
void Smoother_1(hls::stream<double>& rhs_in, hls::stream<double>& data, hls::stream<double>& sol, hls::stream<double>& rhs_out) {
    struct Smoother_1Kernel Smoother_1_inst;
    processMIMO<16384, 32, 32, 3>(rhs_in, data, sol, rhs_out, 32, 32, Smoother_1_inst, BorderPadding::BORDER_CLAMP);
}
```
ExaSlang

- Multi-layered DSL
- Description of multigrid-based numerical solvers
- Layer 4 aimed at computer scientists
  - Explicitly parallel by providing simple communication statements
  - Definition of “arrays”, stencils, loops
  - Explicit addressing of different multigrid levels

Level Specifications

Referencing of multigrid levels

- Absolute: @0, @coarsest
- Relative: @current, @coarser, @finer

Used to implement multigrid recursion.
ExaSlang

Fields and Layouts

- Represent multi-dimensional arrays
- Size(s) determined automatically
- Layout determines datatype, communication
- Multiple fields can have the same layout

```plaintext
Layout NoComm <Real> @all {
    ghostLayers = [0, 0]
    duplicateLayers = [1, 1]
}
Field Solution <global, NoComm, 0.0> @all
```
ExaSlang

Stencils

\[
\text{Stencil Laplace @all} \{ \\
[0, 0] \Rightarrow 4.0, \quad [1, 0] \Rightarrow -1.0, \quad [-1, 0] \Rightarrow -1.0 \\
[0, 1] \Rightarrow -1.0, \quad [0, -1] \Rightarrow -1.0
\}
\]

Loops

\[
\text{loop over Solution @current} \{ \\
\text{Solution2 @current} = \text{Solution @current} + (((1.0 / \text{diag(Laplace @current)}) \times 0.8) \times (\text{RHS @current} - \text{Laplace @current} \times \text{Solution @current}))
\}
\]

- Bounds of loop determined by field
- One loop can be mapped to one kernel function
- Arguments for function via dependency analysis
Mapping to FPGAs

- Resolve stencil applications per multigrid levels
- Map `loop over` statements to separate IP core
- Dependency analysis: Add fields to IP core inputs/outputs
- Calculate field (stream) sizes for IP core
- Replace `loop over` statements with `process` statements
- Connect IP cores with streams and insert copy/split kernels to duplicate streams
- Add iteration intervals from simulation
- Resource sharing optimizations
Mapping to FPGAs

Kernels connected via streams:

FIFO buffers needed between cores for different stream sizes, i.e., downsampling and upsampling
Mapping to FPGAs

Stages can be reused due to FIFO buffering:
Results

Setup

- $V(2,2)$ solver for Poisson's equation
- 8 multigrid levels fixed
- Jacobi smoothers
- Jacobi applied multiple times for coarse grid solving
- Input grid size of $4096 \times 4096$

High-level synthesis

- Xilinx Vivado HLS v14.2
- Small support library to help with IP core instantiation
- Buffer sizes calculated using external simulation tool

Results

Resource usage on FPGAs for double precision:

<table>
<thead>
<tr>
<th>FPGA</th>
<th>LUTs</th>
<th>FFs</th>
<th>DSPs</th>
<th>BRAMs</th>
<th>$F_{\text{max}}$[MHz]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kintex-7</td>
<td>140%</td>
<td>43%</td>
<td>111%</td>
<td>124%</td>
<td>232.0</td>
</tr>
<tr>
<td>Virtex-7</td>
<td>73%</td>
<td>29%</td>
<td>33%</td>
<td>53%</td>
<td>229.4</td>
</tr>
</tbody>
</table>

- Sharing of stages due to FIFO buffers
- Increase coarser stages’ iteration intervals (II) for resource sharing
  - Double precision not possible for Kintex-7
  - More stages could be added for Virtex-7

\(^3\) Estimation by Vivado HLS. Place & Route not possible due resource constraints.
\(^4\) PPnR Result
Results

Performance figures for a single V-cycle

<table>
<thead>
<tr>
<th>Target</th>
<th>Runtime [ms]</th>
<th>Throughput [Vps$^5$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA$^6$</td>
<td>83.1</td>
<td>12.3</td>
</tr>
<tr>
<td>Intel i7$^7$</td>
<td>223.1</td>
<td>4.5</td>
</tr>
</tbody>
</table>

$^5$V-cycles per second

$^6$Performance is the same for Kintex-7 (XC7VX485T) and Virtex-7 (XC7K325T). Single precision on Kintex-7, double precision on Virtex-7.

$^7$Intel i7-3770, 3.40 GHz, single thread. Besides AVX, no optimization where applied by our code generator. Double precision. Code is memory-bandwidth bound. Compiled with -O3.
Summary

Conclusions

• Code generation for FPGAs based on HDL
• ExaStencils code generator flexible enough to emit code for a fundamentally different computing model
• Performance of mid-range FPGAs already promising

Future work

• Research (algorithmical) optimization potential
• Smarter grid traversal for 3D
• Automatic calculation of buffer sizes
• Partitioning among multiple FPGA boards
• Automatic application of HLS and hardware synthesizing
• Automatic re-configuration at runtime if convergence prediction insufficient
Thanks for listening.  Questions?

ExaStencils
ExaStencils – Advanced Stencil Code Engineering
http://www.exastencils.org
ExaStencils is funded by the German Research Foundation (DFG)
as part of the Priority Program 1648 (Software for Exascale Computing).