Progress in Porting the LFRic Weather and Climate model to FPGAs using C and OpenCL

Mike Ashworth\textsuperscript{1}, Sergi Siso\textsuperscript{2}, Graham Riley\textsuperscript{1}, Rupert Ford\textsuperscript{2}, Andrew Porter\textsuperscript{2}

\textsuperscript{1} Advanced Processor Technologies Group, Department of Computer Science, University of Manchester, United Kingdom

\textsuperscript{2} The Hartree Centre, STFC Daresbury Laboratory, Warrington, United Kingdom

mike.ashworth.compisci@manchester.ac.uk
Talk Outline

• Update on the matrix-vector kernel (MA, GR)

• Implementation for two kernels of LFRic (MA, GR)

• OpenCL kernels on FPGAs (GR)

• Performance portability with PSyclone and OpenCL (SS, AP, RF)
Project outline

Horizon 2020 FETHPC-01-2016:

**Co-design of HPC systems and applications**

EuroExa started 1st Sep 2017, runs for 3½ years
16 Partners, 8 countries, €20M
Builds on previous projects, esp. ExaNoDe, ExaNeSt, EcoScale

Aim: design, build, test and evaluate an Exascale prototype
Architecture based on ARM CPUs with FPGA accelerators
Three testbed systems: #3 will deliver 2.4 Pflop/s peak
Scalable to 400 Pflop/s at high Gflop/s/W
Low-power design goal to target realistic Exascale system
Architecture evolves in response to application requirements
= co-design

Wide range of apps, incl. weather forecasting, lattice Boltzmann, multiphysics, astrophysics, astronomy data processing, quantum chemistry, life sciences and bioinformatics

@euroexa

euroexa.eu

Kick-off meeting 4th-5th Sep 2017, Barcelona
Brand new weather and climate model: LFRic named after Lewis Fry Richardson (1881-1953)

- Dynamics from the GungHo project 2011-2015
- Scalability – globally uniform grid (no poles)
- Speed – maintain performance at high & low resolution and for high & low core counts
- Accuracy – need to maintain standing of the model
- Separation of Concerns – PSyclone generated layer for automated targeting of architectures
- Operational weather forecasts around 2022 – anniversary of Richardson (1922)
Motivation

• Field Programmable Gate Array (FPGA) is “a matrix of configurable logic blocks connected via programmable interconnects”
• FPGAs offer large gains in performance/W and /$
• Natural route to reduced precision
• Major corporations are using FPGAs in datacentres for cloud services, analytics, communication, etc.
• Hardware traditionally led by Xilinx (ARM CPU + FPGA single chip)
• Intel’s acquisition of Altera led to Heterogeneous Architecture Research Platform (HARP) (also single chip)
• Predictions: up to 30% of datacenter servers will have FPGAs by 2020
Three Steps to (FPGA) Heaven

1. Compile C kernels using Vivado High Level Synthesis -> IP blocks

2. Lay out the design with your IP blocks and built-in IP using Vivado Design Suite -> bitstream

3. Write code to drive the FPGA kernels from the CPU code (Fortran 2003)
FPGA kernels with Vivado HLS – matrix-vector multiplication

Performance Estimate:
• Target 2ns clock: design validated at 2.89ns = 346 MHz
• 2334 cycles for 3840 flops = 1.65 flops/cycle
• Overlapped dmul with dadd
• Starting code was 69841 cycles

Utilization Estimate:
• Try to maximize performance while minimizing utilization
• Shows percentage of chip ‘real-estate being utilized
• Setup two devices /dev/uio0 and /dev/uio1 – two ports on the ZynQ IP block
• Use mmap to map the FPGA memory into user space
• Assign pointers for each data array to location in user space
• For each “chunk” of cells:
  • Assign work to one of the matrix-vector blocks
  • Copy input data into BRAM
  • Set the control word “registers” for the block
  • Start the block by setting AP_START
  • Wait for block to finish by watching AP_IDLE (opportunity for overlap)
  • Copy output data from BRAM
• In practice we fill the whole BRAM, then run all 12 matrix-vector blocks, then copy output data back and repeat

Maintain the LFRic “spirit”: Standard Fortran 2003 using ISO C Interface
Why you should not throw up your hands in horror!

This is far too low-level for me!

…. but ….

• The beauty of the PSyclone approach in LFRic means all this can be hidden from the scientist

• Programming models are developing, becoming easier to use, e.g. OpenCL with HLS

• We are demonstrating capability using low-level tools
LFRic Matrix-Vector Kernel Performance

- Best performance 5.3 Gflop/s
- 510 Mflop/s per block => 1.53 flops/cycle (93% of HLS estimate)
- Parallel efficiency at 12 IP blocks 87%
- Clock scaling 100 to 333 MHz is 94% efficient
- ARM Cortex A53 single core 177 Mflop/s
- ARM quad-core with OpenMP 615 Mflop/s approx.
- FPGA:ARM quad-core speed-up: 8.6x
LFRic Matrix-Vector Kernel - critical performance factors

- Performance of single matrix-vector block
- Clock speed
- Number of matrix-vector blocks
<table>
<thead>
<tr>
<th>Hardware</th>
<th>Matrix-vector performance (Gflop/s)</th>
<th>Peak performance (Gflop/s)</th>
<th>Percentage peak</th>
<th>Price</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZCU102 FPGA</td>
<td>5.3</td>
<td>600</td>
<td>0.9%</td>
<td>$</td>
<td>W</td>
</tr>
<tr>
<td>Intel Broadwell E5-2650 v2 2.60GHz 8 cores</td>
<td>9.86</td>
<td>332.8</td>
<td>3.0%</td>
<td>$$$</td>
<td>WWW</td>
</tr>
</tbody>
</table>

- FPGA performance is 54% of Broadwell single socket
- Should be scaled by price & power
LFRic Matrix-Vector Kernel - discussion

- Performance-price and performance/power
  - GPU: 0.07-0.12 vs. FPGA: 0.23 €/Gflop/s/W
  - GPU: 20 vs. FPGA: 70 Gflops/W
  - FPGAs have a large benefit in power efficiency

- Matrix-vector (MVM) vs. matrix multiply (MXM)
  - For large N, MVM asymptotically approaches computational intensity (CI) of 0.25 flops/byte
  - MXM has a computational intensity of N/12, so even for small matrices (12x12) CI is one flop/byte
  - Matrix-vector is much harder than matrix-multiply

Ashworth et al, “First steps in porting the LFRic Weather and Climate model to the FPGAs of the EuroExa architecture”, Scientific Programming, in press 2019
Implementation in LFRic – intercepting LFRic kernels

- Simply intercept the single-cell kernel
  - e.g. call opt_apply_variable_hx_code
  - target options: Fortran, C, FPGA

- Or replace the loop over cells by a multi-cell call
  - e.g. call multicell_apply_variable_hx_code (1, mesh%get_last_edge_cell(), ...)
  - an obvious optimisation for many architectures
Implementation in LFRic – multiple kernels

• Typical LFRic workload
  Kernel 1 (e.g. apply_variable_hx_code)
  Halo exchange for variable x1
  Kernel 2 (e.g. matrix_vector_code)
  Halo exchange for variable x2

• Implement multiple IP blocks in the Vivado design

• Communicate on-chip via BRAM memory
  TBD

• Only halos sent between CPU & FPGA for MPI

• EuroExa partners working on FPGA-FPGA MPI comms
OpenCL on FPGAs

• OpenCL high-level benefits
  • OpenCL’s execution and memory model is a close match for FPGAs
  • High-level programming interface e.g. SDSoC, SDAccel
  • Partial reconfiguration for dynamic management of kernels *

• Exploring design optimisation space
  • OpenCL host parallelism through command-queues
  • FPGA deployment options and kernel optimisations

• Context of MPI and threads
  • EuroExa TestBed0 in Manchester: 8 x ZYNQ UltraScale+

* Pham et al, “ZUCL: A ZYNQ UltraScale+ Framework for OpenCL HLS Applications”, FSP Workshop 2018
PSyclone aims to provide performance portability while maintaining a good separation of concerns between the science and the computational domains.

New OpenCL back-end to target FPGAs from the same front-end Fortran code.
Hartree is using NemoLite2D (GOcean front-end) as initial example for the OpenCL back-end:

- Vertically averaged version of the dynamical free-surface part of NEMO. It uses a structured grid and the explicit Eulerian forward time stepping method.
- It captures the essence of a real application and it is relatively complex for an FPGA application, time stepping contains 11 kernels with a total of ~300 LOC.
- For now, the GOcean front-end is the only one supported by the OpenCL back-end.
• OpenCL driver layer: host code controls execution of OpenCL kernels. PSyclone generates Fortran code that calls the OpenCL API using the interface provided by the FortCL library [github.com/stfc/FortCL](https://github.com/stfc/FortCL)

• OpenCL Kernels: device code written in OpenCL. Using the PSyIR language-independent representation of the kernels, PSyclone is able to generate an OpenCL version of each kernel

*Simplified subroutine

```c
Schedule[name='compute_cu_code']
Assignment[]
ArrayReference[name='cu']
  Reference[name='i']
  Reference[name='j']
BinaryOperation[operator='MUL']
  BinaryOperation[operator='ADD']
    Literal[value='0.5D0']
  BinaryOperation[operator='SUB']
    Reference[name='i']
    Literal[value='1']
  Reference[name='j']
ArrayReference[name='u']
  Reference[name='i']
  Reference[name='j']

__attribute__((vec_type_hint(double)))
__attribute__((reqd_work_group_size(4, 1, 1)))
__kernel void compute_cu_code(
  __global double * restrict cu,
  __global double * restrict p,
  __global double * restrict u
){
  int cuLEN1 = get_global_size(0);
  int cuLEN2 = get_global_size(1);
  int pLEN1 = get_global_size(0);
  int pLEN2 = get_global_size(1);
  int uLEN1 = get_global_size(0);
  int uLEN2 = get_global_size(1);
  int i = get_global_id(0);
  int j = get_global_id(1);
  cu[j * cuLEN1 + i] = ((0.5e0 * (p[j * pLEN1 + i] + p[j * pLEN1 + (i - 1)])) * u[j * uLEN1 + i]);
}
```
Initial OpenCL results

Initial results on a Xilinx U200 FPGA PCIe card.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Xilinx U200</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs (K)</td>
<td>892</td>
</tr>
<tr>
<td>Registers (K)</td>
<td>1831</td>
</tr>
<tr>
<td>BRAM (36 Kb blocks)</td>
<td>1766</td>
</tr>
<tr>
<td>RAM (288 Kb blocks)</td>
<td>800</td>
</tr>
<tr>
<td>DSP slices</td>
<td>5867</td>
</tr>
</tbody>
</table>

* Current implementation underutilizes the available resources. Only ~20% of FPGA being used.
Future work to close the performance gap

• Blocking
  Aggregating multiple work-items in a single kernel call could improve the performance. OpenCL provides the *local-work-size* parameter to perform this operation.

• Exploit functional parallelism
  At the moment we just use 1 in-order queue. But we know some of the kernels could be executed concurrently using multiple OpenCL queues.

• Fuse kernels
  Generate a more stream-based implementation by fusing kernels that are executed consecutively and/or using OpenCL channels.

• Learn from experience optimising LFRic kernels for FPGAs (UoM)
Summary

- A matrix-vector kernel implementation using Vivado HLS runs on the UltraScale+ FPGA at 5.3 double precision Gflop/s (single precision: similar performance, 63% resources)
- LFRic is running with two kernels offloaded to FPGA
- We are comparing the low-level Vivado route to a high-level OpenCL programming method
- PSyclone is capable of generating OpenCL code to target a wider range of architectures incl. FPGAs
Many thanks
Please connect at
@euroexa or euroexa.eu

Mike Ashworth¹, Sergi Siso², Graham Riley¹, Rupert Ford², Andrew Porter²

¹ Advanced Processor Technologies Group,
Department of Computer Science,
University of Manchester, United Kingdom

² The Hartree Centre, STFC Daresbury Laboratory,
Warrington, United Kingdom

mike.ashworth.compsci@manchester.ac.uk