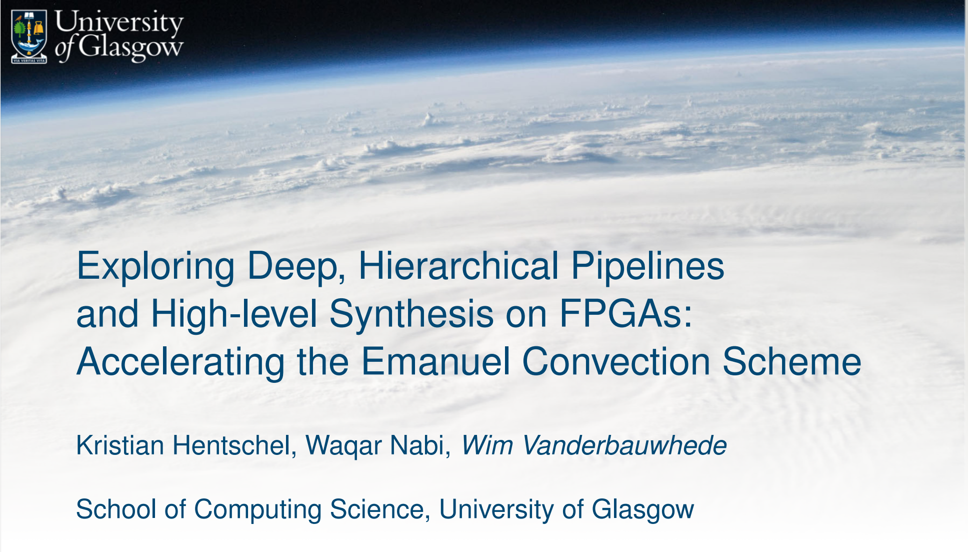


Exploring Deep, Hierarchical Pipelines and High-level Synthesis on FPGAs: Accelerating the Emanuel Convection Scheme

Kristian Hentschel, Waqar Nabi, *Wim Vanderbauwhede*

School of Computing Science, University of Glasgow

University of Glasgow

Introduction

- The research question:
  Is it possible to efficiently port complete, large, sequential floating point algorithms to FPGAs using a high-level programming approach?
- The objectives:
  - Create a complete, performant, efficient FPGA port of the Emanuel convection kernel.
  - Develop legacy code analysis strategy.
  - Explore pipeline parallelism in terms of design patterns.

High-level FPGA Programming

- Best performance for FPGAs requires writing Verilog or VHDL.
- But there are now many high-level approaches, e.g. Maxeler, OpenCL, Vivado, Leg-Up, ...
- We chose Maxeler for this particular work because we wanted to explore the potential of the stream paradigm in detail.
- This work is part of a project to develop a compiler from legacy scientific code to heterogeneous platforms, including FPGAs.

Typical FPGA Acceleration Approach

(Or at least, the sales pitch)

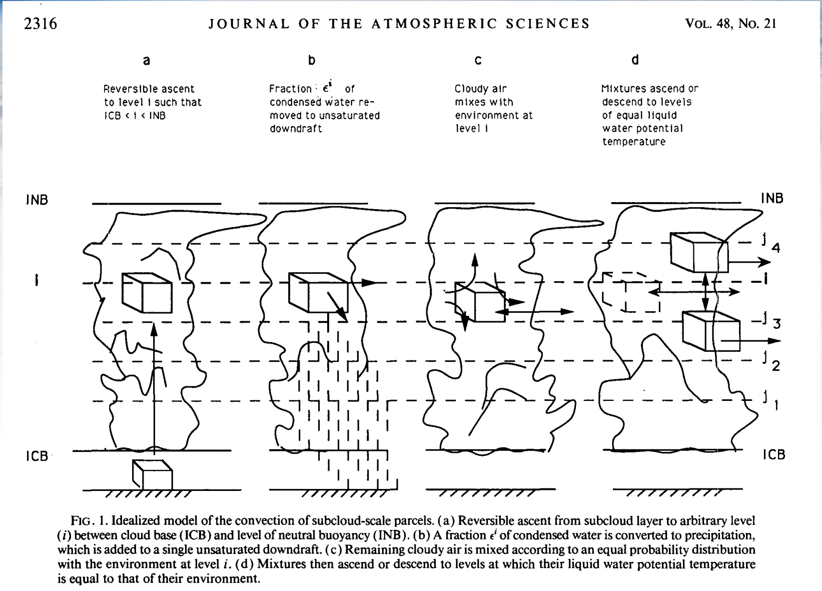
- Profile your code.
- Find the 10% of code that uses 90% of compute time.
- This will be a simple 10-line kernel!
- Put this on the FPGA using our magical tools;
- Success!

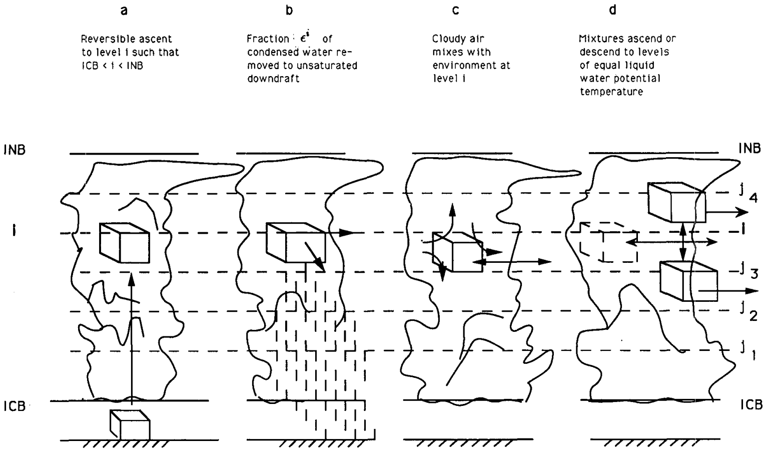
Real-world Algorithms ...

... are not like that

- Very complex (kernels are thousands of lines of code).
- Usually no quick wins.
- Parallelism is not obvious.

The Emanuel Convection Scheme

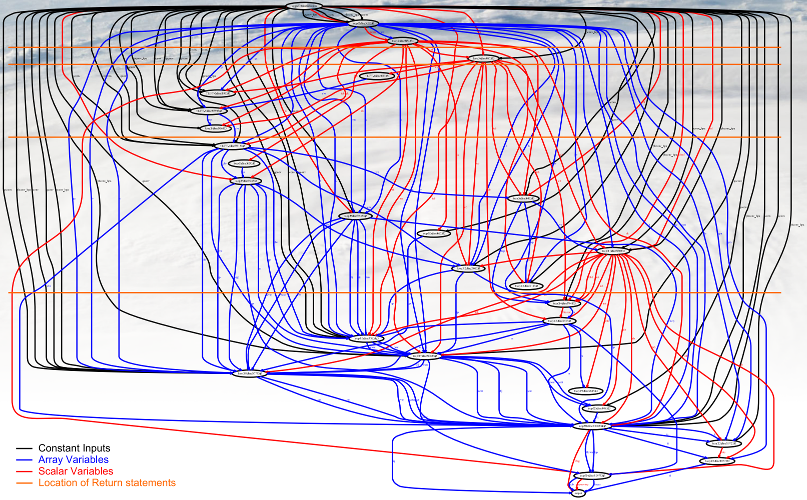




a
Reversible ascent to level i such that ICB < i < INB
b
Fraction ϵ^i of condensed water removed to unsaturated downdraft
c
Cloudy air mixes with environment at level i
d
Mixtures ascend or descend to levels of equal liquid water potential temperature
INB
i
ICB
j 4
j 3
j 2
j 1

FIG. 1. Idealized model of the convection of subcloud-scale parcels. (a) Reversible ascent from subcloud layer to arbitrary level (i) between cloud base (ICB) and level of neutral buoyancy (INB). (b) A fraction ϵ^i of condensed water is converted to precipitation, which is added to a single unsaturated downdraft. (c) Remaining cloudy air is mixed according to an equal probability distribution with the environment at level i. (d) Mixtures then ascend or descend to levels at which their liquid water potential temperature is equal to that of their environment.

Convection Kernel Structure



— Constant Inputs
— Array Variables
— Scalar Variables
— Location of Return statements

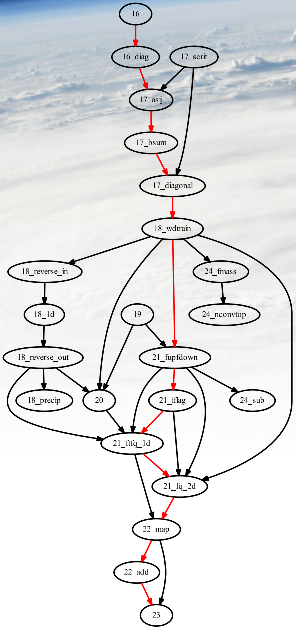
Convection Kernel

- Each node is a complex computation over a set of arrays.
- Many and complex dependencies between the nodes.

WARNING

THIS IS NOT A TOY.

Coarse Pipeline Design



- Loops are split into kernels according to parallelism, dimensions, and dependencies.
- Coarse pipeline: columns are streamed through stages as shown
- Fine pipeline: arithmetic inside each stage

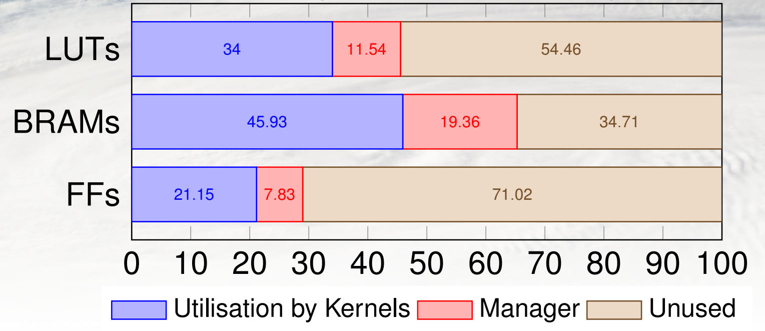
Parallelism Patterns

Loop	Kernels	Ticks	Pattern
16	16	n^2	Map
	16_diag	$2 \cdot n^2$	Diagonal Map
17	17_scrit	n	Map
	17_asij	$2 \cdot n^2$	Row Reduce/Map
	17_bsum	$2 \cdot n^2$	Row Reduce/Map
	17_diag	$3 \cdot n^2$	Diagonal Map
18	18_wdtrain	$15 \cdot n^2$	Fold*
	18_reverse_in	$2 \cdot n$	Reverse
	18	$440 \cdot n$	Fold*
	18_reverse_out	$2 \cdot n$	Reverse
	18_precip	1	Scalar
19	19	$12 \cdot n$	Reduce
20	20	$31 \cdot n$	Reduce
21	21_fupfdown	n^3	Fold*
	21_iflag	$2 \cdot n$	Reduce/Map
	21_ftfq_1D	n	Map
	21_fq_2D	$2 \cdot n^2$	Col Reduce/Map
22	22_map	n	Map
	22_add	$15 \cdot n$	Reduce
23	23	n	Map
24	24_fmass	n^2	Map
	24_nconvtop	$2 \cdot n^2$	Reduce
	24_sub	n	Map

- Map (also Diagonal Map)
- Reduce (also per-Row, per-Column)
- Fold (more general)
- Scalar
- Utilities (Reverse, Input, Output)

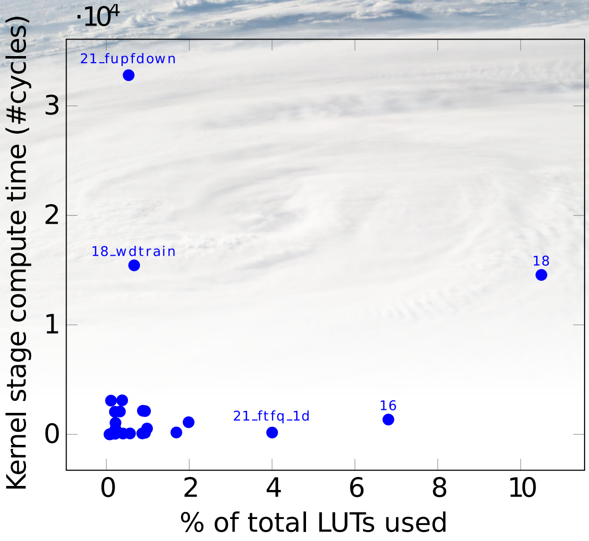
FPGA Resource Usage

For loops 16-24, accounting for about 40% of the algorithm.

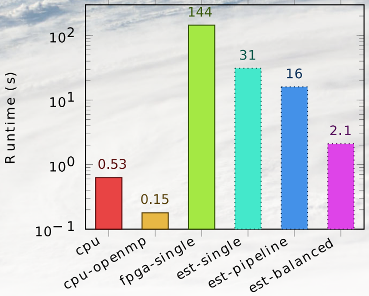


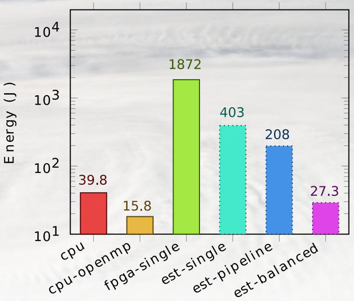
Synthesised at 200MHz, for layers $n = 32$

Kernel Compute Time vs Resource Usage



Performance Results





System power usage:
56W *idle*; 75W *single cpu*, 105W *all cpus (openmp)*, 69W *fpga running*.
⇒FPGA board power usage: 13W.

Discussion

- These results are quite preliminary.
- Current FPGA implementation energy efficiency is already of the same order as multicore CPU performance (16 J vs 27 J).
- Given the complexity of the algorithm, this is already a major achievement.
- There is a lot of room for improvement in our current implementation:
  - We did not analyse the code for loop fusion or fission.
  - We did not perform optimisations to simplify expressions and reduce the number of expensive operations (div, exp, log,...).
  - Buffer allocation is sub-optimal.
  - We did not attempt to optimize FP representation.

Discussion

- Given the current resource utilisation, implementing the above optimisations would likely allow to place at least two instances on the FPGA.
- This would double the energy efficiency.
- In practice, the design space to explore is very large, so we are working on an automated approach to program transformation.
- For more information, see `http://www.tytra.org.uk`

The Future is Bright!

- A very promising development is the next generation FPGA hardware with hardened floating-point units.
- Just appeared on the market!
  - Considerable savings in resources,
  - higher frequencies,
  - better energy efficiency.
- This is the closest an FPGA device has come to the needs of the HPC community.

Summary

- We ported a convection kernel from Fortran to FPGA, significantly larger than most commonly seen algorithms.
- FPGAs now have the resources to represent complete algorithms or applications.
- Even complicated loops can be decomposed into a small set of recurring patterns.
- Coarse pipeline design can be used for sequential type algorithms, but further optimisations need to be developed to achieve good performance.
- The high-level programming approach is a huge improvement over HDL, but as with any parallel platform, achieving good performance is still very hard for non-experts.
- The results demonstrate the potential of FPGAs for acceleration of scientific applications.