Using Kiwi for Big Genomic Data

Easy to Use Hardware Acceleration using FPGA

University of Cambridge
Computer Laboratory

Dr David Greaves
david.greaves@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

Computer Laboratory
Vet School Collaborations

1. with Dr Mark Holmes:

Obtaining the spa type of Staphylococcus aureus:

Do not assemble the whole gene,
It is critical to get the repeats correct,
Not interested in the other genes present.

2. with Dr Andrew Grant, Olu Oshota:

H/W Accelerated Seed+Search mapping

Working with FASTQ (Salmonella examples)
Replicate the results from the Novoalign package.
Burrows-Wheeler or Hash-based.

Big Data Orchestrators.

- Flume Java, MC Fast Flow, MillWheel
- Cloud Dataflow (Google's replacement for Map Reduce)
- Dryad/Linq or Hadoop
- CILK, MPI, Wool, ...
- Ciel (Skywriting)*
- Mirage - Uni-kernels directly on Zen*

*Originated at
Univ. Cambridge
Computer Laboratory

GPGPU is accepted as an accelerator – but hard to use?
Kiwi* aims to make FPGA or CGRA easy to use.

UNIVERSITY OF CAMBRIDGE

Computer Laboratory Answer? CIEL

CIEL: a universal execution engine for distributed data-flow computing

Derek G. Murray Malte Schwarzkopf Christopher Smowton
Steven Smith Anil Madhavapeddy Steven Hand
University of Cambridge Computer Laboratory

Abstract

This paper introduces CIEL, a universal execution engine for distributed data-flow programs. Like previous execution engines, CIEL masks the complexity of distributed programming. Unlike those systems, a CIEL job can make data-dependent control-flow decisions, which enables it to compute iterative and recursive algorithms.

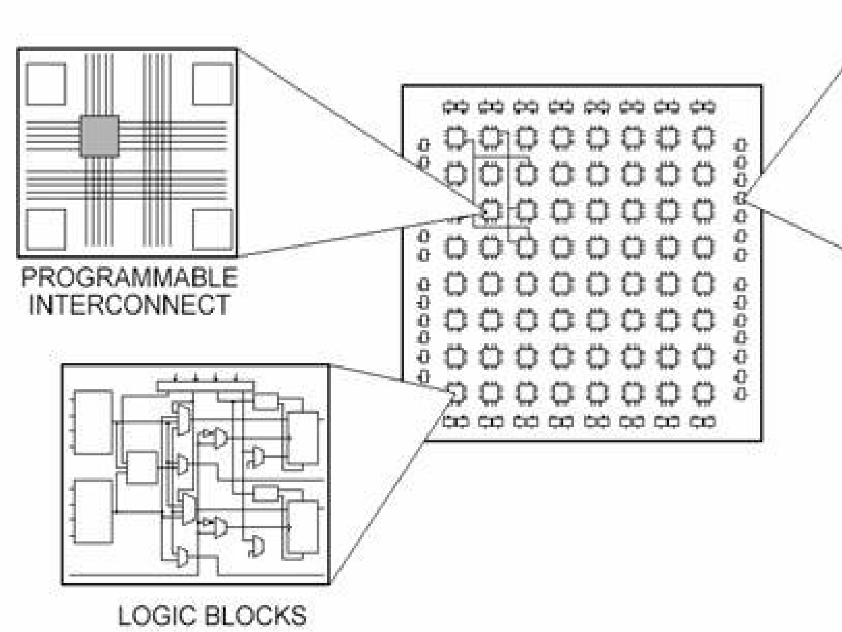
We have also developed Skywriting, a Turing-complete scripting language that runs directly on CIEL. The execution engine provides transparent fault tolerance and distribution to Skywriting scripts and high-

task-parallel algorithms using imperative and functional language syntax [31]. Skywriting scripts run on CIEL, an execution engine that provides a *universal* execution model for distributed data-flow. Like previous systems, CIEL coordinates the distributed execution of a set of data-parallel tasks arranged according to a data-flow DAG, and hence benefits from transparent scaling and fault tolerance. However CIEL extends previous models by *dynamically* building the DAG as tasks execute. As we will show, this conceptually simple extension—allowing tasks to create further tasks—enables CIEL to

[NSDI 2011]

UNIVERSITY OF CAMBRIDGE

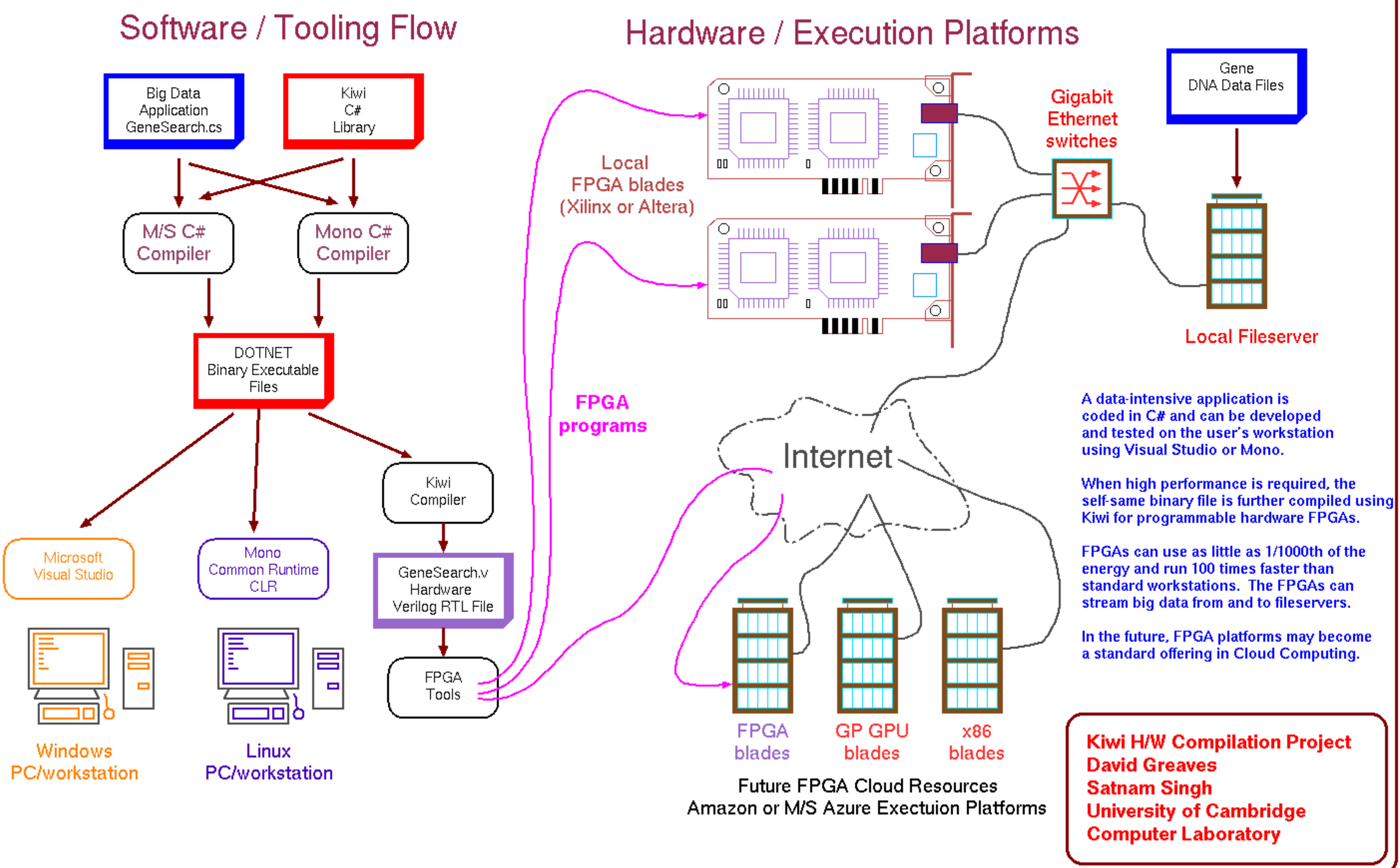
FPGA = Field Programmable Gate Array
CGRA = Coarse-grain Reconfigurable Array



Programmable hardware

- Dispenses with fetch/execute cycle
- Massively Parallel
- Down to 1/1000th the energy
- Up to 100x performance (depending on parallelism).

Kiwi - Accelerating Data-Intensive Applications using Networked FPGA (in the Cloud?)



First Result...

Design	RTL Length	State	CUPs/Clock
Hand	396 lines	59877 bits	8/19 = 0.42
Kiwi	27421 lines	68666 bits	8/20 = 0.40

Table 2. Comparison with hand-coded design.

Design	FPGA PART	Device	Utilization	Levels	Clock	CUP/s
Hand coded	Altera Stratix III	EP3SL340	5536 ALMs	28	138 MHz	58×10^6
Hand coded	Xilinx Virtex V	XC5VLX155T	5215 LUTs	25	101 MHz	42×10^6
Kiwi	Altera Stratix III	EP3SL340	20925 ALMs	37	83 MHz	33×10^6
Kiwi	Xilinx Virtex V	XC5VLX155T	55306 LUTs	86	46 MHz	18×10^6

Table 3. FPGA Performance Results (figures from Synplicity Premier).

`Synthesis of a Parallel Smith-Waterman Sequence Alignment Kernel into FPGA Hardware',
S Singh, DJ Greaves, and S Sanyal.
At Many-Core and Reconfigurable Supercomputing Conference 2009 (MRSC09), Berlin

Static Verus Dynamic Typed Languages for Hardware Acceleration.

Acceleration pitfalls for Dynamic Typed Languages

- *Runtime add or delete members in classes...*
- *Be aware which loops are to be unwound ...*
- *Using eval ...*
- *Changing vector lengths inside loops ...*

Are R and Python suitable for hardware acceleration ?

Or must we convert to strongly-typed 'clean' languages like C#, Java and Ocaml ?

END OF PRESENTATION

Smith-Waterman Genome Matcher coded in C# ...

```
public class SwElement
{ int width, unit;
  public int max;
  public int [] prev, here;
  public byte [,] slices;   // Local part of the PAM array
  public Kiwi.Channel < short > left_score, right_score;
  public Kiwi.Channel < byte > left_data, right_data;
  public Thread thread;
  short diag_left_left = 0;

  public SwElement(int u, int h)  // Constructor
   { width = h; unit = u;
     here = new int[width];
     prev = new int[width];
     slices = new byte[width, 20];
   }
```

```
public short run()
 {
   max = 0;
   byte dbval = left_data.Read();
   short topScore = left_score.Read();
   right_data.Write(dbval);


   for (int qpos = 0; qpos < width; qpos++) prev[qpos] = here[qpos];


   for (int qpos = 0; qpos < width; qpos++)
   {
     if ((qpos % unwind_factor)== 0) Kiwi.Pause();
     int above = prev[qpos];
     int left = qpos==0 ? topScore: here[qpos-1];
     int diag = (qpos == 0) ? diag_left_left:  prev[qpos - 1];
     int score = slices[qpos, dbval];
     int nv = Math.Max(0, Math.Max(left - 10, Math.Max(above - 10, diag + score)))
     if (nv > max) max = nv;
     here[qpos] = nv;
     if (qpos == width-1) right_score.Write((short)nv);
   }
   diag_left_left = topScore;
   return max;
 }
```

Dr. David Greaves. MIET.

- University Lecturer
- Chair of the CST Tripos
- Research Interests:
  - Hardware Compilers,
  - Simulation and Modelling,
  - Automated Reliable Component Composition.

UNIVERSITY OF CAMBRIDGE