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
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*

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

*Originated at Univ. Cambridge Computer Laboratory
Computer Laboratory
Answer? CIEL

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

Derek G. Murray Malte Schwarzkopf Christopher Snowton
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
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?)

Software / Tooling Flow

- Big Data Application: GeneSearch.cs
- Kiwi C# Library
- M/S C# Compiler
- Mono C# Compiler
- DOTNET Binary Executable Files
- Kiwi Compiler
- GeneSearch.v Hardware Verilog RTL File
- FPGA Tools

Hardware / Execution Platforms

- Gigabit Ethernet switches
- Local FPGA blades (Xilinx or Altera)
- Gene DNA Data Files
- Local Fileserver
- Internet
- Windows PC/workstation
- Linux PC/workstation
- Future FPGA Cloud Resources
- Amazon or M/S Azure Execution Platforms

A data intensive application is coded in C# and can be developed and tested on the user's workstation using Visual Studio or Mono.

When high performance is required, the self-same binary file is further compiled using Kiwi for programmable hardware FPGAs.

FPGAs can use as little as 1/1000th of the energy and run 100 times faster than standard workstations. The FPGAs can stream big data from and to file servers.

In the future, FPGA platforms may become a standard offering in Cloud Computing.

Kiwi H/W Compilation Project
David Greaves
Satnam Singh
University of Cambridge
Computer Laboratory
First Result...

<table>
<thead>
<tr>
<th>Design</th>
<th>RTL Length</th>
<th>State</th>
<th>CUPs/Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand</td>
<td>396 lines</td>
<td>59877 bits</td>
<td>8/19 = 0.42</td>
</tr>
<tr>
<td>Kiwi</td>
<td>27421 lines</td>
<td>68666 bits</td>
<td>8/20 = 0.40</td>
</tr>
</tbody>
</table>

Table 2. Comparison with hand-coded design.

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

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# ...

```csharp
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++)
    {
        int prev = 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);
    }

    return max;
}

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];
    }
}
```
Dr. David Greaves. MIET.

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