Restoration of Star-Field Images Using High-Level Languages and Core Libraries

Robin Bruce, Caroline Ruet, Dr Malachy Devlin, Prof Stephen Marshall
Overview

• Introduction
• MED – SA algorithm
• ANSI C implementation
• DIME-C implementation
• Comparison software/hardware
• Application
• Conclusion & further work
Introduction

• There is a need for objective research in reconfigurable computing (RC)
  • Don’t just pick battles you know you’ll win
• Need to evaluate effectiveness of RC as a general purpose solution
  • How does it work on arbitrarily-selected problems?
• There is a range of measures that we can apply to determine the performance improvement
Hardware Comparisons

- Can Compare FPGAs to:

- Digital Signal Processors (DSPs)
- Application-Specific Integrated Circuits (ASICs)
- Microprocessors
- Other, could include
  - Graphics Processing Units (GPUs)
  - Cell BE Processor
  - Clearspeed CX600
Hardware Comparisons II

• Can compare with respect to:
  • Raw performance
  • Power consumption
  • Unit cost
  • Board footprint
  • Non-Recurring Engineering Cost (NRE)
  • Design time and Design cost

• The key metrics for a particular application may also include ratios of these metrics, e.g. performance/power, or performance/unit cost.
Application Choice

• Implementation of the Minimum Entropy Deconvolution algorithm using Simulated Annealing method: representative of a computationally intense image-processing application

• Chosen Fairly Arbitrarily
  • Only knew that it was a compute-intensive algorithm
  • Did not know how suitable the algorithm was for implementation on RCs before committing to it
Chosen Comparison

• Comparing a 90nm-process commodity microprocessor with a platform based around a 90nm-process FPGA
  • 3.2 GHz Intel Pentium D processor with 2 GB of DRAM, with the gcc compiler
  • Nallatech H101-PCIXM card, with the DIME-C compiler
    • Xilinx Virtex-4 LX160 FPGA, 512 MB of DRAM and 4 banks of 200MHz, 4 MB SRAM.
• Focussing on design time and raw performance improvement.
MED – SA algorithm

• Restore blurred images
• MED algorithm with SA used to converge towards the globally-optimum solution
• $y = x \ast h + n$
  • $y$: observed image, $x$: original image, $h$: Gaussian filter, $n$: white Gaussian noise

• Estimate $x$ from $y$

• Computation of 2 gradients: $\Delta x = \partial E / \partial x$, $\Delta d = \partial E / \partial d$
MED – SA algorithm

• Assumptions

  • *PSF is a Gaussian function*

  \[
  h(d) = \begin{cases} 
  \gamma \exp \left( - \frac{m_1^2 + m_2^2}{d} \right), & \text{for } m_1, m_2 = -2, -1, 0, 1, 2 \\
  0, & \text{otherwise}
  \end{cases}
  \]

  \[d \in [0, \infty)\]

  • *m1, m2 designates the size of the PSF*
  
  • *d corresponds to the width of the PSF (blurring level)*
  
  • *γ is a constant to normalise the Gaussian function:*

  \[
  \sum_{m_1=-\infty}^{+\infty} \sum_{m_2=-\infty}^{+\infty} h(d) = 1
  \]
MED – SA algorithm

• Algorithm – minimising the Energy $E$

• **Step 0:** Set $p=0$ and initialise $x_p$, $T_p$, $d_p$ and $\alpha$, $\beta$, $\lambda$

• **Step 1:** Compute the energy $E_p(x_p, h(d_p))$

$$\begin{align*}
E(x, h(d)) &= \frac{(1 - \lambda) \left\{ \sum_{k_1} \sum_{k_2} x^2(k_1, k_2) \right\}^2}{\sum_{k_1} \sum_{k_2} x^4(k_1, k_2)} + \lambda \sum_{k_1} \sum_{k_2} [x(k_1, k_2) * h(k_1, k_2) - y(k_1, k_2)]^2 \\
&= \frac{\text{sizeof}(x)}{\text{sizeof}(x)}
\end{align*}$$
MED – SA algorithm

- **Step 2:** Select a candidate solution

\[ x'_{p+1} = x_p - a \Delta x_p \]

\[
\frac{\partial E}{\partial x_p(n_1, n_2)} = 4 x_p(n_1, n_2) \left( \sum_{k_1} \sum_{k_2} x_p^2(k_1, k_2) \right) \cdot \left( 1 - x_p^2(n_1, n_2) \right) \left( \sum_{k_1} \sum_{k_2} x_p^4(k_1, k_2) \right) \\
+ 2 \lambda \sum_{k_1} \sum_{k_2} \sum_{m_1} \sum_{m_2} \left[ x_p(k_1 - m_1, k_2 - m_2) \cdot h_p(m_1, m_2) - y(k_1, k_2) \right] \cdot h_p(k_1 - n_1, k_2 - n_2)
\]

\[
d'_{p+1} = d_p - \beta \Delta d_p
\]
MED – SA algorithm

- **Step 3:** Compute the energy $E'_{p+1}(x'_{p+1}, h(d'_{p+1}))$
  $$\Delta E = E'_{p+1} - E_p$$

- **Step 4:**
  If: $\exp\left(-\frac{\Delta E}{T_p}\right) > r$ where $r$ is a random number $\in [0,1]$  
  Then: $x_{p+1} = x'_{p+1}$, $d_{p+1} = d'_{p+1}$ and $T_{p+1} = T_p$

  Else: $x_{p+1} = x_p$, $d_{p+1} = d_p$ and $T_{p+1} = f(T_p)$ where $f(.)$ is a decreasing function

- **Step 5:** $p = p + 1$, 
  $\#_\text{iterations} = \#_\text{iterations} - 1$

- **Step 6:** Output $x_{p+1}$ is the estimation image
ANSI C Implementation

- Algorithm organisation

- The C program is not initially optimised
DIME-C Implementation

- Code modification
- Loop Fusion
- Pipelining
- Spatial parallelism
- Resource optimisation
Example Optimisation

- **Filter Application**
  - number of cycles: \( \approx 480,111 \)  
    (before optimisation: \( 12,000,133 \))
  - number of slices: \( = 27733 \)  
    (before optimisation: \( 3184 \))

*Graphical representations of the filter implementation*
Core Libraries

- Made use of single-precision mathematical operators that are integrated into DIME-C
- Project depended on random number generator and exponential function
  - Not in compute intensive region of algorithm
  - Functions acted as an enabler to full algorithm implementation
- Hear more about Core Libraries from me later today
Implementation Procedure

1. Created a DIME-C project using the original source from the ANSI C project
2. Adapted source to allow compilation in both DIME-C and ANSI C environments
3. Took advantage of the most obvious pipelining opportunities to create 1st FPGA implementation
4. Examining the source code and the output of DIME-C, created an equation that expressed the runtime of the algorithm in cycles, as a function of the parameters of the algorithm, divided into key sections.
5. Determined for a typical set of algorithm parameters the section that took up the majority of the runtime, and optimised the DIME-C for this section to create the 2nd FPGA implementation
6. Repeated sections 4&5 to produce the 3rd FPGA implementation,
Time to Solution

• Developing the initial ANSI C Implementation
  • 125 Person Hours

• Developing the DIME-C Implementation
  • 35 Person Hours

• Most time spent developing the software
Software versus Hardware

• Several generations of the FPGA implementation compared to software

<table>
<thead>
<tr>
<th></th>
<th>Software</th>
<th>1st FPGA</th>
<th>2nd FPGA</th>
<th>3rd FPGA</th>
<th>4th FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td></td>
<td>7.98×10¹⁰</td>
<td>8.72×10¹⁰</td>
<td>4.30×10¹⁰</td>
<td>2.59×10¹⁰</td>
</tr>
<tr>
<td>Time in Seconds</td>
<td></td>
<td>216</td>
<td>798.00</td>
<td>87.24</td>
<td>42.96</td>
</tr>
<tr>
<td>Speedup vs. Software</td>
<td></td>
<td>1</td>
<td>0.27</td>
<td>2.48</td>
<td>5.03</td>
</tr>
<tr>
<td>% Contribution of:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>De_Dx</td>
<td></td>
<td>5.02</td>
<td>45.94</td>
<td>93.29</td>
<td>88.91</td>
</tr>
<tr>
<td>Filter</td>
<td></td>
<td>94.74</td>
<td>51.86</td>
<td>2.24</td>
<td>3.71</td>
</tr>
<tr>
<td>Rest of Algorithm</td>
<td></td>
<td>0.24</td>
<td>2.20</td>
<td>4.47</td>
<td>7.38</td>
</tr>
</tbody>
</table>

• DeDx remains the focus of a 5th version
Example Results

- Simulation of real Black and White pictures
  - 200 x 200 image
  - 7 x 7 filter
Conclusion

- Good performance: speedup $\approx 8$
- Design productivity was high using DIME-C
- Increased performance and productivity possible using libraries of low-level IP cores
- 100-Page report available for those who want to know more
Microprocessor Speedups

Performance (vs. VAX-1/1780)

- 1978
- 1980
- 1982
- 1984
- 1986
- 1988
- 1990
- 1992
- 1994
- 1996
- 1998
- 2000
- 2002
- 2004
- 2006

- 1
- 10
- 100
- 1000
- 10000

- 25%/year
- 52%/year
- ??%/year
Speedup in Context

- Moore’s Law Tells us Performance Doubles Every 18 months
  - Does it really?
- Hennessey & Patterson (2007) tell us that processor performance improved by 52% a year until 2002.
- Since 2002 it’s been running at around 20% a year
Speedup in Context II

• If an FPGA gives you an 8x speedup now, how many years would it take for the microprocessor to catch up?

• Assumption Alert!
  • Benchmark used to evaluate processors gives a good idea of how our application would perform
  • Comparing two best-effort implementations on the same process node, FPGA and uP

• 8x Speedup would take 11-12 years to attain at 20% per annum improvements
The Magic Numbers

• How much is being 12 years ahead of the competition worth?

• *Reconfigurable Computing must offer* (insert *magic number*) \( X \) improvement over *conventional computing* to see widespread adoption
  
  • Such a blanket statement is meaningless

• Depends on the economics of the application
Acknowledgements

- Research is sponsored by Nallatech
- Institute for System Level Integration made everything possible
- Caroline did all the hard work
Quote

• Alan Perlis - When someone says "I want a programming language in which I need only say what I wish done," give him a lollipop.