Institute for System Level Integration
THE ALBA CAMPUS

NALLATECH

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

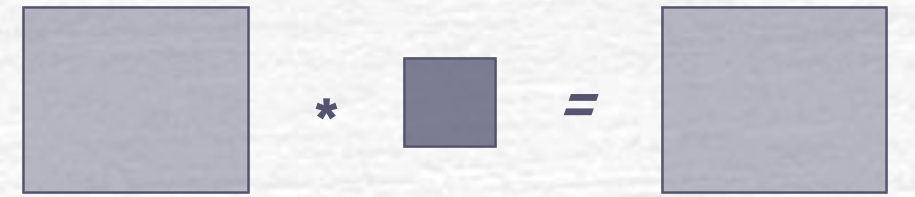
- Chosen Fairly Arbitrarily
  - Only knew that it was a compute-intensive algorithm
  - Did not know how suitable the algorithm was for implementation on RCs before committing to it

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
- $y = x * h + n$
  - y: observed image, x: original image, h: Gaussian filter, n: white Gaussian noise
- Estimate x from y



Estimated image *Estimated filter* *Observed image*

- 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*

$$h(d) = \begin{cases} \gamma \exp\left(-\dfrac{m_1^2 + m_2^2}{d} \right), & for \quad m1, m2 = -2, -1, 0, 1, 2 \\ 0, & otherwise \end{cases}$$

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

  - *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 a, β, λ
  - *Step 1*: Compute the energy $E_p(x_p, h(d_p))$

$$E(x,h(d)) = \frac{(1-\lambda)\dfrac{\left[\sum_{k_1}\sum_{k_2} x^2(k_1,k_2)\right]^2}{\sum_{k_1}\sum_{k_2} x^4(k_1,k_2)} + \lambda \sum_{k_1}\sum_{k_2}\left[x(k_1,k_2) * h(k_1,k_2) - y(k_1,k_2)\right]^2}{sizeof(x)}$$

MED – SA algorithm

- *Step 2*: Select a candidate solution

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

$$\frac{\partial E}{\partial x_p(n_1,n_2)} = 4x_p(n_1,n_2)\frac{\sum_{k_1}\sum_{k_2}x_p^2(k_1,k_2)}{\sum_{k_1}\sum_{k_2}x_p^4(k_1,k_2)}\cdot\left[1-x_p^2(n_1,n_2)\frac{\sum_{k_1}\sum_{k_2}x_p^2(k_1,k_2)}{\sum_{k_1}\sum_{k_2}x_p^4(k_1,k_2)}\right]$$

$$+2\lambda\sum_{k_1}\sum_{k_2}\sum_{m_1}\sum_{m_2}\left[\begin{matrix}x_p(k_1-m_1,k_2-m_2)\\ \cdot h_p(m_1,m_2)-y(k_1,k_2)\end{matrix}\right]\cdot h_p(k_1-n_1,k_2-n_2)$$

$$d'_{p+1} = d_p - \beta\Delta d_p$$

$$\frac{\partial E}{\partial d_p} = 2\lambda\sum_{k_1}\sum_{k_2}\sum_{m_1}\sum_{m_2}\left[\begin{matrix}x_p(k_1-m_1,k_2-m_2)\\ \cdot h_p(m_1,m_2)-y(k_1,k_2)\end{matrix}\right]\cdot\sum_{m_1}\sum_{m_2}\frac{(m_1^2+m_2^2)}{d_p^2}x_p(k_1-m_1,k_2-m_2)\cdot h_p(m_1,m_2)$$

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: $x_{p+1} = x'_{p+1}, \quad d_{p+1} = d'_{p+1} \quad and \quad T_{p+1} = T_p$

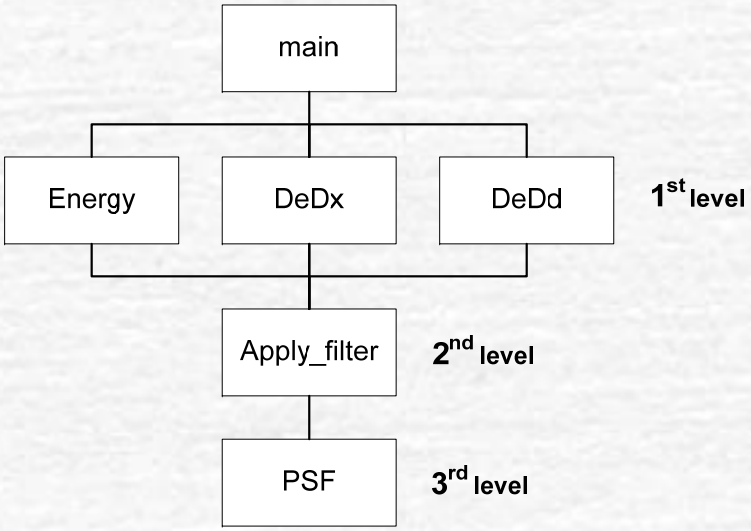
Else: $x_{p+1} = x_p, \quad d_{p+1} = d_p \quad and \quad T_{p+1} = f(T_p)$

where $f(.)$ is a decreasing function

- *Step 5*: p = p + 1, *#_iterations = #_iterations – 1*
- *Step 6*: Output *xp+1* is the estimation image

ANSI C Implementation

- Algorithm organisation

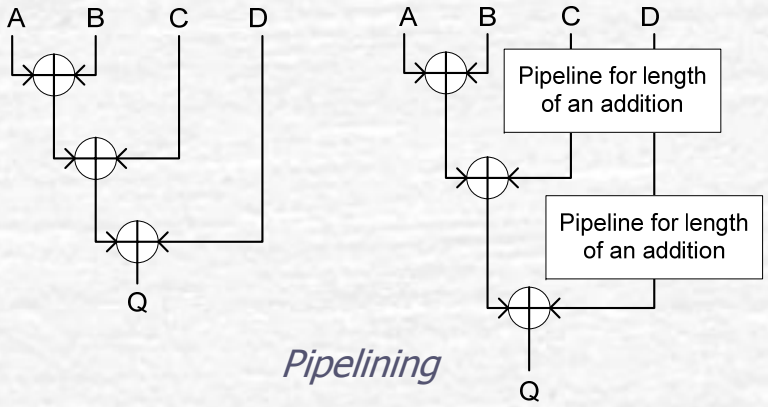


Functional Hierarchy

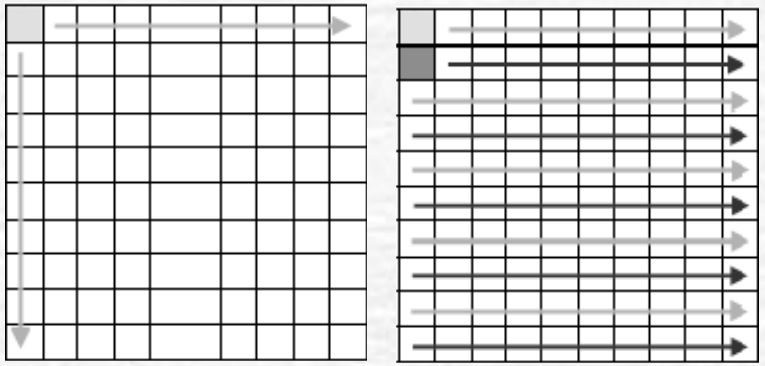
- The C program is not initially optimised

DIME-C Implementation

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



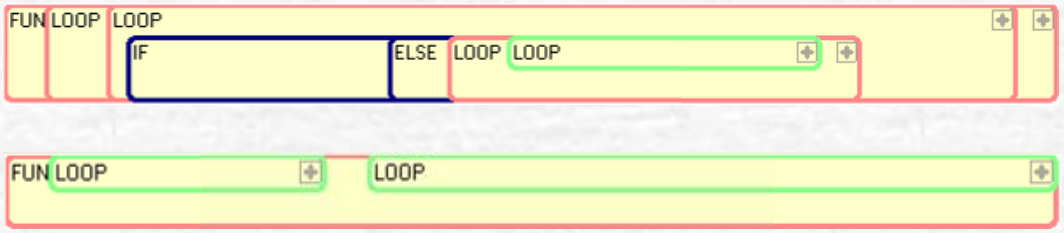
Pipelining



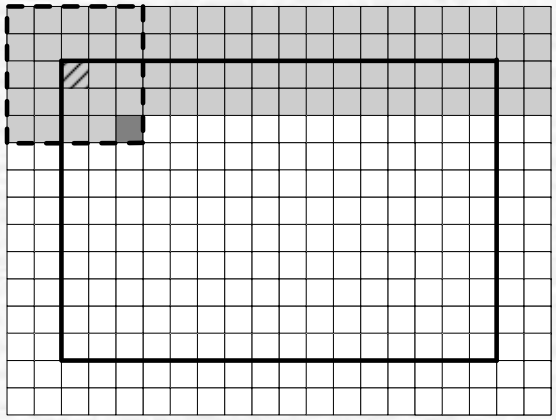
Spatial parallelism

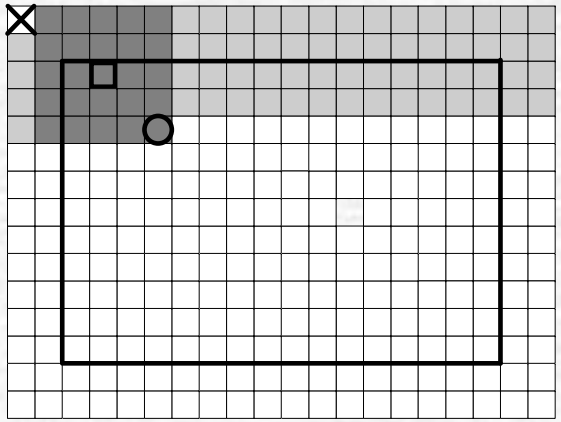
Example Optimisation

- Filter Application
  - number of cycles: ≈ 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

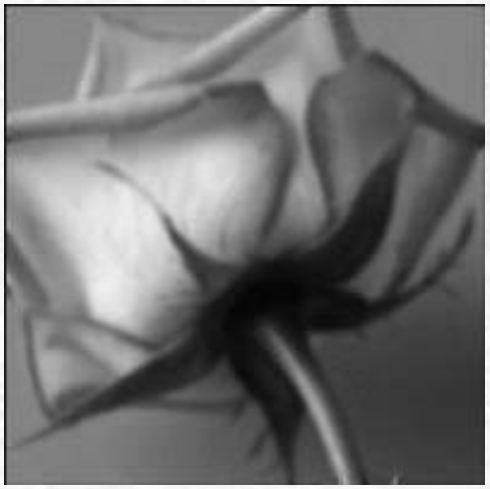
	Software	1st FPGA	2nd FPGA	3rd FPGA	4th FPGA
Cycles		7.98×10^{10}	8.72×10^{10}	4.30×10^{10}	2.59×10^{10}
Time in Seconds	216	798.00	87.24	42.96	25.92
Speedup vs. Software	1	0.27	2.48	5.03	**8.33**
% Contribution of:					
De_Dx		5.02	45.94	93.29	88.91
Filter		94.74	51.86	2.24	3.71
Rest of Algorithm		0.24	2.20	4.47	7.38

- DeDx remains the focus of a 5th version

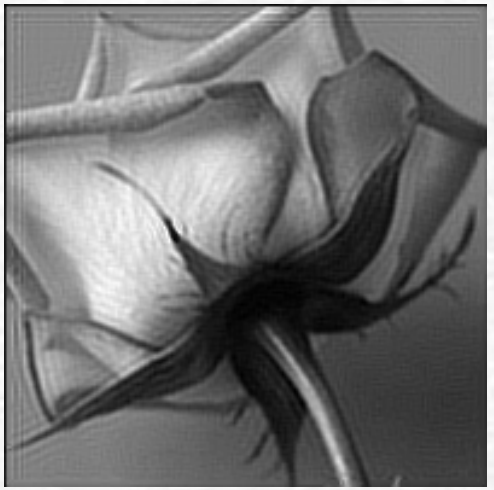
Example Results

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

Original image



Observed image: blurred and noisy



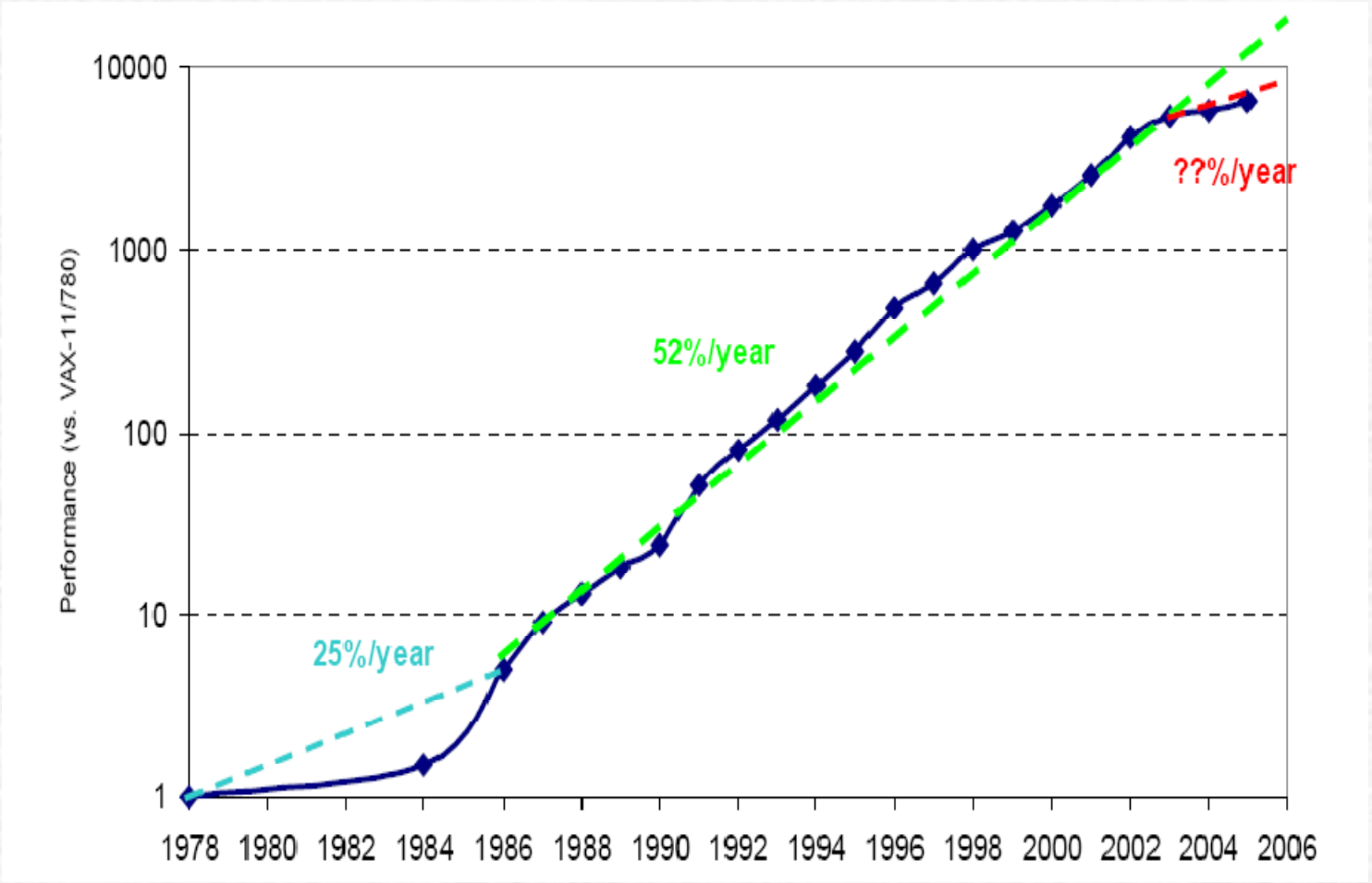
Restored image 300 iterations

Conclusion

- Good performance: speedup ≈ 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



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.