RESTORATION OF STAR-FIELD IMAGES USING HIGH-LEVEL LANGUAGES AND CORE LIBRARIES

Robin Bruce – Institute for System Level Integration, Alba Campus, Livingston, Scotland. robin.bruce@sli-institute.ac.uk
Tel: +44 (0) 117 377 7144
Caroline Ruet – L'Institut National des Sciences Appliquées, Rennes, France. caroline.ruet@ens.insa-rennes.fr
Dr Malachy Devlin – Chief Technology Officer, Nallatech, Glasgow, Scotland. m.devlin@nallatech.com
Prof. Stephen Marshall – University of Strathclyde, Glasgow, Scotland. s.marshall@eee.strath.ac.uk

1. INTRODUCTION

Research into the use of FPGAs in Image Processing began in earnest at the beginning of the 1990s. Since then, many thousands of publications have pointed to the computational capabilities of FPGAs. During this time, FPGAs have seen the application space to which they are applicable grow in tandem with their logic densities. When investigating a particular application, researchers compare FPGAs with alternative technologies such as Digital Signal Processors (DSPs), Application-Specific Integrated Circuits (ASICs), microprocessors and vector processors. The metrics for comparison depend on the needs of the application, and include such measurements as: raw performance, power consumption, unit cost, board footprint, non-recurring engineering cost, design time and design cost. The key metrics for a particular application may also include ratios of these metrics, e.g. power/performance, or performance/unit cost. The work detailed in this paper compares a 90nm-process commodity microprocessor with a platform based around a 90nm-process FPGA, focussing on design time and raw performance.

The application chosen for implementation was a minimum entropy restoration of star-field images (see [1] for an introduction), with simulated annealing used to converge towards the globally-optimum solution. This application was not chosen in the belief that it would particularly suit one technology over another, but was instead selected as being representative of a computationally intense image-processing application.

2. IMPLEMENTATION

The implementation of the algorithm took place in two stages, and involved two of the authors. One author took the role of *application specialist*, while the other took the role of *reconfigurable-computing specialist*.

The software platform was a 3.2 GHz Intel Pentium D processor with 2 GB of DRAM, with the gcc compiler. The targeted FPGA platform was a Nallatech H101-PCIXM card, with the DIME-C compiler. The H101-PCIXM card has a Xilinx Virtex-5 LX100 FPGA, 512 MB of DRAM and 4 banks of 200MHz, 4 MB SRAM.

In the first stage of implementation, the application specialist developed the algorithm from theory to implementation in ANSI C on a commodity microprocessor. This stage represented the majority of the time spent in the project, around 100 person-hours. The application specialist carried out this work, without any significant input from the reconfigurable-computing specialist. Once the algorithm was functioning satisfactorily in software, the second stage began.

In the second stage of the algorithm, the application specialist and the RC specialist worked together to optimise the algorithm, to migrate it to the DIME-C environment, and to characterise its performance. This process took approximately 25 person-hours. The ANSI C implementation was optimised for performance in order to make for the fairest comparison. The software performance was improved by several orders of magnitude during this time; otherwise, the software-hardware comparison would have been more weighted in favour of the FPGA.

A typical procedure for the porting of algorithms to software is to first profile the software-implemented algorithm, then implement on the FPGA the functions that represent the majority of the calculation time. There are disadvantages to this approach. It neglects to take into account the data transfers that are necessary between the RC platform and the microprocessor-based system before and after each function call. When factored in, these data transfers can negate any performance improvement in the hardware-implemented function. The approach taken here was to implement the entire algorithm on the FPGA, so that the data transfer time is negligible in comparison to the compute time. This means that all improvements to the computation time of a section of the algorithm translate into a measured performance improvement. The full implementation of the algorithm would have been impossible without access to a core library of mathematical functions, to provide the *exponential* function and *pseudo-random number generator* necessary for simulated annealing. The implementation process consisted of the following steps:

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,

The performance improvements with each revision of the FPGA-implemented algorithm are shown below. Data transfer times were negligible and did not contribute to the final result:

	1st FPGA	**2nd FPGA**	**3rd FPGA**	**Software**
Cycles	81,180,026,000	10,104,026,000	5,725,226,000	
Time (seconds)	811.80026	101.04026	57.25	216
Speedup vs. Software	0.266075303	2.13776172	3.77	1
% Contribution of:				
De_Dx	6.65	53.44	94.32	
Filter	93.13	44.77	2.54	
Rest of Algorithm	0.22	1.78	3.14	

As the *filter* section was improved, the performance of the algorithm improved. The evolution of the characteristic equation can be seen below. It should be obvious from the table that the algorithm section *De_Dx* would be the focus of a 4th revision.

$$Filter_{FPGA1} = 3 \cdot (2n_2 + 101) \cdot c \cdot l \cdot (2n_1 + 1) \cdot n\_iter$$

$$Filter_{FPGA2} = (\frac{c \cdot l}{2}) \cdot (\frac{3n_1 n_2}{2} + 151) \cdot n\_iter$$

$$Filter_{FPGA3} = 3 \cdot (c \cdot l + (2n_2 + 1) \cdot c \cdot n\_iter$$

REFERENCES

Wu H, Barba J, *Minimum entropy restoration of star field images*. Address: *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, Vol. 28, No. 2. (1998), pp. 227-231.