Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study

Ben Cope, *Member*, *IEEE*, Peter Y.K. Cheung, *Senior Member*, *IEEE*, Wayne Luk, *Fellow*, *IEEE*, and Lee Howes, *Member*, *IEEE*

Abstract—A systematic approach to the comparison of the graphics processor (GPU) and reconfigurable logic is defined in terms of three throughput drivers. The approach is applied to five case study algorithms, characterized by their arithmetic complexity, memory access requirements, and data dependence, and two target devices: the nVidia GeForce 7900 GTX GPU and a Xilinx Virtex-4 field programmable gate array (FPGA). Two orders of magnitude speedup, over a general-purpose processor, is observed for each device for arithmetic intensive algorithms. An FPGA is superior, over a GPU, for algorithms requiring large numbers of regular memory accesses, while the GPU is superior for algorithms with variable data reuse. In the presence of data dependence, the implementation of a customized data path in an FPGA exceeds GPU performance by up to eight times. The trends of the analysis to newer and future technologies are analyzed.

Index Terms—Graphics processors, reconfigurable hardware, real-time and embedded systems, signal processing systems, performance measures, video.

1 Introduction

An exciting emerging trend is the use of homogeneous multiprocessor architectures to accelerate video processing applications [1], [2], [3]. Examples include graphics processors (GPUs) and the Cell Broadband Engine. These devices compete with digital signal processors and SRAM-based field programmable gate arrays (FPGAs) as accelerators to a general-purpose processor (GPP). An FPGA is a device based on reconfigurable logic fabric. This work quantifies the comparison of the GPU and FPGA to explore which is the superior video processing accelerator. The approach is demonstrated using an nVidia GeForce 7900 GTX and the Xilinx Virtex-4 FPGA.

A practical application of the comparison results is to provide a heuristic by which one may choose between alternate accelerators for a target application domain. Alternatively, a theoretical basis for hardware-software codesign between FPGA and GPU cores in a multicore system-on-chip. To exemplify, the case studies presented in Table 4 may each form a block in a video acquisition, enhancement, and encoding system. From the analysis in this work, it is observed that it is desirable to implement primary color correction (PCCR) and resizing on the GPU. The FPGA is most suited to histogram generation and motion vector estimation. For small 2D convolution kernels the GPU is favorable, however, for larger kernels an FPGA is most suited. A novel and exciting finding in this work is a structured comparison approach which can be used to qualitatively reason the preferences stated here. The transfer of the approach and results to updated GPU architectures and other application domains is presented throughout the paper.

The case study implementations originate from a comparison study of the graphics processor and reconfigurable logic devices for the implementation of video processing algorithms [1] and further novel implementations in [2]. The FPGA implementation of motion vector estimation is taken from work by Sedcole [4].

The following original contributions are made:

1. Formulation of the performance comparison of a GPU to an FPGA. The motivation is to adapt work by Guo et al. [5] on GPP and FPGA comparison to address the challenges and interpretations of comparing GPUs and FPGAs *(Section 3)*.
2. A case study of the performance comparison using the nVidia GeForce 7900 GTX GPU and the Xilinx Virtex-4 FPGA. The following contributory points are made:
   a. Analysis of five case study algorithms which populate the design space of arithmetic complexity, memory access requirements, and data dependencies *(Section 4)*.
   b. Quantitative comparison of the three throughput rate drivers of iteration level parallelism (ItLP), clock speed, and clock cycles per output (CPO) *(as summary in Section 6)*.
   c. A study of the speedup of each device, over a GPP, in regard to arithmetic complexity *(Section 5.1)*.
   d. Comparison of on-chip and off-chip memory access requirements *(Section 5.2)*.

- *B. Cope and P.Y.K. Cheung are with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2BT, UK.*
- *W. Luk and L. Howes are with the Department of Computing, Imperial College London, London SW7 2BZ, UK.*

Manuscript received 5 May 2008; revised 12 Mar. 2009; accepted 9 Sept. 2009; published online 11 Dec. 2009.
Recommended for acceptance by W. Najjar.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-05-0196.
Digital Object Identifier no. 10.1109/TC.2009.179.

e. The methods used to map algorithms with data dependencies onto a GPU and FPGA are compared in *Section 5.3*.
f. Comparison limitations are discussed in *Section 5.4*.

3. The work is applied to the GeForce 8 and 9 generations of GPUs in *Sections 3.3, 5.1.3, 5.2.5, 5.3.3*, and *6.2*.
4. A projection of the scalability of device architectures with respect to Ahmdahl's Law *(Section 7.1)*.

The paper is arranged as follows: Section 2 discusses related work. Section 3 formulates the performance comparison. Section 4 presents five case study algorithms. Section 5 includes the case study performance comparison. The results from Section 5 are combined in summarizing the throughput rate drivers in Section 6. Section 7 summarizes the key findings.

2 Related Work

A number of prior works present a quantitative comparison of FPGA and GPU [1], [2], [6], [7], [8], [9], [10].

Xue et al. [6] compare the implementation of a Fluoro-CT Reconstruction algorithm on an FPGA and a GPU. Performance is shown to be approximately equal. Updated work by Mueller [7] on Computed Tomography shows a four to five times speedup from a GeForce 8800 GTX device over older generations of FPGAs. A like-for-like hardware comparison is absent from the results.

Howes et al. [8] present the automated compilation of source code for GPU, FPGA, and Playstation 2 using the ASC compiler. The GPU is shown to be superior for Monte Carlo computation. Performance figures are highly sensitive to the ASC compiler.

Baker et al. [9] compare device cost, power, and performance. It is shown that in speedup over a GPP, the GPU is superior in terms of price per unit speedup and an FPGA in speedup per unit of energy. These results suggest that a GPU is beneficial for super computing systems and the FPGA is beneficial for embedded systems.

Morris and Aubury [10] compare GPU, Cell, and FPGA performance for financial applications. For a moderate FPGA clock speed of up to one fifth the speed of a GPU, the FPGA solution provides a superior trade-off of performance and accuracy over similar GPU generations (to the FPGA) and a Cell; as an example up to 4.5 times higher than a GPU for a 50 percent drop in precision. For equal precision, the FPGA has marginally higher performance.

Kelmelis et al. [11] discuss the combination of a Cell B.E., GPU, and FPGA together to form a high-performance computing cluster. No performance figures or benchmarks are considered to justify comment. Mueller et al. [7] also present a discussion of all three architectures from work on Computed Tomography. In giga-flop performance, the GPU is superior [7], [11]. A Cell B.E. has a peak 256 GFLOPS performance whereas a GeForce 8 series GPU has a peak of approximately double this value. A Cell B.E. provides increased flexibility over the GPU because parallel multiprocessors (SPEs) can be individually controlled. The benefit is that an application can be pipelined across SPEs. An FPGA provides further flexibility over the Cell B.E.

Owens et al. [12] present a survey of the use of GPUs for general-purpose applications. Memory access reduction operations, as analyzed in Section 5.3, and the lack of memory scatter capability are cited as weaknesses of GPUs. Owens' results concur with observations made by the authors in [1], [2]. (In a later publication, Owen et al. observe that GeForce 8 series GPUs provide limited support for implementing scatter and reduction operations [13].)

Govindaraju et al. [14] analyze GPU memory system behavior. Using the 3C's cache model FFT (GPU-FFT) and sorting implementations are optimized. For FFT, a GeForce 7900 GTX provides a 1.5-2 times improvement over an Intel processor. An equivalent Virtex-4 FPGA implementation with the Sundance floating point FFT core [15], operating at 200 MHz, performs a 1 M sample FFT in 21 ms. The GPU-FFT algorithm run by the author on a GeForce 7900 GTX card performs a 1 M sample FFT in 19 ms. The GPU implementation is fractionally superior.

Angelopoulou et al. [16] present a discrete wavelet transform (DWT) implementation on a Virtex-4 FPGA, and Wong et al. [17] an equivalent DWT on a GeForce 7800 GT GPU. An 89 times performance benefit is observed for the FPGA over a GPP [16], and a 3 times benefit for the GPU over a GPP [17]. The FPGA has a superior performance due to a specialized data path and optimized memory reuse strategy.

In [18], a binomial American pricing option implementation is analyzed. A GeForce 7900 GT GPU is observed in this setup to provide equivalent performance to a Virtex-4 FPGA for a "like-for-like" numerical precision. For a GeForce 8600, the GPU provides three times performance improvement over the Virtex-4. If fixed-point representation is used for the FPGA, the performance is comparable to the GeForce 8.

To the authors' knowledge, this paper presents the first detailed analysis of the performance drivers for the GPU and FPGA. Related work above offers motivations and analysis but provides no systematic approach to performance comparison.

Section 3.2 is an adaptation of Guo et al.'s [5] analysis of the speedup factors of an FPGA over GPPs, as summarized below.

In their speedup analysis, Guo et al. consider clock cycle efficiency in isolation. The number of GPP and FPGA clock cycles required for a given application are shown in (1) and (2), respectively. In Table 1, the terminology is summarized.

$$Cycle_{gpp} = Instr_{iter:gpp} \times N_{iter} \times CPI_{gpp}, \tag{1}$$

$$Cycle_{fpga} = \frac{N_{iter}}{ItLP_{fpga}}. \tag{2}$$

The relative speedup of FPGA devices over a GPP is shown in (3) as the ratio of the clock cycle requirements.

$$Speedup_{fpga \rightarrow gpp} = \frac{Cycle_{gpp}}{Cycle_{fpga}}. \tag{3}$$

Equation (4) expresses instruction inefficiency ($Ineff_{gpp}$) which is the fraction of instructions not related directly to computation. The instruction count per operation ($Instr_{iter:gpp}$) is separated into support instructions ($Instr_{sppt:gpp}$) and those related directly to computation ($Instr_{oper:gpp}$). Through substitution of (1), (2), and (4), the speedup of an FPGA is given in (5).

TABLE 1
A Summary of Terms and Definitions for Quantitative Analysis of the Relative Speedup of Devices

Term	Definition
$Cycle_{xx}$	Clock cycles per iteration for device xx
N_{iter}	Number of iterations (outputs)
$Instr_{iter:xx}$	Instructions per iteration for device xx
$Instr_{sppt:xx}$	Support instructions for device xx
$Instr_{oper:xx}$	Calculation related instructions for device xx
$Ineff_{xx}$	Instruction inefficiency for device xx
$ItLP_{xx}$	Iteration level parallelism for device xx
CPI_{xx}	Average clock cycles per iteration for device xx
CPO_{xx}	Average clock cycles per output for device xx
$Speedup_{xx \rightarrow yy}$	Speedup of device xx over yy

$$Ineff_{gpp} = \frac{Instr_{sppt:gpp} + Instr_{oper:gpp}}{Instr_{oper:gpp}}, \tag{4}$$

$$Speedup_{fpga \rightarrow gpp} = ItLP_{fpga} \times Instr_{oper:gpp} \times Ineff_{gpp} \times CPI_{gpp}. \tag{5}$$

Equation (5) expresses three advantages of an FPGA. First, the capability to run multiple pipeline instances concurrently ($ItLP_{fpga}$). Second, the number of operation specific instructions ($Instr_{oper:gpp}$) for a GPP is often higher than FPGA ALU operations. This is due to the requirement to map an algorithm to the fixed ISA on the GPP. Also, some functions are "free" on an FPGA, e.g., "divide by two." Third, the inefficiency of a GPP ($Ineff_{gpp}$) due to a need for "support" instructions, e.g., program counter manipulation. Cycles per instruction (CPI) for a GPP may be less than one to indicate support for multiple instructions per clock cycle; this represents an advantage.

3 The Performance Comparison Challenge

In this section, the performance comparison of the GPU and FPGA is motivated. Three performance drivers are explained in Section 3.1. The relationship is quantified in Section 3.2.

3.1 Throughput Rate Drivers

The peak throughput rate (T_{peak}) of a video processing system is the maximum number of pixels that can be input, output, and processed per second. To understand the architectural features which influence T_{peak}, the relationship in (6) is used. T_{peak} is proportional to: $ItLP$, clock rate (C), and inversely CPO. Cycles per output is the average number of clock cycles between outputs for a single, iteration level, pipeline.

$$T_{peak} \propto \frac{ItLP \times C}{CPO}. \tag{6}$$

GPPs are designed with a high clock rate to implement a large instruction set of operations largely sequentially. A 6-15 times higher clock rate than FPGA implementations [5], and a six to eight times higher clock rate than a GPU [19], is typical of a GPP. For the case studies in this work, GPU clock rate is two to six times higher than that of FPGA implementations. Clock rate is compromised for reconfigurability on an FPGA.

TABLE 2
Architectural Strengths for Throughput Rate Drivers

	Cycles per Output	Iteration Level Parallelism	Clock Rate
General Purpose Processor	Low	Low	High
Graphics Processor	Medium	Medium–High	Medium
Reconfigurable Logic	High	High (Flexible)	Low

For a GPU, $ItLP_{gpu}$ equals the number of fragment processing pipelines. For example, the nVidia GeForce 6800 GT and 7900 GTX have an $ItLP_{gpu}$ of 16 and 24, respectively. In comparison, a GPP has an $ItLP_{gpp}$ of one. (Current devices have up to eight cores, which remain small with respect to a GPU.) An FPGA implementation is a trade-off between iteration level parallelism and operation level parallelism to best suit algorithmic requirements within device size constraints.

The number of cycles per output in an FPGA implementation is typically one to two orders of magnitude lower than for a processor [5]. There are two reasons for this: an FPGA has no instruction fetch and decode overheads; and operation level parallelism can be arbitrarily exploited (within resource constraints).

FPGA implementations achieve high peak throughput through the astute use of parallelism. This results in a low number of cycles per output. For moderately sized implementations, the number of cycles per output and iteration level parallelism often equals one. In this scenario, peak throughput equals the clock rate.

The GPU has a lower cycle per output count than a GPP due to a reduced instruction overhead.

Table 2 summarizes each architectures strengths in achieving a high throughput. The iteration level parallelism of the GPU is medium-high because, in general, a greater degree of iteration level parallelism may be exploited on an FPGA. This is exemplified in histogram bin calculation (case study HEqu) in which 256 "output" values are computed in parallel ($ItLP_{fpga} = 256$). In reality, the iteration level parallelism for the FPGA may also be higher than one for the other benchmarks in Table 4.

It is observed that the GPU provides a trade-off between the GPP and FPGA with respect to all factors.

3.2 Analysis of the Factors which Affect Speedup

The three factors in Table 2 are now used to quantify the relative performance, or speedup, of using a GPU, FPGA, or GPP.

Guo et al. [5] provide a useful starting point for comparing the GPU and FPGA. However, a number of differences between a GPP and a GPU must be addressed. A summary of key GPP architectural features detrimental to implementing video processing algorithms, is shown in Table 3, column one. The second column includes a summary of the differences, mostly beneficial, for a GPU.

Instruction inefficiency requires further explanation. The GPU's single program multiple data (SPMD) architecture model addresses the overhead of program counter manipulation. However, the SPMD model also introduces increased flow control overheads. Of particular concern is the performance penalty of conditional branching due to a large degree of multithreading [20].

TABLE 3
Benefits of a GPU over a General-Purpose Processor

General Purpose Processor	GPU
Large Overhead in Extracting Parallelism from an Algorithm	Parallelism is Inherent to the Fixed Data path of Multi-Processors
Instruction Inefficiency (Program Counter (PC), Flow Control, Data Load/Store)	SPMD (shared PC), *Increased Flow Control Overheads*, Data Load/Store Hidden by Multi-Threads (*Sometimes*)
Large Instruction Count	Specialised Vector ISA[1]
High Ratio of Cache to ALUs (Low Computational Density)	Low Ratio of Cache to ALUs (High Computational Density)

[1] The GeForce 8 and 9 generations of GPUs support only a scalar ISA

The data load instruction inefficiency overhead can also be reduced on a GPU through multithreading. The latency of a particular data fetch is "hidden" behind the processing or data fetches for other pixels within the same thread batch.

From initial consideration of the benefits in Table 3, the GPU appears a more favorable platform for implementing video processing algorithms than the GPP. Similar to an FPGA, the GPU can exploit a large degree of iteration level parallelism, while instruction inefficiency issues are overcome and instruction count ($Instr_{oper:gpu}$) minimized. The flow control overhead of a GPU is an equivalent disadvantage to the pipeline stalls observed for input-dependent flow control in FPGA implementations.

Equivalent formulations to (5), for the factors effecting the speedup of the GPU over a GPP, and of an FPGA over a GPU, are now derived. The latter is chosen arbitrarily and the GPU performance may exceed that of an FPGA (a "slowdown").

First an expression for the number of GPU clock cycles ($Cycles_{gpu}$) must be obtained. A GPU is referred to as a multiprocessor system. For this work, each "processor" is a single pipeline ($ItLP_{gpu}$). Parallel pipelines in the GPU are, in fact, closely coupled by SPMD and multithreaded operation. Equation (7) shows the representation of number of cycles for the GPU which includes the iteration level parallelism. The remainder of (7) is similar to the GPP cycle definition in (1).

$$Cycle_{gpu} = \frac{Instr_{iter:gpu} \times N_{iter} \times CPI_{gpu}}{ItLP_{gpu}}. \quad (7)$$

The inefficiency factor for the GPU is the same as that for the GPP in (4) with *gpp* substituted for *gpu*. A key difference in the interpretation of *inefficiency* is that GPU support instructions include only memory read and flow control operations. Swizzle flow control operations reorder data in processor registers and are often free (have no clock cycle penalty) [20].

A GPU's speedup over a GPP is shown in (8). This is the ratio of GPU ($Cycles_{gpu}$) to GPP ($Cycles_{gpp}$) clock cycles. The GPU speedup factors, over a GPP, can occur from: reduced instruction inefficiency (a fixed processor pipeline), reduced instructions per operation (ISA efficiency), reduced clock cycle per instruction (increased operation level parallelism), and iteration level parallelism (multiple fragment pipelines).

$$Speedup_{gpu \to gpp} = \frac{Ineff_{gpp}}{Ineff_{gpu}} \frac{Instr_{oper:gpp}}{Instr_{oper:gpu}} \times \frac{CPI_{gpp}}{CPI_{gpu}} ItLP_{gpu}. \quad (8)$$

Equations (9)-(11) show the derivation of the "speedup" of an FPGA over a GPU. A setup where FPGA clock cycles per output equals one is assumed. Equation (11) includes the effects of instruction inefficiency, cycles per instruction, number of instructions, and iteration level parallelism in isolation.

$$Speedup_{fpga \to gpu} = \frac{Cycle_{gpu}}{Cycle_{fpga}}, \quad (9)$$

$$= Instr_{iter:gpu} \times CPI_{gpu} \times \frac{ItLP_{fpga}}{ItLP_{gpu}}, \quad (10)$$

$$= Ineff_{gpu} \times Instr_{oper:gpu} \times CPI_{gpu} \times \frac{ItLP_{fpga}}{ItLP_{gpu}}, \quad (11)$$

$$= \frac{ItLP_{fpga}}{ItLP_{gpu}} \times \frac{Ineff_{gpu} \times Instr_{oper:gpu} \times CPI_{gpu}}{CPO_{fpga}}. \quad (12)$$

If FPGA cycles per output is not equal to one, for example, the non-full-search motion vector estimation (NFS-MVE) case study, speedup is as shown in (12). The numerator in the second fraction is GPU clock cycles per output.

The speedup factors presented above capture the performance differences between the GPU, FPGA, and the GPP. The primary concern is the performance difference between the GPU and FPGA in (11) and (12). The speedup factors over a "host" GPP are shown in (5) and (8). These formulae express the justification for using each device as an accelerator.

3.3 What About GeForce 8 Series GPUs?

The comparison methodology is unchanged for newer GPUs. Fundamentally, the architecture is unchanged in regard to iteration level parallelism, clock cycles per instruction, and instruction efficiencies. One note is that the processor cores operate at a faster clock rate (for example, 1,500 MHz in GeForce 9 series) in comparison to the remainder of the graphics core (600 MHz in GeForce 9 series). In the comparison methodology in Section 3, the processor core clock rate should be considered as the reference clock rate. It is observed that this will increase (by 2.5 times for the GeForce 9) the processor memory access time (in clock cycles).

4 Case Study Algorithms

The five case study algorithms considered in this work are characterized by arithmetic complexity, memory access requirements, and data dependence as shown in Table 4. A detailed discussion of each algorithm is omitted because the characterization in Table 4 is sufficient to understand this paper. The reader is directed to [1], [2], [21], [22] for further algorithm details.

Arithmetic complexity includes the proportion of mathematic, branching, and vector operations.

Memory access requirements are analyzed as total number, access pattern, reuse potential, and memory intensity. Reuse potential for video processing is the number of input frame pixels which are used to compute two outputs with a Manhattan distance of one pixel location. Memory intensity is the proportion of memory accesses to the number of arithmetic instructions.

Data dependence identifies the operations which must be computed before the next operand. An example is the creation of histogram bins which requires an accumulate for

TABLE 4
Algorithm Characterization for Five Case Studies (Algorithms Taken from [1], [2], [21], [22])

Algorithm	Arithmetic Complexity			Memory Access Requirements				Data Dependence
	Math	Branch	Vector	Number (per output†)	Memory Intensity	Pattern	Reuse Potential	
Primary Colour Correction	low	low	medium					
Input Correction (IC)	medium	low	medium	1	low	predictable	$\frac{0}{1}$	null
Histogram Correct (HC)	medium	high	low					
Colour Balance (CB)	medium	medium	medium					
2D Convolution	null	null	medium	N^2	high	predictable	$\frac{N(N-1)}{N^2}$	low
Video Frame Resizing	null	null	high	16	medium	predictable	$\frac{0}{16}$ to $\frac{16}{16}$ (variable)	low
Histogram Equalisation								
Generation (256 bins)	null	null	low	XY‡		predictable	$\frac{XY}{XY}$	high
Intensity Equalisation	null	null	null	$1+1$§	high	random	$\frac{0}{1}+\frac{\sigma\sharp}{1\S}$	null
3-step Non-Full-Search Motion Vector Estimation	low	medium	high	$16^2(9+\ldots 8+8)+16^2$	medium	locally random	$\frac{44(44-16)}{44^2}$, $\frac{0}{16^2}$	medium

† The term output is used to indicate that for each function the output may not be a pixel in an output frame. For Histogram Equalisation Generation there are 256 bin outputs from the algorithm. For 3-step NFS-MVE there are $\frac{X}{16} \times \frac{Y}{16}$ motion vector outputs from the algorithm. For all other algorithms the outputs are individual pixels in an output frame. ‡ The product XY is the input frame resolution. § A single memory access from a 256 location lookup table. ♯ The variable σ indicates a dependence on the variance of the intensity when accessing the 256 location lookup table mentioned in note §.
Key: N - dimensionality of 2D convolution ($N \times N$), X,Y - respectively the width and height of the video (measured in pixels).

each memory access before the next accessed memory location can be processed.

It is observed that Table 4 shows five algorithms which fill a broad spectrum of the characterization space. The algorithms and their characteristics will be referenced throughout this paper.

Fig. 1 shows a summary of throughput performance for the case studies. The performance is expressed in millions of generated output pixels per second. An exception is three-step NFS-MVE where performance is expressed relative to input frame pixels processed per second.

The GPU exhibits a large degree of variance in throughput performance over the case study algorithms. In contrast, FPGA implementations have a more uniform performance trend. This is due to two factors. First, the exploitation of arbitrary data-path pipelining and parallelism in FPGA implementations. Second, the constraint on GPU throughput rate of the temporal processing nature of software implementations. Other factors include the behavior of the memory subsystem and device clock speed.

The performance for each algorithm will be interrogated later in the paper. However, the following questions emerge, from the performance results in Fig. 1, which motivate further analysis.

1. What is the source of high throughput for the GPU and FPGA for "compute bound" algorithms, particularly in comparison to a GPP?
2. The GPU performance degrades sharply with window size for 2D convolution. What is the clock cycle overhead of "memory access bound" algorithms for the GPU?
3. What factors affect the variability, for each device, in throughput for resizing algorithms? Put more generally, how does variable data reuse affect performance?

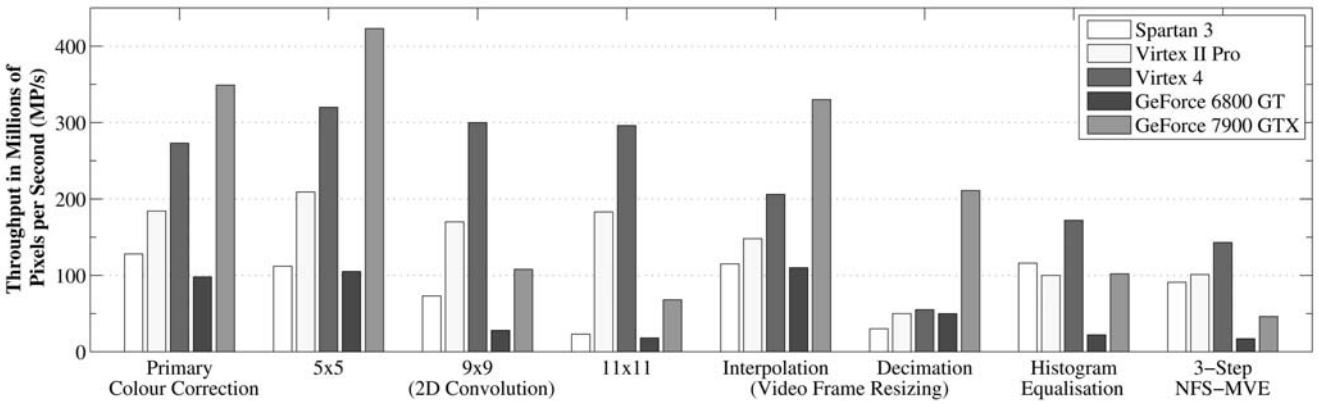

Fig. 1. A summary of throughput performance for case study algorithms implemented on sample nVidia GPUs and Xilinx FPGAs. Hereafter, the algorithms are referred to as: primary color correction (PCCR), 2D convolution (Conv), video frame resizing (Resizing), interpolation (Interp), decimation (Deci), histogram equalization (HEqu), and three-step non-full-search motion vector estimation (NFS-MVE).

4. What GPU features lead to a low performance for a NFS-MVE implementation with respect to the HEqu case study?

Further analysis is concerned with quantifying key performance drivers. We focus on the Virtex-4 and GeForce 7900 GTX. Both adopt 90-nm technology and are used as a case study comparison. It would be straightforward, given equivalent implementations, to extend the comparison to newer technologies, including the nVidia GeForce 9, Xilinx Virtex-5, or Altera Stratix IV devices.

Throughput rate alone is insufficient for a detailed analysis. Therefore, to answer questions 1-4, the analysis proceeds as follows: In Section 5.1, arithmetic complexity is considered to address question 1. Questions 2 and 3 are addressed through exploring memory access requirements in Section 5.2. In Section 5.3, question 4 is addressed though considering performance implications of data dependence. The formulation from Section 3.2 is used throughout. Throughput rate drivers are quantified in summary of the results in Section 6.

5 Case Study: Factors Affecting Speedup on the GeForce 7900 GTX and Virtex-4

5.1 Experimental Setup

For the FPGA, VHDL and Xilinx ISE 8.2i are used. Post place and route timing is taken to determine throughput. The results are verified using high definition video samples [23]. Uniform input variables for the case studies are chosen to be 18-bit fixed-point representation and the design is bit width optimized to less than $\frac{1}{2}$LSB of error. This setup maximally utilizes the functional bit width of embedded multipliers. Aesthetically satisfying results are observed for all case studies.

All FPGA case studies, with the exception of NFS-MVE for which the off-chip memory access requirements for the performance quoted here are justified in [4], are streamed raster-scan implementation. It can be assumed that enough off-chip FPGA memory access bandwidth is available to stream the video data. Therefore, in-system performance analysis results are omitted.

For the GPU, the nVidia cgc compiler 1.5 and OpenGL are used. The results are verified against a "golden" C++ model. Video frames are uploaded as image textures to on-card video memory and results written back to the same video memory. In turn, the results are read back to the host CPU for verification. The storage format for video frame data is 8-bit per three-component RGB. For intermediate processing results, the floating point format is used to satisfy the dynamic range. Storage format affects video card data upload and download time, also the cache and memory access behavior of the GPU. Four factors, of upload, download, execution, and pipeline setup time, contribute to GPU compute time. In this work, execution time, obtained using the OpenGL timing query extension, is quoted. A mean of 200 samples is taken to improve accuracy. Additionally, pipeline setup time is considered for multi-pass algorithms in Sections 5.2 and 5.3. Fig. 1 includes both execution and pipeline setup time.

A single core 3.0 GHz Pentium 4 with 1 GB of RAM is the "host" CPU. For PCCR case study, CPU [GPP] implementations are coded in C++ and utilize the MMX/SSE instruction set.

TABLE 5
Instruction Set Efficiency of the Nvidia GeForce 7900 GTX GPU Relative to an Intel 3.0 GHz Pentium 4 Processor

	GPU			GPP		
	CPO	CPI	Ineff	CPO	CPI	Ineff
Full	44.7	0.402	1.649	445	1.049	9.925
IC	14.3	0.432	1.194	113	1.022	6.058
HC	10.2	0.57	2.545	119	1.030	25.674
CB	17.2	0.35	1.667	128	1.022	6.262
R2Y	5.2	0.58	1.500	31	1.065	4.158
Y2R	4.3	0.61	1.333	26	1.022	5.087

TABLE 6
Number of MacroBlock Operations versus Processor Instruction Count for Primary Color Correction

	Macro-Block Operations (FPGA)	$\text{Instr}_{oper:gpu}$ ($\text{Instr}_{iter:gpu}$)	$\text{Instr}_{oper:gpp}$ ($\text{Instr}_{iter:gpp}$)
Full	187	74 (122)	42.75 (424.31)
IC	61	31 (37)	18.25 (110.56)
HC	16	11 (28)	4.5 (115.53)
CB	64	36 (60)	20 (125.23)
R2Y	17	8 (12)	7 (29.10)
Y2R	29	6 (8)	5 (25.44)

For each design, sensible performance optimizations are made.

Board upload and download time is omitted for each device. The aim is to compare the advantages of each device, abstract from a given implementation in for example a video card.

5.1.1 Feature 1: Arithmetic Complexity

The capability of a GPU or FPGA to achieve "speedup," for arithmetic operations, over a GPP is determined by the parallelism which can be extracted from an algorithm. Of fortune is that video processing algorithms are often amenable to large degrees of parallelization. There are, however, constraints for each device.

An FPGA is constrained by the efficiency with which an algorithm is mapped onto computational resources. For a GPU, a fixed instruction set architecture must be exploited efficiently.

The arithmetic capabilities of the GPU and FPGA are now analyzed through considering the arithmetic intensive subblocks of the PCCR case study. These include those detailed in Table 4 plus two color space conversions (R2Y and Y2R [1]).

The factors affecting speedup, specifically processor instruction inefficiency issues, are explored in Section 5.1.1. In Section 5.1.2, nonarithmetic intensive algorithms are considered.

5.1.2 Intrinsic Computational Benefits over a GPP

First features of the GPU ISA which result in an impressive speedup over a GPP, rivaling or exceeding that of an FPGA, are presented.

A summary of the ISA efficiency features of the GPU, relative to a GPP, are shown in Table 5. The key features are the cycles per instruction and the inefficiency factor of each processor as defined in Section 3.2. Number of operational instructions is shown in Table 6. Instruction disassembly for the Pentium 4 GPP is taken using Intel VTune 3.0.

TABLE 7
Speedup for Primary Color Correction

	Speedup $_{fpga \rightarrow gpp}$	Speedup$_{gpu \rightarrow gpp}$ (less ItLP$_{gpu}$)	Speedup$_{fpga \rightarrow gpu}$ (CPI$_{gpu} \times$Ineff$_{gpu}$)
Full	445	283 (11.8)	1.86 (0.60)
IC	113	190 (7.9)	0.59 (0.46)
HC	119	281 (11.7)	0.42 (0.92)
CB	128	178 (7.4)	0.72 (0.48)
R2Y	31	142 (5.9)	0.22 (0.65)
Y2R	26	145 (6.0)	0.18 (0.72)

The number of operational instructions required by the GPP is lower than that for the GPU. There are two reasons. First, four pixels are processed in parallel on the GPP. This fully utilizes the 4-vector nature of SSE instructions. On the GPU output pixel values are calculated in isolation and typically only three of the maximum four ALU inputs are utilized. Second, the GPU instruction set is smaller than that of the GPP and may require more than one ALU or math instruction to implement the same operation. For example, a square root operation requires a reciprocal square root followed by a reciprocal instruction.

Despite the increase in operational instructions by approximately 50 percent, a 2 times decrease in CPI, and 12 times decrease in CPO are observed for the GPU over a GPP. The difference in cycles per instruction is attributed to the large operation level parallelism of the GPU [20]. A reduction of up to 10 times in the instruction inefficiency ($Ineff_{xx}$) measure contributes to the remainder of the difference.

Next, we compare processor instruction count ($\mathrm{Instr}_{oper:xx}$) to the number of "macroblock" ALU operations in an FPGA implementation. This is exemplified for PCCR in Table 6. FPGA macroblock operations are counted as, for example, single add, multiply, and comparison instantiations. A three to four times higher number of operations is required for an FPGA than the processors. This is due to the fine granularity of FPGA macroblocks with respect to the vectorized ISA of the processors.

For a more precise comparison, the total instructions per iteration ($Instr_{iter:xx}$) is included in Table 6. A GPP requires a total number of instructions up to four times higher than the number of FPGA operations. This is similar to observations made by Guo et al. [5]. The total instruction count is lower for a GPU than for FPGA operations. This is due to the GPU's low instruction inefficiency of up to a factor of 10 times relative to the GPP.

The speedup factors for the PCCR case study are shown in Table 7, where $ItLP_{fpga} = 1$ and $ItLP_{gpu} = 24$. A GPU provides up to a 280 times speedup over a GPP. If iteration level parallelism is removed, peak speedup is 12 times.

A peak 445 times speedup is observed for an FPGA over the GPP. The speedup is due to the high instruction inefficiency of the GPP and higher operation level parallelism on the FPGA.

The speedup of an FPGA over the GPU is significantly less than that for each device over the GPP. A range of two times speedup for the FPGA to five times slow down is observed. This is expected because both provide a large arithmetic operation speedup over the GPP. A high degree of iteration level parallelism and a low number of cycles per instruction on the GPU are a good match to an FPGA's operation level parallelism. Note $ItLP_{fpga} = 1$, the FPGA is quite underutilized for all but "Full."

TABLE 8
A Summary of Number of Cycles per Operational Instruction ($CPI_{gpu} \times Ineff_{gpu}$) for the GPU for the Case Studies

	CPI_{gpu}	$Ineff_{gpu}$	$CPI_{gpu} \times Ineff_{gpu}$
PCCR	0.40	1.65	0.66
Conv (5×5)	0.50	1.51	0.76
Conv (9×9)	0.59	1.50	0.89
Conv (11×11)	0.64	1.50	0.96
Interp (avg)	0.47	1.24	0.58
Deci (avg)	1.00	1.24	1.24
HEqu	0.41	1.77	0.73
NFS-MVE	0.71	1.91	1.36

It is interesting to observe the product of CPI_{gpu} and $Ineff_{gpu}$ as shown in the fourth column of Table 7. This product represents the average number of clock cycles taken per operational instruction ($Instr_{gpu:oper}$). It is observed that for each PCCR block, the GPU produces up to two computational results per clock cycle per iteration. This is the origin of the impressive performance observed for the GPU for implementations of computationally intensive algorithms.

5.1.3 Nonarithmetic Intensive Algorithms

The above results focus on the scenario where GPU support instruction overhead is negligible. For the general case, this condition may not hold. The overall cycle per instruction (CPI_{gpu}), inefficiency ($Ineff_{gpu}$), and cycles per operational instruction ($CPI_{gpu} \times Ineff_{gpu}$) for all algorithms is summarized in Table 8.

The GPU inefficiency is observed, in Table 8, to be consistently low from 1.24 to 1.91. This is an order of magnitude lower than the GPP inefficiency observed by Guo et al. of 6-47 [5].

The computationally bound PCCR algorithm requires the lowest clock cycle per GPU instruction (CPI_{gpu}) of all case studies.

A small rise in cycles per instruction (CPI_{gpu}) is observed with increasing convolution size. With all other factors constant, this translates to a marginal rise in the memory access "cost." A more significant difference is observed between interpolation and decimation of a factor of two times. This represents a significant rise in the memory access instruction overhead.

The HEqu and NFS-MVE case studies both contain a large proportion of memory access and flow control. This results in a large inefficiency factor. The instruction per operational instruction for NFS-MVE is two times that for HEqu.

Decimation and NFS-MVE have a high cycle per operational instruction count ($CPI_{gpu} \times Ineff_{gpu}$) relative to the other algorithms, of less than one operation result per cycle. This is investigated in Sections 5.2 and 5.3, respectively. Other case studies produce up to two results per clock cycle.

5.1.4 GeForce 8 Series GPUs

For GeForce 8 GPUs instruction inefficiency would reduce due to the scalar processing units. This is because of the prior under utilization of 4-vector ISA as cited in Section 5.1. A downside of this is that the GPU instruction count will increase due to scalar computation. However, this overhead

is likely to be hidden by the SIMD nature of the GPU architecture.

One important point to highlight about the scalar processors on GeForce 8 GPUs is that operation level parallelism (peak bandwidth) is reduced from the 4-vector ISA of the GeForce 7. This, however, has the additional intrinsic benefit of a higher clock frequency due to a simpler processor architecture.

The overhead of flow control, as mentioned in Section 3.2, remains in newer generations of GPUs [3]. For arithmetic intensive algorithms it is expected that number of cycles per output will only reduce slightly due to a small increased in instruction efficiency from the scalar ISA.

5.2 Feature 2: Memory Access Requirements

Memory access requirements are considered in terms of on-chip and off-chip accesses. For an FPGA implementation, the memory system is a part of a specially designed data path. The GPU memory system is fixed and optimized for graphics rendering.

The challenges associated with on-chip and off-chip memory accesses for the target devices are described in Sections 5.2.1 and 5.2.2, respectively. Two specific memory access case study issues are subsequently considered. GPU input bandwidth limitations are explored through the 2D convolution case study in Section 5.2.3. In Section 5.2.4, the effect of variable data reuse is analyzed using the video frame resizing case study.

5.2.1 On-Chip Memory Accesses

GPU on-chip memory access refers to texture accesses for each output pixel in each fragment pipeline. The memory hierarchy includes L1 cache local to texture filter blocks and multiple shared L2 cache blocks [14]. Input memory bandwidth (IBW_{gpu}) can be estimated from the number of clock cycles per internal memory accesses (CPIMA) as shown in (13) and (14). The term "input bandwidth" is used because off-chip bandwidth also affect this value.

$$CPIMA_{gpu} = \frac{CPO_{gpu}}{Instr_{mem:gpu}}. \tag{13}$$

$$IBW_{gpu} = \frac{1}{CPIMA_{gpu}} \times ItLP_{gpu} \times C_{gpu} \times 4\ Bytes\ per\ pixel. \tag{14}$$

An FPGA is often considered to have infinite on-chip bandwidth, because an FPGA is fundamentally memory (lookup tables and registers). Consider on-chip memory as only on-chip embedded RAM (named Block Select RAM (BRAM) on Xilinx devices). The peak on-chip bandwidth, on the moderately sized XC4VSX25 Virtex-4, is then 288 GB/s. This is five times higher than the value estimated for the GeForce 7900 GTX (68 GB/sec, estimated through repeated reads to one memory location).

The total on-chip memory access requirements for each case study are shown in Table 9. GPU and FPGA requirements are equal for PCCR, 2D convolution, and resizing. Differences occur in the multipass implementations of HEqu and NFS-MVE. An increase of approximately 10 and 2 times, respectively, for the GPU, relative to FPGA, is required. These overheads are due to data dependencies as explained in Section 5.3.

TABLE 9
On-Chip Memory Access Requirements for GPU and FPGA Implementations of the Case Study Algorithms

	GPU On-Chip Reads	FPGA On-Chip Reads
PCCR	XY	XY
Conv ($n \times n$)	XYn^2	XYn^2
Resizing (2D)	$16XY/s_x s_y$	$16XY/s_x s_y$
Resizing (1D)	$XY4(1+\frac{1}{s_{max}})$	$XY4(1+\frac{1}{s_{max}})$
HEqu	38XY	4XY
NFS-MVE	58XY	27XY

s_x, s_y are the resizing factors with minimum value $s_{min} = \min(s_x, s_y)$ and maximum value $s_{max} = \max(s_x, s_y)$. The frame size is XY.

TABLE 10
Off-Chip Memory Access Requirements for GPU and FPGA Implementations of the Case Study Algorithms

	GPU		FPGA	
	Reads (min)	Writes	Reads	Writes
PCCR	XY	XY	XY	XY
Conv ($n \times n$)	XY	XY	XY	XY
Resizing (2D)	XY	$\frac{1}{s_x s_y}$XY	XY	$\frac{1}{s_x s_y}$XY
Resizing (1D)	s'XY	s''XY	XY	$\frac{1}{s_x s_y}$XY
HEqu	$\sim$ 3XY	$\sim$ 2XY	2XY	XY
NFS-MVE	$\sim$ 8XY	$\sim$ 2XY	2XY	$\frac{1}{16^2}$XY

Additionally to notes in Table IX: $s' = (1+\frac{1}{s_{max}})$, $s'' = \frac{1}{s_{max}}(1+\frac{1}{s_{min}})$.

5.2.2 Off-Chip Memory Accesses

Off-chip memory accesses are to a large quantity of DRAM located "off-die" from a device.

Traditionally, GPUs require a high memory bandwidth to support graphics rendering. Object vertices, input textures, intermediate fragment information, render target pixels, and processor instruction code must be stored. The GeForce 7900 GTX has a peak memory bandwidth of 51.2 GB/sec [19]. For comparison, this is an order of magnitude higher than a GPP (6 GB/sec for a 3 GHz Pentium 4 [24]). The capability to efficiently utilize this off-chip "pin speed" dictates achievable bandwidth.

The predictable nature of video processing memory access patterns can be exploited to minimize FPGA off-chip memory access requirements. The required off-chip bandwidth is often one or two orders of magnitude less than that required for a GPP [24] and a GPU, respectively. As a result, FPGA memory PHYs may be run at a half or quarter the data rate of GPU DRAM interfaces.

Off-chip memory access requirements, for each case study, are shown in Table 10. FPGA implementations exploit maximal data reuse. The FPGA on-chip memory can be tailored application specifically, whereas GPU on-chip memory arrangement is fixed. Each off-chip location is written to or read from once. Only the minimum off-chip read requirements of the GPU are shown in Table 10. The actual number is subject to cache behavior.

For PCCR, Conv and Resizing (2D), off-chip memory access requirements are comparable between the FPGA and GPU.

A difference in off-chip access requirements occurs for 1D (separable) resizing. This is due to the intermediate write of results to off-chip memory between "1D" passes. In an FPGA implementation, these data are buffered in on-chip line buffers. The multipass HEqu and NFS-MVE case studies

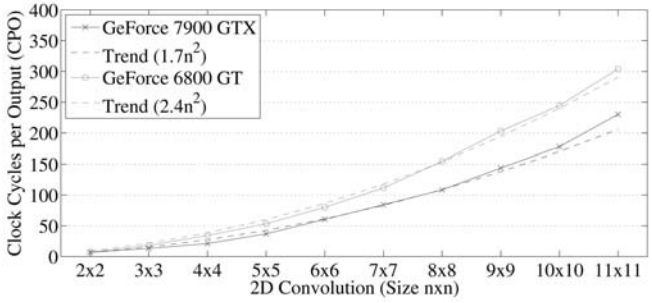

Fig. 2. Clock cycles per output for 2D convolution implementations on the GPU (determining GPU input bandwidth).

also require a large GPU off-chip memory bandwidth of up to 4 times for reads and 2×16^2 times for writes. The effect is minimized in our performance comparison due to the high (51.2 GB/sec) off-chip memory bandwidth of the GPU. For future systems, for example a system-on-chip, off-chip memory access overhead is an increasingly significant factor. On GeForce 8 GPU off-chip memory access requirement is reduced by an advanced on-chip memory system. The support for this has its own overheads and multipass implementations are still required [13].

5.2.3 Case Study: Input Bandwidth

For 2D convolution, the memory access requirements are comparable for both devices. This makes it a desirable case study to compare performance impacts of changing number of internal memory accesses.

In Table 8, it is shown that the GPU CPI are approximately the same for all 2D convolution kernel sizes. (A marginal rise occurs, from a value of 0.50 to 0.63, between 3×3 and 11×11 sized kernels.) The inefficiency measure is identical over all kernel sizes at a value of 1.5. Instruction set inefficiency, therefore, has no effect on the $0.07n^2$ speedup for the FPGA implementation over the GPU. To identify the factors which do influence the speedup, the number of cycles per output is plotted over varying 2D convolution sizes in Fig. 2.

The GPU has a consistent cache performance for 2D convolution over varying sizes and is bounded by the number of memory access requests. From this observation, the GPU performance is estimated using the trend lines in Fig. 2.

A performance trend of $1.7n^2$ and $2.4n^2$ matches the performance of the GeForce 7900 GTX and GeForce 6800 GT, respectively. This value (1.7 or 2.4) is the clock cycles per internal memory access ($CPIMA_{gpu}$) shown in (13). It follows that the achieved internal bandwidth (IBW_{gpu}) of the GeForce 7900 GTX and GeForce 6800 GT is estimated, using (14), to be 36.7 and 9.33 GB/s, respectively.

For the FPGA, an unbounded input bandwidth is observed due to pipelined line buffer reads, so there is no clock cycle penalty as n is increased. The clock cycle cost is, in fact, an increased latency. The result is a throughput, for FPGA implementations, which is approximately linear when compared to the GPU. This is due to flexible parallelism, pipelining, and streaming of data. The penalty is an increasing resource requirement.

For maximum data reuse, the FPGA implementation stores pixels in line buffers. For a 720p frame size, $720 \times (n-1) \times 24$ bits (8 bits per color space component) are required, if a vertical raster-scan input is assumed. The memory resource requirement exceeds the GeForce 7900 GTX predicted cache size (128 KBytes [14]) for convolution kernel sizes greater than 7×7 pixels. As convolution size increases, on-chip block memory usage and routing resources increase. This results in timing closure at lower frequencies for increasing n in Fig. 1.

TABLE 11
Input Bandwidth (IBW_{gpu} in GB/sec) for 2D and 1D Resizing Implementation Methods on a GeForce 7900 GTX GPU

Input	Output	2D Method CPO_{gpu}	2D Method IBW_{gpu}	1D Method CPO_{gpu}	1D Method IBW_{gpu}
Interpolation					
576p	720p	47.90	20.84	21.14	18.95
720p	1080p	47.77	20.90	21.96	18.94
480p	720p	47.90	20.84	18.83	19.49
576p	1080p	47.32	21.10	18.45	18.96
480p	1080p	47.32	21.10	17.57	19.16
Decimation					
1080p	480p	101.56	9.83	61.29	12.91
1080p	576p	83.93	11.90	47.14	15.21
720p	480p	68.05	14.67	34.53	18.07
1080p	720p	65.85	15.16	33.85	18.43
720p	576p	62.41	16.00	30.96	18.13

The worst case scenario computational resource requirement (multipliers and addition) for FPGA increases as a factor of $O(n^2)$. However, this factor is in practice reduced due to resource usage optimizations [25], [26], [27] and convolution matrix sparsity.

5.2.4 Case Study: The Effect of Variable Data Reuse Potential

The variable data reuse requirements of the resizing case study make it suitable for analyzing on-chip memory flexibility.

An FPGA implementation of interpolation is limited by off-chip memory writes. The required output bandwidth is higher than the required input bandwidth. For decimation, this scenario is reversed and the limiting factor is input bandwidth. The setup used here is a single pipeline ($ItLP_{fpga} = 1$) with a system clock rate fixed at the output bandwidth requirement. For interpolation, one pixel is clocked out per clock cycle. For decimation, $1/s_x s_y$ pixels are clocked out per clock cycle, where s_x and s_y are the horizontal and vertical resizing factors, respectively.

Separability of the resizing algorithm into two 1D passes is shown above to result in a variation in on-chip and off-chip memory access requirements. For the FPGA implementation, the effect of this is a reduction in multiplier resource usage proportional to two times. The effect on the GPU is observed to be a significant change in input bandwidth as explained below.

A summary of the performance improvement from the 1D method is shown in Table 11. The clock cycles per output of the separable implementation is the combination of the two 1D passes. A significant 11 GB/sec of variation in input bandwidth is observed between the 2D methods for interpolation and decimation. The memory access performance drops by a factor of two times over the sample resizing ratios.

Between differing interpolation ratios, memory access behavior is consistent. Considering the data reuse potential characteristic (in Table 4), this is because full data reuse or a step of one Manhattan location occurs between neighboring

TABLE 12
Speedup of FPGA (Virtex-4) versus the GPU (GeForce 7900 GTX) for Varying Bi-Cubic Resizing Ratios (1D Method)

Input	Output	$s_x s_y$	CPO_{fpga}	CPO_{gpu}	Speedup
Interpolation					
576p	720p	2.08	1	21.14	0.88
720p	1080p	2.25	1	21.96	0.92
480p	720p	3.00	1	18.83	0.78
576p	1080p	4.69	1	18.45	0.77
480p	1080p	6.75	1	17.57	0.73
Decimation					
1080p	480p	0.15	6.75	61.29	0.38
1080p	576p	0.21	4.69	47.14	0.42
720p	480p	0.33	3.00	34.53	0.48
1080p	720p	0.44	2.25	33.85	0.63
720p	576p	0.48	2.08	30.96	0.62

outputs. With an increased interpolation factor, the ratio of the occurrence of full data reuse to a step of one rises. This results in a small increase in performance as one traverses down Table 11. For decimation, the pattern is different. The lower data reuse potential of decimation, with respect to interpolation, is the reason for this increase. As the decimation factor falls (moving down Table 11) performance improves. This is due to an increase in the average data reuse potential between neighboring output pixels.

Up to 1.6 times improvement in internal memory bandwidth is observed for the 1D method of decimation. This is due to improved cache performance where cache misses only occur due to one resizing dimension on each pass. The result is up to 2.5 times reduction in number of cycles per output. For interpolation, a 1.1 times lower achievable input bandwidth is observed. The reduction is due to the cache performance already being efficient for interpolation. Separation into two passes actually reduces reuse potential by necessitating steps of one location between output pixels, in one dimension, on each pass. Despite this, the result is over two times reduction in cycles per output.

A peak 2.5 times improvement is achieved from using the 1D resizing method. A limitation of the separation of an algorithm on a GPU is an increase in the overhead from the host processor setup and pipeline control. A difference of the order of one to two milliseconds is observed between 1D and 2D case studies.

The speedup of an FPGA over a GPU for varying resizing ratios is shown in Table 12 with the multipass overhead omitted.

A factor of up to 2.6 times speedup in favor of the GPU is observed between the two devices. The FPGA performance is bound by the number of cycles per output ($CPO_{fpga} = \frac{1}{s_x s_y}$) for decimation. Although the cycles per output for the FPGA is up to 21 times lower than the GPU, this is insufficient to surpass the GPU's iteration level parallelism benefit of 24 times.

If the clock speed ratio of over three times is included, the GPU has up to eight times performance benefit over the FPGA.

5.2.5 GeForce 8 and 9 Series GPUs

Memory organization and flexibility is cited as a major improvement in GeForce 8 GPUs.

In [28], nVidia promote their 2D convolution performance. Additionally to improvements in clock speed, memory bandwidth, and iteration level parallelism, a two times performance improvement is observed between using a naive and optimized algorithm. In this case, the naive approach is equivalent to the standard approach taken for convolution and resizing in Section 5.2. This shows that for filter algorithms approximately two times performance improvement is observed from the change in architecture.

The off-chip bandwidth performance has doubled to 64 GB/s for the GeForce 9 series GPU. This is likely to improve performance for off-chip bandwidth limited applications such as decimation and histogram equalization.

5.3 Feature 3: Data Dependence

In this section, the effect of data dependence is analyzed.

An FPGA has an inherent advantage for achieving speedup under data dependence constraints. The fine-grained architecture means that a specially designed data path can be implemented to exploit parallelism in the dimension that data dependence does not restrict. For NFS-MVE, four motion vectors are computed in parallel processing elements. Each processing element computes operations serially. For histogram generation, all 256 intensity bins are calculated in parallel. Data-path specialization provides favorable flexibility for these two case studies.

For a GPU, the architecture has a fixed degree of iteration and operation level parallelism determined by the number and nature of fragment pipelines. The fixed memory hierarchy, multithreading, and feed-forward programming model, also affect design choice. The design constraints and considerations are numerous. However, with astute implementation choice an impressive performance is achieved for HEqu and NFS-MVE. These are algorithms one may consider unmatched to the GPU.

Section 5.3.1 presents working examples of GPU optimization in the presence of data dependence using the HEqu and NFS-MVE case studies. The techniques are compared to the FPGA strategy in Section 5.3.2.

5.3.1 Optimizing for Data Dependence

The key component of the NFS-MVE algorithm implementation on a GPU is a reduction function [29], in which a nonoverlapping or partially overlapping window of input frame pixels is required from search and reference frames to calculate each output pixel. For HEqu, a similar scenario exists from one input frame. Overlaps can be controlled to be larger for HEqu than for NFS-MVE because input frame pixel order is not important. In both cases, a large reduction operation is required. A method 1 and method 2 scenario is considered for each case study as follows:

In method 1, the inner reduction is performed in full for each case study. For HEqu, this results in an on-chip read over a 16×9 window of pixels. For NFS-MVE, this results in a read over a window of 16×16 pixels (in both search and reference frames).

In method 2, each reduction is divided into two nested rendering passes with equally sized reductions.

Table 13 summarizes the performance for each method.

The CPO difference between methods 1 and 2 for NFS-MVE, in Table 13, is approximately three times. This is significantly greater than that for HEqu, for which a negligible difference is observed. For each case study, the total number of instructions per output and inefficiency

TABLE 13
Performance Summary for Alternative Reduction Options for Implementing Histogram Equalization and Non-Full-Search MVE on the GeForce 7900 GTX GPU for a 720p Video Frame

	Outputs	Instructions per Output (Inefficiency)	CPO	CPI	CPIMA (On-chip Reads per Output)
Histogram Equalisation (HEqu)					
Method 1 (4 pass) - one reduction over a 16×16 *window*					
1.1	6400	1049 (1.75)	1048	0.99	7.28 (144)
Full	912600	252 (1.54)	102	0.41	2.69 (38)
Method 2 (5 pass) - two reductions over 4×4 *windows*					
2.1	76800	58 (1.26)	56	0.96	4.64 (12)
2.2	6400	33 (1.57)	63	1.92	5.28 (12)
Full	912600	250 (1.52)	100	0.40	2.64 (38)
3-step Non-Full Search Motion Vector Estimation (NFS-MVE)					
Method 1 (2 step, 3 pass) - one reduction over a 16×9 *window*					
1.1	32400	2k6 (1.60)	3k8	1.43	7.32 (513)
Full	3600	71k (2.00)	133k9	1.88	9.65 (13881)
Method 2 (3 step, 3 pass) - two reductions over 4×3 *windows*					
2.1	518400	155 (1.46)	75	0.48	2.26 (33)
2.2	32400	39 (1.70)	147	3.79	9.24 (16)
Full	3600	68k (1.91)	48.5k	0.71	3.30 (14718)

factors are approximately equal for each method. The performance deviation is a result of memory access behavior. This is reflected directly in the change in CPI and CPIMA in columns five and six of Table 13.

The HEqu case study has equal number of on-chip and off-chip reads for each method. The total on-chip reads, per output, for NFS-MVE actually rises between methods 1 and 2, despite this the performance improves. Total number of off-chip memory writes also increases between methods 1 and 2 for NFS-MVE. The NFS-MVE case study is now analyzed further.

The range of required memory locations for MVE step 1.1 is summarized, for a 1D case, on the left-hand side of Fig. 3. Each kernel function (output) requires 16 search frame pixels. This is the same number of pixels for three neighboring outputs (e.g., between outputs 1, 2, and 3) but nonoverlapping between neighboring sets of three outputs (e.g., between sets $\{1, 2, 3\}$ and $\{4, 5, 6\}$). For the same kernel, 16 pixels from a reference frame are required. For three neighboring outputs, an increasing overlap of 8, 12, or 14 pixels occurs for passes 1, 2 and 3, respectively. It is important to note the large steps in memory access locations, for reference and search frames, between neighboring outputs.

On the right-hand side of Fig. 3, the access pattern for method 2 is summarized. This is an expansion of outputs 1, 2, and 3 from method 1, which now require a total of 12 outputs. The important feature to notice is that the range of required search and reference frame locations between n outputs is reduced by a factor of three and four for the reference and search frames, respectively. This reduces the large memory access steps from method 1. The memory access jumps between each set of 3 outputs is now a jump between sets of 12 outputs. The penalty is the need to combine these results in step 2.2 (see Table 13).

The issues associated with the full 2D memory access are an extension of the representation in Fig. 3. In the 2D case, the reduction in memory access range and increase in required number of outputs is the square of the 1D case.

It is observed in Table 13, when comparing steps 1.1 and 2.1 for the NFS-MVE case study, that the optimization

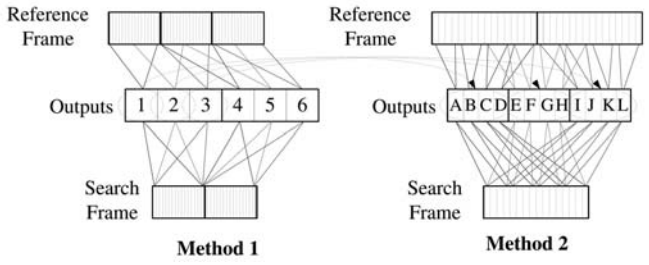

Fig. 3. 1D representation of the memory access requirement for the motion vector estimation case study for methods 1 and 2 (pass 1).

detailed above provides a three times improvement in memory system performance (CPIMA). The penalty is observed in step 2.2 where the memory access performance is actually worsened. No data reuse is possible between outputs. This is because in the 1D case of Fig. 3, there is no overlap when accumulating the sets of outputs $\{A, B, C, D\}$, $\{E, F, G, H\}$, and $\{I, J, K, L\}$. For the 2D case, the reduction is over nonoverlapping 4×4 windows. Despite this, the required number of reads per output and instruction count is significantly low such that the penalty is amortized by the performance benefit provided by step 2.1.

5.3.2 A Comparison to the FPGA "Strategy"

It is interesting to consider how similar memory access challenges are overcome in a customized data-path implementation on an FPGA. The key difference is that the content of memory, and the organization of memory, is fully controllable. This produces a large design space of possible optimizations for the memory hierarchy, as shown by Liu et al. [30]. Of interest, here is that data are presented at the compute elements by some overall control logic, in contrast to relying on the performance of a fixed cache hierarchy.

A further benefit is that the degree of parallelism can be controlled. For NFS-MVE, only four outputs are computed in parallel ($ItLP_{fpga} = 4$). Despite this, a large speedup over the GPU is observed. This is due to a lower instantaneous on-chip memory buffer requirement. The off-chip reads are minimized by reducing the amount of input frame data that is required, and may later be reused, at any one time. In contrast, the GPU has a larger iteration level parallelism in the number of fragment pipelines. If one considers the number of concurrently live threads (estimated experimentally to be 1,200) parallelism is even greater. A GPU has too much parallelism for an efficient direct implementation of NFS-MVE to be achievable for the given cache size.

A heuristically controlled memory system and a choice of the degree of iteration level parallelism are inherent benefits of the FPGA over the GPU under data dependence constraints.

5.3.3 Data Dependencies in the GeForce 8/9

An impressive performance improvement for histogram generation on GeForce 8 GPUs is cited in [31]. The advantageous improvements are cited as the shared memory (between "warps" of 32 threads) and lock-step thread execution which enables synchronization of threads. The peak throughput is quoted as 5.5 GB/s. It is clear that for algorithms with the localized but data-dependent memory access characteristics of histogram equalization the newer GPUs provides substantial performance improvements.

TABLE 14
Resource Usage for Case Study Algorithms on a Virtex-4 FPGA Device (Percentage Utilization of the XC4VSX25 Device)

Case Study	CLBs	XtremeDSP Blocks	Block Select RAMs
PCCR	905 (33.7%)	43 (89.6%)	0 (0%)
Conv (5×5)	472 (17.6%)	75[1] (156.3%)	8 (11.1%)
Conv (9×9)	1444 (53.7%)	243[1] (506.3%)	16 (22.2%)
Conv (11×11)	2446 (91.0%)	363[1] (756.3%)	20 (27.8%)
Resizing	671 (24.9%)	36 (75.0%)	24 (33.3%)
HEqu	3415 (127.0%)	0 (0%)	0 (0%)
NFS-MVE [4]	- (< 100%)	0 (0%)	- (< 100%)

[1] Maximum possible utilisation figures quoted

5.4 Limitations of the Performance Comparison

5.4.1 FPGA Resource Usage

If one considers the alternative use of each device in a system-on-chip architecture, then it may be desirable to compare target devices of equivalent die area.

It is assumed that FPGA manufacturers fabricate their largest device on the maximum feasible die area (1.5 inch $\times$ 1.5 inch) and one may scale die size approximately with CLB (the fundamental logic unit in Xilinx FPGAs) count [32]. For the GPU, the international technology road map for semiconductors states that GPUs are of size 220 mm^2 [33]. The largest FPGA device is of the order of 6.5 times the size of a GPU.

Subsequently, the Virtex-4 XC4VSX25, which has approximately 12 percent of the CLBs of the XC4VLX200 device, is assumed to have an equivalent die area to a GeForce 7900 GTX. Case study resource utilization of this device is shown in Table 14.

The resizing and primary color correction algorithm resource utilization requirements are met by the XC4VSX25 device. In [4], NFS-MVE is successfully targeted at the XC4VSX25 device. For HEqu, the device is over-mapped by 27 percent in slice logic. However, the utilization may be significantly reduced by implementing a proportion of the 256 accumulators and the LUT decoder within the XtremeDSP slices and BRAM.

2D convolution is over-mapped in XtremeDSP slices. For this case study, the addition tree is implemented in slice logic; therefore, XtremeDSP slices represent only multiplier logic. Optimizations to reduce the number of multipliers, or transfer functionality to BRAMs, can reduce this requirement significantly [25], [34].

With the exception of 2D convolution, device resource constraints can be met for all algorithms for an FPGA device of equal size to the GPU. This shows that per unit die area, the GPU has superior performance, of up to three times, for all but the NFS-MVE and HEqu case studies (see Fig. 1). These are two examples where a specialized data path in an FPGA implementation can outperform a GPU per unit die area.

Kuon and Rose identify an average 40 times die area advantage for an ASIC over an FPGA [35]. This factor reduces to 21 times when considering coarse-grained FPGA resources [35]. The difference observed in this work is lower for two reasons. First, the GPU contains a large portion of redundant logic; only 50 percent of a GPU is dedicated to the shader pipeline [36]. Second, FPGA implementations use fixed-point-number representation.

5.4.2 Numerical Precision

The numerical precision adopted for FPGA in this work is fixed-point representation. This exploits and demonstrates an inherent advantage of the FPGA where accuracy is traded for speedup. A brief theoretical observation is made below regarding floating point format on the FPGA.

Consider floating point precision for the FPGA implementation of the arithmetic intensive PCCR algorithm. This is an equivalent numerical precision to the GPU (also floating point).

PCCR requires 187 FPGA numerical operation blocks as specified in Section 5.1. A moderate 15 percent CLB and 8 percent XtremeDSP utilization of the largest Virtex-4 (XCVSX512) device is required to implement this algorithm in fixed-point 18-bit precision. The required operations are 43 multipliers plus 144 other operations including mainly addition, comparators and two CORDIC blocks. A Xilinx Coregen single-precision Virtex-4 floating point multiplier, requires five XtremeDSP blocks and 48 CLBs. An adder, requires four DSPs and 86 CLBs. If one counts DSP and CLB utilization in multipliers and adders alone, the count is 695 DSPs and 12,384 CLBs. This exceeds the resources of the XCVSX512 in CLBs (201 percent) and XtremeDSP slices (136 percent). This is not counting additional resources to route the block and to perform rotation (previously a CORDIC block) and comparators.

A two to three times improvement in device density is required to support floating point operations for the primary color correction case study on even the largest current device. This highlights that the computational density factor for a GPU over an FPGA in a like-for-like numerical precision comparison is in excess of 12 times. This is due to a compound six times larger die area size and an over-mapping of resources of at least two times.

The remainder of the case studies, with the exception of HEqu, may for a given scenario also require floating point precision.

5.4.3 Implementation Effort

Design time is currently omitted. Qualitative low to high perceived effort is discussed below.

PCCR is low effort on a GPU and medium on an FPGA, due to implementing low level functional blocks. A C to gates high-level language may improve FPGA effort. Convolution is low effort on each device. Resizing is low effort on a GPU and medium-high on a FPGA, where the reuse heuristic and multiple clock domains are required. HEqu is low effort on FPGA. Both HEqu and MVE are high effort on a GPU, due to optimization requirements. For the FPGA, the effort is low-medium for MVE because the Sonic-on-chip system-level design architecture is used [4].

Interestingly, a low effort correlates with higher performance. This is intuitive in that in this case the algorithm maps well to the architecture and is thus easier to implement.

6 Result Summary: Throughput Rate Drivers

To collate the results and analysis in this paper, the throughput rate drivers for the case study in Section 5 are summarized in Section 6.1. A GeForce 8 GPU case study, with comparison to the results in Section 6.1, is discussed in Section 6.2.

TABLE 15
Clock Speed in Mega-Hertz (MHz) for FPGA Implementations of Case Study Implementations on Three Target Devices

	Virtex-4	Virtex-II Pro	Spartan-3
PCCR	296	200	139
Conv (5×5)	347	227	122
Conv (9×9)	326	184	80
Conv (11×11)	321	199	25
Resizing	206	148	115
HEqu	172	116	105
NFS-MVE	143	100	91
Min/Max Ratio to GeForce 7900 GTX	1.87/4.55	2.86/6.5	4.68/26
Min/Max Ratio to GeForce 6800 GT	1.01/2.45	1.54/3.5	2.52/14

TABLE 16
A Speedup ($Speedup_{fpga \rightarrow gpu}$) Comparison between the GeForce 7900 GTX GPU and the Virtex-4 FPGA Device

Algorithm	CPO_{gpu}/CPO_{fpga}	$ItLP_{fpga}/ ItLP_{gpu}$	Speedup
PCCR	44.70/1	1/24	1.86
Conv 5×5	36.90/1	1/24	1.54
Conv 9×9	143.88/1	1/24	5.99
Conv 11×11	230.21/1	1/24	9.59
Interp (avg)	19.59/1	1/24	0.82
Deci (avg)	41.86/3.75	1/24	0.51
HEqu	100.45/1	256/24,1/24	4.19
NFS-MVE	48533.33/1664	4/24	4.86

6.1 The GeForce 7800 GT and Virtex-4 Case Study

The range of observed values for the throughput rate drivers for the case study presented in Section 5 are summarized here.

A summary of the achieved clock rates for three FPGA devices is shown in Table 15. The case study implementations reported here are not optimized for clock speed performance; however, sensible design decisions are made with respect to pipelining. The ratio to the clock rate of two sample GPUs is included.

Comparing the Virtex-4 and GeForce 7900 GTX, up to 4.55 times speedup of the GPU over an FPGA is observed. This occurs on the NFS-MVE case study. Evidently, a higher clock rate does not always result in a higher throughput performance.

A large difference in clock speed is observed for convolution size 11×11 on the Spartan-3 device. This occurs because the slice usage, on the largest device, is 99 percent of the total resources.

For the remainder of the case study algorithms, more moderate clock speed differences are observed between GPUs and FPGAs.

If the Spartan-3 device is neglected, the range of clock cycle differences is a factor of 1.01-6.5 times higher for the GPU with respect to FPGA designs. An average of 2.86 times clock speed benefit for the GPUs over FPGAs is observed. This difference is comparable to the ratio observed by Kuon and Rose for 90 nm dedicated ASICs over FPGAs of up to five times [35]. A GPU may be considered as a dedicated ASIC for computer graphics.

In summary, the GPU has an intrinsic clock cycle performance benefit of up to 4.55 times for our case study (V4 and GF7).

If the clock speed difference is removed, the speedup relationship between the FPGA and the GPU is as summarized in Table 16. This shows the inherent architectural differences.

Despite a high degree of iteration level parallelism on the GPU ($ItLP_{gpu} = 24$) and a vectorized ISA, a speedup of two times, in favor of the FPGA, is observed for PCCR. The key is the CPO advantage from arbitrary FPGA operation level parallelism.

For Conv, "speedup" is approximately $0.07n^2$ for a window size of $n \times n$. An FPGA presents a speedup when n is greater than 3.77. The CPO advantage rises sharply due to the fixed memory access overhead in a GPU. An FPGA facilitates a specialized data path for data reuse and arbitrary operation level parallelization.

A GPU is superior for resizing. For interpolation, the speedup is 1.2 times and for decimation the factor is two times in favor of the GPU. The input and output bandwidth requirements for the FPGA account for this discrepancy, as explained in Section 5.2.

For histogram generation (the key part of HEqu), the iteration level parallelism of the FPGA implementation is 256. The FPGA also has a large benefit in clock cycles per output of 100 times.

For NFS-MVE, the speedup factor for the FPGA is 4.86 times. The FPGA implementation of NFS-MVE is based on the Sonic-on-Chip platform, in which four processing element cores calculate motion vectors in parallel ($ItLP_{fpga} = 4$) [4]. Each processing core requires 1,664 clock cycles (CPO_{fpga}) to compute a single motion vector. The GPU has an iteration level parallelism advantage of six times; however, this is surpassed by the FPGA's 30 times clock cycle per output advantage.

The GPU clock cycle per output figures in Table 16 do not include the multipass overhead. If this is included, CPO increases to 153 for HEqu and 86,667 for NFS-MVE. With this included the speedup factor for FPGA increases to 6.4 and 8.7 times, respectively. The overhead is higher for NFS-MVE due to a nine pass implementation; HEqu requires only five passes.

An FPGA has a significant advantage over a GPU in CPO of up to 230 times for our case study. The advantage for a GPU is up to 24 times higher iteration level parallelism. For all case studies, with the exception of resizing, a 1.5-9.6 times speedup from using an FPGA is observed over the GPU.

6.2 Case Study: Application to a GeForce 8800 GTX

Consider applying the comparison in Section 6 to work by Che et al. [3] as shown in Table 17. Che et al. compare a Virtex-II Pro FPGA, GeForce 8800 GTX GPU, and Intel Xeon GPP for the Gaussian Elimination, Data Encryption Standard (DES), and Needleman-Wunsch algorithms. These are algorithms from 3 of the 13 Berkeley dwarves for successful architectures.

In Che et al.'s setup, the GPU core clock rate is 575 MHz (1,350 MHz processor performance) and FPGA clock rate is 100 MHz. The GPU has an iteration level parallelism of 128 cores and no iteration parallelism is employed on the FPGA. Therefore, the GPU has a fixed clock cycle benefit of 13.5 times and iteration level parallelism benefit of 128 times. The total is a 1,728 times intrinsic benefit for the GPU. For newer FPGA devices, more aligned to the generation of GPU, timing closure at faster speeds is expected. Also Che et al. do

TABLE 17
Throughput Rate Drivers for a Comparison of GeForce 8 and Virtex-II Pro (Quoting Clock Cycles per Iteration) [3]

Algorithm	GPU	FPGA	GPP
DES	1.7×10^8	83	–
Gaussian Elimination	2.2×10^8	2.6×10^5	3.2×10^6
Needleman-Wunsch	9.0×10^7	2.0×10^4	3.0×10^5

not use the full FPGA resource so these factors should be considered when reading this analysis.

For Gaussian Elimination in Table 17, the input size is 64×64. Gaussian Elimination requires strided memory accesses to a matrix which is poorly suited to a GPU memory system. This is the reason for the three orders of magnitude clock cycle deficit with respect to the FPGA. The poor mapping is also emphasized by a two order of higher clock cycle per output requirement than the GPP. A further demonstration of the inefficiency is that as the input size is increased, the FPGA clock cycle benefit is consistent. The clock cycle benefit for the FPGA is almost eliminated if the GPU intrinsic benefic of 1,728 is included.

DES requires bitwise operations and is thus well suited to FPGA implementation. For a 64-bit block only 83 clock cycles are required for computation on the FPGA which is much smaller than the GPU. An FPGA has seven orders of magnitude clock cycle benefit over a GPU in this case. This reduces to approximately four orders of magnitude when clock speed is considered. It is expected that GPU performance will improve for larger input sample sizes, where the GPU's iteration level parallelism can be exploited [3].

For Needleman-Wunsch, the FPGA also has the lowest clock cycle count in Table 17 for an input size of 64×64. As input size is increased, the GPU and FPGA clock cycle difference narrows to two orders of magnitude. In this case, the GPU outperforms the FPGA because of a 1,728 times intrinsic benefit.

Che et al.'s work shows, albeit for a different application domain, that for sample algorithms from 3 of the 13 Berkeley dwarves for successful systems an FPGA remains beneficial in clock cycles per output over a GPU. In each case, if the iteration level parallelism is included both the FPGA and GPU provide a speedup of at least five times over the GPP.

From the GeForce 7 to 9 series of GPUs there is an increase in processor clock rate, by 2.5 times, and iteration level parallelism, by 5 times. It is interesting to note that the core graphics core clock rate actually drops from 650 MHz in the GeForce 7 to 600 MHz in the GeForce 9. This is demonstrative of the current trend to strive for parallelism over increase core performance.

For the results in Table 15, a GeForce 9 would have up to a 12 times benefit in clock cycle performance over the FPGA. Also, in Table 16 the iteration level parallelism benefit of the GeForce 9 (with 128 processor cores) over the Virtex-4 is from 128 to a disadvantage of 0.5 times. The clock cycles per output is a more difficult metric to approximate as was reasoned in Sections 5.1.3, 5.2.5, and 5.3.3.

7 Summary

Performance drivers for the graphics processor and reconfigurable logic, which relate to algorithm characteristics, have been presented using "off-the-shelf" devices. Three algorithm characterization features have been proposed and shown throughout, in particular in Section 7.2, to represent the performance challenges of each architecture. This analysis is possible due to the quantified performance comparison proposed in Section 3.2.

To conclude the findings, the following summaries are included. A projection of the scalability of the architectures in Section 7.1. In Section 7.2, the questions from Section 4 are addressed. Throughput rate drivers are summarized in Section 7.3. In Section 7.4, future work is proposed.

7.1 Future Scalability Implications

The progression of Moore's Law will inevitably provide increasing parallelism for both GPU and FPGA devices. A number of interesting observations from the work in this paper are made below.

For an FPGA, a key benefit is the choice of the degree of parallelism to implement. In regards to Ahmdahl's Law [37] in (14) this is an arbitrary increase in parallelism (n) within resource constraints, where r_p and r_s are parallel and serial algorithm portions, respectively. The limitation in achieving speedup is the design time and complexity in exploiting the available resources. Also serial algorithm portions (r_s) can be supported to limit host processor intervention.

$$Speedup = \frac{1}{r_s + \frac{r_p}{n}}. \tag{15}$$

The key difference for the GPU is that parallelism is fixed in operation and iteration dimensions. The density advantage of the fixed architecture results in a higher value of n for equivalent die area to an FPGA. The GPU data path and memory system limit acceleration capabilities. This is exemplified with the HEqu and NFS-MVE, where increased serialization (multiple passes) must be introduced to achieve a high GPU throughput performance. This fundamentally limits acceleration potential to be less than an FPGA as n tends to infinity.

Newer multiprocessor architectures, for example, the Cell and GeForce 8800 GTX, support increased on-chip memory control. This alleviates some memory system restrictions incidental to the GeForce 7 GPUs [13], as discussed in part in Section 5.2.5. However, for each device, a fixed degree of operation and iteration level parallelism must be exploited. It is in this capacity that the FPGA is inherently beneficial over all stated multiprocessors.

In raw compute power, the GPU is observed in Section 5.4 to be advantageous when compared with a fixed-point-FPGA implementation. For a floating point FPGA implementation the GPU is superior by a further 12 times. The potential value of n on a GPU is evidently higher. An FPGA is superior for algorithm characteristics of large degrees of data dependence and number of memory accesses.

The GPU memory system, degree of multithreading, and iteration parallelism are key architectural features which result in both impressive algorithm acceleration and limitations in the presence of large amounts of data dependence. However, GPUs have been reported to be less power efficient than FPGAs [18].

7.2 Answers to the Questions from Section 4

For compute bound algorithms, the exploitation of large degrees of parallelism at an iteration and operation level provides impressive speedup for both the GPU and FPGA over a GPP. A GPU is benefited from a low clock cycle per

computation instruction which is an order of magnitude lower than that of a GPP *(Question 1)*.

The memory bound 2D Convolution algorithm is used to show the fixed clock cycle cost of a GPU. For the GeForce 7900 GTX the effective input bandwith is 36 GBytes/sec for this case. The fixed clock cycle cost of this device results in inferior performance to the FPGA beyond a window size of 4×4 pixels *(Question 2)*.

The GPU is highly sensitive to the amount of data reuse which can be exploited. This results in up to two times drop in input bandwidth for decimation. FPGA performance is limited, for decimation, by the discrepancy between input and output bandwidths. The presence of a variable data reuse requirement for resizing increases the data-path delay for the FPGA implementation. A GPU is superior for all resizing options *(Question 3)*.

The effect of data dependence has been analyzed. A large number of required passes for NFS-MVE, nine in contrast to five for HEqu, results in up to two times performance degradation from pipeline setup costs alone. In trade-off on-chip memory access performance can be increased at a cost of extra rendering passes. Ultimately, the nature of the memory access requirements, as for decimation, limits GPU performance *(Question 4)*.

7.3 Throughput Rate Drivers

The difference in device *clock speed* amounts to a peak advantage of 4.55 times, and a minimum of 1.86 times, in favor of the GPU. This represents an intrinsic benefit of the fixed GPU architecture with respect to the interconnect flexibility overhead of reconfigurable logic fabric.

In *cycles per output*, the FPGA implementations present an advantage of up to 230 times, with a minimum of a factor of 11.2 times, over GPU implementations. The innate spacial processing nature of hardware implementations, over the temporal nature of software, is the key contributor to this benefit.

For a GPU, the degree of *iteration level parallelism* is fixed. The GeForce 7900 GTX is observed to provide a benefit of up to 24 times over FPGA implementations. A key factor is evidently the design decision of single pipeline FPGA implementations.

7.4 Future Work

This includes the exploration of alternative architectures; for example, the GeForce 8 series of GPUs and Altera Stratix-IV FPGAs. It would be interesting to discover by how much the limited control of the on-chip memory on the GeForce 8 can improve performance for the NFS-MVE case study in particular. The application of the characterization scheme to commonly used benchmark sets and new application domains will also be explored. A perhaps grander vision of the work is as a theoretical basis for discerning, from a given "new algorithm," the most suitable target architecture or properties thereof. The envisioned process is to first characterize the application using the proposed three algorithm features. From a prior separation of the space—into regions most suited to different architectural features—a suitable architecture is determined.

Acknowledgments

The authors thank Jay Cornwall from the Department of Computing, Imperial College London, and Simon Haynes and Sarah Witt from Sony Broadcast & Professional Europe for their invaluable technical support and guidance. The authors gratefully acknowledge support from Sony Broadcast & Professional Europe, the Donal Morphy Scholarship, and the UK Engineering and Physical Sciences Research Council (under Grant EP/C549481/1).

References

[1] B. Cope, P.Y.K. Cheung, W. Luk, and S. Witt, "Have GPUs Made FPGAs Redundant in the Field of Video Processing?" *Proc. IEEE Int'l Conf. Field-Programmable Technology,* pp. 111-118, Dec. 2005.

[2] B. Cope, P.Y.K. Cheung, and W. Luk, "Bridging the Gap between FPGAs and Multi-Processor Architectures: A Video Processing Perspective," *Proc. Application-Specific Systems, Architectures and Processors,* pp. 308-313, July 2007.

[3] S. Che, J. Li, J.W. Sheaffer, K. Skadron, and J. Lach, "Accelerating Compute Intensive Applications with GPUs and FPGAs," *Proc. Symp. Application Specific Processors,* pp. 101-107, 2008.

[4] N.P. Sedcole, "Reconfigurable Platform-Based Design in FPGAs for Video Image Processing," PhD dissertation, Imperial College, Univ. of London, Jan. 2006.

[5] Z. Guo, W. Najjar, F. Vahid, and K. Vissers, "A Quantitative Analysis of the Speedup Factors of FPGAs over Processors," *Proc. 2004 ACM/SIGDA 12th Int'l Symp. Field Programmable Gate Arrays (FPGA),* pp. 162-170, Feb. 2004.

[6] X. Xue, A. Cheryauka, and D. Tubbs, "Acceleration of Fluoro-CT Reconstruction for a Mobile C-Arm on GPU and FPGA Hardware: A Simulation Study," *Proc. SPIE, Medical Imaging 2006: Physics of Medical Imaging,* pp. 1494-1501, 2006.

[7] K. Mueller, F. Xu, and N. Neophytou, "Why do Commodity Graphics Boards (GPUs) Work so Well for Acceleration of Computed Tomography?" *Proc. SPIE Electronic Imaging (Keynote),* 2007.

[8] L. Howes, O. Beckmann, O. Mencer, O. Pell, and P. Price, "Comparing FPGAs to Graphics Accelerators and the Playstation 2 Using a Unified Source Description," *Proc. Int'l Conf. Field-Programmable Logic,* pp. 119-124, Aug. 2006.

[9] Z.K. Baker, M.B. Gokhale, and J.L. Tripp, "Matched Filter Computation on FPGA, Cell and GPU," *Proc. Int'l Symp. Field-Programmable Custom Computing Machines,* pp. 207-216, Apr. 2007.

[10] G. Morris and M. Aubury, "Design Space Exploration of the European Option Benchmark Using Hyperstreams," *Proc. Field-Programmable Logic,* pp. 5-10, Aug. 2007.

[11] E. Kelmelis, J. Humphrey, J. Durbano, and F. Ortiz, "High-Performance Computing with Desktop Workstations," *WSEAS Trans. Math.,* Jan. 2007.

[12] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, and T.J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Proc. Eurographics 2005, State of the Art Reports,* pp. 21-51, Aug. 2005.

[13] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, and J.C. Phillips, "GPU Computing," *Proc. IEEE,* vol. 96, no. 5, pp. 879-899, May 2008.

[14] N.K. Govindaraju, S. Larsen, J. Gray, and D. Manocha, "A Memory Model for Scientific Algorithms on Graphics Processors," *Proc. ACM/IEEE Super Computing,* pp. 89-98, 2006.

[15] Sundance "Vender Published Specification of IP Core," http://www.sundance.com/docs/FC100_user_manual.pdf, 2007.

[16] M.E. Angelopoulou, K. Masselos, P.Y.K. Cheung, and Y. Andreopoulos, "Implementation and Comparison of the 5/3 Lifting 2D Discrete Wavelet Transform Computation Schedules on FPGAs," *J. VLSI Signal Processing,* pp. 3-21, 2007.

[17] T.T. Wong, C.S. Leung, P.A. Heng, and J. Wang, "Discrete Wavelet Transform on Consumer-Level Graphics Hardware," *IEEE Trans. Multimedia,* vol. 9, no. 3, pp. 668-673, Apr. 2007.

[18] Q. Jin, D. Thomas, W. Luk, and B. Cope, "Exploring Reconfigurable Architectures for Binomial-Tree Pricing Models," *ACM Trans. Reconfigurable Technology and Systems,* vol. 4943, pp. 245-255, 2008.

[19] Nvidia, "Published Technology Size and Release Dates on Company Web Page," http://www.nvidia.com, 2007.

[20] M. Pharr and R. Fernando, *GPU Gems 2.* Addison Wesley, 2005.

[21] R.C. Gonzalez and R.E. Woods, *Digital Image Processing,* second ed. Prentice Hall, 2002.

[22] Y.Q. Shi and H. Sun, *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms and Standards.* CRC Press, 1999.
[23] Sony Broadcast and Professional Europe, "Sample High Definition Video Files," Provided in 720p and 1080i Frame Formats, 2005.
[24] D. Manchoa, "General Purpose Computations Using Graphics Processors," *Entertainment Computing,* vol. 38, no. 8, pp. 85-87, 2005.
[25] C.-S. Bouganis, G.A. Constantinides, and P.Y.K. Cheung, "A Novel 2D Filter Design Methodology for Heterogeneous Devices," *Proc. 13th Ann. IEEE Symp. Field-Programmable Custom Computing Machines,* pp. 13-22, Apr. 2005.
[26] R.P. Tidwell, "Alpha Blending Two Data Streams Using a DSP48 DDR Technique," Xilinx Application Note: XAPP706 (v1.0), 2005.
[27] S. Perry, "Design Patterns for Programmable Computing Platforms," Altera: Technical Presentation, 2005.
[28] V. Podlozhnyuk, "Image Convolution with Cuda," http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/convolutionSeparable/doc/convolutionSeparable.pdf, 2008.
[29] B. Cope, P.Y.K. Cheung, and W. Luk, "Using Reconfigurable Logic to Optimise GPU Memory Accesses," *Proc. ACM/SIGDA Design, Automation and Test in Europe,* pp. 44-49, Mar. 2008.
[30] Q. Liu, K. Masselos, and G.A. Constantinides, "Data Reuse Exploration for FPGA Based Platforms Applied to the Full Search Motion Estimation Algorithm," *Proc. 16th IEEE Int'l Conf. Field Programmable Logic and Applications,* Aug. 2006.
[31] V. Podlozhnyuk, "Histogram Calculation in Cuda," http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/histogram256/doc/histogram.pdf, 2008.
[32] N. Campregher, P.Y.K. Cheung, G.A. Constantinides, and M. Vasilko, "Analysis of Yield Loss Due to Random Photolithographic Defects in the Interconnect Structure of FPGAs," *Proc. 2005 ACM/SIGDA 13th Int'l Symp. Field-Programmable Gate Array,* pp. 138-148, Feb. 2005.
[33] ITRS Working Group, "ITRS Report on System Drivers," http://www.itrs.net/reports.html, 2005-2006.
[34] G.W. Morris, G.A. Constantinides, and P.Y.K. Cheung, "Migrating Functionality from ROMs to Embedded Multipliers," *Proc. 12th IEEE Symp. Field-Programmable Custom Computing Machines,* pp. 287-288, Apr. 2004.
[35] I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs," *Proc. 2006 ACM/SIGDA Int'l Symp. Field-Programmable Gate Arrays,* pp. 21-30, Feb. 2006.
[36] M. Harris, "Information on the GeForce 7800 GTX Graphics Processor," Personal Correspondence with NVIDA Expert, 2006.
[37] G.M. Amdahl, "Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities," *Proc. Am. Federation of Information Processing Soc. (AFIPS) Conf.,* vol. 30, pp. 483-485, 1967.

Ben Cope received the MEng and PhD degrees in electronic and electrical engineering from Imperial College London, UK. His research interests are video processing and reconfigurable computing using FPGAs and GPUs. His work includes performance comparison, architecture modeling, and investigating the cooperative use of reconfigurable logic and GPUs for video processing acceleration. He now works as a senior engineer at the Altera European Technology Centre working on wireless systems solutions. He is a member of the IEEE.

Peter Y.K. Cheung (M'85-SM'04) received the BS degree from Imperial College of Science and Technology, UK, in 1973. Since 1980, he has been with the Department of Electrical Electronic Engineering, Imperial College, where he is a professor of digital systems. Before this, he was with Hewlett Packard, Scotland. His research interests include VLSI architectures for signal processing, asynchronous systems, reconfigurable computing using FPGAs, and architectural synthesis. He was elected as one of the first Imperial College Teaching Fellows in 1994 in recognition of his innovation in teaching. He is a senior member of the IEEE.

Wayne Luk (F'09) received the MA, MSc, and DPhil degrees in engineering and computing science from the University of Oxford, UK. He is a professor of computer engineering with Imperial College London and was a visiting professor with Stanford University, California, and with Queen's University Belfast, UK. His research includes theory and practice of customizing hardware and software for specific application domains, such as multimedia, financial modeling, and medical computing. His current work involves high-level compilation techniques and tools for high-performance computers and embedded systems, particularly those containing accelerators such as FPGAs and GPUs. He is a fellow of the IEEE.

Lee Howes received the MEng degree in computing from Imperial College London in 2005 and is working toward the PhD degree also at Imperial College London. His research concentrates on the use of metadata annotations to optimize compilation of high-performance software. He currently works on OpenCL and related technology at AMD. He is a member of the IEEE.