A Systematic Design Space Exploration Approach to Customising Multi-Processor Architectures: Exemplified using Graphics Processors

Ben Cope[1], Peter Y.K. Cheung[1], Wayne Luk[2] and Lee Howes[2]

[1]Department of Electrical & Electronic Engineering, Imperial College London, UK
[2]Department of Computing, Imperial College London, UK

Abstract. A systematic approach to customising Homogeneous Multi-Processor (HoMP) architectures is described. The approach involves a novel design space exploration tool and a parameterisable system model. Post-fabrication customisation options for using reconfigurable logic with a HoMP are classified. The adoption of the approach in exploring pre- and post-fabrication customisation options to optimise an architecture's critical paths is then described. The approach and steps are demonstrated using the architecture of a graphics processor. We also analyse on-chip and off-chip memory access for systems with one or more processing elements (PEs), and study the impact of the number of threads per PE on the amount of off-chip memory access and the number of cycles for each output. It is shown that post-fabrication customisation of a graphics processor can provide up to four times performance improvement for negligible area cost.

1 Introduction

The graphics processor architecture is used to demonstrate a systematic approach to exploring the customisation of Homogeneous Multi-Processor (HoMP) architectures for specific application domains. Our approach involves a novel design space exploration tool with a parameterisable system model.

As motivation for the exploration tool presented here, consider the following projections from the Tera Device [1] and HiPEAC [2] road maps:

I. Memory bandwidth and processing element (PE) interconnect restrictions necessitate a revolutionary change in on-chip memory systems [1, 2].
II. It is becoming increasingly important to automate the generation of customisable accelerator architectures from a set of high level descriptors [1, 2].
III. Continuing from II, architecture customisation may be applied at the design, fabrication, computation or runtime stage [1].

The statements above are not mutually exclusive: an answer to statement I may be a customisation from statement II. It is important to note the following key words.

First *customisation*, which represents pre-fabrication (pre-fab) and post-fabrication (post-fab) architectural customisations. Pre-fab customisation is the familiar approach

to determine 'fixed' architecture components. Post-fab customisation is a choice of reconfigurable logic (RL) architectural components (hardware), or a programmable instruction processor (software), to facilitate in-field modifications. Although we refer to RL customisations, the work in this paper is also applicable to instruction processors.

Second, *high-level descriptors*. The increased complexity of the application and architecture domains, necessitate architectural exploration at a suitably high degree of abstraction, with a high-level representation of each domain.

Third, a focus on *interconnect* and *memory systems*. These factors frequent each road map [1, 2]. It is becoming increasingly challenging to present the required input data to, and distribute output data from, processing elements.

It is hoped that the exploration tool presented in this paper can be used to explore the above observations. The aim is to provide significant insight into some of the associated challenges, faced when designing the exploration process and system model.

The example taken in and focus of this work, is to target the graphics processor architecture at the video processing application domain. The approach and model are sufficient to be extended to other architectures and application domains.

The contributions of this work, and the related sections, are as follows:

1. Definition of a classification scheme for the options for post-fab customisation of a HoMP using RL. The scheme is demonstrated by analysing prior art (Section 3).
2. A systematic design space methodology to explore the customisation options for a HoMP. The key feature is the notion of pre- and post-fab options (Section 4).
3. The design space options for a HoMP are presented. A system model is described which implements these options (Section 5).
4. An analysis of the effect of processing pattern on the performance of a model with a single processing element (PE) and a single execution thread (Section 6).
5. Extension of the above single PE analysis, in contribution 4, to a multiple PE and multi-threaded example (Section 7).
6. Case studies including decimation and 2D convolution are used to explore the architectural trends of graphics processors (Section 8).
7. Proposal and exploration of a graphics processor post-fab customisation motivated by results from contributions 4 through 6 (Section 9).

In addition to the above, Section 2 discusses related work; Section 10 considers the impact of our work on other HoMPs; and Section 11 summarises our findings.

This paper is an extended version of [3]. The additional novel contributions to [3] are 1, 4 and 5. There are also technical enhancements to other contribution areas.

2 Related Work

A popular design space exploration approach is the Y-Chart [4]. The Y-Chart combines architecture and application models in 'mapper' and 'simulator' stages to produce performance predictions. In turn, these predictions motivate application and architecture model modifications. A key strength of the Y-Chart is an iterative update of application and architectural choice based on a model of system performance [4, 5].

For this work, a standard Y-Chart approach is insufficient. Two issues are as follows. First, the Y-Chart is too high-level to provide a useful insight into the exploration process. For a constrained design space, a more detailed description is preferable, as is shown in Section 5. Second, the choices of architecture features which support the mapping of application to architecture should be made more explicit than in the Y-Chart. To overcome the second issues, a third design space variable of physical mapping is introduced. This is an overlap of the application and architecture design space, and is the architectural design decisions which support the programming model. For HoMP architectures, the programming model is one of the most challenging parts of the design process. Figure 2(a) is observed to be a suitable adaptation of the Y-Chart approach.

When creating a model for design space exploration one is presented with a tradeoff between a higher level of abstraction, to broaden the design space, and low level architectural detail, to make the results meaningful. Related work on architecture models, Kahn Process Networks and the SystemC library are discussed below.

The following model the low level aspects of the graphics processor architecture.

Moya [6] created a cycle-accurate model of a graphics processor named ATTILA. Long simulation times prohibit its use for broad design space exploration. Also, the fine detail of ATTILA limits its scope to prototyping minor architecture modifications.

QSilver [7] is another fine-grained graphics processor architectural model. One application is to explore thermal management. QSliver is, similarly to [6], too low-level for rapid and straight forward design space exploration.

nVidia provide a graphics shader performance model named nvshaderperf [8]. This is an accurate profile of the computational performance of kernel functions, but provides no information on memory system performance. In the system model in Section 5, nvshaderperf is used to estimate computational cycle count for processing elements.

Govindaraju provides a useful estimate of graphics processor memory system cache arrangement in [9]. For the nVidia GeForce 7800 GTX, Govindaraju estimates cache block size at 8×8 pixels, and cache size at 128 KBytes. The results follow estimates by Moya [6] of a 16 KByte cache with 8×8 pixel cache lines for the older GeForce 6800 GT. A small cache size is well suited to graphics rendering.

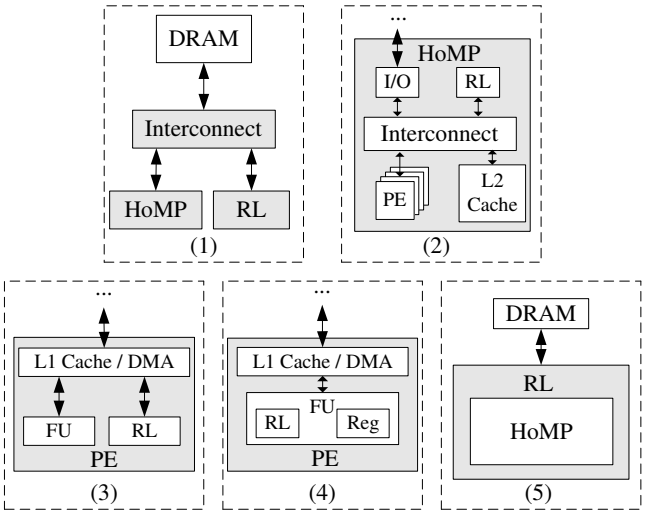
In Section 5, a model is presented which provides a tradeoff between the fine-detail in [6, 7], and high-level or feature-specific models in [8, 9]. The advantage of our model is that architectural modifications can be rapidly prototyped, through modelling [non-cycle accurate] performance trends. Memory system estimations from [6, 9] are used for model parametrisation to enable results verification in Section 8.

The interconnects between components of the system model in Figure 4 can be interpreted conceptually as a Kahn Process Network (KPN) [10]. Each processing group can be thought of as a KPN ‘Process’. The buffer which queues memory accesses between a processing element (PE) and the memory management unit (MMU) is equivalent to an unbounded KPN ‘channel’. To ensure that the appropriate latencies are simulated in the system model, flags are passed between process nodes (PEs).

The IEEE 1666-2005 SystemC class library is used to implement the abstract transaction level model (TLM) of the architecture in Section 5. Related work in [7, 11] demonstrates the SystemC class library to be a flexible platform for design space exploration. Specifically, Rissa [11] presents the advantages of SystemC over a register

transfer level (RTL) description, in VHDL or Verilog. A simulation time speedup of 360 to $10,000$ times is achieved by Rissa for SystemC models over RTL descriptions.

For a comprehensive treatment of the SystemC language and transaction level models the reader is directed to [12].



Key: I/O is the input/output interface, FU is the functional unit of a processing element (PE), RL is reconfigurable logic and Reg is a set of local register files

Fig. 1. A classification scheme for the post-fabrication customisation options for a HoMP

3 Classification of Customisation Options

In this Section the options for supporting post-fab customisation of a HoMP are classified. This demonstrates the wider applicability of the exploration tool in Section 4. The work is motivated by Todman's hardware-software co-design classification [13], for a single processor coupled in a classified manner to reconfigurable logic fabric. The advantageous factors of the customisation options in Figure 1, are as follows.

In contrast to a traditional HoMP architecture, for example a graphics processor, post-fab customisation enhances an architecture's flexibility. This makes it adaptable to a wider variety of application domains. For example, it is shown in [14] that a small reconfigurable logic block can improve graphics processor memory access performance by an order of magnitude for case study examples.

An advantage over a fully reconfigurable platform (for example, a field programmable gate array (FPGA) from Altera Co. or Xilinx Inc.) is that the architectural design space is bounded. Put another way, the architecture has a clearly defined datapath onto which an algorithm must be mapped. The result is a reduced design time. A fully reconfigurable platform requires the design of a specialised datapath.

The key benefits of post-fab customisation of an HoMP are a constrained application design space alongside support for specialisation to an application domain.

In the remainder of this Section the classification of customisation options is explained and prior works used to exemplify the classified options.

A summary of the qualitative level of integration and lowest level of shared memory for each option in Figure 1 is summarised in Table 1. Key features are discussed below.

	Role	Effect on Core Type	Level of Integration	'Shared' Memory
1	Off-Chip Co-Processor	Heterogeneous	Low	DRAM
2	On-Chip Co-Processor	Heterogeneous		L2 Cache
3	Local Co-Processor	Homogeneous		L1 Cache
4	Custom Instruction	Homogeneous		Registers
5	Glue Logic	Homogeneous	High	–

Table 1. A Summary of the Roles for RL within a HoMP

For classifications (1) to (4) the level of shared memory is the key identifier. As the level of integration increases, the granularity of the separation of tasks between a RL element becomes finer grained, from a co-processor to a custom instruction. For class (1), different algorithms and video frames may be computed on the RL co-processor and the HoMP. In contrast, in class (4) a single instruction from the assembly code of a processor PE may be accelerated on a RL core.

Class (5) presents an orthogonal use of RL to classes (1)–(4). Instead of performing computation on a part or whole algorithm, RL is used to optimise the architecture in such a manner as to improve HoMP performance for a given algorithm. This is termed as 'glue logic' and is an exciting new area in which the use of RL can thrive.

Prior works which exemplify the options in Figure 1 are now discussed.

The literature contains numerous works promoting multi-chip solutions to using RL (in the form of FPGAs) and a graphics processor as class (1) co-processors [15–18].

Moll [15] presents Sepia, where an FPGA is used to merge outputs from multiple graphics processors. The FPGA performs a subsection of the target 3D visualisation algorithm which makes this a class (1) use of RL.

Manzke [16] combines an FPGA and graphics processor devices on a PCI bus with a shared global memory. The goal is to produce a scalable solution of multiple boards. In Manzke's work the FPGA is master to the graphics processor. For Sepia, the graphics processor output drives the FPGA operation with prompt from a host CPU [15].

An equivalent setup to [15–18] for a Cell BE is proposed by Schleupen [19], this is also an example of a class (1) use of RL.

The work in [15–19] can alternatively be considered as a prototype for a single die solution containing a HoMP, RL and shared memory (class (2)).

Although not fully programmable, the Cell BE DMA engine exemplifies a class 3 use of RL. In a more abstract sense Sun's forthcoming SPARC-family Rock processor is another class 3 example. Although there is no separate hardware, 'scout threads', speculative clones of the primary thread, use the hardware multi-threading support to run ahead of stalls to execute address generation code and pre-fetch data into the cache [20].

Dale [21] proposes small scale reconfiguration within graphics processor functional units. This is a class (4) approach. A functional unit is substituted with a flexible arith-

metic unit (FAC) which can be alternately an adder or multiplier. A moderate 4.27% computational performance speed-up for a 0.2% area increase is achieved. Although the speed-up is small, this demonstrates the potential of the use of reconfiguration in graphics processors at the lowest denomination of the architecture.

Yalamanchili [22] presents two class (5) options. First, a self-tuning cache which matches the memory access requirements to the cache usage heuristic. Second, tuned on-chip interconnects to increase bandwidth for critical paths.

In [14], the authors propose REDA, a reconfigurable engine for data access targeted at graphics processors. REDA is embedded into the graphics processor memory system to optimise its memory access behaviour. This is a class 5 use of RL.

Coarse-grained reconfigurable architectures (CGRA), such as MathStar's Attrix FPOA device [23], are another example of a class (5) use of reconfigurable logic.

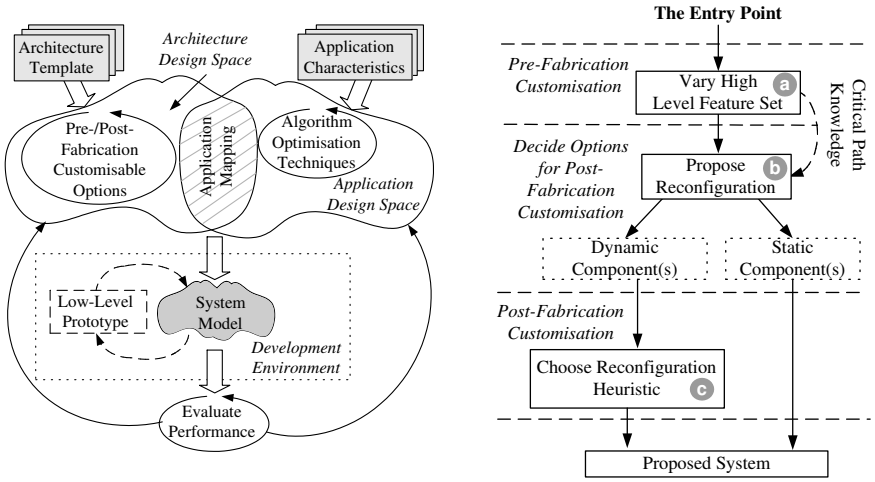
There are also a number of prior works which present equivalent solutions, to those shown above, for the case of *heterogeneous* multi-processors.

Chen et al [24] use RL as a controller in a system-on-chip solution. The RL core makes a complex system-on-chip appear as a single co-processor to an external host processor. This is class (5) glue logic.

Verbauwhede [25] presents RINGS. Three locations for RL in a network-on-chip are presented as register mapped (class (4)), memory-mapped (class (3)) and network mapped (class (2)). The terminology describes the hierarchy at which the RL core is implemented and, in similarity to Table 1, the shared memory. Verbauwhede [25] also presents a reconfigurable interconnect arbitration scheme which is a class (5) scenario.

A run-time management scheme for multi-processor systems-on-a-chip is presented by Nollet [26]. It is proposed that RL may be used to implement a flexible hardware management unit. This is also a class (5) use of RL.

It is observed that the scheme in Figure 1 is well suited to classifying a spectrum of uses of RL in HoMPs, with equivalent interpretations for heterogeneous architectures.



(a) Exploration Method (based on the 'Y-Chart' [4]) (b) Evaluating Customisation Options

Fig. 2. A Systematic Approach to Design Space Exploration

4 Design Space Exploration Approach

This section summarises the proposed approach, with is depicted in Figure 2. An overall picture of the exploration methodology is shown in Figure 2(a) and the process of evaluating customisation options in Figure 2(b). The approach is described as follows.

In Figure 2(a), the entry point to the design space exploration approach is alternative architecture templates and application characteristics. The architecture template and application model described in Section 5 initialise our design space.

The architecture design space is explored through considering pre- and post-fabrication customisation options. These can be arbitrarily chosen from the features of the template architecture. This process is explained in Figure 2(b).

An application design space is traversed by considering algorithm optimisations.

As described in Section 2, the choice of application and architecture is not mutually exclusive, and an application mapping region is defined. The addition of application mapping is particularly important for HoMPs. This is because the choice of application mapping affects the architecture choice and ultimately the programming model.

Once a set of architecture and application features have been chosen, from the respective design spaces, the options are used to parameterise the system model.

When a reconfigurable logic post-fab customisation option is proposed, the design may require a prototype to determine the area cost, maximum clock speed or power consumption. Alternatively, one may require a HoMP 'test run' to verify a proposal.

The combination of system model and low-level prototyping form the development environment. At progressively later stages in a design process, increased portions of a proposed architecture are prototyped in such a manner.

The output from the development environment is performance figures which are used to evaluate the suitability of the proposed architecture against application requirements. Example requirements in our case are to minimise clock cycle count or number of off-chip memory accesses. The process is iterated to alter application and/or architecture feature choices through educated conjecture after the performance evaluation.

The application of the approach in Figure 2(a) to the evaluation of customisation options is defined in Figure 2(b).

There are three key stages to the evaluation of customisation options. These are summarised below alongside examples of where these stages are employed.

Stage a: The exploration of pre-fab customisation options, which also defines the architecture critical paths (Sections 6, 7 and 8).

Stage b: From the analysis of critical paths post-fab customisation options are proposed (Section 9). The proposal is to supplement current blocks with RL (as chosen pre-fab).

Stage c: A heuristic for blocks supporting post-fab modifications is chosen. This determines the configuration to be applied for a particular algorithm (Section 9).

It is observed that **stage a** is typically the largest part of the exploration process and thus consumes the greatest portion of the work presented here.

A tool flow for the approach is summarised below.

As described in Section 2, SystemC is used to implement the system model. A C++ wrapper encloses the SystemC model to support modification of the two design spaces.

To enable rapid prototyping of low level modules the VHDL language is used. Open loop tests may alternatively be implemented on a graphics processor, using for example

Cg and the OpenGL API. This part of the tool set is not used in this work, however, it is used in [14] which is cited in Section 9 as an application of the exploration process.

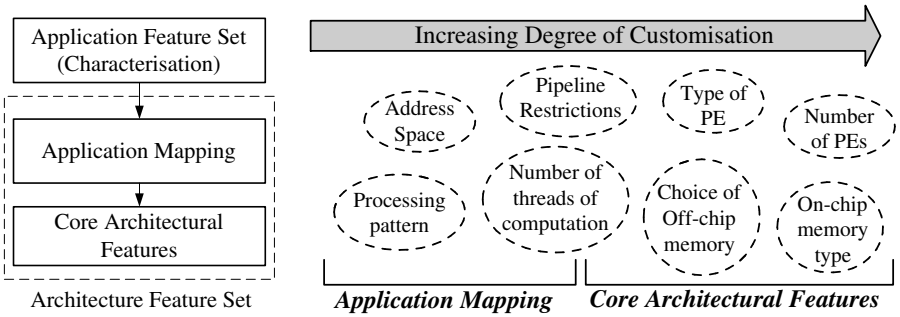
For visualisation of the performance results the Mathworks MATLAB environment is used. In addition, system trace files record the behaviour of a definable subset of signals. This setup minimises the impact on simulation time.

At present the process is fully user driven, however, it is opportunistically possible to automate **Stage a** in Figure 2(b). An example of how can be found in work by Shen on the automated generation of SystemC transaction level models [27].

5 The System Model

In this section, the design space of HoMP architectures is described alongside a model to explore the graphics processor architecture. The motivation of the model and design space is to support the methodology proposed in Section 4.

As highlighted in Section 4, the 'Y-Chart' [4] is augmented with an application mapping sub-set. The architecture design space is therefore divided into core architectural features and application mapping features, as shown in Figure 3(a). Note that application mapping is grouped with core application features to form the architecture feature set. The core architectural features represent the underlying architecture which is transferrable between different application mappings (and programming models).



(a) Feature Set Hierarchy (b) Dividing the Architectural Design Space

Fig. 3. The Architectural Design Space for the HoMP Class of Architectures

In Figure 3(b), the architecture features are presented against an increasing degree of customisation. Regions of the design space represent the customisation options. This work focuses on number of PEs, on-chip memory type (cache size), and number of threads in Section 8 and in Sections 6 and 7 processing pattern.

To explore the design space of the graphics processor for the case study of this work a high-level system model is shown in Figure 4. The astute reader will observe how, with modification, this model may be targeted at alternative HoMPs.

Figure 4(a) shows a one PE example. The pattern generation module supplies each PE with the order in which to process pixels. PE input pixel data is received from off-chip memory through the on-chip memory system (read-only in this example). When

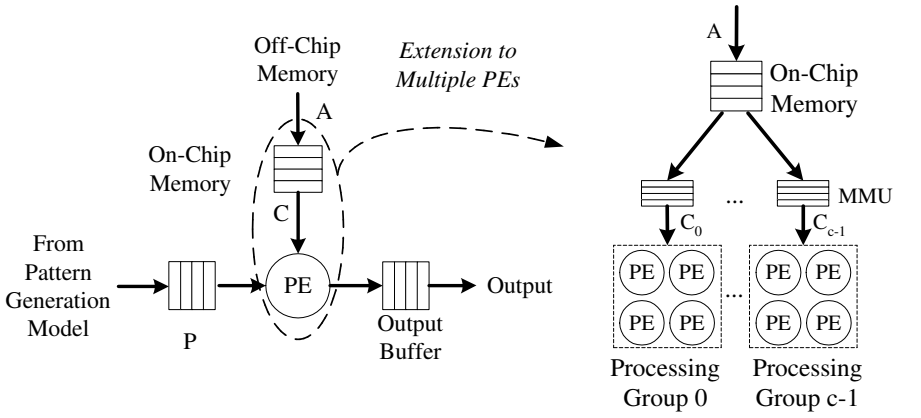
Key	Value
P	Pixel processing pattern order
(x_p, y_p)	General pixel address)
$\widehat{T}$	Thread batch size (thread level parallelism)
n	number of processing elements (PEs)
W	Pattern of accesses for each output (represented as an offset from current output pixel)
C	On-chip memory access pattern (intersection of P and W
A	Off-chip memory access pattern (C subject to cache behaviour)
n_{conv}	2D convolution kernel dimensionality (symmetric)
N_{in}	Number of input pixels per row of input frame
(s_x, s_y)	Resizing ratio for interpolation / decimation
CPO	Clock cycles per output pixel
$\|\|$	Absolute operator, used to represent size e.g. $\|C\|$ is total number of off-chip accesses

Table 2. Symbols used in Formulae

processing is complete, PE outputs are written to off-chip memory through an output buffer. Figure 4(b) shows the extension to multiple PEs. The PEs are arranged in processing groups, in this example in groups of four. A memory management unit (MMU) arbitrates memory accesses through a given choice of on-chip memory. This setup mimics a graphics processor [6, 28].

The pixel processing pattern P is an arbitrary sequence. This is a simplification of the graphics processor setup where processing order is output from a rasteriser.

A popular video memory storage format and rendering rasterisation ordering is the z-pattern [29]. In general a pixel address $\{x_p, y_p\}$ is calculated as follows. Consider that pixel iterator p is represented as a bit-vector ($p = p_{n-1}...p_2p_1p_0$), where location zero is the least significant bit. Then the x_p and y_p values are the concatenations of even ($x_p = p_{n-2}...p_4p_2p_0$) and odd ($y_p = p_{n-1}...p_5p_3p_1$) bit locations respectively.



(a) A Single Processing Element (PE)

(b) Extension to Multiple (n) Processing Elements (PEs)

MMU is an acronym for memory management unit

Fig. 4. High Level Representation of the Design Space Exploration Model

For horizontally raster scanned video an equivalent processing pattern description to that for the z-pattern is $x_p = p_{\frac{n}{2}-1}...p_0$ and $y_p = p_{n-1}...p_{\frac{n}{2}}$. A raster scan or z-pattern can be generated using an n-bit counter and bit rearrangement.

The model is implemented such that $\widehat{T}$ threads can be computed across the n PEs. For simplification $\frac{\widehat{T}}{n}$ threads are live on each PE at any instant. On a graphics processor, a thread may in general be scheduled to different PEs at different points in the computation. However, this arrangement is deemed sufficient.

A graphics processor's thread batch size can be estimated using a computationally intensive kernel to minimise the effect of cache behaviour and to emphasize steps in performance between thread batches. The chosen kernel is the American Put Option financial model [30] which requires 446 computational instructions to one memory access per kernel. Figure 5 shows the performance results for the nVidia GeForce 7900 GTX graphics processor for increasing output frame size from 1 to 10000 pixels.

It is observed that steps in time taken occur at intervals 1300 outputs. This is the predicted thread batch size ($\widehat{T}$).

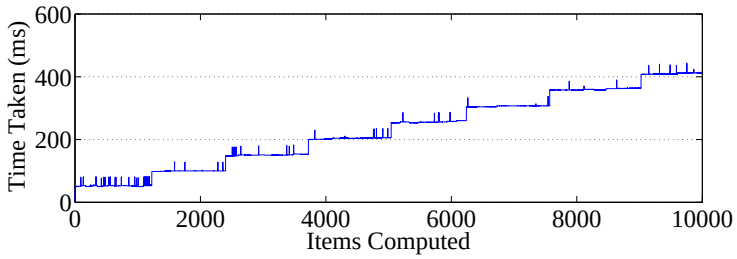


Fig. 5. Concurrent Thread Estimation for the nVidia GeForce 7900 GTX (1300 per batch)

For the nVidia GeForce 6800 GT steps are observed at intervals of 1500 pixels.

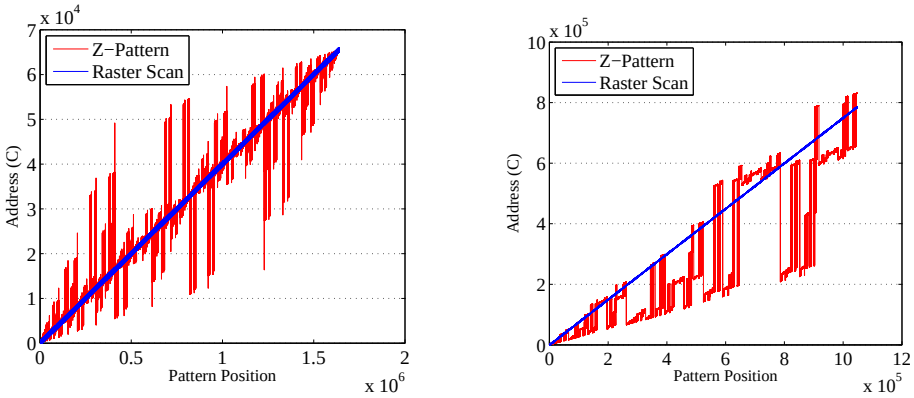
Application Model: The PE application model is a function of computation delay (taken from [8]) and memory access requirements. The pseudo-code for the memory accesses for one PE is shown in Figure 6. A set of $\frac{\widehat{T}}{n}$ output locations is input from the pattern generation block (Line 0). Memory requests occur as a set of W accesses (Line 1). Inside the outer loop requests are made for each thread i (Line 2) on Line 3. The PE then waits until all requests are satisfied (Line 4) and then iterates for the next value of w. Once all read requests are made, output pixel values are written to an output buffer (Line 5). The code iterates until the end of the processing pattern occurs. Function f is an arbitrary linear or non-linear address mapping.

0. Get $\frac{\widehat{T}}{n}$ thread addresses from P
1. For all Accesses $w = 0$ to $W - 1$
2. For all Threads $i = 0$ to $\frac{\widehat{T}}{n} - 1$
3. Request Address $f(i, w)$
4. Wait until All Read Requests Granted
5. Output $\frac{\widehat{T}}{n}$ thread results

Fig. 6. A Model of the Behaviour of a PE

6 System Model with a Single Processing Element

A system model with one PE, as shown in Figure 4(a), and one execution thread ($\widehat{T} = 1$) is considered in this Section. It is interesting to compare results for a z-pattern and horizontal raster scan processing order. For each scenario the on-chip (C) and off-chip (A) memory access pattern are shown in Figures 7 and 8 respectively.



(a) Convolution size 5×5 (b) $1080p$ to $480p$ Decimation

Fig. 7. On-chip memory access (C) performance for decimation and 2D convolution for a model with one PE and one execution thread. Output frame size is 256×256 pixels.

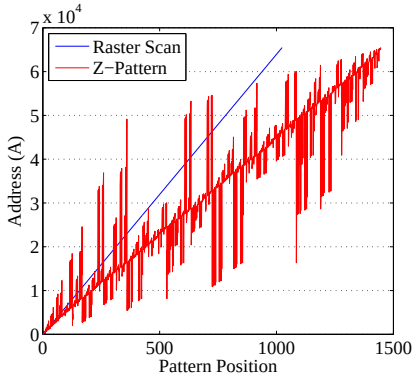
For the z-pattern processing order the variation in required on and off-chip memory addresses is significantly larger than that for raster scan processing. To quantify this difference, for on-chip reads, consider the convolution case study.

For raster scan the peak distance between reads for each output is n_{conv} rows of an image, where n_{conv} is the convolution size. In Figure 7(a) this equals $\sim 5 \times 256$ pixel locations as shown by the heavy type line. In general the maximum step size is $\sim n_{conv} N_{in}$ pixels, where N_{in} is the number of pixels per row of the input frame.

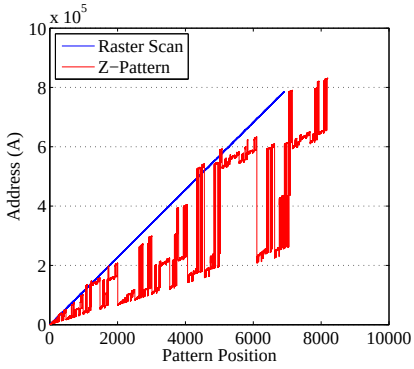
In contrast, for the z-pattern the peak range of memory accesses for one output is requests in opposing quadrants of the input video frame. This equals $\sim 2\left(\frac{256}{2}\right)^2$ pixel locations and is demonstrated in Figure 7(a) with variations in excess of $30k$ pixel locations. In general the maximum variation is $\sim 2\left(\frac{N_{in}}{2}\right)^2$ pixels. This is significantly larger than for raster-scan.

For decimation, Figure 7(b) demonstrates a similar scenario. Input frame size is $s_x^{-1} \times s_y^{-1}$ times larger than for convolution, where s_x and s_y are the horizontal and vertical resizing ratios respectively. The irregular pattern for the z-pattern occurs because input frame dimensions are buffered up to a power of two.

The off-chip memory access patterns (A) for each case study are shown in Figure 8. These patterns approximate the on-chip accesses as expected. A two to three order of magnitude reduction in the number of on-chip ($|C|$) to off-chip ($|A|$) accesses is observed in all cases. This indicates good cache performance. The raster access pattern has the lowest value of $|A|$. This is in fact the optimum value for each case study. For the z-pattern, $|A|$ is within 1.5 times that for a raster scan pattern. The difference is due



(a) Convolution size 5×5

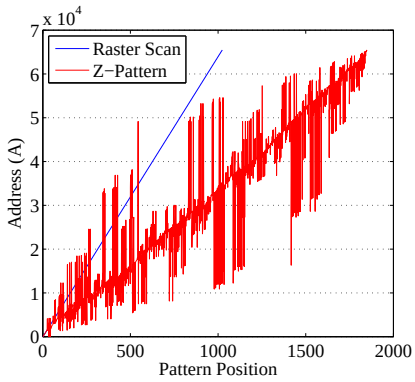


(b) $1080p$ to $480p$ Decimation

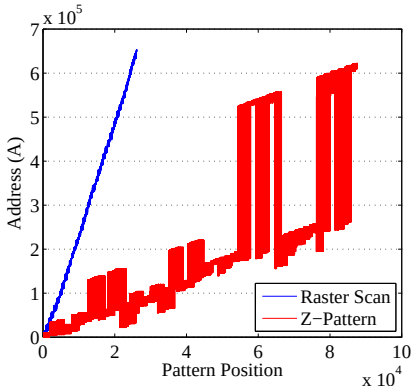
Fig. 8. Off-chip memory access (A) performance for decimation and 2D convolution for a model with one PE and one execution thread. Output frame size is 256×256 pixels.

to the larger degree of variance in on-chip accesses (C). A greater variation in memory address location also correlates with a poor DRAM memory access performance [31].

Due to the large reduction between $|C|$ and $|A|$, the performance for each choice of access pattern is not bounded by off-chip memory accesses. The estimated number of clock cycles required is $4.06M$ for decimation and $2.1M$ for 5×5 convolution in both access pattern scenarios. It is interesting to now consider the extension of these issues to a system model for the case of multiple threads ($\widehat{T}$) and PEs.



(a) Convolution Size 5×5



(b) $1080p$ to $480p$ Decimation

Fig. 9. Off-chip memory access (A) performance for a model with sixteen PEs and $\widehat{T} = 1500$ (equivalent to nVidia GeForce 6800 GT). For an output frame size of 256×256 pixels.

7 System Model with Multiple Processing Elements

To demonstrate an extended case, off-chip memory access performance for the algorithms from Section 6 and a model with 16 PEs and 1500 threads is shown in Figure 9.

Despite a large degree of multi-threading, the memory access behaviour for 2D convolution in Figure 9(a) is similar to that in Figure 8(a). This is because of the large probability that data will be reused between output pixels. For the raster scan pattern $|A|$ is the optimum value, as was the case in Figure 8(a). For the z-pattern, a small increase in $|A|$ is observed from Figure 8(a) due to memory access conflicts between threads.

A significant change in the pattern of A occurs for the decimation case study in Figure 9(b). For both processing patterns an increased number of off-chip accesses ($|A|$) is required in comparison to Figure 8(b) to satisfy on-chip memory access requirements. The lowest performance is observed for the z-pattern where $|A|$ is only an order of magnitude less than number of on-chip accesses ($|C|$) (approximately that in Figure 7(b)). Three factors influence the increased in $|A|$ for the z-pattern case as explained below.

First, decimation has less potential for pixel reuse (between neighbouring outputs) than convolution. The decimation factor in Figure 9(b) is $s_x = 3, s_y = 2.25$. This translates to a proportion of pixel reuse between two outputs of $\frac{4}{16}$ to $\frac{8}{16}$. In comparison, for convolution size 5×5, the pixel reuse is $\frac{20}{25}$. For decimation a greater number of threads require a different cache line to the previous thread. This increases cache misses.

Second, the variation in C. The choice of a non-power of two resizing ratio is shown to make this pattern irregular in Figure 7(b). This increases conflict cache misses.

Third, the cache replacement policy is also inefficiently utilised due to the non-power of two input frame size.

The increase in $|A|$ in Figure 9 is reflected in the number of clock cycles per output (CPO_m). For convolution CPO_m increase between raster and z-pattern method from 58 to 62. The extra latency for increased number and variance of A, for the z-pattern method, is mostly hidden through the combination of multi-threading and a large number (5×5 in Figure 9(a)) of spatially local on-chip memory accesses.

For decimation, the change in CPO_m between raster and z-pattern methods is more significant. In this case the raster scan and z-pattern scenarios require 92 and 250 CPO_m respectively. The z-pattern method is off-chip memory access bound under these conditions. A raster scan processing pattern is advantageous under the case study scenario of low data reuse potential. This is explored further in Section 9.

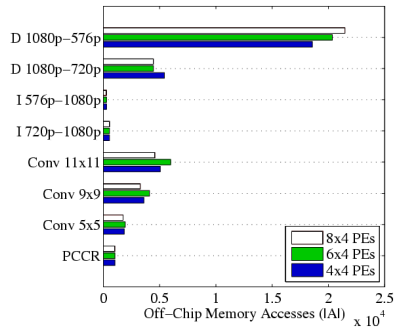
8 Architecture Trends

In this section the model is used to explore architectural trends for number of PEs, cache size and number of threads. This exemplifies **Stage a** in Figure 2(b).

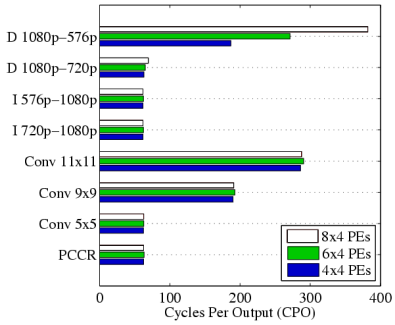
A summary of the number of off-chip memory accesses ($|A|$) and clock cycles per output (CPO) for changing number of PEs, for four case study algorithms, is shown in Figures 10(a) and 10(b). For all tests the system model setup captures the performance of the GeForce 6800 GT, the number of computational threads is $\widehat{T} = 1500$, and a z-pattern processing order is used throughout. Number of PEs is the variable.

The case study algorithms are bi-cubic decimation, bi-cubic interpolation, 2D convolution and primary colour correction algorithms. The last three are taken from [28].

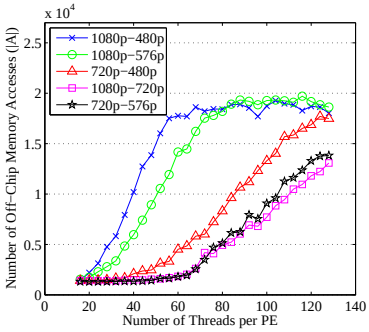
For primary colour correction, convolution and interpolation CPO remains consistent across all numbers of PEs. Primary colour correction is a computationally bound algorithm so this is as expected, the value of $|A|$ is minimum at 1024.



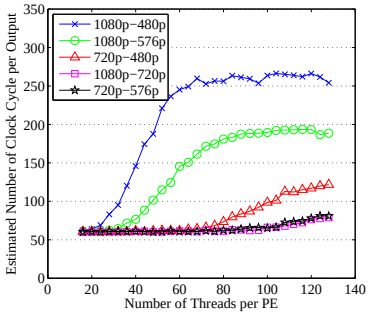
(a) $|A|$ for Varying Case Studies and PEs with Cache Size 16KB



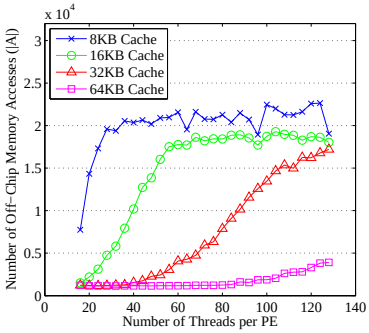
(b) CPO_m for Varying Case Studies and PEs with Cache Size 16KB



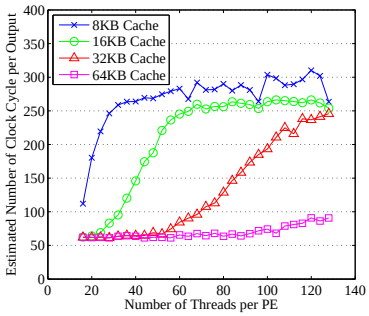
(c) $|A|$ for Varying Decimation Ratios with 4×4 PEs and Cache Size 16KB



(d) CPO_m for Varying Decimation Ratios with 4×4 PEs and Cache Size 16KB



(e) $|A|$ for Varying Cache Size for Decimation $1080p$ to $480p$ and 4×4 PEs



(f) CPO_m for Varying Cache Size for Decimation $1080p$ to $480p$ and 4×4 PEs

Fig. 10. Performance for Varying Design Space Parameters (as Indicated) and Case Studies (with a fixed 256×256 pixel input frame size)

For the setup in Figure 10, each MMU has direct access to on-chip memory. Therefore, the on-chip memory bandwidth scales linearly with the number of PEs. This explains the equal value of CPO for all variants of number of PEs.

A change of up to three times in $|A|$ is observed for 5×5 to 11×11 sized convolutions. In all cases CPO remains unchanged. The increase in $|A|$ is hidden by multi-threading and a large number of on-chip memory accesses.

In similarity to the exploration in Section 7, interpolation exhibits a greater potential for data reuse between neighbouring outputs (from $\frac{12}{16}$ to $\frac{16}{16}$) than 2D convolution (fixed at $\frac{(n_{conv}-1)n_{conv}}{n_{conv}^2}$). CPO is therefore also equal for each interpolation ratio. This is despite a difference in $|A|$ of approximately two times. The multi-threaded computation is again successful in hiding off-chip memory access requirements. For convolution and interpolation the critical path is the *on-chip memory bandwidth*.

The most significant variations in $|A|$ and CPO occur for the decimation case study. Whilst $1080p$ to $720p$ decimation has a consistent CPO across all numbers of PEs the $1080p$ to $576p$ case shows more significant variations. This is due to a larger scaling factor for the $1080p$ to $576p$ case. In this scenario $|A|$ (and cache misses) is large enough to make decimation off-chip memory access bound. The performance degradation is proportional to the number of PEs.

Decimation is investigated further in Figures 10(c) to 10(f).

In Figures 10(c) and 10(d) varying decimation ratios are plotted for a setup of 16 PEs and a 16 KByte cache size, which is equivalent to a GeForce 6800 GT. As the number of threads per PE is increased CPO and $|A|$ increase. This effect is most prominent for larger decimation factors. It is observed that as the resizing factor increases, the CPO trend adopts increasing similarity to the $|A|$ trend. This is concurrent with the system being *off-chip memory bandwidth bound*.

For scenarios where the system becomes memory bound there are two choices. First to reduce the number of threads ($\widehat{T}$) or secondly to increase cache size. For the worst performing decimation size ($1080p$ to $480p$) this tradeoff is shown in Figures 10(e) and 10(f). It is observed that as the cache size is increased, $|A|$ (and ultimately CPO) decreases sharply. An approximately linear relationship between CPO and cache size is observed. Ultimately further non-modelled factors affect the performance as cache size is increased, for example, increased cache access latency.

To summarise the relationship between thread count $\widehat{T}$ and required memory size B_{req} for good cache performance is shown in Equation 1, where R and N_{reads} are the amount of data reuse (exemplified above) and number of reads respectively.

$$B_{req} > N_{reads} + (1 - R)\widehat{T} \tag{1}$$

If B_{req} exceeds on-chip memory size a large value of $|A|$ is expected. As B_{req} approaches the on-chip memory size, $|A|$ increases subject to the choice of reuse heuristic (in this case a 4-way associative cache). This effect is exemplified for increasing $\widehat{T}$ in Figures 10(c) to 10(f). A shift in the graphs is observed for varying cache size as the number of threads and memory size change. For decimation the reuse R is inversely proportional to the resizing factors s_x and s_y in x and y dimensions respectively.

The model performance for case study algorithms in Figures 10(a) and 10(b) can be compared to results for two sample graphics processors as shown in Table 3.

	CPO$_{gf6}$	**CPO**$_{gf7}$	**CPO**$_m$
PCCR	60	43	63
2D Conv (5×5)	41	37	62
2D Conv (9×9)	162	144	187
2D Conv (11×11)	263	230	282
Interp (576p-1080p)	52	47	61
Interp (720p-1080p)	53	48	61
Deci (1080p-576p)	90	84	187
Deci (720p-480p)	78	68	86
Deci (1080p-720p)	69	66	75

Table 3. Verification of the Model. CPO is cycles per output for the model (m), nVidia GeForce 6800 GT (gf6) and nVidia GeForce 7900 GTX (gf7)

Over all case studies the CPO for the model (CPO_m) approximate those for the nVidia GeForce 6800 GT (CPO_{gf6}) and follow the trend for the GeForce 7900 GTX (CPO_{gf7}). Cycle estimates are higher for the model because its architecture is not pipelined to the extent of a graphics processor.

For the GeForce 6800 GT, the number of clock cycles per internal memory access (CPIMA) for 2D convolution is a constant 2.4. The results for the model (CPO$_m$) range from 290 to 62 cycles per output for convolution size 11×11 and 5×5 respectively. This equals to a value of CPIMA of approximately 2.4 to 2.5. The model mimics the on-chip memory timing of the graphics processor well.

Model implementations of small decimation sizes are well matched to the performance of the graphics processor. A small overhead in number of cycles per output is again observed over equivalent implementations on the GeForce 6800 GT.

An anomaly occurs for decimation size $1080p$ to $576p$ where model results deviate from those for the graphics processors. Four potential reasons for this are as follows.

1. The multi-threaded behaviour is not fully pipelined within the MMU. For the case of large decimation sizes this amplifies memory access cost.
2. The computation model does not fully synchronise the execution of all PEs. This is again troublesome for algorithms with poor memory access behaviour.
3. The cache size estimate of 16 KBytes for the nVidia GeForce 6800 GT may be incorrect. If a cache size of 32 KBytes is taken CPO$_m$ reduces to 74.
4. Although a latency and delay based model of off-chip DRAM is created, the latency of the entire interface to the DRAM and finer detail of the DRAM is omitted.

Despite the above limitations the model is observed in Table 3 to, under correct parametrisation, correlate well with the performance of two sample graphics processors.

The run time for the system model is between 1 and 5 minutes for all case studies. Although the model is not optimised, it is sufficient for exploration with large input frame sizes. For a frame size of 2048×2048 pixels the simulation time increases to a manageable 25 minutes for the decimation case study.

9 Post-Fabrication Customisable Options

In this Section the results in Sections 6, 7 and 8 are used to reason post-fab customisation options (this exemplifies **Stage b** in Figure 2(b)).

First, the off-chip memory system performance is the critical path for large decimation factors. This prompts investigation into ways to improve the memory system for the memory access pattern of decimation. In [14] the authors investigate this option which is an execution of **Stage c** in Figure 2(b) and promotes the exploration tool.

Second, to change the choice of PE in Figure 4. In general a PE may not be a processor and may support reconfiguration. This option is considered by the authors in [28]. An example application is to support local data reuse for a PE to overcome the on-chip memory bandwidth critical path for convolution and interpolation.

Third, a final option not previously considered is to alter the processing pattern. The opportunity for this has been demonstrated in Sections 6 and 7. This option is now used to demonstrate **Stage c** of the approach in this work as outlined below.

To quantify the changing of processing pattern over different cache sizes and decimation factors consider a summary of the performance of a raster and z-pattern as shown in Table 4. In either case the pattern is used for both processing order and memory storage with all else constant. It is observed that for large decimation factors up to a four times reduction, in both number of memory accesses and cycles per output, is achieved from using a raster scan pattern.

	Z-Pattern			**Raster Scan**		
	16KB	**32K**	**64K**	**16K**	**32K**	**64K**
A	127319	66984	10223	29828	8910	7596
	(258)	(146)	(63)	(92)	(62)	(62)
B	87671	15387	6084	26133	8112	5138
	(180)	(63)	(62)	(82)	(62)	(62)
C	29144	6481	6481	15779	12473	3591
	(79)	(62)	(62)	(68)	(66)	(62)
D	12756	3124	3124	13835	4332	2695
	(62)	(62)	(62)	(66)	(62)	(62)
E	12347	2770	2770	12967	4783	2568
	(63)	(62)	(62)	(66)	(62)	(62)

A=1080p to 480p, B=1080p to 576p, C=720p to 480p, D=1080p to 720p and E=720p to 576p
Table 4. Number of Off-Chip Memory Accesses and (Cycles per Output) for Varying Processing Patterns and Decimation

As reasoned in Section 7, the justification is that, for the raster scan case, conflict cache misses only occur due to the horizontal resizing factor. For the z-pattern approach, cache misses occur due to both horizontal and vertical resizing factors due to the 2D nature of the z-pattern. As cache size is increased, the benefit of the raster scan approach diminishes. This is because the algorithm becomes on-chip memory access limited under these conditions, for which the access time and latency is fixed. For smaller decimation factors the z-pattern can be beneficial over the raster scan approach. This occurs when the horizontal resizing factor exceeds the vertical factor. A vertical raster pattern could be used to alleviate this issue.

The choice of processing and memory storage pattern is shown to have a significant effect on a subset of algorithms with low data reuse potential. For a graphics application the z-pattern is the optimal choice. This therefore presents an avenue for post-fab

customisation to switch between alternative processing patterns depending on the target application domain. The mapping between a z-pattern and raster scan pattern requires bit reordering as explained in Section 5. In the case of two alternative patterns this is implemented with one multiplexor and a configuration bit.

Intentionally, this example is straight forward as a demonstration of the exploration process. It is observed through related work [14, 22, 25] that the exploration of post-fab customisation options can provide even higher performance improvements, of up to an order of magnitude, albeit for a higher area cost than the example here.

As with [14] the example presented above is a class 5 customisation from Figure 1.

10 Implications for other Graphics Processors and the Cell BE

Whilst the results above are based on the nVidia GeForce 6 and 7 series graphics processors, the current state of the art has progressed, examples are considered below.

The architecture template of the AMD ATI Radeon and nVidia GeForce 8 (more recently GeForce 9) series graphics processors is fundamentally similar to the model in Figure 4. A large number of PEs are arranged in processing groups and arbitrate through a local MMU to access off-chip memory through shared on-chip memory (cache).

One difference for the Radeon and GeForce 8 graphics processors is that fragment and vertex pipelines are combined in a unified shader. However, the level of abstraction in Figure 4(b) could equally represent a unified shader, in contrast to only the fragment pipeline. For 2D video processing, vertex processing requirements can be disregarded because they are trivially four corner coordinates of the output frame.

The processing elements in the GeForce 8 graphics processors are different from prior GeForce generations. For example, the GeForce 8 now contains scalar PEs. An advantage of scalar processors is a reduction in cycle count through increased processor utilisation, over a 4-vector processor performing computation on 3-component video data. This modification is trivial to support in the PE model in Figure 4.

If the implementations from Section 8 were directly ported to a Radeon or GeForce 8 architecture a similar performance trend would be observed, with variations due to a different trade off of number of PEs, on-chip memory size and number of threads.

Current state of the art AMD ATI Radeon and nVidia GeForce 8 generation of graphics processors have an enhanced and more flexible 'application mapping' which is programmable through the CTM (close to metal) and CUDA (compute unified device architecture) programming environments respectively. An immediate advantage is that off-chip memory accesses can be reduced for previously multi-pass algorithms through storage of intermediate results in on-chip memory for later reuse. In addition the contents of on-chip memory can be controlled. This presents an exciting new domain of algorithm optimisations, for example, the ability to control, within limits, the contents of on-chip memory may improve performance for the decimation case study.

The Cell BE presents a shift from the model adopted in Figure 4. In addition to shared global memory a large memory space is local to each processing group. This can be considered as local to the MMU. However, DMA access can be made between MMUs over the EIB^{TM} bus. Processing groups also operate independently which poses further opportunities for algorithm optimisations.

One intriguing possibility is to consider post-fab customisation of the Cell BE DMA engine. In one instance the customisable DMA engine may be used to implement an address mapping function similar to that in [14]. Alternatively, a grander opportunity is a configurable DMA engine that on-prompt generate its own addressing patterns.

In general, for alternative HoMPs the core architecture features in Figure 3 are consistent, with minor variations. The key difference is in application mapping characteristics. These include the choice of address space (local, global and control) and restrictions on PE execution behaviour.

The results in Sections 6, 7 and 8 show some of the architectural trends for the core architecture features which are present in all HoMPs. However, for each HoMP a different application mapping is chosen. This translates to a new algorithm set of optimisation techniques. Re-parametrisation of the model's application mapping feature set, and choice of algorithm optimisations, is required to support alternative HoMPs.

11 Summary

A novel design space exploration tool has been presented with the application of exploring the customisation options for a Homogeneous Multi-Processor (HoMP). The tool has been demonstrated using the example of an architecture which captures the behaviour of a graphics processor and an application domain of video processing.

To provide a broadened prospective of the work a classification scheme for post-fab options was presented in Section 3. The effectiveness of the classification has been demonstrate through its application to prior art and to classify the proposal in Section 9.

Our exploration tool is divided into a systematic approach to exploring customisation options and a system model. The systematic approach in Section 4 is an adapted version of the well known Y-Chart method, with an adaptation to capture specifically the architectural features which support the programming model. As part of the approach the customisation options are separated into post- and pre-fabrication options. The associated model, in Section 5, comprises high-level descriptors and is implemented using the SystemC class library and a Kahn process network structure.

Architecture performance is explored using the model. In Section 6, the effect of processing pattern on a single PE and thread example is analysed. This analysis is extended to the multiple PE and multiple thread case in Section 7. The analysis in both sections promotes the post-fabrication customisation option presented in Section 9.

Architecture trends are explored using four case study examples in Section 8. The options of number of PEs, number of threads and cache size are demonstrated. Alongside these results the model is verified and critiqued against two graphics processors. The behaviour of the graphics processors is shown to be captured by the model.

The architecture trends and analysis from Sections 6 and 7 are used to propose post-fabrication customisation options in Section 9. A positive result is to customise processing pattern which improves performance by four time for a negligible area cost. This is a class 5 'glue logic' use of reconfigurable logic from Section 3.

A grander result of the paper is that the work demonstrates the strengths of the exploration tool and classification in the design of a customised HoMP. We hope that the

work will stimulate future research in this area.

In addition to automation as mentioned in Section 4, further work would involve exploring customisation options for other homogeneous multi-processors including the Cell BE, Radeon and GeForce 9 series. The Intel Larrabee, due to be released in 2009, is a newer architecture which may also present exciting opportunities for customisation.

It is also interesting to investigate customising a processor's memory subsystem. In particular customisation of the mapping of off-chip to on-chip memory in a Cell BE DMA engine. Finally, it is also important to study customisation of system interconnects [22] and heterogeneous architectures.

Acknowledgement: we gratefully acknowledge support from Sony Broadcast & Professional Europe and the UK Engineering and Physical Sciences Research Council.

References

1. Vassiliadis, S., et al: Tera-device computing and beyond: Thematic group 7. Roadmap: ftp://ftp.cordis.europa.eu/ pub/fp7/ict/docs/fet-proactive/masict-01_en.pdf (2006)
2. Bosschere, K.D., et al: High-performance embedded architecture and compilation roadmap. In: Transactions on HiPEAC LNCS 4050. (2007) 5–29
3. Cope, B., Cheung, P.Y.K., Luk, W.: Systematic design space exploration for customisable multi-processor architectures. In: SAMOS. (July 2008) 57–64
4. Keinhuis, B., et al: An approach for quantitative analysis of application-specific dataflow architectures. In: ASAP. (July 1997) 338–350
5. Lieverse, P., et al: A methodology for architecture exploration of heterogeneous signal processing systems. Journal of VLSI Signal Processing **29**(3) (2001) 197–207
6. Moya, V., Golzalez, C., Roca, J., Fernandez, A.: Shader performance analysis on a modern GPU architecture. In: IEEE/ACM Symposium on Microarchitecture. (2005) 355–364
7. Sheaffer, J.W., Skadron, K., Luebke, D.P.: Fine-grained graphics architectural simulation with qsilver. In: Computer Graphics and Interactive Techniques. (2005)
8. Nvidia: nvidia shaderperf 1.8 performance analysis tool. http://developer.nvidia.com/object/nvshaderperf_home.html
9. Govindaraju, N.K., Larsen, S., Gray, J., Manocha, D.: A memory model for scientific algorithms on graphics processors. In: ACM/IEEE Super Computing. (2006) 89–98
10. Kahn, G.: The semantics of a simple language for parallel programming. In: IFIP Congress. (1974)
11. Rissa, T., Donlin, A., Luk, W.: Evaluation of systemc modelling of reconfigurable embedded systems. In: DATE. (March 2005) 253–258
12. Donlin, A., Braun, A., Rose, A.: Systemc for the design and modeling of programmable systems. In: Proceesings of FPL LNCS 3203. (August 2004) 811–820
13. Todman, T.J., Constantinides, G.A., Wilton, S.J., Mencer, O., Luk, W., Cheung, P.Y.: Reconfigurable computing: Architectures and design methods. IEE Computers and Digital Techniques **152**(2) (2005) 193–207
14. Cope, B., Cheung, P.Y.K., Luk, W.: Using reconfigurable logic to optimise gpu memory accesses. In: DATE. (2008) 44–49
15. Moll, L., Heirich, A., Shand, M.: Sepia: Scalable 3d compositing using pci pamette. In: FCCM. (April 1999) 146–155

16. Manzke, M., Brennan, R., O'Conor, K., Dingliana, J., O'Sullivan, C.: A scalable and reconfigurable shared-memory graphics architecture. In: Computer Graphics and Interactive Techniques. (August 2006)
17. Xue, X., Cheryauka, A., Tubbs, D.: Acceleration of fluoro-ct reconstruction for a mobile c-arm on gpu and fpga hardware: A simulation study. SPIE Medical Imaging 2006 **6142**(1) (2006) 1494–1501
18. Kelmelis, E., Humphrey, J., Durbano, J., Ortiz, F.: High-performance computing with desktop workstations. WSEAS Transactions on Mathematics **6**(1) (January 2007) 54–59
19. Schleupen, K., Lekuch, S., Mannion, R., Guo, Z., Najjar, W., Vahid, F.: Dynamic partial fpga reconfiguration in a prototype microprocessor system. In: FPL. (August 2007) 533–536
20. Tremblay, M., Chaudhry, S.: A third-generation 65nm 16-core 32-thread plus 32-scout-thread cmt sparc processor. In: Proceedings of the IEEE ISSCC. (February 2008) 82–83
21. Dale, K., et al: A scalable and reconfigurable shared-memory graphics architecture. In: ARC LNCS 3985. (March 2006) 99–108
22. Yalamanchili, S.: From adaptive to self-tuned systems. In: Symposium on THE FUTURE OF COMPUTING in memory of Stamatis Vassiliadis. (2007)
23. MathStar: Field programmable object arrays: Architecture. http://www.mathstar.com/Architecture.php (2008)
24. Chen, T.F., Hsu, C.M., Wu, S.R.: Flexible heterogeneous multicore architectures for versatile media processing via customized long instruction words. IEEE Transactions on Circuits and Systems for Video Technology **15**(5) (2005) 659–672
25. Verbauwhede, I., Schaumont, P.: The happy marriage of architecture and application in next-generation reconfigurable systems. In: Computing frontiers. (April 2004) 363–376
26. Nollet, V., Verkest, D., Corporaal, H.: A quick safari through the mpsoc run-time management jungle. In: Workshop on Embedded Systems for Real-Time Multimedia. (October 2007) 41–46
27. Shin, D., et al: Automatic generation of transaction level models for rapid design space exploration. In: Proceedings of Hardware/software codesign and system synthesis. (October 2006) 64–69
28. Cope, B., Cheung, P.Y.K., Luk, W.: Bridging the gap between FPGAs and multi-processor architectures: A video processing perspective. In: Application-specific Systems, Architectures and Processors. (2007) 308–313
29. Priem, C., Solanki, G., Kirk, D.: Texture cache for a computer graphics accelerator. United States Patent No. US 7,136,068 B1 (1998)
30. Jin, Q., Thomas, D., Luk, W., Cope, B.: Exploring reconfigurable architectures for financial computation. In: ARC LNCS 4943. (March 2008) 245–255
31. Ahn, J.H., Erez, M., Dally, W.J.: The design space of data-parallel memory systems. In: ACM/IEEE Super Computing. (November 2006) 80–92