An architectural level design methodology for smart camera applications

Sankalita Saha*

MCT/NASA Ames Research Center,
Moffett Field, CA, USA
E-mail: ssaha@mail.arc.nasa.gov
*Corresponding author

Vida Kianzad

The MathWorks,
Natick, MA, USA
E-mail: vkianzad@mathworks.com

Jason Schlessman

Department of Electrical Engineering,
Princeton University,
Princeton, NJ, USA
E-mail: jschless@princeton.edu

Gaurav Aggarwal

ObjectVideo Inc.,
Reston, VA, USA
E-mail: gaggarwal@objectvideo.com

Shuvra S. Bhattacharyya and Rama Chellappa

Department of Electrical and Computer Engineering,
Institute for Advanced Computer Studies,
University of Maryland,
College Park, USA
E-mail: ssb@umd.edu
E-mail: rama@umd.edu

Wayne Wolf

School of Electrical and Computer Engineering,
Atlanta, Georgia, USA
E-mail: wolf@ece.gatech.edu

Abstract: Today's embedded computing applications are characterised by increased functionality, and hence increased design complexity and processing requirements. The resulting design spaces are vast and designers are typically able to evaluate only small subsets of solutions due to lack of efficient design tools. In this paper, we propose an architectural level design methodology that provides a means for a comprehensive design space exploration for smart camera applications and enable designers to select higher quality solutions and provides substantial savings on the overall cost of the system. We present efficient, accurate and intuitive models for performance estimation and validate them with experiments.

Keywords: design space exploration; smart camera; synchronisation.

Reference to this paper should be made as follows: Saha, S., Kianzad, V., Schlessman, J., Aggarwal, G., Bhattacharyya, S.S., Chellappa, R. and Wolf, W. (2009) 'An architectural level design methodology for smart camera applications', *Int. J. Embedded Systems*, Vol. 4, No. 1, pp.83–97.

Biographical notes: Sankalita Saha is a Research Scientist in the Intelligent Systems Division at MCT/NASA Ames Research Center at Moffett Field, CA. She earned her BTech Degree in Electronics and ECE from IIT Kharagpur, India in 2002 and a Doctoral Degree in Computer Engineering from University of Maryland, College Park, MD in 2007.

Vida Kianzad is a Senior Software Engineer with the Simulink application group at the Mathworks. She received a PhD Degree in Computer Engineering from University of Maryland, College Park in 2006.

Jason Schlessman earned the BS and MS in Computer Engineering from Lehigh University in 2001 and 2002, respectively. He earned the MAE in Electrical Engineering from Princeton University in 2004.

Gaurav Aggarwal received the BTech Degree in Computer Science from IIT Madras, in 2002 and the Doctoral Degree in Computer Science from the University of Maryland, College Park, in 2008. He is currently working as a Senior Research Scientist with ObjectVideo Inc.

Shuvra S. Bhattacharyya is a Professor in the Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies at the University of Maryland, College Park. He received a PhD Degree from the University of California, Berkeley in 1994.

Rama Chellappa is a Minta Martin Professor of Engineering and the Director of Center for Automation Research at University of Maryland. He received his PhD Degree from Purdue University in 1981.

Wayne Wolf is Farmer Distinguished Chair and Georgia Research Alliance Eminient Scholar at the Georgia Institute of Technology. He received his BS, MS, and PhD in Electrical Engineering from Stanford University in 1980, 1981, and 1984, respectively.

1 Introduction

There has been an exponential rise in the number of devices integrated on a single chip in recent times. Consequently, system designers can implement even more complex tasks and place more functionality into a small area or limited set of components. This trend has made it increasingly difficult for designers to keep track of the full range of possible implementation options; to make a high quality system design, relative to the underlying design space; and deliver a bug free system within the given time-to-market window. As a result, there is a widening gap between the silicon capacity and the design productivity.

In this paper, we present a study of design, modelling, architecture exploration and synthesis for two smart camera applications – an embedded face detection system and a 3D facial pose tracking system. Smart cameras have attracted great attention in recent years due to rapidly growing demand for various biometric applications in video surveillance, security system access control, video archiving, and tracking. Face detection and 3D tracking are important applications for smart cameras. Face detection pertains to discerning the existence of human faces within an image or video sequence. 3D facial pose tracking pertains to detecting movement of the human face or facial pose in video. Both of these applications have an implicit and in many cases critical need for real-time performance.

Furthermore, these applications are mostly targeted for mobile and outdoor deployment, which mandates the consideration of power consumption, memory size, and area when implementing the associated embedded system. However, neither of these applications – neither face detection nor 3D facial pose tracking – is a simple problem. Both require computational power well beyond that deemed acceptable for mobile systems. Hence, the real-time realisation of them can only be achieved by aggressive application of parallel processing and pipelining as provided by multiprocessor systems. Such implementation involves the interaction of several complex factors including scheduling, inter-processor communication, synchronisation, iterative execution, and memory/buffer management. Addressing any one of these factors in isolation is itself typically intractable in any optimal sense. At the same time, with the increasing trend toward multi-objective implementation criteria in the synthesis of embedded software, it is desirable to understand the joint impact of these factors.

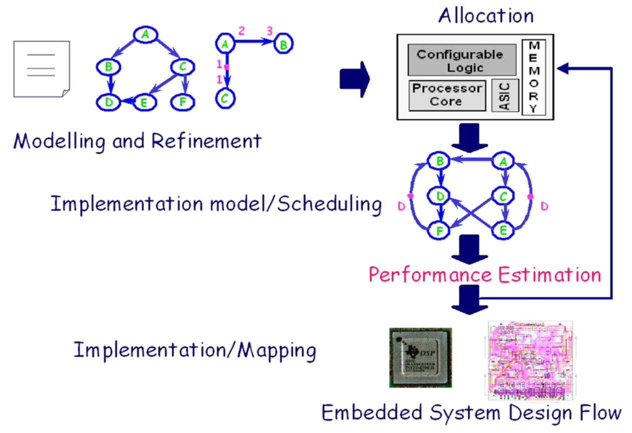
With the aforementioned design complexity and requirement issues in mind, we study in this work the design and synthesis of

- an embedded face detection system for a class of reconfigurable system-on-chips
- an embedded 3D facial pose tracking system for a class of multiprocessor systems and PDSPs.

Following a standard top-down design flow (see Figure 1), we offer solutions and methodologies for the modelling of the application to be developed, modelling of the implementation to be derived, and estimation of performance for candidate implementations. We also briefly cover the partitioning step. In addition to providing details and experimental results (for the mentioned design steps) for a useful family of face detection and 3D facial pose tracking architectures, the following contributions on design methodology emerge from our study.

- We develop a number of useful generalisations to the synchronisation graph performance analysis model (Sriram and Lee, 1997). First, we demonstrate the first formulation and use of multirate synchronisation graphs, which we show to be useful for compactly representing repetitive patterns in the execution of a multiprocessor image processing system. Second, we integrate aspects of ordered transaction execution (Sriram and Lee, 1997) with more conventional, self-timed execution in a flexible and seamless way. This demonstrates a new class of hybrid self-timed/ordered transaction designs. Furthermore, it demonstrates that the synchronisation graph modelling methodology can be used to unify analysis across the entire spectrum of systems encompassing pure self-timed execution, pure ordered transaction execution, and the set of hybrid self-timed/ordered transaction possibilities that exist between these extremes. Third, we show how our multirate synchronisation graph approach, together with hybrid self-timed/ordered transaction scheduling, can be used to effectively design systems involving extensive multi-dimensional processing (in our case, processing of video frames). Previous development of synchronisation graph modelling has focused primarily on single-dimensional, signal processing systems.

Figure 1 Embedded system design flow (see online version for colours)



- We then apply our generalised synchronisation graph modelling approach as a designer's aid for architecture analysis and exploration. Here, the synchronisation graph is a applied as a conceptual tool that can be easily targeted and adapted to different implementation alternatives due to the high level of abstraction at which it operates. In contrast, previous development of synchronisation graphs has focused on their use as an intermediate representation in automated tools (Sriram and Bhattacharyya, 2000).
- We show how for application/domain-specific design, the system designer can effectively use our new modelling approach as a way to understand and explore performance issues (regardless of whether the model is supported by the synthesis tools employed). Our parameterised construction of the multirate synchronisation graph for the targeted class of application/architecture mappings together with our analysis of graph cycles in this parameterised synchronisation graph concretely demonstrates the steps that are needed for this type of design methodology.

2 Related work

With an increase in the complexity and performance requirements of applications, designers have resorted to looking beyond pure hardware or software solutions. As a result of this, the overall design space for a system has vastly increased, calling for systematic methods to explore this space and utilise the available resources effectively. An efficient design space exploration tool can dramatically impact the area, performance, and power consumption of the resulting systems. An optimised implementation on such hybrid platforms requires sophisticated techniques. These techniques start with the specification and design requirements of the system, and search the solution space to find one or more implementations that meet the given constraints.

This problem consists of three major steps,

- resource (computation/communication) allocation
- assignment and mapping of the system functionality onto the allocated resources
- scheduling and ordering of the assigned functions on their respective resources.

Addressing each of these steps is an intractable problem, and most of the existing techniques are simulation-based heuristics. Some recent studies have started to use formal models of computation as an input to the partitioning problem to allow the exploration of much larger solution spaces (Ziegenbein et al., 1998), and to use such models for design space exploration and optimisation (Karkowski and Corporaal, 1998). A methodology for system level design space exploration is presented by Auguin et al. (2001), but the focus is purely on partitioning and deriving system specifications from functional descriptions of the application. Peixoto and Jacome (1997) give a comprehensive framework for algorithmic and design space exploration along with definitions for several system-level metrics (Peixoto and Jacome, 1997). However, the associated methodology is complex and does not address the special needs of embedded systems. Kwon et al. (2004) propose a design exploration framework that makes estimations about performance and cost (Kwon et al., 2004), but the performance estimation techniques are based on instruction set simulation of architectures, which restricts the platforms to which it can be applied. Miramond and Delosme (2005) also propose methodologies for design space exploration as well performance estimations, but they restrict themselves to purely reconfigurable architectures (Miramond and Delosme, 2005).

Due to the limitations described above in this body of work, further attention is needed for design methodology and architectural exploration. There is little work that treats the problem of embedded system design with stringent, integrated attention to data accesses, interprocessor communication, and synchronisation, as well as resource assignment and task scheduling. Such an integrated approach is especially important in real-time image processing, where large volumes of data must be handled with predictable performance. Bridging this gap and facilitating more systematic coupling between embedded system algorithms and their embedded implementations are major objectives of this paper. The first version of our proposed methodology, for this purpose, and preliminary results were presented in Kianzad et al. (2005). In this paper, we present a more detailed version of the methodology, along with additional experimental results. More specifically, we present results for an additional application (3D facial pose tracking) on two different platforms (a shared memory multiprocessor system and a PDSP). These results further demonstrate our methods and provide evidence of their versatility.

3 Smart camera applications

3.1 Face detection algorithm

Face detection research has been an active area of research for the past few decades. There are several approaches that make use of shape and/or intensity distribution on the face. In this work, we use a shape-based approach as proposed by Moon et al. (2002). A face is assumed to be an ellipse. This method models the cross-section of the shape (ellipse) boundary as a step function. Moon et al. (2002) prove that the Derivative of a Double Exponential (DODE) function is the optimal one-dimensional step edge operator, which minimises both the noise power and the mean squared error between the input and the filter output. The operator for detecting faces is derived by extending the DODE filter along the boundary of the ellipse. The probability of the presence of a face at a given position is estimated by accumulating the filter responses at the centre of the ellipse. Quite clearly, this approach of face detection seems like a natural extension of the problem of edge detection at the pixel level to shape detection at the contour level.

At the implementation level, this reduces to finding out correlations between a set of ellipse shaped masks with the image in which a face is to be detected. Figure 2 shows the complete flow of the employed face detection algorithm. A few examples of detection outputs using the described approach are presented in Figure 3.

Figure 2 The flow of the employed face detection algorithm

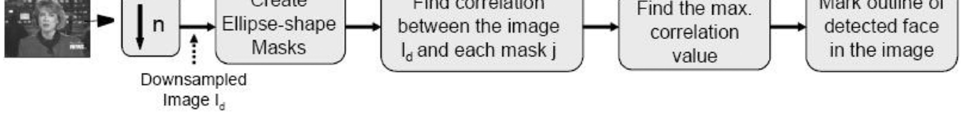
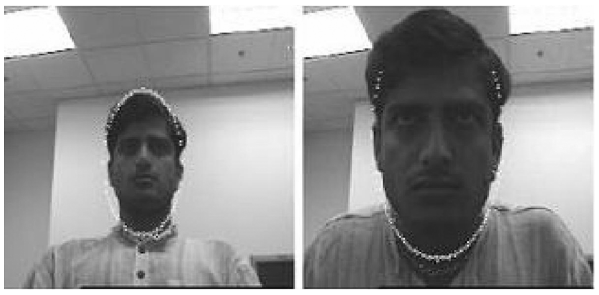

n
Create Ellipse-shape Masks
Find correlation between the image I_d and each mask j
Find the max. correlation value
Mark outline of detected face in the image
Downsampled Image I_d

Figure 3 Results of applying the face detection algorithm to two images



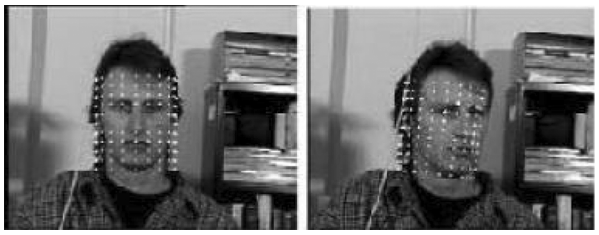
3.2 3D facial pose tracking algorithm

The aim in facial pose tracking is to recover the 3D configuration of a face in each frame of a video. The 3D configuration consists of three translation parameters and three orientation parameters that correspond to the yaw, pitch and roll of the face. The 3D tracking algorithm considered in this work uses the particle filtering technique along with geometric modelling (Aggarwal et al., 2005).

There are three main aspects that capture the 3D tracking system. The first is the model to represent the facial structure. The second is the feature vector used. The third is the tracking framework used. A model attempts to approximate the shape of the object to be tracked in the video. In our application, a cylinder with an elliptical cross-section is chosen as a model to represent the 3D structure of face.

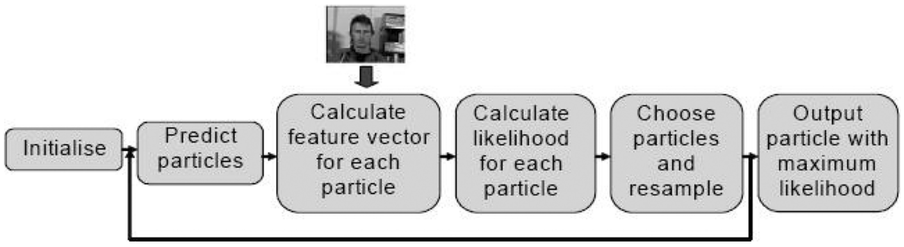
The feature vector represents characteristics from the image. A rectangular grid superimposed around the curved surface of the elliptical cylinder is used for this purpose: the mean intensity for each of the visible grids/cells forms the feature vector. Given the current configuration, the grids can be projected onto the image frame and the mean can be computed for each of them. Figure 4 illustrates the model along with the feature vector.

Figure 4 An example of 3D facial pose tracking with a cylindrical mesh



For the tracking framework, i.e., estimating the configuration or pose of the moving face in each frame of a given video, a particle filter based technique is used.

Figure 5 Flow of 3D facial pose tracking algorithm


Initialise
Predict particles
Calculate feature vector for each particle
Calculate likelihood for each particle
Choose particles and resample
Output particle with maximum likelihood

As mentioned before, the motion of the face is characterised by three translation and three orientation parameters. For each new image frame read in from the camera, multiple predictions for these parameters are made where each prediction is a particle. The feature vector is then extracted for each particle. The particle that yields the best likelihood value gives the position of the face in the frame. The number of particles to be used in the system is decided by the user and is constant for one application. For updating the particles for the next frame, a small number of particles from the current frame are chosen based on their likelihood values and resampled. The complete algorithmic flow is given in Figure 5. The prediction is done by using a Gaussian distributed random generator.

4 Modelling approach

In this work, we build on the synchronisation graph model (Sriram and Bhattacharyya, 2000) to analyse and optimise multiprocessor implementation issues of our system. This representation is based on iterative Synchronous Dataflow (SDF) graphs (Lee and Messerschmitt, 1987). A brief introduction to these concepts is presented in this section.

In SDF, an application is represented as a directed graph in which vertices (actors) represent computational tasks, edges specify data dependencies, and the numbers of data values (tokens) produced and consumed by each actor is fixed. Delays on SDF edges represent initial tokens, and specify dependencies between iterations of the actors in iterative execution. Mapping an SDF-based application to a multiprocessor architecture includes

- assignment of actors to processors
- ordering the actors that are assigned to each processor
- determining precisely when each actor should commence execution.

In this work, we focus on the *self-timed scheduling* strategy and the closely-related *ordered transaction* strategy (Sriram and Bhattacharyya, 2000). In *self-timed scheduling*, each processor executes the tasks assigned to it in a fixed order that is specified at compile time. Before executing an actor, a processor waits for the data needed by that actor to become available. Thus, processors are required to perform run-time synchronisation when they communicate data. This provides robustness when the execution times of tasks are not known precisely or when they may exhibit occasional deviations from their compile-time estimates. It also eliminates the need for global clocks because there is no need to coordinate all processors in lockstep.

The *ordered transaction* method is similar to the *self-timed* method, but it also adds the constraint that a global, linear ordering of the interprocessor communication operations (communication actors) is determined at compile time, and enforced at run-time (Sriram and Lee, 1997). The linear ordering imposed is called the *transaction order* of the associated multiprocessor implementation. Enforcing of the transaction order eliminates the need for run-time synchronisation and bus arbitration, and also enhances predictability. Carefully constructed transaction orders can also lead to more efficient patterns of communication operations (Khandelia and Bhattacharyya, 2000).

The synchronisation graph G_s (Sriram and Bhattacharyya, 2000) is used to model the self-timed execution of a given parallel schedule for an iterative dataflow graph. Given a self-timed multiprocessor schedule for graph G, we can derive G_s by instantiating a vertex for each task, connecting an edge from each task to the task that succeeds it on the same processor, and adding an edge that has unit delay from the last task on each processor to the first task on the same processor. Each edge (v_j, v_i) in G_s is called a *synchronisation edge*, and represents the synchronisation constraint:

$$\forall k, \mathrm{start}(v_i, k) = \mathrm{end}(v_j, k - \mathrm{delay}(v_j, v_i)), \tag{1}$$

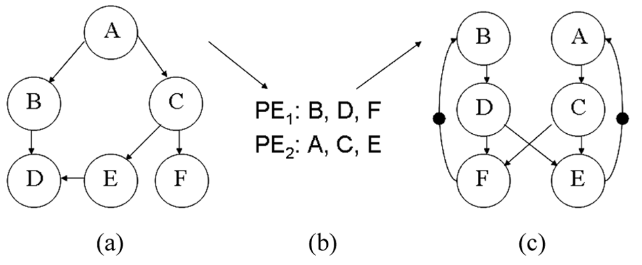
where, $\mathrm{start}(v, k)$ and $\mathrm{end}(v, k)$, respectively, represent the time at which invocation k of actor v begins execution and completes execution, and $\mathrm{delay}(e)$ represents the delay associated with edge e.

Once we construct G_s for a system, we use the Maximum Cycle Mean (MCM) of the graph for performance analysis. The MCM is defined by

$$MCM(G_s) = \max_{\text{Cycle C in G}_s} \left\{ \frac{\sum_{v \in C} t(v)}{\mathrm{Delay}(C)} \right\}, \tag{2}$$

where, $\mathrm{Delay}(C)$ denotes the sum of the edge delays over all edges in cycle C. The MCM is used in a wide variety of analysis problems, and a variety of techniques have been developed for its efficient computation (e.g., see Dasdan and Gupta (1998)). Examples of an application graph, a corresponding self-timed schedule, and the resulting synchronisation graph are illustrated in Figure 6.

Figure 6 An example of (a) an application graph; (b) an associated self-timed schedule and (c) the synchronisation graph that results from this schedule



5 Architectural exploration

5.1 Target architecture

Along with exploring two different applications in this work, we targeted three different architectures to demonstrate the correctness, flexibility, and robustness of our proposed methodology. The face detection system was implemented on a reconfigurable system on chip, while the 3D object tracking system was implemented on a multiprocessor system and a PDSP-based system. The multiprocessor-based implementation represents a pure software implementation on a general-purpose platform, while the PDSP-based system and the reconfigurable SoC, which includes both hardware and software resources, are more strongly representative of the embedded image processing domain.

Our methodology works for all of these representative architectures. In the following sections, brief descriptions are provided for each of these types of target platforms.

5.1.1 Reconfigurable system on chip

An example of such a system is Xilinx's ML310 board, which contains a Virtex II Pro Field Programmable Gate Array (FPGA) device. This board supports both hardware (FPGA-based) and software (PowerPC-based) design. It also includes on-chip and off-chip memory resources. Access to the off-chip memory is assumed to be through a shared bus. On-chip memory access can be performed through a shared bus or via DMA.

5.1.2 Multiprocessor system

We have also used a shared-memory, multiprocessor system, more generally known as a scalable shared memory multiprocessor. In a shared-memory system, every processor has direct access to the memory of every other processor, i.e., it can directly load or store any shared address. In addition, pieces of memory that are private to the processor can also be present and have to be explicitly specified by the user.

5.1.3 PDSP system

Programmable Digital Signal Processors (PDSPs) are specialised microprocessor platforms for implementation of signal processing applications. PDSPs contain special instructions and architectural features that are specific to signal processing applications, which yields higher efficiency for signal processing compared to other types of microprocessors. Examples of PDSPs include the TMS320c64xx series from Texas Instruments, TigerSHARC from Analog Devices, and ZSP500 from LSI Logic.

5.2 Profiling

Software profiling is usually the first step to any performance optimisation approach. Profiling gives us information about the execution times of different program modules and hence where the most time is spent. Since the applications targeted in this work are computation intensive, one of our primary concerns is performance efficiency. It is therefore necessary to identify the critical sections of the code. These critical sections may be moved to hardware for embedded architectures or parallelised using threads in multiprocessor systems.

In this work, we have used two profilers. For profiling the face detection algorithm, whose software implementation is in C, the FLAT profiler was used. The FLAT profiler provides loop/function level profile information (Suresh et al., 2003). The FLAT profiler identifies the mask correlation module (see Table 1) as the candidate core – a core is defined in this context as the set of all loops whose execution cost is higher than a threshold value – for optimisation. The output of FLAT is summarised in Table 1.

Table 1 Output from the FLAT profiler for the face detection application

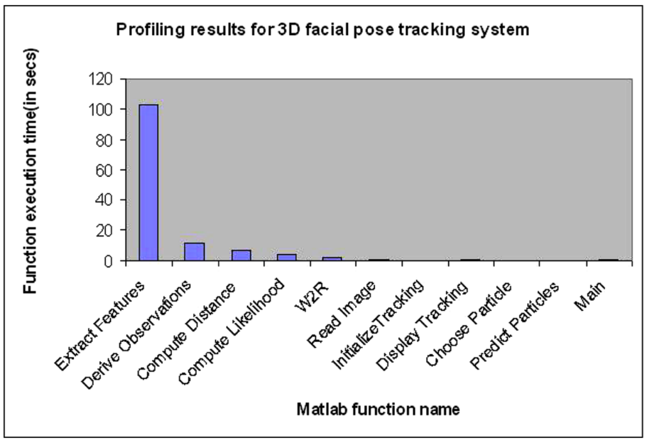
Loop name	*Frequency*	*Loop size*	*Total ins.* (10^6)	*%Exec*
Program	1	96,912	87,053	100.0
Mask correlation	56,392	600	83,978	96.47

For the second application – the 3D facial pose tracking system – the software implementation is in MATLAB, and the MATLAB profiler was used. The MATLAB profiler provides information about individual function execution times along with the total execution times and the number of calls made to each function. The profiling results for individual execution time are shown in Figure 7. In this figure, the execution time is the total time spent in the function for a single execution of the program.

5.3 Architecture modelling

Both of the applications explored in this work are characterised by computation intensive operations that are inherently parallelisable. Profiling identified the functions/operations that are critical and computationally most expensive. If possible, these operations should be moved to specialised processing hardware elements or parallelised (e.g., using threads in a multiprocessor system). In the next two sections, the high level architectural model for each application is described.

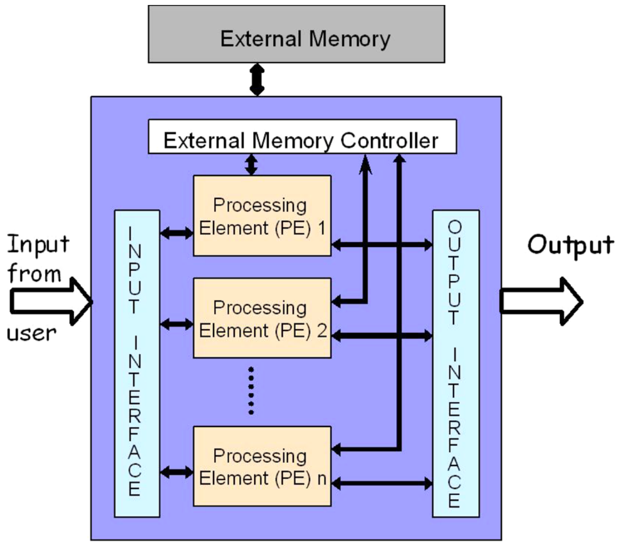
Figure 7 MATLAB profiler result for the 3D facial pose tracking system (see online version for colours)



5.3.1 *Architecture modelling for face detection system*

Figure 8 shows the high-level architecture model for the face detection system. From the results of profiling, one may clearly conclude that the mask correlation function is the most computationally expensive function. But this function is inherently parallelisable as the correlation of each mask with the image is independent of the others.

Figure 8 Face detection system architecture (see online version for colours)


External Memory
External Memory Controller
Processing Element (PE) 1
Processing Element (PE) 2
Processing Element (PE) n
INPUT INTERFACE
OUTPUT INTERFACE
Input from user
Output

Based on these conclusions, we derived a system model that consists of multiple Processing Elements (PEs) that can concurrently execute multiple instances of the correlation function and thereby process masks simultaneously. From the face detection algorithm, shown in Figure 2, a large set of masks is created with which the image frame is compared. This mask set can be created offline and moved to external memory. Each PE reads masks with which it performs the correlation operation from the external memory one at a time. Also no two PE uses the same mask. Thus, each PE requires at least one mask available on chip. To facilitate faster processing, we stored two masks per PE at a time; when a PE operates on the first mask the second one is read. Thus, this model requires us to have multiple masks and copies of the frame (for concurrent access by the PEs) available on the chip, e.g., three PEs require three frame copies present on chip.

To minimise the rate of memory accesses and hence power consumption, we partitioned the image into stripes and processed the image one stripe at a time where a stripe is defined to be the minimum size of the image that can be processed on one pass. We run our mask set on a given stripe of the image and find the maximum correlation value, repeat the process for the next stripe, and continue in this manner until we have exhausted all the stripes, and hence the image.

For a set of N masks and n PEs (i.e., n masks can be processed simultaneously), it will take $m = \lceil N/n \rceil$ processing passes to cover all masks for a single stripe. Here, $\lceil x \rceil$ denotes the greatest integer that is greater than or equal to the rational number x.

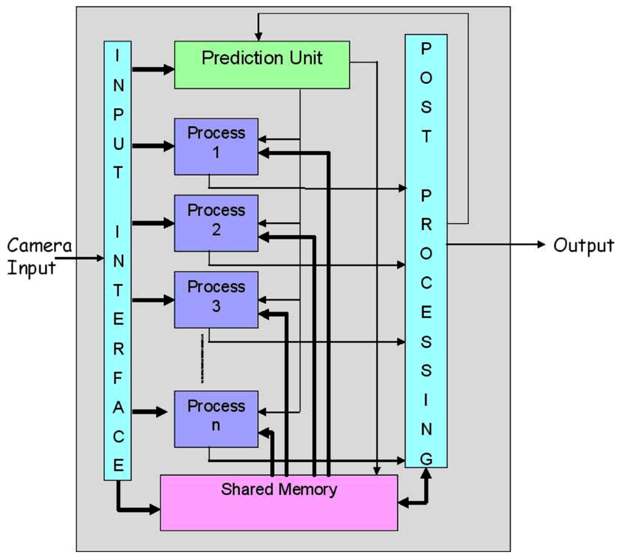
5.3.2 *Architecture modelling for 3D facial pose tracking system*

In this system, there was no need for external memory to store offline data, in contrast to the face detection system. All data is generated online and subsequently updated. Based on the predicted translation and rotation vectors, the different positions of the cylindrical grid in the current image frame are generated. The prediction for the current frame is not used beyond the current frame and hence may be discarded after the computations of the frame is over. Based on the profiling results, it may be observed that the functions derive observations and extract features together consume most of the total execution time. These functions calculate the positions of the cylindrical grid for the different prediction vectors (or particles) and the corresponding feature vectors. Since the calculations corresponding to one particle is independent from that of the rest, they may be done in parallel. Also, this system is a feedback system. The particles are updated based on observations made on the current image frame. Based on these observations, the architecture model shown in Figure 9 was derived.

Each process reads the current image and processes it for a given particle. This requires either the presence of multiple copies of the image frame or simultaneous multiple access to the frame. Note that the parallel processes only read the image frames and do not write to them. When all the processes have finished their operations, the results are collected and processed to get the final result; only then a write to the image frame is done. Thus, a shared-memory multiprocessor system is well-suited for this application.

The input interface reads the image and writes it to the shared memory, and it also performs initialisation. The prediction unit makes predictions for the translation and rotation vectors based on the best particles from the previous frame, i.e., it predicts the particles, which are then read by the processes. The post-processing unit collects the results from all of the processes, outputs the best prediction as the result, and performs a write to the image to mark the best predicted cylindrical grid.

Figure 9 3D facial pose tracking system architecture (see online version for colours)



6 Synchronisation graph modelling

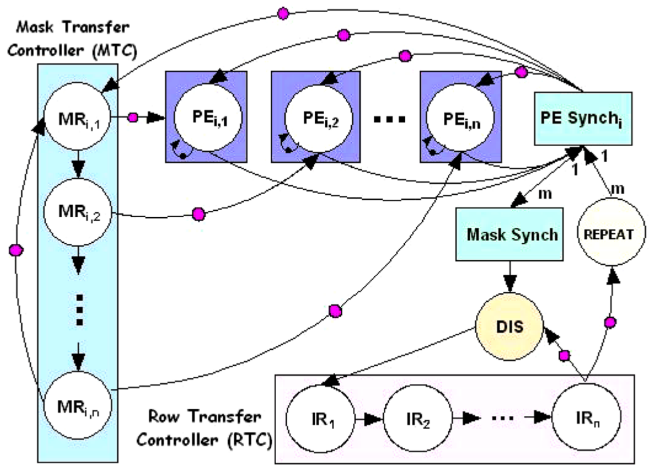
In this section, we present the multirate synchronisation graph modelling technique as a performance analysis tool. This modelling technique combines the characteristics of the self-timed scheduling strategy and the ordered-transaction strategy, and applies multirate dataflow representations for concise representation of architectural resources that operate at different rates. We illustrate the methodology by applying the technique to the two applications discussed earlier.

6.1 *Multirate synchronisation graph modelling for face detection system*

Figure 10 shows the implementation, transaction/execution, order and synchronisation model of our system. In this, n is the no. of PEs, N is the total number of masks and $m = \lceil N/n \rceil$. The various actors in this figure are explained below:

- $MR_{i,j}$ (*Mask read*): This actor represents the reading of mask i to processing element j, where i varies from 1 to m and j varies from 1 to n. This process takes $t(MR)$ time units.
- MTC (*Mask Transfer Controller*): The masks are stored in the external memory, the MTC controls the reading to the dedicated block RAMs (BRAMs) for each PE. The MTC conducts mask transfers one-at-a-time according to a repeating, predetermined sequence in a fashion analogous to that of the ordered transaction strategy.
- $PE_{i,j}$ (*Processing Element*): This actor represents the processing of the ith mask by PE_j, which takes $t(PE)$ time units.
- DIS (*Downsampled Image Source*): This actor represents the downsampling of an image stripe whose execution time is $t(DIS)$ time units.
- IR (*Image Read*): This actor represents the reading of the downsampled stripe into the BRAMs one row at a time with execution time of $t(IR)$ time units.
- $REPEAT$: This actor is a conceptual actor that ensures that exactly m mask sets are processed for each new row of image data. No data actually needs to be replicated by the $REPEAT$ actor; the required functionality can be achieved through simple, low-overhead synchronisation and buffer management methods.
- $PESynch_i$: This actor represents the synchronisation unit that synchronises the start of the ith iteration of the PEs with the reading of mask set $(i + 1)$. This unit receives data from the PEs and the $REPEAT$ actor. The messages from the PEs confirm that they have completed the processing of one mask. These messages are sent at the end of each iteration. The production rate of m and consumption rate of 1 (shown on the $(REPEAT, PESynch_i)$ edge) indicate that the $PESynch$ unit has to execute m times before the $REPEAT$ actor is invoked again.
- $MaskSynch$: This actor is again a conceptual actor, and need not be mapped directly to hardware. It represents the synchronisation between m executions of the $PESynch$ actor and the corresponding execution of the DIS actor. Therefore, for each execution of the DIS actor, the $PESynch$ executes m times.

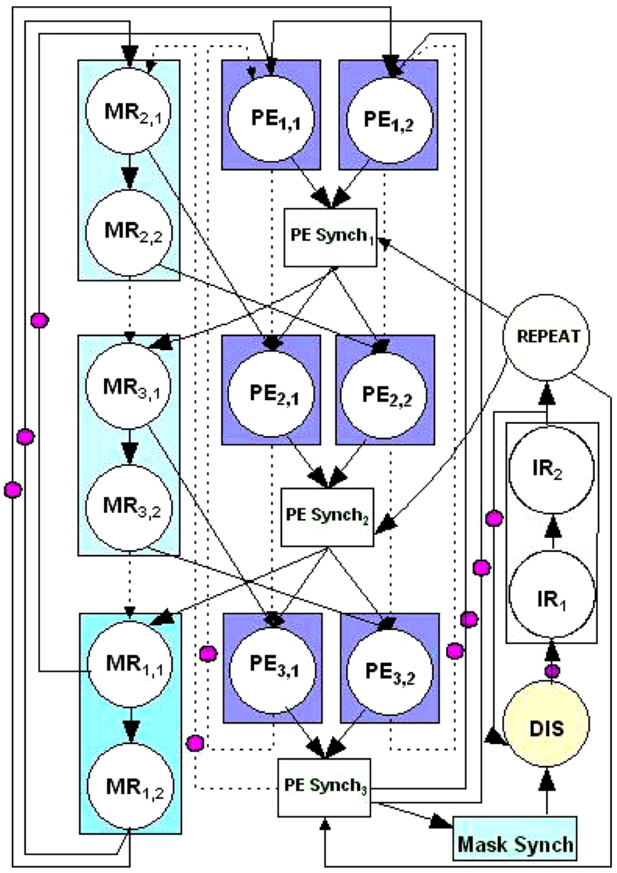
Figure 10 Multirate synchronisation graph for face detection system (see online version for colours)



Unlike conventional ordered transaction implementation, however, a transaction ordering approach is not used in our design for all dataflow communications in the enclosing system. We use a self-timed model to coordinate interaction between the MTC and the PE cluster. At the start of reading a new set of masks, the controller must synchronise with the PEs to make sure that they have finished processing of the current masks. This synchronisation process is represented as the edge directed from the $PESynch$ actor to the starting actor of the MTC block. The edge delay connecting $MR_{i,n}$ to $MR_{i,1}$ represents the initially-available mask data from pre-loading the first set of masks for a new image to their associated BRAMs.

A properly-constructed ('consistent'), multirate SDF graph unfolds unambiguously into a Homogeneous SDF (HSDF) graph (Lee and Messerschmitt, 1987), which in general leads to an expansion of the dataflow representation. An example of an SDF-to-HSDF transformation is given in Figure 11 for $m = 3$ and $n = 2$. Currently, this expansion is done manually. In general the SDF to HSDF transformation can be automated. This follows from developments in Lee and Messerschmitt (1987). Automating this and other aspects of our proposed design flow are useful directions for further work. For performance analysis, it is necessary to reason in terms of directed cycles in the HSDF representation. This is discussed further in Section 6.3.

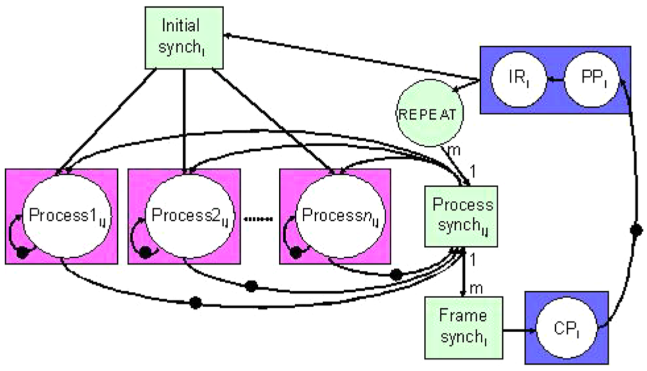
Figure 11 Example of an unfolded HSDF graph for face detection system, $m = 3$, $n = 2$ (see online version for colours)



6.2 *Multirate synchronisation graph modelling for 3D facial pose tracking system*

The multirate synchronisation graph for the 3D facial pose tracking system that we implemented on the shared memory multiprocessor system is illustrated in Figure 12. Here, n is the total no. of processes, N is the number of particles and $m = \lceil N/n \rceil$. The actors in this graph are described below:

Figure 12 Multirate synchronisation graph for 3D facial pose tracking system (see online version for colours)



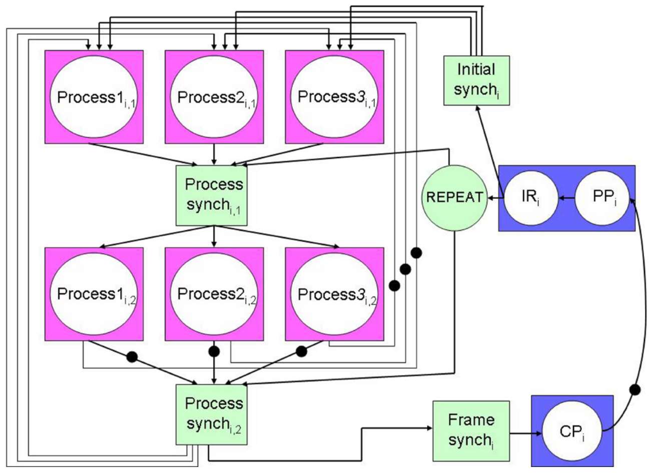
- $Processn_{i,j}$ (*Process*): This actor represents the processing of the ith frame by the nth process; j varies from 1 to m, where $m = \lceil N/n \rceil$ is the number of times a process executes for a given frame; N is the total number of particles; and n is the number of processes. The execution time of $Processn_{i,j}$ is denoted by $t(Process)$ units.
- CP_i (*Choose Particle*): The actor CP_i chooses p best particles based on their likelihood values and resamples them to form N particles for the ith image frame and takes $t(CP)$ time units.
- PP_i (*Predict Particle*): The actor PP_i predicts particles i.e., the translation and rotation vectors from the N particles chosen by actor CP_{i-1} and takes $t(PP)$ time units.
- IR_i (*Image Read*): This actor represents the reading of the ith frame. The execution time is $t(IR)$ time units.
- $ProcessSynch_{i,j}$: This actor performs synchronisation of the processes. Each process executes independently of the others, and need to synchronise only at the end of their execution. The execution time of this actor is $t(ProcessSynch)$.
- $REPEAT$: The $REPEAT$ actor is, as explained earlier, a conceptual actor that ensures m executions of each process. The execution time of the $REPEAT$ actor is given by $t(REPEAT)$.
- $InitialSynch_i$ and $FrameSynch_i$: These actors represent the synchronisation at the beginning and end of the ith frame, respectively, and take

$t(InitialSynch)$ and $t(FrameSynch)$ units of time to execute. The production and consumption rates of m and 1 on the $(REPEAT, ProcessSynch_{i,j})$ edge, and 1 and m on the $(ProcessSynch_{i,j}, FrameSynch_i)$ edge ensure that the actor $ProcessSynch_{i,j}$ executes m times and thereby all the $Process$ actors execute m times.

In this model, the coordination between the processes and the rest of the actors as well as the manner in which a process executes for different iterations is self-timed, whereas the execution manner of the rest of the actors (CP_i, PP_i, IR_i) follow the ordered transaction strategy. The execution starts with the PP_i actor predicting particles for the ith frame followed by reading of the image by the IR_i actor. The $Process$ actors start their executions after receiving inputs from the $ProcessSynch_{i,j}$ actor and the $InitialSynch_i$ actor which indicate that the processes have finished their previous executions and that the image has been read along with particles predicted. Once all the processes have finished their executions they send a synchronisation signal to the $ProcessSynch_{i,j}$ actor. This iterates m times, after which the $ProcessSynch_{i,j}$ actor sends a signal to the $FrameSynch_i$ actor, which in turn sends a signal to the CP_i actor, and the execution thread moves to the next frame. The delays on the edges from the $Process$ actors to the $ProcessSynch_{i,j}$ actor and the CP_i to PP_i actor are required for proper initialisation. The unfolded HSDF graph for $m=2, n=3$, and one frame is shown in Figure 13.

Our PDSP implementation involved the original system without application of the architecture model and hence did not involve any parallelisation. This is because we employ a uniprocessor PDSP target. The resulting multirate synchronisation graph and the corresponding HSDF expansion are trivial, involving only one cycle each.

Figure 13 Example of an unfolded HSDF graph for 3D facial pose tracking system, $m = 2$, $n = 3$ (see online version for colours)



6.3 Performance analysis

We use the modelling approach described above to understand and explore performance issues. From the models, it can be observed that the HSDF cycles (cyclic paths in the HSDF graph) can be decomposed into a limited set of classes, where each class exhibits very similar patterns of cyclic paths. Also, many cycles in the graph are isomorphic, where by isomorphic cycles we mean those in which the vertices and edges can be placed in one-to-one correspondence with one another so that corresponding vertices have the same execution times and corresponding edges have the same delays. Understanding these isomorphic relationships allows us to greatly reduce the number of MCM computations that actually need to be considered. Similarly, understanding the patterns of variation across certain sets of similarly- structured cycles makes it easy to extract the critical cycles (the cycles with maximum MCM) from these sets.

Table 2 Cycle mean expressions for the multirate synchronisation graph for the face detection system

No.	*Description of class of cycle*	*Cycle*	*Cycle mean expression*
1	Reading of masks	$MR_{2,1} \rightarrow MR_{2,2} \cdots \rightarrow MR_{2,n} \rightarrow MR_{3,1} \cdots \rightarrow MR_{1,1} \cdots \rightarrow MR_{1,n} \rightarrow MR_{2,1}$	$\frac{m \times n \times t(MR)}{D_0}$
2	Reading of image rows	$DIS \rightarrow IR_i \rightarrow DIS$	$\frac{t(DIS) + n \times t(IR)}{D_1 + D_2}$
3	Reading of image rows with synchronisation	$DIS \rightarrow IR_i \rightarrow REPEAT \rightarrow PESynch_m \rightarrow MaskSynch \rightarrow DIS$	$\frac{t(DIS) + n \times t(IR) + 3}{D_1}$
4	PEs synchronisation	$PESynch_m \rightarrow PE_{1,i} \rightarrow PE_{2,i} \cdots \rightarrow PE_{m,i} \rightarrow PESynch_m$ or $PESynch_m \rightarrow PE_{1,i} \rightarrow PESynch_1 \rightarrow PE_{2,i} \cdots PESynch_m$	$\frac{m \times (t(PE) + 1)}{D_1}$
5	Synchronisation of PEs and reading of masks	$PESynch_m \rightarrow MR_{2,1} \cdots \rightarrow MR_{3,1} \cdots \rightarrow MR_{3,n} \cdots \rightarrow MR_{1,1} \cdots \rightarrow MR_{1,n} \rightarrow PE_{1,i} \rightarrow PE_{2,i} \cdots \rightarrow PESynch_m$ or $PESynch_m \rightarrow MR_{2,1} \cdots \rightarrow MR_{3,1} \cdots \rightarrow MR_{3,n} \cdots \rightarrow MR_{1,1} \cdots \rightarrow MR_{1,n} \rightarrow PE_{1,i} \rightarrow PESynch_1 \rightarrow PE_{2,i} \cdots \rightarrow PESynch_m$	$\frac{m \times n \times t(MR) + m \times (t(PE) + 1)}{D_3 + D_4}$

Table 3 Cycle mean expressions for the multirate synchronisation graph for 3D facial pose tracking system

No.	*Description of class of cycle*	*Cycle*	*Cycle mean expression*
1	Process only	$Processk_{i,1} \rightarrow Processk_{i,2} \cdots \rightarrow Processk_{i,n} \rightarrow Processk_{i,1}$	$\frac{m \times t(Process)}{D_2}$
2	Process with synchronisation	$Processk_{i,1} \rightarrow ProcessSynch_{i,1} \rightarrow \cdots \rightarrow Processk_{i,m} \rightarrow ProcessSynch_{i,m} \rightarrow Processk_{i,1}$	$\frac{m \times t(Process)}{D_1}$
3	Processing without process actors	$PP_i \rightarrow IR_i \rightarrow REPEAT \rightarrow ProcessSynch_{i,m} \rightarrow FrameSynch_i \rightarrow CP_i \rightarrow PP_i$	$\frac{t(PP) + t(IR) + t(FrameSynch) + t(CP)}{D_0}$
4	Processing with process actors	$PP_i \rightarrow IR_i \rightarrow InitialSynch_i Processk_{i,1} \rightarrow ProcessSynch_{i,1} \rightarrow \cdots \rightarrow Processk_{i,m} \rightarrow ProcessSynch_{i,m} \rightarrow FrameSynch_i \rightarrow CP_i \rightarrow PP_i$	$\frac{t(PP) + t(IR) + t(InitialSynch) + m \times t(Process) + t(FrameSynch) + t(CP)}{D_0}$

For example, in Figure 11, there are many cycles involving PE_is and $PESynch_i$s, and an exhaustive analysis leads to the identification of all of them. However, since we are mainly interested in the MCM value, we look for longer cycles and hence consider the cycle that involves all the $PESynch_i$s and PE_is. Tables 2 and 3 tabulate several of the different classes of cycles, along with a description and the MCM for each class for the two examples discussed above. In the tables, "description of class of cycle" states the operation that is done by the actors of the cycle, and 'cycle' presents the cycle by traversing the actors of the cycle. The MCM is obtained by extracting the critical cycle in each class, i.e., the cycle yielding the highest value of MCM.

7 Experimental results

To demonstrate the accuracy and versatility of our proposed methodology, we evaluated our proposed design on various platforms. The face detection algorithm was implemented on the ML310 board from Xilinx. Key features of this board include 30,000 logic cells, 2,400 Kb of BRAM (36 of 18Kb BRAMs), dual PPC405 processors, and 256 MB DDR DIMM (external memory).

The 3D facial pose tracking algorithm was implemented on two platforms. The serial version was evaluated on the Texas Instrument TMS320c64xx processor series by simulation using The Texas Instruments Code Composer Studio tool-set, while the parallel version was run on a Sunfire 6800 containing 24 SUN UltraSparc III machines running at 750 MHz using 72 GB of RAM.

Below we present the results and design space exploration outcomes for the two applications.

7.1 Design space exploration for face detection system

As can be seen from Table 2, the overall system performance is a function of $m, n, t(PE), t(MR)$ and $t(DIS)$. In this work, we assumed that the number of masks and the mask sizes are fixed, where, as mentioned before, the face detection algorithm is shape-based and the face is modelled as an ellipse. For operational value, the implementation should be able to handle variability in size of the faces as that information is not usually available a-priori. We handled this by creating several elliptical masks of varying sizes. This entails building of a large mask set and consequently a large number of correlation computations.

We make the following valid assumptions to reduce the mask set size.

- The number of masks required is bounded by the possible ellipticity of faces and by the size of the image.
- Our target application is a smart-camera based vision system: given a particular smart-camera, the size of the images shot by that camera can be assumed to be fixed.

Under these assumptions, and above mask parameters, the algorithm performed robustly for faces of very different sizes.

Given the above justification, it can be assumed that $t(MR)$ and m are constant (the mask size considered is 65×81 and the size of the mask set to 93). $t(PE)$ is a function of several parameters but we considered the following parameters only:

- degree of fine grain parallelism (i.e., how many simultaneous operations can be performed within each PE)
- image size
- the resolution (number of rows/columns considered) at which the image is compared with the mask.

$t(DIS)$ is a function of the frame sizes. To keep the design space manageable, we fixed the frame size (frame size $= 240 \times 320$, stripe size $= 65 \times 160$), and varied the number of PEs (i.e., n), the steps – the granularity at which the image is correlated with a mask, and the fine-grain parallelism (explained later) up to the permissible HW limits. The execution times were obtained by multiplying the number of execution cycles for each node by the inverse of the clock frequency, which is 125 MHz for the given board. The delays were given by the number of cycles required for the initial values to load from the source actor to the destination actor in the synchronisation graph. From this, we obtained the cycle means for the stated classes of cycles, which yielded the throughput as the minimum inverse of the cycle mean over all possible cycles. To obtain a satisfactory throughput, the parameters had to be varied without affecting accuracy. We observed that the throughput could not be improved by merely increasing the number of PEs. The number of columns being skipped could be increased as well, but only to a certain extent, so that the accuracy of the face detection was not affected. Also, the actual throughput was a function of the number of image stripes present, and hence was affected by the resolution of the camera.

The number of PEs that may be implemented was limited by the area constraints. The board has 136 BRAMs present of which few were allocated for the use of other modules such as the down-sampling unit, the PowerPC, and so on. Each PE required 8 BRAMs, which set an upper bound of number of PEs to 15. Also, parallelisation was possible within each PE: since the multiplications required for the calculation of each correlation value are independent of each other, at the same instant more than one multiplication could be performed. The number of multipliers available on the board limited this parallelisation: there are 136 multipliers on board, which limited the number of PEs to 13. The I/O buffers present on the board also imposed serious restrictions on the number of PEs. Since each PE communicates with the down-sampling unit, the BRAMs, the external DDR SDRAM memory controller, and the output interface, it has a significant number of I/O ports – this limited the maximum number of PEs that could be practically implemented to 6.

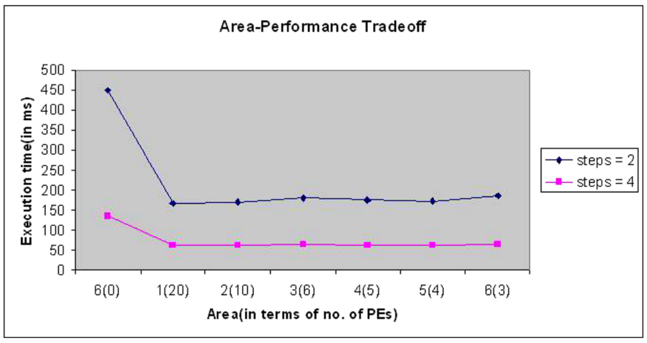
We obtained execution times with parameters bounded by the above analysis. The results are presented in Table 4. Here, n is the number of PEs, 'degree of parallelism' is the number of additional multiplications done simultaneously in each PE, and 'steps' is as explained earlier. Thus, there are two entries in the table for $n = 6$, in the first one, no additional multiplications were done, while in the second one, three additional multiplications were done in parallel. There are differences in the estimated and experimental values in the table. In our model, for simplicity we assumed a small constant value for all synchronisation actors (*PESynch*), which is a major reason for the difference.

In Section 7.3, we give the fidelity analysis for this system. This analysis shows that even though there are differences between estimated and experimental results, the estimation provides for reliable *comparison* between different design points, which is the main objective during design space exploration. We also present the area-performance trade-off curve obtained from the multirate synchronisation graph in Figure 14, where we quantify area by the number of PEs and performance by the execution time.

Table 4 Execution times for 1 frame for different design parameters

		Estimation (ms)		*Experimental* (ms)	
n	*Degree of parallelism*	*Steps* = 2	*Steps* = 4	*Steps* = 2	*Steps* = 4
6	0	451	136	697	205
1	20	168	62	227	79
2	10	170	62	227	79
3	6	180	65	249	85
4	5	176	63	227	79
5	4	173	63	227	79
6	3	184	66	249	85

Figure 14 Area-performance trade-off results for face detection system. Area is in terms of no. of PEs and degree of parallelism within each PE. Performance is measured by execution time (see online version for colours)



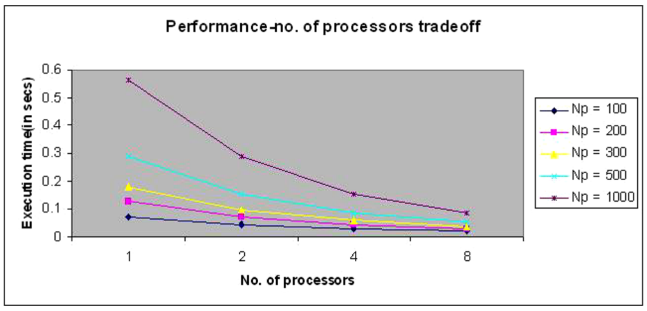
The maximum frame rate of applications in security and video surveillance toward which this work is targeted is 30 frames per second (fps). With the current board and available hardware resources, the implementation achieves a maximum frame rate of 12 fps. This entails the discarding of every two out of three frames at most, which can be tolerated for such applications. Alternatively, the models and design space exploration techniques employed in this paper can be applied on more powerful boards to explore implementations that approach or achieve the 30 fps target without discarding any frames.

7.2 3D facial pose tracking system

7.2.1 Design space exploration for multiprocessor system

The design space for this system involved two main parameters – the number of processes used and number of particles employed. The number of particles determines the accuracy and robustness of the tracking system, while the number of processes controls the execution time. The minimum number of particles required is 100, although for 50 particles, the system performs reasonably well. The execution time increases in proportion with increase in the number of particles. Thus, though increase in the number of particles implies more accurate tracking, the increase cannot be arbitrary and an upper limit has to be imposed to meet frame rate requirements. We observed that though there was significant improvement from 100 particles to 1000, the improvement started saturating after 1000. Hence, we set the upper limit to 1000 particles. The trade-off between the number of processes and execution times for different values of the number of particles is shown in Figure 15.

Figure 15 Performance number of processors trade-off results for 3D facial pose tracking system. Performance is measured by execution time (see online version for colours)



The cycle mean values for the important cycles are presented in Table 3. From this table, one may observe that the most critical cycle is the fourth one and it should yield the MCM value. Further calculations verified this observation. We assumed that execution time of the $ProcessSynch$ actor is negligible compared to the $Process$ actor and ignored it in MCM calculations. Also since the $REPEAT$ actor is just a conceptual actor, its execution time was ignored.

Table 5 presents the comparison of execution time results for one frame that are predicted from the synchronisation graph model using *MCM* values and the actual experimental values. N_p is the number of particles, and n is the number of processes. The estimated and experimental values match very closely for lower numbers of processes, but for larger numbers of processes, the estimations are not as good. The reason behind this is the assumption of negligible execution time of the $ProcessSynch$ actor, which does not remain valid for larger number of processes. For large numbers of processes, there is a significant overhead for scheduling the processes as well as associated synchronisations. This overhead was not accounted for in the estimations. Accounting accurately for such overhead is a useful direction for further study.

Note that when N is not a multiple of n, during the mth ($m = \lceil N/n \rceil$) iteration, some of the processes remain in an idle waiting state. We used dynamic scheduling in our implementation. Hence, the runtime system determined the scheduling of the processes, and thereby which process remain idle during the last iteration for a given frame.

The maximum frame rate obtained for this application is 33 fps for 100 particles using eight process, which is higher than the required 30 fps typical for such applications. The implementation achieves the required frame rate to a good approximation for most of the cases, which illustrates the efficiency of the architecture model.

The OpenMP parallel programming model was used for this implementation (Dagum and Menon, 1998). OpenMP is a language extension in C, C++ and FORTRAN. It provides scope for private and shared variables, as well as directives for parallelising loops and regions. The user inserts the compiler directives in the code where parallelisation is to be done, and the compiler reads these directives and converts the program to a multithreaded implementation.

7.2.2 *Design space exploration for PDSP implementation*

The PDSP system simulation was done using Code Composer Studio (version 2) from Texas Instruments. The TMS320c64xx processor series was used with an instruction cycle time of 40 ns. The sizes of the RAMs used were 320 KB and 1.6 MB for instruction and data, respectively.

The main objective behind this experimentation was to study the implications of embedding such an application through a PDSP platform. We observed that the memory requirement is higher than other typical PDSP applications. Also, without parallelisation, such implementations are far from meeting the required frame rate, which became obvious from the high execution times observed in this implementation.

Synchronisation graph modelling – though trivial in this case because the target was a uniprocessor platform – was carried out to estimate the execution times.

Table 5 Execution times for 1 frame for different design parameters for 3D facial pose tracking system implemented on shared memory multiprocessor system

	Estimation (ms)				*Experimental* (ms)			
N_p	$n = 1$	$n = 2$	$n = 4$	$n = 8$	$n = 1$	$n = 2$	$n = 4$	$n = 8$
100	0.072	0.045	0.031	0.024	0.072	0.048	0.037	0.030
200	0.127	0.072	0.045	0.031	0.135	0.080	0.059	0.048
300	0.181	0.099	0.058	0.038	0.188	0.120	0.080	0.060
500	0.291	0.154	0.086	0.052	0.310	0.186	0.117	0.095
1000	0.564	0.291	0.154	0.086	0.611	0.346	0.229	0.172

The results are shown in Table 6. The estimated results match very well with the experimental results, yielding a perfect fidelity of 1.

Table 6 Execution times for 1 frame for different design parameters for 3D facial pose tracking system implemented on a PDSP

No. of particles	*Estimation* (s)	*Experimental* (s)
50	4.17	4.23
100	7.09	7.21
150	10.02	10.2
200	12.94	13.21
250	15.87	16.21

7.3 *Fidelity analysis*

In this section, we present fidelity calculations of our performance estimations as the design parameters are varied. The fidelity provides a measure of the accuracy with which an estimation technique provides comparisons between different design points. The fidelity is defined by the following expression.

$$\text{Fidelity} = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} f_{ij}, \tag{3}$$

where,

$$f_{ij} = \begin{cases} 1, & \text{if } (\text{sign}(S_i - S_j) = \text{sign}(M_i - M_j)); \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Here, the S_is denote the simulated execution times, and the M_is are the corresponding estimates from the MCM expression. For the face detection system for a fixed frame size and the 14 design points presented in Table 4, we obtain a fidelity of 0.868. For the 3D facial pose tracking system, the fidelity obtained is 0.921. In spite of the assumptions that resulted in differences in the estimated and experimental results for both implementations, a high value of fidelity is obtained. This demonstrates the accuracy and robustness of our modelling technique.

8 Conclusions

In this paper, we have presented a new methodology for architectural design, modelling and exploration for smart camera applications on embedded systems. Our methodology generalises the synchronisation graph modelling technique and applies it as a conceptual tool for design. The technique considers synchronisation graph modelling for multidimensional signals, and integrates the methods of self-timed and ordered-transaction scheduling. We have shown the practical utility of our methodology by applying it to two image processing applications – face detection and 3D facial pose tracking – and a variety of target platforms. We have emphasised how the methodology helps designers to evaluate multiple design alternatives, and explore the associated trade-offs in an efficient and intuitive manner.

For the face detection system, we were able to obtain a maximum frame rate of 12 fps for the given ML310 target board. We conclude that increased resources or a tolerance for periodically dropping frames is required to achieve the target frame rate of 30 fps. For the 3D facial pose tracking system, the PDSP implementation showed that without parallelisation, reasonable performance cannot be achieved. The multiprocessor implementation gave, in exchange for significantly increased cost, very good performance results. Our multiprocessor implementation achieved the target frame rate of 30 fps for many cases, and in the best case, gave an overachieving frame rate of 33 fps.

Important directions for future work include better modelling of synchronisation actors for more accurate performance estimations. Also, exploring other techniques for hardware/software design such as partitioned co-synthesis are interesting avenues for future experiments.

Acknowledgement

This research was supported by grant number 0325119 from the US National Science Foundation.

References

Auguin, M., Capella, L., Cuesta, F. and Gresset, E. (2001) 'CODEF: a system level design space exploration tool', *IEEE International Conference on Acoustics, Speech, and Signal Processing*, May, Vol. 2, pp.1145–1148.

Aggarwal, G., Veeraraghavan, A. and Chellappa, R. (2005) '3D facial pose tracking in uncalibrated videos', *International Conference on Pattern Recognition and Machine Intelligence (PReMI)*, December, pp.515–520.

Dagum, L. and Menon, R. (1998) 'OpenMP: an industry-standard API for shared-memory programming', *IEEE Computational Science and Engineering*, Vol. 5, No. 1, pp.46–55.

Dasdan, A. and Gupta, R.K. (1998) 'Faster maximum and minimum mean cycle algorithms for system performance analysis', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, No. 10, pp.889–899.

Karkowski, I. and Corporaal, H. (1998) 'Design space exploration algorithm for heterogeneous multi-processor embedded system design', *Proceedings of Design Automation Conference*, June, pp.82–87.

Khandelia, M. and Bhattacharyya, S.S. (2000) 'Contention-conscious transaction ordering in embedded multiprocessors', *Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors*, July, pp.276–285.

Kianzad, V., Saha, S., Schessman, J., Aggarwal, G., Bhattacharyya, S.S., Wolf, W. and Chellappa, R. (2005) 'An architectural level design methodology for embedded face detection', *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, New Jersey, September, pp.136–141.

Kwon, S., Lee, C., Kim, S., Yi, Y. and Ha, S. (2004) 'Fast design space exploration framework with an efficient performance estimation technique', *Proceedings of 2nd Workshop on Embedded Systems for Real-Time Multimedia*, pp.27–32.

Lee, E.A. and Messerschmitt, D.G. (1987) 'Static scheduling of synchronous dataflow programs for digital signal processing', *IEEE Transactions on Computers*, Vol. 36, No. 1, January, pp.24–35.

Miramond, B. and Delosme, J-M. (2005) 'Design space exploration for dynamically reconfigurable architectures', *Proceedings of DATE*, pp.366–371.

Moon, H., Chellappa, R. and Rosenfeld, A. (2002) 'Optimal edge-based shape detection', *IEEE Transaction on Image Processing*, Vol. 11, pp.1209–1227.

Peixoto, H.P. and Jacome, M.F. (1997) 'Algorithm and architecture-level design space exploration using hierarchical data flows', *Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors*, July, pp.272–282.

Sriram, S. and Lee, E.A. (1997) 'Determining the order of processor transactions in statically scheduled multiprocessors', *Journal of VLSI Signal Processing*, Vol. 15, No. 3, March, pp.207–220.

Sriram, S. and Bhattacharyya, S.S. (2000) *Embedded Multiprocessors: Scheduling and Synchronization*, Marcel Dekker, Inc., New York, NY.

Suresh, D.C., Najjar, W.A., Villareal, J., Stitt, G. and Vahid, F. (2003) 'Profiling tools for hardware/software partitioning of embedded applications', *Proceedings of ACM Symposium on Languages, Compilers and Tools for Embedded Systems (LCTES)*, June, pp.189–198.

Ziegenbein, D., Ernest, R., Richter, K., Teich, J. and Thiele, L. (1998) 'Combining multiple models of computation for scheduling and allocation', *Proceedings of Codes/CASHE*, pp.9–13.