Design Methodology for Pipelined Heterogeneous Multiprocessor System

Seng Lin Shee, Sri Parameswaran

School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia
National Information and Communications Technology Australia (NICTA), Sydney, Australia*

{senglin, sridevan}@cse.unsw.edu.au

Abstract

Multiprocessor SoC systems have led to the increasing use of parallel hardware along with the associated software. These approaches have included coprocessor, homogeneous processor (e.g. SMP) and application specific architectures (i.e. DSP, ASIC). ASIPs have emerged as a viable alternative to conventional processing entities (PEs) due to its configurability and programmability. In this work, we introduce a heterogeneous multi-processor system using ASIPs as processing entities in a pipeline configuration. A streaming application is taken and manually broken into a series of algorithmic stages (each of which make up a stage in a pipeline). We formulate the problem of mapping each algorithmic stage in the system to an ASIP configuration, and propose a heuristic to efficiently search the design space for a pipeline-based multi ASIP system.

We have implemented the proposed heterogeneous multiprocessor methodology using a commercial extensible processor (Xtensa LX from Tensilica Inc.). We have evaluated our system by creating two benchmarks (MP3 and JPEG encoders) which are mapped to our proposed design platform. Our multiprocessor design provided a performance improvement of at least 4.11X (JPEG) and 3.36X (MP3) compared to the single processor design. The minimum cost obtained through our heuristic was within 5.47% and 5.74% of the best possible values for JPEG and MP3 benchmarks respectively.

Categories and Subject Descriptors

C.1.3 [**Other Architecture Styles**]: Heterogeneous (hybrid) systems; C.4 [**Performance of Systems**]: Design studies

General Terms

Design, Experimentation, Performance

Keywords

Hardware/software partitioning, architecture, ASIP

1. Introduction

The miniaturization of transistors in processors has led to a greater increase in chip density and functionality. This results in smaller die size and lower power consumption, making it possible for more portable devices to be developed and manufactured. Chip manufacturers have seamlessly increased the capability and performance of such processor systems by taking advantage of these additional transistors. Conventional approaches include superscalar [26], SIMD and coprocessor [13] configurations.

Multiprocessor System-On-Chip (MPSoC) continues to gather momentum, primarily driven by the marketing of multi-core chips from Intel, IBM and AMD. The motivation for multi-core processors comes from the increasing difficulty of improving application performance by solely improving clock speed of systems, and the ease of verification. Thus, performance improvement could now be achieved by exploiting the parallelism within the algorithm.

MPSoC can be categorized into two domains; one, homogeneous; and two, heterogeneous multiprocessor system. Homogeneous systems consist of processors which are identical, as used in Symmetric Multiprocessing (SMP) systems. Heterogeneous processor systems utilize various types of processing entities to maximize performance while minimizing area and power consumption. Such systems may consist of a network of DSP, coprocessors and ASIC components fabricated onto the same silicon die. Each different component would be mapped and assigned to specific functions, thus executing multi-threaded applications. Such a system typically exhibits coarse grained parallelism.

In [21, 22], the authors give examples of such systems which provide a platform where multiple processing entities perform computation on different parts of the system concurrently. These systems can be considered as: one, single-ISA systems; and two, multi-ISA systems. Multi-ISA systems consist of totally different processors which can range from various DSP to CPU implementations [22, 27, 23]. Single-ISA heterogeneous systems [11] allow any application stage to be assigned and mapped to any core in the system with little reconfiguration and modification.

Existing approaches to heterogeneous processor architectures typically map critical regions of software into hardware (i.e. DSP, ASIC etc.). Each hardware component is optimized and suited to its particular mapped region to maximize performance. To increase efficiency and performance of critical systems, Application Specific Instruction-set Processors (ASIPs) [1, 2, 3, 5] have been introduced into such processor architectures. An ASIP's instruction set and its underlying architecture can be configured to a specific application in order to improve efficiency. ASIPs provide a good trade-off between efficiency and flexibility, as the same design can be re-used between different products variants and updated with little additional cost.

Multiprocessors utilizing extensible processors will make a significant contribution to the embedded system domain. In this work, we propose a system configured in a pipelined manner suited for data streaming applications. However, design space exploration of the possible pipeline configuration still remains an art form. While recent research has targeted mapping and selection for heterogeneous processors consisting of custom components and DSPs, we propose a methodology to configure a pipelined system which is properly balanced. Careful trade-off between the number of processors in the pipeline and extensible options of the processors have to be performed to maximize the overall performance. We would like to ensure that the increase in area due to a multiprocessor system is offset

*National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

by an even higher increase in performance. Our work allows the reuse of existing ASIP design exploration for each individual processor. We explore the design space to approach a near optimal configuration for a heterogeneous processor system, configured in a pipeline approach.

The rest of this paper is organized as follows: Section 2 gives a broad overview of the multiprocessor research thus far and Section 3 specifies the benchmark applications and platform in this work. Section 4 gives a holistic view of the design flow for a heterogeneous implementation ASIP system. The problem is formalize in Section 5 and an efficient heuristic that achieves near-optimal results is presented in Section 6. Section 7 reports the experimental methodology used in this work. In Section 8, we present our experimental results and analyze the performance of the multiprocessor architecture. Finally, in Section 9 the conclusions are summarized.

2. Related Work

Various heterogeneous multiprocessor systems have been implemented, primarily in the automotive real-time systems [7] and video / image encoding domain. The authors in [27] explored the use of a heterogeneous system in a real-time video and graphics streams management system, while in [32], the authors applied an adaptive job assignment scheme to perform data partitioning for a multiprocessor implementation of MPEG2 video encoding. A heterogeneous multiprocessor (five cores) for HDTV systems was developed in [10].

Gopalakrishnan et. al. [15] used heterogeneous systems in a different manner. Their work generalizes the approach started by Baruah [9] which replicates recurring tasks on multiple processing units to ensure a degree of fault tolerance. Maintaining replicas of a task at different processors ensures that single processor failures will be tolerated well.

A multicore system requires various communication schemes to provide the neccesary link between each core in the system. Kim et. al. [19] developed a new CDMA-based on-chip interconnection network using a Star NoC topology. To enable quick design of a multicore processor system and the evaluation of its interconnect system, Wieferink et. al [31] developed a methodology for retargetable MPSoC integration at the system level based on LISA [30] processor models and the SystemC [4] framework.

Single core applications utilize instruction level parallelism, enabled by pipelined processors. Hardware-software implementations can be further enhanced using pipeline schedulings [12]. Extending this scheme, multiprocessors are able to exploit task level parallelism by executing different task on separate cores simultaneously.

Several pipelining methods have been explored. Jeon et. al. [17] partitioned loops into several pipeline stages. The iterative algorithm proposed increased parallelism and reduced the hardware cost of the designed system. Kodaka et. al. [20] combined the both course grain and fine grain parallelism (which includes loop pipelining) using a single OSCAR chip multiprocessor. The work exploits course grain task, loop parallelism and instruction level parallelism using the OSCAR compiler. The OSCAR chip is comprised of several processor-elements (PEs) connected to local memory and shared memory, facilitating data transfer among processors.

A declustering technique for scheduling processes onto a multiprocessor system was proposed in [25]. This technique exposes parallelism instances in an Synchronous Data Flow (SDF) graph in order of importance and attains a cluster granularity that fits the characteristics of the architecture depending on the number of processors intended. However, the work mainly targets shared-memory multiprocessors and do not take into account a pipeline heterogeneous multiprocessor architecture which is used in our work.

Banarjee et. al. [8] incorporated heterogeneous digital signal processors with macro pipelining based scheduling. The technique utilized a signal flow graph (SFG) as a basis for partitioning. The work shows that heterogeneous multi-cores are able to improve the throughput rate several times that of the conventional homogeneous multiprocessor scheduling algorithms.

ASIPs in multiprocessor systems were first explored in [29]. The work proposed a methodology to simultaneously select custom instructions, assign and schedule application tasks on extensible processors. Our work complements this as we explore a higher level of abstraction to select different customized processors which would be suitable in a multiprocessor pipeline architecture.

Givargis et. al. [14] proposed a technique for efficiently exploring the power/performance design space of a parameterized system-on-chip (SoC) architecture to find all Pareto-optimal configurations. A directed graph was used to capture the interdependencies and algorithms that search the configuration space, incrementally and prune inferior configurations. In contrast to [14], we explore the design space of *pipelined* multiprocessor SoC configurations and the area-performance trade off behavior.

In [24], a case study has been performed which evaluates the performance of such pipeline multiprocessor systems against a distributed systems architecture. The work utilized ASIPs and it was shown that selective optimization of the individual cores provide necessary performance improvement to balance the overall latency of the pipeline stages in the system. However, there was no formal approach to explore this design space.

We make the following contributions in this paper:

1. We formulate the problem of mapping processor configurations in the context of heterogeneous multiprocessor pipeline architectures. To the best of our knowledge, our work is the first to address this problem using ASIPs.
2. We propose a heuristic to rapidly produce a near optimal configuration for given benchmark applications which are partitioned into stages. We conduct a design exploration of pipeline multiprocessor designs for MP3 and JPEG encoders.
3. We show the proposed techniques by enhancing a commercial design flow (using Tensilica's Xtensa LX platform) and applied them to real embedded streaming applications (*e.g.* JPEG encoder, MP3 encoder).

3. Background

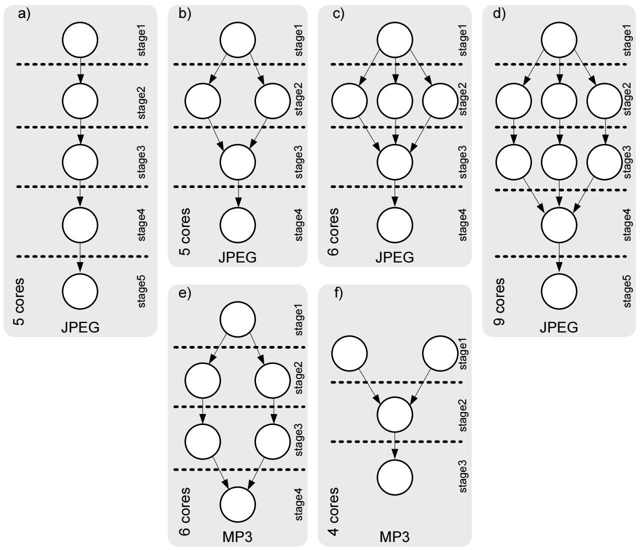


Figure 1: Design Space

Our work is based on optimizing sequential applications, which have the following characteristics: 1) Each streaming application contains a kernel which is partitioned to several pipeline stages. This kernel is run multiple times (in JPEG for example, this will be run every frame). Minor loops which have such characteristics would be considered as atomic and would not be further partitioned. 2) The application exhibits a dataflow software architecture. Input data is sequentially processed deterministically and output as results in the same order and manner.

An application which has the above characteristic can be partitioned to represent different stages in a pipeline flow. The partitioned application is derived from a standard sequential program written in C. Figure 1 shows possible designs that can be implemented as a pipeline multiprocessor implementation. These designs have been manually created by the designer as examples for the design exploration. Each design has a set of cores, where each pipeline stage is executed by at least one processor. A particular stage takes inputs from the previous stage which is connected via FIFO queues. Thus, these connected systems allow each core to run independently of the other, provided that each stage has the necessary input to begin data processing.

Each multiprocessor configuration from Figure 1 has a large design space. Each processor in the pipeline system can be configured and mapped to special purpose hardware. A particular configuration which is generated for a design has to be optimal (or near optimal). This near optimal configuration can be achieved by changing hardware parameters of each processor to achieved the required performance at the lowest possible cost. Finally, the system is optimized, such that the area increase incurred by pipelining the systems is more than offset by the increase in performance.

3.1 Benchmark Applications

Readily partitioned benchmark programs are not freely available to the research community. We created our own set of benchmark applications based on single processor benchmarks. Two freeware compression algorithms, MP3 and JPEG encoding algorithms, were chosen and ported to the Tensilica Xtensa LX [5] platform architecture.

The data flow graphs are obtained from these benchmark applications by analyzing the data stream throughout the benchmark applications. These benchmarks are partitioned manually into various pipeline / data flow stages, adhering to the respective standards (JPEG & MP3). The partitions are then mapped to stages in a pipeline system (refer to Figure 1). These stages are then written as standalone programs in Xtensa LX processors.

We created four multiprocessor configurations for the JPEG encoder and two configurations for the MP3 encoder. Figure 1 shows the set of designs of the two benchmark applications. The figure also shows the connectivity of each stage in the pipeline; each arrow denotes a FIFO connection to the next stage in the pipeline. Due to space restriction, we do not show the mapping of task to the various stages in the pipeline implementation of Figure 1.

3.2 System Architecture

The Xtensa LX [5] is part of the Tensilica line of cores which is configurable, extensible and supported by automatic hardware and software generation tools. The core is synthesizable and allows designers to configure each implementation to match the target application requirements. It supports extended instructions include fusion instructions [28], SIMD/vector instructions and FLIX [6] instructions.

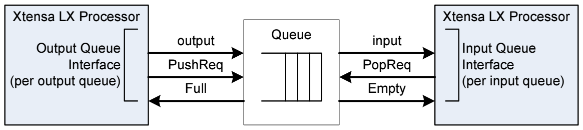

Xtensa LX Processor
Output Queue Interface (per output queue)
output
PushReq
Full
Queue
input
PopReq
Empty
Xtensa LX Processor
Input Queue Interface (per input queue)

Figure 2: Xtensa LX Queue Interface

The key feature which is used in this work is the queue interface (introduced in Xtensa LX - refer to Figure 2). This feature support external communications at a much wider bandwidth than existing interconnects. Queue interfaces (TIE instructions) are used to *pop* an entry from an input queue for incoming data or push data to an outgoing queue. The Xtensa Toolset automatically generates the logic to stall the processor when it reads an empty input queue or writes to a full output queue.

We have profiled the single processor implementations of the MP3 and JPEG encoders on several architectural configurations and have decided on the implementations for MP3 (*LX1*) and JPEG (*LX2*) respectively (refer to Table 1). The table also shows the extra configurable options that are implemented in the *LX1* core and not the *LX2*, due to the higher computation demand of the MP3 benchmark application. The system is currently simulated in a cycle accurate Xtensa Modeling Protocol (XTMP) environment.

	LX1	LX2
Benchmark	MP3	JPEG
Speed	533 MHz	
Process	90nm GT	
Pipeline length	5	
Size	79,885 gates	63,843 gates
Core Size	0.41 mm^2	0.32 mm^2
Core Power	81.28 mW	74.35 mW
MAC16	✓	
MULUH/MULSH	✓	
NSA/NSAU	✓	
Synchronize Instruction	✓	
Conditioned Store Synchronize	✓	
Max Instruction Width	8 bytes	
PIF Interface Width	128 bits	

Table 1: Processor Configuration

The overall core size for each core include the instruction and data cache area and extended instruction size. This is explained in Section 8.

4. The System

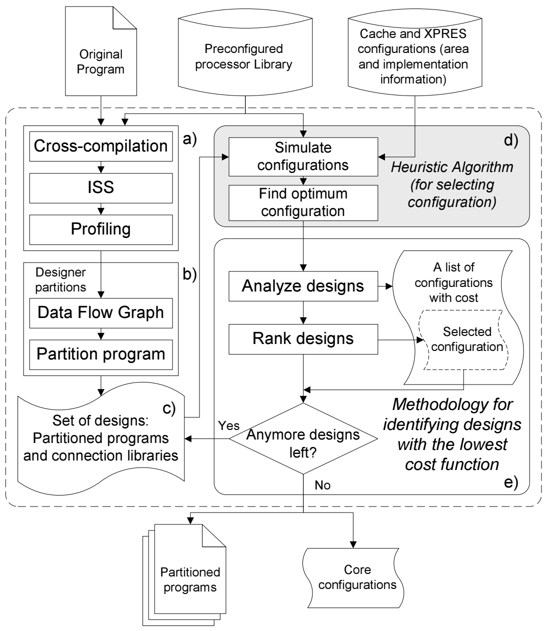

Original Program
Preconfigured processor Library
Cache and XPRES configurations (area and implementation information)
Cross-compilation
a)
ISS
Profiling
Designer partitions
b)
Data Flow Graph
Partition program
Set of designs: Partitioned programs and connection libraries
c)
Simulate configurations
Find optimum configuration
d)
Heuristic Algorithm (for selecting configuration)
Analyze designs
Rank designs
A list of configurations with cost
Selected configuration
Yes
Anymore designs left?
No
Methodology for identifying designs with the lowest cost function
e)
Partitioned programs
Core configurations

Figure 3: Design flow for exploring the heterogeneous multiprocessor design space

The design flow for obtaining the best configuration for a given application is summarized in Figure 3. The input to the system consists of: an application written in C/C++, a library of pre-configured processors and a set of cache and XPRES configurations (with the respective area utilization information). The process starts with a program which is then compiled and profiled to detect the hotspots in the algorithm (Figure 3-a). The designer then derives a data flow graph which describes the flow of data through the program. This information can be used to manually (or automatially) partition (Figure 3-b) the program into multiple modules (Figure 3-c).

The designer may produce a set of possible architectures as shown in Figure 1. Each design may have different number of pipeline stages and different parallel pipeline flows. Each design would consist of individual standalone programs; each capable of running independently on a microprocessor core. A heuristic (refer to Section 6) is used to rapidly explore the design space to find the best architectural configuration (Figure 3-d).

The algorithm would produce a configuration which is near optimal. The algorithm is run on all possible designs until the most optimum configuration is found. This heuristic would be used on all possible architectures from the designer (Figure 3-e). The design flow would eventually produce a set of partitioned programs, including their core configurations (i.e. cache and XPRES configurations).

5. Problem Definition

The selection and mapping of different regions of software to multitudes of hardware configuration have been widely explored. Previous approaches to hardware-software codesign using ASIPs in a multiprocessor configuration utilize various mappings to NP-hard problems, notably the 0-1 knapsack problem and its derivatives.

Assumptions

We make the following assumptions:

- Each design terminates with only one output processor
- Each stage in the pipeline refers to a physical processor with the assigned task of the stage
- Runtime is calculated assuming the processors are not stalled due to an empty input queue (*POP* stall) or a full output queue (*PUSH* stall). This assumption is valid since the overall computation time for a pipeline will be dominated by the stage with the longest execution time (critical stage in the pipeline). Thus, for the purpose of estimating the pipeline design performance, these stalls can be ignored. (within 2% accuracy in our experiments - Refer to Figure 5)
- Whenever a stage has parallel pipelines, we assume that all parallel parts are identical in terms of the number of pipeline stages and processors, such as in stages 2 & 3 of Figure 1(d). This assumption allows us to simplify the design process and as extension to this work, we can have asymmetrical pipeline stages which is beyond the scope of this paper.

Formulation

A pipeline program can be represented as a Data Flow Graph (DFG) where vertices represent tasks and edges represent FIFO connections between processors. The designer begins with a list of N possible multiprocessor designs. These different designs would already be partitioned prior to this stage either manually or automatically.

$$G = (V, E) \tag{1}$$

A program can be represented as a process graph, G. The partitioned processes which represent the stages of the pipeline are given in set

$$V = \{v_j : 1 \leq j \leq J\} \tag{2}$$

where J is the maximum number of nodes in the design. Each v_j represents a processor which is mapped to partitioned code segments (refer to Section 3). FIFO connections for the design are represented by

$$E = \{(v_i, v_j) : 1 \leq i < j \leq J,\ v_i \in V,\ v_j \in V\} \tag{3}$$

The execution time and area cost of the various process stages have been profiled and can be obtained via the functions $\mathbb{R}_k()$ and $\mathbb{C}_k()$. The *CFG* set contains the processor configurations on which the different program stages will be running. The different processor configurations refer to the various combination of cache configurations and the enabling of extended instructions generated via the Tensilica XPRES Tool [5]. We define the functions below:

$$\mathbb{R}_k : V \longrightarrow \mathbb{Z}^3 \mid \forall k, 1 \leq k \leq K \tag{4}$$

$$\mathbb{L}_k : V \longrightarrow \mathbb{Z} \mid \forall k, 1 \leq k \leq K \tag{5}$$

$$\mathbb{C}_k : V \longrightarrow \mathbb{Z} \mid \forall k, 1 \leq k \leq K \tag{6}$$

where $K = |CFG|$ is the total number of configurations available. Equation 4 is an integer set of tuples containing the runtime values

System Configuration	Exhaustive	Heuristic
5 processors (4 stages)	2s	<1s
5 processors (5 stages)	15m	<1s
6 processors	1085m	<1s
9 processors	inconclusive	<1s

Table 2: Exploration time

of the *initialization*, *core iterations* and *finalization* of the program stage. Equation 5 and 6 both refer to the respective sets of latency and cost for the particular pipeline stage implementation.

A set of configuration defines the configuration for each core, v_j. The runtime, **R** of each set of configurations is obtained by using an instruction set simulation (refer to Section 7). The total cost of the design would be sum of the cost of each individual processor implementation in the pipeline design.

$$\mathbf{C} = \sum_{j=1}^{J} \mathbb{C}_k(v_j) \tag{7}$$

where k is the corresponding configuration number of v_j, and J is the total number of processors in the design. Finally, the cost function which needs to be minimized is defined as

$$\Theta = \mathbf{R} \times \mathbf{C} \tag{8}$$

where **R** and **C** are all integer values. Given the definitions above, we now try to solve the Θ minimization problem using the algorithm in Figure 4.

```
' $\check{C}_j$ is a set of area cost of $K$ implementations of pipeline $j$
Let $\check{C}_j = \{c_k : \forall k, c_k = \mathbb{C}_k(v_j), 1 \leq k \leq K, v_j \in V_n\}$

**Find all possible design configuration combinations:**
Let $\mathbf{K}^* = \{(k_1 ... k_J) : \forall j, k_j \in K, 1 \leq j \leq J\}$

For each design $k^* \in \mathbf{K}^*$
    ' Find total area cost
    Area Cost $= \sum^J (c_j)$ where $c_j = \mathbb{C}_{k_j}(v_j)$ and $k_j \in \mathrm{K}^*$
    ' Find runtime for this particular implementation
    Runtime = simulation output with configuration $k^*$
    $\Theta$ = Runtime $\times$ Area Cost
End for

**Configuration $k^*$ with smallest $\Theta$ would be selected**
```

Figure 4: Exhaustive Search to obtain a pipeline multiprocessor design with the lowest cost (Runtime × Area Cost)

6. Heuristic

The problem formulated above (as an algorithm) is exhaustive, and will not converge to a solution quickly, due to the large design space. The permutations for different implementations would result in an exponential complexity of order, $O(n^p)$, where n is the maximum number of possible processor configurations, and p the number of processors in the multiprocessor configuration. Table 2 shows the overwhelming computation time for such an exhaustive search. To more effectively explore the design space, we developed a heuristic to closely match the optimal configuration given by the algorithm in Section 5 without simulating all possible configurations.

The different implementations of the pipeline stages would result in different execution times within the pipeline. As faster stages will stall for slower ones, we assume that on average, each pipeline stage latency would be equivalent to the latency of the critical stage. We now redefine the runtime, **R** of a configuration to include runtimes of each individual core. The runtime of a particular configuration can be defined as

$$\mathbf{R} = \mathbb{R}^{init}(v_1) + \sum_{j=1}^{J} \mathbb{L}(v_j) + (I - 1) \times \mathbb{L}(v_{crit}) + \mathbb{R}^{final}(v_J)$$

where I is the number of iterations and $\mathbb{R}^{init}$,$\mathbb{R}^{process}$ and $\mathbb{R}^{final}$ are mapping functions to the initialization, core iteration and finalization execution times respectively. In pipeline systems, increasing workload in the pipeline would bring the system closer to the theoretical performance improvement [16]. Similarly, as the number of iterations, I increases to a significantly large number, the sum of the latencies of the pipeline could be ignored. The equation above can then be simplified to

$$\mathbf{R} = \mathbb{R}^{init}(v_1) + \mathbb{R}^{process}(v_{crit}) + \mathbb{R}^{final}(v_J) \tag{9}$$

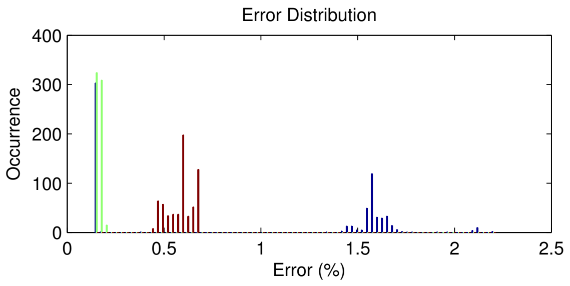


Figure 5: Error distribution when using equation 9

This permits us to calculate the execution time of the pipeline system by using only runtimes of the initial processor, the critical stage processor and the final processor. Figure 5 shows the distribution of errors to the runtime information when the above equation is used. These errors were produced using full simulation and calculated times of the five and nine processor JPEG systems. It is seen from Figure 5 that the estimated runtime errors are within 2.5% of the actual value.

Based on Equation 9, we develop the heuristic shown in Figure 6. The heuristic relies on the fact that our simulation period is of the order $O(n \times p)$, compared to the complexity of $O(n^p)$ in Section 5, where n and p are the maximum number of possible processor configurations and pipeline stages respectively.

```
Get minimum core iteration runtime of each processor:
$\mathbf{R}^- = \{r_j : \forall j, 1 \le j \le J, r_j = \text{MIN}^{k \in K}(\mathbb{R}_k^{process}(v_j)),$ where $v_j \in V\}$

Find critical node:.
' Critical node is the processor with the worst minimum core iteration runtime
Critical node is $v_{crit}$ where $\forall k \in K, \mathbb{R}_k^{process}(v_{crit}) = \text{MAX}(\mathbf{R}^-)$

Start with critical node:
For each configuration $k$ in set $K$:
    Calculate Cost= $\mathbb{R}_k^{process}(v_{crit}) \times \mathbb{C}_k(v_{crit})$
Next $k$
Configuration $k$ with smallest cost would be selected
Critical_Runtime $= \mathbb{R}_k^{process}(v_{crit})$

Evaluate all other nodes:
For every other node, $v_j \in V$:
    Filter all configurations $k$ where $\mathbb{R}_k^{process}(v_j) <$ Critical_Runtime
    For each remaining configuration $k$ in $K$ configurations:
        If this is the first node, calculate Cost= $\mathbb{R}_k^{init}(v_j) \times \mathbb{C}_k(v_j)$
        If this is the last node, calculate Cost= $\mathbb{R}_k^{final}(v_j) \times \mathbb{C}_k(v_j)$
        Otherwise, calculate Cost= $\mathbb{C}_k(v_j)$
    Next $k$
    Configuration $k$ with smallest cost would be selected
Next node

Output: The set of $k$'s obtained for each processor
```

Figure 6: A heuristic for mapping the pipeline stages to the various hardware implementations which tries to maximize the performance per area ratio.

7. Experimental methodology

We used Tensilica's Xtensa RA2006.4 Toolset for the Xtensa LX family of processors. The toolset provides a set of compilation tools to compile C/C++ code for the architecture described in Table 1. The Tensilica Instruction Set Simulator (ISS) and Xtensa Modelling Protocl (XTMP) environment were used to run the multi-core systems. For each system, multiple Xtensa cores were instantiated and XTMP was used to connect the cores together, including memory models and peripherals. The ISS directly models the Xtensa pipeline and operates as a system-simulation component using the XTMP environment. With XTMP, different multiprocessors configuration could be set up and simulated rapidly.

The simulator allows for communication between the cores and peripherals using a cycle-accurate, split-transaction simulation model without using a clock. The ISS was used to generate profiling data for all cores in the system, which were then analyzed using Tensilica's *gprof* profiler. The profiles can include the cycles for all functions executed by the cores. The ISS can also print a summary of the total cycle count and global stalls of each core.

Each individual core is connected via the queue interface provided by the Xtensa LX core using the XTMP environment. Queue models have been created and used in the XTMP environment as libraries. In our work, we simulate all queues with a very large amount of queue buffers, so that no *PUSH* stalls will occur. If a fixed queue size is required, our architecture design can always be easily mapped to a Kahn Process Network [18], and using the available tools for KPNs, the optimal queue size can be calculated.

We created our benchmark programs by identifying the various stages of the MP3 and JPEG encoders and mapping them to individual processors. We partition and allocate these stages base on the open standards of the respective encoders. We created four multiprocessor configurations for the JPEG encoder and two configurations for the MP3 encoder. An XTMP simulation program, specially customized to generate profiling, runtimes of each stage in the pipeline and other relevant benchmark information is created for each of these multiprocessor systems.

The toolset also includes the XPRES (Xtensa PRocessor Extension Synthesis) compiler which creates tailored processor descriptions for the Xtensa processors from native C/C++ code. We are able to reuse the existing ASIP design flow to create custom RTLs for each core in the system. Using the designer-defined input of C programs to be analyzed, XPRES extends the base processor with new instructions, operations and register files using TIE extensions. It does so by automatically generating a new TIE file which can be included when recompiling the source code. Half the set of the configuration options defined in Equations 4, 5 and 6 are XPRES enabled. However, it should be noted that the XPRES tool configuration was not run in the MP3 design space due to the long simulation time for MP3 encoding. Nevertheless, our heuristic still provides a configuration close to the optimum configuration in the MP3 design space.

Area cost include the base processor, instruction & data caches and the TIE instructions. A raw image of size 227 by 149 pixels is used as raw input stream to the JPEG encoder systems, whereas a 6 second PCM encoded music clip is used as the input stream to the MP3 systems.

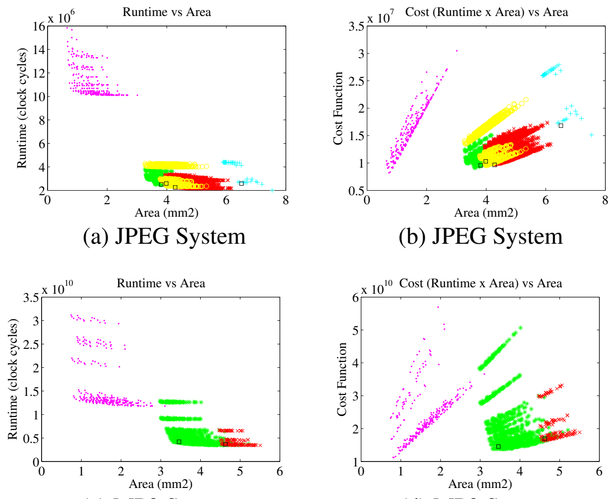
8. Results & Analysis

Figure 7 shows the design space exploration for both the JPEG and MP3 multiprocessor systems. Figure 7(a) shows the design space of the JPEG algorithm implementation. The subfigures on the left show the runtime performance of the benchmarks vs area. In all four figures, the group of data points on the left corner of each graph corresponds to the single processor implementation. The square markers on the graphs are the points obtained via our heuristic algorithm.

Table 3 shows the runtimes and cost functions obtained via the heuristics which we develop in Section 6. The first column shows the number of processors in the design. The second column denotes the number of pipeline stages in the system. If there are more processors then pipeline stages, this denotes that a parallel pipeline stage exists. MP denotes multi-pipeline while SP denotes single pipeline. Columns four and seven show the runtime and cost obtained via our

Processor Count	Stages	Description	H Runtime	E Runtime	Runtime Deviation	H Cost	E Cost	Cost Deviation
5	4	JPEG : MP	2507481	2457092	2.05%	9568917	9072915	5.47%
5	5	JPEG : SP	2585894	2332559	10.86%	10305171	9540790	8.01%
6	4	JPEG : MP	2255188	2158028	4.5%	9662435	9643188	0.20%
9	5	JPEG : MP	2582436	2005718	28.75%	16814581	15122927	11.19%
4	3	MP3 : MP	4204536116	3509943826	19.79%	14549794245	13759966039	5.74%
6	4	MP3 : MP	3658802853	3428272126	6.72%	16933507412	16320671462	3.75%

Table 3: Runtime and cost from the heuristic. The table shows how much these values differ from the best possible value.



(a) JPEG System (b) JPEG System

(c) MP3 System (d) MP3 System

Figure 7: Comparison of JPEG and MP3 multiprocessor pipeline systems

heuristics. The exhaustive versions are shown in columns five and eight. The deviations of the runtime and cost values are stated in columns six and nine.

The deviation of the heuristics obtained is shown as a percentage of the best possible values obtained from an exhaustive search. Do note that our algorithm emphasizes on reducing the cost function rather than maximizing performance of the application. Nevertheless, our heuristic still produces runtime values close enough to the best possible runtime.

From our heuristic, we are able to determine a near optimal configuration. These are the design with five processors (four stages) for JPEG and the design with four processors (three stages) for MP3. The design for the systems are shown in Figures 1(b) and 1(f) respectively.

For JPEG encoding, we obtained around 5.47% of the optimum value, while in MP3 encoding we manage to get close to within 5.74%. The heuristic analysis provided us with the configuration which is close to optimal for the particular MP3 and JPEG encoders.

In Figures 7(a) and 7(c), we show that our heuristic provides us with points close to optimum runtime while minimizing the cost function. With the five processor (four stages) parallel pipeline JPEG implementation, we are able to obtain at least 4.11X speedup over the single processor implementation and 3.36X speedup, with the four processor (three stages) parallel pipeline MP3 encoder.

9. Conclusion

In conclusion, we have formalized the problem of mapping processor configurations in the context of ASIP multiprocessor system which are implemented in a pipeline manner. We have also presented a heuristic to obtain a near optimal configuration (smallest cost) given a partitioned benchmark program. This is complemented with a full methodology that uses this heuristic to rapidly explore the architectures provided from the designer and thus explore and select the architecture which provides the best performance per area. This framework utilizes Tensilica's Xtensa LX [5] configurable cores which provide the queue interface that is used to connect each processor in the system in a pipelined configuration. We have explored the design space of such an architecture by using the existing ASIP design flow to rapidly select the best cache configurations and extended instruction to provide the neccesary speedup while minimizing area; thus providing a good performance to area ratio.

10. References

[1] Altera Nios Processor. Altera Corp. (http://www.altera.com).
[2] ARCtangent. ARC International (http://www.arc.com).
[3] SP-5flex. 3DSP Corp. (http://www.3dsp.com).
[4] SystemC Initiative. (http://www.systemc.org).
[5] Xtensa Processor. Tensilica Inc. (http://www.tensilica.com).
[6] Flix: Fast relief for performance-hungry embedded applications. Tensilica Inc. (http://www.tensilica.com/pdf/FLIX_White_Paper_v2.pdf), 2005.
[7] J. Axelsson. A Case Study in Heterogeneous Implementation of Automotive Real-Time Systems. In *CODES'98*, Seattle, 1998.
[8] S. Banerjee, T. Hamada, P. M. Chau, and R. D. Fellman. Macro Pipelining Based Scheduling on High Performance Heterogeneous Multiprocessor Systems. *Signal Processing, IEEE Transactions on*, 43(6):1468 – 1484, 1995.
[9] S. Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *RTAS'04*, pages 536 – 543, 2004.
[10] A. Berić, R. Sethuraman, C. A. Pinto, H. Peters, G. Veldman, P. van de Haar, and M. Duranton. Heterogeneous Multiprocessor for High Definition Video. In *ICCE'06*, pages 401 – 402, 2006.
[11] T. D. Braun, H. J. Siegel, and A. A. Maciejewski. Heterogeneous computing: Goals, methods, and open problems. In *HiPC 2001*, volume 2228, pages 302 – 320, Hyderabad, India, 2001. Springer.
[12] K. S. Chatha and R. Vemuri. A Tool for Partitioning and Pipelined Scheduling of Hardware-Software Systems. In *ISSS'98*, pages 145 – 151, Hsinchu, 1998.
[13] CriticalBlue. Coprocessor synthesis – increassing system on chip platform ROI. Technical report, CriticalBlue, June 2004.
[14] T. Givargis, F. Vahid, and J. Henkel. System-Level Exploration for Pareto-Optimal COnfigurations in Parameterized System-on-a-Chip. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 10(4):416 – 422, 2002.
[15] S. Gopalakrishnan and M. Caccamo. Task Partitioning with Replication upon Heterogeneous Multiprocessor Systems. In *RTAS'06*, pages 199 – 207, 2006.
[16] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 3rd edition, 2003.
[17] J. Jeon and K. Choi. Loop Pipelining in Hardware-Software Partitioning. In *ASP-DAC'98*, pages 361 – 366, Yokohama, Japan, 1998.
[18] G. Kahn. The semantics of a simple language for parallel programming. In *IFIP'74*, pages 471 – 475, Stockolm, Sweden, 1974.
[19] M. Kim, D. Kim, and G. E. Sobelman. MPEG-4 performance analysis for a CDMA network-on-chip. In *ICCCAS'05*, pages 493 – 496, 2005.
[20] T. Kodaka, K. Kimura, and H. Kasahara. Multigrain Parallel Processing for JPEG Encoding on a Single Chip Multiprocessor. In *IWIA'02*, pages 57 – 63, 2002.
[21] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan. Heterogeneous Chip Multiprocessors. *Computer*, 38(11):32 – 38, November 2005.
[22] D. Pham. The design and implementation of a first-generation cell processor. In *ISSCC 2005*, pages 184 – 186. IEEE CS Press, 2005.
[23] F. Salice, L. Del Vecchio, L. Pomante, and W. Fornaciari. Partitioning of Embedded Applications onto Heterogeneous Multiprocessor Architectures. In *ACM symposium on Applied computing*, pages 661 – 665, Melbourne, Florida, 2003.
[24] S. L. Shee, A. Erdos, and S. Parameswaran. Heterogeneous Multiprocessor Implementations for JPEG : A Case Study. In *CODES+ISSS'06*, Seoul, Korea, 2006.
[25] G. C. Sih and E. A. Lee. Declustering: A New Multiprocessor Scheduling Technique. *IEEE Transactions of Parallel and Distributed Systems*, 4(6):625 – 637, 1993.
[26] J. E. Smith and G. S. Sohi. The Microarchitecture of Superscalar Processors. *Proceedings of the IEEE*, 83(12):1609 – 1624, 1995.
[27] M. T. J. Strik, A. H. Timmer, J. L. van Meerbergen, and G.-J. van Rootselaar. Heterogeneous multiprocessor for the management of real-time video and graphics streams. *Solid-State Circuits, IEEE Journal of*, 35(11):1722 – 1731, 2000.
[28] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha. Custom-instruction synthesis for extensible-processor platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(2):216–228, 2004.
[29] F. Sun, S. Ravi, A. Raghunathan, and N. K. Jha. Synthesis of Application-specific Heterogeneous Miltiprocessor Architectures using Extensible Processors. In *VLSID'05*, pages 551 – 556, 2005.
[30] V. Živojnović, S. Pees, and H. Myer. LISA-machine description language and generic machine model for HW/SW co-design. In *Workshop on VLSI Signal Processing*, pages 127–136, 1996.
[31] A. Wieferink, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, G. Braun, and A. Nohl. System Level Processor/Communication Co-exploration Methodology for Multiprocessor System-on-Chip Platforms. *Computers and Digital Techniques, IEE Proceedings*, 152(1):3 – 11, 2005.
[32] N. Zhang and C.-H. Wu. Study on Adaptive Job Assignment for Multiprocessor Implementation of MPEG2 Video Encoding. *Industrial Electronics, IEEE Transactions on*, 44(5):726 – 734, 1997.