An Automated Exploration Framework for FPGA-based Soft Multiprocessor Systems

Yujia Jin Nadathur Satish Kaushik Ravindran Kurt Keutzer
University of California at Berkeley, CA, USA
{yujia, nrsatish, kaushikr, keutzer}@eecs.berkeley.edu

ABSTRACT

FPGA-based soft multiprocessors are viable system solutions for high performance applications. They provide a software abstraction to enable quick implementations on the FPGA. The multiprocessor can be customized for a target application to achieve high performance. Modern FPGAs provide the capacity to build a variety of micro-architectures composed of 20-50 processors, complex memory hierarchies, heterogeneous interconnection schemes and custom co-processors for performance critical operations. However, the diversity in the architectural design space makes it difficult to realize the performance potential of these systems. In this paper we develop an exploration framework to build efficient FPGA multiprocessors for a target application. Our main contribution is a tool based on Integer Linear Programming to explore micro-architectures and allocate application tasks to maximize throughput. Using this tool, we implement a soft multiprocessor for IPv4 packet forwarding that achieves a throughput of 2 Gbps, surpassing the performance of a carefully tuned hand design.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Pipeline processors*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

FPGA, Soft multiprocessors, IPv4 packet forwarding, Design space exploration, Integer Linear Programming

1. INTRODUCTION

A *soft multiprocessor* is a network of programmable processors crafted out of processing elements, logic blocks and memories on an FPGA. They allow the user to customize the number of programmable processors, interconnect schemes, memory layout and peripheral support to meet application needs. Deploying an application on the FPGA is tantamount to writing software for this multiprocessor system. Results in [9] show that soft multiprocessor systems are viable alternatives for high performance applications. They avoid risks due to high silicon development costs and design turnaround times, while providing a software abstraction to enable a quick implementation on the FPGA. They also open FPGAs to the larger world of software designers.

Modern FPGAs provide the processing capacity to build a variety of micro-architectural configurations. Today, we can build multiprocessors composed of 20-50 processors (and growing with Moore's law), complex memory hierarchies, heterogeneous interconnection schemes and custom co-processors for performance critical operations. Future projections forecast that embedded systems will soon be composed of over 100 processors on a single chip to guarantee acceptable performance [4]. However, the diversity in the architectural design space makes the task of determining an efficient multiprocessor configuration tuned for a target application challenging. Currently, the designer must manually explore the large and complex design space of micro-architectures for an application to achieve the full performance potential of FPGA multiprocessors.

The objective of this paper is to address the following question: How do we design efficient systems of soft multiprocessors for a target application? To ease this design challenge, we advance an automated framework to assist the designer in exploring the design space of soft multiprocessor micro-architectures. The objective is to identify the best multiprocessor on the FPGA for a target application and optimally map the application tasks and communication links to this micro-architecture. We construct analytical models of the architecture and application and solve the exploration problem using Integer Linear Programming (ILP).

In [9] we presented a hand-tuned soft multiprocessor design for the data plane of the IPv4 packet forwarding application that achieves a throughput of 1.8 Gbps. To evaluate the effectiveness of our framework, we use it to automatically explore the design space of micro-architectures for the same packet forwarding application. Using our tool, we optimally allocate the application tasks and links to maximize throughput and compare area and performance of the resulting design to the hand-tuned implementation.

The rest of the paper is organized as follows. We first describe some related work on design space exploration for multiprocessors in Section 2. Section 3 presents our framework for micro-architecture exploration and task allocation. We follow with details on the ILP formulation in Section

CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA.

4. In Section 5, we apply our exploration framework to determine a multiprocessor for IPv4 packet forwarding and compare its performance to the hand-tuned implementation in [9]. We end with some conclusions and future directions in Section 6.

2. RELATED WORK

Design space exploration of micro-architectural alternatives is a pivotal problem in the design of multiprocessor systems. The general problem of design space exploration is hard since there are a wide variety of architectures and applications each with their own set of objectives and constraints. Many techniques have been proposed for multiprocessor design space exploration such as list scheduling heuristics, genetic algorithms, simulated annealing and linear programming. A systematic survey of existing techniques can be found in [6].

Although the general exploration problem is difficult to solve, the problem we consider in this paper is more specific. Following the MESCAL methodology [7], we define our exploration problem by identifying the architectural design space and describing our target class of applications. We limit the architectural design space to multiprocessor micro-architectures built from a network of processors interconnected using buses and point-to-point FIFO links. We can choose from different types of hard and soft processors, memories and communication elements that the FPGA supports and customize the topology of the multiprocessor. In the application space, we target dataflow and stream oriented applications which are suitable for soft multiprocessor systems. This is an important class of high performance applications that are common in the networking and signal processing domains. A few examples of these applications are the networking and DSP benchmarks in the EEMBC suite [1].

Our exploration problem takes in a single application described in the form of an application graph (described in more detail in Section 3.2). Our exploration objective is to maximize throughput, which is the most important measure of performance in stream-oriented applications. The output of our exploration is a soft multiprocessor system customized for the target application. This problem is more manageable than the general design space exploration problem, which can include arbitrary hardwired logic. It is also more focused because we are targeting the multiprocessor system to a specific application rather than a domain of applications. However, this problem is still challenging because of the freedom to customize the computation, memory and communication capabilities to achieve high performance.

There have been other attempts at solving design space exploration problems with similar application and architecture features. The problem considered in this paper is most similar to the ones addressed by Hoang in [8] and Grajcar in [5]. Both these papers start with a similar application graph and optimize for throughput or makespan, which are closely related problems. However, the techniques used to solve the optimization problem in these papers differ from our approach. Hoang [8] proposes a heuristic to schedule DSP programs onto multiprocessors for maximum throughput. Grajcar [5] uses a genetic algorithm based on list scheduling to minimize the makespan of a set of tasks on a bus-based multiprocessor system. By contrast, we use an ILP based formulation to determine an optimal soft multiprocessor and allocate application tasks and links to maximize throughput. Such a formulation is flexible and can be easily adapted to different problem restrictions. Although finding the optimal design may be time consuming, in recent years ILP solver technologies have advanced significantly and many large problems can be routinely solved [3]. When near optimal designs are acceptable, additional time can be saved by setting the ILP solver to stop within an acceptable error margin.

3. FRAMEWORK FOR MICRO-ARCHITECTURE EXPLORATION

The starting point of the framework is the application graph that describes tasks and communication links. The application graph has an associated *timing model* that co-models computation and communication of the tasks. Our main contribution is an exploration tool to determine a valid soft multiprocessor configuration that maximizes throughput for a given application graph and timing model. We formulate the exploration problem as an Integer Linear Program (ILP). The outputs of the exploration step are (a) a micro-architecture configuration of processors and communication channels, and (b) a mapping of the application tasks and links onto the processors and channels of the micro-architecture.

In the following sections we provide details regarding the micro-architecture building blocks and the application model and present our ILP formulation to determine an optimal configuration that maximizes throughput.

3.1 Micro-architecture Building Blocks

To simplify the discussion of our problem formulation, we restrict the micro-architecture building blocks to processors and point-to-point FIFO communication channels. The set P denotes the set of available processors to build the multiprocessor. The set C is a set of FIFO communication channels (of infinite bandwidth) between pairs of processors. For simplicity, we assume there is only one channel connecting two processors. The element $c_{p,p'} \in C$ denotes a channel between processors $p, p' \in P, p \neq p'$.

3.2 Application Graph and Timing Model

The application graph is a directed acyclic graph $A = (T, L)$. A node $t \in T$ is a program task that is entirely executed in a single processor. $L \subseteq T \times T$ is the set of directed communication links between tasks. A link $l = (t, t') \in L$ connects tasks $t, t' \in T, t \neq t'$.

We associate a timing model with the application graph consisting of task execution time and communication overhead annotations (real numbers). Execution time for task $t \in T$ on processor $p \in P$ is given by the parameter $e_{t,p} \geq 0$. The communication overhead for a link $l = (t, t') \in L$ on channel $c_{p,p'} \in C$ is given by two parameters, $w_{l,c} \geq 0$ and $r_{l,c} \geq 0$. The parameter $w_{l,c}$ represents the time to perform a write of a block of data for task t on channel c. Similarly, the parameter $r_{l,c}$ represents the time to read a block of data for task t' on channel c. In the case when tasks t, t' in link $l = (t, t')$ are assigned to the same processor $p \in P$, the communication link is not mapped to a physical channel between processors and hence the read and write times for link l are assumed to be 0.

3.3 Valid Multiprocessor Configurations

Given an application graph $A = (T, L)$, a set of processors P, and a set of communication channels C, a valid configuration ν is an assignment with maps $\nu : T \rightarrow P$ and $\nu : L \rightarrow P \cup C$. Valid configurations resulting from the task allocation problem have the following characteristics:

(a) Each task $t \in T$ is assigned to a single processor $p \in P$.

(b) Each link $l \in L$ is assigned to either a processor $p \in P$ or a communication channel $c_{p,p'}$ connecting processors $p, p' \in P$.

(c) If a link $l = (t, t')$ is assigned to a processor $p \in P$, then the tasks t and t' must also be assigned to the same processor p. Logically, $\nu(l) = p \Leftrightarrow \nu(t) = p \wedge \nu(t') = p,\ \forall l = (t, t') \in L, \forall p \in P$.

(d) If a link $l = (t, t')$ is assigned to a communication channel $c_{p,p'} \in C$, then task t must be assigned to processor p, and task t' must be assigned to processor p' Logically, $\nu(l) = c_{p,p'} \Leftrightarrow \nu(t) = p \wedge \nu(t') = p',\ \forall l = (t, t') \in L, \forall c_{p,p'} \in C$.

3.4 Optimization Objective

The optimization objective is to find a valid configuration with maximum throughput. For a given application graph, we can restate the throughput maximization objective as minimizing the *makespan* of the system. The makespan of a schedule is the latest completion time for any task in it. Given a valid configuration ν satisfying properties (a-d) above, we can compute its makespan as follows. Let $T_p \subseteq T$ be a subset of tasks assigned to processor $p \in P$, i.e. $T_p = \{t \in T : \nu(t) = p\}$. Let $L_p^S \subseteq L$ be a subset of links that "start" at processor p in the given configuration, i.e. $L_p^S = \{l \in L : \nu(l) = c_{p,p'},\ p' \in P\}$. Similarly, let $L_p^E \subseteq L$ be a subset of links that "end" at processor p under ν, i.e. $L_p^E = \{l \in L : \nu(l) = c_{p',p}, p' \in P\}$. Then the total execution time Φ_p of processor p under the given configuration ν is:

$$\text{(e)} \qquad \Phi_p = \sum_{t \in T_p} e_{t,p} + \sum_{l \in L_p^S} w_{l,\nu(l)} + \sum_{l \in L_p^E} r_{l,\nu(l)}$$

In other words, the execution time of a processor is a sum of the execution time of the tasks assigned to it and the communication write overhead ($w_{l,c}$) for each outgoing link and the communication read overhead ($r_{l,c}$) for each incoming link. The makespan M of a valid configuration ν is the maximum execution time across all processors:

$$\text{(f)} \qquad M = \max\{\Phi_p : p \in P\}$$

3.5 Example

Figure 1 shows a simple example for the micro-architecture exploration under our simplified scenario. The application graph, A, consists of 2 independent branches, where each branch is a chain of 3 tasks. The architecture elements, P and C, are two different processors which can be optionally linked together with a FIFO queue. The execution time, $e_{t,p}$, for each task on each processor is shown in the figure. For both processors the FIFO access times, $r_{l,c}$ and $w_{l,c}$, are 10 cycles for each read or write access. An obvious mapping is to assign each application graph branch to a single processor. This produces a makespan of 70 cycles and it is not optimal. Figure 1 shows another valid configuration that achieves the optimal makespan of 60 cycles.

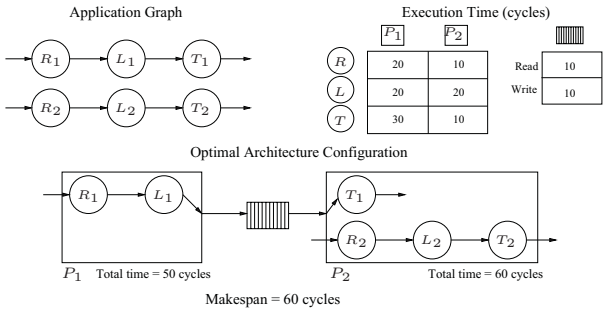

Figure 1: Example for micro-architecture exploration.

4. DETAILED PROBLEM FORMULATION

We now present the details of our ILP formulation to explore the space of FPGA micro-architectures and allocate application tasks and links to maximize throughput. The problem variables are described below (line below the variable explains what the variable denotes when set to 1).

$X_{t,p}$	$\in \{0,1\}$	$\forall\ t \in T, \forall\ p \in P$
		task t assigned to processor p
$Y_{l,p}$	$\in \{0,1\}$	$\forall\ l \in L, \forall\ p \in P$
		link l assigned to processor p
Z_l	$\in \{0,1\}$	$\forall\ l \in L$
		link l assigned to some channel
$XZ_{l,p}^S$	$\in \{0,1\}$	$\forall\ l \in L, \forall\ p \in P$
		p start of channel to which l assigned:
		i.e. $XZ_{l,p}^S = X_{t,p} \wedge Z_l,\ l = (t, t')$
$XZ_{l,p}^E$	$\in \{0,1\}$	$\forall\ l \in L, \forall\ p \in P$
		p end of channel to which l assigned:
		i.e. $XZ_{l,p}^E = X_{t',p} \wedge Z_l,\ l = (t, t')$
Φ_p	≥ 0	$\forall\ p \in P$
		the total execution time of processor p
M	≥ 0	max execution time over all processors

The X variables denote task assignment to processors. The Y, Z, XZ^S and XZ^E variables concern link assignment to processors or channels. $XZ_{l,p}^E$ indicates if a processor p is the end of a channel covering link l. Similarly, $XZ_{l,p}^S$ indicates if a processor p is the start of a channel covering link l. These variables express the connection of a task with a link (properties (c) and (d) of a valid configuration). Φ_p and M are part of the makespan computation.

4.1 Constraints

The constraints in the ILP formulation are detailed below.

$$\min\ M \quad \text{(makespan)}$$

$$\sum_{p \in P} X_{t,p} \quad = \quad 1, \forall\ t \in T \ \text{(a)}$$

$$\sum_{p \in P} Y_{l,p} + \sum_{p \in P} XZ^S_{l,p} = 1, \forall\, l \in L \text{ (b)}$$

$$\sum_{p \in P} Y_{l,p} + \sum_{p \in P} XZ^E_{l,p} = 1, \forall\, l \in L \text{ (b)}$$

$$Y_{l,p} + XZ^S_{l,p} = X_{t,p}, \quad \forall\, l = (t,t') \in L \text{ (c,d)}, \forall\, p \in P$$

$$Y_{l,p} + XZ^E_{l,p} = X_{t',p}, \quad \forall\, l = (t,t') \in L \text{ (c,d)}, \forall\, p \in P$$

$$\sum_{t \in T} X_{t,p}\ e_{t,p} + \sum_{l \in L} (XZ^S_{l,p}\ w_l + XZ^E_{l,p}\ r_l) = \Phi_p, \forall\, p \in P \text{ (e)}$$

$$\Phi_p \leq M, \forall\, p \in P \text{ (f)}$$

We associate each constraint with the properties (a-d) of a valid configuration and (e-f) for makespan computation. The first constraint ensures that every task is mapped to exactly one processor. This corresponds to property (a) of a valid configuration. The constraint for (b) ensures that every link is either covered by a processor or by a channel. The (c) and (d) properties taken together form an implication $Y_{l,p} \vee XZ^S_{l,p} \Rightarrow X_{t,p}$, and a similar implication for $X_{t',p}$. The other direction of the implication is also true; any link adjacent to a task covered by a processor has to be covered either by the *same* processor, or by a channel. Thus we can strengthen the above implication to the equality $Y_{l,p} + XZ^S_{l,p} = X_{t,p},\ l = (t,t') \in L, p \in P$. The resulting formulation after strengthening to equality satisfies both (c) and (d) properties. Constraint (e) computes the processor execution time. We assume that the channel read and write access times are only dependent on the amount of data communicated across the link that it supports. Therefore the channel subscript (c) is dropped from the definitions of r and w in Section 3.2. Constraint (f) checks that all execution times are within the makespan. The problem objective is to minimize makespan.

There are alternative ways of formulating the above constraints. Constraints (b),(c) and (d) could be rewritten to avoid using the XZ^S and XZ^E variables by replacing them with Z. However, variables similar to XZ^S and XZ^E are still required to calculate the communication delay, and hence the throughput. To correctly relate these variables with Z, we must introduce additional constraints expressing the logical and relation. However, the use of these constraints results in a much weaker ILP formulation. ILP solvers like CPLEX [2] cannot efficiently handle such constraints, resulting in long solution times. On the other hand, the above formulation can be solved efficiently by CPLEX.

4.2 Extensions

The formulation presented above has been simplified for the sake of clarity. In our complete ILP formulation for design space exploration, we also incorporate states in our application graph and memory elements in our architectural design space. It is easy to extend the above formulation to deal with these additional elements. We add variables to denote the assignment of states to memory elements. We add constraints to ensure that all states are covered. We also augment the performance calculation constraints with memory access times. To calculate the access time, we include memory performance profiles as a parameter to the formulation. With the above extensions to include states and memory accesses, we can model applications more completely. In our experiments we use the complete formulation for design space exploration.

Other constraints may be added by the designer in the above formulation to suit a specific design scenario. For example, designers often know that a particular multiprocessor topology would best suit a particular application. They may also have detailed knowledge about architecture features that limit performance. As an example, a bus may be known to have a large arbitration overhead. In such a case, a designer may want to limit the number of masters on that bus. In general, designer guidance may save a large amount of time in the exploration process. Our ILP based formulation is flexible enough to allow designers to add a variety of constraints for efficient design space exploration.

5. EXPERIMENTS

In this section, we use our framework to explore the design space for the header processing of IPv4 packet forwarding mapped onto Xilinx Virtex-II Pro 2VP50 FPGA. The 2VP50 consists of 23,616 slices and 522 KB on-chip BlockRAM memory. The building block of the multiprocessor system is the Xilinx MicroBlaze soft processor IP, which is a part of the Embedded Development Kit(EDK). The MicroBlaze processor occupies approximately 450 slices (2% of the 2VP50 FPGA area). The soft multiprocessor on the Xilinx FPGAs is a network composed of (a) multiple soft MicroBlaze cores, (b) the IBM PowerPC 405 cores, (c) distributed BlockRAM memories, (d) IBM CoreConnect buses: the On-chip Peripheral bus (OPB) and the Processor Local Bus (PLB), and (e) point-to-point FIFOs called Fast Simplex Links (FSL) [11].

We first describe the IPv4 packet forwarding application. We then describe the assumptions driving our ILP implementation and show the resulting design. Finally, we compare our result against the hand-tuned implementation in [9].

5.1 IPv4 Packet Forwarding Application

The IPv4 packet forwarding application runs at the core of network routers and forwards packets to their final destinations. The forwarding application consists of finding the next-hop router address and the egress port to which the packet should be sent. The data plane of the application involves three operations: (i) check whether the input packet is uncorrupted, (ii) find the next-hop and egress port using the destination address, and (iii) update header checksum and time-to-live fields (TTL), and forward the packet. To handle gigabit rates, routers must be able to forward millions of packets per second. The next-hop lookup is the most intensive data plane operation. The address lookup requires searching the forwarding table for the longest prefix that matches the packet destination address. A natural way to represent prefixes is a tree-based data structure (called a *trie*) that uses the bits of the prefix to direct branching. There are variations to the basic trie scheme that attempt to trade off the memory requirements of the trie table and the number of memory accesses required for lookup [10]. A commonly used scheme is a fixed-stride multi-bit trie. The stride is the number of bits inspected at each step of the prefix match algorithm [10]. In our experiments, the stride order is (12 4 4 4 4 4): the first-level stride inspects 12-bits

of the IP address and subsequent strides inspect 4 bits at a time, requiring a maximum of 6 memory accesses for an address lookup. An additional memory access is required to determine the egress port for the matched prefix.

The design objective is to maximize router throughput. In our experiments, we measure the number of packets processed per second by the multiprocessor design. We compute throughput by multiplying this packet rate with packet size. To model the worst-case scenario for the data plane forwarding performance, we make three assumptions: (a) All packet sizes are 64 bytes - this is the minimum size for an Ethernet frame. (b) All address prefixes in the route table are the full 32 bits in length - hence the trie lookup algorithm takes 7 memory accesses to find the next hop. (c) Results of the prefix search algorithm are not cached - the lookup algorithm must be executed for every packet header. We do not consider control plane processing, such as route table updates and ICMP error messages, since they occur infrequently and hence have a low impact on the core router performance. We also do not consider the packet payload transfer, since header processing is the most compute intensive part of the application.

Figure 2 shows the application graph for IPv4 packet forwarding. It contains 9 tasks. The first branch of 7 tasks are associated with the table lookup operation. Each of these tasks requires a memory access into the routing table. The remaining 2 tasks are associated with the TTL and version verification and checksum certification. The graph shows the parallelism that is inherent in the application: tasks belonging to different branches can be executed in parallel. For the IPv4 application, the seven memory lookups must be done in sequence. However, the two verification tasks can be executed at any point in the application. We replicate the graph multiple times to increase the amount of parallelism available in the application. The replicated graph will result a more efficient soft multiprocessor system because of the additional parallelism available.

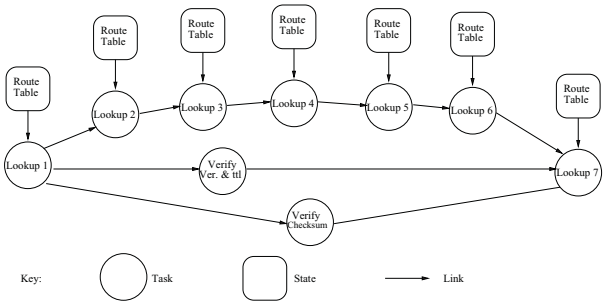

Figure 2: Application graph for IPv4 header processing.

5.2 Design space exploration for IPv4 packet forwarding

In the ILP formulation for IPv4 packet forwarding, we use FSL queues and OPB buses for the communication elements. We limit memory to on-chip block RAM (BRAM). For the Xilinx Virtex-II Pro 2VP50 FPGA, the maximum BRAM available is set to 500KB. We only use MicroBlaze processors. We conservatively estimate that this Virtex device can support up to 12 MicroBlaze processors. Thus we upper bound the number of MicroBlazes in the ILP formulation to 12. We also add constraints to reflect the architecture limitations. First, BRAM is dual ported. So we limit the number of communication connections to any memory unit to two. Second, we observe that OPB bus performance drops significantly when the number of bus masters exceeds two. So we limit the number of processors that can connect to a single OPB to two.

For the exploration, we use CPLEX [2] as our ILP solver. During the exploration we select the best design based on the ILP results and synthesize it to verify performance. Verification may fail because we do not consider routing detail in the ILP and the architecture performance may not be linear. If the verification fails, we add constraints to eliminate the current design and repeat the process.

Figure 3 shows the multiprocessor solution for header processing after the exploration. It contains 3 pipeline stages, with 4 processors in the first two stages and 3 processors in the last stage. The IP address lookup contains a total of 7 memory accesses. The first two stages involve 3 memory accesses each. The third stage has a single memory access. The verify operations are divided between the first and third stages. The processors in the third pipeline stage process packets 25% faster than the rate of the former stages. Hence, only three processors are needed in this stage. The throughput of the design obtained is 2 Gbps.

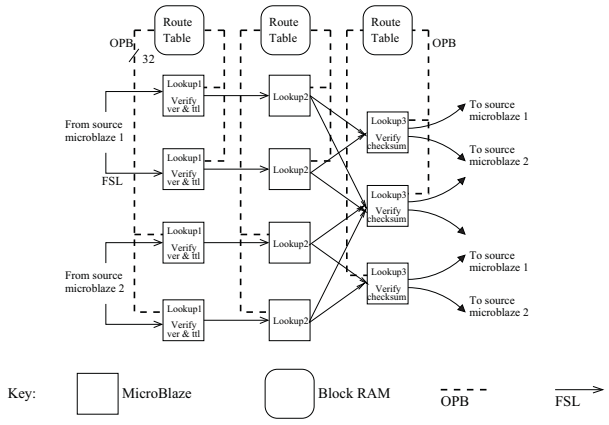

Figure 3: Multiprocessor design solution for IPv4 header processing after automated exploration.

5.3 Comparison to hand-tuned design

In this section, we first briefly summarize the hand-tuned design presented in [9]. We then compare this to the results of our automated exploration.

Figure 4 shows the final hand-tuned multiprocessor design for header processing as presented in [9]. In this design there are 4 arrays of pipelined MicroBlaze processors. FSL links transfer the header between processors in a pipeline array. All processors in lookup stages 1 and 2 access the same part of the route table in shared memory over the OPB bus. As in our automated exploration framework, each OPB can only be connected to two masters. The BRAM memory is dual-ported. Hence, the same route table memory can be serviced by 2 OPB buses. Thus, the choice of 4 branches is optimum for multiprocessor designs where shared resources are accessed over the OPB. The measured throughput of the header processing multiprocessor in Figure 4 is 1.8 Gbps.

The design from our automated exploration has higher

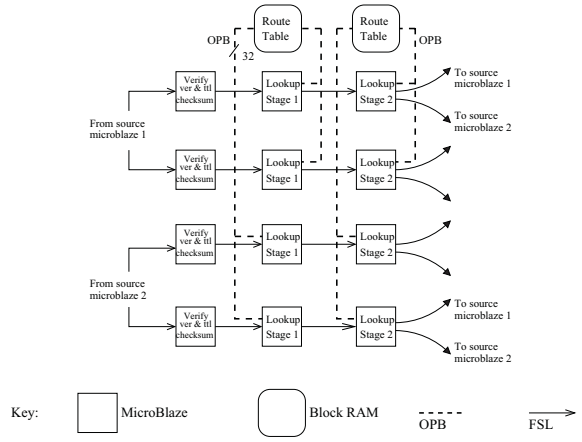

Figure 4: Hand-tuned multiprocessor design for IPv4 header processing.

throughput than the hand-tuned design, while using fewer processors. This is because the result of the design automation balances the workload across all the processors extremely well. In comparison, the hand-tuned multiprocessor design in Figure 4 is less balanced. The first verify stage is slightly underutilized as compared to the latter stages, leading to lower throughput. It would be difficult for a designer to balance out all stages evenly, while using the right number of processors per stage. This problem can however be optimally handled by an automated tool like the one presented in this work.

6. CONCLUSIONS

Modern FPGAs provide the processing capacity to build a variety of micro-architectural configurations. However, the vast diversity in the architectural design space makes the task of determining an efficient multiprocessor configuration tuned for a target application challenging. Currently, the designer must manually explore the large and complex design space of micro-architectures for an application to achieve the full performance potential of FPGA multiprocessors. So how can we improve the design process for soft multiprocessors for a target application?

To ease the design challenges and improve the design process, we have proposed an automated framework to assist the designer in exploring the design space of soft multiprocessor micro-architectures. The objective is to identify the best multiprocessor on the FPGA for a target application and optimally map the application tasks and communication links to this micro-architecture. Our framework uses analytical models of the architecture and application and solves the exploration problem using Integer Linear Programming (ILP).

To evaluate the effectiveness of our framework, we applied the framework to automatically explore the design space of micro-architectures for the IPv4 packet forwarding application. It produced an efficient soft multiprocessor design that achieved a throughput of 2 Gbps, surpassing a carefully tuned hand design.

While the initial experimental results for our exploration framework are promising, there are several areas for improvement. First, additional experiments are needed to further demonstrate the effectiveness and flexibility of our approach. We plan to apply the framework to other network applications as well as to target multimedia applications. Second, our current formulation ignores the arbitration overhead when computing the communication access time. This can be a significant source of error when there are a large number of masters on a bus. We plan to extend the framework to include arbitration overhead to eliminate this source of error. Lastly, our framework allows the designer to guide the exploration process by inserting additional constraints in the formulation. While the ILP formulation is already flexible enough to do this, we plan to add an easy to use interface so that these types of constraints can be added systematically.

7. ACKNOWLEDGMENTS

We thank Akash Deshpande of Teja Systems for suggesting the investigation of soft multiprocessor systems. We also thank André DeHon for his guidance and comments. Finally, we thank the anonymous reviewers for their valuable suggestions.

8. REFERENCES

[1] EEMBC. http://www.eembc.org/.

[2] ILOG CPLEX. http://www.ilog.com/products/cplex/.

[3] A. Atamtürk and M. W. Savelsbergh. Integer Programming Software Systems. Technical Report BCOL.03.01, IEOR, University of California at Berkeley, January 2003.

[4] Chris Rowen, Tensilica Inc. Fundamental Change in MPSoCs: A fifteen year outlook. In *MPSOC'03 Workshop Proceedings*. International Seminar on Application-Specific Multi-Processor SoC, 2003.

[5] M. Grajcar. Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System. In *In Proc. of the Design Automation Conference (DAC)*, volume 17, pages 280–285, June 1999.

[6] M. Gries and Y. Jin. Comprehensively Exploring the Design Space. In M. Gries and K. Keutzer, editors, *Building ASIPs: The MESCAL Methodology*, pages 131–178. Springer Inc., 2005.

[7] M. Gries and K. Keutzer, editors. *Building ASIPs: The MESCAL Methodology*. Springer Inc., 2005.

[8] P. D. Hoang and J. M. Rabaey. Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput. In *IEEE Transactions on Signal Processing*, volume 41, pages 2225–2235, June 1993.

[9] K. Ravindran, N. Satish, Y. Jin, and K. Keutzer. An FPGA-based Soft Multiprocessor System for IPv4 Packet Forwarding . In *International Conference on Field Programmable Logic and Applications (FPL)*, August 2005.

[10] M. Ruiz-Sánchez, E. Biersack, and W. Dabbous. Survey and Taxonomy of IP Address Lookup Algorithms. *Network, IEEE, Vol.15, Iss.2*, pages 8–23, March-April 2001.

[11] Xilinx, Inc. *Embedded Systems Tools Guide*, Xilinx Embedded Development Kit, EDK version 6.2i edition, June 2004.