Hardware/Software Partitioning and Pipelining

Smita Bakshi†
Department of Electrical & Computer Engineering
University of California
Davis, CA 95616

Daniel D. Gajski
Department of Information & Computer Science
University of California
Irvine, CA 92697

Abstract

For a given throughput constrained system-level specification, we present a design flow and an algorithm to select software (general purpose processors) and hardware components, and then partition and pipeline the specification amongst the selected components. This is done so as to best satisfy the throughput constraint at minimal hardware cost. Our ability to pipeline the design at several levels, enables us to attain high throughput designs, and also distinguishes our work from previously proposed hardware/software partitioning algorithms.

1 Introduction

Digital systems, especially in the domain of digital processing and telecommunications, are immensely complex. In order to deal with the high complexity, increased time-to-market pressures, and a set of possibly conflicting constraints, it is now imperative to involve design automation at the highest possible level. This “highest possible level” may vary on the design, but given the fact that an increasing number of designs now contain a combination of different component types, such as general-purpose processors, DSP (digital signal processing) cores, and custom designed ASICs (application specific integrated circuits), we consider the highest level of design to be the one involving the selection and interconnection of such components. We refer to this as system-level design.

The reason why a system is best composed of different component types is due to the different characteristics of the components, which may be targeted at satisfying different constraints. Off-the-shelf processors offer high-programmability, lower design time, and a comparatively lower cost and lower performance than an equivalent ASIC implementation. On the other hand ASICs are more expensive to design and fabricate, but offer comparatively higher performance. Thus, for a given system, these components are selected such that the performance critical sections are performed rapidly on ASICs, and the less critical sections, or the sections that require higher programmability, are performed on the processors.

Our work addresses throughput constrained systems. Given a specification of such a system, we select processors and hardware resources to implement the system and then partition and pipeline it amongst the selected components so as to best satisfy the given throughput constraint at minimal hardware cost. The throughput of a system is the rate at which it processes input data, and this is often the prime constraint on digital signal processing systems including most image processing applications. In order to meet the throughput constraints of these systems, it is not only sufficient to perform the critical sections in hardware, but it is also necessary to pipeline the design. Pipelining divides the design into concurrently executing stages, thus increasing its data rate. Our work supports pipelining at four different levels of the design, namely the system, behavior, loop and operation level.

Over the past five years, several co-design systems [1] [2] [3] [4] for hardware/software partitioning have been developed. However, these tools assume that the tasks in the system execute sequentially, in a non-pipelined fashion. Furthermore, they also assume that the operations within each task execute sequentially. In comparing our work with the synthesis systems mentioned above, we have, in general, extended the design space explored by these systems by allowing designs to be pipelined at the system and the task level. Hence, our work extends current algorithms by performing pipelining at several levels, and by so doing, it is capable of achieving partitions with high throughput values that are unattainable without pipelining.

2 Problem definition

Our problem of hardware/software partitioning and pipelining may be defined as follows:

Given:

1. A specification of the system as a control flow graph
(CFG) of behaviors or tasks.

2. A hardware library containing functional units characterized by a three-tuple <type, cost, delay>.

3. A software (processor) library containing a list of processors characterized by a four-tuple <type, clock speed, dollar cost, metrics file>.

4. A clock constraint and a throughput constraint for the complete specification.

**Determine:**

1. An implementation type (either software or hardware) for every behavior.

2. The estimated area for each hardware behavior, as well as the total hardware area for the complete specification.

3. The processor to be used for each software behavior, as well as the total number of processors for the complete specification.

4. A division of the control flow graph into pipe stages of delay no more than the given throughput constraint.

Such that:

1. Constraints on throughput are satisfied, and
2. Total hardware area (for the given clock) is minimized.

The throughput constraint specifies the difference in the arrival time in nanoseconds of two consecutive input samples. We also refer to this time as the PS (pipe stage) delay, since this would be the required delay of a pipe stage in the design, if it were to be pipelined.

The hardware library consists of a set of functional units with their corresponding delay (in ns) and area (in gates) and the software library contains a list of processors with their corresponding clock speeds, dollar cost and metrics file. The metrics file gives the number of instruction cycles and the number of bytes required to execute each of a list of 3-address generic instructions on that processor. This information characterizes a processor and is required to estimate the execution time of a behavior on a specific processor [5].

The output consists of a pipelined and partitioned CFG where every behavior has been mapped to either hardware or software and the graph has been divided into pipe stages of delay no more than 4000 ns. Every hardware behavior is associated with an estimate of its execution time and the number and type of components (selected from the hardware library) needed to obtain that execution time. For instance, behavior $E$ has a throughput of 4000 ns and requires 3 instances of $Mpy1$, 1 instance of $Mpy2$ and 2 instances of $Add2$, bringing the total area to 430 gates. Similarly, every software behavior is associated with a processor from the software library, and its execution time on that processor. For instance, behavior $A$ implemented on the Pentium processor, has an execution time of 3100 ns.

Finally, the CFG has also been partitioned into three pipe stages such that the throughput of the system is 4000 ns, that is each pipe stage has a delay of no more than 4000 ns. The hardware and software partitioning and pipelining has been done with the aim of satisfying the throughput constraint at minimal hardware cost. This scheduling and pipelining information is represented in the System Pipeline diagram, which depicts all the pipe stages and the execution schedule of behaviors within each pipe stage.

Before we describe our algorithm, note some assumptions of our model:

1. The system architecture contains processors, ASICs (application specific integrated circuits), and memory chips that all communicate over buses. The memory stores data that needs to be transferred between pipe stages as well as any globally defined data that may be accessed by multiple processors and/or ASICs. In this paper, we assume that all hardware behaviors are mapped onto 1 ASIC. After the pipelining and the partitioning, this ASIC may be further partitioned into smaller ASICs [6] [7].

2. Two software behaviors may share the same processor, irrespective of the pipe stages they execute in.

3. Two hardware behaviors may only share resources if they execute sequentially in the same pipe stage.

### 3 Algorithm

An overview of our algorithm for hardware/software partitioning and pipelining is presented in Figure 2. Given a SpecChart [1] specification, hardware and software libraries, a throughput and clock constraint, the first step consists of deriving the control flow graph from the given specification. We then estimate [5] the execution time of all behaviors on
all the available processors in the software library. This gives us the PET (processor execution time) table, an example of which is shown in Figure 3 for the given CFG and software library in Figure 1. Based on the assumption that a software implementation is always less costly than an equivalent hardware implementation for a given behavior, our algorithm attempts to execute as many as possible behaviors on processors. Thus, any behavior that has an execution time less than the given throughput constraint on at least one processor can be executed in software and only those behaviors that have an execution time greater than the throughput constraint on all processors need be executed in hardware.

For instance, for the example in Figure 3 behaviors A, B and D can be executed on a processor and behaviors C and E need to be executed in hardware.

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Processor</th>
<th>Execution Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>pentium</td>
<td>3100</td>
</tr>
<tr>
<td>A</td>
<td>68000</td>
<td>3800</td>
</tr>
<tr>
<td>B</td>
<td>pentium</td>
<td>4200</td>
</tr>
<tr>
<td>B</td>
<td>68000</td>
<td>2200</td>
</tr>
<tr>
<td>C</td>
<td>powerPC</td>
<td>18900</td>
</tr>
<tr>
<td>C</td>
<td>68000</td>
<td>19800</td>
</tr>
<tr>
<td>D</td>
<td>Pentium</td>
<td>900</td>
</tr>
<tr>
<td>D</td>
<td>68000</td>
<td>12200</td>
</tr>
<tr>
<td>E</td>
<td>Pentium</td>
<td>14870</td>
</tr>
<tr>
<td>E</td>
<td>68000</td>
<td>21080</td>
</tr>
</tbody>
</table>

Figure 3: Step 2: hardware/software partition.

4 Experimental results

We have integrated the hardware/software partitioning and pipelining algorithm within SpecSyn [1], a system synthesis tool. Our experiments are designed to evaluate the quality of the hardware estimation and the quality of the
hardware/software partitioning and pipelining algorithm. We present a synopsis of our results.

The quality of the hardware estimation was evaluated by comparing manually obtained designs of the IDCT, the FFT, and 6 blocks from the MPEG II decoder, against those obtained by the estimation algorithm. In general, for a given throughput constraint, our estimates were within 10% of the area of the manual designs. The estimation errors were mainly because our algorithm does not handle multi-functional units, and is currently incapable of using multiple bitwidth implementations of the same component type.

The quality of the hardware/software partitioning and pipelining algorithm was evaluated by comparing the manual design exploration process against the results obtained by our algorithm, for the MPEG II decoder example. The manual design process started with an all software non-pipelined design and then moved the critical behaviors to hardware, as well as pipelined the system, till its throughput was within about 3000 ns (this represents a decoding rate of 30 frames per second). Similarly, we ran our algorithm for a range of PS delay constraints (700,000 ns to 3,000 ns), starting from an all software solution and moving towards an all hardware solution. Results of the comparison are shown graphically (in part) in Figure 4.

The results indicate that the design exploration conducted by our algorithm closely matches the manual exploration. The consistent difference in area between designs on the two curves is because our designs do not include the controller area, whereas the manual designs do. Despite the inaccuracy of our estimates, its fidelity is high, indicating that, with further improvements, it will be feasible to replace the manual exploration process by our algorithm.

The results also indicate that the designs obtained by our algorithm, while in some cases able to share the same processor amongst 2 behaviors, hence requiring a fewer number of processors. (Note that the Pentium processor was selected to be the best from a library of about 6 processors, including the Sparc, PowerPC, and 68000 processors). More importantly, the results show that the fastest design attainable by our algorithm (2980 ns) is about 25% faster than that obtained by the manual design. This is because our algorithm performs pipelining at the system, behavior, loop and operation levels, which is difficult for designers to perform manually.

It is important to note that while the manual design and exploration took approximately 6 man months, our algorithm took approximately 3 minutes per design, with a total of about 30 minutes on a SUN SPARC 5 workstation. Details of these experiments are provided in [8] and [9]. Given the similarity of both results, this is a significant saving in design time.

5 Conclusion

Our design flow and algorithms may be improved in several ways. At the behavior level, our resource estimation algorithm can be improved by allowing the use of multi-functional components, by allowing multiple bitwidths of the same component type by extending our algorithms to pipeline in the presence of loop-carried dependencies, and by providing estimates for the cost of multiplexers and buses. Our design model and algorithms can support these extensions and incorporating these extensions is part of ongoing work. At the system level, we can improve the pipelining and partitioning algorithms by incorporating some measure of processor cost into our cost metric, by taking interface cost and delay into account, as well as by partitioning the system amongst multiple ASICs.

Our current results indicate that, with these modifications, our algorithm will serve as a practical solution to the hardware/software partitioning problem. We believe this will greatly increase design quality and reduce design time.

References