INTEGRATED COUPLING AND CLOCK FREQUENCY ASSIGNMENT OF ACCELERATORS DURING HARDWARE/SOFTWARE PARTITIONING

Scott Sirowy and Frank Vahid*
Department of Computer Science and Engineering - University of California, Riverside
*Also with the Center for Embedded Computer Systems, University of California, Irvine

Abstract: Hardware/software partitioning moves software kernels from a microprocessor to custom hardware accelerators. We consider advanced implementation options for accelerators, greatly increasing the partitioning solution space. One option tightly or loosely couples each accelerator with the microprocessor. Another option assigns a clock frequency to each accelerator, with a limit on the number of distinct frequencies. We previously presented efficient optimal solutions to each of those sub-problems independently. In this paper, we introduce heuristics to solve the two sub-problems in an integrated manner. The heuristics run in just seconds for large examples, yielding 2x additional speedup versus the independent solutions, for a total average speedup 5x greater than partitioning with a single coupling and single frequency.

1. INTRODUCTION

Partitioning an application's kernels to execute on a custom hardware accelerator rather than on a microprocessor—known as hardware/software partitioning—is a well-known technique for improving application performance [1] and improving energy consumption [2]. Such partitioning is relevant to both ASIC (application-specific integrated circuit) and FPGA (field-programmable gate array) implementation. The rise of FPGAs in commercial microprocessor platforms [3] makes such partitioning increasingly important.

Sirowy, S., Vahid, F, 2007, in IFIP International Federation for Information Processing, Volume 231, Embedded System Design: Topics, Techniques and Trends, eds. A. Rettberg, Zanella, M., Dömer, R., Gerstlauer, A., Rammig, F., (Boston: Springer), pp. 145–154.

Most previous hardware/software partitioning approaches did not consider different couplings of the accelerators with the microprocessor. However, modern platforms, including FPGAs, support at least two couplings. *Tightly coupled* accelerators have direct access to the microprocessor memory or cache, and thus operate at a single clock frequency, which will necessarily be the slowest frequency of any of those accelerators. *Loosely coupled* accelerators instead access memory through a bridge, and thus may each have unique optimized clock frequencies. Thus, there exists a tradeoff to couple an accelerator tightly or loosely based on the importance of sinlge cycle memory access or running at the fastest possible clock frequency. Figure 1 shows a typical architecture that supports multiple couplings. The two tightly coupled accelerators have single cycle access to memory at the expense of both being clocked at 58 MHz even though one could have been clocked at 166 MHz. We refer to the problem of coupling a set of accelerators tightly or loosely as the *two-level microprocessor-accelerator partitioning problem.*

Modern platforms, including FPGAs, may support several different frequencies on a single chip. For example, the Xilinx Spartan 3 supports four distinct clock frequencies, while the Xilinx Virtex II supports up to eight [4]. Much current research investigates methods to take advantage of multiple clock domains for heterogeneous core architectures, systems-on-a-chip, etc., for both performance and energy benefits [5]. However, the number of accelerators often exceed the number of available clock frequencies. In this case, the accelerators must be grouped to share clock frequencies, necessarily running at the slowest frequency of the group. For example, in Figure 1, the four loosely coupled accelerators must share two clock frequencies. We refer to the problem of assigning a fixed number of clock frequencies for minimal application execution time as the *clock frequency assignment problem*.

Most previous approaches do not consider clock frequency assignment for the accelerators. While the tightly coupled accelerators should all execute using the same frequency, the loosely coupled accelerators could potentially each execute with different frequencies. In previous work, we solved the coupling assignment problem optimally, assuming enough available clock frequencies to support unique frequencies for each loosely coupled accelerator [6]. In a separate work, we solved the problem of assigning a limited number of frequencies to the set of loosely coupled accelerators such that performance was maximized [7]. In this work, we show that solving the two problems in an integrated manner can yield significant performance improvements over solving them sequentially

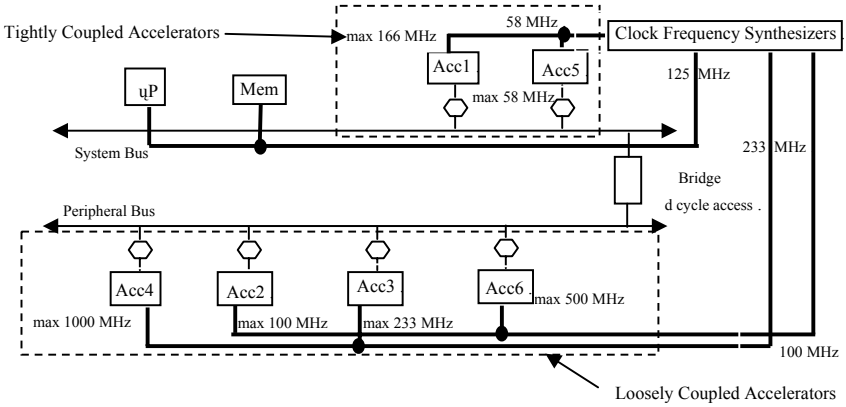

Figure 1. A two-level system architecture that is driven by four clock frequencies. The system bus has two tightly coupled accelerators that run at a slower clock frequency but have single cycle access to memory.

2. PROBLEM DEFINITION AND DESCRIPTION

We previously solved the two-level microprocessor-accelerator partitioning problem and the clock frequency assignment problem optimally using novel dynamic programming techniques for each. This section reviews those solutions, and then defines a new problem integrating both problems.

2.1 Two-Level Microprocessor-Accelerator Partitioning

The problem of partitioning accelerators to either a tightly coupled set or a loosely coupled set, assuming that each loosely coupled accelerator could run at its own unique clock frequency, used the following objective function for minimizing the execution time of all the accelerators:

$$TC([\sum_{i=1}^{n}(comp\_cycles_i + mem\_accesses_i)]/\min\_clock)$$

$$+ LC(d * \sum(mem\_accesses_i / clk\_freq_i)$$

$$+ \sum_{i=1}^{n}(comp\_cycles_i)/clk\_freq_i))$$

We solved our problem optimally using a novel dynamic programming algorithm we refer to as the *n-knapsack dynamic programming*, or ***NKDP***, solution. A complete description of the solution is given in [6].

2.2 Clock Frequency Assignment Partitioning

In the clock frequency assignment problem, we again considered a set of accelerators *A* which had already been determined by a previous hardware/software partitioning decision. Given a maximum number of unique clock frequencies *F* available to the accelerators, the *clock-frequency assignment problem* is to:

Find a positive integer value for every a_i.freq, such that each a_i.freq is less than a_i.maxfreq for every i, the number of distinct a_i.freq values is less than or equal to F, and the execution time E is minimized.

We also developed a novel dynamic programming algorithm to solve the clock partitioning problem optimally. The complete solution description is given in [7].

2.3 Integrated Two-Level Partitioning and Clock Frequency Assignment

The integrated coupling and clock frequency assignment problem takes as input a set of functions to be implemented as accelerators, determined by a previous hardware/software partitioning decision (our problem and partitioning may iterate). Each accelerator is annotated with four numbers, determined from synthesis and simulation of each function: The number of memory accesses, the total number of computation cycles, the synthesized area, and the maximum possible clock frequency. The number of memory accesses and computation cycles may represent averages or worst-case numbers, depending on whether the designer seeks to optimize for overall average or worst-case performance.

The extra cycles of the bridge is also given. This memory access penalty is an architectural feature of the bridge, and not a per-application number, so the number is fixed for all applications. A loosely coupled accelerator would incur this latency penalty each time it made an access to memory, since the accelerator is connected to the memory through the bridge.

All tightly coupled accelerators, having single-cycle access to memory or cache, must run at a single clock frequency – this assumption matches several modern commercial FPGAs that incorporate microprocessors. Because all those accelerators must run at one clock frequency, they all must run at the frequency of the *slowest* tightly coupled accelerator in the group.

The tightly coupled accelerators' frequency need not be the same as the microprocessor's frequency.

Loosely coupled accelerators, in contrast, could potentially run at their unique, fastest clock frequency. However, since modern FPGA platforms impose a limit on the number of available clock frequencies, several of the loosely coupled accelerators may also need to be merged together and share the same clock frequency. This means several of the accelerators will not be able to run at their own unique clock frequency. The number of available clock frequencies F is usually given in the documentation for the particular FPGA being used. For instance, a Xilinx Spartan 3 board supports up to four unique clock frequencies, while the Xilinx Virtex II supports up to eight clock frequencies.

Formally, the problem takes as input a set of accelerators $A=\{a_1,a_2,\ldots,a_n\}$. Each function is annotated with several different weights: $a_i.comp\_cycles$, $a_i.mem\_accesses$, $a_i.area$, $a_i.max\_freq$, and $a_i.frequency$. The term $a_i.frequency$ is not given and must be determined. The memory access penalty through a bridge is given as d, and the number of available clock frequencies is given as F. The objective function is to thus minimize the application execution time as follows:

Find a positive integer value for every $a_i.freq$, such that each $a_i.freq$ is less than $a_i.maxfreq$ for every i, the number of distinct $a_i.freq$ values is less than or equal to F, one group has single cycle access to memory while the rest have d cycle access, and the execution time E is minimized.

3. HEURISTICS

We present two heuristics to solve the clock frequency assignment problem for two-level microprocessor-accelerator platforms. Before that, a straightforward *sequential* approach performs two-level microprocessor-accelerator assignment first assuming unlimited distinct clock frequencies, followed by clock frequency assignment on the loosely coupled accelerators with (F-1) clock frequencies (since one clock frequency must necessarily be used for the tightly coupled accelerators). Each sub-problem can be solved optimally using our previous techniques.

Because the running time of NKDP is $O(Sn^2)$, where S is the area constraint, and the running time of the clock frequency assignment algorithm is $O(nF^2)$, the overall worst case time complexity of the sequential approach is $O(Sn^2 + nF^2)$. However, since the assumption that the two-level microprocessor-accelerator partitioning algorithm can operate every loosely coupled accelerator at its own distinct clock frequency is potentially

violated, the two level partitioning becomes suboptimal, and therefore the entire solution is suboptimal.

3.1 No Penalty Migration

Our first heuristic was based on the observation that when the ***NKDP*** algorithm partitions the accelerators into both a tightly coupled and loosely coupled set, there may be accelerators in the loosely coupled set that are clocked with a faster maximum frequency than the tightly coupled set. This is because the NKDP algorithm decided that having a faster frequency was more important than having single cycle access to memory. However, with the number of clocks constrained in clock frequency assignment, that accelerator's frequency may be reduced below the tightly coupled clock set frequency. Thus, migrating the accelerator from the loosely coupled set to the tightly coupled set makes sense (assuming it fits the area constraint) since the accelerator would run faster as a tightly coupled accelerator than merged with a slower accelerator in the loosely coupled set. Because the accelerator's fastest possible frequency is faster than the already established tightly coupled set clock frequency, the heuristic can migrate the accelerator to the tightly coupled set at no penalty to the tightly coupled set. We call this *No Penalty Migration*. After the heuristic migrates an accelerator from the loosely coupled set to the tightly coupled set, clock frequency assignment is again run on the remaining accelerators in the loosely coupled set to determine if a new assignment exists, since one less accelerator may result in a better partitioning of the available clock frequencies to the remaining loosely coupled accelerators.

3.2 Nested Dynamic Programming

We also developed a heuristic in which we tried to integrate the two solutions by having the two-level microprocessor-accelerator algorithm call the clock frequency assignment algorithm each time the knapsack algorithm returns a possible solution. We call this the *Nested Dynamic Programming* heuristic. The *No Penalty Migration* heuristic assumes the initial two-level microprocessor-accelerator partitioning chose the best two-level assignment, meaning the tightly coupled frequency should be maintained.

However, the clock frequency assigned to the tightly coupled accelerators may not be optimal when considering the clock frequency assignment problem too, and thus no amount of clock frequency assignment and migration on the remaining accelerators would result in the optimal solution. Because the two-level microprocessor-accelerator dynamic programming

algorithm runs knapsack *n* times, resulting in *n* potential solutions, running the clock frequency assignment dynamic programming algorithm on each of those solutions would result in a more accurate solution space, since more options are allowed into the tightly coupled accelerator set.

The solution to each knapsack is passed to the clock frequency partitioning algorithm. The clock frequency assignment algorithm determines the clock frequency assignment for the loosely coupled accelerators. The best solution is maintained and returned. We note the "best" solution is returned as opposed to the "optimal" solution from the original NKDP algorithm, because the heuristic still potentially violates the assumption that the NKDP algorithm assumes each loosely coupled accelerator can run at its own distinct clock frequency. The heuristic is only guaranteed to return optimal results when the number of clock frequencies exceeds the number of accelerators that require a distinct clock frequency. The worst case running time of the nested dynamic programming heuristic is also $O(n^2 (S + F^2))$, since the nested dynamic programming algorithms run the clock frequency assignment algorithm *n* times.

4. EXPERIMENTS AND RESULTS

This section describes results of applying the two heuristics to a commercial quality H.264 video decoder from Freescale Semiconductor. We implemented the heuristics on a 2.66 GHz 1GB RAM Pentium 4 PC. We targeted synthesis to a Xilinx IV Pro, and gathered information on cycles per function and maximum clock frequency of each accelerator. We also tested our heuristics using a wide range of synthetic benchmarks.

H.264 is a proprietary video decoder developed by the Video Coding Experts Group (VCEG), and part of the MPEG-4 standard. Unlike common benchmarks taken from publicly available reference implementations, the decoder's code was highly optimized, and thus did not consist of just two or three critical functions, but rather of 42 critical functions that together accounted for about 90% of execution time. We utilized Stitt's partitioning into accelerators [9], which was straightforward, involving implementing an accelerator for each critical function. We gathered computation cycle and memory access information through synthesis and simulation, and clocked each accelerator targeted for Xilinx's Virtex IV Pro. The variation in maximum frequencies ranged from 40 MHz to 285 MHz.

Figure 2 shows the results running the heuristics on the highly optimized H.264 video decoder. The speedups are normalized to results when all accelerators use only one clock frequency and one coupling. Figure 2 shows that one additional clock frequency allowed the heuristics to couple the 42

accelerators either tightly or loosely, and thus gain a 3.5x speedup over the single frequency, single coupled implementation. The inclusion of additional clock frequencies further improves the speedup to almost 4x. For the H.264 application, the *No Penalty Migration* and *Nested Dynamic Programming* heuristics performed similarly, attaining almost the same speedup. Although both heuristics have the same worst case runtime, the *No Penalty Migration* heuristic consistently attained results faster than the *Nested Dynamic Programming* heuristic. We also note that as the number of clock frequencies increases, the improvements of both the *No Penalty Migration* and *Nested Dynamic Programming* heuristics compared to the sequential approach become almost negligible.

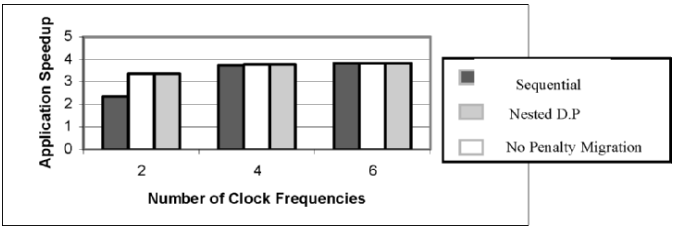

Figure 2. Results of the heuristics on a commercial quality video decoder. Compared to a single-frequency, single-coupling implementation of the accelerators, the heuristics improve the execution time by almost 4x.

To further test our heuristics, we applied our heuristics to several synthetic examples, which included a wide range of accelerators. Each example supported a large range of computation cycles, memory accesses, and clock frequencies. Figure 3 highlights results of comparing the *No Penalty Migration* and *Nested Dynamic Programming* heuristics to an implementation that did not consider coupling or multiple clock frequencies. Figure 3(a) shows the benefit of just including one additional clock frequency, and thus introducing the ability to tightly or loosely couple each accelerator. With only two clock frequencies, Figure 3(a) shows the heuristics are able to achieve on average 4x speedup. Note that in every case the *Nested Dynamic Programming* heuristic finds the best partitioning of the accelerators. The *Nested Dynamic Programming* heuristic also took the longest to complete, finishing many seconds later in the larger examples. The *No Penalty Migration* heuristic yielded an average 15% improvement in application running time over the straightforward sequential approach. The *Nested Dynamic Programming* heuristic gained an additional 15% improvement over *No Penalty Migration*. This was because both the sequential search and *No Penalty Migration* partitioned several accelerators to the tightly coupled set without knowledge of the fact that there were only

two clock frequencies available. The *Nested Dynamic Programming* heuristic was able to test all combinations of accelerators in the tightly coupled set, and therefore was able to find a superior solution.

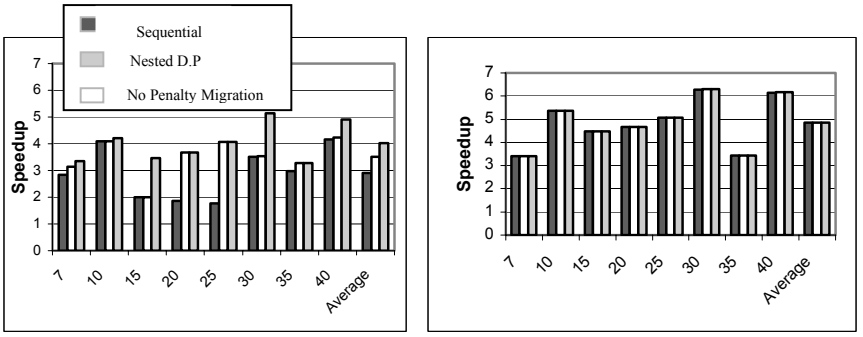

Figure 3. Application speedups for synthetic examples with varying numbers of accelerators: (a) two clock frequencies, (b) eight clock frequencies. Substantial speedup is achieved for increasing numbers of clock frequencies compared to single-frequency, single-coupling implementations.

However, as the number of clock frequencies increased, the heuristics achieved nearly the same speedups. The reason is because as the number of available clocks increase, it is more likely that the initial partitioning of the tightly coupled set is correct, meaning only minor gains could be made over a straightforward sequential search. On average across 2 to 6 clock frequencies, *No Penalty Migration* yielded a 5% improvement over a sequential search, while *Nested Dynamic Programming* provided a 10% improvement. Comparing Figures 3(a) and 3(b), one sees that additional available clock frequencies does improve speedups over single-coupled single-frequency partitions, from an average of 4x in (a) to nearly 5x in (b), with one example achieving almost 6.5x performance improvement.

For all the examples, our heuristics ran in seconds, compared to an exhaustive search, which did not complete in any reasonable amount of time when the number of accelerators exceeded fifteen. The *No Penalty Migration* heuristic completed its search consistently faster than *Nested Dynamic Programming* heuristic while also finding a better solution for platforms with only a few available clock frequencies. However, the *Nested Dynamic Programming* heuristic might be much easier to implement in a framework where coupling and clock assignments have already been implemented.

5. CONCLUSION

We showed that the consideration of both coupling and multiple clock frequencies can lead to substantial speedup over an application implementation that does not consider either. We also showed that the integration of both coupling and multiple clock frequencies can lead to application speedups of over 5x compared to a single-coupling single-frequency implementation. We developed two new heuristics that integrated coupling and clock frequency assignment, running in just seconds.

ACKNOWLEDGEMENTS

This work was supported by grants from the National Science Foundation (CNS-0614957) and the Semiconductor Research Corporation (2005-HJ-1331), and by donations from Xilinx, Inc. Freescale Semiconductor supplied the commercial H.264 decoder

REFERENCES

1. Gupta, R. and G. De Micheli. Hardware-Software Cosynthesis For Digital Systems. IEEE Design and Test of Computers. Pages 29-41, September 1993
2. Henkel, J. A low power hardware/software partitioning approach for core-based embedded systems. In Proceedings of the 36th ACM/IEEE Design Automation Conference, 122–127, 1999.
3. Corp. 2005. FPSLIC (AVR with FPGA), http://www.atmel.com/products/FPSLIC/.
4. Virtex II and IV. Xilinx Corp., http://www.xilinx.com
5. Hu, J., Y. Shin, N. Dhanwada, and R. Marculescu. Architecting Voltage Islands in Core-Based System-on-a-Chip Designs. Int. Symp. on Low Power Electronics and Design (ISLPED), 2004, pp. 180-185.
6. Sirowy, S., Y. Wu, S. Lonardi, and F.Vahid. Two-Level Microprocessor-Accelerator Partitioning. Design and Test Europe(DATE) 2007.
7. Sirowy, S., Y. Wu, S. Lonardi, and F. Vahid. Clock-Frequency Assignment for Multiple Clock Domain Systems-on-a-Chip. Design and Test in Europe(DATE). 2007.
8. Lengauer, T. 1990. Combinatorial Algorithms for Integrated Circuit Layout. John Wiley & Sons, Inc., New York, NY.
9. Stitt, G., F. Vahid, G. McGregor, B. Einloth Hardware/Software Partitioning of Software Binaries: A Case Study of H.264 Decode. Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES/ISSS), Sep. 2005