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][2][3][4][5][6] and improving energy consumption [7][8][9][10]. 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

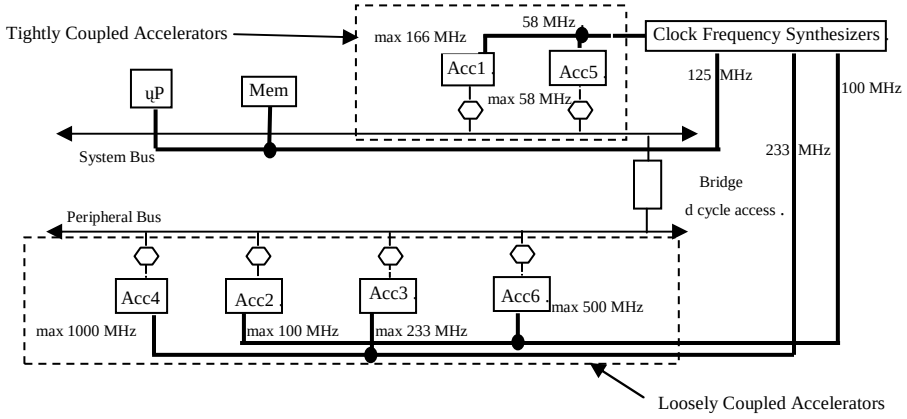
[11][12][13][14][15][16][17] makes such partitioning increasingly important.

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 to have single cycle memory access or to run 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[17]. 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 [18][19][20][21][22]. 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 so as to minimize the application's 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 [23]. 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 [24].

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.



In this work, we show that solving the two problems in an integrated manner can yield significant performance improvements over solving them sequentially. Section 2 discusses the problem definition and our previous sub-problem solutions. Section 3 provides two new heuristics to solve the integrated coupling and clock frequency assignment problem. Section 4 gives results.

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 of minimizing the execution time of all the accelerators:

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

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

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

min_clock is the speed at which the tightly coupled set must run to guarantee single cycle access to memory (or cache). The *d* term for the loosely coupled set is the memory latency penalty incurred for accessing memory through a bridge. The tradeoff is whether it is advantageous for an accelerator to have single cycle access to memory at the expense of possibly being clocked at a slower frequency, versus being run at its own fastest possible clock frequency but with a memory access penalty through a bridge. We developed an exhaustive optimal search algorithm (which was too slow for practical sized examples), and a greedy search heuristic. The greedy heuristic begins by mapping all the accelerators to the loosely coupled set, and migrates the accelerators to the tightly coupled set based on the accelerator's contribution to the total execution time and how many memory accesses it requires. The greedy heuristic achieved performance 15% slower than optimal.

Seeking a fast solution with better results, we eventually developed an optimal solution that runs in what is known as psuedo-polynomial time (the partitioning problem is known to be NP-complete [25], so a truly polynomial-time solution is not practical). The key to our solution is the idea that the two-level accelerator partitioning problem with *n* functions can be decomposed into *n* 0-1 knapsack problems. In the classic 0-1 knapsack problem, the goal is to choose a subset of the items whose total value is maximized while at the same time the sum of the weights does not violate the constraint on the overall capacity given the value and the weight of *n* items to be stored, and the capacity of the knapsack *S*. This problem is NP-complete, but can be solved optimally with a dynamic programming approach using a well-known pseudo-polynomial time.

We refer to our solution as the *n-knapsack dynamic programming*, or ***NKDP***, solution. The idea is that if we knew the slowest accelerator in the tightly coupled set (let the accelerator be X), we can optimally map all the functions to the tightly and loosely coupled sets.

First, we map X to the tightly coupled set, since based on our assumption, this is the slowest accelerator in the tightly coupled set. We then map all accelerators slower than X to the loosely coupled set, because otherwise we would violate the assumption that X was the slowest accelerator in the tightly coupled set. For all functions that are as fast or faster than X, we

compute the reduction in the function's execution time should it be mapped to the tightly coupled set as opposed to the loosely coupled set. We can do this because we know what its clock frequency would be if it were mapped to the tightly coupled set (the same as X), or to the loosely coupled set (the accelerator's given frequency). Note that the reduction in execution time can be negative, which means mapping the function to the tightly coupled set will lengthen its execution time. If that happens, the function is mapped to the loosely coupled set immediately. The remaining accelerators can then be mapped to either the tightly or loosely coupled sets using the classic 0-1 knapsack problem. The weight of each item is the size of the accelerator, the value of each item is the reduction of the function's execution time calculated in the previous step, and the capacity of the knapsack is the area constraint of the overall tightly coupled group minus the area of X.

The above steps will yield the optimum solution if X is known. Of course, we do not know X in advance, but that does not matter since we can try all the possible choices of X. For each function, we assume the function is X, and we run the above steps to obtain a locally-optimal solution. Among all the locally-optimal solutions thus obtained, the one that has the minimum overall execution time must be globally optimal.

2.2 Clock Frequency Assignment Partitioning

In the clock frequency assignment partitioning problem, we again considered a set of accelerators *A* which had already been determined by a previous hardware/software partitioning decision, and for which different clock frequencies could be assigned (thus corresponding to the loosely coupled processors of the previous problem). For each accelerator a_i in the set *A*, we are given several weights. The weight $a_i.cycles$ corresponds to the number of clock cycles that the accelerator contributes to the total clock cycles for the application, not including cycles required for accessing memory. The weight $a_i.maxfreq$ represents the fastest clock frequency at which this accelerator may execute. That frequency would typically be determined by synthesizing the accelerator and then taking the inverse of the critical path. The weight $a_i.freq$ represents the frequency at which accelerator a_i is being clocked in an implementation. This number is not given, but rather must be determined. The determined number must be less than or equal to $a_i.maxfreq$.

The application's execution time *E* is the sum of the application's computation time and communication time. The computation time equals the *cycles* multiplied by 1/*freq* values for every accelerator. The communication time equals the total number of memory accesses multiplied by the memory access time. We originally included communication time in our problem

formulation, but found that component of time unnecessary to include during clock-frequency assignment. The reason is that communication time equals the number of memory accesses by each accelerator times the time associated with each access, which is essentially invariant. The time associated with each access consisted of two parts, one part dependent on the accelerator's frequency and hence foldable into the accelerator's compute time, and the other part independent of the accelerator frequencies, instead dependent on the frequency of the microprocessor and memory, which do not impact the *relative* total execution time of a given partitioning. Note that this non-overlapping computation/communication model of execution time, while different from the model uses in multi-processor based hardware/software partitioning, holds for accelerator-based hardware/software partitioning.

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 found there existed enough substructure in the problem to develop a fast and efficient dynamic programming algorithm that could solve the clock frequency partitioning problem optimally.We assume (without loss of generality) that accelerators a_1, a_2... a_M, are pre-sorted in decreasing order of maximum frequency and each frequency is unique. Let *X(A,C)* equal the total execution time of the first *A* accelerators using the first *C* clock frequencies. We define the following recurrence relation as a function:

If (A=0) then X(A,C)=0
Else If (C=0) then X(A,C)= infinity
Else

$$X(A,C) = Min_{i=1}^{A}\left(\frac{\sum_{j=i}^{A} a_j.cycles}{a_A.mxfreq} + X(i-1, C-1)\right)$$

If A=0, there are no accelerators, and thus the execution time is 0. If C=0, there are no clock frequencies available, so execution time is infinite. We intentionally define X to return 0 for X(0,0).

The "Min" term compares the alternative solutions that assume the present accelerator's (a_A) maximum frequency is assigned to the present accelerator only, to the present accelerator and the next accelerator, to the

present accelerator and the next two accelerators, etc. The expression inside that term computes the total execution time for this cell as the sum of the execution times for the accelerators assigned to the present maximum frequency, added to the previously-computed best solution for the other accelerators with one less available clock frequencies.

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 solving 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 $O(Sn^2 + nF^2)$. This is because the sequential approach runs each algorithm exactly once. In every case, the Sn^2 term would dominate the nF^2 term, meaning the real complexity is $O(Sn^2)$. 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.

Because the clock frequency assignment algorithm is running a maximum of *n* times (if we have to migrate every single accelerator from the loosely coupled set to the tightly coupled set), the overall worst case time complexity is $O(n^2 (S + F^2))$.

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, and comes up with a potential solution *n* times, 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 being 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 [6], which was straightforward, involving implementing an accelerator for each critical function. We gathered computation cycle and memory access information through simulation and synthesis, 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 very 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. This is because as the number of clock frequencies increases, the more "correct" the original partitioning of the accelerators to the tightly coupled and loosely coupled sets becomes, and therefore little additional work is needed. Even for an application as large as H.264, every heuristic ran in seconds, making its inclusion into a larger scale exploration environment feasible. However, the *No Penalty Migration* heuristic would fare better in real time or dynamic exploration environments.

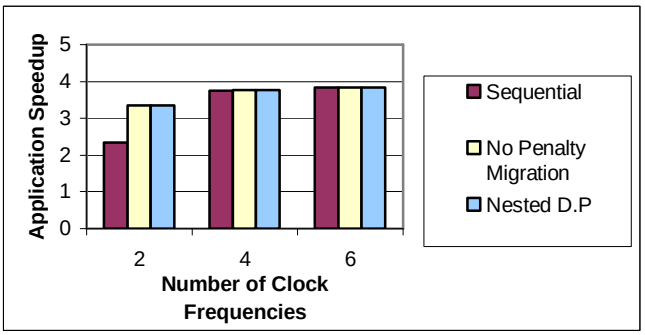


Figure 2. Results of the heuristics on a commercial quality video decoder. Compared to a single-frequency, single-coupling implementation of the accelerators, the heuristics improve the execution time by almost 4x.

To further test our heuristics, we applied our heuristics to several synthetic examples, which included a wide range of accelerators. Each accelerator in turn 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 of about 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.

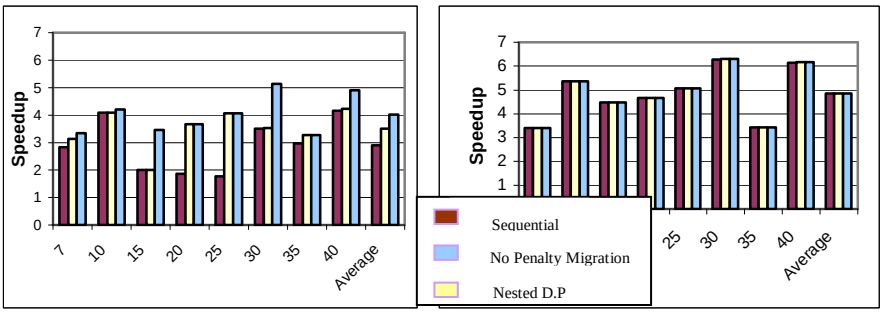


Figure 3. Application speedups for synthetic examples with varying numbers of accelerators: (a) two clock frequencies, (b) eight clock frequencies. Substantial speedup is achieved for increasing numbers of clock frequencies compared to single-frequency, single-coupling implementations.

However, as the number of clock frequencies increased, the heuristics achieved nearly the same speedups. The reason is because as the number of clocks increases, the more likely that the initial partitioning of the tightly coupled set was 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. Of course, we tradeoff speed for the exact solution given by an exhaustive search.

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. CONCLUSIONS AND FUTURE WORK

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.

Our formulation assumed mutually exclusive memory accesses and computation. However, in many cases, these two activities may actually overlap. We plan on extending the integrated coupling and clock frequency assignment problem to handle concurrent memory accesses and computation, which will require a more advanced communication and architecture model. We also plan on searching for an optimal solution to the integrated two-level partitioning and clock frequency assignment problem.

6. 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

7. REFERENCES

1. Banerjee, S. and N. Dutt. Efficient search space exploration for HW-SW Partitioning. Int. Symp. on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2004.
2. Eles, P., Z. Peng, K. Kuchcinsky, and A. Doboli. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. Design Automation for Embedded Systems, vol2, no 1, 5-32 Jan. 1997

3. Gupta, R. and G. De Micheli. Hardware-Software Cosynthesis For Digital Systems. IEEE Design and Test of Computers. Pages 29-41, September 1993
4. Kalavade, A. and P.A. Subrahmanyam. Hardware/software partitioning for multi-function systems. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1997.
5. Miyamori, T., and U. Olukotun. A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications. FPGAs for Custom Computing Machines (FCCM). 1998, pp. 2 – 11.
6. 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.
7. Chattopadhyay, A., and Z. Zilic. GALDS: A Complete Framework for Designing Multiclock ASICs and SoCs. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 6, June 2005.
8. Cohn, J.M., D.W. Stout, P.S. Zuchowski, S.W. Gould, T.R. Bednar, and D.E. Lackey. Managing power and performance for System-on-Chip designs using Voltage Islands. Int. Conf. on Computer-Aided Design (ICCAD), 2002, pp. 195-202.
9. 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
10. Stitt, G., F. Vahid, and S. Nematbakshi. Energy Savings and Speedups From Partitioning Critical Software Loops to Hardware in Embedded Systems. IEEE Transactions on Embedded Computer Systems, January 2004.
11. Corp. 2005. FPSLIC (AVR with FPGA), http://www.atmel.com/products/FPSLIC/.
12. Celoxica. http://www.celoxica.com.
13. Cray XD1. Cray Supercomputers. http://www.cray.com/products/xd1/index.html.
14. Critical Blue. 2005. http://www.criticalblue.com
15. Excalibur. Altera Corp., http://www.altera.com
16. Mimosys. http:// http://www.mimosys.com/.
17. Virtex II and IV. Xilinx Corp., http://www.xilinx.com
18. 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.
19. Kumar, R., K.I. Farkas, N.P. Jouppi, P. Ranganathan, and D.M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. Int. Symposium on Microarchitecture (MICRO), 2003.
20. Muttersbach, J., T Villiger, H Kaeslin, N Felber, and W. Fichtner. Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-Chip Systems. IEEE Int. ASIC/SOC Conference, 1999.
21. Semeraro, G., G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. Int. Symp. on High-Performance Computer Architecture (HPCA), 2002.
22. Zhang, Y., XS Hu, and DZ Chen. Task scheduling and voltage selection for energy minimization. Design Automation Conference, 2002
23. Sirowy, S., Y. Wu, S. Lonardi, and F.Vahid. Two-Level Microprocessor-Accelerator Partitioning. Design and Test Europe(DATE) 2007.
24. 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.
25. Lengauer, T. 1990. Combinatorial Algorithms for Integrated Circuit Layout. John Wiley & Sons, Inc., New York, NY.