Codex-dp: Co-design of Communicating Systems Using Dynamic Programming*

Jui-Ming Chang
Cadence Design Systems, Inc.
555 River Oaks Parkway, M/S 2B1
San Jose, CA 95134

Massoud Pedram
Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089

Abstract

In this paper, we present a novel algorithm based on dynamic programming with binning to find, subject to a given deadline, the minimum-cost coarse-grain hardware/software partitioning and mapping of communicating processes in a generalized task graph. The task graph includes computational processes which communicate with each other by means of blocking/nonblocking communication mechanisms at times including, but also other than, the beginning or end of their lifetime. The proposed algorithm has been implemented. Experimental results are reported and discussed.

1 Introduction

Previous work on system level synthesis has focused mainly on fine-grain hardware/software partitioning. Examples include Vulcan II [1] and Cosyma [2]. These programs automatically partition the input specification into basic blocks (or fine-grain operations) and move the basic blocks to hardware or software components while satisfying the given constraints. The resulting fine-grain partitioning may, however, move logically coherent blocks across different parts or put logically unrelated blocks in the same part. Furthermore, the resulting partitioning creates an implementation which is very different from the initial specification, and hence, is not convenient for human designers to debug or improve upon.

In contrast, coarse-grain partitioning does not decompose the initial specification into basic blocks and does not assign a process in the initial specification to several processors. It is therefore able to preserve the granularity and modularity of the initial specification. In addition, coarse-grain partitioning can exploit the designers' expertise more easily and can achieve a desired partitioning which satisfies some macroscopic choices more readily [3]. Finally, the resulting solution has more logical coherence which facilitates the top-down design process and allows for debugging of the hardware/software.

Many of the coarse-grain partitioning algorithms start from a task graph which consists of a set of communicating processes. In the published literature, task graphs that describe the set of communicating processes (or tasks) are directed acyclic graphs (DAGs) which use nodes to represent processes and arcs to represent precedence relations or communication among the processes. In these task graphs, the communication is assumed to take place from the end of one process (node) to the beginning of another process. We refer to this type of communication as *end/begin* communication. The coarse-grain processes may however communicate with each other at times other than the end or the beginning of their lifetimes. We refer to this type of communication as *midway* communication and to the task graph with midway communication as a *generalized task graph*. The problem we attempt to solve is then stated as follows:

Problem 1.1 *Given a generalized task graph consisting of processes which communicate with each other by arbitrary blocking/nonblocking communication mechanisms and a library containing several possible mappings (or implementations) for each process, simultaneously schedule and map the computational and communication processes to HW/SW resources so as to minimize the total area cost while satisfying a given deadline.*

The cost of mapping a process to a library unit (implementation) cannot be determined exactly because of the possibility of sharing the same unit between different processes (e.g., using time-division multiplexing or TDM for short). The cost should account for this possibility and include the area and delay overhead associated with the *context switching*. We assume in this paper that TDM will be used whenever possible, and that the overhead of the context switching is accounted for in the area/delay cost of processes which share the same unit.

A task graph with midway communication becomes a directed multi-graph, i.e., there may exist multiple arcs from one node to another node. The task graph may be periodic. We can handle the case where the period is no less than the deadline by performing the same schedule on every period.

The hardware components which are available in the library can be classified into computational or communication units. Both classes can be further divided into programmable or non-programmable. Examples of programmable computational units are CPUs, DSPs and examples of non-programmable computational units are ASICs and custom ICs. Examples of programmable communication units are FIFOs with controllers, bidirectional handshake controllers, DMA controllers, bus arbiters, or shared memory access and examples of non-programmable communication units are special purpose, customized communication units. All computational and communication units

*This research was supported in part by SRC under contract no. 98-DJ-606.

in our library are assumed to be compatible with industry interface standards such as the evolving *Virtual Socket Interface*. As a result, we can mix and match various IP blocks.

We allow the resource sharing of programmable components by different processes according to TDM, even if the process lifetimes overlap. For nonprogrammable resources, the sharing can only happen if the process lifetimes do not overlap or the processes are mutually exclusive.

Our algorithm consists of three major phases. First, processes are decomposed into subprocesses which perform parts of the required computation. The correct precedence relationships implied by the specified communication mechanism are then added in by a systematic transformation process. Second, the decomposed subprocesses are scheduled so as to ensure that the subprocesses which belong to the same original process are mapped to the same hardware type (for example, the same CPU with the same utilization factor). We refer to the condition that all of the subprocesses which are obtained from the same original process are mapped to the same hardware type with the same utilization factor as *type consistency constraint*. This constraint is necessary because we assume that the original coarse-grain process has strong internal communication (variable reference, etc.). As a result, we do not want the subprocesses which are decomposed from the same original coarse grain process to be mapped to different hardware units in the final solution. The scheduling is done using a *dynamic programming* based algorithm which finds the cost-optimal process mapping, while satisfying a given task deadline. The third phase is a hardware allocation and binding (sharing) phase which ensures that the decomposed subprocesses will be mapped not only to the same hardware type, but also to the same hardware instance. The allocation and binding determines the sharing of hardware among all coarse-grain processes in the system.

The paper is organized as follows. In Section 2, we summarize related work for coarse-grain HW/SW partitioning. Section 3 introduces our transformation rules for process decomposition. In Section 4, we present our dynamic programming algorithm for solving Problem 1.1. In Section 5, we describe the allocation and binding algorithm to be used after the scheduling step. Experimental results and conclusions are provided in Sections 6 and 7, respectively.

2 Related work

There are two published works [4] [5] on fine-grain hardware/software partitioning which use dynamic programming. In both of these works, the target architecture contains a single microprocessor and a single hardware chip. The authors try to find the best combination of non-overlapping sequences of fine-grain "basic scheduling blocks" which fit the available hardware (ASIC or FPGA) and result in maximum speedup by moving the scheduling blocks from software to hardware). The problem is similar to the *knapsack problem*, and the dynamic programming formulation is used only to avoid repeated computations in their iterative procedure.

There have been a number of research publications on coarse-grain HW/SW partitioning which handle task graphs with only end/begin type communication [6] [7] [8] [9]. In these task graphs, the total time used by a process is simply the summation of the time used to do the computation and time used to do the communication. These works use greedy heuristics [6], branch and bound search [7], or MILP solvers [10] [8] as their optimization techniques. In [11] and [1], the authors allow midway communication in a fine-grain HW/SW environment. The form of communication allowed is however not as general as the ones proposed in the present paper and not at the coarse-level.

The work reported in [12] for coarse-grain system synthesis, separates the synthesis of computational and communication processes into two distinct stages. In this case, it is very difficult to apply a timing constraint (deadline) on the system because part of the time in the critical path is used to do the computation whereas another part is used to do the communication. In [6], the gradient search method on the solution space is used. In each iteration, the authors perform a *generate and test* operation commonly used in AI. That is, in each iteration, they try to relocate one process from a CPU to another, relocate a message (communication process) from one bus to another, do the rescheduling on the CPUs and buses, and calculate the change on the cost. If the timing constraints on CPUs or buses are violated, they add one more CPU or bus to fix the problem. The synthesis of computational and communication processes can thus be considered to be performed *simultaneously* during each iteration of the search on the solution space. The algorithm is, however, greedy and non-optimal. In our work, the timing constraint is applied to all of the computation and communication subprocesses in all critical paths and thus the synthesis of the two kinds of processes is performed simultaneously. Since our algorithm is based on dynamic programming it produces the optimal solution.

In summary, previous works do not address the problem of coarse-grain mapping of communicating processes. This problem is the subject of the present paper.

3 Process Decomposition in a Task Graph

This phase decomposes the communicating processes into some smaller computational subprocesses and communication processes. The decomposition step ensures that all of the precedence relationships imposed by the required blocking/nonblocking communication mechanisms are added. Transformation of the communicating processes into computational subprocesses and communication processes for blocking send/blocking receive, nonblocking send/blocking receive, blocking send/nonblocking receive and nonblocking send/nonblocking receive are shown in Fig. 1(a), (b), (c) and (d), respectively. In Fig. 1, process S represents the actual process which sends the data from the sending process and R represents the process which sends the *reply* or *acknowledgment* from the receiving process. The arcs with single tail denote the precedence relationships between the nodes connected by that arc. The arcs with double tails denotes the precedence relationship between the two subprocesses that are decomposed from the same original coarse-grain process. Note that there is strong internal communication (variable accesses) and logical coherence between two such subprocesses and thus they should be finally mapped to the same hardware/software instance.

For a task graph with complex communications among processes, we follow the transformation rules shown in Fig.

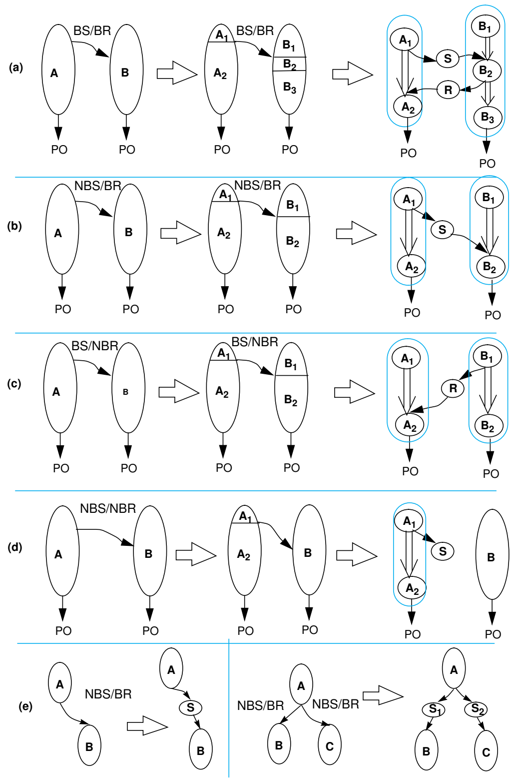

Figure 1: Decomposition of communicating processes.

1 to create to the decomposed task graph. When there is more than one midway communication for a given process (cf. process A in Fig. 2(a)), the decomposition of this process depends on whether it is single threaded or multi-threaded. For a single threaded process, the midway communication is referenced to the same time line as that of the thread. In this case, the appropriate transformation rules are applied to all midway communications at different points of the time line (cf. Fig. 2(b)). For a process with multi-threads, the midway communications may be referenced to different time lines for different threads. In this case, we add two dummy nodes Y_1 and Y_2 (with zero cost and zero delay) at the beginning and the end of that process to synchronize the multiple threads. The appropriate transformation rule is then applied on each thread that serves the time line for the corresponding midway communication (cf. Fig. 2(c)).

4 Scheduling Using Dynamic Programming

The scheduling algorithm is based on dynamic programming as is described next.

4.1 Area vs. delay curves

Before the scheduling, all processes are assigned an area vs. delay curve which represents the area cost and delay for mapping the process to different types of processors. The corner points on those curves are non-inferior points. A point is inferior to another point if both its cost and delay are equal or higher. The area cost of a process mapped to a processor type X is the chip area of the hardware realization of processor X. In case the utilization factor is less

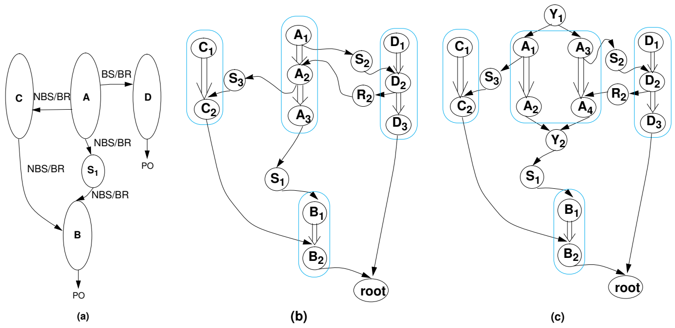

Figure 2: Example for decomposition of single and multi-threaded processes.

than 100%, then the area cost is multiplied by the utilization factor. Similarly, the delay cost of a process mapped to this processor is the total computation time for the process running on that type of hardware. In case the processor is shared among multiple processes, the delay cost of each process accounts for the overhead of context switching.

In this paper, we only consider a task graph which is composed of computational and communication processes with deterministic characteristics. The data size for each communication process is known (*a priori*) as part of the input specification, and the corresponding delay for mapping to different communication units is estimated by behavioral simulation and profiling. For communication processes, the area estimate does include the area used by communication controller, buses, and local buffers for both the sender and the receiver. The area of a communication process that uses programmable communication controller with some utilization factor $< 100\%$ is estimated as the total cost times the utilization factor. For communication units, which are shared by several communication processes, the cost and delay includes the overhead of context switching.

4.2 Simple task graphs

For a task graph without re-convergent fanout and with only end/begin type communications, the algorithm used in [13] can be directly used without going through the process decomposition phase. This algorithm would then produce the optimal hardware/software mapping for a tree-structured task graph (and a good solution for a DAG-structured task graph) under a given timing constraint (deadline) in *pseudo-polynomial* time. The only modification is to replace the end/begin type communication with a sending process S and to add the required arcs to the task graph as shown in Fig. 1(e).

The algorithm assumes that we are given the area vs. delay curves for different module alternatives (implementations) which match each node of the task graph. Then the algorithm performs a post-order traversal which adds the area vs. delay curves of the children of a node and the module alternatives for the node to build the area vs. delay curve of this node. This step will also use the lower bound merge to delete all inferior points. The post-order traversal will continue until the graph roots are reached. Then a pre-order traversal will commence at the roots using user specified arrival time constraint. The minimum area point on the area vs. delay curve of the root which satisfies the arrival time

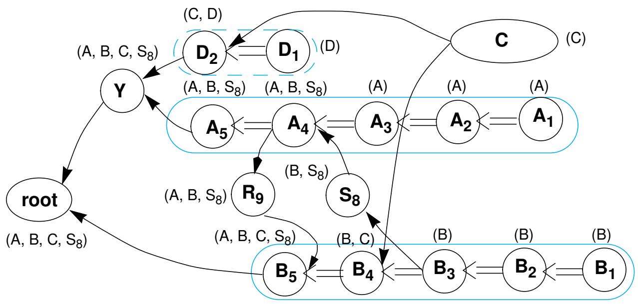

Figure 3: Example for computing the binning strings (shown in parantheses).

constraint will determine the module alternative to be used at the root. The pre-order then traverses the children of the root with the new arrival time constraint calculated as the arrival time at the root minus the delay of the module used at root. The recursive procedure will continue until all leaves have been visited.

4.3 Complex task graphs

Handling task graphs with processes that have reconvergent fanout and use midway communication during their lifetime is more difficult. This is because processes in the task graph have to be decomposed into subprocesses, and the communication processes which reflect the blocking/nonblocking communication mechanism have to be inserted. Furthermore, after the decomposition phase, the dynamic programming paradigm must be modified to ensure that the subprocesses which belong to the same original process are mapped to the same hardware or software component instance to maintain the logical coherence and performance. This is achieved in two steps; during scheduling, we ensure that the decomposed subprocesses which correspond to the same original process are mapped to the same HW or SW type with the same utilization factor. During the allocation and binding, we ensure that these subprocesses are further mapped to the same HW or SW component instance.

Theorem 4.3.1 *Problem 1.1 is* NP-complete.[1]

In practice, the midway communication among coarse-grain processes occurs frequently. The dynamic programming approach of [13], may generate a point on the area vs. delay curve of a reconvergent fanout node x which requires inconsistent type assignments for some of the nodes in the transitive fanin cone of x. This is obviously wrong. In addition, using the original algorithm in [13], during the post-order graph traversal, we may drop some points which are actually required to generate the optimal solution.

4.3.1 Creating the binning strings

To satisfy the type consistency constraint, we modify the dynamic programming algorithm as follows. First, we add binning strings to each node. The purpose of the binning strings is to ensure that the dynamic programming algorithm uses type consistent mapping solutions for nodes along different paths to any reconvergent fanout node (see Fig. 3). The computation of the binning strings relies on the notion of primary and secondary reconvergent nodes of a given node in the graph and uses the Floyd-Warshall algorithm [15] to compute the transitive closure of a graph. Details are omitted due to lack of space. Second, in the solution of [13], the post-order and pre-order traversals are performed on the individual PO's sequentially. This approach may however lead to a type inconsistent solution. We thus add some dummy nodes and a root with zero cost and zero delay to merge different $PO's$ into a single root.

As a result of the binning string computation, each node (subprocess) will have several bins, and each bin will have an associated **tag** which describes the implementations used for each process in the binning string of the node. For example if the binning string of node X is (A, B) and if there are 3 types of mapping for process A and 4 types of mapping for process B, then there will be a total 12 bins for node X. The first bin will be tagged as (A = T1, B = T1), the second bin will be tagged as (A = T1, B = T2), and so on.

4.3.2 Post-order traversal

Suppose we are processing node X with two children Y and Z (which have already been processed during the post-order traversal). We check the binning strings of Y and Z against that of X. If the binning strings of any child and its parent are different, we have to normalize the dimension of the bins of the child to that of the parent. For a child node with binning string shorter than that of its parent node, we expand the dimension of the bins of the child node by duplicating the corresponding curve for the bins which are added. For example, if child node Y has binning string (B) and its parent X has binning string (A, B), and assuming that there are two types of mappings for both A and B, say types E and F. We duplicate the original curve of node Y for ($B = E$) tag and create two identical curves for tags ($A = E, B = E$) and ($A = F, B = E$). Similar duplication step is applied to the curve of node Y for the ($B = F$) tag. For a child node with longer binning string than that of its parent node, we reduce the dimension of the bins of that child node by merging the curves which belong to bins that differ only in the ID missing from the binning string of the parent. For example, if child node Z has binning string (A, B, C) and its parent X has binning string (A, B), then we will do a superimpose followed by lower bound operation on the curves of bins corresponding to tags $(A = E, B = E, C = E)$ and $(A = E, B = E, C = F)$ to obtain the unified curve for the new tag $(A = E, B = E)$. Similar operations are needed for all other combinations of A and B implementations.

After we normalize the dimension of each child node, the curve representing the accumulated cost vs. delay on the parent can be constructed by adding the curves of each child and including the contribution of the module alternative matched at the parent. This must be done for every bin, one at a time. Addition must occur in the common region among all curves to ensure that the resulting merged function reflects feasible matches at the children of n. The curve for successive matchings at the same node n are then merged by applying a *lower-bound merge* operation on the corresponding curves. Because our decomposed task graph

[1] All proofs can be found in [14]; they are omitted here to save space.

is a DAG instead of a tree, we face the problem of how to pass up the cost of a multiple fanout node to its parents during the post-order traversal. We use a heuristic whereby the cost value of a multiple fanout node is divided by its fanout count when propagated upward in the DAG. This heuristic produces the **exact** total cost at the root as long as multiple primary outputs are merged into a single root (cf. 4.3.1). The proof is straight forward (similar to flow conservation in network flow problem).

The curve addition and merging are performed recursively until the root of the root is reached. The resulting curve is saved in the corresponding bin of the graph at its corresponding node. The set of (t, c) pairs corresponding to the composite curve for the tag at the root node gives the set of all possible arrival time-cost trade-offs for the user to choose from.

4.3.3 Pre-order traversal

Pre-order traversal begins at the root of the decomposed task graph and proceeds toward the leaves. Consider a node X of the graph. The (output) arrival time and the type constraint for the node are known. Our task is to determine the arrival times and the type constraints for each of its child nodes.

Consider a child Z of node X. We are assured that at least one of the tagged curves of X is consistent with the type constraint passed down to X. If there is exactly one such curve stored at X, we pick the minimum-cost point of the curve which satisfies the arrival time constraint of X. Otherwise, there are more than one tagged curves that are consistent with the type constraints passed down to node X. In this case, we find the corresponding best cost point on each curve (which satisfies the timing constraint) and among them pick the solution which has the overall minimum-cost. Next, we update the type constraint for node Z as the Union of type constraint passed down to node X and the constraint implied by the tag of the chosen point on the tagged curve (or bin) and set the timing constraint of Z as the timing constraint at X minus the delay of the match at X.

A multiple-fanout node is visited multiple times during the pre-order traversal. During each visit, the arrival time and possibly type constraint of the node may change to guarantee that arrival time and type consistency constraints for all paths emanating from that node toward the root of the graph are satisfied. Due to the introduction of a single root during the binning string computation, we do not encounter conflicting type consistency constraints from different fanout branches of such a node.

Theorem 4.3.2 *Dynamic programming with binning solves Problem 1.1 optimally while satisfying the type consistency constraints.*

4.4 Complexity Analysis

Let us scale delay values for all nodes (subprocesses) under different process mapping to become integers. Furthermore, we denote the maximum computation time for a tree-structured decomposed task graph (using the worst-case integer delay values on any path) by T_{max} and assume that T_{max} is bounded from above by an integer Q. Let $\mid \mathcal{I} \mid = n$ where n is the total number of nodes (*decomposed* computation and communication (sub)processes) in the decomposed task graph .

Suppose that the maximum number of possible process mappings for each subprocess (node) is K and the maximum length among all binning strings is m. Using our process decomposition method, all communication (sub)processes have fanout count ≤ 1. In a decomposed task graph with n nodes, $0 \leq m < n$. There can be at most K^m bins in each node in the decomposed task graph. Then the maximum possible number of points in each node of the decomposed task graph is $Q \cdot K^m$.

The number of area-delay points on each node in the decomposed task graph is bounded from above by $Q \cdot K^m$. The algorithm thus has a time complexity of $n \cdot Q \cdot K^m$. Delay function merging and addition are done in linear time in the number of points on the curves involved in the operations. Therefore, our algorithm runs in $n \cdot Q \cdot K^m$ time.

Note that the value of m is, in general, dependent on the structure of the decomposed task graph. In the worst case, m can be as large as n. In practice, m is, however, much smaller than n. For example, for the frequently encountered simple task graphs defined in Section 4.2, m is zero. In this case, the algorithm has pseudo-polynomial time complexity on the decomposed task graph.

5 Allocation and Binding

As a result of the scheduling phase, the computational subprocesses decomposed from the same original coarse-grain process are mapped to the same type of processor implementation or custom ICs. They have not however been mapped to the same instance of the processor or custom IC.

Our first step is to *regroup* these subprocesses back into their coarse-grain process and assign them to the same processor instance. From then on, the allocation and binding procedure will treat the regrouped subprocesses as a single process.

Processes are generally separated into different classes if they are mapped to different types of hardware units. Within each class, the allocation and binding (sharing) is then performed.

Processes which are mapped to programmable units can share the same instance of the unit through TDM even if their lifetimes overlap. In addition to the programmable communication units, part of the buses or the shared memory and/or local buffers needed for communication may be shared in a TDM fashion by the the corresponding communication processes. The requirements for sharing one programmable unit instance are that the processes are mapped to the same type of unit, and the sum of the utilization factors of those processes is less than 100%. We perform the allocation and binding by using a *modified bin packing algorithm* which ensures that every regrouped coarse-grain process is bound to the same hardware instance throughout its lifetime. Note that the TDM sharing does not change the global timing obtained in the scheduling phase. Details are omitted to save space.

For non-programmable units such as custom ICs or other communication units, sharing is possible only if the process lifetimes do not overlap or the processes are mutually exclusive.

6 Experimental Results

Our dynamic programming with binning, named Codex-dp (for Co-design of Communicating Systems Using Dynamic Programming), is implemented in C and tested on a number of circuits. Experimental results are presented in Table 1. Prakash1 and Prakash2 are taken from [10]; Yen is taken from [6], and Bender is an example from [8]. In every example, we do the process decomposition and insert appropriate communication processes in the original task graphs. Our module library consists of a number of processors and communication units (Intel Pentium II and Motorola 68030 processors, TI 302C25 DSP, Intel DMA controller, 10Mb/s Ethernet controller, etc.) We allow the sharing of these resource through TDM. A pre-processing step determines the area/delay cost of each process when it is mapped to various hardware units in the library.

We also report results on 4 more examples from various sources. The task graph for example 1 with deadline = $80.0(ms)$ is taken from the CPM system [16]. The task graph for example 2 with deadline = $100.0(ms)$ contains end.begin type communication only, but has reconvergent fanout structure. The task graph for example 3 is shown in Fig. 2(a); its decomposed task graph is shown in Fig. 2(c) and has deadline = $50.0(ms)$. The task graph for example 4 is a large task graph taken from [16] which performs the voice activity detection in a GSM phone. For this example, we used three different deadlines and report the results in row ex4-1, ex4-2, ex4-3. The corresponding deadlines were set to 170, 300, 510 (ms), respectively.

In Table 1, column 2 shows the values of m and n seen by Codex-dp (cf. 4.4). Column 3 gives the total number of processor and communication units needed after the allocation and binding. Columns 4 and 5 give the CPU time used by Codex-dp (in $seconds$ on a 200 MHz Pentium Pro) and the estimated area cost (in cm^2) required to implement each circuit. Column 6 gives the numbers of variables, inequalities and equations if the scheduling is formulated as a mixed integer linear program, MILP assuming that each process has four possible implementations (not presented here due to lack of space). In column 7, we show the complexity of using exhaustive search after regrouping all of the computational subprocesses back into their original coarse-grain processes and still assuming that each process has four possible implementations.

It can be seen that Codex-dp produces the optimal scheduling results in a very short time compared to the expected time for MILP or exhaustive search. For example, in ex4, the MILP solution cannot be obtained due to the large number of variables in the MILP (195 variables, 123 equations, 51 inequalities). The row entries for ex4-1, ex4-2, ex4-3 show the trade-off between area cost and total computation time. Decreasing the deadline constraint increases the area cost of the optimal solution.

Ckt	m, n	pu,cu cnt	Codex-dp c-time	Codex-dp cost	MILP form.	Exh. srch
Prak1	2,11	1,1	0.036	63.7	55,13,11	4^{11}
Prak2	6,22	2,1	0.507	122.5	110,26,22	4^{22}
Yen	1,12	1,1	0.043	63.7	60,13,12	4^{12}
Bend.	7,13	2,2	1.143	137.2	60,17,12	4^{13}
ex1	0,13	1,1	0.086	79.7	65,12,13	4^{13}
ex2	1,14	2,1	0.114	148.5	70,15,0	4^{14}
ex3	4,18	3,1	0.107	171.5	63,21,7	4^{8}
ex4-1	9,39	3,3	6.329	200.9	195,51,123	4^{22}
ex4-2	9,39	2,3	6.412	151.9	195,51,123	4^{22}
ex4-3	9,39	2,2	6.356	127.4	195,51,123	4^{22}

Table 1: Experimental results.

7 Conclusion

We have presented an algorithm based on dynamic programming with binning to solve a min-cost, time-constrained simultaneous scheduling and mapping problem for a set of computational processes which communicated by means of blocking/nonblocking communication mechanism at times other than the beginning or end of their lifetime. The proposed algorithm produces optimal results, and is much faster to solve than the MILP formulation. A final resource allocation and sharing step will follow the dynamic programming step and produce the actual instantiation of the processor types to hardware instances. This last step is done using a modified bin packing heuristics.

References

[1] R. Gupta and G. D. Micheli. System-level Synthesis using Re-programmable Components. In *Proceedings European Design Automation Conference*, 1992.

[2] R. Ernst, J. Henkel, and Th. Benner. Hardware/Software Co-Synthesis for Microcontrollers. *IEEE Design and Test Magazine*, 10(4), December 1993.

[3] G. De Micheli and editor M. Sami. *Hardware/Software Co-Design, pages 22, 84, 85, 96, 217.* Kluwer Academic Publishers, 1995.

[4] A. Jantsch, P. Ellervee, J. Oberg, A. Hemani, and H. Tenhunen. Hardware/Software Partitioning and Minimizing Memory Interface Traffic. In *Proceedings European Design Automation Conference*, 1994.

[5] P. Knudsen and J. Madsen. PACE: A Dynamic Programming Algorithm for Hardware/Software Partitioning. In *Proceedings IEEE International Workshop on Hardware/Software Codesign*, 1996.

[6] T.-Y. Yen W. Wolf. Communication Synthesis for Distributed Embedded Systems. In *Proceedings IEEE International Conference on Computer-Aided Design*, 1995.

[7] J. D'Ambrosio and X. Hu. Configuration-Level Hardware/Software Partitioning for Real-Time Embedded Systems. In *Proceedings IEEE International Workshop on Hardware/Software Codesign*, 1994.

[8] A. Bender. MILP Based Task Mapping for Heterogenous Multiprocessor Systems. In *Proceedings European Design Automation Conference*, 1996.

[9] S. Narayan and D.D Gajski. Synthesis of System-Level Bus Interface. In *Proceedings European Design Automation Conference*, 1994.

[10] S. Prakash and A. Parker. SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems. *Journal of Parallel and Distributed Computing*, 16, December 1992.

[11] T. Benner, R. Ernst, and A. Ősterling. Scalable Performance Scheduling for Hardware-software Co-synthesis. In *Proceedings European Design Automation Conference*, 1995.

[12] J.-M. Davaeu, T. Ismail, and A.A. Jerraya. Synthesis of System-Level Communication by Allocation-Based Approach. In *Proceedings IEEE International Symposium on System Synthesis*, 1995.

[13] J.-M. Chang and M. Pedram. Energy Minimization Using Multiple Supply Voltages. In *Proceedings International Symposium for Low Power Electronic and Design*, August 1996.

[14] J.-M. Chang and M. Pedram. Codex-dp: Co-design of Communicating Systems Using Dynamic Programming. Technical Report CENG 98-04, University of Southern California, 1998.

[15] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. The MIT Press, McGraw-Hill, 1990.

[16] R. Steele. *Mobile Radio Communications*. Pentech Press, London, 1995.