MAGELLAN: Multiway Hardware-Software Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs *

Karam S. Chatha
Department of ECECS, ML 30
University of Cincinnati,
Cincinnati, OH 45221-0030
kchatha@ececs.uc.edu

Ranga Vemuri
Department of ECECS, ML 30
University of Cincinnati,
Cincinnati, OH 45221-0030
ranga@ececs.uc.edu

ABSTRACT

The paper presents MAGELLAN, a heuristic technique for mapping hierarchical control-dataflow task graph specifications on heterogeneous architecture templates. The architecture can consist of multiple hardware and software processing elements as specified by the user. The objective of the technique is to minimize the worst case latency of the task graph subject to the area constraints on the architecture. The technique uses an iterative approach consisting of closely linked hardware-software partitioner and scheduler. Both the partitioner and scheduler operate on the task graph in a hierarchical top down manner. The technique optimizes deterministic loop constructs by applying clustering, unrolling and pipelining. The technique considers speculative execution for conditional constructs. The number of actual hardware/software implementations of a function in the task graph are also optimized by the technique. The effectiveness of the technique is demonstrated by a case study of an image compression algorithm.

1. INTRODUCTION

System-level hardware-software (HW-SW) codesign is an effective methodology for designing embedded systems. HW-SW codesign consists of two basic design stages: mapping or partitioning of an application on to architecture elements and scheduling the execution of application components. The paper presents a heuristic technique called MAGELLAN for mapping an embedded system specification on to a user specified architecture template.

Application Domain: MAGELLAN is aimed at design of computation intensive applications like JPEG and MPEG algorithms. These applications can be specified as a set of coarse granularity tasks. The applications are characterized by predominantly data dependences between tasks with few control flow constructs. At a finer granularity the control constructs consist mainly of loops. At a coarse granularity level the applications can contain feed back control loops for adaptive behavior.

*This work was partially supported by the ARPA RASSP program and US-AF, Wright Lab, under contract numbers F33615-93-C-1316 and F33615-97-C-1043 respectively.

CODES '2001 Copenhagen, Denmark

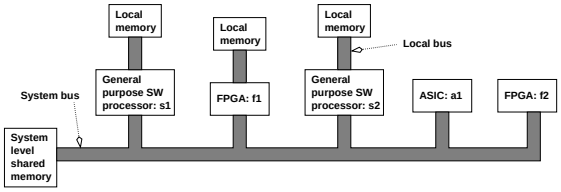

Figure 1: System Architecture

System Architecture: MAGELLAN implements the applications on a user specified architecture template. The template was developed based on existing (and proposed) implementations of the applications in our domain. The D30V MPEG multimedia processor [1] and SuperENC [2] (an MPEG-2 video encoder chip) consist of processor cores, dedicated logic and memory elements connected through a single system bus. Hence, MAGELLAN assumes that the system architecture consists of a single bus connected to shared memory and processing elements (Figure 1). The shared memory is used for inter-processor communication. The number and type of processing elements are specified by the user. The user specifies the area constraint ($\alpha(PE)$) for all the HW coprocessors. Additionally the user specifies the read (τ_{rd}) and write (τ_{wr}) times per data item for all the memory elements.

Application Specification: MAGELLAN models the application as a hierarchical control-dataflow task graph. The control behavior is described with the help of hierarchy. Hence, at each level of hierarchy the behavior of the application is described by a data flow task graph. The task graph is described with the help of ports, port maps, channels and tasks. MAGELLAN supports 5 basic types of tasks: leaf tasks, call tasks, case tasks, loop tasks and hierarchical tasks. Additionally it provides two other types of tasks: control tasks and counter tasks that can only exist as components of case or loop tasks. Figure 2 depicts a task graph that contains a loop task (T1), leaf task (T2) and case task (T3).

The ports are used for task data input/output. The channels specify data dependences between output port of one task and input ports of other tasks. Call tasks model function call behavior. Case tasks contains one control task that specifies the control behavior and a set of path tasks that execute in mutual exclusion based on the control behavior. A loop task can be of two types : deterministic loop task or non-deterministic loop task. The iteration count of a deterministic loop task is known a priori and is specified by the counter task. The loop behavior is specified by a loop body task. A non-deterministic loop task has unknown iteration count and it contains a control task instead of a counter task.

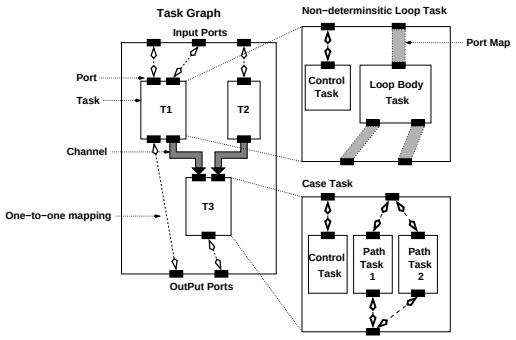

Figure 2: Application Specification

As the name suggests a hierarchical task models hierarchical abstraction.

The user specifies the size of the data transfered across each channel ($\omega(C)$) and dependence distance of each channel ($\delta(C)$). Associated with each leaf task v is an estimate set $\varepsilon(v) = \{\{PE_1, \mathcal{D}_1\}, \ldots, \{PE_i, \mathcal{D}_i\}$ where PE_i is a processing element and D_i is the set of estimated design points of the task on PE_i. The set D_i is given by:

$$\mathcal{D}_i = \begin{cases} \{\tau\} & \text{if } PE_i \text{ is SW} \\ \{\{\tau_1, \alpha_1\}, \ldots, \{\tau_j, \alpha_j\}, \ldots\} & \text{if } PE_i \text{ is HW} \end{cases}$$

where τ is the timing estimate and α is the area estimate. MAGELLAN accepts multiple design points for each task on each HW coprocessor.

Problem Description: The objective of MAGELLAN is to:

1. Map the hierarchical control-dataflow task graph on to architecture elements,
2. Select a design point for each leaf, counter or control task that is mapped to a HW coprocessor,
3. Schedule task execution, inter-processor communication on system bus and intra-processor communication,

such that the worst case execution time of the application is minimized subject to the area constraints on the various HW coprocessors.

The paper is organized as follows: Section 2 gives an overview of MAGELLAN, Section 3 discusses the previous work, Section 4 presents the scheduler, Section 5 discusses the partitioner and Section 6 presents experimental results and concludes the paper.

2. MAGELLAN: AN OVERVIEW

MAGELLAN uses the partitioner and scheduler in an iterative manner to obtain a mapping of the application on the system architecture (see Figure 3). Both the scheduler and partitioner apply a hierarchical top down approach. MAGELLAN calls the scheduler to generate an initial solution. The scheduler obtains an initial solution by mapping, optimizing and scheduling the application. The optimizations performed by the scheduler include loop unrolling, loop pipelining, speculative scheduling of path task components of a case task and optimizing the number of actual implementations of a call task. The partitioner tries to improve on the solution generated by the scheduler by trying out several moves. The partitioner moves vary depending upon the task type. The partitioner evaluates each move by invoking the scheduler. The scheduler maps and schedules the remaining tasks, and gives feedback to the partitioner in terms of resulting latency of the task graph. The partitioner tries several moves and implements the move that results in largest improvement over the initial solution. MAGELLAN continues in an iterative manner until the partitioner cannot find a move that results in an improvement. The algorithm then exits and outputs the solution.

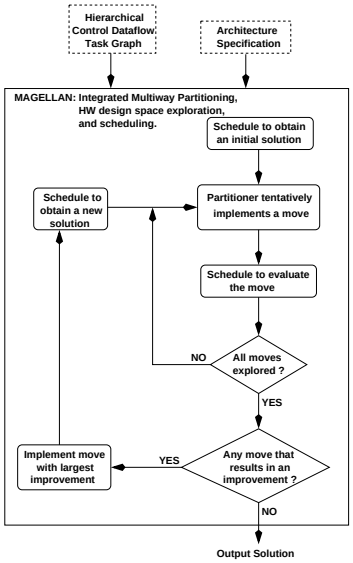

Figure 3: MAGELLAN: An overview

3. PREVIOUS WORK

Vulcan [3], Cosyma [4], Design assistant [5], Bakshi et al. [6], Marwedel et al. [7] and Jeon et al. [8] have proposed approaches for automated HW-SW partitioning. [3] and [4] map the nodes by using a partitioning algorithm based on a cost function. In contrast MAGELLAN uses an integrated scheduler with the partitioner which is much more accurate for performance evaluation. [5] and [6] use a modified list based scheduler for partitioning and scheduling. Their technique is limited due to the greedy approach adapted by list scheduling algorithm. MAGELLAN overcomes this limitation by using an iterative partitioner. [7] proposed an integer linear programming (ILP) approach for integrated HW-SW partitioning and scheduling. However, the ILP based approach is limited by the large run times. [8] require the user to specify number of pipeline stages and they apply loop pipelining before HW-SW partitioning. MAGELLAN optimizes the number of pipeline stages by using a modified retiming heuristic (RECOD [9]). In contrast to [8], MAGELLAN maps the loop tasks with an objective of maximizing their throughput and then applies pipelined scheduling. To the best of our knowledge none of the existing approaches on automated partitioning optimize a hierarchical control-dataflow task graph. Further none of these approaches partition on user specified architectures.

In contrast to approaches mentioned in the previous paragraph that partitioned on a fixed system architecture, Wolf et al. [10], COHRA [11], MOGAC [12] and COSYN [13] proposed approaches for architecture synthesis. [10] use a preemptive scheduling algorithm that is too expensive both in time and space for many high-volume low cost embedded systems [14]. Hence, MAGELLAN adapts a non-preemptive static scheduling approach. In contrast to MAGELLAN, MOGAC adapts a stochastic approach based on genetic al-

```
Algorithm Magellan_Scheduler (AR, TG⟨T⟩, S_table, S, I, τ_exec)
begin
    L_ready = ready_tasks(T)
    while (L_ready ≠ ∅)
        v = heuristic_select(L_ready)
        .
        .
        .
        if (tasktype(v) = HIERTASK)
            schedule_hiertask (AR, v⟨T⟩, S_table, S, I, τ^end_exec) endif
        .
        .
        .
        update(L_ready)
    endwhile
    τ_exec = τ^end_exec
end

Algorithm schedule_hiertask(AR, v⟨T⟩, S_table, S, I, τ^end_exec)
begin
    if (state(v) ≠ FIXED)
        τ^end_exec = ∞
        for all (PE^i_sw ∈ AR)
            explore(PE^i_sw, v⟨T⟩, S_table, I', τ')
            if (τ' < τ^end_exec) τ^end_exec = τ', I = I' endif
        endfor
        for all (eligible PE^i_hw ∈ AR)
            explore(PE^i_hw, v⟨T⟩, α(v, PE^i_hw), S_table, I', τ')
            if (τ' < τ^end_exec) τ^end_exec = τ', I = I' endif
        endfor
        Magellan_Scheduler (AR, v⟨T⟩, S_table, FALSE, I', τ')
        if (τ' < τ^end_exec) τ^end_exec = τ', I = I' endif
    endif
    if (S = TRUE) remove_temp(S_table), add(v, S_table)
    else temp_add(v, S_table) endif
end
```

Figure 4: Algorithm for scheduling

gorithm. While COSYN and COHRA also adapt a heuristic technique, they do not optimize hierarchical control-dataflow graphs.

4. SYSTEM SCHEDULER

MAGELLAN uses a modified list based scheduler for mapping and scheduling of the task graph specification. The scheduler selects a task from the ready based on: 1. The task is a non-deterministic loop task, 2. Urgency of the task [15]. The non-deterministic loop task is given higher priority since it is difficult to a priori estimate its total execution time.

A pseudo code for the scheduling algorithm is shown in the top half of Figure 4. In the function call $\mathcal{AR}$ is the architecture description, $\mathcal{TG}(\mathcal{T})$ is the task graph with task set $\mathcal{T}$, $\mathcal{S}_{table}$ is the schedule table, S is a boolean variable, $\mathcal{I}$ is the implementation set that records the processor and design point mapping and τ_{exec} is the latency of the task graph. S is $TRUE$ ($FALSE$) when the function is called for scheduling (design space exploration) by the partitioner (hierarchical task). The scheduler performs design space exploration for the hierarchical task by making temporary entries in the schedule table that are replaced at a later stage by the selected implementation. Based on the task type of the task that is selected from the ready list, the scheduler calls the appropriate sub-function. For example *schedule_hiertask*() as shown in the figure.

4.1 Scheduling of leaf, control and counter tasks

The scheduler decides the mapping of a leaf task based on its suitability and threshold suitability ($Th(PE_i)$) of the processing element [15]. Suitability of a leaf task for a particular HW design point or SW processor varies from 0 to 1. A value closer to 1 (0) makes the task highly suitable (unsuitable) to be mapped to the corresponding implementation. $Th(PE_i)$ has a value between 0 and 1. It is used to select tasks that are suitable to be mapped to the processing element PE_i. The scheduler considers mapping a leaf task to a HW coprocessor PE_i only if the suitability of at least one of the design points is greater than $Th(PE_i)$. The scheduler explores the following implementations:
1. Single HW: The scheduler maps the task to the highest suitability design point that satisfies the threshold value and area constraint on the corresponding HW coprocessor.
2. Single SW: If none of the HW design points of a task satisfy both the threshold suitabilities and area constraint on corresponding HW coprocessors, the scheduler explores SW implementation. It maps the task to the SW processor that has the highest suitability value.

4.2 Scheduling of a call task

The scheduler determines from the schedule table if there exists an implementation for the call task. An implementation can exist for the call task if another call task that exhibits the same behavior has already been scheduled. The new call task can be mapped to the existing implementation if the earliest possible schedule time of the new call is greater than the execution end time of the previous call. Hence, the scheduler tries to limit the actual number of physical implementations of the call task. In the case that the call task has not been implemented or there exists a scheduling conflict, the scheduler does design exploration based on the task type of the corresponding callee (function) task.

4.3 Task area constraint and implementation selection

The scheduler uses a hierarchical top down approach for design space exploration and scheduling of hierarchical, loop and case tasks. It associates an area constraint $\alpha(v, PE_i) = \frac{\alpha_{sum}(v)}{\alpha_{sum}(\mathcal{TG})} \cdot \alpha(PE_i)$ with each hierarchical, loop and case task v for every HW coprocessor PE_i. $\alpha_{sum}(v)$ ($\alpha_{sum}(\mathcal{TG})$) is the summation of median area design points of constituent tasks of v (task graph $\mathcal{TG}$) that satisfy the threshold suitability on PE_i. $\alpha(v, PE_i)$ is the area available to the scheduler for implementing the task v on PE_i. This area may be modified by the moves applied by the partitioner. The scheduler saves the results of design space exploration for the previous iteration. Hence, if the available area for the task on a HW coprocessor remains unchanged, the scheduler saves time by not repeating the exploration.

The scheduler selects the design alternative that gives the earliest end time as the implementation for the hierarchical, loop and case tasks.

4.4 Scheduling of a hierarchical task

The scheduler explores the following implementations:
1. Single SW: The scheduler maps the hierarchical task on each SW processor and calculates the end time for the task.
2. Single HW: The scheduler maps the hierarchical task on all eligible HW coprocessors and calculates the end time for the task. A HW coprocessor is considered eligible if majority of the constituent tasks of the hierarchical task satisfy

the threshold suitability of the coprocessor.
3. Multiple processor: The scheduler explores multiple processor implementation by making a recursive call on itself with the hierarchical task as argument.
A pseudo code description of the algorithm is shown in the bottom half of Figure 4. The function performs design space exploration if the mapping of the task is not $FIXED$. The function invokes $explore()$ to explore single HW and single SW implementations for the task. It then makes a recursive call on the top level scheduling function to explore multiprocessor implementation by setting the parameter S to $FALSE$. The function selects an implementation that results in earliest possible end time (τ_{exec}^{end}) for the task. If S is set to $TRUE$ in the call to $schedule\_hiertask()$, then the function removes any temporary entries from the schedule table and adds an entry for the selected implementation. Alternatively the function makes a temporary entry in schedule table if S is set to $FALSE$.

4.5 Scheduling of a deterministic loop task

Scheduling of a deterministic loop task is the most computation intensive operation since MAGELLAN considers loop unrolling and pipelining for the task. The scheduler applies a modified retiming heuristic (RECOD [9]) for obtaining pipelined schedules. The maximum unrolling degree for the loop task is calculated based on the number of iterations and area constraint $\alpha(v, PE_i)$. The following discussion on design space exploration considers an example loop task shown in Figure 5(a).
1. Single SW: The scheduler maps the entire loop on each SW processor (see Figure 5(b)) and obtains the end time for the task.
2. Multiple SW iteration locality unrolled: The scheduler unrolls the loop and maps all tasks belonging to an iteration of the loop to the same SW processor (see Figure 5(c)).
3. Single HW pipelined: The scheduler explores single device pipelined implementations of the loop on all eligible HW coprocessors (see Figure 5(d)).
4. Single HW unrolled/pipelined: The scheduler explores single device unrolled/pipelined implementations of the loop on all eligible HW coprocessors (see Figure 5(e)).
5. Multiple processor (HW and SW) pipelined : The tasks are mapped to different processors with an objective of minimizing the throughput of the pipelined loop (see Figure 5(f)).
6. Multiple HW task locality unrolled/pipelined: The task locality implementation as the name suggests maps instances of the same task belonging to different iterations of the loop to the same processing element (see Figure 5(g)). Such a design facilitates sharing of HW resources. If in the pipelined schedule two instances of the same task that are mapped to the same HW coprocessor do not execute concurrently, then the scheduler implements the task once.
7. Multiple HW iteration locality unrolled/pipelined: The scheduler explores multiple HW coprocessor iteration locality unrolled/pipelined implementation over all eligible HW coprocessors (see Figure 5(h)).

4.6 Scheduling of a Non-deterministic Loop Task

MAGELLAN does not consider multi-processor pipelining or loop unrolling for a non-deterministic loop task. The scheduler only explores single SW (Figure 5(b)) and single HW pipelined (Figure 5(d)) implementation for the task.

4.7 Scheduling of a case task

Similar to the hierarchical task, the scheduler explores single HW coprocessor, single SW processor and multiple processor implementations for the case task. MAGELLAN explores speculative execution of the path tasks if they are mapped to HW coprocessors. Speculative execution involves scheduling the execution of the path tasks before the execution of the control task. Speculative execution can occur if the intersection of the predecessor task sets of the input ports of the control and path tasks is the null set. The scheduler does not explore speculative execution for a case task if it is the body task or hierarchical successor of the body task of a loop task.

5. SYSTEM PARTITIONER

The partitioner tries to improve upon a solution generated by the scheduler. It does so by trying out several moves. It evaluates several moves and finally implements one move that results in largest decrease in task graph latency. Once a move is implemented by the partitioner, the mapping of the task involved in the move is fixed for the remaining part of the algorithm.

Every task in MAGELLAN is in three states: *free*, *tagged* or *fixed*. Initially all tasks are in a *free* state. In a particular move mode, a task is in a *tagged* state if all the moves have been evaluated on it. In each iteration of the move mode, the partitioner tries out moves on each *free* task and changes its state to *tagged*. Once all the *free* tasks have been changed to *tagged*, the partitioner selects and implements a move that results in largest improvement in task graph latency. The state of the task involved in the move is changed to *fixed* and state of all *tagged* tasks is set to *free*.

The partitioner operates in two move modes: hierarchical move mode and leaf move mode. The partitioner stays in hierarchical move mode until it cannot find any move that results in an improvement. Then the partitioner goes in to leaf move mode. The partitioner iterates between the two move modes until no move results in an improvement or all tasks have been mapped by the partitioner. MAGELLAN exits and returns a solution when the partitioner is unable to apply any more moves.

5.1 Hierarchical move mode

In the move mode the partitioner applies moves on loop, case, hierarchical and call tasks. If the area constraint on the hierarchical task ($\alpha(v, PE_i)$) is too tight to implement a move, it is increased by the partitioner to a minimum value that allows the move to be evaluated. The area constraints on the other hierarchical tasks are also changed to reflect this increase. For all the moves described below the actual selection of the design points is done by the scheduler.

5.1.1 Moves for a deterministic loop task

1. Cluster move: The loop task is clustered with the predecessor and successor tasks. Then the entire cluster is mapped to each HW coprocessor subject to the area constraint on each coprocessor. The scheduler obtains a pipelined implementation on the HW coprocessors. The entire cluster is also mapped to each SW processor.
2. Cluster move with unrolling on single HW: The partitioner calculates the maximum unrolling degree for the loop task on each HW based on the area constraint on the HW copro-

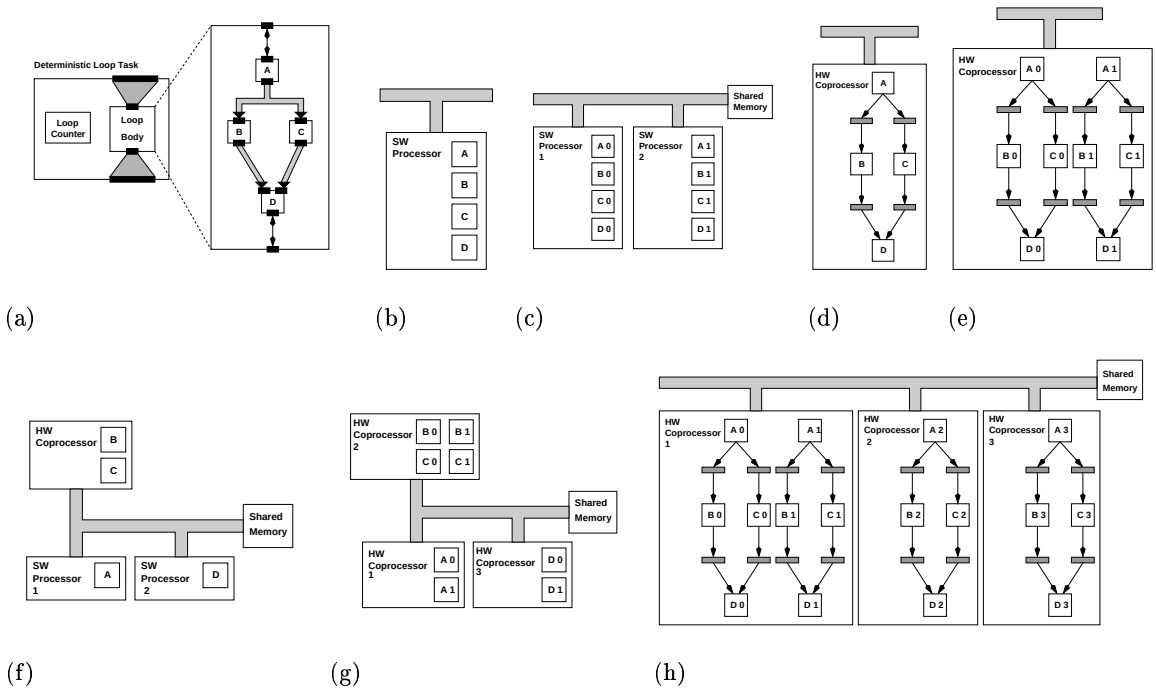

Figure 5: Design space exploration for a deterministic loop task

cessor ($\alpha(PE_i)$). The partitioner allocates enough area to the loop task v so that the scheduler can unroll the task.
3. Single processor move: In this move the partitioner maps only the loop task to each HW and SW processing element.
4. Pipelining and Unrolling moves: The partitioner moves are identical to implementations 2, 4, 6 and 7 that were described for the scheduler.
5. Hierarchical traversal: After evaluation of the above mentioned moves, the partitioner traverses down the hierarchy of the loop task and considers moves on the loop body task if it is a hierarchical task. The moves that are explored on the loop body task depend upon the type of loop body task. If the loop body task is not a hierarchical, loop or case task the partitioner does not explore any moves on the loop body task in the hierarchical move mode.

5.1.2 Moves for a non-deterministic loop task

The partitioner evaluates the cluster move, single processor move and hierarchical traversal for a non-deterministic loop task.

5.1.3 Moves for hierarchical and case tasks

The partitioner evaluates the single processor move and hierarchical traversal for hierarchical and case tasks.

5.1.4 Moves for a call task

The number of implementations of a callee (function) task and the mapping of the call tasks on the callee implementations is determined by the scheduler. A callee task implementation can have multiple call tasks associated with it. The followings moves are applied only once for each callee task implementation in each iteration of the hierarchical move mode:
1. Changing the implementation of a callee task : The type of moves applied on each callee implementation depend upon the type (deterministic loop task, case task etc) of callee task.
2. Eliminating an implementation of a callee task : The scheduler generates the minimum number of callee implementations such that there do not exist scheduling conflicts between any two call tasks. The partitioner move reduces the number of implementations of the callee task. This leads to scheduling conflicts and hence delay in scheduling of corresponding call tasks. However, it also increases the available area for implementing other tasks if the callee task was mapped to a HW coprocessor. Hence, this move is only applied if the callee task is implemented on at least one HW coprocessor.

5.2 Leaf move mode

5.2.1 Moves for a leaf task

Depending upon the present mapping of the leaf task, the partitioner explores several moves:
1. HW coprocessor implementation: The partitioner moves the leaf task to other design points in the HW coprocessor and evaluates each design point. The partitioner then moves the task to all other design points on other HW coprocessors and evaluates the moves. Finally the partitioner moves the task to all SW processors.
2. SW processor implementation The partitioner moves the leaf task to other SW processors and evaluates the move. It also moves the task to all HW design points on all HW coprocessors.

5.2.2 Moves for counter and control tasks

The counter (control) task can exist only as a constituent task of a deterministic loop (non-deterministic loop or case) task. The mapping of the control and counter tasks are determined by the mapping of the corresponding loop body or case path task. Therefore, the partitioner does not change the mapping of the control or counter task in a given solu-

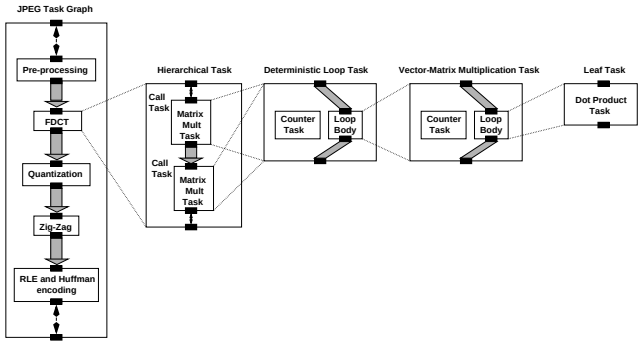

Figure 6: JPEG Image Compression

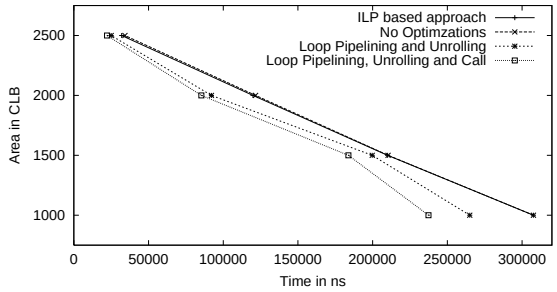

Figure 7: Design Optimizations

tion. If the counter or control has been mapped to a HW coprocessor, the partitioner moves the task to other design points on the same HW coprocessor.

6. RESULTS AND CONCLUSION

The effectiveness of MAGELLAN is demonstrated by a case study of the JPEG image compression algorithm. In JPEG the forward discrete cosine transform (FDCT) task is the most computationally expensive task. As shown in Figure 6 it primarily consists of two matrix multiplications and is therefore highly suitable for loop optimizations. The FDCT task was expressed as hierarchical task that consisted of two calls to matrix multiplication task. The matrix multiplication task is deterministic loop task with a counter of 8 and whose loop body task is a vector-matrix multiplication task. The vector-matrix multiplication task is also a loop task with a counter of 8 and contains the dot product task as its loop body. We obtained the SW estimates for the tasks by profiling on a 266 MHz pentium system. We obtained four HW design points for the dot product task, four HW design points for the quantization task and one HW design point each for the preprocessing, zigzag, RLE and Huffman encoding tasks. The HW design points were obtained for Xilinx 4000 series device by using a high-level synthesis tool.

We first compare the results of MAGELLAN with all loop and call optimizations switched off against an ILP based approach [5] that also does not perform any loop or call optimizations (see Figure 7). MAGELLAN gives close to optimal results when compared with the ILP based approach. The ILP based approach required on an average 9134.98 seconds to produce the results while MAGELLAN without optimizations required 0.95 seconds. This result establishes the quality of the solution generated by MAGELLAN and forms the basis for evaluating the results of the following experiment.

We demonstrate the effectiveness of the optimizations performed by MAGELLAN by comparing the results against the optimal solution obtained in the previous experiment (see Figure 7). The plots give the result of MAGELLAN with a.) loop unrolling & pipelining and b.) function call optimization, loop unrolling & pipelining. As can be seen from the plots the each of the optimizations performed by MAGELLAN result in significant reduction of latency of the design. The improvement is 31.01 % at higher area (2500) constraints and 22.84 % at lower area (1000) constraints. Thus MAGELLAN improves on existing automated partitioning approaches since they also do apply loop or call optimizations. Further with full optimizations MAGELLAN required on an average 11.76 seconds (maximum 17.32s) to generate the results on a Sun SPARCstation 20. Hence, MAGELLAN can generate highly optimized designs in a short period of time.

The paper presented MAGELLAN, a heuristic technique for multiway HW-SW partitioning and scheduling of hierarchical control-dataflow task graphs. The technique improves on existing techniques by operating on hierarchical specifications and applying various algorithm optimizations.

7. REFERENCES

[1] H. Takata et al. "The D30V/MPEG Multimedia Processor". In *IEEE Micro*. IEEE Computer Society Press, July-August 1999.

[2] M. Ikeda et al. "SuperEnc: MPEG-2 Video Encoder Chip". In *IEEE Micro*. IEEE Computer Society Press, July-August 1999.

[3] R.K. Gupta. *Co-Synthesis Of Hardware And Software For Digital Embedded Systems*. Kluwer Academic Publishers, 1995.

[4] R. Ernst et al. "The COSYMA environment for hardware/software cosynthesis of small embedded systems". *Journal of Microprocessors and Microsystems*, 20, 1996.

[5] A. Kalavade. *"System-Level Codesign of Mixed Hardware-Software Systems"*. PhD thesis, University of California, Berkeley, 1995.

[6] S. Bakshi and D.D. Gajski. "A Scheduling and Pipelining Algorithm for Hardware/Software Systems". In *International Symposium on System Synthesis*, September 1997.

[7] R. Niemann and P. Marwedel. "Hardware/Software Partitioning using Integer Programming,". In *European Design and Test Conference, ED&TC*, 1996.

[8] J. Jeon and K. Choi. "Loop Pipelining in Hardware-Software Partitioning". In *Proceedings of ASPDAC*, 1998.

[9] K.S. Chatha and R. Vemuri. "RECOD: A Retiming Heuristic To Optimize Resource and Memory Utilization in HW/SW Codesigns". In *International Workshop on Hardware/Software Codesign*, March 1998.

[10] T. Yen and W. Wolf. *Hardware-Software Co-Synthesis Of Distributed Embedded Systems*. Kluwer Academic Publishers, 1996.

[11] B.P. Dave and N.K. Jha. "COHRA: Hardware-Software Co-Synthesis of Hierarchical Heterogenous Distributed Embedded Systems". 17(10), October 1998.

[12] R.P. Dick and N.K. Jha. "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Cosynthesis of Distributed Embedded Systems". 17(10), October 1998.

[13] G. Lakshimnarayana B.P. Dave and N.K. Jha. "COSYN: Hardware-Software Cosynthesis for Heterogeneous Distributed Embedded Systems". *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, 7(1), March 1999.

[14] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-Vincentelli. "Scheduling for Embedded Real-Time Systems". January-March 1998.

[15] K.S. Chatha and R. Vemuri. "An Iterative Algorithm for Hardware-Software Partitioning, Hardware Design Space Exploration and Scheduling. *"Design Automation for Embedded Systems*, (5):281–293, 2000.