

PAP: Power-Aware Partitioning of Reconfigurable Systems

Vijay Kappagantula
Texas A&M University
College Station, Texas 77843
kpramod@ee.tamu.edu

Rabi N. Mahapatra
Texas A&M University
College Station, Texas 77843
rabi@cs.tamu.edu

ABSTRACT

Existing approaches in hardware-software partitioning do not consider the maximum available system power while making the partitioning decision. Such a decision is sensitive to design efficiency when the target system is reconfigurable and supports multiple applications. Since partitioning and scheduling are interdependent, neglecting the available system power could result in an infeasible schedule. In this paper, we present a power-aware partitioning technique for reconfigurable systems that could support a set of applications. The applications specified at a task level of granularity are partitioned and scheduled such that timing and maximum available system power constraints are satisfied with minimum reconfigurable logic.

Keywords

Partitioning, scheduling, codesign, multifunction, power-aware

1. INTRODUCTION

Today's system architects adopt heterogeneous architectures as design paradigms for high performance and configurable devices as components for cost effective solutions. The HW-SW codesign methodology helps designers to embrace complex designs by reusing the IP's and shorten the time market gap. A generic system consists of one or more CPUs (software processing entity), reconfigurable hardware (FPGA) and/or hardware/ASIC processing elements. In HW-SW codesign, applications are usually specified at task level granularity. The key issues involve in codesign are partitioning, synthesis and cosimulation before final implementation. The task partitioning is a critical design step that entails appropriate mapping (hardware or software) and scheduling of tasks on the processing elements such that the timing constraints of applications are met with minimum cost (hardware area and/or power). Partitioning problem is non-trivial since its solution space increases exponentially with number of tasks and their implementation techniques.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HPCA-9/SSRS '03 Anaheim, California USA

Copyright 2003 ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

An efficient partitioning strategy is always in demand to reduce the design time while satisfying the performance.

Although most attention is focused on design speed and accuracy while partitioning and scheduling, power consumption is becoming more dominant in complex system design. Therefore, the system architects need to make the design power-aware. An efficient power-aware design could be achieved by integrating task level power consumptions during the partitioning stage of codesign.

The reconfigurable components in a system offer significant benefits in terms of handling design variations. A complex system can be spatially decomposed into simpler functions (tasks) and appropriately scheduled for execution. Similarly, multiple functions can be integrated into a single system (multifunctional) by sharing some of the hardware modules (when possible) for cost effective design.

The power-performance trade-off is critical to successful system design. In a reconfigurable system, tasks could be swapped between the components (CPUs and FPGA) to meet different performance goals. Based on the power budget of a system, more tasks could be executed on the hardware (FPGA) to enhance the processing speed. Similarly, new tasks may be admitted in a multifunctional system if the power consumptions of incoming tasks meet the available power at that instant. New research opportunities in system architecture are evident due to synergism of codesign methodology and advent of reconfigurable architectures.

In this paper, a power aware partitioning (PAP) heuristic algorithm has been proposed for single functional systems. The algorithm has been extended to consider multifunctional system that includes sharing of common tasks between applications (if any) to provide cost effective solution. Through case studies, the efficacy of proposed approach has been demonstrated.

The rest of the paper is organized as follows. Section 2 defines the problem formally. Section 3 analyzes the work that has already been done in this area. We present the power-aware partitioning algorithm for single functional systems in Section 4 and extend it to multifunctional systems in Section 5. We discuss the experimental results in Section 6.

2. PROBLEM DEFINITION

We assume a HW/SW co-design architecture model that consists of a single programmable processor (CPU), a reconfigurable hardware (FPGA) and an underlying communication architecture (shared bus) between them. Though it is possible to consider certain tasks mapped only to hardware or software, the CPU and the reconfigurable hardware are

assumed to be able to execute all the tasks in the set of applications independently. The architecture is constrained by the maximum power rating P_{max} of the non-rechargeable battery source and the total available hardware area Ah_{total} (total number of logic gates/CLB's/slices of the reconfigurable hardware).

Each individual application is described as a Directed Acyclic Graph (DAG) $G = (V, E)$. Node $v \in V$ represents a task and each edge $e \in E$ represents data dependence between the connecting task nodes. Each task v is either mapped to the CPU or to the reconfigurable hardware. Each task in the DAG is associated with five parameters (T_{sv} , P_{sv} : Execution time and instantaneous power dissipation when mapped to CPU, Thv , Phv : Execution time and instantaneous power dissipation when mapped to the FPGA, Ahv : Number of CLB's used when mapped to the FPGA). The execution time of a task is assumed to non-uniform when executed on both the CPU and the FPGA resource. The instantaneous power dissipation is assumed to be uniform during the time of execution of a task on a processing element. An edge $u \rightarrow v$ in the DAG implies that task u is a predecessor of task v . The weight of the edge between two tasks represents the communication time between the tasks. The weight is determined based on the underlying communication architecture between the CPU and the FPGA and the amount of data communicated between the tasks. The weight of an edge between tasks where both tasks are mapped to the same resources is assumed to be negligible.

The problem of power-aware partitioning of multifunctional systems could be formally defined as follows: *Given an application set denoted by $\{A_1, A_2, \dots, A_n\}$, where each individual application A_j is specified as a DAG with a periodic deadline D_j , total available hardware Ah_{total} and maximum power rating of the system P_{max} , generate a partition for each application such that they meet their timing and maximum power rating constraints with minimum hardware cost. More than one application could be active at run time.*

3. RELATED WORK

While we are not aware of any partitioning methodology that concurrently partitions and schedules an application and takes into account the maximum power rating of the system while partitioning and scheduling an application, a number of algorithms have been proposed for partitioning of single and multifunctional systems such that they meet their timing requirements only. Some of the works include iterative improvement, heuristics and mathematical programming techniques [1, 2, 3, 4, 5, 6, 7, 8]. In [6], an integrated partitioning and scheduling algorithm for hardware-software partitioning is discussed but their work is limited to applications where tasks have uniform execution times when mapped to software and negligible execution times when mapped to hardware. [7] presents two partitioning algorithms for multifunctional systems by modifying the GCLP algorithm discussed in [5]. These works verify if the application meet its timing requirement but do not consider the power rating while making the partitioning decision. In [9], a scheduling technique for power critical systems is presented where tasks are rescheduled using the available slacks such that the application meets its timing and power requirements. Since their approach assumes an initial mapping and schedule, it leaves little flexibility for rescheduling tasks. In

our approach, concurrent partitioning and scheduling provides more flexibility in scheduling the tasks and ensures that the application meets its timing and power requirements.

4. PAP : POWER AWARE PARTITIONING ALGORITHM

In this section, we present the power-aware partitioning algorithm for single functional systems.

4.1 Algorithm Foundation

Power-aware partitioning algorithm is based on iterative improvement techniques. The algorithm initially maps all the tasks in the application to software. The start times of the tasks on the CPU are determined on the basis of a priority function. The priority function used ranks the tasks in the order of decreasing depth in the task graph. One task per iteration is selected to be moved to hardware based on the tasks's *mobility* indices and a *Task Selection Routine*. The start time of the execution of the selected task η is determined. The power profile of the schedule representing the instantaneous power consumption of all tasks is computed and verified to see if it meets the maximum power rating of the system. The partitioning process is repeated until the schedule of the application meets its timing constraint or until all available hardware logic is exhausted.

4.2 Task Mobility

The *mobility* of a task in the application provides information about the parallelism that could be achieved by moving the task from software to hardware and executing it in parallel with other tasks. The mobility of a task determines its earliest possible start and latest possible finish times. The mobility indices of the tasks in the application are dependent on the schedule of the application and may change at each iteration of the partitioning algorithm. A task is defined as *mobile* if its latest possible finish time is greater than its earliest possible start time and *immobile* otherwise. The following procedure summarizes the computation of the mobility of a task.

Let t_i denote the execution time of task i .

$$t_i = ts_i, \text{ task } i \text{ is mapped to SW} \quad (1)$$

$$th_i, \text{ task } i \text{ is mapped to HW} \quad (2)$$

Let E_i denote the earliest possible start time and L_i denote the latest possible finish time of task i . E_i and L_i provide the lower and upper time bound for determining the start time of the task.

$$E_i = \max_{k \in \text{pred}(i)} \eta(k) \quad (3)$$

where $\text{pred}(i)$ denotes the immediate predecessor set of task i , $\eta(k)$ denotes the finish time of task k . Similarly

$$L_i = \min_{k \in \text{succ}(i)} \{\eta(k) - t_i\} \quad (4)$$

where $\text{succ}(i)$ denotes the immediate successor set of task i , $\eta(k)$ denotes the start time of task k . The *mobility* of a task i , $\mu(i)$ is defined as follows:

$$\mu(i) = \begin{cases} 1, & L_i > E_i \\ 0, & L_i = E_i \end{cases} \quad (5)$$

4.3 Task Selection Routine

At every step of the partitioning algorithm, one task is selected to be moved from software to hardware and executed in parallel with other tasks. The task to be moved is selected on the basis of a priority function. The software tasks are ranked in the decreasing order of their execution times and their *mobility* indices are used to select the task to be moved to hardware. The following procedure summarizes the *Task Selection Routine*.

Procedure: **Task Selection Routine**
 Inputs: ts_i , for all $i \in N_s$, N_s : Set of software tasks
 Output: Selected Task
 S.1 Rank the tasks in N_s in the order of decreasing software execution times ts_i
 S.2 Compute the *mobility* $\mu(i)$ for all $i \in N_s$
 S.3 If $\mu(i) = 0$ for all $i \in N_s$
 task i with maximum execution time ts_i is selected to be moved to hardware
 terminate procedure
 Else
 task $i \in N_s$ with maximum execution time ts_i is considered
 If $\mu(i) = 1$
 task i is selected to be moved to hardware
 terminate procedure
 Else If $\mu(i) = 0$
 remove task i from N_s
 go to S.3

4.4 Power Characteristic of the Partitioning Schedule

The embedded system is specified to perform at a maximum power rating denoted by P_{max} . The *power profile* of the schedule is defined as the instantaneous power consumption of the tasks. The Power profile (P_σ) is computed as follows:

$$P_\sigma(t) = \sum P(i), \text{ for all } i \in \text{set of tasks active at time instant } t \quad (6)$$

The schedule is called *power-valid* if the power profile of a single iteration of the application is less than or equal to the maximum power rating of system (P_{max}). If the power profile of the schedule at any time instant exceeds the maximum power rating of the system, then the power profile at that time instant is having a *power spike*. The schedule with one or more power spikes is an infeasible schedule. The power spike could be removed by either rescheduling the execution or changing the map of one or more tasks active at that time instant.

The total energy of the schedule (E_σ) is computed as follows:

$$E_\sigma = \int_0^{T_\sigma} P_\sigma(t) dt, \quad T_\sigma \text{ is the finish time of a single iteration of the application} \quad (7)$$

4.5 Time-Valid Schedule

The finish time of a single iteration of the application (T_{exec}) is defined as the time when all the tasks in the ap-

plication finish their execution. It is defined as:

$$T_{exec} = \max(\eta(i) + t_i), \text{ for all } i \in N \quad (8)$$

where $\eta(i)$ is the start time, t_i is the execution time and N is the total number of tasks in the application.

The schedule of the application is called *Time-valid* if T_{exec} is less than or equal to D , where D is the application deadline.

4.6 Communication Model for Partitioning

The mapping of tasks in the application to either hardware or software depends on the underlying communication architecture (bus, protocol) and the communication delay. The performance gain obtained by moving tasks to hardware could be lost if the communication overhead is too large or due to non-availability of resources (bus). Thus the scheduling of tasks and their communications on the bus are interdependent. The partitioning algorithm should also schedule the communication between communicating components of the system. In order to schedule the communication, we need to estimate the delay between any two communicating tasks which is presented in the next section.

4.6.1 Calculation of the communication delay

We have used a 32 bit, 33 Mhz PCI architecture bus as the communication interface between the CPU and the reconfigurable hardware (FPGA). The communication delay $t_{comm}(u, v)$ is the weight of the edge between tasks u and v in the application specified at a task level of granularity. The communication delay t_{comm} is assumed to be time required for the transmission of one sample of data on the channel (PCI Bus). The communication delay is computed according to the method in [10] as follows:

$$t_{comm} = \frac{\frac{CC * N_{sample}}{N_{bus}} + AC}{F} \quad (9)$$

where: CC is 2, the communication cycles required per data element. N_{sample} is the total amount of data being transmitted between the communicating tasks. It is assumed to be a constant and equal to 1000 (32 bit) words. N_{bus} is 32, the channel bit width. AC is 2, the arbitration cycles required. F is 33 MHz, the frequency of the channel.

In our analysis, we have assumed the communication delay between any two communicating tasks to be uniform. The above communication delay estimation could be extended for communicating tasks where the amount of data being transmitted is different and results in non-uniform delays.

4.6.2 Power overhead due to communication

The power dissipation for the communication of data is computed based on the scheme described in [11]. The power dissipation P_{bus} is estimated as follows:

$$P_{bus} = \frac{1}{2} * C_{bus} * V^2 * m * n \quad (10)$$

where: C_{bus} is assumed to $10pF$, the capacitance of the bus. V is 3.3 V, the V_{cc} voltage. m is the number of words sent per second which is computed based on the sample data. n is 32, number of bits per word.

We had defined the *power profile* at any time instant as the sum of the instantaneous power consumption of all tasks active at that time instant. The *power profile* of the schedule

needs to be modified to include the power dissipation on the bus. The modified *power profile* (P_σ) is defined as:

$$P_\sigma(t) = \sum P(i) + P_{bus}(t), \text{ for all } i \in \text{set of tasks} \quad (11)$$

active at time instant t

4.6.3 Scheduling the communication

In the partitioning process, the *mobility* of the task defines the earliest possible start time (lower time bound) and the latest possible finish time (upper time bound) when the task could be scheduled. These time bounds define an interval where the task could be scheduled and executed in hardware. The communications for the task need to be scheduled on the underlying architecture (bus) during this time interval. The scheduling of communication between communicating tasks i and j where i is mapped to software and j is mapped to hardware is scheduled such that:

$$\eta(i) + t_i \leq \eta(comm) + t_{comm} < L_j \quad (12)$$

$$E_j < \eta(j) + t_j \leq L_j \quad (13)$$

where $\eta(i)$ and $\eta(j)$ are the start times of tasks i and j . t_i and t_j are the execution times of tasks i and j . E_j be earliest possible start time and L_j be latest possible finish time for task j . $\eta(comm)$ is the start time of the communication between i and j . $t_{comm}(i, j)$ denote the communication time between tasks i and j .

To ensure a feasible schedule for the application, the communications for a task mapped to hardware are scheduled such that:

1. There are no resource (bus) conflicts.
2. The execution of the task and its communications lie within the interval specified by its *mobility*.

4.7 PAP Algorithm

Algorithm: **PAP**

Input: Application specified as a task graph (TG), Task metrics ($ts_i, th_i, P_{s_i}, P_{h_i}, Ah_i$) for all $i \in N$, N is the total number of tasks in TG . Communication costs t_{comm} , Maximum Power Rating P_{max} . Total Available Hardware Area Ah_{total} , Deadline D

Output : All mapped tasks, *Time-Valid* and *Power-Valid* Schedule

Procedure: While{Schedule is not *Time-Valid* }

S.1 Initially all tasks $\in N$ are mapped to software. Schedule is assumed to be *Power-Valid*

S.2 Compute t_{exec} , the finish time of a single iteration for the application

S.3 If ($t_{exec} < D$)

Schedule is *Time-valid*. Terminate Algorithm

S.4 Select task i to be mapped to hardware, $i \in N$ using the *Task Selection Routine*

S.5 Determine the start time $\eta(i)$ for task i , $i \in N$ using its *mobility* $\mu(i)$

Update the Bus Activity using the communication costs

Reschedule the task graph TG and compute t'_{exec} , the new finish time of one iteration for the application

Update the total hardware used Ah

S.6 If ($t'_{exec} < t_{exec}$) and If (schedule is *Power-Valid*) and If ($Ah < Ah_{total}$)

$t_{exec} = t'_{exec}$ (Update t_{exec})

Goto Step S.2

Else If ($Ah > Ah_{total}$)

Invalidate the selection of the current task for hardware mapping for all future iterations
Goto Step S.4
Else If (Schedule is not *Power-Valid*)
Invalidate the selection of the current task for hardware mapping for the next iteration
Goto Step S.4
Else If ($t'_{exec} > t_{exec}$)
Invalidate the selection of the current task for hardware mapping for the next iteration
Goto Step S.4

4.8 Example

Figure 1 shows an example of the PAP algorithm. The initial all software implementation is shown in Figure 1(a). Task 1 (shaded gray) is selected to be moved to hardware (FPGA) in iteration 1. The generated schedule is power-valid but does not meet the deadline D (Figure 1(b)). In the second iteration, task 2 (shaded gray) is selected to be moved to hardware. A possible schedule which meets the timing deadline but fails to meet the maximum power rating P_{max} is shown in Figure 1(c). The execution and communication (shaded black) for task 2 are rescheduled within the time interval defined by its mobility such that the new schedule is both time and power valid (Figure 1(d)).

5. PARTITIONING OF MULTIFUNCTIONAL SYSTEMS

In multifunctional partitioning, the set of applications active at run time are analyzed to determine the similar tasks across the set of applications. The applications which are specified at the task level of granularity are combined into a single task graph. The power-aware partitioning (PAP) algorithm is then applied to the combined task graph. The partitioning algorithm uses the information about similar tasks and hardware re-usability to map and schedule the tasks of each individual application.

All the tasks belonging to set of active applications are initially mapped to software. The tasks are referenced by their application name to distinguish similar tasks. Our approach assumes the similar tasks to be identical but could be extended to other forms of similarities between tasks across the set of applications. During each iteration, one task of each application is considered for hardware mapping. The order in which the applications are considered is based on their criticality's and is significant because the mapping information of tasks is propagated between applications in the current and successive iterations.

5.1 Criticality of Applications

At each step of the partitioning algorithm for multifunctional systems, the set of active applications $\{A_1, A_2, \dots, A_n\}$ are ordered based on their criticality's. The set of active applications are combined into a single task graph (CTG). The criticality of an application (AC) is a measure of its

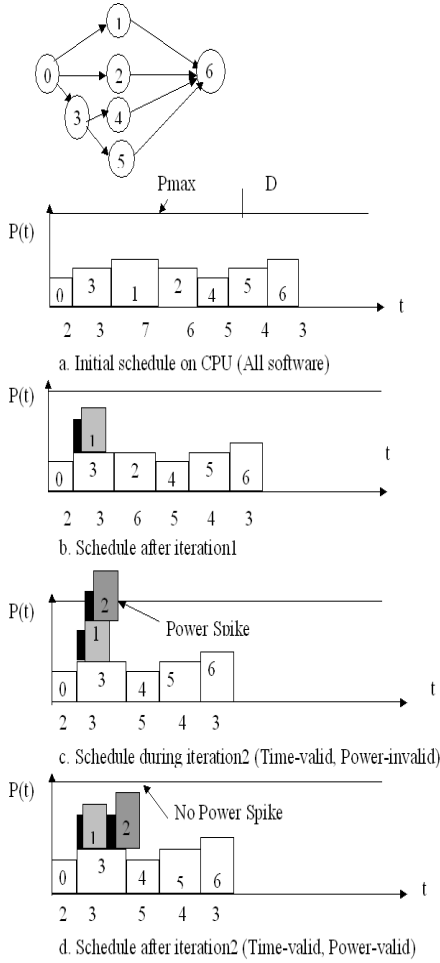


Figure 1: A example of the PAP algorithm

computational complexity and is described as:

$$AC_i = \frac{T_{CTG}}{D_i} \quad (14)$$

where: AC_i is the application criticality of A_i , $A_i \in \{A_1, A_2, \dots, A_n\}$. D_i is the deadline of A_i . T_{CTG} is the finish time of one iteration of CTG.

When a task of the CTG is moved from software to hardware, the finish time of one iteration (T_{CTG}) is modified. Therefore the application criticality's need to be computed at the beginning of each iteration of the partitioning algorithm. The application with maximum criticality is considered first and a task from this application is selected to be moved from software to hardware using the *Modified Task Selection Routine*.

5.2 Modified Task Selection Routine

The set of active applications specified at a task level of granularity are combined into a single task graph. All the tasks are initially mapped to software. During each step of the partitioning algorithm, one task of each application belonging to the set of active applications is moved from software to hardware and executed in parallel with other tasks. The priority function used to move a task from software to hardware for single functional systems has been described

in Section 4.3. In the case of multifunctional systems, the information about similar tasks across the set of active applications and hardware re-usability is used to select the task to be moved from software to hardware. Thus, the *combined priority function* to move a task of one application from software to hardware uses information about its execution time, *mobility* within the application (*self priority*) and information about similar tasks and hardware availability for re-use across the set of applications (*shared priority*). The *combined priority (CP)* of a task is defined as:

$$CP(i) = SeP(i) + ShP(i) \quad (15)$$

where $SeP(i)$ and $ShP(i)$ are the self and shared priorities of task i .

Next we present the computation of the *Self Priority* of all the software tasks of an application which is part of the combined task graph.

Procedure: Self Priority of software tasks in an Application

Inputs: $A_k, A_k \in \{A_1, A_2, \dots, A_n\}$

Output: Self Priority of all software tasks

S.1 Initialize counter $Count = 0$

S.2 Compute the *mobility* $\mu(i)$ for all $i \in N_s$. N_s is set of software tasks in A_k

S.3 Determine $N_{s1} \in N_s$, set of all software tasks with non-zero mobility and $N_{s2} \in N_s$, set of all software tasks with zero mobility.

S.5 Sort all tasks $\in N_{s1}$ in non-increasing order based on their execution times.

S.6 Sort all tasks $\in N_{s2}$ in non-increasing order based on their execution times.

S.7 Extract task i , $i \in N_{s1}$ with maximum execution time ts_i

S.7.1 Compute Self Priority as follows

$$SeP(i) = \frac{N_s - Count}{N_s}$$

S.7.2 Increment $Count$

S.7.3 Remove task i from N_{s1}

S.7.4 Goto Step S.7

S.8 Extract task i , $i \in N_{s2}$ with maximum execution time

S.8.1 Computer Self Priority as follows:

$$SeP(i) = \frac{N_s - Count}{N_s}$$

S.8.2 Increment $Count$

S.8.3 Remove task i from N_{s2}

S.8.4 Goto Step S.8

The *self priority* $SeP(i)$ of software task i is computed based on its execution time and its mobility with respect to other tasks in the same application. By moving a software task with highest *self priority* to hardware, a greater performance gain could be achieved but it results in an increase in the total hardware area. Moving a task which is repeated across the set of applications such that the hardware implementation could be reused and results in a reduction in the total hardware cost.

The *shared priority* of a software task i is computed as the total number of times a similar task across the set of applications has been mapped to hardware Num_i . The measure is normalized across all the software tasks of the application

N_s . The *shared priority* ($ShP(i)$) of a software task i is described as:

$$ShP(i) = \frac{Num_i}{\max Num_j} \quad \text{for all } j \in N_s \quad (16)$$

As described in Equation 15, each software task of an application is assigned an *combined priority* based on its *self priority* and *shared priority*. The combined priority is used to bias similar tasks for hardware mapping which could result in reduction in the total hardware used.

5.3 Partitioning of Multiple Applications

Algorithm: MPAP

Input: Application A_i , Deadline D_i for all $A_i \in \{A_1, A_2, \dots, A_n\}$

Total Available Hardware Area Ah_{total}

Maximum Power Rating of the system P_{max}

Output: Time and Power valid schedules for the set of applications.

Procedure: While the schedules for the set of applications are not time and power valid

S.1 Set of applications are combined to form a single task graph CTG . All tasks are initially mapped to software. Schedule is assumed to be *Power-Valid*.

S.2 The Application Criticality AC_i for application A_i is computed, for all $A_i \in \{A_1, A_2, \dots, A_n\}$.

S.3 The applications are ranked by the criticality's such that the one with maximum application criticality is considered first.

S.4 A task of the application is selected for hardware mapping using the modified task selection routine and using steps S.5 and S.6 of the PAP algorithm. Repeat for other applications in the ordered set $\{A_1, A_2, \dots, A_n\}$.

S.5 For all $A_i \in \{A_1, A_2, \dots, A_n\}$

If (Schedule is both time and power valid)

Remove A_i from $\{A_1, A_2, \dots, A_n\}$

End algorithm if schedules of all applications are time and power valid.

Else

Repeat from step S.2

5.4 Time complexity of the MPAP algorithm

In each iteration, for an combined task graph (CTG) containing N tasks, the tasks's *mobility* indices are computed in $O(N)$ time and the tasks are sorted in $O(N \log N)$ time. The self and combined priorities of the tasks can be computed in $O(N)$ time. Therefore the modified task selection routine takes $O(N \log N)$ time. The rescheduling during each iteration takes $O(N)$ time. Since there can be at most N iterations and the initial all software implementation on a single CPU takes $O(N^2)$, the time complexity of the MPAP algorithm is $O(N^2 \log N)$.

6. CASE STUDIES

We consider practical signal processing applications with periodic timing constraints. Two examples used are: 8 Khz 16-QAM Modem and DTMF Codec. These applications are specified in CGC domain of the Ptolemy system[12]. The tasks/blocks in the applications are generated in C language. The software time (ts_i) and software power estimates (Ps_i) for each task i are obtained on the StrongARM SA-1100 processor (operating at 206 MHz) using *JouleTrack* [13]. For the hardware implementations, VHDL RTL was

Table 1: Results from the PAP algorithm and Extensive Search

Example	Method	Power(W)	Finish Time(μs)	Simulation Time(s)
16-QAM	PAP	8	773	0.7
		6	780	0.7
		2	903	0.7
	Ext. Search	8	671	15310
DTMF	PAP	8	791	0.8
		6	791	0.8
		2	966	0.8
	Ext. Search	8	685	22160

developed for all the functional blocks. The target reconfigurable hardware resource in our study is the *Xilinx-Virtex2* (XCV4000). The hardware execution time (Th_i) is computed as the worst case propagation delay after Place and Route (PAR) on the FPGA using the Xilinx ISE 4.2 simulator. The hardware power dissipation (Ph_i) for each task i in the application is obtained using the Xpower tool which is part of the Xilinx ISE 4.2 simulator. The underlying bus architecture between the software processor and the FPGA is assumed to be the 33 MHz PCI bus.

Next we present two experiments. In the first experiment described, the PAP algorithm is independently applied to the two applications. The partitioning and scheduling results are compared with results obtained from extensive search. In the second experiment, the MPAP algorithm is applied to the modem and codec applications running concurrently as a multifunctional system. The results are compared with two single function implementations using the PAP algorithm.

6.1 Experiment : PAP vs Extensive Search

We assume that the incoming data for both the examples are periodic with a deadline of $800\mu s$. The PAP algorithm was applied to both the cases with three different power constraints (8W, 6W, 2W) to see if the results meet the prescribed deadlines. In order to compare the performance of the PAP algorithm, we applied the extensive search technique to partitioning. To be fair, we considered equal FPGA areas (# of slices used as obtained from the PAP algorithm) in addition to the given power and deadline constraints for both cases. We found that the extensive search algorithm takes about four to six hours compared to the less than a second for PAP when simulated using a UltraSPARC5. From Table 1, it is also observed that the finish times of both the modem and codec implementations obtained using the PAP algorithm are comparable to extensive search results for the same power and FPGA resources. For an available power of 2W, both applications fail to meet the deadline requirement.

6.2 Experiment: Single vs Multifunctional Partitioning

Here, we first partition the two applications separately using the PAP algorithm assuming system power rating of 8W. The Table 2 lists the total reconfigurable logic (# of slices used) for each example. A hybrid system running both the

Table 2: Total Hardware Area for the PAP and MPAP algorithms when applied to the 16-QAM Modem and DTMF Codec

Application/s	Algorithm	# of Slices
16-QAM and DTMF	PAP	842
	MPAP	803

modem and codec applications without sharing the FPGA resources would consume 842 slices of FPGA. When MPAP is applied to the multifunctional system sharing the FPGA resources with the same power constraint, we found a 5% saving in the FPGA slices. Depending on the set of applications and shared tasks, the saving could be significant.

7. CONCLUSION

In this paper, an efficient power-aware partitioning heuristic is proposed for reconfigurable systems. Using power constraint as input, the algorithm concurrently partitions and schedules tasks in an application to meet the deadline. The algorithm has been applied to both single and multiple function systems using two case studies. The multifunctional partitioning algorithm shares resources between functions and proves to be an area efficient solution. Since the proposed PAP/MPAP algorithm's run time is low, several utilities can be addressed when applied to hybrid system design. For example, various tasks/functions in an application can be swapped between processor and reconfigurable fabrics to meet different power-performance goals as needed. As our future research, we plan to extend the scheme to distributed embedded systems and adopt voltage scaling selectively.

8. REFERENCES

- [1] F. Vahid, J. Gong, and D. Gajski. A binary-constrained search algorithm for minimizing hardware during hardware-software partitioning. In *European Design Automation Conference*, pages 214–219, 1994.
- [2] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli. Hardware/software partitioning with iterative improvement heuristics. In *9th International Symposium on System Synthesis*, pages 71–76, 1996.
- [3] R. Niemann and P. Marwedel. An algorithm for hardware /software partitioning using mixed integer linear programming. In *ED&TC*, 1996.
- [4] R. K. Gupta and G. De Michelli. Hardware-software cosynthesis for digital systems. *IEEE Trans. Design and Test of Computers*, 10(3):29–41, Sept. 1993.
- [5] A. Kalavade and E. A. Lee. The extended partitioning problem: Hardware/software mapping and implementation-bin selection. In *Sixth International Workshop on Rapid Systems Prototyping*, pages 12–18, 1995.
- [6] H. Liu and D. F. Wong. Integrated partitioning and scheduling for hardware/software co-design. In *International Conference on Computer Design*, pages 609–614, 1998.
- [7] A. Kalavade and P. A. Subrahmanyam. Hardware/software partitioning for multifunction systems. *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, 17(9):819–837, Sept. 1998.
- [8] M. Potkonjak and W. Wolf. Cost optimization in asic implementation of periodic hard-real time systems using behavioral synthesis techniques. In *International Conference on Computer Aided Design*, pages 446–451, 1995.
- [9] J. Liu, P. H. Chou, N. Bagherzadeh, and F. Kurdahi. Power-aware scheduling under timing constraints for mission-critical embedded systems. In *Design Automation Conference*, pages 840–845, 2001.
- [10] P. V. Knudsen and J. Madsen. Integrating communication protocol selection with partitioning in hardware/software codesign. In *11th International Symposium on System Synthesis*, pages 111–116, 1998.
- [11] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmit. Ptolemy: A framework for simulating and prototyping heterogenous systems. *International Journal of Computer Simulation*, 4(2):155–182, April 1994.
- [12] *The Almagest: Manual for Ptolemy Version 0.7*. Department of EECS, University of California, Berkeley, California, 1988.
- [13] A. Sinha and A. P. Chandrakasan. Jouletrack-a web based tool for software energy profiling. In *Design Automation Conference*, pages 220–225, 2001.