Malaysian Simulation SOCIETY
http://www.ams-mss.org
APPLICATIONS OF MODELLING AND SIMULATION
eISSN 2600-8084
VOL 1, NO. 1, 2017, 1-14

Hardware/Software Partitioning of Streaming Applications for Multi-Processor System-on-Chip

J. W. Tang[1], Y. W. Hau[2] and M. N. Marsono[1*]

[1]Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor, Malaysia.
[2]IJN-UTM Cardiovascular Engineering Centre, Faculty of Biosciences and Medical Engineering, Universiti Teknologi Malaysia

[*]Corresponding author: nadzir@fke.utm.my

Abstract: Hardware/software (HW/SW) co-design has emerged as a crucial and integral part in the development of various embedded applications. Moreover, the increases in the number of embedded multimedia and medical applications make streaming throughput an important attribute of Multi-Processor System-on-Chip (MPSoC). As an important development step, HW/SW partitioning affects the system performance. This paper formulates the optimization of HW/SW partitioning aiming at maximizing streaming throughput with predefined area constraint, targeted for multi-processor system with hardware accelerator sharing capability. Software-oriented and hardware-oriented greedy heuristics for HW/SW partitioning are proposed, as well as a branch-and-bound algorithm with best-first search that utilizes greedy results as initial best solution. Several random graphs and two multimedia applications (JPEG encoder and MP3 decoder) are used for performance benchmarking against brute force ground truth. Results show that the proposed greedy algorithms produce fast solutions which achieve 87.7% and 84.2% near-optimal solution respectively compared to ground truth result. With the aid of greedy result as initial solution, the proposed branch-and-bound algorithm is able to produce ground truth solution up to 2.4741e+8 times faster in HW/SW partitioning time compared to exhaustive brute force method.

Keywords: Branch-and-bound; hardware/software partitioning; multi-processor; system-on-chip; streaming application

1. INTRODUCTION

System-on-chip (SoC) including hardware/software (HW/SW) co-design has been widely used in numerous embedded systems. Moreover, modern Multi-Processor System-on-Chip (MPSoC) allows software parallelism, giving rise to more possibility in HW/SW co-design. Thus, efficient HW/SW partitioning technique is required to ensure a high throughput and cost-effective embedded system while satisfying the shorter time-to-market.

Streaming applications in embedded systems are common nowadays such as video or voice related programs. These data-intensive streaming applications require high computation throughput to provide good quality-of-service. Moreover, emerging technologies such as cloud computing necessitate better quality of stream throughput [1]. Poorly developed embedded system will become bottleneck for the advancing of data-intensive technologies. A stream application can be represented by pipelined execution of multiple tasks. HW/SW partitioning plays an important role for selecting software execution or hardware acceleration for each task. Hardware-executed tasks usually perform faster at a cost of increased hardware area and higher power consumption. As tasks with similar function often exist in streaming applications such as MP3 decoder [2] and JPEG encoder [3], sharing of hardware accelerator among these tasks could reduce area consumption and may not degrade the system throughput if execution time of these tasks is insignificant compared to other tasks.

This paper formulates the HW/SW partitioning problem to maximize streaming throughput while meeting area constraint of a multi-processor system. Two greedy heuristic algorithms are proposed to produce fast near-optimal solutions, while branch-and-bound algorithm is proposed to replicate brute force result but with significant lower partitioning time by utilizing greedy results as initial best solution. The rest of the paper is organized as follow. Section 2 presents the related works in HW/SW partitioning. Section 3 describes the model of streaming application while Section 4 formulates the HW/SW partitioning problem. The proposed software-oriented and hardware-oriented greedy heuristic algorithms are presented in Section 5 while branch-and-bound algorithm is described in Section 6. Section 7 shows the experimental results on two test cases of MP3 decoder and JPEG encoder, as well as the partitioning speed benchmark against ground truth brute force algorithm. Section 8 concludes this paper.

2. RELATED WORKS

There are several significant researches in HW/SW partitioning automation to improve system performances in different aspects due to the increasing complexity of SoC. Traditionally, HW/SW partitioning is carried out manually based on system

designer's experience [4, 5]. Many approaches have been proposed recently to fulfill and optimize different objectives and costs. Although there is a wide variety of problem formulation and cost definition, they are highly dependent on targeted system architectures.

The most common problem described in literature, e.g [6–14] is the optimization of execution time, hardware area and communication cost, targeted for simple single-software single-hardware system. There are fairly little works conducted for throughput optimization compared to overall execution time due to the difficulty of its estimation and formulation. Hence, [2, 15–18] have integrated pipeline scheduling of tasks in order to construct throughput formulation together with other co-design constraints. Apart from that, other works have also incorporated different cost metrics and objectives in HW/SW partitioning, including power consumption [3, 19–21], and software memory usage [3, 19, 20], as summarized in Table 1.

Table 1. Related works in HW/SW partitioning

Ref.	**Objectives/Costs**						**Features**			
	Exec. time	Through-put	HW area	Comm.	SW mem.	Power consump.	Multi -proc.	HW sharing	Multi-choices HW	NoC
[2]		✓	✓	✓			✓			✓
[6-14]	✓		✓	✓						
[3, 19, 20]	✓		✓		✓	✓				
[15]		✓	✓							
[16, 17]		✓	✓				✓			
[18]		✓	✓			✓				
[21]	✓		✓			✓	✓			
[22]	✓		✓							
[23]	✓		✓	✓				✓		
[24]	✓		✓	✓					✓	

Other features such as multi-processor and hardware sharing have also been explored in literatures [2, 16, 17, 21, 23, 24] to further enhance the efficiency and optimize HW/SW partitioning for different modern SoC system. References [16, 17, 21] have proposed the HW/SW partitioning for multi-processor system by considering software parallelism. Reference [23] has proposed hardware accelerator sharing (among hardware tasks) to reduce area cost. Multiple hardware architecture choices is discussed in [24], allowing tasks to be mapped to different hardware accelerator alternatives with different performance and cost. On the other hand, a HW/SW partitioning for Network-on-Chip (NoC) has been proposed in [2], that includes the consideration of network router communication costs among tiles. Branch-and-bound algorithm has been proposed in [15, 18], although they are not targeted for multi-core with hardware sharing possibility.

This paper addresses HW/SW partitioning for optimization of throughput subject to a predefined hardware area constraint considering software parallelism (multi-processor system) and hardware sharing capability. To the best of our knowledge, there are no similar works highlighting the same objectives and features, thus distinguishing this work with others in literature.

3. APPLICATION MAPPING

Conventionally, HW/SW partitioning algorithm can be defined in the following notations. Task graph of an application can be characterized by a Directed Acyclic Graph (DAG), G = (T, E), $T = \{t_1, t_2, ..., t_n\}$ and $E = \{e_1, e_2, ..., e_n\}$ where s, h, a : $T \rightarrow R+$ and $c : E \rightarrow R^+$ as shown in Figure 1. Each task, t_i contains a tupple $< s_i, h_i, a_i >$ denoting software execution time, hardware execution time and hardware area cost, respectively. On the other hand, each edge, e_i includes c_{ij} which represents the communication cost between tasks t_i and t_j. All tasks are to be bi-partitioned into hardware executable, H or software executable, S, satisfying $H \cup S = T$ and $H \cap S = \emptyset$. Software processors are assumed to be available in the given system, thus their areas are not considered in the model. All software processors are also assumed to be identical where the software execution time for each task is similar on any software processor.

Traditionally, a task graph is bi-partitioned into hardware and software executables, targeting shorter execution time of the application. However in most stream applications such as multimedia programs, some tasks are split into several tasks with similar functions to increase the system throughput by executing these tasks concurrently using different cores (such as MP3 Decoder task in [2] and JPEG Encoder task in [3]). The existence of these similar tasks motivates the necessity of hardware cores sharing as an option to utilize area costs, giving rise to a bigger search space in HW/SW partitioning

problem.

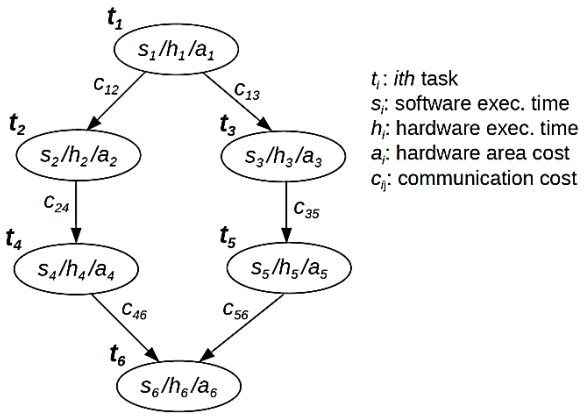


Figure 1. Different types of costs in task graph specify an application

As this paper focuses on throughput optimization in multi-processor system with hardware sharing capability, HW/SW partitioning is aiming to distribute tasks among all available software cores and hardware cores. Any task can be assigned to any software core. However, only tasks with similar functionality can share the same hardware cores. For instance, task graph in Figure 2 is mapped to a system consisting two software processors (PE_7 and PE_8). Core PE_5 is assigned to execute both similar tasks t_4 and t_5. Tasks that are mapped to software or shared hardware will render their respective hardware cores unused and unimplemented, thus at the same time, reducing hardware area cost.

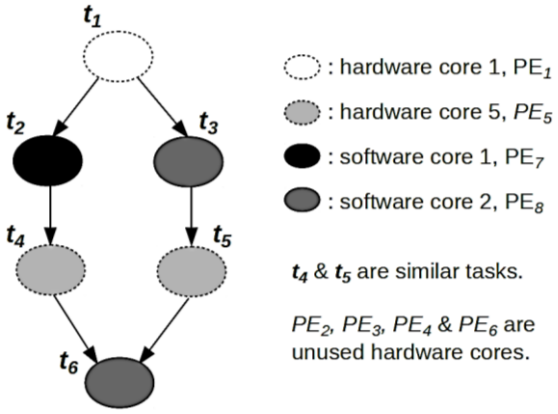


Figure 2. HW/SW partitioning in a system consisting two software processors with hardware sharing capability

System communication cost is highly dependent on the targeted multi-core architecture. Figure 3 summarizes possible communications between software and hardware cores. There are local connections and interconnection among software cores and hardware cores. In a multi-processor system, data sharing among the software processors include memory sharing, message passing interfaces (MPI), as well as packets forwarding through routers in NoC [25]. Most common communications between hardware and software are done through data passing from software memory to hardware core via system bus with handshaking protocol, rendering HW/SW interconnection the slowest. Hardware-to-hardware local connections are usually more efficient due to the possibility to utilize specialized interconnect interface such as dual-port RAM or FIFO.

As different communication interfaces have different latency and throughput, exact HW/SW interconnection delay cannot be directly estimated. With the possibility of having high throughput communications in all connections, this paper assumes that communication overheads are insignificant compared to task execution time, thus does not incorporate any communication costs in HW/SW partitioning problem (similar as [3, 15–22]).

Assuming all tasks are scheduled and pipelined-executed in the same time step, the overall system throughput is defined by the reciprocal of the time step, which is determined by the minimum allowable task processing time (critical time). Figure 4 illustrates the task executions time-line for a stream application based on task partitioning in Figure 2. Data are streamed through hardware (PE_1 to PE_6) and software cores (PE_7 and PE_8) according to a certain partitioning order. These data are required to be processed in six tasks by different cores at different time steps. PE_8 emerges as the slowest core with execution of two tasks, t_3 and t_6 every time step, thus resulting in the slowest task completion time (critical time) in each time

step to be $s_3 + s_6$.

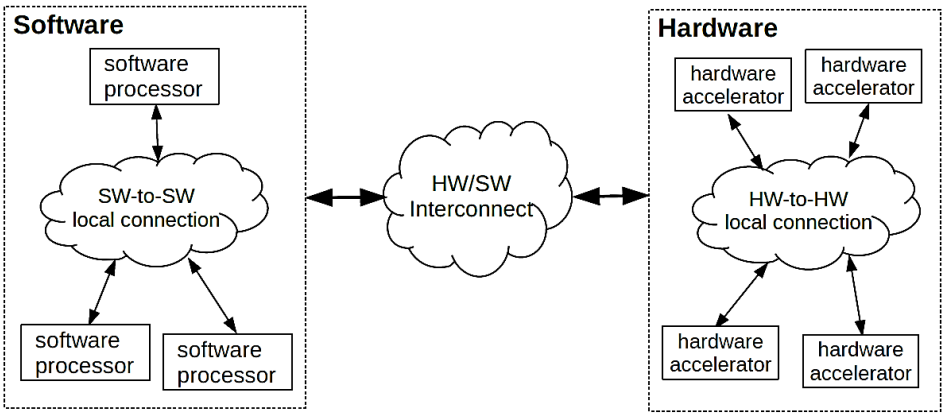


Figure 3. Communications among hardware and software tasks in a multi-core architecture

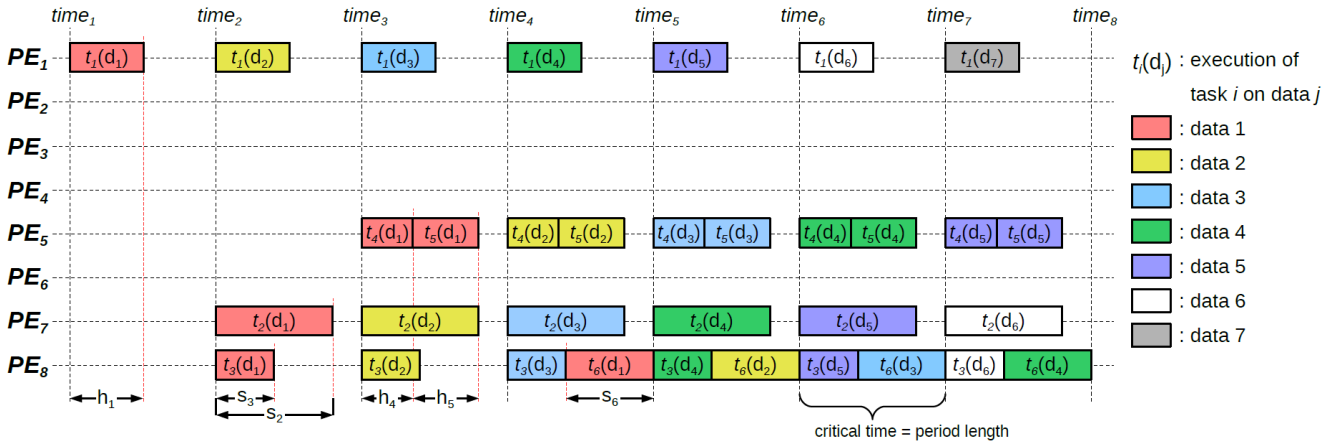


Figure 4. HW/SW pipelining of tasks with core sharing capability

PE_5 is shared by t_4 and t_5 as they are similar in function with both tasks are executed in serial. Although serial execution is slower compare to parallelism in their specialized hardware cores, the reduction in execution speed of these tasks does not affect the overall throughput as they are not the slowest (critical time) in the system. Hence, hardware sharing of the tasks is purely beneficial in area reduction with no trade-off in throughput.

4. PROBLEM FORMULATION

Given a DAG representation of a stream application with N_t tasks and a multi-core system with N_s software processor, HW/SW partitioning partitions the tasks to N_s software cores and N_h hardware cores which maximizes the system throughput, subject to an area constraint, A. Assuming all tasks have their own specialized hardware cores, therefore the number of hardware cores is equal to number of tasks ($N_h = N_t$). Thus, the number of total cores, N_c for a given problem is $N_c = N_s + N_t$.

As hardware sharing is possible in this partitioning problem, the allowable task-to-core mapping is defined by all available software cores and allowable hardware cores. The allowable task-to-core mapping can be represented in a mask M , constructed of a $N_c \times N_t$ matrix with column index i and row index j indicate task identifier and core identifier respectively, and $m_{ij} \subset M$. If $m_{ij} = 1$, then t_i are permitted to be assigned to core PE_j. For instance, Equation (1) illustrates an example of mask M for the tasks graph described in Figure 2, assuming the targeted system contains two software processors.

The mask M is sorted in increasing row index j from hardware cores to software cores. Since all tasks are assumed to have their own specialized hardware cores, for simplicity, these cores are allocated with the core identifiers similar to their respective task identifiers (i = j). All software cores are assumed to be able to execute any task. As a result, an identity matrix is obtained for $1 \leq j < N_t$ while an all-ones matrix for $N_t \leq j < N_c$.

$$M = \begin{array}{c} \\ PE_1 \\ PE_2 \\ PE_3 \\ PE_4 \\ PE_5 \\ PE_6 \\ PE_7 \\ PE_8 \end{array} \begin{array}{c} \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array} \quad (1)$$

On the other hand, similar tasks may have two or more admissible spaces (HW cores) due to the possibility of hardware sharing. The feasible way for hardware sharing can be formulated as a combination selection, $\sum_{r=1}^{n-1} {}^{n}C_r$, where n is the number of similar tasks and r is the number of shared hardware cores. To reduce the search space by eliminating similar hardware sharing selections (as in Equation (1), assigning both t_4 and t_5 to shared core PE_4 or PE_5 give similar result), each task t_i is only assigned to a core PE_j whose identifier is greater or equal to the task identifier, $j \geq i$. The total possible permutation search space can be easily determined from M and is evaluated as $\prod_{i=1}^{N_t} \sum_{j=1}^{N_c} m_{ij}$.

In addition, mask M is capable of distinguishing differences in costs for each task t_i assigned in different cores PE_j. Along the increasing j, each task can be mapped to different core, starting from its own specialized hardware core, followed by sharing hardware core, and finally on software cores. Thus, it is undeniable that assigning tasks to greater j reduces more hardware area cost while increasing its execution time.

Similarly, HW/SW assignment action can be represented mathematically by a $N_c \times N_t$ sized assignment matrix X. $x_{ij} = 1$ denotes the mapping of t_i to core PE_j. Since each task can only be assigned to one available core and all tasks must be mapped at the end of the P partition, summation of each column in X must equal to one, formulated as $\sum_{j=1}^{N_c} x_{ij} = 1, \forall i = \{1, 2, \ldots N_t\}$. Equation (2) illustrates an example of assignment matrix X for the mapping in Figure 2. Only one task assignment appears in each column while multiple assignments appear in row with software cores (PE_8) or shared hardware core (PE_5). Row without any assignment indicates the unimplemented cores.

$$X = \begin{array}{c} \\ PE_1 \\ PE_2 \\ PE_3 \\ PE_4 \\ PE_5 \\ PE_6 \\ PE_7 \\ PE_8 \end{array} \begin{array}{c} \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array} \quad (2)$$

With the aid of the assignment matrix, X, the partitioning problem discussed in this paper is formulated as a minimization problem in Equation (3), aiming to find maximum throughput (minimum critical time) subject to area constraint, A. H_t and S_t denote the execution time of the slowest hardware and software respectively, formulated in Equation (4) and Equation (5) respectively.

$$\begin{aligned} &minimize \quad \max(H_t, S_t) \\ &subject\ to \quad \sum_{j=1}^{N_t} \left(\max_{1 \leq i \leq N_t} x_{ij}.a_i \right) \leq A, x_{ij} \in \{0,1\} \end{aligned} \quad (3)$$

$$H_t = \max_{1 \leq i \leq N_t} \left(\sum_{i=1}^{N_t} h_i.x_{ij} \right) \quad (4)$$

$$S_t = \max_{N_t \leq j \leq N_c} \left(\sum_{i=1}^{N_t} s_i.x_{ij} \right) \quad (5)$$

Based on Equation (3), the system throughput and hardware area usage for any given mapping can be easily enumerated. Critical time is determined by identifying the longest execution time of each core while execution time of each core is total execution time of the assigned tasks. Hardware area is computed by summation of all hardware areas except for unused cores.

5. GREEDY ALGORITHM

Two greedy heuristic algorithms, *Alg-greedy1* and *Alg-greedy2* are proposed in this paper to optimize HW/SW partitioning problem described in Section 4. The former is software-oriented while the latter is hardware-oriented, both aiming to maximize throughput (i,e: minimize critical time) for a given area constraint, without exploring all possible partitioning.

Similar to conventional greedy algorithm [7], both algorithms exploit the profit-to-cost ratio to make a decision. As the proposed greedy algorithms are intended to maximize throughput, different profit-to-cost ratio is proposed. As each task is allowed to be assigned to more than two cores with different profit or cost, the profit-to-cost ratio of each task has to be calculated specifically based on the execution speed of the tasks on each allowable cores. Assuming that a task t_i is pre-assigned to core PE_j, the profit-to-cost ratio, PCR of the task to move to another core PE_k is formulated in Equation (6). If the task is moved among hardware cores, the PCR is calculated by dividing its hardware speed to area. However, if a software core is involved either as a source or a destination core, PCR is calculated as software speed-to-area ratio.

$$PCR_{ijk} = \begin{cases} \frac{h_i}{a_i} & if\ j \le N_t\ and\ k \le N_t \\ \frac{s_i}{a_i} & otherwise \end{cases} \tag{6}$$

$\forall i = \{1, 2, \ldots N_t\}$, $\{\forall i, \forall k\} = \{1, 2, \ldots N_c\}$, where PCR_{ijk} is profit-to-cost ratio for moving task t_i from PE_j to PE_k.

5.1 Greedy 1 (Alg-greedy1)

This algorithm first considers all-software system. Then, tasks are converted to hardware one-by-one to improve throughput until no more tasks could be repartitioned or mapped to hardware that preserves the area constraint, as shown in Algorithm 1.

ALGORITHM 1: *Alg-greedy1*

```
generate PCR matrix
distribute all tasks to SW cores
while 1 do
  best mapping= current mapping
  find slowest core
  for t_i= tasks in slowest core sorted in descending PCR do
    if upward(-j) move is available then
      move t_i to next upward(-j) core
      if area after move ≤ area constraint then
        break for loop
      else
        undo move
      end if
    end if
  end for
  if no more available move then
    break while loop
  end if
end while
return best mapping
```

It is obvious that tasks can only be moved in upward (-j) direction represented in mask M to gain higher throughput by investing area cost. Each task will have a predefined move one step at a time from software to shareable hardware, and finally as dedicated hardware core. Thus, by identifying sources and destination cores for each step, PCR can be pre-generated in a matrix as summarized in Equation (7) based on Equation (6).

$$PCR = \begin{array}{c} \\ j = 1 \\ \vdots \\ j = N_t \\ j = N_{t+1} \\ \vdots \\ j = N_c \end{array} \begin{array}{c} i = 1 \quad \ldots \quad i = N_t \\ \begin{bmatrix} h_i/a_i & \ldots & h_i/a_i \\ \vdots & \ddots & \vdots \\ h_i/a_i & \ldots & h_i/a_i \\ s_i/a_i & \ldots & s_i/a_i \\ \vdots & \ddots & \vdots \\ s_i/a_i & \ldots & s_i/a_i \end{bmatrix} \end{array} . M \tag{7}$$

All tasks are initially distributed among software cores as equal as possible. This is achieved by sorting all tasks based on execution time in descending order, followed by assigning them one-by-one to the least utilized core in each step. As system throughput is always determined by the slowest core, only tasks assigned in this core will be examined and remapped at each step. Based on the obtained PCR matrix, the algorithm iterates and reassigns the highest PCR task upward (-j) with the condition that area cost after reassignment does not exceed the predefined area constraint. Termination of the loop occurs when there is no further move which could satisfy the area constraint, returning the best mapping as the greedy result.

5.2 Greedy 2 (Alg-greedy2)

Alg-greedy2 initializes all tasks as dedicated hardware cores, and reassigns tasks one-by-one to software to reduce area cost

by sacrificing throughput until area constraint is satisfied, as illustrated in Algorithm 2. Similarly, a PCR matrix is pre-generated based on Equation (6) before the tasks reassignments. Starting by assigning all tasks in uppermost location of matrix M, the movable task with lowest PCR is mapped downward (+j) in each iteration until area constraint is attained. Lastly, all software tasks are distributed as equal as possible for all software cores.

ALGORITHM 2: *Alg-greedy2*

```
generate PCR matrix
assign all tasks to dedicated HW cores
while area≥ area constraint do
  best mapping= current mapping
  for t_i= all tasks sorted in ascending PCR do
    if downward (+j) move for t_i is available then
      if if downward (+j) move is SW core then
        move t_i to least utilized SW core
      else
        move t_i to next downward (+j) core
      end if
      break for loop
    end if
  end for
end while
redistribute all SW tasks equally
return best mapping
```

6. BRANCH-AND-BOUND

The branch-and-bound algorithm (*Alg-bnb*) is utilized to produce optimal HW/SW mapping based on area constraint. As mentioned in Section 4, HW/SW partitioning problem involves permutation space where each task has two or more core assignment options given an allowable task-to-core mapping. Thus, the entire design space can be specified in a balanced search tree constructed based on the mask M. Each node represents an unmapped task and the branches from each node are the allowable task-to-core mapping of the represented task, resulting in the depth of tree equals to the number of tasks. For instance, Figure 5 illustrates the search tree from the mask M described in Equation (1).

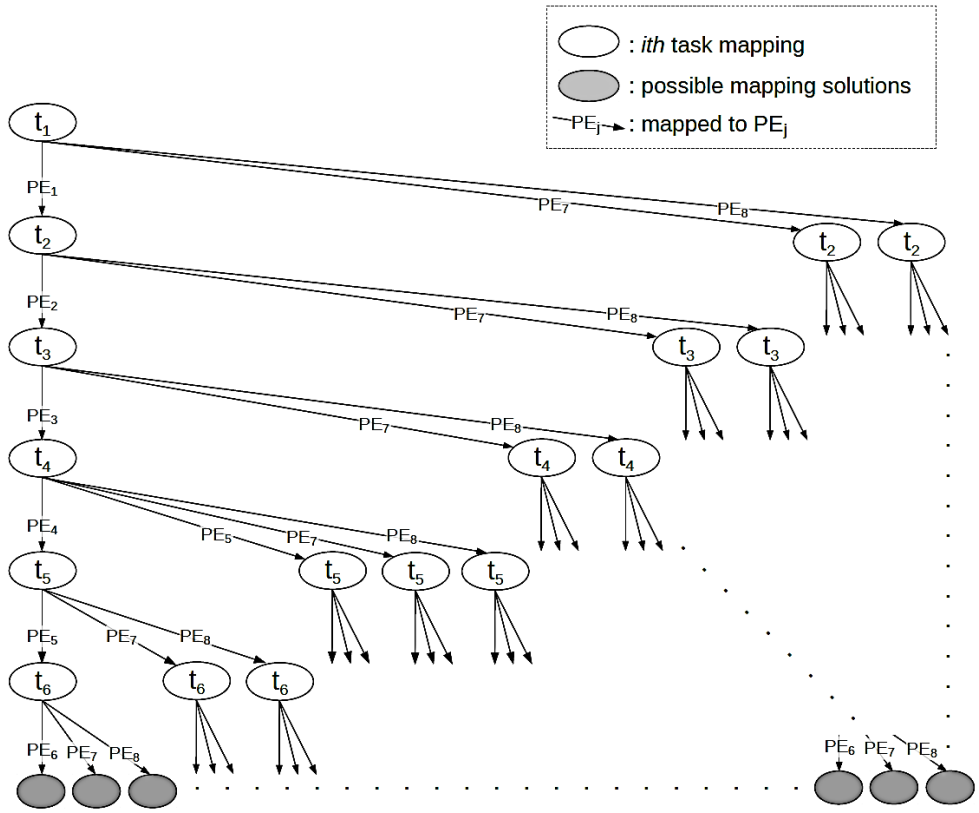


Figure 5. Example of a balanced search tree containing entire design space for HW/SW partitioning

Similar to conventional branch-and-bound [15, 18], *Alg-bnb* explores all possible solutions by branching from the top of the tree. At each level of search tree, the algorithm decides the mapping of each unassigned task. Several bounding criteria are applied to discard sub-tree of any branch that cannot produce any feasible result or all possible solutions from the branch will not produce better result than current best solution. *Alg-bnb* is summarized in Algorithm 3.

ALGORITHM 3: *Alg-b&b*

```
sort tasks in descending area
best critical time, B= best of Alg-greedy1 and Alg-greedy2
best mapping= best of Alg-greedy1 and Alg-greedy2
add root node of search tree into a min-heap, H
while H is not empty do
  read and remove min upper bound node, N from H
  for C_i= children of N do
    if (CurrentAreaUsage(C_i)> area constraint, A) then
      eliminate C_i
    else if (LowerBound(C_i)> B) then
      eliminate C_i
    else if C_i is a solution node then
      B= min(B, CriticalTime(C_i))
      update best mapping
    else if (UpperBound(C_i)> B) then
      eliminate C_i
    else
      add C_i into H
    end if
  end for
end while
return best mapping
```

The *Alg-bnb* algorithm starts by assigning the best greedy result of *Alg-greedy1* and *Alg-greedy2* as the initial best throughput. Best-first search are applied, traversing down the tree by prioritizing the branches with the best estimated upper bound. The maximum throughput (i.e., minimum critical time) node is chosen to spawn its children in each step. Since hardware sharing gives better solution in most of the time, branches with hardware sharing are chosen when there are two or more branches with the best upper bound value.

There are three bounding criteria to be checked in the following order where branches are eliminated if any of the criteria are met:

1. Current area usage of the partial mapping exceeds the area constraint, A.
2. Current throughput (lower bound) is lower than the best found throughput.
3. Maximum attainable throughput (upper bound) is lower than the best found throughput.

As the upper bound check is comparatively time consuming, it is performed after the other two criteria. When reaching a solution node, the best throughput is updated if a better solution is achieved. The algorithm terminates after all branches have been explored or eliminated.

6.1 Upper Bound Estimation

Unlike lower bound which is straight-forward, upper bound requires several changes on the problem model. The upper bound at any given branch with different partial-mapped of HW/SW can be solved with the assumption that every unassigned tasks can be fragmented to smaller part with smaller area usage and execution time. Based on the assumption, several changes have been made in HW/SW partitioning model as follow:

1. The slowest core can distribute its tasks to other core partially until the area constraint is met or no other possible core to execute the tasks.
2. All software tasks are always distributed in perfect equal with each core holding the identical fractions of every software tasks. Hence, all software cores are estimated as one combined core which executes task faster. Hence all software time are decreased by a factor of the number of software cores, $s'_i = s_i/N_s$, $\forall i = \{1, 2, \ldots N_t\}$, where s'_i is the combined software time.
3. Hardware sharing is allowed for fractioned tasks. All equal fractions of similar tasks can use the same hardware core with only one fraction of area consumption. This can be estimated by reducing the area usage for shared hardware. If several tasks share a (shared) hardware, their area usage is divided by the number of allowable task-to-core mapping of the hardware core specified in mask M. Thus, the area usage of each task varies when occupying

different hardware cores which can be formulated in Equation (8), acting as a modified area usage during hardware sharing for best case estimation.

$$a'_{ij} = \frac{a_i}{\sum_{i=1}^{N_t} m_{ij}} \tag{8}$$

$\forall i = \{1, 2, \ldots N_t\}$, and $\forall j = \{1, 2, \ldots N_t\}$, where a'_{ij} is the area usage of task t_i occupying hardware core, PE_j. The overall area used for all fractionally-mapped tasks can be estimated as $\sum_{i=1}^{N_t} \sum_{j=1}^{Nt} a'_{ij}.x'_{ij}$, with x'_{ij} is the fractioned assignment of task t_i on core PE_j.

By establishing these changes, the upper bound can be computed in software-oriented way similar to *Alg-greedy1*. The algorithm starts with initialization of all unmapped tasks, t'_i to the combined software core. t'_i is then distributed into hardware core (in -j direction of mask M) fractionally prioritizing highest modified profit-to-cost ratio, PCR′. This modified PCR' is calculated based on the modified software time, s'_i and area usage, a'_{ij} as formulated in Equation (9). This approach will increase the throughput (decrease critical time) at the best rate for each unit area spent.

$$PCR'_{ijk} = \begin{cases} \frac{h_i}{a'_{ik}} & if\ 1 \le j < N_t \\ \frac{s_i}{a'_{ik}} & if\ N_t \le j < N_c \end{cases} \tag{9}$$

$\forall i = \{1, 2, \ldots N_t\}$, and $\forall j = \{1, 2, \ldots N_c\}$, where PCR'_{ij} is profit-to-cost ratio for moving task t_i from PE_j to PE_k.

The upper bound estimation algorithm is summarized in Algorithm 4. In each iteration of the algorithm, fractions of tasks are distributed from software to hardware to increase speed by utilizing hardware area. During task distribution, the software cores are always kept as the slowest (in critical time) among all cores. Hence, if the software execution time decreases until it reaches any of the hardware execution time, all of these slow cores (i.e, cores with execution time equals to critical time) will distribute their highest PCR′ tasks fractionally to other allowable cores at the same decreasing rate in execution time, therefore ensuring the increase of system throughput (i.e., decrease of critical time) by maintaining equality of execution time in all slow cores.

ALGORITHM 4: *Upper bound estimation*

```
combine SW cores (re-calculate SW time, s'_i for unmapped tasks, t'_i)
update area usage, a'_ij of t'_i
generate PCR' matrix
assigned t'_i to combined SW core
upper bound= current throughput
while area usage ≤ area constraint, A do
  for PE_j= cores with execution time equals to critical time do
    T'_j= moveable t'_i with highest PCR' on PE_j
    if no moveable task then
      return upper bound
    end if
  end for
  determine the moving fraction, f_j of each T'_j
  move all T'_j in fraction of f_j upward (-j direction in M)
  if area usage ≤ area constraint, A then
    update upper bound throughput (critical time)
  end if
end while
return upper bound
```

As the granularity of the fractions in each iteration will affect the algorithm time consumption, fractions are determined based on current execution times for all cores and residual area (different between area constraint and mapped area). It is obvious that the same tasks (in slow cores) have to be distributed until the following conditions are met:

1. The critical time reaches the next slowest execution time among non-slow core.
2. Any of the distributing task has reached its minimum possible execution time (maximum speed).
3. The residual area has depleted.

The critical time decrease gradually as the distribution of the tasks in all slow cores. If the area allows, the critical time would eventually equals to the execution time of the slowest non-slow core (condition 1), making this core as a new member to the slow cores. Any additional area will be used to distribute the tasks in the newly updated slow cores. The critical time cannot be decreased any further if any of the tasks (in slow cores) has reached its maximum speed (condition 2) or the area constraints are met (condition 3).

Hence, the fraction of execution time (area) to be distributed at each step is the minimum time (area) gap for these conditions to happen, min(G_1, G_2, G_3), where G_1, G_2 and G_3 are the time (area) gaps to satisfy condition 1, 2 and 3 respectively. The iteration also terminates when the area constraint is met (condition 2) or any of the slowest cores cannot distribute its tasks any more to other cores (condition 3), producing the upper bound throughput as an estimation of best attainable throughput for any partial mapping.

7. NUMERICAL EXPERIMENTATION

7.1 Test Cases: MP3 decoder and JPEG encoder

The proposed greedy algorithms are coded in Matlab environment. Two common data intensive streaming multimedia applications: MP3 decoder (see [2]) and JPEG encoder (see [3]) are selected as test cases. MP3 decoder is tested for HW/SW partitioning of two software cores and three software cores systems while JPEG encoder is only tested for two software cores system. These experiments have been conducted using both proposed algorithms to maximize system throughput (minimize critical time) for different area constraints.

The task graph describing the specifications of MP3 decoder is illustrated in Figure 6. On the other hand, task graph of JPEG encoder is illustrated in Figure 7. The specifications of tasks area are shown in Table 2. There are several similar tasks in both MP3 decoder and JPEG encoder applications, providing hardware sharing possibility which can be easily specified in mask M.

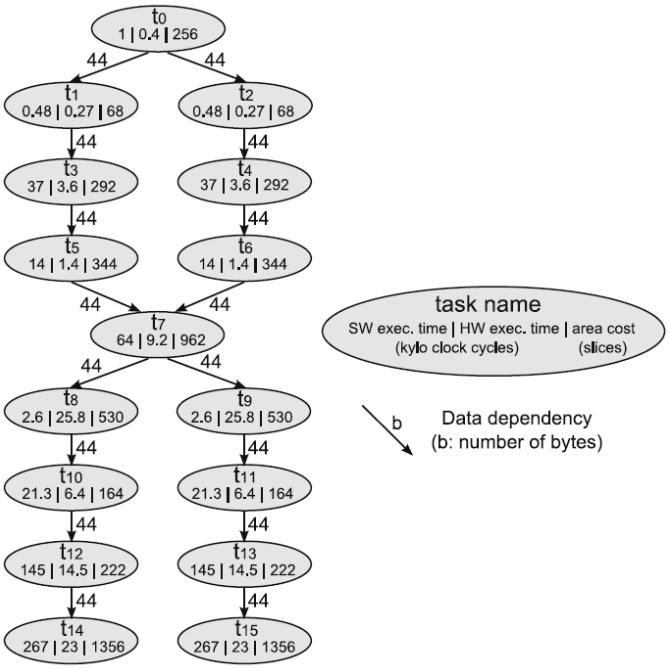


Figure 6. MP3 decoder application graph in [2]

The results of critical time-to-area constraints of greedy algorithms are shown in Figure 8. *Alg-bnb* produces optimal throughput for any area constraint, it emerges as the ground truth solution for all test cases. When area constraint is too low to be occupied by any of the hardware cores, the system throughput is determined by equal distribution of total software execution time to the number of software cores. As area constraint increases, the proposed algorithms attempt to utilize the available resources effectively. This improves throughput by choosing the correct tasks for hardware execution as well as hardware sharing. Hence, even with infinite area constraints, all tasks do not necessary consume hardware area. Several tasks can still be performed in software or in shared hardware, resulting in the maximum achievable throughput (minimum possible critical time) attained without the need for full hardware acceleration.

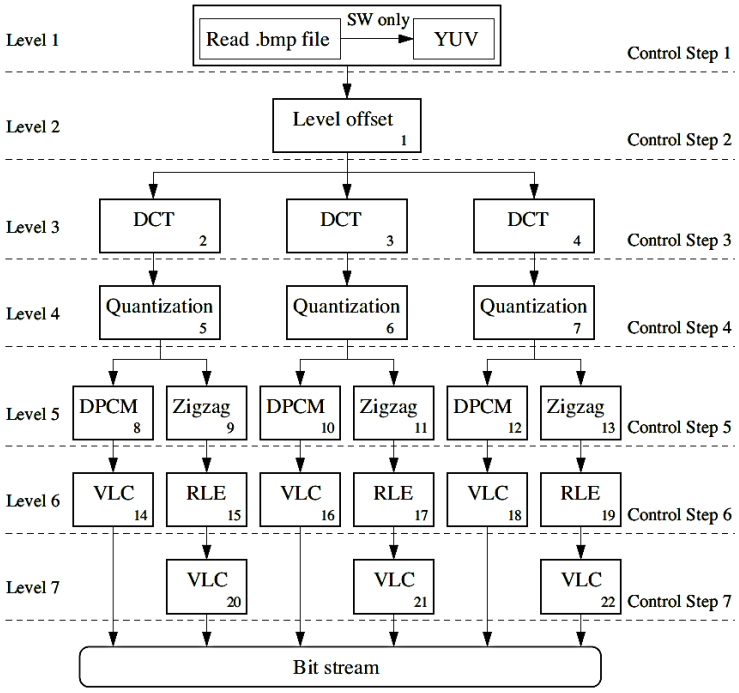


Figure 7. JPEG encoder application graph in [3]

Table 2. Task specifications for JPEG encoder [3]

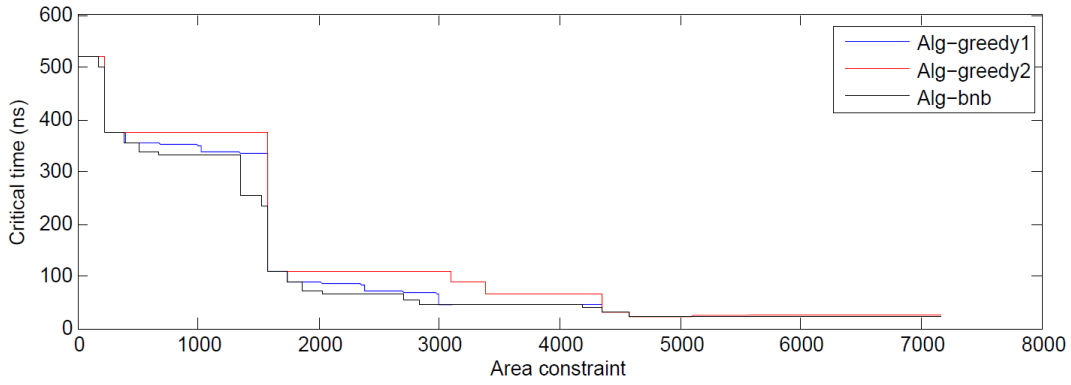
Task	Module	Execution time		HW area, a_i ($\times 10^{-3}$)
		h_i (ns)	s_i (μs)	
t_1	Level offset	155.264	9.38	7.31
t_2	DCT	1844.822	20000	378
t_3	DCT	1844.822	20000	378
t_4	DCT	1844.822	20000	378
t_5	Quantization	3512.32	34.7	11
t_6	Quantization	3512.32	33.44	9.64
t_7	Quantization	3512.32	33.44	9.64
t_8	DCPM	5.334	0.94	2.191
t_9	ZigZag	399.104	13.12	35
t_{10}	DCPM	5.334	0.94	2.191
t_{11}	ZigZag	399.104	13.12	35
t_{12}	DCPM	5.334	0.94	2.191
t_{13}	ZigZag	399.104	13.12	35
t_{14}	VLC	2054.748	2.8	7.74
t_{15}	RLE	1148.538	43.12	2.56
t_{16}	VLC	2197.632	2.8	8.62
t_{17}	RLE	1148.538	43.12	2.56
t_{18}	VLC	2197.632	2.8	8.62
t_{19}	RLE	1148.538	43.12	2.56
t_{20}	VLC	2668.288	51.26	1.91
t_{21}	VLC	2668.288	50	1.91
t_{22}	VLC	2668.288	50	1.91

7.2 Optimality of Greedy Algorithms

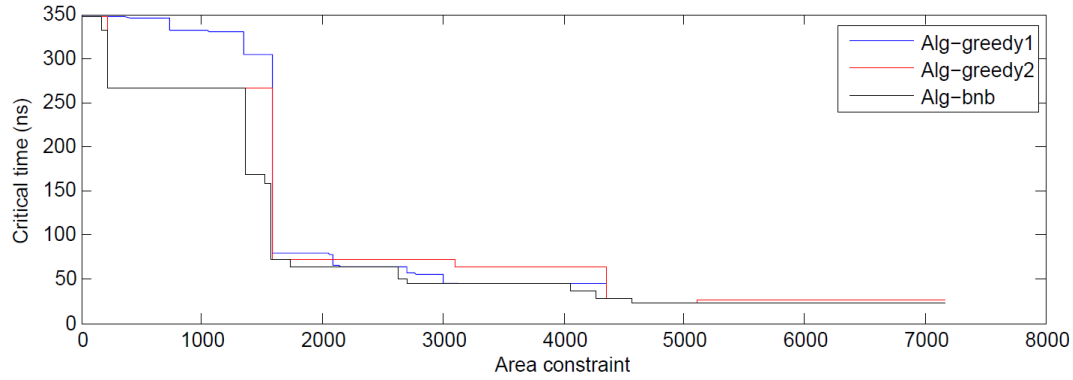
Solutions from the greedy algorithms are compared with ground truth result to obtain their optimality. *Optimality* is defined as the average ratio of ground truth result to the obtained greedy result for all DAGs, and is illustrated in Equation (10). An obtained result is 100% optimal if it is identical with ground truth result.

$$Optimality = 100 \times \underset{all\ DAG}{\text{average}} \left(\frac{R_t}{R_g}\right) \tag{10}$$

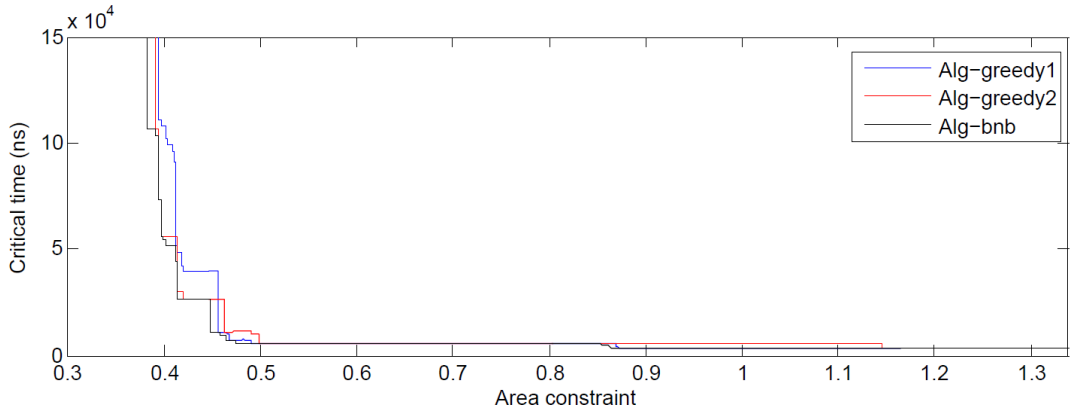
where R_g and R_t are critical time results from greedy and brute force ground truth respectively.



(a) Two SW cores for MP3 decoder



(b) Three SW cores for MP3 decoder



(c) Two SW cores for JPEG encoder

Figure 8. Critical time-to-area constraint result of MP3 and JPEG streaming applications

Apart from the applications in test cases, random task graphs with different sizes are also utilized to determine the performance for each algorithm. Random graphs are generated using TGFF [26], with 100±90 hardware speed, 1000±900 software speed and 100±50 hardware area with 2 and 3 software processors. The optimality of the proposed greedy heuristics are shown in Table 3. Both *Alg-greedy1* and *Alg-greedy2* results are near-optimal with an average of 87.7% and 84.2% from ground truth solution respectively. However, the optimality for both greedy algorithms differ for each DAG.

7.3 Partitioning Speed Benchmark

The partitioning time of the proposed greedy heuristics and branch-and-bound algorithms are benchmarked using the test applications and random graphs. Estimated partitioning time for brute force method is also considered in the comparison where the estimation can be done by extrapolating the time taken from a fraction of solutions to total possible permutations. The partitioning time to permutation size for all algorithms is shown in Figure 9.

Table 3. Optimality of proposed greedy algorithms.

Test Case	No. of Tasks	No. of SW Cores	Optimality (%)	
			Alg-greedy1	*Alg-greedy2*
MP3	16	2	94.2	77.1
MP3	16	3	92.1	84.1
JPEG	22	2	95.0	90.9
Random1	14	2	93.2	90.3
Random1	14	3	87.3	93.0
Random2	7	2	81.7	84.7
Random2	7	3	85.2	91.9
Random3	10	2	91.7	76.9
Random3	10	3	89.0	82.1
Random4	7	2	88.2	77.7
Random4	7	3	92.0	87.4
Random5	13	2	76.7	79.4
Random5	13	3	83.6	81.3
Random6	9	2	89.6	76.5
Random6	9	3	76.5	89.7

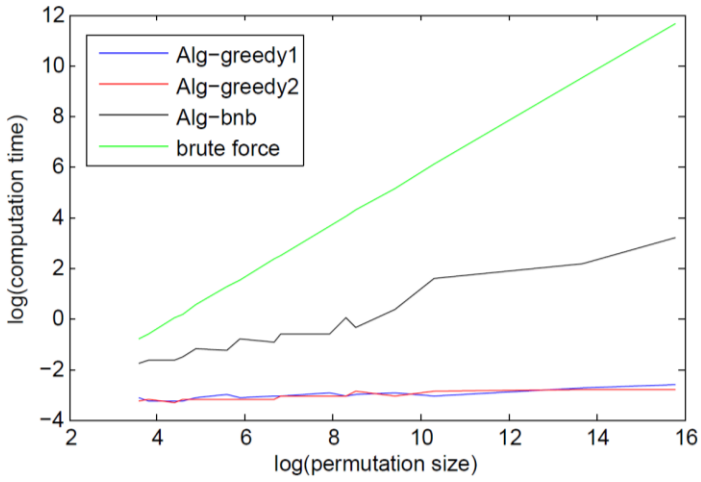


Figure 9. Critical time-to-area constraint result of MP3 and JPEG streaming applications

Both greedy algorithms have almost similar performances averaging with 2.6790e+13 and 3.0384e+13 times faster for *Alg-greedy1* and *Alg-greedy2* respectively, compared to brute force algorithm. *Alg-bnb* performs up to of 2.4741e+8 times faster than brute force for the largest permutation size, proven to be a more practical choice for generating optimal solution. The partitioning speeds with respect to exhaustive brute force method of all proposed algorithms increase exponentially with permutation size.

8. CONCLUSION

Efficient partitioning of HW/SW tasks produces cost-effective solution in throughput-to-area trade off. This paper formulated the optimization problem of streaming throughput in MPSoC with hardware sharing capability given a predefined constraint of hardware area cost. Software-oriented and hardware-oriented greedy heuristic algorithms were proposed to maximize application throughput without performing full search. Branch-and-bound algorithm with best-first traversing technique was also proposed to produce the optimal HW/SW partitioning result in faster partitioning time compared to exhaustive brute force.

These algorithms were empirically tested on MP3 decoder and JPEG encoder test cases that are two well-known streaming multimedia applications. Results show that these greedy heuristics compute 13 order of magnitude faster than the ground truth brute force method while give a near-optimal solution with 87.7% and 84.2% for *Alg-greedy1* and *Alg-greedy2* respectively. With the aid of both greedy algorithms to provide the initial best solution, proposed branch-and-bound algorithm is able to produce ground truth result up to 8th order of magnitude faster than the ground truth brute force method. The proposed approach may be potentiality extended to more specific architecture with formulated communication costs, thus resulting in more sophisticated HW/SW partitioning.

ACKNOWLEDGMENT

This work is supported in part by the Collaborative Research in Engineering, Science & Technology (CREST) grant P17C114 (UTM vote no. 4B176) and Universiti Teknologi Malaysia Matching grant (UTM vote no. 00M75).

REFERENCES

[1] S. Wang and S. Dey, "Adaptive mobile cloud computing to enable rich mobile multimedia applications," IEEE Transactions on Multimedia, vol. 15, no. 4, pp. 870–883, 2013.

[2] S. Le Beux, G. Bois, G. Nicolescu, Y. Bouchebaba, M. Langevin, and P. Paulin, "Combining mapping and partitioning exploration for noc-based embedded systems," Journal of Systems Architecture, vol. 56, no. 7, pp. 223–232, 2010.

[3] T.-Y. Lee, Y.-H. Fan, Y.-M. Cheng, C.-C. Tsai, and R.-S. Hsiao, "Enhancement of hardware-software partition for embedded multiprocessor fpga systems," in Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2007. IIHMSP 2007., vol. 1. IEEE, 2007, pp. 19–22.

[4] B. Mei, P. Schaumont, and S. Vernalde, "A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems," 2000.

[5] F. Clouté, J.-N. Contensou, D. Esteve, P. Pampagnin, P. Pons, and Y. Favard, "Hardware/software co-design of an avionics communication protocol interface system: an industrial case study," in Proceedings of the Seventh International Workshop on Hardware/Software Codesign, 1999.(CODES'99). IEEE, 1999, pp. 48–52.

[6] P. Arató, Z. Á. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 10, no. 1, pp. 136–156, 2005.

[7] W. Jigang, T. Srikanthan, and G. Chen, "Algorithmic aspects of hardware/software partitioning: 1d search algorithms," IEEE Transactions on Computers, vol. 59, no. 4, pp. 532–544, 2010.

[8] G. Li, J. Feng, C. Wang, and J. Wang, "Hardware/software partitioning algorithm based on the combination of genetic algorithm and tabu search," Engineering Review, vol. 34, no. 2, pp. 151–160, 2014.

[9] G. Lin, W. Zhu, and M. M. Ali, "A tabu search-based memetic algorithm for hardware/software partitioning," Mathematical Problems in Engineering, vol. 2014, 2014.

[10] J. Wu, P. Wang, S.-K. Lam, and T. Srikanthan, "Efficient heuristic and tabu search for hardware/software partitioning," The Journal of Super-computing, vol. 66, no. 1, pp. 118–134, 2013.

[11] H. Ye and J.-G. Wu, "Computing models and algorithms for complex co-design systems," Journal of University of Electronic Science and Technology of China, vol. 40, no. 3, pp. 333–345, 2011.

[12] X. Zhao, H. Zhang, Y. Jiang, S. Song, X. Jiao, and M. Gu, "An effective heuristic-based approach for partitioning," Journal of Applied Mathematics, vol. 2013, 2013.

[13] J. Henkel and R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 9, no. 2, pp. 273–289, 2001.

[14] A. Kalavade and E. A. Lee, "The extended partitioning problem: hardware/software mapping, scheduling, and implementation-bin selection," Design Automation for Embedded Systems, vol. 2, no. 2, pp. 125–163, 1997.

[15] K. S. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 10, no. 3, pp. 193–208, 2002.

[16] S. Bakshi and D. D. Gajski, "Hardware/software partitioning and pipelining," in Proceedings of the 34th annual Design Automation Conference. ACM, 1997, pp. 713–716.

[17] S. Bakshi and D. D. Gajski, "Partitioning and pipelining for performance-constrained hardware/software systems," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 7, no. 4, pp. 419–432, 1999.

[18] N. N. Bình, M. Imai, A. Shiomi, and N. Hikichi, "A hardware/software partitioning algorithm for designing pipelined asips with least gate counts," in Design Automation Conference Proceedings 1996, 33rd. IEEE, 1996, pp. 527–532.

[19] M. B. Abdelhalim and S.-D. Habib, "An integrated high-level hardware/software partitioning methodology," Design Automation for Embedded Systems, vol. 15, no. 1, pp. 19–50, 2011.

[20] P. K. Nath and D. Datta, "Multi-objective hardware–software partitioning of embedded systems: A case study of jpeg encoder," Applied Soft Computing, vol. 15, pp. 30–41, 2014.

[21] E. Sha, L. Wang, Q. Zhuge, J. Zhang, and J. Liu, "Power efficiency for hardware/software partitioning with time and area constraints on mpsoc," International Journal of Parallel Programming, pp. 1–22, 2013.

[22] A. Bhattacharya, A. Konar, S. Das, C. Grosan, and A. Abraham, "Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm," in International Conference on Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008. IEEE, 2008, pp. 171–176.

[23] J. Jeon and K. Choi, "Loop pipelining in hardware-software partitioning," in Design Automation Conference 1998. Proceedings of the ASP-DAC'98. Asia and South Pacific. IEEE, 1998, pp. 361–366.

[24] J. Wu, Q. Sun, and T. Srikanthan, "Algorithmic aspects for multiple-choice hardware/software partitioning," Computers & Operations Research, vol. 39, no. 12, pp. 3281–3292, 2012.

[25] D. A. Patterson and J. L. Hennessy, Computer organization and design: the hardware/software interface. Newnes, 2013.

[26] R. P. Dick, D. L. Rhodes, and W. Wolf, "Tgff: task graphs for free," in Proceedings of the 6th international workshop on Hardware/software codesign. IEEE Computer Society, 1998, pp. 97–101.