Task Allocation Strategies for FPGA Based Heterogeneous System on Chip

Atanu Majumder, Sangeet Saha(✉), and Amlan Chakrabarti

A.K. Choudhury School of IT, University of Calcutta, Kolkata, India
a.majumder007@gmail.com, sangeet.saha87@gmail.com, amlanc@ieee.org

Abstract. FPGA based heterogeneous System On Chips (SOCs) have become a prospective processing platform for modern performance-sensitive systems, like automotive, avionics, chemical reactor etc. In such system, "*makespan*" time minimization plays a crucial role to achieve higher throughput as well as performance efficiency and thus, efficient task allocation schemes are indeed essential. This paper presents two task allocation algorithms for such FPGA based heterogeneous SOCs. The first allocation strategy is based on well known "Branch and Bound" optimization technique. Secondly, we proposed a novel heuristic based allocation mechanism, *TAMF (**T**ask **A**llocation **M**echanism for **F**PGA based heterogeneous SOC)*. The simulation based experimental results reveal that both the strategies are able to provide lower *makespan* time over various simulation scenarios with acceptable runtime overheads. Achieved simulation results are further tested through a validation, carried out on practical ZYNQ SOC platform using standard benchmark task sets.

Keywords: Tasks allocation · FPGA · Branch and bound · *makespan* time

1 Introduction

Embedded systems are now-a-days employing FPGAs as a prospecting computing platform in various fields, ranging from avionic and automotive systems, nuclear reactors [1] to synthetic vision, object tracking [2] etc. These FPGAs provide the performance efficiency of a dedicated hardware as well as flexibility of a general purpose processor. Recently, FPGAs are also being integrated in *System on Chip (SOC)* along with pre-fabricated Hard-core processors and Soft-core processors (Constructed using FPGA logic) [3]. Such "Heterogeneous" systems contain different Processing Elements (PEs) with certain performance characteristic and can execute a set of tasks by assigning them on suitable PEs.

As an interesting example, such heterogeneous SOCs may be used as a performance efficient processing platform for automotive systems. In automotive systems [4], an ample amount of routine tasks need to be executed at system instantiation before its full phase functioning begins. Suppose, this SOC will perform the parallel execution of those initial set of tasks by appropriately assigning them on individual PEs, such that the completion time of all tasks are minimized and the system can quickly start functioning, after its instantiation. Therefore, it is essential to have well defined tasks allocation strategy, feasibility criteria for achieving high resource utilization with performance efficiency.

In this paper, we propose two novel static tasks allocation strategies for FPGA based heterogeneous SOCs. The first strategy is optimal "Branch and Bound" [5] based task allocation technique and the second one, is based on heuristic approach, coined as *TAMF* allocation strategy. The contributions of this work can be summarized as follows:

- ILP based problem formulation for tasks assignments on FPGA based heterogeneous SOCs.
- Employing the optimal "Branch and Bound" strategy for tasks assignments.
- Proposal of a new *TAMF* heuristic for faster tasks allotments.
- The experimental results reveal the efficacy of the proposed approaches in various simulation scenarios and also showed that the performance of *TAMF* is comparable to optimal Branch and Bound technique in most of the cases.
- Validation of the proposed tasks allocation strategies on actual FPGA based heterogeneous SOC named ZYNQ [3], using benchmark task sets.

The rest of the paper is organized as follows. The next section provides a brief discussion on important related works conducted. This is followed by a discussion on the system model adopted in this work in Sect. 3. In Sect. 4, we present the formulation of the tasks allocation problem. The proposed Branch and Bound based strategy and TAMF with illustrating examples, are discussed in Sects. 5 and 6, respectively. Section 7 presents simulation based experimental results along with analysis and discussion on the same. The validation of the approaches on ZYNQ platform is discussed in Sect. 8. The paper finally concludes in Sect. 9.

2 Related Work

The generic problem of efficient task allocation on heterogeneous multiprocessor systems (with CPUs and GPUs or Processors having different instruction sets) have drawn considerable research interest in last few years [6,7]. One stream of researchers [8,9] have delved towards finding out the efficient allocation schemes for *dependent* task sets. On the other hand, a plethora of research works [10,11] are available which dealt with *independent* tasks sets. Authors in [10], present an Integer Linear Programming (ILP) based tasks allotment scheme which guarantees the execution speedup upto a certain bound. In [11], authors proposed two tasks assignment algorithms and discussed about the performance between migrative allocation approach which allows tasks migration among different types of processors and the non-migrative approach. Further, the problem of tasks handling in heterogeneous multiprocessor systems has spun-off in another direction where tasks could be *static* or *dynamic* in nature. In [12], authors considered static allocation where the tasks remain quantitatively constant through out the schedule. Similarly in [13], authors proposed a dynamic allocation technique for heterogeneous multiprocessor system where tasks arrive in arbitrary instances.

Minimization of the *makespan*[1] time remains one of the principal research focus for researchers during allocation of tasks on heterogeneous multiprocessors. Thus, the researchers reported both fast heuristic based algorithms as well as optimal algorithms. Heuristics based independent task allocation strategies such as greedy approach, ant colony optimization techniques can be found in [14]. In [15], authors proposed "Branch and Bound" based optimization strategy to achieve optimal solution. However, literatures about tasks allocation on FPGA based heterogeneous SOCs with proper optimization technique is merely a handful. The existing research works [16,17] mainly employed hardware and software tasks partitioning as a tasks allocation strategy. The computation intensive tasks are being executed using FPGA fabric as a hardware task whereas tasks having less computation requirements are assigned to CPUs for execution as a software task. However, such allotments are often plagued with some limitations.

3 System Model and Assumptions

The system model, considered in this work is a FPGA based heterogeneous SOC. Our heterogeneous SOC contains an FPGA logic area and pre-fabricated Hard Cores (HCs). Further, the FPGA logic consists of Reconfigurable Regions or tiles (RRs) to execute hardware tasks and Soft Cores (SCs) (which constructed using FPGA logic) for software tasks execution. Hence, the system model under consideration contains three distinct types of PEs that is RR, SC, HC which completely resembles with modern ZYNQ [3] architecture. Depending upon the SOC architecture, a particular type of PE could exist in multiple units.

Being an embedded system, the initial distributions of tasks are known at design time and each task is capable of running on each type of PEs. The proposed system model is further characterized by the following assumptions:

- A task cannot execute simultaneously on two distinct PEs at same instant and will be strictly executed on a single PE (till it execution requirements fulfills), which implies that tasks are non-migrative in nature.
- Tasks are independent and thus, each PEs can operate in parallel to execute an individual task.
- The cost of execution of each task over each distinct type of PEs is calculated through profiling and stored in offline.
- PEs are capable to suffice the memory and I/O requirements of any task.

At system instantiation (time, $t = 0$), a fixed number of tasks will arrive for possible allotment on SOC and will be stored in a queue. Our strategy will attempt to allocate tasks, such that the makespan time gets minimized. Algorithms will operate through a dedicated HC (termed as *allocator*) and will allocate tasks on

[1] It is the total length of the schedule i.e. when all the tasks have finished their execution.

respective PEs, as per the outcome. The pictorial representation of the proposed system model is shown in Fig. 1. A practical validation of the proposed system model on physical ZYNQ platform is illustrated in Sect. 8.

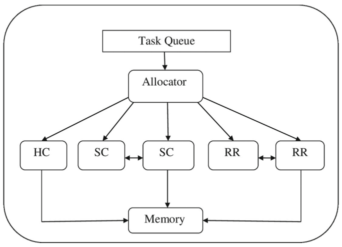


Fig. 1. System model

4 Problem Formulation

Let us assume a task set $\tau = \{T_1, T_2, ..., T_N\}$, arrives for possible allotment over $\mathcal{M}$ distinct types of PEs where each j^{th} type of PE is comprised of k^j units. Such multiple units of a particular type of PE is individually referred as *"core"*. It may be noted that the cost of execution of a task, will be same for all the cores which belong to a particular type of PE.

At a particular instant t, we can allocate atmost $(\sum_{j=1}^{\mathcal{M}} k^j)$ tasks for parallel execution over all resources. Let, $C_{i,j}$ denotes the cost of execution incurred by T_i when assigned to j^{th} type of PE. Let us define a binary variable t_{ik^j} as follows:

$$t_{ik^j} = \begin{Bmatrix} 1, \text{When } T_i \text{ is assigned to any one of the } k^j \text{ cores of } j^{th} \text{ type PE} \\ 0, \qquad\qquad\qquad\qquad \text{otherwise} \end{Bmatrix}$$

Hence, we can illustrate the makespan time mk as:

$$mk = Max\{\sum_{i=1}^{\eta} \sum_{i'=1}^{k^j} C_{ij} * t_{ik^j}\}, \quad \forall j \in \mathcal{M} \tag{1}$$

where, η denotes the number of tasks assigned to j^{th} type of PE.

In order to achieve lower mk, the objective function can be defined as:

$$\text{Minimize} \quad mk \tag{2}$$

Subject to the following constraints:

- The maximum number of tasks that can operate in parallel at a particular instant, can be atmost equal to the total number of available resources.
- No task can simultaneously execute on different resources. Thus, at a particular instant, following equation has to be satisfied.

$$\sum_{j=1}^{M} \sum_{i'=1}^{k^j} t_{ik^j} \leq \sum_{j=1}^{M} k^j \tag{3}$$

5 Branch and Bound Based Allocation Strategy

The above problem formulation clearly depicts that the proposed task allotment phenomena turns into an ILP based optimization problem and hence, NP-complete in nature [18]. Once a problem enrolls into the NP-complete category, it is very unlikely that an algorithm with polynomial time complexity can be designed in order to solve that one. The optimal Branch and Bound [5], implicitly enumerates all of the feasible solutions by forming and traversing a *"state space tree."*[2] The tree is constructed using *DFS* [19] algorithm. At each node of the *state space tree*, the *Lower Bound (LB)* has to be calculated such that, search effort for probing the solution space can be reduced.

The cost incurred by the allocated tasks and the probable (underestimate) cost that will be incurred by the remaining unallocated tasks, will reflect as a sum in *LB*. Similarly, *Upper Bound (UB)*, is calculated by considering the worst case cost (instead of minimum cost) of the unallocated tasks.

At a particular node (N_T), if the calculated *LB* appears to be larger or equal to the best solution result found so far, then the enumeration of the subtree rooted at N_T can be skipped. It definitely implies that some nodes and subtrees will be *"pruned-off"* but it will obviously not jeopardize the optimality.

The pseudo-code for the proposed Branch and Bound strategy is shown in Algorithm 1. In the following section, we will illustrate an example which will depicts our proposed Branch and Bound allocation strategy through a test case.

Algorithm 1. Branch and Bound Based Allocation Strategy

```
   Input: Task set $\tau$, $\mathcal{M}$ types of PEs (including $k^j$ units)
   Output: Allocation of tasks which provide minimum $mk$.
1  For each PE, calculate the minimum task execution costs;
2  Sum up those minimum execution costs and assign to Start_Value;
3  while $\tau$ is not empty do
4      for each task $T_i \in \tau$ do
5          Find the LB for each type of PEs;
6      Find task $T_\beta$ which provides min-LB (minimum LB) for say, $j^{th}$ PE;
7      {In case of multiple existence of min-LB, choose any arbitary $T_\beta$};
8      Calculate UB of $T_\beta$ for that $j^{th}$ PE;
9      Allocated_Value = Summation of the costs of already allocated task/tasks;
10     if (min-LB + Allocated_Value) $\geq$ Start_Value AND min-LB $<$ UB then
11         Assign $T_\beta$ to $j^{th}$ PE;
12         Remove $T_\beta$ from the $\tau$;
13     else
14         Choose the next min-LB;
15         Go to step 10;
16 Find mk using Eq. 1.
```

[2] A tree constructed in the solution space.

5.1 An Example of Branch and Bound (BnB) Based Strategy

Let us consider six tasks, $\tau = \{T_1, T_2, ..., T_6\}$ that appear for the possible allotment (such that *makespan* time get minimized) in our proposed FPGA based heterogeneous SOC. Let us assume that system under consideration contains, one unit of RR ($k^1 = 1$), two units of SC (hence, $k^2 = 2$; $core_1$ and $core_2$ of SC) and two cores of HC ($k^3 = 2$), one of which is dedicatedly acting as *allocator*. The performance of each task over these three ($\mathcal{M} = 3$) distinct types of PEs is measured through profiling and displayed in Table 1.[3] It may be noted that the execution performance for both the cores of SC will be same and hence, not explicitly shown in the Table 1.

Table 1. Tasks parameters

PEs	Tasks					
	T_1	T_2	T_3	T_4	T_5	T_6
HC	4	5	1	14	5	5
SC	3	7	3	11	4	3
RR	7	9	5	9	9	5

Table 2. Tasks LBs

PEs	Tasks					
	T_1	T_2	T_3	T_4	T_5	T_6
HC	15	16	12	25	16	14
SC	12	16	15	20	13	**12**
RR	14	16	15	16	16	12

The minimum execution cost demanded by HC is one time-unit for executing T_3. Similarly, SC demands minimum three time-units for both the cores and RR demands minimum five time-units while executing the tasks T_6 and T_3, respectively. Hence, $Start\_Value = 1 + 3 + 3 + 5 = 12$.

A particular LB value can be calculated as follows: let us consider, T_6 is allocated in SC and thus, the incurred cost of T_6 will be **3** time-units. This allocation implies that, SC ($core_1$) and T_6 will be out of consideration for the next course of calculation. The minimum cost that will be incurred by *remaining unallocated tasks* can be found as: "minimum execution cost demand by HC": 1 + "minimum execution cost demand by SC ($core_2$)": 3 + "minimum execution cost demand by RR": 5. Hence, the corresponding LB becomes: $\mathbf{3} + (1 + 3 + 5) = 12$. Similarly, by considering each individual allotment possibilities, LBs for each task corresponding to each type of processing resource is calculated and shown in tabular form in the Table 2.

Following the steps 6–8 of Algorithm 1, *min-LB* can be found as 12 of T_6 for SC ($core_1$). Hence, UB of T_6 for SC can be calculated as follows: "maximum execution cost for HC": (**14**) + "maximum execution cost for SC ($core_2$)": (**11**)+ "maximum execution cost for RR": (**9**) + cost of T_6 for SC ($core_1$) (**3**). Hence, UB of task T_6 for SC becomes: 37. Before the task allocation procedure begins, *Allocated_Value* is initialized as zero. Thus, it may be observed that T_6

[3] The values corresponding to tasks are depicting the execution cost in terms of time-units.

is satisfying the conditions stated in step 10 of the Algorithm 1. Hence, T_6 successfully assigned to SC ($core_1$) and removed from the task set τ. In the same way, following the steps of Algorithm 1, the allocation procedure will continue until τ becomes empty. The final task allocation scenario and corresponding mk is shown in Table 3.

6 TAMF-Working Principle

It is worth mentioning that Branch and Bound strategy demands higher degree of computations and may become computationally intensive when the number of task increases. Hence, we propose a novel heuristic based tasks allocation strategy called TAMF. TAMF will be executed once at system instantiation and tasks will be allocated as per the achieved task-to-PE mapping information.

At the beginning of TAMF, the task set τ gets ready with sorted execution cost of each task (in non-decreasing order) for different types of PEs. We assume that each type of PE provides minimum execution cost for atleast one task. Hence, TAMF will choose the element of the sorted sequence for allocation. It ensures that the task will be allocated to that particular PE for which it has the minimum execution cost. Now as soon as core/cores of a PE, finishes its pre-assigned execution, it will look for the next most eligible candidate task T_α. T_α can only be allocated to a "free"[4] PE, if and only if the following two conditions hold alternatively:

i. The execution cost of the T_α is minimum for the free PE (among all PEs).
ii. The execution cost of the T_α for the free PE $\leq$ average execution cost of T_α.

Here, both the conditions attempt to ensure that T_α should completes with its lower execution requirement and thus, maximizing the probability of having minimum mk. Moreover, the allocation of T_α to any other PE might not be able to provide better mk. TAMF will continue its own operation till any unallocated task remains. The pseudo-code for TAMF is depicted in Algorithm 2.

Table 3. Tasks allotment and mk: BnB Based

PEs	Allocated task(s)	Consumed cost	mk
HC	T_2, T_3	$5+1=6$	9
SC	$Core_1 <= T_1, T_5$	$3+4=7$	
	$Core_2 <= T_6$	3	
RR	T_4	9	

Table 4. Tasks allotment by TAMF

PEs	Time								
	1	2	3	4	5	6	7	8	9
HC	T_2					T_3			
SC ($core_1$)	T_1			T_6					
SC ($core_2$)	T_5								
RR									T_4

[4] The PE finished its earlier execution and currently not executing any task.

Algorithm 2. TAMF

```
Input: Task set $\tau$, $\mathcal{M}$ types of PEs
Output: Allocation of tasks for achiveing minimum $mk$.
1 {$T_\alpha$: next candidate task; $C_{\alpha j}$: execution cost of $T_\alpha$ in $j^{th}$ PE; $C_{\alpha j}^{min} = \min\{C_{\alpha j}\}, \forall j \in \mathcal{M}$: minimum execution cost of $T_\alpha$ in $j^{th}$ PE; $C_{\alpha}^{avg} = \sum_{j=1}^{M} C_{\alpha j}/\mathcal{M}$: average execution cost of $T_\alpha$ over all PEs};
2 while $\tau$ is not empty do
3     for each $j^{th}$ free PE, $j \in \mathcal{M}$ do
4         if $C_{\alpha j} == C_{\alpha j}^{min}$ OR $C_{\alpha j} \leq C_{\alpha}^{avg}$ then
5             Assign $T_\alpha$ to the $j^{th}$ PE;
6             Remove $T_\alpha$ from $\tau$;
7             $j^{th}$ PE become free when $C_{\alpha j}$ completes.;
8 Find $mk$ using Eq. 1;
```

6.1 TAMF in Work: An Example

Let us consider, the same set of tasks τ as shown in Table 1. As stated earlier, in this case τ contains all the parameters in a sorted fashion. Let us consider T_1 as a first candidate for allocation. It may be noted that the execution cost of T_1, related to each PEs is already being stored in a sorted sequence. T_1 consumes minimum execution cost 3 time units while executes in SC. On the other hand, the average execution cost of T_1 over all PEs becomes: $\{(3+4+7)/3\}$. It may be observed that T_1 satisfies the condition, stated in step 4 of the algorithm 2. Hence, T_1 is allocated on SC (arbitrarily, $core_1$) and removed from τ.

By following the same way, T_2 gets allocated in HC but T_3 could get allocated in HC when T_2 finishes. Similarly, other tasks will be allocated by the following algorithm 2. The respective allocation and execution sequence of each task is shown in Table 4. It is very much evident that all the tasks complete their execution requirements within 9 time units and thus, $mk = 9$.

7 Experiments and Results

The performance of the Branch and Bound based strategy and TAMF have been evaluated by conducting simulation based experiments using randomly generated task sets whose execution cost corresponding to a PE have been taken from normal distribution. The performance metrics used for evaluation are *makespan* time as defined in Eq. 1 and the Computational Overheads, CO (measured in terms of consumed clock cycles/CPU ticks). Data sets for various values of N (10 to 30)[5], have been generated on systems containing total 2 to 8 cores of three types of PEs. Each result is generated by executing 50 different instances of each data set type and then taking the average over these 50 runs.

[5] Routine tasks within an embedded system typically lies within this range [20].

Table 5 shows the performance variation of Branch and Bound (BnB) based strategy and TAMF over different number of processing resources. It may be observed from the third and fourth column of the table, that BnB based technique can effectively eliminate the possible numbers of solution (by traversing less number of nodes) through an intelligent pruning technique. Moreover, as N increases, the computation overhead related to the LB estimation also increases (lines 4–5, Algorithm 1) and thus, the CO_{BnB} of overall strategy increases. Another notable observation reveals that when the number of processing resources increases by keeping N fixed, the number of traversed node (AF prun) as well as CO_{BnB} also increases, this is mainly because the additional overheads incurred in LB calculation for the increment in processing resources. This observations is also supported by Fig. 2(a).

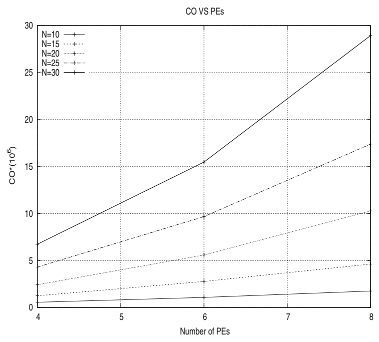
Table 5 also establishes the fact that TAMF demands less computational overhead than Branch and Bound based strategy. This observation may be attributed to the fact that being a heuristic based policy, TAMF does not implicitly enumerate all possible solutions instead, employs task selection mechanism with linear overhead and thus, costs less CO_{TAMF}. Moreover, like BnB based strategy, increase in N also endorses increment in TAMF's overheads. However, if BnB based strategy and TAMF runs on an *allocator* whose frequency is 1.5 GHz then CO_{BnB} and CO_{TAMF} for $N = 10$ (with six processing resources) becomes 0.72 and 0.16 in milliseconds, respectively. Figure 2(b), depicts bar chart showing *makespan* time (mk) produced by Branch and Bound based strategy and TAMF for number of tasks (N) varying from 10 to 30 with constant number of processing resources as 8. From the respective figure, it is very much evident that in most

Table 5. Performance of Branch and Bound (BnB) based strategy and TAMF

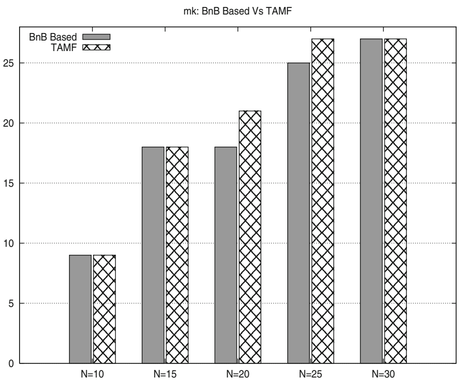
PEs[a]	N	BF Prun	AF Prun	CO_{BnB}	CO_{TAMF}
HC - 2	10	$> 100^2$	330	1080551	243112
	15	$> 100^2$	720	2775883	384098
SC - 2	20	$> 100^2$	1260	5490998	535518
	25	$> 100^2$	1950	9675934	679567
RR - 2	30	$> 100^2$	2790	15481722	832551
HC - 3	10	$> 100^2$	440	1755676	252107
	15	$> 100^2$	960	4621855	391578
SC - 3	20	$> 100^2$	1680	10279888	552077
	25	$> 100^2$	2600	17389423	724583
RR - 2	30	$> 100^2$	3720	28938997	877661

N: Number of tasks, **BF Prun**; **AF Prun**: Numbers of expected and actual nodes traveled respectively, $\boldsymbol{CO_{BnB}}$: Computational overheads in terms of Clock ticks for Branch and Bound based strategy, $\boldsymbol{CO_{TAMF}}$: Computational Overheads for TAMF

[a] PEs with individual number of cores



(a) CO vs PEs: BnB Based Strategy



(b) mk vs N: BnB Based and TAMF

Fig. 2. Performance of BnB based strategy and TAMF

of the cases, the efficiency of Branch and Bound based strategy and TAMF (in terms of mk) is comparable. Through a deeper observation, it can be observed that for $N = 20$ tasks, BnB based strategy provides $mk = 18$ (slightly lower) while TAMF produces $mk = 21$. However, TAMF will cost CO_{TAMF}=552112 clock cycle which is 18.6% lower than the BnB based strategy (refer, Table 5).

8 Validation in Physical ZYNQ Platform

Besides of the thorough theoretical and simulation studies, we have also validated the proposed strategies in actual FPGA based heterogeneous SOC (ZYNQ:ZC702) with synthetic task sets. The platform contains three distinct types of PEs that is the FPGA chip, dual core ARM Cortex-A9 processor and MicroBlaze (MB) processor. In the ZYNQ board, these ARMs are located in PS (Processing System) region and FPGA logics are separate as PL (Programmable logic) region [3]. In PL, the Reconfigurable Regions (RRs) are done to carry out the hardware task execution and Soft Core (SC) MB executes software tasks. In PS, ARMs are pre-fabricated and hence, termed as Hard Core (HC). It may be observed that the ZYNQ architecture resembles our adopted system model. Xilinx PlanAhead 14.4 and XPS, EDK, SDK 14.4 [3] tools are have been used for this validations.

8.1 Customization of the Platform

For the purpose of the meaningful validation, the ZYNQ platform needs to be architecturally customized.

- In the PL, RR was properly marked and the execution of each task on that RR is ensured by properly maintaining the UCF [3] constraints.

- Two MBs are added in the PL using XPS IP-core. RR and MB are connected through inter PL bus architecture.
- The RR, MB both are operated with PLs clock (FCCLK) of 50 MHz frequency.
- One ARM core (*core*-0) is kept for possible task allotment and another ARM (*core*-1) works as *allocator*. These ARMs are operating using 667 MHz frequency.
- Both the PS and PL are communicating through GP0 and GP1 port. PS and PL are also connected with OCM (On-Chip Memory) using AXI bus.

8.2 Synthetic Task Set Creation

To cope-up with the customized platform, it is essential to have proper set of tasks. In our validation, we have constructed and profiled the sets of task. These task sets are taken from well known Benchmark task set named ITC'99 benchmarks [21]. This Benchmark consists of numerous task such as *"Adder"*, *"Decoder"*, *"Integer to Float conversion"* etc.

- Synthetic hardware tasks are created using VHDL code and performance of each such tasks is measured forming proper test-bench.
- Software tasks are written in system-c code. The execution performance of those tasks are measured for both MB and ARM. The execution cost of sample tasks is shown in Table 6.
- At design time, each hardware task is stored in its executable format (as *.bit*) in external memory. In ZYNQ, such *.bit* files are stored in *SD* card.
- Software tasks are stored in executable (*.elf*) format in the external *SD* card.
- This external memory is present in PS region and linked with the *allocator*.

8.3 Implementation and Outcomes

Both the strategies (Branch and Bound based and TAMF) are coded in system-c, compiled and stored (as respective executable format) in the *SD* card. At the system instantiation, ZYNQ starts booting from *SD* card and *core*-1 of ARM (*allocator*), initiates the execution of the respective strategy. After completion, the tasks allocation information are stored in a log file, in the external memory. Now *core*-1, will start allocation by transferring the respective *.bit* file to the PL (if a task is assigned to PL region) and *.elf* to MB, *core*-0 of ARM respectively. In the similar way, the *allocator* further reads the task assignment information from the stored log file and allocates tasks (in their respective executable format) from external memory to appropriate PEs.

Table 7 shows the performance of BnB based allocation strategy and TAMF over different number of processing resources, on the ZYNQ platform. From the table, it can be concluded that the trends of results in the actual platform concur with the outcomes obtained through simulation studies.

Table 6. Benchmark tasks execution overhead (clock ticks unit) on ZYNQ

PEs	Dec	Add	I2F	Lg2
HC	460158	455920	464819	451817
RR	750	35112	70089	465335
SC	41312	42736	87733	581298

Table 7. Performance on ZYNQ

PEs	N	BF Prun	AF Prun	CO_{BnB}	CO_{TAMF}
HC - 1	10	$> 100^2$	330	1268233	483468
SC - 2	-	-	-	-	-
RR - 2	20	$> 100^2$	1260	1988597	791436

9 Conclusion

In this paper, we presented methodologies for allocating task sets on FPGA based heterogeneous SOC such that *makespan* is minimized. An optimal strategy and heuristic based technique is discussed. We designed, implemented and evaluated the algorithms using simulation based experiments and the simulation results were further validated through real implementation on ZYNQ platform.

Acknowledgments. This work was supported in part by the TCS Research Fellowship Award, granted to Sangeet Saha and TEQIP Phase-II project of University of Calcutta, India.

References

1. Hayashi, T., Kojima, A., Miyazaki, T., Oda, N., Wakita, K., Furusawa, T.: Application of FPGA to nuclear power plant I&C systems. In: Yoshikawa, H., Zhang, Z. (eds.) Progress of Nuclear Safety for Symbiosis and Sustainability, pp. 41–47. Springer, Tokyo (2014). doi:10.1007/978-4-431-54610-8_5
2. Jin, J., Lee, S., Jeon, B., Nguyen, T.T., Jeon, J.W.: Real-time multiple object centroid tracking for gesture recognition based on FPGA. In: Proceedings of the 7th International Conference on Ubiquitous Information, Management and Communication, p. 80. ACM (2013)
3. Crockett, L.H., Elliot, R.A., Enderwitz, M.A., Stewart, R.W.: The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc. Strathclyde Academic Media (2014)
4. Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K., Lange, K.: Autosar-a worldwide standard is on the road. In: International VDI Congress Electronic Systems for Vehicles, vol. 62 (2009)
5. Corrêa, R., Ferreira, A.: Branch and bound. Parallel Algorithms for Irregular Problems: State of the Art, pp. 157–176 (2013)
6. Moreira, O., Valente, F., Bekooij, M.: Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In: Proceedings of the 7th ACM & IEEE International Conference on Embedded Software, pp. 57–66. ACM (2007)
7. Satish, N.R., Ravindran, K., Keutzer, K.: Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors. In: Proceedings of the 8th ACM International Conference on Embedded Software, pp. 149–158. ACM (2008)

8. Dhingra, S., Gupta, S.B., Biswas, R.: Hybrid gasa for bi-criteria multiprocessor task scheduling with precedence constraints. Comput. Appl. Int. J. **1**(1), 11–21 (2014)
9. Biswas, S.K., Rauniyar, A., Muhuri, P.K.: Multi-objective bayesian optimization algorithm for real-time task scheduling on heterogeneous multiprocessors. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 2844–2851. IEEE (2016)
10. Baruah, S.K., Bonifaci, V., Bruni, R., Marchetti-Spaccamela, A.: ILP-based approaches to partitioning recurrent workloads upon heterogeneous multiprocessors. In: ECRTS, pp. 215–225 (2016)
11. Raravi, G., Andersson, B., Nélis, V., Bletsas, K.: Task assignment algorithms for two-type heterogeneous multiprocessors. Real-Time Syst. **50**(1), 87–141 (2014)
12. Kofler, K., Grasso, I., Cosenza, B., Fahringer, T.: An automatic input-sensitive approach for heterogeneous task partitioning. In: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing, pp. 149–160 (2013)
13. Tabatabaee, H., Akbarzadeh-T, M.R., Pariz, N.: Dynamic task scheduling modeling in unstructured heterogeneous multiprocessor systems. J. Zhejiang Univ. Sci. C **15**(6), 423–434 (2014)
14. Luo, P., Lü, K., Shi, Z.: A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. J. Parallel Distrib. Comput. **67**(6), 695–714 (2007)
15. Chow, K.W., Liu, B.: On mapping signal processing algorithms to a heterogeneous multiprocessor system. In: 1991 International Conference on Acoustics, Speech, and Signal Processing, ICASSP 1991, pp. 1585–1588. IEEE (1991)
16. Pagani, M., Marinoni, M., Biondi, A., Balsini, A., Buttazzo, G.: Towards real-time operating systems for heterogeneous reconfigurable platforms. In: OSPERT 2016, pp. 49–54 (2016)
17. Li, L., Sun, J., Li, W., Lv, Z., Guan, F.: Hardware/software partitioning based on hybrid genetic and tabu search in the dynamically reconfigurable system. Int. J. Control Autom. **8**(1), 29–36 (2015)
18. Papadimitriou, C.H.: Computational Complexity. Wiley, Chichester (2003)
19. Everitt, T., Hutter, M.: Analytical results on the BFS vs. DFS algorithm selection problem. Part I: tree search. In: Pfahringer, B., Renz, J. (eds.) AI 2015. LNCS, vol. 9457, pp. 157–165. Springer, Cham (2015). doi:10.1007/978-3-319-26350-2_14
20. Chattopadhyay, S.: Embedded System Design. PHI Learning Pvt. Ltd., Delhi (2013)
21. Davidson, S.: Itc'99 benchmark circuits-preliminary results. In: Proceedings of International Test Conference 1999, pp. 1125–1125. IEEE (1999)