# An Exact Optimization and a Heuristic for Resource Management with Prediction

## I. INTRODUCTION

In modern heterogeneous architectures, multiple computational resources are brought together in order to provide high performance for different applications. Integrating multiple heterogeneous cores in a single architecture is an important technique for obtaining performance benefits; however, this can only be attained if the platform based on this architecture is equipped with an appropriate resource manager (RM) for making decisions such as task mapping, task scheduling, and voltage and frequency scaling. Issues are made more complicated since, most of the times, the platforms are exposed to a fluctuating workload not know at design time. In this context, considering a prediction of the future workload, in addition to the current state of the platform, should potentially improve the efficiency of the resource management decisions.

The goal of this paper is presenting an exact optimization and a heuristic for resource allocation when the resource manager (RM) is able to predict the incoming request. The rest of the paper is organized as follows. The system model is given in Sec. II. In Sec. III, the notations and conventions that are used in the following sections are presented. Then, the problem of exact optimization of task mapping and scheduling is addressed in Sec. IV-A. Finally, a heuristic solution to the problem will be introduced in Sec. IV-B.

## II. SYSTEM MODEL

We consider a heterogeneous platform consisting of $N$ computation resources $r_i$ (CPUs+GPUs); ($i = 1, \ldots, N$). The platform executes a fluctuating workload as the response to a stream of requests. In order to focus on the main point of interest which is prediction, we consider a relatively simple workload model. Each request $req_j$ triggers a specific activity denoted as a task $\tau_j$. Each task $\tau_j$, ($j = 1, \ldots, L$) is characterized by:

- arrival time $s_j$, the time of the arrival of request $req_j$;
- deadline $d_j$, relative to the arrival time;
- average energy consumption $e_{j,i}$, when $\tau_j$ is executed on resource $r_i$, for all $r_i$;
- worst case execution time (WCET) $c_{j,i}$ when $\tau_j$ is executed on resource $r_i$, for all $r_i$;
- energy and time overhead $em_{j,k,i}$ and $cm_{j,k,i}$ due to migration of task $\tau_j$ from resource $r_k$ to resource $r_i$, for each $r_i$ and $r_k$.

Following each request $req_j$, at time $s_j$, the RM has to decide the resource to which to map the corresponding task $\tau_j$ and the time moment at which to schedule the start of its execution. We assume that the tasks are firm real-time, which means that they have to meet their deadlines in order for their output to be of any use. If they miss their deadlines their result would be useless. Tasks are preemptable, except when executed on particular resources, in which case they cannot be preempted and continued afterwards, but need to run to the end in order to produce results.

At each task arrival the RM considers the current context of active tasks under execution and the new task $\tau_j$ to be activated as response to the request $req_j$. It tries to find a mapping and schedule for task $\tau_j$ such that it satisfies its deadline. To this end, it might preempt running tasks, remap, and reschedule them - taking the involved migration overheads into consideration (with

exception of e.g. GPUs). If there is no solution such that all tasks meet their deadlines, $\tau_j$ will not be admitted. If prediction is employed, in addition to the current context and the arriving task $\tau_j$, the RM also considers the task $\tau_p$ corresponding to the predicted request $req_p$ and its predicted arrival time $s_p$, when deciding on mapping and scheduling of $\tau_j$. If the RM cannot find a solution, it ignores $\tau_p$. The RM takes its decisions such that energy consumption is minimized.

## III. PREREQUISITES

In what follows, we will use the following conventions:

- For any activation of the RM, at a certain time $t$, we denote with $\overline{S}$ the set of all tasks that have been admitted before the time $t$ and are not yet finished plus the task arrived as result of the current request and (if with prediction) the task corresponding to the predicted request.

- When the resource manager is activated at a certain time $t$, let us consider $\tau_j$ as one of the tasks currently running on resource $r_i$. We remind that $c_{j,i}$ is the WCET of the task. We denote by $cp_{j,i}$ the (worst case) run time not yet consumed for $\tau_j$ on $r_i$ at current time $t$ (if the task is not yet started $cp_{j,i} = c_{j,i}$). If the RM decides to migrate $\tau_j$ to another resource $r_k$ then the execution time not yet executed is $cp_{j,k} = c_{j,k} \times (cp_{j,i}/c_{j,i})$.

- The time window considered by the RM at each activation at time $t$ is the interval between the current time $t$ and the time moment defined by the latest deadline of all tasks in the set $\overline{S}$. We denote the length of this time window by $\overline{K} = \max\limits_{\tau_j \in \overline{S}}(t_{\text{left}_j})$, where $t_{\text{left}_j} = s_j + d_j - t$; here $s_j + d_j$ is the absolute deadline of task. Inside this time window $\overline{K}$ the RM will decide on the mapping and scheduling of all tasks in $\overline{S}$. On each resource the scheduling is performed according to the optimal earliest deadline first (EDF) policy. If no prediction is used there is no preemption between two activations of the RM. Thus, the RM will order the tasks on each resource according to their deadline. Here, by task we mean complete tasks (if they have not been started yet) or the pieces of tasks remained to be executed for tasks under execution at time $t$. If prediction is used and the predicted task has a deadline earlier than another task in the set $\overline{S}$, then the schedule produced by RM is considering the preemption caused by the predicted task. Such preemption is not applied to a GPU. Let us mention that the mapping and scheduling of the predicted task are only used as a constraint in order to find an efficient mapping and schedule for the current task, that takes the future arrival into consideration. The actual predicted task will be effectively mapped and scheduled when and if it actually arrives.

- We use Mixed Integer Linear Programming (MILP) as our exact optimization method. A method called big-M [1] is frequently utilized in Sec. IV-A, which has many applications; M is a sufficiently large positive number. One of the various applications of Big-M is to assuring equality of variables only when a certain binary variable takes on one value, but to leave the variables "open" if the binary variable takes on its opposite value. The other application is that it is required to have if-then decisions among constraints in some cases which is solved also by utilizing this method.

- In MILP one can have only linear constraints. The product of two binary variables or product a binary variable and a real variable can be linearized based on the technique shown in [1].

- The notation $t$ in the equations denotes the current time $t$ at which the RM is activated.

## IV. RESOURCE MANAGEMENT WITH PREDICTION

### A. MILP formulation for exact optimization

The formulation of MILP is as follows:

$$\text{minimize} \quad \sum_{j|\tau_j \in \overline{S}} \sum_{i=1}^{N} x_{j,i} \times (e_{j,i} + em_{j,k,i})$$

$$\text{subject to:} \forall \tau_j \in \overline{S} : \quad \sum_{i=1}^{N} x_{j,i} = 1 \tag{1}$$

$$\forall \tau_j \in \overline{S} : \quad \sum_{i=1}^{N} x_{j,i} \times cpm_{j,i} \le t_{\text{left}_j} \tag{2}$$

The mapping variables are denoted by $x_{ji}$ where $x_{ji} = 1$ if task $\tau_j$ is mapped to resource $i$; otherwise $x_{ji} = 0$. We denote by $cpm_{j,i}$ the total execution time of $\tau_j$ including the migration cost of the case that the task is relocated during the current time window; $cpm_{j,i} = cp_{j,i}$ if the task is not relocated and $cpm_{j,i} = cp_{j,i} + cm_{j,k,i}$ if $\tau_j$ is migrated from $r_k$ to $r_i$. If $\tau_j$ is migrated from $r_k$ to $r_i$ during the current time window, $em_{j,k,i}$ is the energy overhead for the migration; otherwise it is zero. The constraints in (1) enforce that each task is mapped to one and only one resource. The constraints in (2) ensure that, if $\tau_j$ is mapped to $r_i$, its execution time on this resources is not longer than $t_{\text{left}_j}$; otherwise, its deadline cannot be met. The scheduling constraints (3) ensure that all tasks mapped to resource $r_i$ meat their deadline. This constraint applies to all resources $r_i$, except the resource to which the predicted task $\tau_p$ is mapped. SL is the list of tasks in $\overline{S}$ sorted by their deadline. The summation is over the index $k$ in the sorted list. We remind (see Sec. III) that, according to EDF, the RM sorts the tasks mapped to each resource according to their deadline. The constraints impose that each task finishes before its deadline.

$$\forall \tau_j \in SL : \quad \sum_{k=1}^{j} x_{k,i} \times cpm_{k,i} \le -t_{\text{left}_j} \times (-M \times x_{p,i} - 1) \tag{3}$$

If $\tau_p$ is mapped on $r_i$ ($x_{p,i} = 1$), constraints (3) cannot ensure schedulability. If the deadline of $\tau_p$ is later than that of all other tasks in $\overline{S}$, there will be no preemption. It is apparent that the predicted task cannot start its execution earlier than its arrival. The task will be scheduled at the time $max(s_p, f_i)$, where $s_p$ is the arrival time of $\tau_p$ and $f_i$ is the time moment when all tasks mapped to $r_i$ (except $\tau_p$) finish their execution. Let us define $g_i$ as the gap between the arrival of $\tau_p$ and the time when $r_i$ becomes idle. In the case that $s_p \le f_i$, $gap_i \le 0$ and $\tau_p$ can start its execution immediately after $f_i$; otherwise, $\tau_p$ cannot start its execution immediately after $f_i$, and we should wait until its arrival ($gap_i > 0$). We define a binary variable $z_i$, which will be enforced to 1 by utilizing the big-M method to implement the following if-then decision: if $g_i > 0$ then $z_i = 1$; otherwise, $z_i = 0$. If there is no preemption, the constraints in (4) and (5) guarantee schedulability.

$$f_i - t + x_{p,i} \times cp_{p,i} \le -t_{\text{left}_p} \times (-M \times z_i - 1) \tag{4}$$

$$s_{p,i} - t + x_{p,i} \times cp_{p,i} \le -t_{\text{left}_p} \times ((-M \times (1 - z_i)) - 1) \tag{5}$$

If the deadline of the predicted task $\tau_p$ is earlier than that of some tasks in the set $\overline{S}$ then one of the tasks will be preempted. We divide the ordered list of tasks SL into two sublists: SL1 consists of those tasks whose deadline is earlier than the one of $\tau_p$ or equal. SL2 is the list of tasks with deadlines later than $\tau_p$. The tasks in SL1 will not be preempted by $\tau_p$ and constraints (6) ensure their schedulability.

$$\forall \tau_j \in SL1 : \quad \sum_{k=1}^{j} x_{k,i} \times cpm_{k,i} \le -t_{\text{left}_j} \times (-M \times (1 - x_{p,i}) - 1)) \tag{6}$$

We denote by $f_i$ the finishing time of the last task in sublist SL1 that is mapped to resource $r_i$. In the case when $s_p \le f_i$, $\tau_p$ can start its execution immediately after $f_i$, and we have that $gap_i \le 0$; otherwise, $\tau_p$ cannot start its execution immediately after $f_i$, and we should wait until its arrival ($gap_i > 0$). We define a binary variable $z_i$, which will be enforced to 1 by utilizing the big-M method to implement the following if-then decision: if $g_i > 0$ then $z_i = 1$; otherwise, $z_i = 0$. If $z_i = 0$, the schedulability constraints for the tasks inside SL2 are as follows:

$$\forall \tau_j \in SL2 : \quad f_i - t + \sum_{k=1}^{j} x_{k,i} \times cpm_{k,i} \le -t_{\text{left}_j} \times (-M \times z_i - 1) \tag{7}$$

The last case to be considered is if the predicted task $\tau_p$ arrives after the moment $f_i$ ($z_i = 1$). In this case, the RM has to plan for a preemption. Potentially any of the tasks in SL2 that are mapped to the same resource with $\tau_p$ could be preempted which one depends on the arrival time of $\tau_p$. The preempted task $\tau_j$ is divided by the preemption point into two chunks, before and after the preemption point, respectively. We denote the start time of the execution of a chunk of task $\tau_j$ mapped to resource $r_i$ by $sc_{j,i,k}$ (k=1 for the first chunk and 2 for the second) and the end time by $ec_{j,i,k}$. These start and end times are optimization variables. We define $K = \{1, 2\}$ and we have the following constraints to guarantee schedulability. In constraints (8) we ensure that the start time of the predicted task is greater than its arrival time. In constraints (9) we enforce the length of the second chunk of $\tau_p$ to zero since it cannot be preempted. Constraints (10) enforce that the end time of a chunk is after the start time of that chunk. The start of the second chunk should be after the end of the first, which is enforced by constraints (11). Constraint (12) guarantees that no deadlines are violated. The constraints (13) or (14) must be satisfied to ensure that chunks do not overlap for each $r_i$; $b_{j_1,j_2,k_1,k_2}$ is a binary value. Constraints (15) enforce that the total execution time of the two chunks is equal with the execution time of the task. If $z_i = 0$, constraints (16) and (17) are used to enforce the length of chunks to zero. In this case all constrains in (8)–(15) would be redundant. As mentioned in Sec. III, the product of variables in these constraints can be linearized based on the technique shown in [1].

$$sc_{p,i,1} \geq s_p \times z_i \times x_{p,i} \tag{8}$$

$$sc_{p,i,2} = ec_{p,i,1} \text{ and } ec_{p,i,2} = ec_{p,i,1} \tag{9}$$

$$\forall \tau_j \in SL2: \ \forall k \in K: sc_{j,i,k} - ec_{j,i,k} \leq M \times (1 - z_i \times x_{j,i}) \tag{10}$$

$$\forall \tau_j \in SL2: \ ec_{j,i,1} - sc_{j,i,2} \leq M \times (1 - z_i \times x_{j,i}) \tag{11}$$

$$\forall \tau_j \in SL2: \ ec_{j,i,2} \leq t_{\text{left}_j} \times (z_i \times x_{j,i}) \tag{12}$$

$$\forall \tau_{j_1} \in SL2: \ \forall \tau_{j_2} \in m_2, j_1 \neq j_2: (\forall k_1 \in K: (\forall k_2 \in K: ec_{j_1,i,k_1} - sc_{j_2,i,k_2} \leq M \times b_{j_1,j_2,k_1,k_2})) \tag{13}$$

$$\forall \tau_{j_1} \in SL2: \ \forall \tau_{j_2} \in m_2, j_1 \neq j_2: (\forall k_1 \in K: (\forall k_2 \in K: ec_{j_2,i,k_2} - sc_{j_1,i,k_1} \leq M \times (1 - b_{j_1,j_2,k_1,k_2}))) \tag{14}$$

$$\forall \tau_j \in SL2: \ \sum_{k=1}^{2}(ec_{i,j,k} - sc_{i,j,k}) = cpm_{j,i} \times z_i \times x_{j,i} \tag{15}$$

$$\forall \tau_j \in SL2: \ \forall k \in K: sc_{j,i,k} \leq M \times z_i \times x_{j,i} \tag{16}$$

$$\forall \tau_j \in SL2: \ \forall k \in K: ec_{j,i,k} \leq M \times z_i \times x_{j,i} \tag{17}$$

Since the GPUs are under non-preemptive EDF scheduling, the tasks on the GPUs cannot be broken down into chunks and the length of the second chunks of the tasks has to be 0. Thus, if $r_i$ is GPU, the constraints (9), should be satisfied for all other tasks in addition to the predicted task.

Due to its complexity, the MIP-based optimization described in this section is not applicable in practice. Nevertheless, we use it in our experiments in order to evaluate the efficiency of the fast heuristic proposed in the next section.

*B. Fast heuristic*

For our fast heuristic, we consider the processing resources as knapsacks with certain capacities, and the tasks are the items with certain weights. The capacity of each resource $r_i$ is expressed in available processing time. At each activation of the RM, this capacity is equal with length of time window $\overline{K}$ introduced in Sec. III. The weight of task $\tau_j$ on $r_i$ is equal to $cpm_{j,i}$. Our proposed resource management algorithm is based on the knapsack heuristic presented in [2]. It has the worst case complexity of

**Algorithm 1** Mapping Heuristic

---

**Require:** $\overline{S}$ , $N$ , $p_{j,i}$, $cp_{j,i}$, $\overline{K}$, $em_{j,k,i}$, $cm_{j,k,i}$

**Ensure:** $\overline{y_j} = map(\tau_j)$

1: $U = \{1 \dots |\overline{S}|\}$           ▷ index of active tasks

2: $R = \{1 \dots N\}$           ▷ index of resources

3: **for** *each* $r_i$ *in* $R$ **do**

4:     $\overline{K}_i = K$

5:     **for** *each* $\tau_j$ *in* $U$ **do**

6:        $f_{j,i} = e_{j,i} + em_{j,k,i} + M \times ((cpm_{j,i}) > t_{\text{left}_j})$

7: **while** $U \neq \emptyset$ **do**

8:     $d^* = -\infty$

9:     **for** *each* $\tau_j \in \overline{S}$ **do**

10:        $F_j = \{r_i \in R | cpm_{j,i} \leq \overline{K}_i\}$

11:        **if** $F_j \neq \emptyset$ **then**

12:           $i^* = argmin\{f_{j,i} | i \in F_j\}$

13:           **if** $F_j \setminus \{i^*\} = \emptyset$ **then**

14:              $d = +\infty$

15:           **else**

16:              $i^{'} = argmin\{f_{j,i} | i \in F_j \setminus \{i^*\}\}$

17:              $d = f_{j,i'} - f_{j,i^*}$

18:              **if** $d > d^*$ **then**

19:                 $d = d^*$

20:                 $j^* = j$

21:                 $F_{j^*} = F_j$

22:     **while** $y_{j^*} = 0$ **do**

23:        **if** IsSchedulable$(j^*, i^*)$ **then**

24:           $y_{j^*} = i^*$

25:           $\overline{K}_{i^*} = \overline{K}_{i^*} - cpm_{j^*,i^*}$

26:           $U = U \setminus \{j^*\}$

27:        **else**           ▷ $\tau_{j^*}$ cannot be scheduled on resource $i^*$

28:           $F_{j^*} = F_{j^*} \setminus \{i^*\}$

29:           **if** $F_{j^*} = \emptyset$ **then**           ▷ no more resources

30:              *break*

31:           **else**

32:              $i^* = argmin\{f_{j^*,i} | i \in F_{j^*}\}$           ▷ pick next best $r_i$

$O(NLlogL)$, where $N$ is the number of resources and $L$ is the number of tasks. The actual complexity depends on the number of tasks in set $\overline{S}$ (see Sec. III), which at any activation of the resource manager is much smaller than $L$. The proposed heuristic is described in Algorithm 1.

Lines 1–6 initialize the algorithm. Note that $\overline{y_j} = map(\tau_j)$ is a vector, that specifies, for each task $\tau_j$ the index $i$ of the resource that the task is mapped to. If no feasible mapping is found for certain tasks, their $\overline{y_j}$ will be zero, and these tasks will not be allowed to become active by the RM. Let $f_{j,i}$ be a measure of desirability of assigning task $\tau_j$ to resource $r_i$. A smaller $f_{j,i}$ means a lower level of energy consumption, so smaller values are preferred. On line 6, $M \times ((cpm_{j,i}) > t_{\text{left}_j})$ is added to $f_{j,i}$ in order to make it undesirable if $\tau_j$ is not executable on $r_i$. In this algorithm, the tasks that are not yet mapped to any resource are considered iteratively (line 7), and the task $\tau_{j^*}$ that has the maximum difference between the smallest and the second smallest desirability is determined (lines 9–20). Once task $\tau_{j^*}$ is identified, one has to decide where it should be mapped (considering the schedulability constraints). The RM tries to map the task to resource $r_{i^*}$ for which $f_{j^*,i^*}$ is minimum (lines 23–26). If the scheduling constraint for resource $i^*$ is violated (considering the tasks mapped to it so far), it tries to map $\tau_{j^*}$ to the resource $i^*$ for which $f_{j^*,i^*}$ is the next smallest (lines 27–32). The iteration is continued until the RM manages to map $\tau_{j^*}$ to a resource (fulfilling the scheduling constraints), or until all resources have been considered, and no feasible mapping has been found (lines 29–30).

The IsSchedulable function in Algorithm 2 checks the schedulability of $\tau_{j^*}$ on resource $r_{i^*}$ given the set of tasks that are mapped on $r_{i^*}$ so far. The scheduling in function IsSchedulable is performed according to the principles outlined in Sec. III and Sec. IV-A. It is based on the EDF ordering of tasks on each resource and on checking that termination occurs before the deadline. Preemption caused by the predicted task is considered except for nonpreemptable resources, like GPUs. If all tasks are schedulable, IsSchedulable returns true. If, finally, a mapping has been produced such that all tasks meet their deadline the arriving task is admitted. The details of IsSchedulable function are as follows. In line 2, it checks if $r_{i^*}$ has no capacity for $\tau_{j^*}$, it return False. If $\tau_p$ is among the tasks mapped on $r_{i^*}$, it cannot start its execution earlier than its arrival. Thus, this condition should be checked (lines 8). If it is satisfied, its index should be saved (line 9), and there will be a gap between the arrival time of $\tau_p$ and the time when it can be scheduled on $r_{i^*}$ based on EDF; otherwise, based on the EDF ordering of tasks on $r_{i^*}$, it checks termination occurs before the deadline for all tasks. In the case that there is a gap between the arrival of $\tau_p$ and the time that it can start its execution based of EDF ordering, the gap can be filled with tasks that have later deadlines (lines 16–39). One should be careful here as well, since the execution of tasks is non-preemptive on GPUs. Therefore, if $r_{i^*}$ is a GPU, we cannot break it down into two chunks (lines 21–23).

---

**Algorithm 2** Schedulability Check
---

**Require:** $\overline{y_j}$, $cp_{j,i}$, $\overline{K}_i$, $cm_{j,i}$, $t$

1: **function** IsSchedulable($j^*, i^*$)

2:     **if** $\overline{K}_{i^*} < cpm_{j^*,i^*}$ **then**                                ▷ check if there is a room on $r_{i^*}$ for $\tau_{j^*}$

3:        **return** $False$

---

## V. Conclusion and summary

In this paper, the goal is resource management with prediction. To this end, we have presented a MILP formulation and a fast heuristic for resource management with prediction. We described both methods in details in Sec. IV.

```
4:    SR = The list of tasks mapped on r_{i*} plus τ_{j*} sorted by their deadline
5:    t_sum = 0
6:    ind_p = −1                                    ▷ a variable to keep the index of τ_p, in the case p ∈ SR
7:    for k in 1 to |SR| do
8:        if s_k − t > t_sum then
9:            ind_p = k
10:           break
11:       else
12:           t_sum = t_sum + cpm_{k,i*}
13:           if t_sum > t_left_k then
14:               return False
15:   if ind_p ≠ −1 then
16:       ind_frac = −1     ▷ a variable to keep the index of preempted task, which fills the gap with some of its portions
17:       gap = (s_{ind_p} − t) − t_sum ▷ gap is the difference between the current value of t_sum and the time that τ_p will
      arrive relative to the current moment t
18:       for k in ind_p to |SR| do
19:           ind_frac = k
20:           pr = min(1, gap/cpm_{k,i*})                        ▷ pr is the portion of τ_k that can be executed in the gap
21:           if i* is GPU then                                  ▷ execution of tasks is non-preemptive on GPUs
22:               pr = 0
23:               break
24:           t_sum = t_sum + pr * cpm_{k,i*}
25:           if t_sum > t_left_k then
26:               return False
27:           gap = gap − cpm_{k,i*}
28:           if pr < 1 then
29:               break
30:       t_sum = t_sum + pr * cpm_{ind_p,i*}
31:       if t_sum > t_left_{ind_p} then                         ▷ checking the schedulability of τ_p
32:           return False
33:       if ind_frac ≠ −1 then ▷ checking the schedulability of the fragmented task and the rest tasks (which have lower
      priority than τ_p based on EDF)
34:           for k in index_frac to |SR| do
35:               t_sum = t_sum + (1 − pr) * cpm_{k,i*}
36:               if t_sum > t_left_k then
37:                   return False
38:               pr = 0
39:   return True
```

REFERENCES

[1] D.-S. Chen *et al.*, *Applied integer programming: modeling and solution*. John Wiley & Sons, 2011.
[2] S. Martello, *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons Ltd., 1990.