# Chapter 14
# Designing Distributed Real-Time Systems to Process Complex Control Workload in the Energy Industry


**Eduardo Valentin, Rosiane de Freitas and Raimundo Barreto**

**Abstract** The energy industry demands computing system technologies with advanced state-of-the-art techniques to achieve reliability and safety for monitoring and properly dealing with several complex constraints. These computing systems also require delivering correct data at the right time imposing hard real-time constraints, because there are lots of situations where missing critical data may be catastrophic. The challenges faced by computer engineers in the energy industry also include designing distributed real-time systems to process such complex control workload. Besides, the computing system may also demand high energy consumption on its own. In this chapter, we demonstrate how to construct a mathematical formulation applicable for these computing systems and how to solve it to distribute the hard real-time workload of the process control systems considering technological constraints and optimizing for low power consumption of such computing systems. We present two computational techniques of resolution: an exact algorithm based on Branch-and-Cut and a meta-heuristic based on Genetic Algorithm. While the exact algorithm combines a branch-and-cut strategy with response time based schedulability analysis, the genetic algorithm still considers the response time schedulability analysis but follows an evolutionary solving strategy. Both computational techniques deliver solutions for heterogeneous computing systems with a control application, considering precedence, preemption, mutual exclusion, timing, temperature, and capacity constraints. In computational experiments, we present the usage of such techniques in a case study based on a control system for a power plant monitoring application.

E. Valentin (✉) · R. de Freitas · R. Barreto
Instituto de Computação - Ufam, Federal University of Amazonas, Manaus, AM, Brazil
e-mail: eduardo.valentin@icomp.ufam.edu.br

R. de Freitas
e-mail: rosiane@icomp.ufam.edu.br

R. Barreto
e-mail: rbarreto@icomp.ufam.edu.br



C. Kahraman and G. Kayakutlu (eds.), *Energy Management—Collective and Computational Intelligence with Theory and Applications*, Studies in Systems, Decision and Control 149, https://doi.org/10.1007/978-3-319-75690-5_14

## 14.1 Introduction

Engineers working in the energy field have an increasing need for state-of-the-art computer system technology. Challenges involve reliability and safety in refinery and power distribution operations. Delivering correct data at the right time imposes hard real-time constraints in these systems, because in many situations missing critical data may be catastrophic. Nevertheless, the computing systems applied in the energy sector may also demand high energy consumption on its own due to the necessity to continuously deliver reliable and trustworthy results.

In large data centers, for example, power consumption is a major concern due to the increasing expense in room cooling systems and mainly due to the expensive power bills. For instance, according to Eric Schmidt, CEO of Google, "What matters most to the computer designers at Google is not speed but, power, low power, because data centers can consume as much electricity as a city" (Markoff and Lohr 2002). Also, according to the Department of Energy (DoE) of the United States, power is one of the major challenges to overcome to achieve the needed computing excellence required to advance in many applications of the energy industry (Department of Energy (DoE) 2014).

In this chapter, we demonstrate how to construct a mathematical formulation applicable for these computing systems and how to solve it to distribute the hard real-time workload of the process control systems considering technological constraints and optimizing for low power consumption of such computing systems.

The organization of this chapter is as follows. We present how a typical computing system architecture and how a hard real-time task model for a energy sector monitoring application look like in Sects. 14.2 and 14.3, respectively. We show how a mathematical formulation can be written associating combinatorial optimization with schedulability analysis in Sect. 14.4. We also explain two strategies to solve such formulation, evolutionary based and branch-and-cut based, in Sects. 14.5 and 14.6. We also exemplify how such advanced techniques can be applied in a case study for a monitoring control application of the energy sector in Sect. 14.7. We close this chapter with final comments in Sect. 14.8.

## 14.2 Typical Multi-processor Architecture

Practitioners execute applications with hard deadline restrictions on multiple heterogeneous processors due to the expected energy consumption reduction. Nevertheless, developing software with timing constraints for multiple heterogeneous processors is a complex task. Scheduling becomes especially hard to deal with, particularly under low power constraints.

Adopting multiple processing elements to enhance the computing capability and to reduce the power consumption is a common design strategy, especially for embedded systems. Therefore, the heterogeneous multicore platforms have become

the de facto solution to cope with the rapid increase of system complexity, reliability, and energy consumption (He and Mueller 2012).

For this reason, a simple way to create a processing model for the energy industry applications is to use as reference a Multi-Processor System-On-Chip (MPSoC) architecture. We can state then that the system is composed by a set, H, of m processors, $H = \{H_1, H_2, \ldots, H_m\}$. Each core may operate on l different performance states, $1 \leq k \leq l$. The frequency of performance state k on the processor i is $F_{ik}$ and the power consumption is $P_{ik}$. The set of frequencies of one core is not necessarily the same of other cores. Also, a task may have different code size and execution time for different processors, due to instruction set and performance state differences. The idle power of processor i is $P_{idle,i}$.

## 14.3 Hard Real-Time Workload Model

A typical hard real-time workload can be represented by a task model of periodic tasks. A task model M is a set composed by n task $\tau_j$. A task $\tau_j \in M$, with $1 \leq j \leq n$, has the properties: worst-case execution cycle $WCEC_j$; worst-case execution time $C_j(f)$, which is a function of frequency f, thus $C_j(f) = \frac{WCEC_j}{f}$; period of execution $T_j$; deadline $D_j$. A task $\tau_j$ also has the following properties, specific to fixed priority policies: fixed priority $p_j$; set of high priority tasks $hp_j$ representing the tasks $\tau_j$ with a priority higher than the priority of $\tau_j$. The response time $R_j$ is dependent not only on task set characteristics, but also on the target platform, and on the task allocation and frequency distribution that have been selected for the workload. A task model can be locally processed in a single processor using a fixed priority based on-line scheduler, such as Deadline Monotonic.

Deadline Monotonic (DM) is a fixed priority based on-line scheduler in which task priorities decrease with larger deadlines. Audsley et al. (1993) extend the schedulability test proposed by Lehoczky et al. (1989) for DM, considering the release jitter $J_j$ and the local blocking delay $B_j$ due to semaphore usage. The delay $B_j$ caused by low priority tasks accessing shared resources in the same processors using Priority Ceiling Protocol can be estimated as $B_j = \max_{jk}\left\{D_{jk} | \left(p_j < p_i\right) \wedge (C(S_k) \geq p_i)\right\}$, where $C(S_k)$ is the ceiling priority of the shared resource $S_k$. The schedulability test proposed by Audsley is $R_j \leq D_j$, $\forall 1 \leq j \leq n$, where $R_j = I_j + J_j$.

The task influence $I_j$ in multiple processors may be calculated as $I_j^{n+1} = C_j + B_j^r + B_j + \sum_{p \in hp(j)} \frac{I_j^n + J_p + B_p^r}{T_p} \times C_p$. Precedence constraints can be represented by including the maximum response time of the predecessors tasks in the $J_j$ component of the task $\tau_j$. Also, when precedence constraints occur across different processors, this imposes an additional messaging cost that may be incorporated in the emitting task to perform inter-processor communication. When the Multiprocessor Priority Ceiling Protocol is in place to avoid priority inversion

issues, the remote blocking delay $B_j^r$ is an upper bound for the blocking time suffered by task $\tau_j$ from other tasks in a different processor. Response time tests are computationally expensive but provide exact conditions, i.e., sufficient and necessary. The test uses task's WCEC, periods, and the concept of critical instant phasing (Lehoczky et al. 1989).

## 14.4 Mathematical Formulation

A classical mathematical model that resembles modern heterogeneous multicore platforms is the Multilevel Generalized Assignment Problem—MGAP (Glover et al. 1979), though it was originally conceived in the manufacturing context. The MGAP consists of minimizing the assignment cost of a set of jobs to machines, each having associated therewith a capacity constraint. Each machine can perform a job with different performance states that entail different costs and amount of resources required. The MGAP is originally in the context of large manufacturing systems as a more general variant of the well-known Generalized Assignment Problem (GAP). In this paper, we correlate MGAP model with the problem of assigning frequencies and distributing hard real-time tasks on heterogeneous processors, minimizing energy consumption.

Considering the schedulability test proposed by Audsley, we propose the MGAP formulation using tasks response times as seen in Eqs. 14.1a–14.1f, based on the formulation of Valentin et al. (2016b).

$$\text{Minimize } \Psi(x) \tag{14.1a}$$

$$\text{s.t.} : \sum_{i=1}^{m} \sum_{k=1}^{l} x_{ijk} = 1,\ j \in \{1, \ldots, n\} \tag{14.1b}$$

$$\sum_{j=1}^{n} \sum_{k=1}^{l} \left( \frac{WCEC_{ij}}{F_{ik} T_j} x_{ijk} \right) \leq 1, \quad i \in \{1, \ldots, m\} \tag{14.1c}$$

$$\Psi_i \leq \frac{\kappa_i^{max} - \kappa_{amb}}{\rho}, \quad i \in \{1, \ldots, m\} \tag{14.1d}$$

$$R_j \leq D_j, \quad j \in \{1, \ldots, n\} \tag{14.1e}$$

$$x_{ijk} \in \{0, 1\}, \quad 1 \leq i \leq m,\ 1 \leq j \leq n,\ 1 \leq k \leq l \tag{14.1f}$$

where the tri-indexed decision variable $x_{ijk}$ represents the distribution and assignment, i.e. when $x_{ijk} = 1$ the task $\tau_j$ executes in the processor i at performance state k, or frequency $F_{ik}$, when $x_{ijk} = 0$, the task $\tau_j$ is distributed somewhere else.

A distribution is a partitioned approach in which each processor i executes a local scheduler responsible for a partition of the real-time task workload and migration is not allowed (see the set of constraints 14.1b). The set of constraints 14.1c represent the maximum system utilization capacity of each processor i. The set of constraints 14.1d represent the temperature limits by creating a linear relation, where $\kappa_{amb}$ is the ambient temperature, $\kappa_i^{max}$ is the maximum junction temperature of each processor i, $\rho$ thermal resistance constant, and $\psi_i$ is power consumption of each processor i. This formulation applies each task deadline as a constraint against their response time in the linear programming (see the set of constraints 14.1e). The matrix $R_j$ is the response time of tasks $\tau_j$ for a given allocation configuration. The response time of each task varies depending on the workload distribution and the frequency assignment of the configuration because a change in the value of $x_{ijk}$ may result in a different computation time ($C_i$). Equation 14.1 is applicable for DM scheduling policy ($D_j \leq T_j$).

We are using an objective function $\Psi(x)$ that minimizes energy consumption, accounting dynamic and idle energy, over the time window represented by the hyperperiod of the real-time tasks, i.e., the Least Common Multiple (LCM) of tasks periods. We extend the objective functions presented by Valentin et al. (2016b) by improving the idle energy estimation. Equations 14.2a–14.2c has the objective function.

$$\text{Minimize } \Psi(x) = \sum_{i=1}^{m} \left(E_{dyn,i}(x) + E_{idle,i}(x)\right) \tag{14.2a}$$

$$E_{dyn,i}(x) = \sum_{j=1}^{n} \sum_{k=1}^{1} \left( \left(\frac{LCM}{T_j}\right) C_l WCEC_{ij} V_{dd,ik}^2 x_{ijk} \right) \tag{14.2b}$$

$$E_{idle,i}(x) = P_{idle,i}\, LCM \left(1 - \sum_{j=1}^{n} \sum_{k=1}^{1} \left(\frac{WCEC_{i,j}}{F_{ik} T_j} x_{ijk}\right)\right) \tag{14.2c}$$

where $E_{dyn,i}$ is the energy consumption when processor i is active, $E_{idle,i}$ is the energy consumption when processor i is idle, $\frac{WCEC_{ij}}{F_{ik}T_j}$ represents the task $\tau_j$ utilization, $u_{ijk}$, while executing in processor i at frequency $F_{ik}$ of performance state k, is the circuit capacitance constant, and $V_{dd,ik}$ is the voltage level to achieve frequency $F_{ik}$.

The term $\left(\frac{LCM}{T_j}\right) C_l WCEC_{ij} V_{dd,ik}^2 x_{ijk}$ represents the dynamic energy associated with the instances of execution of task j within the LCM. Each processor idle energy, within the LCM time window, is computed for its estimated idle time in the term $P_{idle,i} LCM \left(1 - \sum_{j=1}^{n} \sum_{k=1}^{1} \left(\frac{WCEC_{i,j}}{F_{ik}T_j} x_{ijk}\right)\right)$.

The objective function represented in Eqs. 14.2a–14.2c may still be seen as a MGAP formulation. Note that, without loss of generality, when we take the term

$P_{idle,i}LCM$ out of the sum, leaving the term $mP_{idle,i}LCM$ to be added to the final objective function value, we have $c_{ijk} = \left[\left(\frac{LCM}{T_j}\right)C_l WCEC_{ij} V^2_{dd,ik} - P_{idle,i}LCM\left(\frac{WCEC_{i,j}}{F_{ik}T_j}\right)\right]$.

## 14.5 Computational Techniques of Resolution

In this section, we explain the algorithmic strategy developed for the mathematical formulation of Sect. 14.4. In Sect. 14.5.1, we explain an evolutionary algorithm which produces an initial solution that can be used by the exact algorithm for finding optimal solutions, described in Sect. 14.5.2.

### *14.5.1 Approximation by Means of Evolutionary Algorithm (EA)*

We wrote an evolutionary algorithm (EA), based on genetic algorithm, for each mathematical model (Valentin 2009). We follow a similar approach as existing in the literature for other formulations on this problem (Goossens et al. 2008). The algorithm's input is the processing model H and the desired task model M (see Sect. 14.3). In our EA implementation, a solution is a chromosome that is a sequence of 0's and 1's and each gene represents one of the elements of the tri-indexed decision variable of the mathematical model. The algorithm can be simplified into two steps: (i) Initialization with random-generated individuals and (ii) Generations composed by individuals selected in tournaments and by the evolutionary operators of elitism and crossover. Algorithm 1 illustrates the overall process of our EA strategy and we describe the pieces of the EA as follows.

In the Initialization, we random-generate individuals. Random-generating individuals do not guarantee their feasibility, i.e. the generated individual may be infeasible. The process of validating or transforming individuals into feasible solution is onerous. Even then, we maintain all generations composed by feasible individuals only. We random-generate a large number of individuals, 5000, to start with a high diversity. If none of them is a feasible solution, we return the empty set $\varnothing$. If we find less than 50 feasible individuals, then we return the one with highest fitness. But when we find 50 feasible individuals, we repeat the following steps for a maximum of 100 generations, or 10 generations with same best fitness, and return the individual with best fitness. We perform the Elitism operator by always including the individual with best fitness in the next generation. We execute Selection by means of a tournament in the current population. Only 5 individuals, randomly selected, participate in the tournament. The winner of the tournament is the individual with best fitness among those participating of it. We also insert in the next generation the result of a Crossover between winners of two tournaments. The

crossover operation between individuals $I_1$ and $I_2$ is done by means of selecting a pivot gene p. The genes lower than p are copied from $I_1$, the remaining genes are copied from $I_2$. When resulting individual is not feasible, we return $I_1$, if fitness($I_1$) > fitness($I_2$), or $I_2$ otherwise. We define the Fitness function to be: 1/E(individual), where the function E(individual) is the estimated energy consumption for the individual in consideration. The function E is computed using the same energy estimation as in the objective functions of the integer programming mathematical formulations.

**Algorithm 1:** Evolutionary Algorithm (EA)

```
Input: ℋ, ℳ
Output: best feasible solution ub found
/* Random-generate 5000 individuals
* (feasible and infeasible),
* to start with high diversity.
*/
i ← initialization(5000);
/* Select 50 feasible individuals. */
p ← feasible(i, 50);
if |p|= 0 then
   return ∅;
end if
b ← prev ← best_individual(p);
if |p|< 50 then
   return b;
end if
g ← e ← 1;
while (g++ ≤ 100)and(e ≤ 10) do
   /* Evolve population. */
   tournament(p);
   selection(p);
   elitism(p);
   ub ← best_individual(p);
   if ub == prev then
      e++;
   end if
   prev ← ub;
end while
return (ub.structure, ub.val);
```

**Algorithm 2:** Branch-and-Cut (B&C)

```
Input: ℋ, ℳ, ub
Output: optimal solution (x*, v*)
v* ← ub.val; x* ← ub.structure;
L ← ILP^0;
while L != ∅ do
   n ← remove_node(L);
   if !schedulability(n) then
      continue;
   end if
   lp ← relaxation(n)
   (x, v) ← solve(lp);
   if x = infeasible then
      continue;
   end if
   p ← cut_planes(x, v);
   if p! = ∅ then
      add(p, lp);
      goto 7;
   end if
   if v ≥ v* then
      continue;
   end if
   if x is integer then
      v* ← v; x* ← x;
      continue;
   end if
   partition(lp);
end while
return (x*, v*);
```

### 14.5.2 Finding Optimal Solutions

We use a general branch-and-cut method combined with schedulability tests to conduct the process of finding optimal solutions. A branch-and-cut is a branch-and-bound with cut generation strategies. The algorithm's input is the processing model H, The desired task model M, and a possible upper bound ub,

with objective function value and the solution structure found by the EA. The algorithm outputs the optimal distribution of hard real-time tasks among the processors that consumes less power among the possible assignments, informing as well in which frequency each tasks may be executed, and the total system estimated energy. The general solving strategy is listed in Algorithm 2.

The algorithm starts by denoting the set L of active problem nodes to contain only the initial Integer Linear Problem. When the EA returns a feasible solution, the upper bound $v^*$ and the optimal solution $x^*$ are set to match the output of the EA, otherwise they are set to $+\infty$ and to NULL, respectively. The algorithm iteratively evaluates each element of the set L. Each problem node is initially tested against the schedulability test that fits for the problem scheduling policy. In the case the schedulability test accepts the node, then a regular branch-and-cut is followed. The linear relaxation of the node is then computed and solved. When the linear relaxation is feasible, a procedure of generation of cutting planes is performed and followed by a fathoming and pruning process. The problem node is then partitioned and new restricted problem nodes are derived and incorporated into L. The iterative process repeats until the set L is empty.

## 14.6 Analysis on EA Parameters

We have tuned the EA algorithm based on an analysis of five of its parameters: number of generations, size of population, number of individuals in the tournament, the use of elitism, and percentage of mutation. We considered the CPU time needed to solve an instance with 30 tasks and 50% of estimated target CPU utilization. In Fig. 14.1 we present some graphics in which the left column shows the average CPU time and in the right column we present the average solution energy consumption, for each analysed EA parameter. We plot only observations that could be collected within an execution of less than one minute of CPU time.

As we can observe in Fig. 14.1, as expected, the execution time of the EA increases with the number of generations used, but we have noticed almost no change in the energy consumption. Similar pattern is seen for the number of individuals participating in the tournaments. We see an improvement in the energy consumption when the size of the population is higher than 20, but increasing the size of the population also increases the EA execution time. We have decided to set the parameters population and generation to 50 and the parameter tournament to 5, to avoid increasing the EA execution time, but still finding solutions with lower energy consumption. We have noticed that when we enable mutation, specially with a rate higher than 7% the execution time of the EA increases considerably, reaching more than 1 min in this analysis, and therefore, we decided to disable mutation. We have not noticed any major difference in the convergence time when enabling or disabling elitism for this particular analysis, but we decided to keep it enabled to avoid loosing promising solutions found across generations.

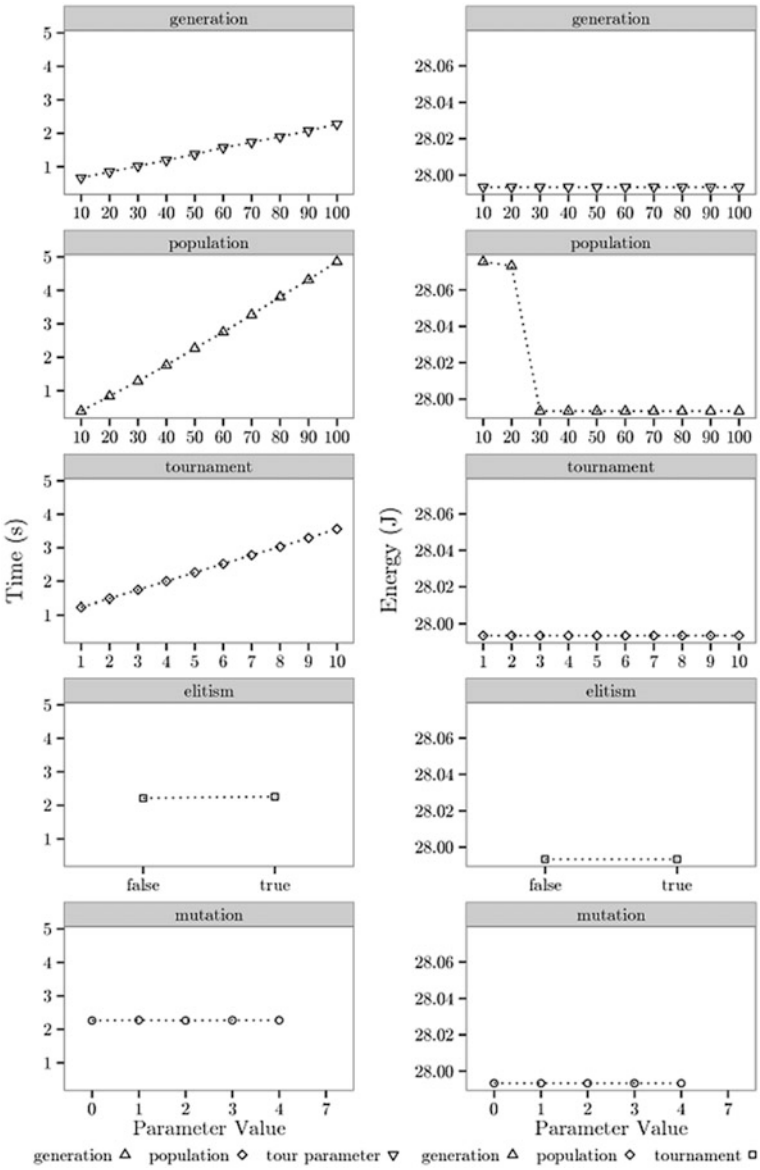


**Fig. 14.1** Analysis of the influence of EA parameters on the EA execution time and on the quality of the objective function (energy). Parameters: number of generations (generation), size of population (population), number of individuals in the tournament (tournament), the use of elitism (elitism), and percentage of mutation (mutation)

## 14.7 Case Study: Power Plant Monitoring Control

In this section, we exemplify how to optimally distribute the hard real-time workload of a power plant monitoring control in a target platform with multiple heterogeneous cores. Power plants depend typically on rotating machines such as steam turbines or generators and should be operated with maximum reliability, capacity, efficiency and minimum operating and maintenance costs. A shutdown of such machinery may be very costly and ideally avoided. Therefore, investing on identifying and potentially eliminating reliability issues through effective condition monitoring and predictive maintenance is key to modern power plant monitoring systems.

Table 14.1 summarizes the task model of the power plant monitoring example. We are considering precedence, preemption, mutual exclusion, temperature, capacity, and timing constraint while distributing the computing workload. We illustrate the precedence constraints (thin arrows) and mutual exclusion constraints (dark thick edges) of this task model in the precedence graph of Fig. 14.2.

**Table 14.1** An example of monitoring control hard real-time task model

| $\tau_i$ | $p_i$ | $T_i$ (ms) | $D_i$ (ms) | $WCEC_i$ ($\times 10^3$) | | | |
|---|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 | 4 |
| 1 | 1 | 200 | 100 | 3000 | 3000 | 3000 | 3000 |
| 13 | 2 | 200 | 100 | 15,000 | 15,000 | 15,000 | 15,000 |
| 14 | 3 | 200 | 100 | 10,000 | 10,000 | 10,000 | 10,000 |
| 15 | 4 | 200 | 100 | 1000 | 1000 | 1000 | 1000 |
| 7 | 5 | 200 | 100 | 2000 | 2000 | 2000 | 2000 |
| 3 | 6 | 200 | 40 | 3000 | 3000 | 3000 | 3000 |
| 16 | 7 | 200 | 200 | 5000 | 5000 | 5000 | 5000 |
| 17 | 8 | 200 | 100 | 7000 | 7000 | 7000 | 7000 |
| 2 | 9 | 200 | 200 | 3000 | 3000 | 3000 | 3000 |
| 11 | 10 | 200 | 200 | 2000 | 2000 | 2000 | 2000 |
| 18 | 11 | 200 | 40 | 6000 | 6000 | 6000 | 6000 |
| 10 | 12 | 200 | 40 | 2000 | 2000 | 2000 | 2000 |
| 4 | 13 | 200 | 100 | 3000 | 3000 | 3000 | 3000 |
| 5 | 14 | 200 | 100 | 3000 | 3000 | 3000 | 3000 |
| 20 | 15 | 200 | 100 | 2000 | 2000 | 2000 | 2000 |
| 22 | 16 | 200 | 100 | 7000 | 7000 | 7000 | 7000 |
| 21 | 17 | 200 | 100 | 1000 | 1000 | 1000 | 1000 |
| 9 | 18 | 200 | 100 | 2000 | 2000 | 2000 | 2000 |
| 8 | 19 | 200 | 100 | 2000 | 2000 | 2000 | 2000 |
| 6 | 20 | 200 | 200 | 3000 | 3000 | 3000 | 3000 |
| 19 | 21 | 200 | 200 | 10,000 | 10,000 | 10,000 | 10,000 |
| 12 | 22 | 200 | 200 | 2000 | 2000 | 2000 | 2000 |

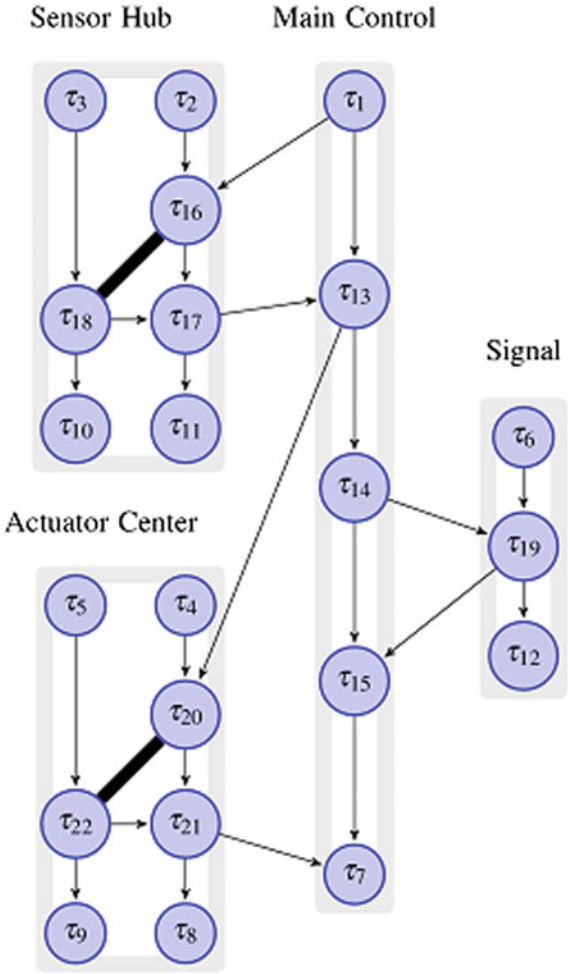


**Fig. 14.2** Precedence graph of power plant monitoring control. Arrows represent a precedence constraint, for example, $\tau_1$ precedes $\tau_{13}$. Dark thick edges represent mutual exclusion constraint, for example, $\tau_{16}$ shares a resource with $\tau_{18}$

The application example we consider, the Power Plant Monitoring Control, is composed of four logical activities that communicate among themselves: the Main Control activity, the Sensor Hub activity, the Actuator Center activity, and the Signal activity. The Main Control activity is responsible for managing the overall control system and communicating with the other activities. The Sensor Hub activity monitors environment, the Actuator Center activity is in charge of performing actions on the event of detection of failure or reliability issues, and the Signal activity reports and records any significant event detected in the system.

The Main Control activity always starts by requesting ($\tau_1$) data from the Sensor Hub. Current environment condition data is then sent back to the Main Control ($\tau_{17}$). The Main Control computes trend based on current and past environment data extrapolating and forecasting any equipment failure or reliability issues and communicates with Actuator Center ($\tau_{13}$) to implement any failure mitigation ($\tau_{20}$) or equipment adjustment ($\tau_{22}$) needed. Main Control also sends ($\tau_{14}$) regular reports of the events that happen in the control system to the Signal activity, which is responsible for activating alarms and warnings.

As an example platform, we are considering four processors: two ARM A57's and two ARM A53's. The ARM A57's may operate on seven different frequencies from 500 MHz to 1.9 GHz, and the A53's may operate on seven different frequencies from 400 MHz to 1.2 GHz. The idle power consumption is 50 mW. The circuit capacitance constant $C_1$ is $1e-9 W \frac{V^2}{Hz}$. The thermal resistance $\rho$ is $0.11 \frac{C}{W}$. In this platform, we are considering the DVFS switching latency as an operation executed within the context switch of tasks with a cost of 30 ms, included in the release jitter $J_j$ of each task. More robust response time analysis considering the switching overhead in clusters and architecture influence (Valentin et al. 2015) may be also combined with the branch-and-cut algorithm when necessary. We list the platform characteristics in Table 14.2.

**Table 14.2** Architecture characteristics of a typical multi-core heterogeneous platform

| CPU | $C_l(WV^2/Hz)$ | $\kappa_i^{max}(C)$ | $\rho(C/W)$ | Voltages (V) | Frequencies (GHz) |
|---|---|---|---|---|---|
| 0 | 1e-09 | 125 | 0.11 | 0.94 | 1.9 |
| | | | | 0.86 | 1.8 |
| | | | | 0.86 | 1.7 |
| | | | | 0.78 | 1.6 |
| | | | | 0.77 | 1.5 |
| | | | | 0.77 | 1.0 |
| | | | | 0.77 | 0.5 |
| 1 | 1e-09 | 125 | 0.11 | 0.94 | 1.9 |
| | | | | 0.86 | 1.8 |
| | | | | 0.86 | 1.7 |
| | | | | 0.78 | 1.6 |
| | | | | 0.77 | 1.5 |
| | | | | 0.77 | 1.0 |
| | | | | 0.77 | 0.5 |
| 2 | 1e-09 | 125 | 0.11 | 0.82 | 1.2 |
| | | | | 0.82 | 1.1 |
| | | | | 0.7825 | 1.0 |
| | | | | 0.7575 | 0.9 |
| | | | | 0.7075 | 0.8 |
| | | | | 0.6825 | 0.7 |
| | | | | 0.6575 | 0.4 |

(continued)

**Table 14.2** (continued)

| CPU | $C_l(WV^2/Hz)$ | $\kappa_i^{max}(C)$ | $\rho(C/W)$ | Voltages (V) | Frequencies (GHz) |
|---|---|---|---|---|---|
| 3 | 1e-09 | 125 | 0.11 | 0.82 | 1.2 |
| | | | | 0.82 | 1.1 |
| | | | | 0.7825 | 1.0 |
| | | | | 0.7575 | 0.9 |
| | | | | 0.7075 | 0.8 |
| | | | | 0.6825 | 0.7 |
| | | | | 0.6575 | 0.4 |

Even though temperature is a constraint left for mechanical engineering, it can play a role while distributing the system workload. The ambient temperature in such machineries will typically be higher than the regular room temperature (25 °C) because the system is exposed to heat flowing from the mechanical engines, reaching as high as 85 °C. The common silicon junction temperature is 125 °C.

After executing the branch-and-cut optimization algorithm considering the task model of Table 14.1 and the target computing model of Table 14.2, we obtain the optimal distribution listed in Table 14.3. The precedence graph with the allocation is also illustrated in Fig. 14.3. For this case study, the optimal energy consumption is 0.1049 J for the duration of the LCM (200 ms) of tasks periods. We initialized the algorithm with the solution structure and an upper bound for the objective function extracted from the configuration found by the evolutionary algorithm. Utilizing this initial upper bound, the full optimization process took less than 1.5 h to finish and the final optimal solution differs from the logical initial distribution. Even though this case study has a set of 22 tasks, this algorithm has a reasonable performance on task models with up to 50 tasks, finishing in less than 30 min with a feasible solution for independent tasks (Valentin et al. 2016a, 2017).

As seen in Table 14.3, the schedulability analysis shows that the computed response time of each task is less than their respective deadline, meeting all timing, precedence, and mutual exclusion constraints. It is worth noting that Table 14.3 includes the inter-processor communication cost of tasks $\tau_4, \tau_5, \tau_6, \tau_{13}, \tau_{14}, \tau_{17}, \tau_{18}$, and $\tau_{19}$. The optimization process converged to an optimal solution in which tasks sharing resources are allocated in the same processor, avoiding remote blocking delays. The optimal configuration for this case study uses only three of the four available processors. The total utilization of the active processors (6.05, 16.98, and 25.00%) is well within their respective theoretical values (100%), safely respecting the capacity constraint. This configuration with low utilization is selected by the algorithm because it consumes the least energy, although it is common practice to design real-time systems with high utilization. Also, the estimated temperature of each ARM processors is less than 87 °C, in the thermal stabilization, giving enough room in the temperature constraint.

We highlight, for example, that the logical distribution setting each application activity to one processor is also feasible. This configuration uses all four processors

**Table 14.3** Optimal workload distribution result of the optimization process

| Processor: 0 | Utilization: 6.05% | Temperature: 86.85 °C | | | | |
|---|---|---|---|---|---|---|
| $\tau_i$ | WCEC ($\times 10^3$) | Frequency (GHz) | Computation (ms) | $T_i$ (ms) | $D_i$ (ms) | $R_i$ (ms) |
| 15 | 1000 | 1.9 | 0.526 | 200 | 100 | 97.331 |
| 20 | 2000 | 1.9 | 1.053 | 200 | 100 | 48.684 |
| 22 | 7000 | 1.9 | 3.684 | 200 | 100 | 13.219 |
| 21 | 1000 | 1.9 | 0.526 | 200 | 100 | 49.797 |
| 7 | 2000 | 1.9 | 1.053 | 200 | 100 | 98.443 |
| 8 | 2000 | 1.9 | 1.053 | 200 | 100 | 52.458 |
| 9 | 2000 | 1.9 | 1.053 | 200 | 100 | 18.512 |
| 10 | 2000 | 1.9 | 1.053 | 200 | 40 | 28.513 |
| 11 | 2000 | 1.9 | 1.053 | 200 | 200 | 38.019 |
| 12 | 2000 | 1.9 | 1.053 | 200 | 200 | 108.909 |
| Processor: 1 | Utilization: 16.98% | Temperature: 86.73 °C | | | | |
| $\tau_i$ | WCEC ($\times 10^3$) | Frequency (GHz) | Computation (ms) | $T_i$ (ms) | $D_i$ (ms) | $R_i$ (ms) |
| 1 | 3000 | 1.9 | 1.579 | 200 | 100 | 1.609 |
| 2 | 3000 | 1.9 | 1.579 | 200 | 200 | 3.188 |
| 3 | 3000 | 1.9 | 1.579 | 200 | 40 | 4.767 |
| 4 | 3001 | 1.9 | 1.579 | 200 | 100 | 6.346 |
| 5 | 3001 | 1.9 | 1.579 | 200 | 100 | 7.926 |
| 6 | 3001 | 1.9 | 1.579 | 200 | 200 | 9.505 |
| 16 | 5000 | 1.9 | 2.632 | 200 | 200 | 12.198 |
| 18 | 6001 | 1.9 | 3.158 | 200 | 40 | 18.483 |
| 17 | 7001 | 1.9 | 3.685 | 200 | 100 | 26.936 |
| 13 | 15,002 | 1.0 | 15.002 | 200 | 100 | 46.707 |
| Processor: 2 | Utilization: 25% | Temperature: 85.04 °C | | | | |
| $\tau_i$ | WCEC ($\times 10^3$) | Frequency (GHz) | Computation (ms) | $T_i$ (ms) | $D_i$ (ms) | $R_i$ (ms) |
| 14 | 10,001 | 0.4 | 25.003 | 200 | 100 | 71.739 |
| 19 | 10,002 | 0.4 | 25.005 | 200 | 200 | 96.774 |
| Processor: 3 | Utilization: 0.0% | Temperature: 85.005 °C | | | | |
| $\tau_i$ | WCEC ($\times 10^3$) | Frequency (GHz) | Computation (ms) | $T_i$ (ms) | $D_i$ (ms) | $R_i$ (ms) |

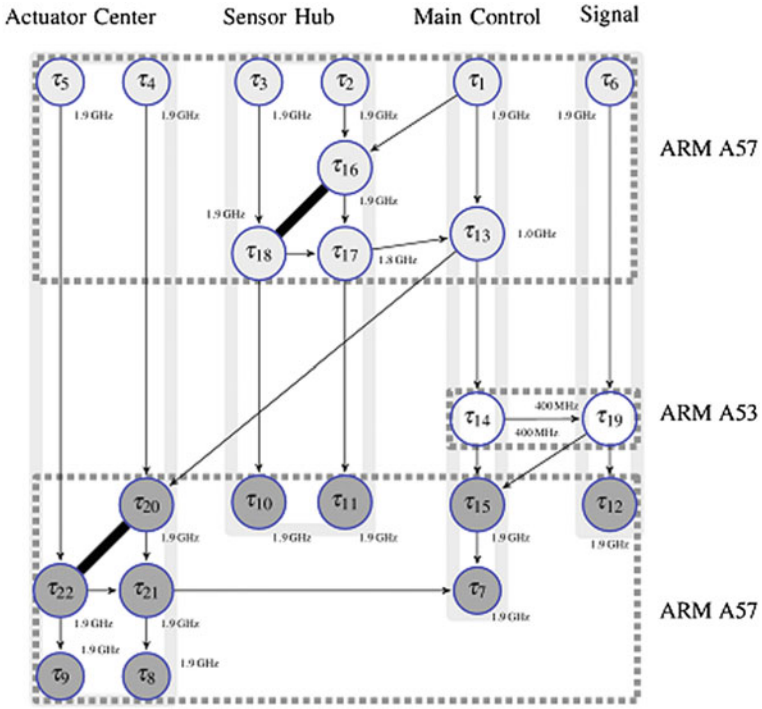


**Fig. 14.3** Precedence graph and task distribution of power plant monitoring control. White nodes are allocated in one ARM A53. Light gray nodes are allocated in one ARM A57. Dark gray nodes are allocated in the other ARM A57. The frequency that each task executes is represented close to each respective node in the graph, for example, $\tau_{19}$ executes at 400 MHz

at their respective maximum frequency. The timing, preemption, precedence, and mutual exclusion constraints are met, given that each task response time is less than their respective deadline. The capacity and temperature constraints are also met. However, this configuration's estimated total system energy is 0.1127 J for the LCM (200 ms) of tasks periods, being at least 7.4% higher than the optimal.

An intuitive approach would be to target a low power configuration, having all tasks allocated to a single ARM A53 CPU, executing at the lowest frequency of 400 MHz. That, however, is not a feasible configuration, given that the capacity constraint is not met because the total CPU utilization would be 117.5% and several tasks would not meet their deadlines in this situation.

Another intuitive approach would be to use again the logical distribution of one activity to one processor, but locking the lowest available frequency, as the utilization of each processor is not high. In this configuration, each processor utilization is less than 32%, but the system is not schedulable because the response time analysis indicates that tasks $\tau_7, \tau_8, \tau_{10}, \tau_{14}, \tau_{15}, \tau_{18}$, and $\tau_{21}$ miss their respective deadlines basically due to the accumulated precedence.

## 14.8 Final Remarks

In this paper, we exemplified how to optimally distribute the hard real-time workload of a power plant monitoring control system. We applied robust methods to avoid infeasible system configurations. Even though they can be computationally expensive, their usage in design time is still justified, given that they help prevent catastrophic scenarios.

We associated combinatorial optimization mathematical formulations and response time based schedulability analysis to optimally distribute the hard real-time workload of power plant monitoring system. We solved the combinatorial problem by using a branch-and-cut algorithm that applies response time analysis while walking through the problem nodes. We showed that all the considered constraints of precedence, preemption, mutual exclusion, timing, temperature, and capacity were met properly in our case study by using the response time analysis with branch-and-cut combined method.

We are evaluating combining response time analysis in a Branch-Cut-Price algorithm as future work. We also envision considering migration by performing sensibility analysis to determine other feasible and optimal configurations to allow for dynamic configuration switching.

## References

Audsley, N., Burns, A., Richardson, M., Tindell, K., & Wellings, A. J. (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal, 8* (5), 284–292.

Department of Energy (DoE). (2014). *Top ten exascale research challenges*. Visited in Jan 2016. URL: https://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf.

Glover, F., Hultz, T. J., & Klingnian, D. (1979). *Improved computer-based planning techniques*, part ii. Interfaces 9/4.

Goossens, J., Milojevic, D., & N´elis, V. (2008). Power-aware real-time scheduling upon dual cpu type multi-processor platforms. In *Proceedings of the 12th International Conference on Principles of Distributed Systems, OPODIS'08*, pp. 388–407. Berlin, Heidelberg: Springer.

He, D., & Mueller, W. (2012). Enhanced schedulability analysis of hard real-time systems on power manageable multi-core platforms. In *Proceedings of the 14th IEEE International Conference on HPCC—9th IEEE ICESS,* pp. 1748–1753, Liverpool.

Lehoczky, J., Sha, L., & Ding, Y. (1989). The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the IEEE Real Time Systems Symposium*. pp. 166–171.

Markoff, J., & Lohr, S. (2002). *Intel's huge bet turns iffy*.

Valentin, E. (2009). *Github—Hydra*. Visited in Feb 2016. URL: https://github.com/toolshydra/Hydra.

Valentin, E., de Freitas, R., & Barreto, R. (2016a). Reaching optimum solutions for the low power hard real-time task allocation on multiple heterogeneous processors problem. In *2016 VI SBESC*, pp. 128–135.

Valentin, E. B., de Freitas, R., & Barreto, R. (2016b). Applying MGAP modeling to the hard real-time task allocation on multiple heterogeneous processors problem. *Procedia Computer Science, 80,* 1135–1146. In *International Conference on Computational Science 2016*, ICCS 2016, 6–8 June 2016, San Diego, California, USA.

Valentin, E., de Freitas, R., & Barreto, R. (2017). Towards optimal solutions for the low power hard real-time task allocation on multiple heterogeneous processors. *Science of Computer Programming*.

Valentin, E., Salvatierra, M., de Freitas, R., & Barreto, R. (2015). *Response time schedulability analysis for hard real-time systems accounting dvfs latency on heterogeneous cluster-based platform* (pp. 1–8). Optimization and Simulation (PATMOS): Power and Timing Modeling.