SENAS: Security driven ENergy Aware Scheduler for Real Time Approximate Computing Tasks on Multi-Processor Systems

Krishnendu Guha
Department of Electrical and Computer Engineering
University of Florida, USA
Email: kguha@ufl.edu

Sangeet Saha, Klaus McDonald-Maier
School of Computer Science and Electronic Engineering)
University of Essex, UK
Email: sangeet.saha@essex.ac.uk, kdm@essex.ac.uk

***Abstract*—Present day real time approximate computing applications like image and video processing involves execution of a set of tasks before a certain amount of time or deadline. In addition to this, present day systems are associated with strict energy budget that cannot be changed post deployment. The tasks comprises of a mandatory and optional part. Completion of all mandatory portions of all tasks before deadline is much more important than result accuracy in such real time approximate computing applications. Based on the energy budget, the optional portions can be executed that determines the quality of service (QoS) of the system. In ideal scenario, sufficient energy budget is present that ensures completion of both mandatory and optional portions in a system with a pre-determined number of processors. However, if fault or malware attack occurs on one or more processors, then the system will cease to work and results may be fatal. In this work, we consider such a scenario where the processors may be faulty and stop functioning in post deployment phases or some malware may cause unexpected delays in processing or may cause unexpected power draining at runtime that will prevent the system from meeting its deadline. We propose a Security driven ENergy Aware Scheduler (SENAS) that works as a self aware agent. Initially, based on the available energy budget, SENAS determines which task is to be executed in which processor of a system. At runtime, SENAS constantly monitors the working of the processors and on detecting any anomaly in any of the processors, it reschedules its tasks at runtime by reducing execution of the optional portions of the tasks and ensuring completion before deadline with high QoS.**

***Index Terms*—Approximate Computation, Energy Aware Scheduling, Real-time Security, Multi-Processor, Power Draining**

I. Introduction

In real-time embedded systems, the correctness of the system not only depends on the precision of the results, but also on the time at which they are produced. Real-time applications in today's real-time embedded systems are often represented as Precedence-constrained Task Graphs (PTGs). Specifically, the entire application consists of a collection of tasks (aka nodes) under precedence constraints and dependencies between tasks [1]. However, power/energy for such systems is particularly important, as these devices often depend upon restricted power source such as batteries to deliver high performance and service quality [2].

This work is supported by UK EPSRC Council through grant number EP/X015955/1 and EP/V000462/1

978-1-6654-7355-2/22/31.00 ©2022$IEEE$

For critical systems, approximated results obtained in time are preferable on accurate results obtained after the deadline. For example, in a video application, initially an inaccurate, but acceptable quality image is generated from the received data. Then, based on the available resources and energy, the obtained image may be further refined [3]. Thus, Approximate Computation (AC) approaches [4] can minimize the possibility of tasks missing their deadline due to the fixed energy budget.

In AC approach, a task is decomposed into a mandatory part, followed by optional parts [5]. The mandatory part must be executed entirely to produce an acceptable result, while optional parts will be executed for refining the result. Application scenarios involve target localization task that is carried out by an unmanned aerial vehicle [6], [7]. Wei et al. introduce the concept of AC to handle uncertainties in energy availability of a real-time system. The idea was to accept approximate results, when energy budget is limited and continue more computations to achieve accurate results, if more energy is available [5]. This work is limited to independent tasks.

Energy aware scheduling of dependent AC tasks are considered in [8], [9]. Here, authors aimed at DVFS based technique to make the scheduling energy efficient. However, as DVFS decreases the supply voltage and frequency to save power, the transient fault rate / soft error rate of the system significantly increase [10], which reduce the system's reliability. In order to circumvent the problem, Xiang et al. proposed the idea of finding "energy efficient frequency" in a multi-core platform [11].

Present day systems have multiple processors. However, faults may occur with processors at runtime that may cease the processor from further functioning. Even malware like hardware trojan horses may infect the processor hardware by slowing down the processor speed, eventually affecting the real time application to miss its deadline [12]. Moreover, for scenarios where there is a strict energy budget, malware may cause unwanted power dissipation, which drains the energy budget of the system and ceases the system from functioning [13]. These prevent real time systems to complete their task before deadline. Hence, it is of utmost importance to deploy self aware security modules that can detect such anomalies and take appropriate measures at runtime.

The key highlights of this work include:

(i) Analysis of various threats for a real time approximate computing application, depicted as a PTG.

(ii) Development of a scheduling strategy that schedules AC tasks and determines the amount of optional portions that need to execute based on availability of energy budget.

(iii) Development of a self aware runtime security mechanism that detects anomaly in processors at runtime and takes appropriate measures to ensure completion before deadline.

(iv) Development of a low overhead SENAS (Security driven ENergy Aware Scheduler) that performs (ii) and (iii), which maximizes QoS in the available energy budget.

This paper is organized as follows. System model is discussed in Section 2, while Section 3 depicts the threat model. The proposed mechanism is presented in Section 4. Section 5 deals with experimentation and result analysis. Finally, the paper concludes in Section 6.

II. System Model

A. Processor Model

This work assumes a homogeneous multi-processor embedded processing platform. This comprises m processors, denoted as $V = \{v_1, v_2, ..., v_m\}$. Each processor has N_l discrete frequency levels, denoted as $L_0, ..., L_{N_l}$. Each level L_j can be characterized by $L_j : v_j, p_j, f_j,\ j \in 1...N_l)$, which represents voltage, average power, and frequency, respectively.

B. Task Model

We model a real-time application ($\mathcal{A}$), as a precedence constrained Directed Acyclic Graph (DAG) $G = (T, E)$, where T is a set of tasks ($T = \{T_i \mid 1 \leq i \leq |T|\}$) and E is a set of directed edges ($E = \{\langle T_i, T_j \rangle \mid 1 \leq i, j \leq |T|; i \neq j\}$), representing the precedence relations between distinct pairs of tasks. An edge $\langle T_i, T_j \rangle$ refers to the fact that task T_j can begin execution only after the completion of T_i.

Moreover, we have considered approximate computation tasks in this paper. Each task T_i $(1 \leq i \leq N)$ is logically decomposed into a mandatory part with execution requirement of M_i and an optional part with execution requirement of O_i. Each task has to execute M_i units in order to provide acceptable result. Execution of the optional part O_i can be started only after the completion of mandatory part M_i. The execution length l_i of task T_i can be defined as :

$$l_i = M_i + O_i \times \mu \tag{1}$$

where, μ denotes the ratio of the optional part, which is being executed and varies between 0 and 1. Thus, $\mu = 1$ denotes that we execute the entire O_i units and we will achieve maximum possible accurate result. It is further assumed that a task T_i may have k_i different versions; that is, $T_i = \{T_i^1, T_i^2, \ldots, T_i^{k_i}\}$. The energy En_i consumed by task T_i of length l_i can be defined as [14]:

$$En_i = P_{eff} \times l_i \tag{2}$$

III. Threat Model

A. Stoppage of working of a processor

This may occur either due to a fault at runtime or due to an inserted malware like hardware trojan that stops its operation. This is diagrammatically depicted in Fig. 2 (b) for the example presented in Fig. 2 (a).

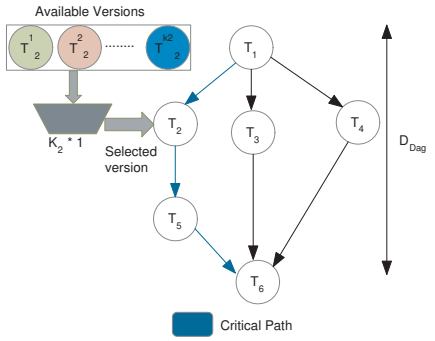
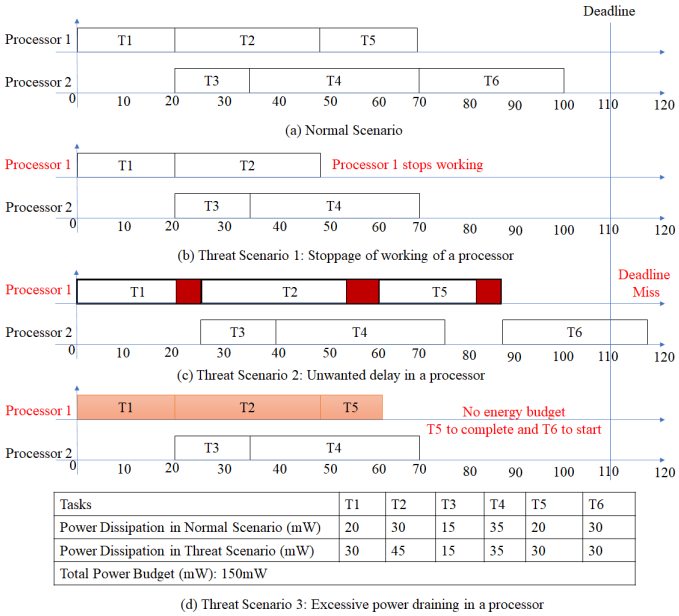


Fig. 1: The Task Graph



(a) Normal Scenario

(b) Threat Scenario 1: Stoppage of working of a processor

(c) Threat Scenario 2: Unwanted delay in a processor

Tasks	T1	T2	T3	T4	T5	T6
Power Dissipation in Normal Scenario (mW)	20	30	15	35	20	30
Power Dissipation in Threat Scenario (mW)	30	45	15	35	30	30
Total Power Budget (mW): 150mW						

(d) Threat Scenario 3: Excessive power draining in a processor

Fig. 2: Threat Model

B. Unwanted delay in task execution

This may occur either due to aging of the processor hardware or due to malware. These prevent the real time system from completing their tasks before deadline. This is shown in Fig. 2 (c) for the example presented in Fig. 2 (a).

C. Excessive power draining

Unwanted delay as discussed before will definitely cause additional power draining. However, adversaries may intentionally implant power dissipating circuitry that may be present in parallel to the main circuit. These may not have any effect on the timing but if unnoticed, will cause excessive power dissipation and ultimately drain the energy budget of the system. If energy budget of the system is drained in initial or mid stages, then tasks at the bottom of the PTG will not have enough energy to complete their execution. This is depicted in Fig. 2 (d) for the example presented in Fig. 2 (a).

IV. Proposed Methodology

In this work, we propose development of SENAS that contains two modules. A scheduling module and a security module. The scheduling module is associated with mapping appropriate task versions to the processors, based on available

Algorithm 1: Task Scheduling in Processors as per Available Energy Budget

Input:
i. $l_i^{\zeta_i}$: Execution length of the selected ζ_i^{th} version of T_i
ii. $E_i^{\zeta_i}$: Energy requirement for T_i's ζ_i^{th} version
iii. D_{Dag}: Deadline of the task graph.
iv. Ordered set of tasks, τ
Output: Generated Schedule

/*..... Initialization.....*/ /* Let FP denote the set of processors currently available for execution; */
Initialize $FP = V$;
$\forall\ v_i \in FP$, Set LD_i= FALSE; /* Initialization, LD_i : A flag which is set to FALSE if the Processor is available for execution; TRUE, otherwise, initially all processors are free */
/*.....TASK MAPPING & EXECUTION.....*/
for $t = 0$; $t \leq D_{dag}\ AND\ (\tau\ \&\ ENG\_BGT) \neq NULL$; $t++$ **do**
  for each available processor *in parallel* **do**;
  if $\exists\ T_j \in \tau \mid$ *All predecessors of* T_j *have finished their execution AND* $FP == NULL$ **then**
    if $ENG\_BGT \geq E_i^{\zeta_i}$ **then**
      Select processor(s) v_i with LD_i == FALSE ;
      Set PL_i = TRUE /* Set v_i to *busy*; */
      Map T_j *in processor* v_i;
      Te_j = t /* Set current time t as the execution start time of T_j*/
      BD_i= $l_j^{\zeta_j}$; /* start execution of T_j; BD_i: an integer variable denoting **B**usy **D**uration which holds the remaining time required to finish the current task in v_i */
      $FP = FP \setminus v_i$; /* Remove v_i from set FP */
    else
      Move to next smaller version (with lower O_i and go to step 9
  else
    BD_i = BD_i-1; /* Decrement remaining time */
    if (BD_i == 0) **then** FP = FP $\cup$ $\{v_i\}$; /* Add v_i to the set of free (available) processors */
      LD_i = FALSE; * Set v_i back to *free*; */
    FT = FT $\cup$ T_j /* Add T_j to set FT of finished tasks */
    $\tau = \tau \setminus T_j$; /* Delete T_j from set τ */
    $ENG\_BGT = ENG\_BGT - E_j^{\zeta_j}$; /* Reduce the consumed energy from the avilable budget */

energy budget at that time. The security module continuously monitors the working of the processors at runtime and on detecting any anomaly, calls the scheduler module to develop a fresh schedule based on available energy at that time. Fig. 3 shows a block diagram of the mechanism.

A. Task Scheduling with respect to Energy Budget

Algorithm 1 assigns tasks to all the nodes, if there exists enough processors and energy budget. Then it only considers nodes / tasks once all of its parent's nodes, i.e. *Predecessors*, have completed their executions. However, before assigning any task, the energy constraint is checked, and if fails, the algorithm tries to assign the lowest version of the task (with smaller O_i). A task (say, T_j) mapped to a processor (say, v_i) will keep running until the task's execution requirement is met. The variable BD_i represents the remaining execution need of T_j in v_i, and after T_j completes its execution, BD_i becomes zero. A task will be added to the set FT and deleted from τ once its execution is complete. The above processes of task mapping and execution continues iteratively till all tasks in τ complete their executions, deadline D_{Dag} is encountered or the given energy budget is exhausted.

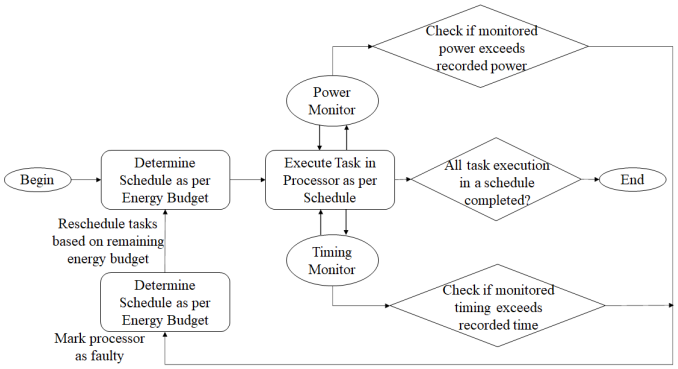


Fig. 3: Block Diagram

Algorithm 2: Runtime Security Mechanism

Input: Monitored Timing/ Power Data from Sensors, Recorded Timing/ Power Data
Output: Mark processor faulty/ normal

for *Task Ti executing in Processor Vi* **do**
  if *Monitored Power Dissipation* $>$ *Recorded Power Dissipation* **then**
    Mark Processor Vi faulty;
    Execute Algorithm 1 with the remaining tasks on remaining unfaulty processors;
else if *Monitored Time* $>$ *Recorded Time* **then**
  Mark Processor Vi faulty;
  Execute Algorithm 1 with present and remaining tasks on remaining unfaulty processors;
else
  Scenario is normal and proceed with next task execution as per schedule;

B. Runtime Security

1) Anomaly Detection: Anomaly refers to the threats discussed in Section 4. These are detected based on two parameters, timing and power.

Based on Timing Detection based on the parameter timing is effective to detect faults, aging and intentional delays. The security module monitors the start time and end time for a particular task in a processor. If the observed execution time is greater than the recorded worst case execution time, then it flags the processor as affected.

Based on Power Though enhanced power dissipation is associated with intentional delayed execution or unintentional aging, however, power monitoring is utmost essential in scenarios where the power dissipating circuitry is not a part of the main circuitry and is in parallel. This causes unintentional excessive power dissipation and drains the energy budget of

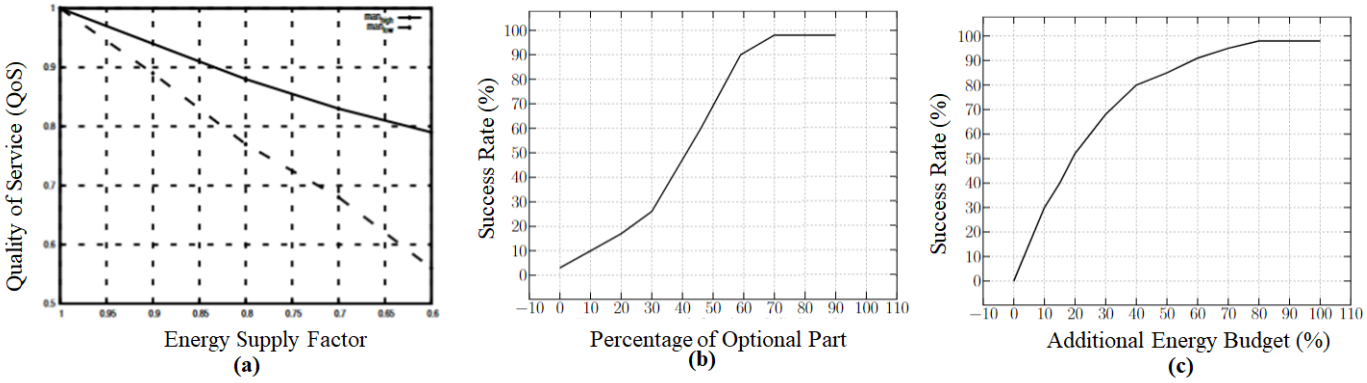


Fig. 4: Graphical Analysis (a) Effect of scheduling on QoS based on energy supply factor (b) Success rate with respect to optional part (c) Success rate with respect to additional energy budget

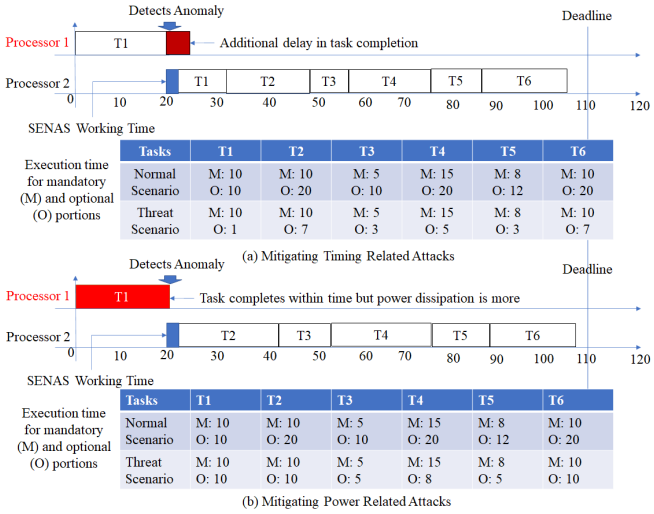


Fig. 5: Threat Mitigation

the system. The security module checks whether the observed power dissipation is more than the recorded worst case power dissipation. If so, then it flags the processor as affected.

2) Ensuring Security: If a processor is flagged, then it recalls the scheduler, which computes a fresh schedule for unaffected processors within the remaining energy budget.

Mitigation of timing related attacks is depicted in Fig.5(a), while that of power related attacks is depicted in Fig. 5(b), for the example considered in Threat Model.

V. Experimentation and Results

A. Experimentation

We have considered a large number of applications, which comprises several tasks. The mandatory and optional portions of the tasks are varied for experimentation.

B. Results

1) Results related to scheduling that maximizes QoS: We have used task graphs for free (TGFF) [15] to generate task graphs. We assumed that E_h denotes the total energy required to execute all the tasks with their highest version. In our experimental study, the supplied energy is defined as $E_s = \eta \times E_h$, where, η denotes the energy supply factor and varies from 1 to 0.6.

It can be observed that QoS increases with energy factor η. This is because higher values of η result in correspondingly higher energy supply and thus increases the possibility of achieving successful execution of more optional portions.

2) Results related to security: Success rate (SR) is the parameter considered for analyze the effectiveness of our proposed methodology. SR is the percentage ratio of the total number of real time applications that can complete within deadline in the available energy budget among the total number of applications considered for experimentation. SR is analyzed over percentage of optional portions in an application and additional energy budget in an application.

Analysis of SR over percentage of optional parts is depicted in Fig. 4 (b). In general, success rate varies proportionally to the percentage of optional parts in an application. This is because lower is the mandatory portions, which has to be completed to ensure success of an application. Thus, with low percentage of optional parts, SR is quite low, but with increase of optional parts, SR steadily increases and stabilizes at the end, as depicted in the figure.

Analysis of SR with respect to additional energy budget is shown in Fig. 4 (c). With increase in energy budget, SR increases exponentially and reaches the ideal scenario when additional energy budget is almost 75%. This is because the additional energy can be effectively used to execute the mandatory parts, which determines the SR, while normal energy can be used to execute the optional parts.

VI. Conclusion

Present day real time applications are AC in nature like image and video applications. In these, deadline is much more important than accuracy. If energy and time is available, results can be refined to ensure accuracy. Vulnerabilities may occur when tasks of such an real time approximate computing application is mapped to a multi processor system. The various types of threats related to timing and power is highlighted in this paper. We propose a real time scheduler that optimizes QoS for a set of tasks and processors. Additionally, we propose a self aware security module that detects vulnerability at runtime and uses the scheduler to ensure completeness of application before deadline.

REFERENCES

[1] G. L. Stavrinides and H. D. Karatza, "Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations," *Journal of Systems and Software*, vol. 83, no. 6, pp. 1004–1014, 2010.

[2] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, and P. Li, "Energy-efficient scheduling for real-time systems based on deep q-learning model," *IEEE Trans. on Sustainable Computing*, vol. 4, no. 1, pp. 132–141, 2017.

[3] H. Aydin, R. Melhem, D. Mosse, and P. Mejía-Alvarez, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 50, no. 2, pp. 111–130, 2001.

[4] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[5] T. Wei, J. Zhou, K. Cao, P. Cong, M. Chen, G. Zhang, X. S. Hu, and J. Yan, "Cost-constrained qos optimization for approximate computation real-time tasks in heterogeneous mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1733–1746, 2017.

[6] X. Wang, J. Liu, and Q. Zhou, "Real-time multi-target localization from unmanned aerial vehicles," *Sensors*, vol. 17, no. 1, p. 33, 2017.

[7] R. Venkatagiri, K. Swaminathan, C.-C. Lin, L. Wang, A. Buyuktosunoglu, P. Bose, and S. Adve, "Impact of software approximations on the resiliency of a video summarization system," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 598–609.

[8] L. Mo, A. Kritikakou, and O. Sentieys, "Energy-quality-time optimized task mapping on dvfs-enabled multicores," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2428–2439, 2018.

[9] L. Mo, A. Kritikakou, and O. Sentieyss, "Approximation-aware task deployment on asymmetric multicore processors," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1513–1518.

[10] M. A. Haque, H. Aydin, and D. Zhu, "On reliability management of energy-aware real-time systems through task replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 3, pp. 813–825, 2016.

[11] Y. Xiang and S. Pasricha, "Fault-aware application scheduling in low-power embedded systems with energy harvesting," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2014, p. 32.

[12] K. Guha, D. Saha, and A. Chakrabarti, "Self aware soc security to counteract delay inducing hardware trojans at runtime," in *30th International Conference on VLSI Design and 16th International Conference on Embedded Systems, VLSID 2017, Hyderabad, India, January 7-11, 2017*. IEEE Computer Society, 2017, pp. 417–422.

[13] K. Guha, A. Majumder, D. Saha, and A. Chakrabarti, "Dynamic power-aware scheduling of real-time tasks for fpga-based cyber physical systems against power draining hardware trojan attacks," *J. Supercomput.*, vol. 76, no. 11, pp. 8972–9009, 2020.

[14] R. Bonamy, S. Bilavarn, and F. Muller, "An energy-aware scheduler for dynamically reconfigurable multi-core systems," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. IEEE, 2015, pp. 1–6.

[15] K. Vallerio, "Task graphs for free (tgff v3. 0)," *Official version released in April*, vol. 15, 2008.