A Simulated Annealing based Energy Efficient Task Scheduling Algorithm for Multi-core Processors

Pratik S. and Abhishek Mishra
Department of Computer Science & Information Systems, Birla Institute of Technology and Science Pilani,

Keywords: Multi-core Processors, Randomized Algorithm, Scheduling, Simulated Annealing, Task Allocation.

Abstract: In this paper we propose a *Simulated Annealing* (SA) based energy-efficient task scheduling algorithm for multi-core processors, the *Simulated Annealing Energy Efficient Task Scheduling Algorithm* (SAEETSA), and compare it with another algorithm, the *Energy Efficient Task Scheduling Algorithm* (EETSA). Our results show that for dual-core processors the SAEETSA algorithm is taking up to 16.78% less energy as compared to the EETSA algorithm, and for tri-core processors, the SAEETSA algorithm is taking up to 26.97% less energy as compared to the EETSA algorithm.

1 INTRODUCTION

SA is a local search-based meta-heuristic method to navigate across large search spaces in optimization problems (Gaspero, 2003), (Kleinberg and Tardos, 2006). It uses a waning probability function to explore paths that are not locally optimum but may lead to future global optimum solutions. Other approaches like steepest hill-climbing, while moving between adjacent points in the solution space, will only move if the next point corresponds to a better solution. This is disadvantageous, because possibly the global optimum or even a better optimum solution may be missed. After all, it can be reached by a path in which a worse solution lies in between the current point and the better optimum. SA helps to solve this disadvantage by not going to a strictly better solution locally but, by a small probability moves to a slightly worse solution. This probability depends on the current 'temperature' which decreases with each iteration by a factor of β. It also depends on the inverse exponential way of "how much worse" a solution is compared to the current optimum discovered so far. Our objective in this paper is to show that SA can significantly improve a simple randomized algorithm.

Multi-core processors have different cores which can run at different speeds. With the increase in core speeds, power consumption also increases. The energy-efficient task scheduling problem is to complete the tasks within a given deadline to minimize energy consumption.

The rest of the paper is organized as follows: section 2 gives an overview of some energy-efficient task scheduling algorithms. Section 3 describes the SAEETSA algorithm based on (Mishra and Tripathi, 2014b). In sections 4 and 5, we compare the SAEETSA algorithm with the EETSA algorithm for dual-core and tri-core processors respectively. We conclude in section 6.

2 LITERATURE OVERVIEW

Various task scheduling algorithms are proposed in the literature for minimizing the parallel execution time of tasks. (P. K. Mishra and Tripathi, 2012b) compare various task scheduling algorithms. (Mishra and Mishra, 2016) propose a local search-based task scheduling algorithm that creates a hybrid of two task scheduling algorithms and gives better parallel execution time than the individual task scheduling algorithms. (P. K. Mishra and Mishra, 2010) have proposed a scheduling heuristic using computation time of tasks and the communication delay between dependent tasks. (P. K. Mishra and Mishra, 2011) have proposed a dynamic priority task scheduling algorithm. (P. K. Mishra and Tripathi, 2012a) have used randomization to improve the previous algorithm (P. K. Mishra and Mishra, 2010). (Mishra and Tripathi, 2011b), (Mishra and Tripathi, 2010) have proposed a task scheduling algorithm that uses cluster-dependent priority function and gives less par-

S., P. and Mishra, A.
A Simulated Annealing based Energy Efficient Task Scheduling Algorithm for Multi-core Processors.
DOI: 10.5220/0010625900003063
In *Proceedings of the 13th International Joint Conference on Computational Intelligence (IJCCI 2021)*, pages 81-87
ISBN: 978-989-758-534-0; ISSN: 2184-2825

allel execution time as compared to some well-known task scheduling algorithms. (A. Mishra and Mishra, 2019) have done performance evaluation of SA-based task scheduling algorithms by varying its various parameters.

Various task scheduling algorithms are proposed in the literature for minimizing the energy consumption of tasks. (Mishra and Tripathi, 2014a) have proposed a pseudo-polynomial time energy-efficient task scheduling algorithm for a given computation load and a given deadline on a multi-core processor with software-controlled dynamic voltage scaling (DVS). (Mishra and Tripathi, 2014b) have proposed a Monte-Carlo energy-efficient task scheduling algorithm for a given set of independent tasks and a given common deadline on a multi-core processor with software-controlled DVS. (Mishra and Tripathi, 2011a) have formulated a problem of energy efficient task scheduling of send-receive task graphs on multiple multi-core processors with software controlled DVS. (Mishra and Trivedi, 2020) have done benchmark comparison of the contention aware nature-inspired meta-heuristic task scheduling algorithms. (S. K. Biswas and Muhuri, 2018) have proposed energy-efficient task scheduling algorithms in multiprocessor systems using archived multi-objective SA. (Y. Yun and Kim, 2019) have proposed an adaptive genetic algorithm for energy-efficient task scheduling on asymmetric multiprocessor system-on-chip.

SA is successfully applied to solve scheduling problems. A recent example of SA based scheduling algorithm is provided by (K. Haridass and McDonald, 2014). They solve the problem of scheduling a log transport system using SA.

3 THE SAEETSA ALGORITHM

Before describing the SAEETSA algorithm, first we describe the *Energy Efficient Task Scheduling Problem* (EETSP) and the EETSA algorithm (Mishra and Tripathi, 2014b). We use the same notation of (Mishra and Tripathi, 2014b). t independent tasks are given. The task set is $\{T_j \mid j \in [1..t]\}$. Task T_j has computation requirement of c_j ($j \in [1..t], c_j \in \mathbb{N}$), given by the set $C = \{c_j \mid j \in [1..t]\}$. A common deadline of D is given ($D \in \mathbb{N}$). The task set $\{T_j \mid j \in [1..t]\}$ has to be scheduled on a multi-core processor with p cores and q possible discrete speeds given by the set $Q = \{s_i \mid i \in [1..q], s_i \in \mathbb{N}\}$. We define the speed profile of the processor, $F = (f_i)_{i=1}^{q}$, to be the vector such that f_i is the number of time slots in $[0,D]$ in which the cores are running at the speed of s_i. We define the sleep profile of the processor, $H = (h_{ki})_{(k,i)\in[1..p]\times[1..q]}$, to be the matrix such that h_{ki} is the number of time slots for s_i on the k'th core that are sleeping. We define the binary partition matrix of the task set, $G = (G_{kj})_{(k,j)\in[1..p]\times[1..t]}$, to be the matrix such that $g_{kj} = 1$ if and only if the the task T_j is allocated on the k'th core (otherwise it is 0). The multi-core processor has software controlled per-chip DVS. By per-chip DVS we mean that we can only change the speed of all cores simultaneously (not individually). If the speed of the multi-core processor is s_i, then either a core will run at the speed of s_i, or it will be in a sleep state (speed $= 0$). Power consumption of a core running at a speed of s is given by $P(s) = \alpha s^3$, where α is a constant. Energy consumption by a core running at a speed of s during τ units of time is given by $E(s,\tau) = P(s)\tau$. A core running at a speed of s can perform a maximum of $c = s\tau$ units of computation during τ units of time. We assume that the DVS software can change the speed of the multi-core processor at periodic checkpoints given by $\delta\tau = 1$ (at the start of each unit time interval). The EETSP problem is to find an allocation of tasks to the cores, speed of the multi-core processor at each unit of time in $[0,D]$, and for each core the unit time intervals in which it is sleeping, so that the energy consumption of the tasks is minimized, and also each task is finished within the given deadline of D. (Mishra and Tripathi, 2014b) have formulated an integer linear program (ILP) for the EETSP problem (equations $(6)-(8)$, and $(11)-(16)$ in (Mishra and Tripathi, 2014b)).

Now we describe the EETSA algorithm for solving the EETSP problem as given in (Mishra and Tripathi, 2014b). The EETSA algorithm is a Monte-Carlo algorithm. It takes as input (m,n,t,C,Q,D), where m and n are integer parameters. The EETSA algorithm is divided into three parts. The first part is lined 01 to 27 in (Mishra and Tripathi, 2014b) in which we make m attempts using randomization to find a feasible solution of the ILP. The second part is lined 28 to 44 in (Mishra and Tripathi, 2014b) in which we make n attempts using randomization to reduce the energy consumption for the case when the first part of the algorithm succeeds in finding a feasible solution of the ILP (otherwise we return "NO SOLUTION" which may be a wrong output). This energy reduction assumes that no core is sleeping. The third part is lined 45 to 50 in (Mishra and Tripathi, 2014b) in which we further try to reduce the energy consumption by assuming that some cores can sleep.

Similar to the EETSA algorithm, SAEETSA algorithm also takes (m,n,t,C,Q,D) as input parameters, and it is also divided into three parts. First part of the SAEETSA algorithm is lines 1 to 4. It

Algorithm 1: The SIMULATED-ANNEALING-ENERGY-EFFICIENT-TASK-SCHEDULING-ALGORITHM.

```
01: procedure SAEETSA(m, n, t, C, Q, D)
02:     Use part 1 of the EETSA algorithm (lines 01 to 25 of the EETSA algorithm in (Mishra and Tripathi, 2014b)) to find a feasible solution (F,G,H) of the ILP by making m randomization steps.
03:     if step 01 fails to find a feasible solution of ILP then
04:         return NO SOLUTION
05:     else
06:         b ← ⌈t/n⌉
07:         E' ← α Σ_{k=1}^{p} Σ_{i=1}^{q} s_i^3 (f_i − h_ki)
08:         (A,F',G',H') ← (O_{1×p},F,G,H)
09:         for j ← 1, t do
10:             for k ← 1,p do
11:                 a_k ← a_k + c_j g_kj
12:             end for
13:         end for
14:         for r ← 1,n do
15:             F'' ← F
16:             for v ← 1,b do
17:                 RANDOM-SELECT (1,D,F)
18:                 M ← Σ_{i=1}^{q} s_i f_i
19:                 r ← RANDOM (0,1)
20:                 E ← α Σ_{k=1}^{p} Σ_{i=1}^{q} s_i^3 (f_i − h_ki)
21:                 if ∧_{k=1}^{p} (a_k ≤ M) then
22:                     if (E ≤ E') ∨ (r ≤ e^{−(E−E')/T}) then   ▷ T is the SA temperature
23:                         (E',F',G',H') ← (E,F,G,H)
24:                     else
25:                         F ← F''
26:                     end if
27:                 end if
28:             end for
29:             T ← Tβ   ▷ β is the SA cooling schedule
30:         end for
31:         L ← O_{1×q}
32:         for k ← 1,p do
33:             E_k ← α Σ_{i=1}^{q} s_i^3 (f_i − h_ki)
34:         end for
35:         for all L ∈ {L | (L ≥ O_{1×q}) ∧ (L ≤ F)} do
36:             for k ← 1,p do
37:                 if Σ_{i=1}^{q} s_i (f_i − l_i) ≥ a_k then
38:                     E'_k ← α Σ_{i=1}^{q} s_i^3 (f_i − h_ki)
39:                     if (E'_k ≤ E_k) then
40:                         (E_k,H_k) ← (E'_k,L)
41:                     end if
42:                 end if
43:             end for
44:         end for
45:         E' ← Σ_{k=1}^{p} E_k
46:         (F',G') ← (F,G)
47:         H' ← (H_k)_{k=1}^{p}
48:         return (E',F',G',H')
49:     end if
50: end procedure
```

is identical to the first part of the EETSA algorithm (Mishra and Tripathi, 2014b). The difference between the two algorithms is in the second part (steps 5 to 30) and the third part (steps 31 to 50). In the second part of the SAEETSA algorithm (reducing the energy consumption from feasible solution of the first part assuming that no cores can sleep), steps 5 to 18 of the SAEETSA algorithm is identical to steps 28 to 39 of the EETSA algorithm (Mishra and Tripathi, 2014b). The difference between the two algorithms is in steps 19 to 30 of the SAEETSA algorithm, and steps 35 to 44 of the EETSA algorithm (Mishra and Tripathi, 2014b). In steps 40 to 44 of the EETSA algorithm (Mishra and Tripathi, 2014b), we follow the greedy approach. We make n randomization steps in which we modify F and try to get a feasible solution. If the energy is minimized, then this F becomes the current F, and we repeat the same process for total n number of times. In steps 14 to 30 of the SAEETSA algorithm, we make n attempts for decreasing the energy consumption. In each attempt, we modify F in step 17 in hope of finding a lower energy value using local neighborhood search using the function RANDOM-SELECT $(1,D,F)$. The function RANDOM-SELECT $(1,D,F)$ first randomly selects two time slots in $[1,D]$. It then applies the neighborhood function $n(F)$ to the selected time slots. The neighborhood $n(F)$ of F is defined as follows: $n(F) = \{F' \mid \exists(j,k),$ such that $\forall i \neq k,j, f_i = f'_i$ and $f_j + f_k = f'_j + f'_k\}$. For example, $(1,2,3,4) \in n(1,2,4,3)$, but $(1,2,3,4) \notin n(1,2,3,5)$. If the energy value is lower, then the modified F becomes the current F (steps 22 and 23), and we repeat the process. However, if we get a higher energy value, then with probability $e^{-(E-E')/T}$, the modified F becomes the current F (steps 19, 22 and 23) in hope of finding better solution some time later (the SA approach). Here E is the current energy consumption, and E' is the lowest energy consumption found so far. T is the temperature. In each iteration we update the temperature by a constant factor β (step 29 is the cooling schedule). We may initialize $T = 15$ and $\beta = 0.9$ which experimentally give the best results.

The third part of the SAEETSA algorithm (steps 31 to 49: further reducing the energy consumption by assuming that some cores can independently sleep) is an optimization of the third part of the EETSA algorithm (steps 45 to 50 in (Mishra and Tripathi, 2014b)). The third part of the EETSA algorithm has a **for all** loop in step 45 in (Mishra and Tripathi, 2014b). It involves energy evaluation corresponding to all possible values of a $p \times q$ matrix H, contributing a factor of D^{pq} in the complexity expression. This is per-chip optimization. Equivalently, we have done per-core opti-

mization in the third part of the SAEETSA algorithm (the **for all** loop in step 35). This reduces the complexity factor of D^{pq} down to pD^q. We have used all possible values of the per-core sleep matrix L in energy calculation (step 35). We have done energy optimization independently for each core (the **for** loop of step 36). Now we will prove the optimality of the third part of the SAEETSA algorithm. Let E_k be defined as the energy consumed by core k (step 33). The total energy consumed E is given by the equation in step 45 ($E = \sum_{k=1}^{p} E_k$).

After having designated F for an iteration of EETSA, it is easy to prove that E is optimal (i.e. least) across assignments of H (Mishra and Tripathi, 2014b) if and only if E_k is optimal for each k. Note that E_k can be optimized separately for each k since E_k and E_l are independent, $k \neq l$, since they use different rows of the sleep matrix H in their evaluation expression.

Consider the 'only if' part of the above claim: let's say to the contrary that E is least but E_k is not least for some k. Since E_k can be replaced by optimal E'_k. This corresponds to a change in some of the values of row k of the sleep matrix H. Change in this row does not affect other rows of H and hence other values of E_l don't change. This means that by the equation in step 45, the new energy computed E' is less than E. But this contradicts our assumption that E is optimal. Hence we have proved that E_k is optimal for each $1 \leq k \leq p$ if E is optimal.

Next, we prove the 'if' part which is trivial: by the equation in step 45, we have that E is least when E_k is least for each $1 \leq k \leq p$.

The above result helps us to reduce the complexity of the SAEETSA algorithm by pruning the search space of the sleep matrix optimization part. Note that we don't have to search through all values of H i.e. D^{pq} values, we may simply find optimal for each core wherein we have to search through all possible values of a single row of H only (the L matrix). This reduces the complexity factor down to pD^q, which is a huge help.

The loop at step 35 runs D^q times. The inner loop (step 36) runs p times. Then the if part at step 37 requires $O(q)$ time. Lines 45 to 48 takes $O(pq + pt)$ time. This gives the time complexity of the third part of SAEETSA as $O(p(q+t+qD^q))$. Complexity of the first and second parts of the SAEETSA algorithm is identical to the complexity of the first and second parts of the EETSA algorithm respectively (Mishra and Tripathi, 2014b). Therefore, the overall time complexity of the SAEETSA comes out to be $O(t(mp+q+\log t)+pn(t+q)+p(q+t+qD^q))$.

4 EXPERIMENTAL RESULTS FOR DUAL-CORE PROCESSORS

Figures 1 and 2 show the experimental results for SAEETSA for $p = 2$ and $q = 3, n = 10$ with 5 node task sets and 10 node task sets respectively. The common deadline for the two cases was fixed at 30 and 50 respectively. SAEETSA algorithm is showing 16.78% and 8.48% average energy optimization as compared to the EETSA algorithm respectively.

5 EXPERIMENTAL RESULTS FOR TRI-CORE PROCESSORS

Figures 3 and 4 show the results for SAEETSA for $p = 3$ and $q = 3, n = 10$ with 5 node task sets and 10 node task sets respectively. The common deadline for the two cases was fixed at 30 and 50 respectively. SAEETSA algorithm is showing 26.97% and 8.82% average energy optimization as compared to the EETSA algorithm respectively.

6 CONCLUSION

The SAEETSA algorithm was an improvement of the EETSA algorithm of (Mishra and Tripathi, 2014b) using SA with reduced time complexity. Our experimental results are showing up to 26.97% improvement over the EETSA algorithm. For future work, we can try to formulate the energy-efficient task scheduling problem of independent tasks with a common deadline on multiple multi-core processors with task migration overheads.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their constructive comments in improving the quality of this paper.

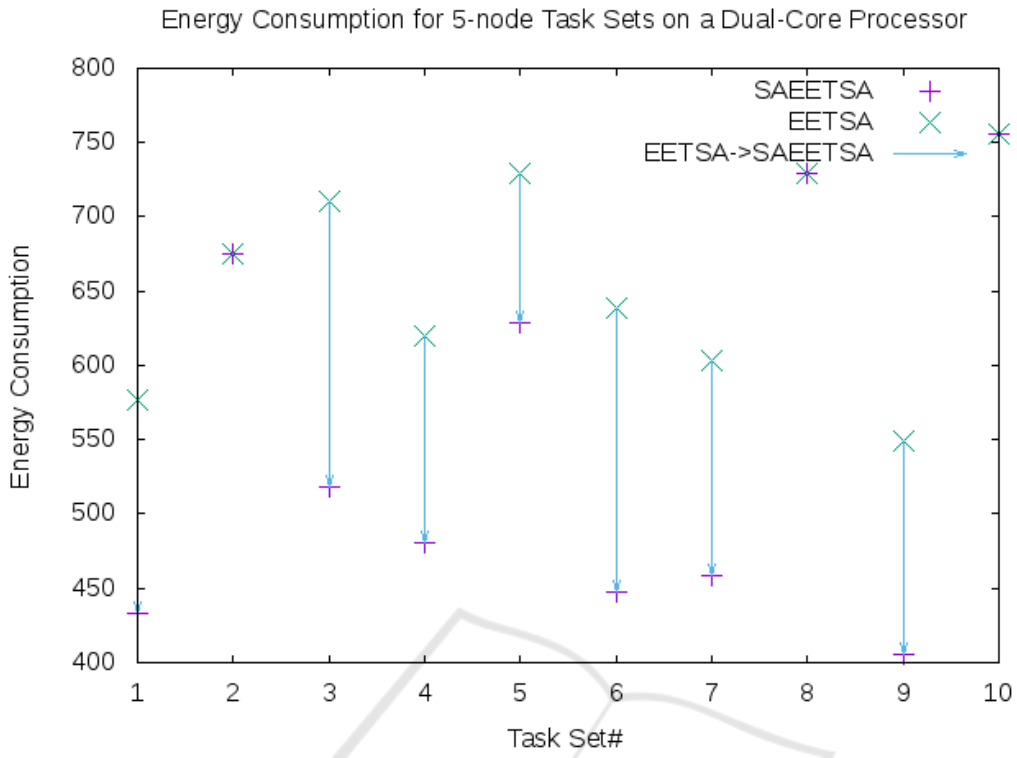

Figure 1: Energy consumption for 5 node task sets on a dual-core processor. The SAEETSA algorithm is consuming 16.78% less energy as compared to the EETSA algorithm.

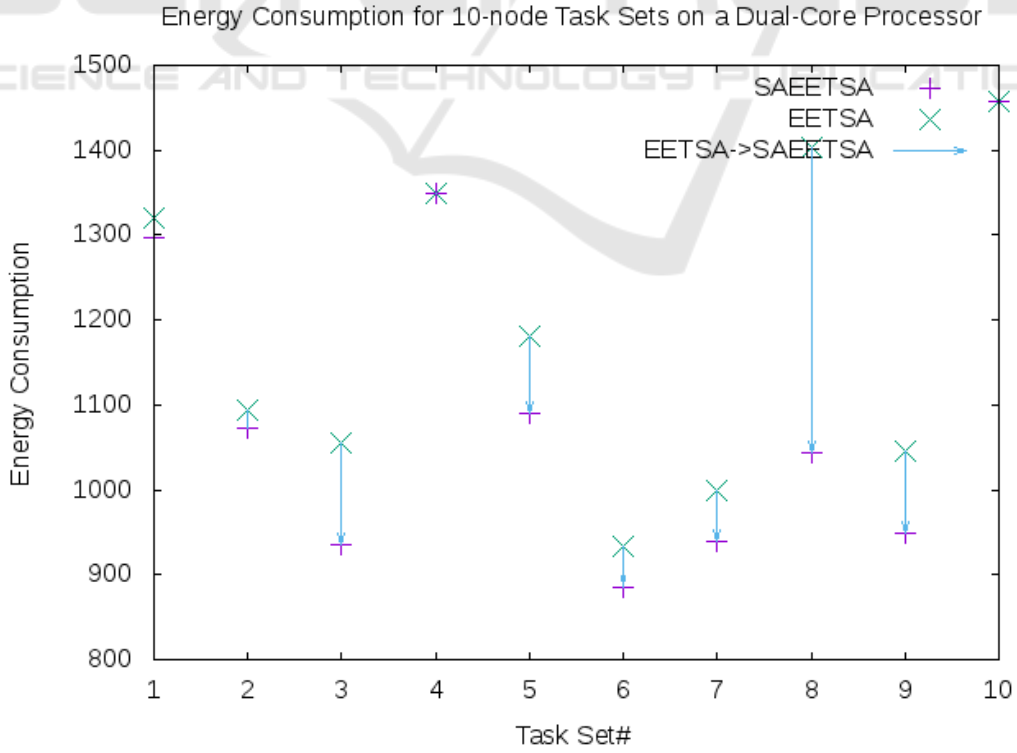

Figure 2: Energy consumption for 10 node task sets on a dual-core processor. The SAEETSA algorithm is consuming 8.48% less energy as compared to the EETSA algorithm.

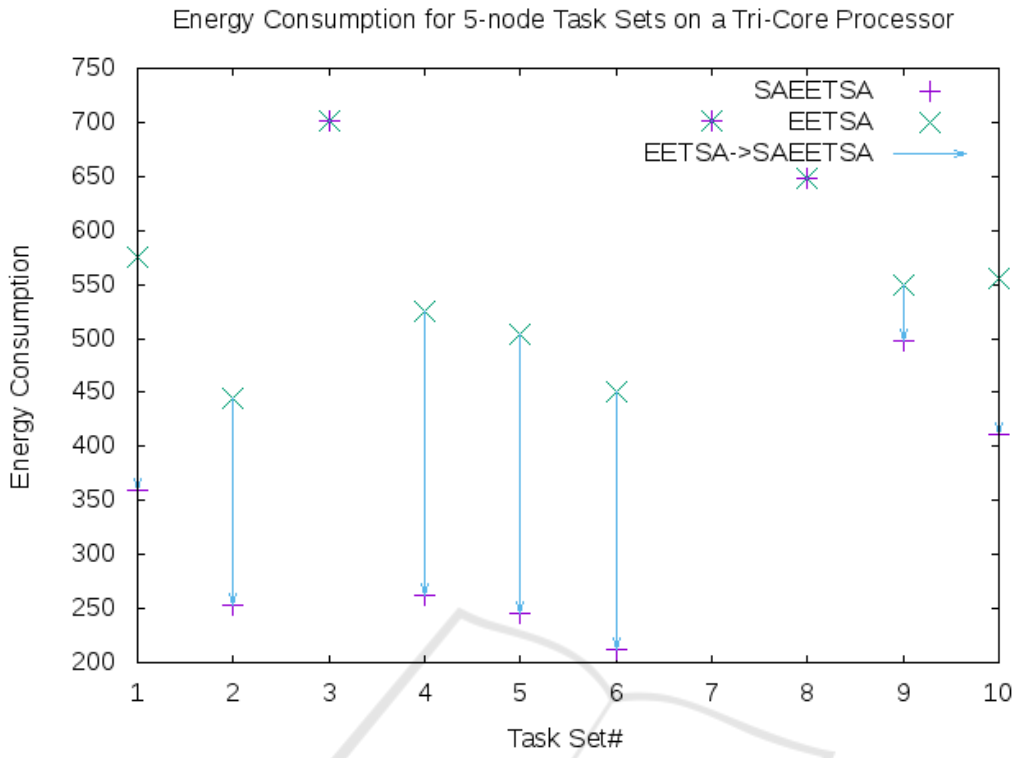

Figure 3: Energy consumption for 5 node task sets on a tri-core processor. The SAEETSA algorithm is consuming 26.97% less energy as compared to the EETSA algorithm.

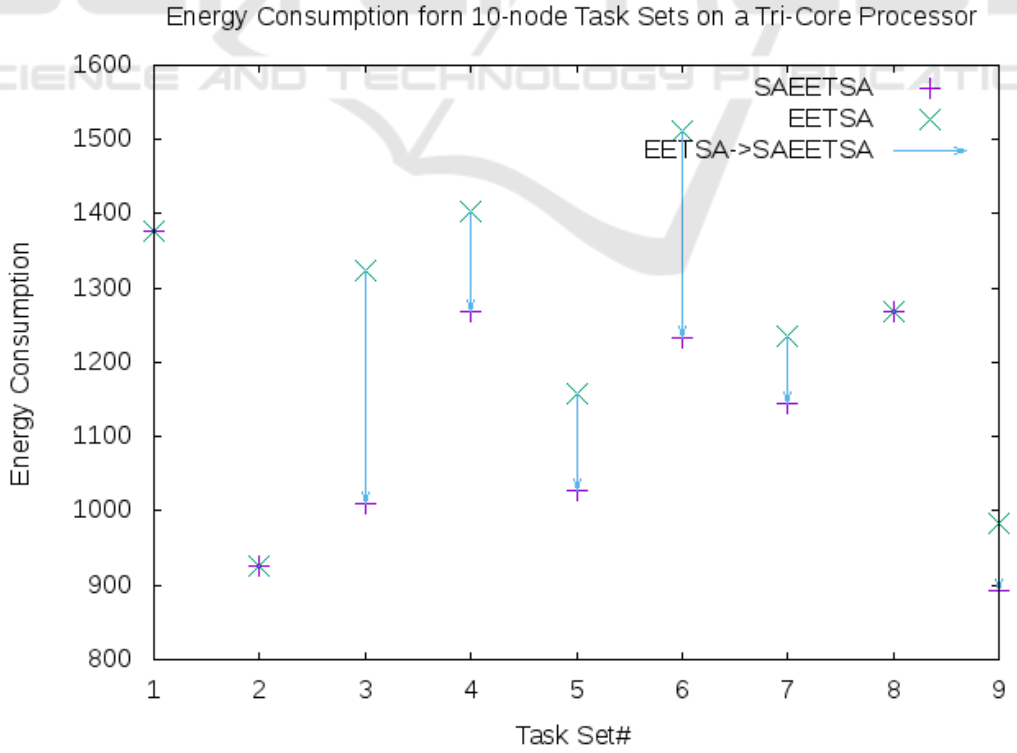

Figure 4: Energy consumption for 10 node task sets on a tri-core processor. The SAEETSA algorithm is consuming 8.82% less energy as compared to the EETSA algorithm.

REFERENCES

A. Mishra, K. S. M. and Mishra, P. K. (2019). Performance evaluation of simulated annealing-based task scheduling algorithms. In *D. Goyal et al. (eds.), Information Management and Machine Intelligence, Algorithms for Intelligent Systems*. Springer.

Gaspero, L. D. (2003). *Local Search Techniques for Scheduling Problems: Algorithms and Software Tools*. PhD thesis, Universita degli Study di Udine.

K. Haridass, J. Valenjuela, A. D. Y. and McDonald, T. (2014). Scheduling a log transport system using simulated annealing. *Information Sciences*, 264:302–316.

Kleinberg, J. and Tardos, E. (2006). *Algorithm Design*. Pearson.

Mishra, A. and Mishra, P. K. (2016). A randomized scheduling algorithm for multiprocessor environments using local search. *Parallel Processing Letters*, 26(1):1650002.

Mishra, A. and Tripathi, A. K. (2010). An extention of edge zeroing heuristic for scheduling precedence constrained task graphs on parallel systems using cluster dependent priority scheme. In *International Conference on Computer & Communication Technology (ICCCT'10)*. IEEE.

Mishra, A. and Tripathi, A. K. (2011a). Energy efficient task scheduling of send-receive task graphs on distributed multi-core processors with software controlled dynamic voltage scaling. *International Journal of Computer Science & Information Technology*, 3(2):204–210.

Mishra, A. and Tripathi, A. K. (2011b). An extension of edge zeroing heuristic for scheduling precedence constrained task graphs on parallel systems using cluster dependent priority scheme. *Journal of Information and Computing Science*, 6(2):83–96.

Mishra, A. and Tripathi, A. K. (2014a). Energy efficient voltage scheduling for multi-core processors with software controlled dynamic voltage scaling. *Applied Mathematical Modelling*, 38:3456–3466.

Mishra, A. and Tripathi, A. K. (2014b). A monte carlo algorithm for real time task scheduling on multi-core processors with software controlled dynamic voltage scaling. *Applied Mathematical Modelling*, 38:1929–1947.

Mishra, A. and Trivedi, P. (2020). Benchmarking the contention aware nature inspired metaheuristic task scheduling algorithms. *Cluster Computing*, 23(2):537–553.

P. K. Mishra, K. S. Mishra, A. M. and Tripathi, A. K. (2012a). A randomized scheduling algorithm for multiprocessor environments. *Parallel Processing Letters*, 22(4):1250015.

P. K. Mishra, K. S. M. and Mishra, A. (2010). A clustering heuristic for multiprocessor environments using computation and communication loads of modules. *International Journal of Computer Science & Information Technology*, 2(5):170–182.

P. K. Mishra, K. S. M. and Mishra, A. (2011). A clustering algorithm for multiprocessor environments using dynamic priorities of modules. *Annales Mathematicae et Informaticae*, 38:99–110.

P. K. Mishra, A. Mishra, K. S. M. and Tripathi, A. K. (2012b). Benchmarking the clustering algorithms for multiprocessor environments using dynamic priority of modules. *Applied Mathematical Modelling*, 36:6243–6263.

S. K. Biswas, R. J. and Muhuri, P. K. (2018). Energy efficient scheduling in multiprocessor systems using archived multi-objective simulated annealing. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE.

Y. Yun, E. J. H. and Kim, Y. H. (2019). Adaptive genetic algorithm for energy-efficient task scheduling on asymmetric multiprocessor system-on-chip. *Microprocessors and Microsystems*, 66:19–30.