Abstract—The massive demand for running parallel applications on distributed systems has led to an upsurge in the system power consumption. These systems often consist of thousands or millions of cores, storage disks, interconnection devices and other power-hungry components. To address this power consumption problem, we propose two energy-aware scheduling algorithms, namely, Energy-Efficiency with Duplication (EED) and Energy-Efficiency with Non Duplication (EEND). Both algorithms, in contrast to their counterparts in the literature, strive to make a balance across the energy consumption, the schedule length, and the number of processors used. Synthetic benchmarks and real-world applications are used to evaluate the performance of our algorithms. A comparison is made with the Traditional Duplication-based Scheduling algorithm (TDS), the Energy Aware Duplication algorithm (EAD), and the Performance-Energy Balanced Duplication algorithm (PEBD). We have also devised a novel objective function that aids in assessing the performance of different algorithms in terms of various performance metrics. Extensive experimental results conducted as a part of this work show that the performance of our algorithms is promising in terms of energy consumption, processing elements used and the schedule length for communication intensive parallel applications.

I. INTRODUCTION

Distributed systems have emerged as the primary infrastructure for scientific and commercial parallel applications. As a result, there exists an urgent need to address the power consumption problem of these systems [1, 2]. In [3] it is reported that the energy consumption of data centers will quadruple by 2020. It is also known that a petaflop system with around 12,000 nodes sustains hardware failures every twenty-four hours according to Arrhenius Law [2].

Even though the energy consumption of CPUs has gained traction in the High Performance Computing (HPC) community, power consumption induced by the interconnection technology remains an open problem. For example, in a Mellanox server blade, the router and links are estimated to dissipate 15W out of the total budget of 40W. The processor has also been allocated the same power budget of 15W [4]. Also, the integrated router and links of the Alpha 21364 processor [5] consume 23W, where 58% of the power is consumed in the link circuitry. Therefore, the urge to address the power consumption problem of the link circuitry becomes crucial for communication-intensive parallel applications.

A scheduling strategy is one that partitions a parallel program represented as a Directed Acyclic Graph (DAG) into sets of tasks (clusters). It thereby assigns each of these clusters onto a distinct processor. Scheduling strategies in the literature differ based on whether task duplication is allowed or not [7]. In task scheduling without duplication, tasks are only allowed to execute once [8]. For duplication based scheduling, a task may have several copies, each belonging to a distinct cluster [7]. In general, the performance of duplication based scheduling approaches is superior to non-duplication based ones in terms of the overall completion time (makespan/schedule length) [8]. However, this is usually achieved at the expense of a higher space complexity and CPU energy consumption [9].

Until recently, scheduling strategies have only focused on minimizing the overall completion time and the algorithm complexity. Due to the importance of energy consumption associated with executing jobs on processors of a distributed system, several techniques including energy-aware scheduling, Dynamic Voltage and Frequency Scaling (DVFS), and memory optimizations have been investigated and developed by researchers [10, 11]. However, these techniques focus only on energy conservation within CPUs, completely ignoring the energy consumption of the link circuitry [11].

In this paper, we propose two energy-aware scheduling heuristics EED, and EEND. EED duplicates tasks on the critical path for every sink task in a recursive manner with the objective of minimizing both the schedule length and the number of processors used. EED merges those resultant clusters with subset relationships. Clearly, this strategy leads to energy savings in both the CPUs and the interconnection devices. Finally, more task replicas are eliminated from the clusters if this can lead to minimizing the overall energy consumption subject to the schedule length constraints. On the other hand, EEND makes energy savings by omitting all task replicas belonging to distinct processors. As this attempt can lead to a severe degradation of the overall schedule length, cluster selection for eliminating task replicas must be

Energy-Aware Heuristics for Scheduling Parallel Applications on High Performance Computing Platforms

Ahmed Ebaid, Sanguthevar Rajasekaran, Reda Ammar
Computer Science & Engineering Department
University of Connecticut
Storrs, Connecticut 06269
Email: {ahmed.ebaid, rajasek, reda} @ engr.uconn.edu

Rasha Ebaid
Statistics Department
Helwan University
Cairo, Egypt 12311
Email: rebaid@hotmail.com

2013 International Conference on Parallel and Distributed Systems

1521-9097/13 $31.00 © 2013 IEEE
DOI 10.1109/ICPADS.2013.54
conducted carefully to allow for a minimal increase in the overall schedule length. In this paper, we also propose a novel objective function that aids in assessing the performance of various algorithms.

The remainder of this paper is organized as follows. Section II presents some related work. The proposed algorithms are presented in section III, followed by an illustrative example. In section IV, performance results are summarized. Finally, Section V concludes the paper.

II. Related Work

Running parallel applications on distributed systems typically involves intense local computations as well as communications. Efficient task scheduling algorithms are called for in this context. Task scheduling heuristics are categorized according to whether task duplication is allowed or not [11]. Two main approaches can be found in the literature for non-duplication based scheduling, namely, list scheduling and cluster-based scheduling. In list scheduling [12], every task is assigned a priority. Then, tasks are assigned to the set of available processors in a descending order of their assigned priorities. In cluster-based scheduling, tasks involved in communication intensive dependencies are assigned to the same processor [13-16]. A major drawback of the former algorithms resides in the quality of the schedule lengths produced as compared to duplication based ones [8, 9]. Duplication based scheduling [9, 17-21], in contrast, considers duplicating a given task among several processors if this can lead to shortening the overall completion time. However, this improvement is usually achieved at the expense of higher energy consumption due to the redundant execution of jobs.

The above algorithms ignore energy consumptions. EAD [11] is an energy-aware duplication algorithm that focuses on cutting down task replicas to conserve energy. However, this is usually achieved at the expense of increasing the overall completion time and the number of processors used. PEBD in [11], gives a higher weight to minimizing the overall completion time at the expense of more energy conservation. EPBTDSC in [22], tries to achieve a balance between completion time and energy savings by comparing the ratio of energy savings and the schedule length against a threshold value prior to adding a task to a given cluster. Current trends lie in implementing algorithms that could employ the power scalability option of CPUs to conserve energy during the application run time. Examples include Dynamic Voltage Scaling (DVS) [23, 24], Dynamic Frequency Scaling (DFS) [25], and Dynamic Voltage and Frequency Scaling (DVFS) [23, 26, 27]. According to [11], the above mentioned algorithms are perfect candidates for computation intensive parallel applications.

In this paper, experimental results are conducted based on only a single real world application, a single processor type and a single interconnects technology due to the page limitation of this paper. Further results are to be submitted to the journal version.

III. The Proposed Algorithms

A. Model & Notations

In this paper, we assume that a parallel program is given in the form of a DAG defined by the tuple \((V, E, w, c)\) where \(V\) is the set of tasks and \(E\) is the set of edges. \(w(v_i)\) represents the execution time needed by task \(v_i\), \(c(v_i, v_j)\) represents the time required to transmit a message from \(v_i\) to \(v_j\) if both tasks reside on different processors and is set to zero otherwise. The directed edges in \(E\) represent the precedence constraints. In particular, the presence of the edge \(e(v_i, v_j)\) implies that the task \(v_j\) can only start execution after task \(v_i\) completes execution. An entry task is defined as a task with no ascendants, whereas a task with no descendants is referred to as a sink task. \(iparent(v_i)\) represents the set of ancestor tasks for task \(v_i\). A topological order is an ordering of the DAG vertices such that a given task will only start execution after it fulfills all of its precedence requirements. A critical path is defined as the set of tasks and edges, forming a path from an entry task to the designated task, for which the sum of execution times and communication delays is maximum [28]. The earliest start time \(est(v_i)\) is the earliest time that task \(v_i\) can start execution. For a join task \(v_i\), its earliest start time \(est(v_i)\) can be expressed as:

\[
est(v_i) = \begin{cases} est(v_j) + w(v_j) & \text{if } (v_j, v_i) \in P_m \\ est(v_j) + w(v_j) + c(v_j, v_i) & \text{if } (v_j, v_i) \notin P_m \end{cases}
\]

(1)

where \(v_j \in iparent(v_i)\), and \(P_m\) refers to the processor labelled \(m\). The earliest completion time \(ect(v_i)\) denotes the earliest time task \(v_i\) finishes execution, such that \(ect(v_i) = est(v_i) + w(v_i)\). Moreover, task \(v_i\)'s Critical Immediate Parent \(CIP(v_i)\) is the ancestor task with the largest earliest completion time plus communication delay to that join task \(v_i\), such that:

\[
v_j = CIP(v_i) \mid \text{ect}(v_j) + c(v_j, v_i) \geq \text{ect}(v_k) + c(v_k, v_i) \forall k \text{s.t. } \{v_j, v_k\} \in iparent(v_i) \text{ and } k \neq j
\]

(2)

If multiple tasks satisfy the above constraint, we arbitrarily select one of them. A scheduling \(\beta\) for a given DAG is defined as the partitioning of DAG tasks into clusters, where \(cluster(v_i)\) is the assemblage of several tasks into the same set including task \(v_i\), where \(v_i\) is the task with the maximum \(ect\) for that given cluster. We define \(tasksLeft\) as the set of tasks that do not belong to \(\beta\) during the clustering process. The overall completion time (schedule length, makespan) denoted by \(\alpha\) can be calculated as follows:

\[
\alpha = \max(ect(v_a))
\]

(3)

where \(v_a\) is the set of sink tasks. A task allocation matrix \(X\) is an \(n \times m\) binary matrix that reflects the mapping of \(n\) tasks onto \(m\) processors. Finally, we define interCommunication as the total time needed to transmit messages between clusters of \(\beta\) and is expressed as:
interCommunication = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \sum_{a=1}^{m} \sum_{b=1, b \neq a}^{m} x_{ia} \cdot x_{jb} \cdot c(v_i, v_j) \tag{4}

As in [7, 8, 10, 11, 17, 21, 28-30], we assume that the underlying target architecture is homogeneous, the network is fully connected, and the number of processing elements available is unlimited.

B. The Energy Consumption Model

In this subsection, we discuss the energy consumption model presented in [31] as being the primary tool for evaluating the energy consumptions of our algorithms. In this model, the authors focus on evaluating the energy consumed by the CPUs as well as the interconnections. The total energy consumption in a homogeneous distributed system can be modeled as:

\[ E = EN + EL \tag{5} \]

where \( EN \) and \( EL \) are the total energy consumptions of the CPUs and interconnection technologies, respectively. For a parallel application with a task set \([V]\) and allocation matrix \(X\), the active CPU energy consumption is modeled as:

\[ EN_{\text{active}} = PN_{\text{active}} \cdot \sum_{i=1}^{n} w(v_i) \tag{6} \]

mell where \( PN_{\text{active}} \) is the CPU energy consumption rate of the active processors, and \( w(v_i) \) represents the time needed by task \( v_i \) to complete its execution.

As highlighted in [11], the CPU energy consumption model is compatible with DVFS technology where processors may have several voltages and frequencies. This can be handled by replacing \( PN_{\text{active}} \) with \( PN_{\text{best-fit}} \). The total energy dissipated by the interconnects during message transmission during active times can be expressed as:

\[ EL_{\text{active}} = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} \sum_{a=1}^{m} \sum_{b=1, b \neq a}^{m} x_{ia} \cdot x_{jb} \cdot PL_{\text{active}} \cdot c(v_i, v_j) \tag{7} \]

where \( PL_{\text{active}} \) is the active energy consumption of the communication links and \( c(v_i, v_j) \) is the time required to transmit a message between tasks \( v_i \) and \( v_j \) that reside in different processors.

C. Energy-Aware Duplication Heuristics

In this subsection, we discuss two energy-aware scheduling algorithms, namely (EED and EEND), that employ duplication and non-duplication based strategies, respectively. The main objective of these algorithms is to generate schedules with minimal energy consumption subject to the schedule length constraints. It is known that the general problem of scheduling task graphs onto multiprocessors is NP-hard [8]. Like many of the algorithms in the literature, our proposed algorithms are also heuristics in the sense that an optimal scheduling may not happen.

1) Energy Efficiency with Duplication (EED): The main purpose of the EED algorithm is to find an energy-aware scheduling for a given DAG without affecting the overall system performance (i.e., schedule length and processing elements used). Our approach starts by allocating a cluster for every sink task in a manner that allows for the coexistence of a given task in multiple clusters. Each of these clusters consists of the designated sink task along with tasks residing on its critical path. Subsequently, the algorithm repeats in a recursive manner for every sink task until a further improvement to the overall completion time is not viable. This approach is similar to the one we have presented in [20].

There are crucial differences between the approach presented in [20] and the one proposed in this paper. For instance, EED attempts to cut down the energy consumption by merging those clusters with the subset relationship given the schedule length constraint obtained from [20]. Undoubtedly, this set-up minimizes both the CPU and the link circuitry energy consumptions where duplicated tasks and the inter-processor communications between the merged clusters are cancelled out. Moreover, EED eliminates some of the task replicas that do not cause deterioration for both the overall energy consumption and the schedule length. An attempt to eliminate task replicas, while minimizing the CPU energy consumption, could increase the interconnection energy consumption. In this context, we have used the energy consumption model presented in [31] to verify that the energy consumption is not increased in our approach. In order to effectively address the problem, we have used an old and a current value (i.e., \( est(v_i) \), \( CIP(v_i) \), etc.) to compare between the states of the DAG before and after each clustering iteration. Here \( \alpha_{old} \) and \( \alpha \) represent old and current values of schedule lengths, respectively. Our main EED algorithm is described in Fig. 2. Initialization and Energy Conservation procedures are described in Figures 1 and 5.

During the Initialization procedure, we set up and initialize variables needed by the EED algorithm. Consequently, given a DAG, the output values for \( est(v_i)_{old} \), \( CIP(v_i)_{old} \) and \( \alpha_{old} \) are calculated under the assumption that every task runs on a distinct processor. In Fig. 1, we start the initialization procedure by ordering tasks with respect to their topological order. This step makes certain that a given task will only be processed after its ancestor tasks are assigned an \( est(v_i) \) value. The values of \( tasksLeft_{old} \) and \( \beta_{old} \) are calculated as shown in steps 2-3. The value of \( \alpha_{old} \) is calculated using Equation (3) based on the values of \( est(v_i)_{old} \) obtained in step 4. Finally, we sort tasks in an ascending order of their earliest completion time which allow them to start at their earliest start times.

In the EED algorithm, we generate clusters in a recursive manner and thereby assign each of these clusters to a distinct processor. Initial values of \( est(v_i)_{old} \), \( CIP(v_i)_{old} \) and \( \alpha_{old} \) obtained from the Initialization procedure are used as an input. Output values of the earliest completion time \( est(v_i) \), schedule
0: Procedure Initialization
0: INPUT: $G = (V,E,w,c)$
0: OUTPUT: $ect(v_i)_{old}$, $CIP(v_i)_{old}$, $\alpha_{old}$
1: Sort tasks of $G$ in topological order.
2: Set $tasksLeft_{old} = \{v_1,v_2,v_3,...,v_n\};|V|=n$.
3: Let $\beta_{old} = \{\}$.
4: Find $ect(v_i)_{old}$, and $CIP(v_i)_{old}$ for all $v_i \in |V|$, $P_i \in P_m$ and $v_i \in P_i$.
5: Find $\alpha_{old} = \max(ect(v_i)) \mid v_i \in \text{sink tasks}$.
6: Sort tasks of $G$ in ascending order of their $ect$.

Fig. 1. Pseudocode of the Initialization Procedure

length $\alpha$, scheduling $\beta$, interCommunication$_{old}$, and the total energy consumption $E$ are calculated accordingly. In Fig. 2, the algorithm starts by allocating a cluster for every sink task $v_s$. Such a cluster consists of task $v_s$ along with other tasks residing on its critical path as in step 1. We, thereby order tasks within a given cluster in an ascending order of their earliest completion times. Thus, the current values of $\beta$, tasksLeft, $ect(v_i)$, $CIP(v_i)$, and interCommunication are calculated. At each clustering iteration, we compare the current values of $\alpha$ and $\beta$ against the most recent values obtained (i.e., $\alpha_{old}$, $\beta_{old}$). Then, a recursive call is made to the EED algorithm based on the former comparison as well as the state of tasksLeft as in the if conditions steps 7, 14, and 18. A final check is made on the set of tasksLeft for non-emptiness and a cluster is created for each task belonging to this set as shown in the if condition of step 22. Finally, the algorithm terminates by a final call to the Energy conservation procedure.

In the Energy Conservation procedure (Figure 3), we cut down the energy consumption by merging those clusters with subset relationships. Given the input values of $ect(v_i)_{old}$, $\alpha_{old}$, $\beta_{old}$ and interCommunication$_{old}$, the output values of $\alpha$, $\beta$, $ect(v_i)$, and $E$ are calculated. A given cluster is considered redundant if tasks belonging to it are a subset of another cluster. In this case, we update our scheduling $\beta$ to exclude this cluster as depicted in step 1 of Fig. 3. Additionally, given the schedule length constraint, two clusters are merged together if the tasks belonging to one of them form a subset of another excluding their designated tasks (i.e., designated task of $cluster(v_i)$ is $v_i$) as depicted by the if condition of step 6. These steps undoubtedly guarantee a reduction of both the CPU and the interconnection energy consumptions since they eliminate redundant tasks and the inter-processor communications among the merged clusters. Finally, our approach considers further reduction to the CPU energy consumption by eliminating those tasks that are common between two given clusters. We use the energy consumption model presented in [31] to verify that the energy dissipated by the interconnects does not exceed the CPU energy consumptions of the eliminated tasks.

2) Energy Efficiency with Non Duplication (EEND): The basic idea behind our EEND algorithm is to omit every task replica from $\beta$ generated by the EED algorithm. Since the performance of duplication based scheduling algorithms is in general, superior to non-duplication based ones in terms of the overall completion time [8, 9], eliminating task replicas from $\beta$ must be conducted cautiously in order to allow for minimal increase in the schedule length.

0: Algorithm EED
0: INPUT: $G = (V,E,w,c)$, $ect(v_i)_{old}$, $CIP(v_i)_{old}$, $\alpha_{old}$
0: OUTPUT: $ect(v_i)$, $\alpha$, $\beta$, $E$.
1: Find $cluster(v_s) \mid \forall v_s \in v_s \ s.t. \ cluster(v_s) = \{v_{s1},CIP(v_{s1})_{old},CIP(CIP(v_{s1})_{old})_{old},...,0\}$, where 0 is a terminating condition.
2: Sort tasks of $cluster(v_i)$ in ascending order of their $ect(v_i)_{old}$.
3: Update $\beta = \{cluster(v_{s1}),cluster(v_{s2}),cluster(v_{s3}),....\}$.
4: Calculate $ect(v_i)$, $CIP(v_i)$, $\alpha$, interCommunication.
5: Update tasksLeft.
6: if ($\alpha > \alpha_{old}$) & $\max(\sum_{v_i} w(v_i)) < \alpha_{old}$) then $\forall v_i \in v_s$, $\forall v_{s1} \in v_s$ then
7: if ($\beta_{old} == \beta$) $(\{tasksLeft = \{\})$ then
8: tasksLeft$_{old} = tasksLeft$
9: GO to step 22
10: end if
11: Set $\beta_{old} = \beta$, $CIP_{old} = CIP$
12: Call EED
13: end if
14: if ($\alpha < \alpha_{old}$) & (tasksLeft $\neq \{\})$ then
15: Set $ect(v_i)_{old} = ect(v_i)$, $CIP(v_i)_{old} = CIP(v_i)$, $\alpha_{old} = \alpha$, tasksLeft$_{old} = tasksLeft$, $\beta_{old} = \beta$ and
interCommunication$_{old} = interCommunication$
16: Call EED
17: end if
18: if ($\alpha < \alpha_{old}$) & $\beta_{old} \neq \beta$ & tasksLeft $= \{\}$ then
19: Set $ect(v_i)_{old} = ect(v_i)$, $CIP(v_i)_{old} = CIP(v_i)$, $\alpha_{old} = \alpha$, tasksLeft$_{old} = tasksLeft$, $\beta_{old} = \beta$ and
interCommunication$_{old} = interCommunication$
20: Call EED
21: end if
22: if tasksLeft $\neq \{\}$ then
23: Find $cluster(v_i) \mid \forall v_i \in \text{tasksLeft \ s.t. \ cluster(v_i) = } \{v_{i1},CIP(v_{i1})_{old},CIP(CIP(v_{i1})_{old})_{old},...,0\}$
24: Recursive clustering similar to steps 5-21 is repeated for each $v_i \in \text{tasksLeft}$.
25: Let $\beta_{temp} = \{cluster(v_{i1}),cluster(v_{i2}),...,cluster(v_{in})\}$
26: Call Energy Conservation Procedure
27: Set $\beta = \{\beta_{temp}\}$
28: end if
29: Assign each cluster in $\beta$ to a distinct processor $P_n$.
30: Calculate $ect(v_i)$, $\alpha$.
31: Call Energy Conservation Procedure
32: Update $ect(v_i)$
Both the EED and the EEND heuristics execute similarly except for the case that the latter allows for the coexistence of no task replicas in $\beta$. To avoid task duplications in $\beta$, the code in Figure 4 is executed at the end of the Energy Conservation Procedure in Fig. 3. Furthermore, this code assures that removing task replicas from a certain cluster leads to a minimal increase of the schedule length as opposed to removing from another.

D. Running Trace of the Algorithm

In this subsection, we illustrate the working flow of both the EED, and the EEND algorithms using the DAG example in Fig. 5a). In this example, we assume that a Core 2 Duo E6550 processor and a Myrinet interconnection technology are used with $P_{\text{active}} = 44W$ and $PL_{\text{active}} = 33.6W$.

Similar to our former approach presented in [20], we calculate the output values such that the final scheduling $\beta = \{\text{cluster}(v_2), \text{cluster}(v_5), \text{cluster}(v_7), \text{cluster}(v_8), \text{cluster}(v_{10})\}$ and the schedule length $\alpha = \text{ect}(v_{10}) = 21$. Furthermore, the interconnection energy consumption value is calculated using equation 7 such that $EL_{\text{active}} = 33.6 \times 8$, where 8 is the interCommunication value as depicted in Fig. 5b).

The scheduling in Fig. 5c) shows the initial output of calling the Energy Conservation procedure in Fig. 3. Based on the if condition of step 6, we observe that $\text{cluster}(v_2)$ excluding $v_2$ is a subset of $\text{cluster}(v_5)$ excluding $v_5$, and $\text{ect}(v_5) + w(v_2) = 11$ is less than $\alpha = 21$ in Fig. 5b). Therefore, we merge $\text{cluster}(v_5)$ and $\text{cluster}(v_2)$. We thereby calculate the values of $\text{ect}(v_2)$ and $\alpha$ and compare then to the previous values obtained. Once the condition $\alpha$ is equal to $\alpha_{\text{old}}$, the merge is considered successful. Otherwise the process is repeated for $\text{cluster}(v_7)$ and $\text{cluster}(v_8)$ in Fig. 5b). Finally, the new clustering $\beta$ is set to exclude $\text{cluster}(v_2)$ and $\text{cluster}(v_7)$ as depicted by Fig. 5c). As a result of the previous merging steps, both the CPU and the inter-connection energy consumptions are decreased by a significant amount as shown in Fig. 5c).

The scheduling in Fig. 5d) is the final output of the EED algorithm. In reference to the if condition of step 14 in Fig. 3, we aim to minimize the CPU energy consumption without affecting the schedule length. This can be done by omitting task replicas within cluster pairs in $\beta$ if such an attempt can decrease the total energy consumption. Thus, $\text{duplicateTasks}$ in Fig. 5c) are $v_1, v_4, v_6$, which are the commons of $\text{cluster}(v_2)$ and $\text{cluster}(v_8)$. Consequently, the CPU energy consumption for the $\text{duplicateTasks}$ is calculated such that $EN_{\text{active-duplicateTasks}} = 44 \times 9$. $\text{cluster}(v_8)$ is set to hold $\text{cluster}(v_8)$ of Fig. 5c) excluding the $\text{duplicateTasks}$. Moreover, the clustering $\beta$ is up-

---

0: Procedure Energy Conservation
0: INPUT: $G = (V, E, w, c)$, $\alpha_{\text{old}}$, $\beta_{\text{old}}$, ect$(v_i)_{\text{old}}$, interCommunication$_{\text{old}}$
0: OUTPUT: $\beta$, ect$(v_i)$, E.
1: if $\beta = \{\text{cluster}(v_{i1..i}) | \{v_{i1..i}\} \in \text{tasksLeft}\}$ then
2: if $(\text{cluster}(v_i) \subseteq (\text{cluster}(v_j)))$ then
3: $\beta_{\text{temp}} = (\beta_{\text{temp}} \setminus \text{cluster}(v_i))$
4: end if
5: end if
6: if tasksLeft $= \{\} \& (\text{cluster}(v_i) \setminus v_i) \subseteq (\text{cluster}(v_j) \setminus v_j) \& ((\text{ect}(v_i) + w(v_i)) \leq \alpha) \& \forall (\text{cluster}(v_i), \text{cluster}(v_j) \in \beta$ then
7: $\text{cluster}(v_i) = (\text{cluster}(v_j) \cup v_i)$
8: $\beta = (\beta \setminus \text{cluster}(v_j))$
9: Calculate ect$(v_i)$, $\alpha$
10: if $\alpha =\alpha_{\text{old}}$ then
11: $\alpha_{\text{old}} = (\beta \setminus \text{cluster}(v_i))$
12: end if
13: end if
14: if $(\text{cluster}(v_i) \cap \text{cluster}(v_j)) | \text{cluster}(v_i), \text{cluster}(v_j) \in \beta, v_i, j$ then
15: $\text{EN}_{\text{active-duplicateTasks}} = \text{PN}_{\text{active}} \cdot \sum_{i=1}^{n} w(v_i) | v_i \in \text{duplicateTasks}$
16: $\text{EN}_{\text{active-duplicateTasks}} = (\text{cluster}(v_j) \setminus \text{duplicateTasks})$
17: Calculate ect$(v_i)$, $\alpha$, interCommunication
18: Let $\beta = \beta \setminus \text{cluster}(v_j)$
19: Let $\beta = \beta \setminus \text{cluster}(v_i)_{\text{temp}}$
20: Calculate ect$(v_i)_{\text{old}}$, interCommunication$_{\text{old}}$
21: Let interCommunication$_{\text{diff}} = \text{interCommunication} - \text{interCommunication}_{\text{old}}$
22: if $(\alpha = \alpha_{\text{old}}) \& (\text{EN}_{\text{active-duplicateTasks}} > \text{interCommunication}_{\text{diff}} \cdot PL_{\text{active}})$ then
23: $\text{cluster}(v_i) = \text{cluster}(v_i)_{\text{temp}}$
24: Set ect$(v_i)_{\text{old}}$, CIP$(v_i)_{\text{old}}$, $\alpha_{\text{old}}$, $\beta_{\text{old}}$, interCommunication$_{\text{old}}$ to hold the current values.
25: end if
26: end if

---

Fig. 3. Pseudocode of the Energy Conservation Procedure

Fig. 4. Pseudocode of the EEND algorithm
dated to include $\text{cluster}(v_8)_{\text{temp}}$ while excluding $\text{cluster}(v_8)$. This setup is followed by updating the values of $\text{ect}(v_1), \alpha, \beta$ and calculating $\text{interCommunication}_{\text{diff}}$. This value of $\text{interCommunication}_{\text{diff}} = 10 - 6$ is calculated based on the current values of $\text{interCommunication} = 10$ obtained from Fig. 5d) and the value of $\text{interCommunciation}_{\text{old}} = 6$ obtained from Fig. 5c). A final check is made to assure that the interconnection energy consumption associated with $\text{interCommunication}_{\text{diff}}$ does not exceed the one for $\text{EN}_{\text{active}}$. $\text{duplicatedTasks}$ is equal to $v_1$, where $v_1$ is common to $\text{cluster}(v_5)$ and $\text{cluster}(v_{10})$. We thereby calculate the schedule length for the cases where task $v_1$ is excluded from $\text{cluster}(v_5)$ and $\text{cluster}(v_{10})$. Since schedule length values obtained from the previous step were 22 and 23, respectively, task $v_1$ is omitted from $\text{cluster}(v_8)$ as this leads to the minimal increase in schedule length.

E. Algorithm Complexity

In this subsection, we derive the time complexity of both the EED and the EEND algorithms. The dominant parts of
the Initialization procedure are the topological order sorting and the earliest start time calculation. The time complexity of the topological order sorting is \( O(|V| \cdot \log |V|) \). The time complexity of the earliest start time is \( O(|V| + |E|) \) since \(|E| \) edges are traversed plus \(|V|\), which is the time needed to traverse all the tasks. In each iteration of the EED algorithm, values of the earliest time are computed in \( O(|V| + |E|) \) time. In the worst case \(|V| \) iterations are required assuming that a task is added per iteration. The time complexity of the Energy Conservation procedure is \( O(|V|^2) \), where the contents of every cluster are compared with one another. In each iteration of the energy conservation procedure, values of the earliest start time are calculated. Hence, the worst case time complexities of both the EED and the EEND algorithms are \( O(|V|^2 \cdot (|V| + |E|)) \).

IV. EXPERIMENTAL RESULTS

In this section, we present the outcomes of comparing the EED and the EEND algorithms against the Traditional Duplication-based Scheduling (TDS), Energy Aware Duplication (EAD) and Performance-Energy Balanced Duplication (PEBD) algorithms. In order to effectively measure the performance and the energy consumption of these algorithms, several metrics were used such as Normalized Schedule Length (NSL), average number of processors, average Sum of Computations, average Sum of Communications and total energy consumption \((E)\). A Normalized Schedule Length (NSL) is defined as:

\[
NSL = \frac{\alpha}{w(v_{si}) + \sum w(v_i)} \cdot \text{critical path} (8)
\]

, where \(v_{si}\) is the sink task with maximum earliest completion time \(ect\). The average number of processors used is defined as the total number of processors used within all experiments divided by the total number of experiments. The average sum of computations is defined as the total execution time of all the tasks belonging to \( \beta \) from all the experiments divided by the total number of experiments. The average sum of communications is defined as the total communication requirements among the clusters in \( \beta \) for all the experiments divided by the total number of experiments. The total energy consumption \(E\) is measured using the energy consumption model presented in [31]. Since the goal behind the EED and the EEND algorithms is to generate schedules with minimum energy consumption given the schedule length and the number of processors constraints, we define a new objective function \(\gamma\) such that:

\[
\gamma = NSL \times \frac{E}{\max(E)} \times \frac{\text{NumberOfProcessors}}{\max(\text{NumberOfProcessors})} \quad (9)
\]

where \(\frac{E}{\max(E)}\) is the normalized value of the total energy consumption and \(\frac{\text{NumberOfProcessors}}{\max(\text{NumberOfProcessors})}\) is the normalized value of the number of processors used. Several other parameters including the Communication to Computation Ratio (CCR) which is the ratio of communication time to computation time, application type and number of processors, are tuned to observe the impact of each of the previously described metrics on the performance of the algorithms. Pertaining to our experiments, CCR values ranging from 1-11 are used. A real world application such as the Sparse Matrix Solver application is used (96 tasks and 67 edges). Moreover, random DAGs with the number of tasks set to 50 and 100 are used, respectively.

![Figure 6](image)

A detailed information about the aforementioned DAGS can be found in [32]. In this study, we have used the Intel Core 2 Duo E6300 processor with \( P_{N_{active}} = 44W \), and a Myrinet interconnection technology with \( P_{L_{active}} = 33.6W \).

Figures 6-8 show the outcomes of comparing the five algorithms in terms of different performance metrics. In Fig. 6, we show a comparison of the five algorithms in terms of the average NSL, the average number of processors, the average sum of computations, the average sum of communications, the total Energy \((E)\), and the objective function \(\gamma\) for different CCR values. These experiments were conducted using random DAGs and with the number of tasks set to 50. Fig. 7 displays a similar comparison when the number of tasks is set to 100. For the comparison in Fig. 8, a real world application was used namely the Sparse Matrix Solver. For this real world application, we only show the average NSL and the total Energy \((E)\) results because of the space requirements of this paper.

In Fig. 6(a), we observe that the average NSL of the EED algorithm improves gradually as the value of CCR increases. This is caused by the fact that TDS, EAD, and PEBD use a similar strategy that involves creating a new cluster once an entry task is reached, completely ignoring the communication overhead in the final scheduling stage. Our EED algorithm overcomes this shortcoming since it overloads clusters with tasks that impose communication overhead. Furthermore, it is noticed that the average NSL of the EEND does not compare favourably against other solutions where duplicated tasks are eliminated from the final scheduling \(\beta\). Fig. 6(b) shows that the numbers of processors used by both the EED and the EEND algorithms are the least among the other algorithms. These outcomes are the result of the fact that EED and EEND algorithms allow for combining different clusters which thereby reduce the number of processors used. As depicted in Fig. 6(c), the average sum of computations of the schedule generated by the EEND algorithm is the least followed by EAD, PEBD, TDS and lastly by the EED algorithm. These results are due to the fact that EEND does not allow for the coexistence of task replicas in the final scheduling. Furthermore, the EAD algorithm puts more weight on the CPU energy consumption than it does for schedule length, which becomes more obvious for larger CCR values. For the average sum of Communications in Fig. 6(d), the output obtained by the EED algorithm is less by a factor of thousands as opposed to the values obtained by the TDS, EAD and PEBD algorithms. For the same experiment, results obtained by the EEND are approximately comparable to the ones obtained by the EED. In Fig. 6(e) the total energy consumption of the EED and EEND reach their peak performance for larger CCR values as opposed to other solutions. It is also noticeable that our
approaches outperforms EAD, TDS and PEBD by a significant amount as in Fig. 6(f) in terms of the objective function $\gamma$.

In view of the experiments in Fig. 7, we observe that the results shown are consistent with the ones obtained in Fig. 6. In reference to Fig. 8, results obtained for the Sparse Matrix Solver application are as well consistent with the ones discussed above. Thus, we conclude that the performance and the energy consumption of the schedules generated by the EED are compelling for communication intensive parallel applications. For the EEND algorithm, results are captivating in terms of energy consumption, number of processors and objective function used. In reference to Fig. 8, we observe that the results for Sparse Matrix Solver application are as well consistent with the ones discussed above.

V. CONCLUSIONS

In this paper, we have addressed the problem of static scheduling of a parallel program represented as a DAG for execution on a distributed system with the objective of improving both the performance and the energy consumption of the schedules generated. In particular, we have proposed two energy-aware scheduling algorithms, namely, EED and EEND, that employ duplication and non-duplication based scheduling...
strategies, respectively. We have also devised a new objective function that can be used to measure the effectiveness of any given algorithm in terms of various performance metrics. For the total energy consumptions, our algorithms can achieve up to 50% improvement over the EAD algorithm. Moreover, the number of processing elements used by our approaches is 60% less than those needed by other approaches. For the average NSL, EED can achieve up to 18% improvement over TDS, while EEND shows 25% less performance than TDS using random DAGs. For the Sparse Matrix Solver, the average NSL values of the EED is better than existing solutions for large CCR values. Furthermore, EED and EEND outperform other algorithms in terms of energy consumption. We therefore conclude that both EED and EEND can substantially improve the performance and the energy consumption of any distributed system running communication intensive parallel applications.

REFERENCES


Fig. 8. Sparse Matrix solver Test a) Average NSL vs. CCR, b) Total Energy $E$ vs. CCR