Journal of Advances in Computer Research
Quarterly ISSN: 2008-6148
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 4, No. 1, February 2013), Pages: 13-29
www.jacr.iausari.ac.ir

Pre-scheduling and Scheduling of Task Graph on Homogeneous Multiprocessor Systems

Marjan Abdeyazdan[1], Saeed Parsa[2], Amir Masoud Rahmani[3]

1,3) Department of Computer Engineering, Science and Research branch, Islamic Azad University, Tehran, Iran

2) Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

m.abdeyazdan@srbiau.ac.ir; parsa@iust.ac.ir; rahmani@srbiau.ac.ir

Received: 2012/09/13; Accepted: 2012/10/24

Abstract

Task graph scheduling is a multi-objective optimization and NP-hard problem. In this paper a new algorithm on homogeneous multiprocessors systems is proposed. Basically, scheduling algorithms are targeted to balance the two parameters of time and energy consumption. These two parameters are up to a certain limit in contrast with each other and improvement of one causes reduction in the other one. The problem is to achieve the trade-off between these two parameters. Pre-scheduling algorithms are mainly aimed at modifying the structure of task graph to gain optimal scheduling.

In the proposed algorithm the suitable number of processors for scheduling the task graph is computed. The idea of Nash equilibrium is mainly applied to compute the appropriate number of processors in such a way that the idle time of the processors is reduced while their processing power is increased. Also, considering the communication costs and interdependencies, the tasks are merged as their earliest start time is reduced. In this way, the length of the critical path is reduced while the degree of parallelism is increased and ultimately the completion time is reduced. Our experimental result on a number of known benchmark graphs demonstrates the effect of our proposed algorithm.

Keywords: *schedule, pre-schedule, task graph, game theory, optimization, Nash equilibrium*

1. Introduction

A scheduling system model consists of an application, a target computing system and performance criteria for scheduling. An application program is represented by a Directed Acyclic Graph (DAG), $G=(V,E)$, where V is the set of v tasks nodes, and E is the set of e directed communication edges between the tasks. Each edge $e_{i,j}$ represents the precedence constraint that v_j cannot be scheduled until task v_i has been completed, hence v_i is a predecessor of v_j and v_j is a successor of vi. In a given task graph, a task without any parent is called an entry task and a task without any child is called exit task. Without loss of generality, it is assumed that there is one entry task to the DAG and one exit task from the DAG. Task merging, clustering and duplication techniques [5,7,8,16,20,30,40] have been widely applied to restructure task graphs for better scheduling in step pre-scheduling. Task merging have been most often targeted at

reducing the length of the critical path of the task graph by increasing the granularity. Proposed algorithm can present suitable result.

The general task scheduling problem includes the problem of assigning the tasks of an application to suitable processors and the problem of ordering task executions on each resources. The problem of optimal scheduling of tasks with required precedence relationship, in the most general case, has been proven to be NP-complete, and optimal solutions can be achieved if adequate time is available for an exhaustive search. Then, many heuristics have been proposed for giving an approximate optimization in polynomial time. The characteristics of an application represented by a directed acyclic graph (DAG) in which the nodes represent application tasks and edges represent inter-task data dependencies. The objective of task scheduling is to map the tasks on the processors and order their execution so that task precedence requirements are satisfied and a minimum overall completion time is obtained [28]. Because of its key importance on performance, this problem has been extensively studied and various algorithms have been proposed in the literature, which are mainly for systems with homogeneous processors. These heuristics are classified into a variety of categories such as list scheduling algorithms [4], clustering algorithms, Genetic algorithms [17,25] and task duplication based algorithms. In list scheduling algorithms, the tasks in a list are assumed priorities and are assigned to the different processors based on descending order of priorities. List scheduling algorithms are generally preferred since they generate good quality schedule.

The multiprocessor task scheduling problem that is an NP-hard problem and multi objective optimization (MOO) [22,35]. Although there are algorithms in the literature for homogeneous processors, we present a novel heuristic scheduling algorithms for a bounded number of homogeneous processors with an objective to simultaneously meet high performance and fast scheduling time, which are called the Pre-scheduling and Scheduling based on Game Theory (PSSGT) algorithm. Based on the schedule obtained by the list algorithm, the PSSGT algorithm lessens the schedule length by inserting the task in big list into the processor with earlier-start-time (EST) and inserting the task in small list into the earlier slack (hole) to use the slack effectively. The algorithm moves the slack upwards from the exit task by delaying the start times of some tasks. The algorithm re-allocates the remaining tasks to the suitable processor that satisfies the precedence sequence and has the minimum earliest-finish-time (EFT) of the task. By using the slack time on the processors effectively, PSSGT algorithm has high performance in terms of both performance metrics.

In the approach presented in this paper, a node is merged with a subset of its parents only if the merge operation reduces the node's earliest time to start. If the merge postpones the execution of the other children of the parent node, the parent node is duplicated. The duplication is performed to enhance the parallelism. However, if there are not enough processors to execute the parallel tasks, the duplication may increase the overall execution time of the tasks. In the task merging presented in this paper, before each parent node can be duplicated, the maximum number of independent tasks within the task graph is recomputed. If after the duplication, the number of independent tasks gets above the number of available processors, the duplication is considered as an opposing factor. In order to compute the number of available processors and benefit merging and also determine the group of tasks. We have applied the game theory and then we have determined the number of available processors where select aware and the

processors use to suitable and also we have determined the group of tasks in order to schedule small task in idle time processors.

The appropriate number of processors for scheduling a given task graph could be computed before scheduling the task graph. Considering each level of the task graph as a player looking for appropriate number of processors to execute its independent tasks addressed by its nodes, the Nash equilibrium idea of game theory could be applied to compute the number of processors in order to minimize the idle time of the processors and overall throughput of processors is maximized where the goal is to balance the completion time of a parallel application with the overall energy consumption (CPU) [6,23,26].

Equilibrium is a key concept in game theory. As optimization problems seek to optimal solutions, a game looks for equilibrium [21]. Given a game with strategy sets for players, Nash equilibrium is a strategy profile in which each player deterministically plays her chosen strategy and no one has an incentive to unilaterally change her strategy. Nash proved that every game with a finite number of players, each having a finite set of strategies, always possesses a mixed Nash equilibrium [27]. The concept of Nash equilibrium has become an important mathematical tool for analysing the behaviour of selfish users in non-cooperative systems. The celebrated result of Nash [32,33] guarantees the existence of Nash equilibrium in mixed strategies for every (finite) strategic game and many algorithms have been devised to compute one [31].

Some research has been presented in the literature for energy aware scheduling of tasks [19], Since finding an optimal schedule is an NP-complete problem in general, researchers have resorted to devising a plethora of heuristics using a wide spectrum of techniques, including branch-and-bound, integer programming, searching, graph-theory, randomization, genetic algorithms, and evolutionary methods [35,37]. Some research has been presented in the literature for energy aware scheduling of tasks with precedence constraints [36,38], and without precedence constrains [34] for parallel machines.

The remaining parts of this paper are organized as follows: Firstly, in Section 2, motivation and related work is presented. In Section 3, a new algorithm is presented. This algorithm uses equations which are fully described in subsections 3.1 to 3.5. In Section 3.1, selecting number of suitable processor, in section 3.2, merging tasks, in section 3.3, grouping of tasks, in sections3.4, 3.5 scheduling of the list big and small is presented. In section 4, the results of applying our algorithm to some benchmark task graphs is compared with the results of applying some known scheduling algorithms. Finally, in Section 5, conclusion is presented.

2. Motivation and related work

Pre-scheduling techniques are mainly applied to reshape task graphs in a form suitable for scheduling. To reshape a task graph, task merging [1], clustering [2] and duplication [3] techniques are most commonly applied. Chou Lai and Yang [20] have proposed a new duplication-based task scheduling algorithm for distributed heterogeneous computing (DHC) systems. For such systems, many researchers have focused on solving the NP-complete problem of scheduling directed acyclic task graphs to minimize the makes pan. There are several non-deterministic approaches to solve the multiprocessor task scheduling problem that is an NP-hard problem. The genetic algorithm presented in [25], has provided relatively better scheduling in comparison

with the other non-deterministic approaches. The algorithm is bipartite in a way that each part is based on different genetic schemes, such as genome presentation and genetic operators. In the first part, it uses a genetic method to find an adequate sequence of tasks and in the second part it finds the best matching processors. However, these non-deterministic approaches provide different scheduling for a given task graph in different runs of their underlying algorithms. In order to cope with large scale task graphs, a cluster-based search (CBS), for scheduling large task graphs in parallel on a heterogeneous cluster of workstations connected by a high-speed network, is presented.

Programs using parallel tasks can be modeled as task graphs so that scheduling algorithms can be used to find an efficient execution order of the parallel tasks. Static scheduling has been well accepted for its predictability and online simplicity. Traditional static schedule generation techniques are usually based on the assumption of constant rate of resource supply known at design time. A pre-schedule is a static schedule without assuming constant and completely predictable rate of resource supply. The concepts of supply function and supply contract are introduced to define the actual online resource supply rate and the constraints to this rate known off-line. Based on these concepts, the pre-scheduling problem is defined and sound pre-schedulers are presented [24].

3. New algorithm PSSGT

There are five major steps in our proposed algorithm, PSSGT. In the first step: determining the appropriate number of processors for initial graph. Second step is merging initial graph and making the last graph and third step is grouping of tasks. Finally, last steps is scheduling of tasks. We are sure that the number of processors in first step appropriate for second step. Because in merging step, the number of tasks in the same level doesn't increase and the criteria for choosing is according to the number of task in the same level. Every task will be merged with its parents. The relationship between parent and child are into two sequential levels. Therefore, the number of parallel processors that chosen in first step are appropriate for merged graph in the second step. Pre-scheduling and scheduling will be done in 5 steps in new algorithm PSSGT:

Step 1: Selecting number of suitable processor
Step 2: Merging tasks
Step 3: Grouping of tasks
Step 4: Scheduling of the list big
Step 5: Scheduling of the list small

3.1 Selecting number of suitable processors (step 1)

Selecting number of processor is vital for entering graph in order to get the better use out of the most important source which is processor (idle time of processor become less). Therefore number of suitable processor for entering graph based on game theory is computed by "G*" formula.

Consider a graph with n level, suitable choice for number of processor using game theory. Processor number in i^{th} level equals "g_i". In other word, number of processors in i^{th} level is "g_i" that maximum needed number of processor in i^{th} level is equal with number of tasks at i^{th} level. So average number of existing processor (for levels of

graph) or in the other word referable average number equals with needed processor number in n level. As shown in equation(1):

$$G = (g_1+g_2+\ldots+g_n)/n = (\sum_{i=1}^{n} g_i)/n \quad (1)$$

Average number of all needed processor in graph is G. Number of needed processor in first level is g_1. Number of needed processor in n^{th} level is "g_n", where "G" is average number of processors. We call each value the "V(G)", every processor for having economic value to remain needs task of exist. Task have fixed capacity, in other word number of tasks in each level is fixed so maximum task number among all levels, realizes maximum number of tasks and if the number of processor is more than maximum number of tasks among level, It is wasted and has no economic value."V(G)" is economic value of each processor and "g_{max}" is biggest value between "g_1" until "g_n". When average number of processor is low, adding another processor harms less to the economic value of existing processor because there is enough of task but when the number of existing processors is much, adding another processor, harms more to the economic value of existing processors. Firstly, adding an extra processor reduces economic value of existing processor. It means to get first offshoot as shown in equation (2):

$$\frac{dV(G)}{dG} = V'(G)<0 \quad (2)$$

Secondly, the worth of economic processor will be decline of you add more processor. To getting second offshoot as shown in equation (3):

$$\frac{d^2V(G)}{dG^2} = V''(G) < 0 \quad (3)$$

It means curve of the ramp "V(G)" is falling and increasing falling. Two features of equations (2), (3) in curve are shown this way in figure 1.

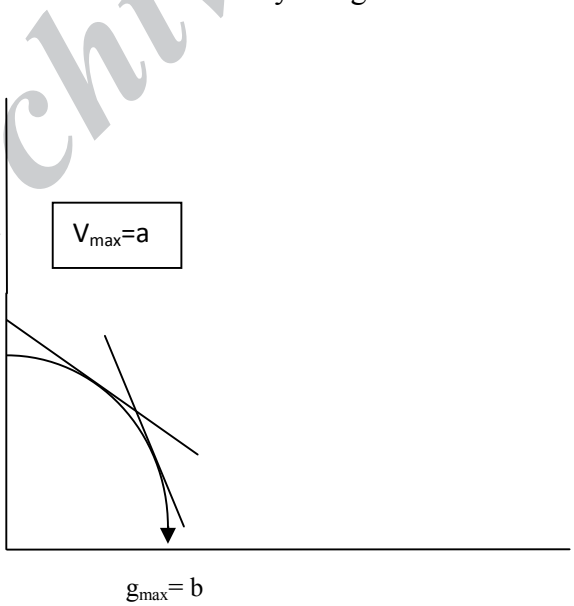


Figure 1. The curve of value processors [29]

The curve above is related to the part of kind parabola in first quarter because of the falling ramp of the curve and its falling. It means the first offshoot is negative and also

cave is downward. It means the second offshoot is negative and it will be presented as shown in equation (4):

$$V=-aG^2+v_{max}(V>=0,G>=0)(4) \quad (4)$$

According to equation (4),kind parabola equation, we know a>0 and v_{max}> 0. So if we want to know in which point of the curve "V(G)" equals zero:

$V= -aG^2+v_{max} \rightarrow 0= -aG_{max}^2+v_{max} \rightarrow aG_{max}^2= v_{max} \rightarrow a=(v_{max} / G_{max}^2) \rightarrow \{v_{max}> 0 , G_{max}^2> 0 \rightarrow a>0\}$

Also for getting maximum amount "V(G)" (maximum profit) "G" must be equal zero. Because in the curve, highest amount for "V(G)" is the lowest amount in G. So:

$V= -aG^2+v_{max} \rightarrow V= -a(0)^2+v_{max} \rightarrow V=0+v_{max} \rightarrow V= v_{max}$

It means the highest profit is in G=0. After getting the first offshoot we have to prove the amount is negative.

$\{\frac{dv(G)}{dG} <0 V= -aG^2+v_{max}\} \rightarrow \{\frac{dv(G)}{dG} = -2aG, a > 0, G> 0, -2 < 0\} \rightarrow \{\frac{dv(G)}{dG} = -2aG < 0\}$

According to the curve, domain of "G" is positive and "−2aG" less than zero. So the first offshoot result is negative. Also we have to prove the second offshoot is negative. It means:

$\{\frac{d^2V(G)}{dG^2} = V''(G) < 0, V= -aG^2+v_{max}\} \rightarrow \{\frac{dv(G)}{dG} = -2aG, a > 0, G > 0\} \rightarrow \{\frac{dv(G)}{dG} = -2aG<0\}$

$\{\frac{d^2V(G)}{dG^2} = V''(G) = -2a , a >0 , -2 < 0\} \rightarrow \{\frac{d^2V(G)}{dG^2} = V''(G) = -2a< 0\}$

Because "*a*" bigger than zero then "-2a"less than zero. So result of all phrases is phrase negative and the result of second offshoot is negative. According to the curve if ramp of the line be tangent on the negative curve, the first offshoot is negative and if cave be downward the second offshoot is also negative. It has to making free decision in each task graph. To know how many processors it has to have. Then with this thought that the number of processors one able division. It means processor has to be shifty. And strategy spaces of each point would be like these:

$G_i=[1, \infty)$ $g_i \in G_i$, $i \in N$

Extreme and more processor and one processor at least and strategy space i^{th} level is "G_i". To have so many processors or further than "g_{max}" will be not except able and it has no economic worth. The status of "g_{max}" will be reconsideration under condition of:

$G_i=[1, g_{max}]$ $g_i \in G_i$,$i \in N$

Play form strategic according more information:

Collection of player: $N=\{1,2,\ldots,n\}$

Players strategic: $G_i=[1, g_{max}]$ $g_i \in G_i$, $i \in N$Because we have $V(G) < 0$,

if$g_i> G \rightarrow g_i- G > 0 \rightarrow U_i> 0$ else $g_i- G < 0 \rightarrow U_i< 0$

Therefore utility players (levels):

$$U=(g_i, g_{-i})=(g_i -G)V(G)= (g_i V(G) - G V(G)) \quad (5)$$

$U= (g_i,g_{-i})=g_i V(g_1+g_2+\ldots+g_{i-1}+g_i+g_{i+1}+\ldots+g_n) - G V(g_1+g_2+\ldots+g_{i-1}+g_i+g_{i+1}+\ldots+g_n)$

Where: $g_{-i} = (g_1 + \ldots + g_{i-1} + g_{i+1} + \ldots + g_n)$

Strategic mixture $G^*=(g_1^*,g_2^*,\ldots,g_n^*)/n \in G$ is call "Nash equilibrium" as we have for each player : $u_i(g_i^*,g_{-i}^*) \geq u_i(g_i, g_{-i}^*)$ $\forall g_i \epsilon G_i$ $\forall i \epsilon N$

In the word considering the belief of "i" player to choosing rivals "$g_{-i}^{*"}$ that function $g_i \in G_i$ player "i" must maximum function called G with maximum of $g_i \in G_i$ and that "g" shows "$g_i^{"}$. So the answer of suitable is: Max u_i (g_i, g_{-i}*) $\forall i \epsilon N$ $g_i \epsilon G_i$

According to the function best answer we can say the "Nash equilibrium" in a G game comes when we have: $g_i^* \epsilon$ B_i (g_{-i}^*) { $g_i^* \epsilon G : u_i(g_i^*, g_{-i}^*) \geq u_i(g_i, g_{-i}^*)$} $\forall g_i \epsilon G_i$ $\forall i \epsilon N$

If in collection "$B_i(g_{-i}^*)$" is one number of Nash equilibrium, this way Nash equilibrium is "g_i*". Therefore in our sample to find the Nash equilibrium in the game is mixture $G^* = (g_1^*, g_2^*, \ldots, g_n^*)/n$. Nash equilibrium is game if we have:

Max $u_i(g_i^*, g_{-i}^*)$=Max[$(g_i - G)V(g_i + g_{-i}^*)$]= Max[{$g_i$ $V(g_i + g_{-i}^*) - G$ $V(g_i + g_{-i}^*)$} $g_i \epsilon G_i$]

The result f the first condition has gone like this:

$du_i(g_i^*, g_{-i}^*)/dg_i = V(g_i + g_{-i}^*) + g_i V'(g_i + g_{-i}^*) - (1/n) V(g_i + g_{-i}^*) - G V'(g_i + g_{-i}^*) = 0$

➔ $du_i(g_i^*, g_{-i}^*)/dg_i = V(1-1/n) + V'(g_i^* - G) = 0$, {assume (1-1/n)=1 because number of processors is integer} ➔

$du_i(g_i^*, g_{-i}^*)/dg_i = V + V'(g_i^* - G) = 0$ ➔ $V'(g_i^* - G) = -V$ ➔ $(g_i^* - G) = -(V/V')$ ➔ $g_i^* = -(V/V') + G$

$g_i^* = (-(-aG^2 + v_{max}) / (-2aG)) + G$ ➔ $g^* = ((-aG^2) / (2aG) + (v_{max}) / (2aG)) + G$

➔ { if V=0 ➔ $G = g_{max}$ ➔ $V = -aG^2 + v_{max}$ ➔ $0 = -aG^2 + v_{max}$ ➔ $v_{max} = ag_{max}^2$}

$g_i^* = ((-G/2) + ((ag_{max}^2)/(2aG)) + G) = (g_{max}^2 + G^2)/(2G)$, $G^* = g_i^* (G/g_{max})$

Equation (6) shows balance amount in Nash equilibrium for amount of processors in all levels. As:

$$G^* = (g_{max}^2 + G^2)/(2g_{max}) \quad (6)$$

Because for all level we have: $G = (g_1 + g_2 + \ldots + g_n)/n = (\sum_{i=1}^{n} g_i)/n$

So the suitable number of processor for per level became "G*". Finally "α" is a parameter that depends on the number of available processors and task graph structure. If there are enough processors to execute parallel tasks within the task graph, "α" will be equal to 100. Otherwise, if the number of available processors is less than the number of parallel tasks within the task graph, "α" will be a integer number between 0 and 100. The value of "α" depends on the maximum number of tasks to be executed in parallel, "$g_{max}^{"}$, and the number of available processors, "G*". To estimate the value of "$g_{max}^{"}$, the task graph is topologically sorted. "α" is number between 0 and 100 where depended available processors and "α" represented percentage of exist processors and As shown in equation (7):

$$\alpha = (G^*/g_{max}) * 100 \quad \{always G^* <= g_{max}\} \quad (7)$$

In equation (7), value of "G*" is the number of available processors suitable which compute fully described.

3.2 Merging tasks (step 2)

A task within a task graph starts immediately after all of its parent nodes execution is completed. Therefore, to compute the earliest start time, "EST_v", of a task, "v", the earliest start time and the execution time of its parent nodes and the time it takes to receive the results from its parent nodes are required. In order to compute "EST_v", the following equation can be used:

In equation (8), "$P_i(v)$", indicates the set of the parent nodes of the task "v". If a node "v", has no parent nodes its earliest start time will be zero.

$$ESTv = MAX(EST(p_i) + \tau(p_i) + c(p_i, v)) \quad (8)$$

In equation (8), "$c(p_i, v)$" is the size of the data to be received by the task "v" from the parent "p_i". When combining a node "v" with one of its parents, "p_1" to "p_n", the start time of the siblings of "v" may be increased. The time, "$R_{pi,v}$", at which the outputs of each parent node, "p_i", can be collected by the child, "v", is computed. In order to compute the value of "$R_{pi,v}$" the following equation (9) is applied:

$$R_{p_i,v} = EST(p_i) + \tau(p_i) + c(p_i, v) \quad (9)$$

In equation (9), "$c(p_i, v)$" indicates the size of the data that must be transmitted from "p_i" to "v". After "$R_{pi,v}$" are computed, the parents are sorted in decreasing order of "$R_{pi,}$". Starting with the node with the highest value of "$R_{pi,v}$", the benefit of merging "v" with each of the parents, "p_i", on the earliest start time of "v" is computed. The node "v" is merged with the subset of its parents, which reduce its earliest start time, the most, and their number does not exceed the number of available processors.

To decide whether to merge a task with a subset or all of its parents, equation (10) can be applied. This equation computes the benefit, "Q", of combining a task with a subset of its parents considering the amount of reduction in earliest start time of the task, the execution time of the duplicated nodes and the number of parallel processors.

$$Q = \alpha * \Delta T - (1-\alpha) * \Delta E \quad (10)$$

In equation (10), ΔT indicates the amount of reduction in earliest start time of the node "v", "ΔE" is the sum of the execution time of the duplicated parents of "v" and finally "α" is a parameter that depends on the number of available processors. If there are enough processors to execute parallel tasks within the task graph, "α" will be equal to 100, Otherwise if the number of available processors is less than the number of parallel tasks within the task graph, "α" will be a number between 0 and 100. The merging that amount of "ΔT" is more than "ΔE" are suitable merges and among them maximum benefit is chosen.

The value of "$\Delta T_{v,P_k}$ is computed by applying equation (11):

$$\Delta T_{v,P_k} = EST_{v'} - EST_v - \sum_{p \in P_k} \tau(p) \quad (11)$$

In equation (11), "$\Delta T_{v,Pk}$" indicates the reduced amount in earliest start time of the node v, where $P_k = \{p_1, p_2, \ldots, p_k\}$ and "$\Delta E_{v,Pk}$" represents the total execution time of the duplicated parents. In equation (11),"v'" addresses the node created by merging v with the parent nodes in "P_k" and "EST_v" indicates the earliest start time of the node, "v".

In order to compute the value of "$\Delta E_{v,Pk}$", the set of siblings of "v", "$\Psi_{v,Pk}$", whose earliest start time increases after the merge is selected, firstly. Then, the nodes $p_i \in P_k$ which are also parents of one or more nodes in "$\Psi_{v,Pk}$" are duplicated. The value of "$\Delta E_{v,Pk}$" is computed by applying equation (12) as the sum of the execution time of the nodes "p_i".

$$\Delta E_{v,Pk} = \sum \tau(p_i), \ \forall p_i \in \{P_k \cap Parents(\Psi_{v,Pk})\} \quad (12)$$

The proposed algorithm attempts to increase parallelism in the execution of a given task graph by reducing the earliest start time of the tasks within the task graph. The

earliest start time of a task is dependent on the completion time of its parent nodes and the time it takes to receive the results from the parents.

The propose algorithm attempts to increase parallelism in the execution of a given task graph by reducing the earliest start time of the tasks within the task graph. The earliest start time of a task is dependent on the completion time of its parent nodes and the time it takes to receive the results from the parents. Merging tasks with one or some parents happens, while it brings profit. While merging one task with one or some of parents, duplicating parents merged for siblings, earliest starting time of executing current task lessens, the merging is possible. Therefore profit riches when merging and duplicating parents changing in earliest start time of current task be more than total time of executing duplicated tasks.

3.3 Grouping of tasks (step 3)

In order to group tasks of each level in big and small group for scheduling, we use the same formula "G*" that we had as result in step 1 and call it "T*" that "t_{max}" is the maximum time of executing task in related level and T is the average time of executing tasks of related level and will be computed as the follow in equation (13):

$$T^*=(T_{max}^{\ 2} + T^2) / T_{max} \quad (13)$$

These concepts and equation (13) that we have told are used as the suitable choice in grouping tasks in each task graph and considering having execution time of tasks in each level. We group tasks in two different lists, big and small. Equation (13) is used for tasks in every level in the followed way. Therefore the suitable for grouping in two separate lists big and small for one level equals "T*". In the way number of tasks in each level equal n. "T" is the average time to run whole tasks in related level and "T_{max}" is the maximum time to execute among tasks.

Tasks assign in list big where that have higher executing or equal "T*". Then big list placed for initial scheduling and tasks with less executing time than "T*" assign in the small list and then list small placed in order to schedule in slack (idle time of processors) in order to lessen the idle time of processors and what if there is no possibility of assigning in holes in the current task, in the small list, in the suitable place regarding in order to executing of parents. The list big and small during these levels to fill up and first schedule list big in per level and then schedule list small in per level.

3.4 Scheduling of the list big (step 4)

Regarding "$EST(v_i,p_j)$" in equation (8) is earliest starting time of task i^{th} on the processor j^{th} and "$Avail(v_i,p_j)$" is available time for task i^{th} to the processor j^{th}, "$EFT(v_k,p_l)$" is earliest finishing time parent i on the processor "L" and "$C_{i,k}$"is the cost of common with the task "k".Equation (15) illustrated "$EFT(v_k,p_l)$" where "$W_{i,j}$"is the execution time for task i^{th} to the processor j^{th}.

$$EST(v_i,p_j)=Max\{Avail(v_i,p_j), EFT(v_k,p_l)+C_{i,k}\} \quad (14)$$

$$EFT(v_k,p_l)=W_{i,j}+EST(v_i,p_j) \quad (15)$$

3.5 Scheduling of the list small (step 5)

Existing tasks in the list small will be possibly placed in holes. Possibility of inserting in earliest hole (slack) and idle time of processors between earliest scheduling of task i^{th} and the next task k^{th} on the processor j^{th} that while the following equation is

on, Possibility of inserting in slack exists. Otherwise the current task in the list small in the proper place be assigned after parents and shifts rest of the task on the current processor. Equation (16) illustrated formula of inserting in slacks:

$$Max\ \{ f(v_k,p_j)\ ,\ EST(v_i,p_j)\ \} + W_{i,j} \leq S(v_l,p_j) \quad (16)$$

The processor selection is to schedule the tasks "v_i" onto processor "p_j" that gives the earliest finish time for the task. It uses an insertion based policy which considers the possible insertion of task "v_i" in an earliest idle time slack between the already scheduled task "v_i" and its immediate successor task "v_k" on processor "pj", if it satisfies equation (16). "$S(v_l,p_j)$" is the start time of task "v_l" onto processor "p_j", "$F(v_k,p_j)$" is the finish time of task "v_k" onto processor "p_j".

Delay calculation is to calculate the delay to all tasks. For avoiding increasing the schedule length, the earliest finish time of the exit task, its latest finish time, and the deadline for all the tasks are the same. The delay of a task is the difference between its current finish time and its latest finish time which is the critical path of that task, the maximize finish time starting from that task to exit task. The latest finish time is calculated recursively by traversing the task graph upward. The delay time is calculated as following:

$$Delay(v_i,p_j) = LFT(v_i,p_j) - EFT(v_i,p_j) \quad (17)$$

$$LFT(v_i,p_j) = Max\ v_k succ(v_i)\ (LFT\ (v_k,p_l) - c_{i,k}) \quad (18)$$

Where LFT (v_{exit}, p_j)=EFT (v_{exit}, p_j)

4. Experimental Results

In this section, second step of our proposed algorithm (pre-scheduling), PSSGT, is firstly compared with a known algorithm, GRS [7,8]. Then the length of the critical paths and degree of parallelism in the resultant task graphs are compared. The experiments have been carried out on a micro-computer with Intel(R) Core(TM) i5 CPU M480, 4 GB of RAM (64-bit). All the programs are in C# and developed within the Microsoft Visual Studio 2010.

Recently challenge is about the much number of tasks and processors. In order to be sure that proposed algorithm presented a good result for much number of tasks and processors. We have run all the ten algorithms on five instances of various sizes of both the Gauss–Jordan elimination problem and the LU factorization problem. In our simulation, we assume the number of processors is four. The problem sizes, the computation and communication time considered were summarized in table 1.

Table 1: Experimental setup

Problem	No. of tasks	Computation time	Communication time
Gauss–Jordan elimination	15, 21, 28, 36 (GJ)	40s/task	100s
LU factorization	14, 20, 27, 35(LU)	10s bottom layer task, plus 10s for every layer	80s

For the nondeterministic scheduling algorithms, the simulations were run 100 times and the average values reported. The number of tasks we choose for LU and GJ algorithms is based on their "layered" structure diagrams shown in figures2,3, respectively. Let us explain the reasoning behind the number of tasks chosen for our performance study. We use a number of dashed lines to help explain the chosen tasks' numbers.

Figure 4 shows an example of GJ algorithm with 36 tasks. The numbers of tasks we choose in GJ algorithm is straighter forward. Layers L0, L1, L2, L3, L4, L5, L6, L7, contain 3, 6, 10, 15, 21, 28, and 36 tasks, respectively. Therefore, the total numbers of tasks between layer L0 and L1, L2, L3, L4, L5, L6, and L7 are 3, 10, 15, 21, 28, and 36.

Also figure 5 is an example of LU algorithm with 35 tasks. Each layer starts with one task followed by a number of tasks. The LU diagram increments from layer L1 into L2, L3, L4, L5, and L6, thus the number of task increasing from 9 into 14, 20, 27, and 35, respectively[1].

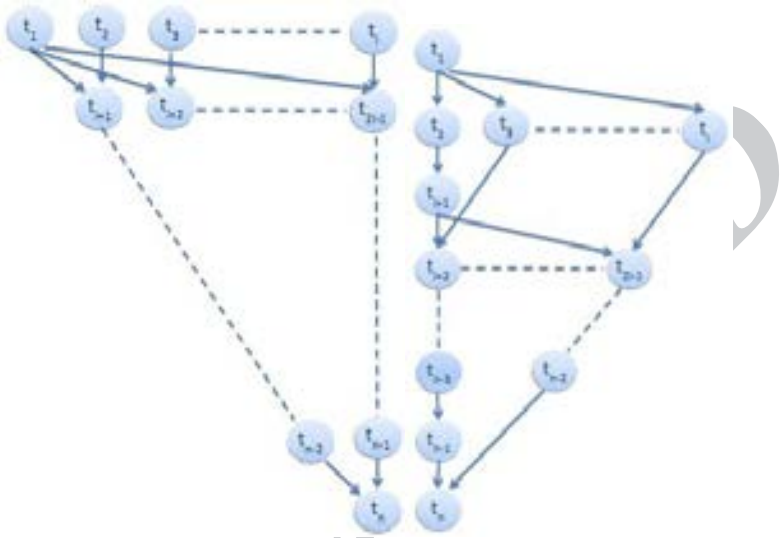


Figure2. The task graph for the Gauss–Jordan elimination algorithm (left side)

Figure3. The task graph for the LU decomposition algorithm (right side)

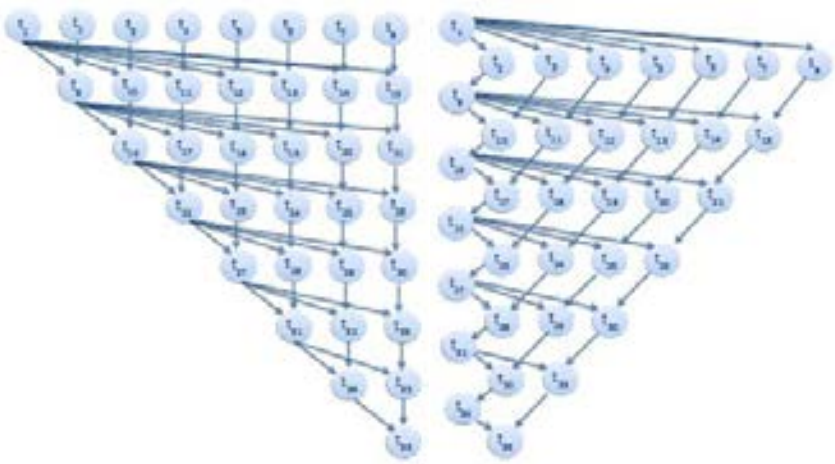


Figure4. The task graph for GJ algorithm with 36 tasks (left side)

Figure 5. The task graph for LU decomposition algorithm with 35 tasks(right side)

Considering our assumption of four processors, a number of tasks longer than 35 would not yield qualitative differences, especially considering that the additional tasks are the same type as the previous ones. As discussed thus far, PSSGT is an effective way to decrease a makes pan. We use elitist selection, meaning that the best individual

always survives in one generation, to keep the best result into the next generation. All GA parameters of this simulation are shown in table 2.

Table 2. GA parameters

Parameters	Value
Population size	30
Mutation rate	0.01
Crossover rate	0.7
Stopping criteria: Stops when it converges in a predefined threshold when the max. no.of generation exceeds 6,000	

The benchmark of task graphs FFT1, FF2, FFT3, FFT4are shown in figures 6, 7, 8, 9 [14,15,18].

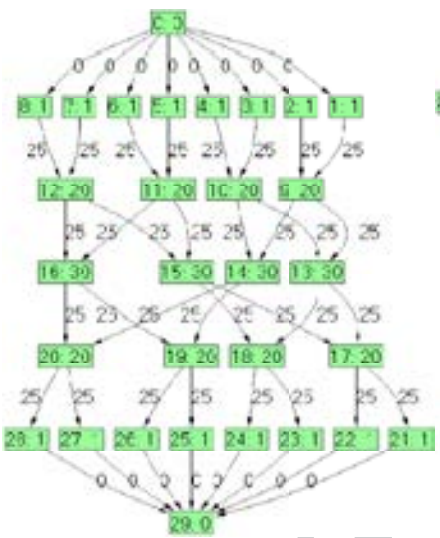


Figure 6. Task graph (FFT1)

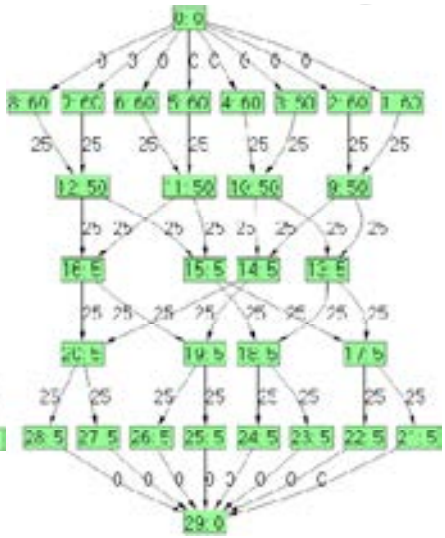


Figure7. Task graph (FFT2)

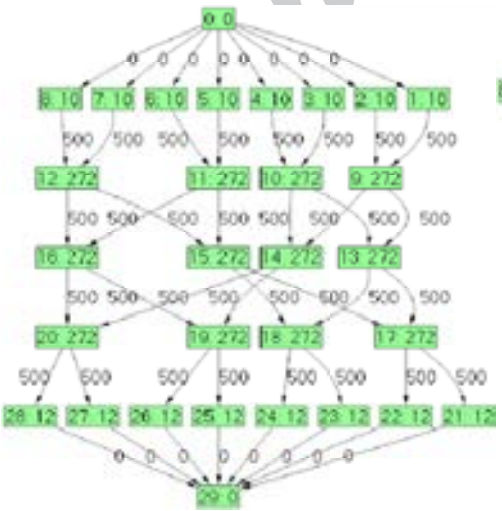


Figure8. Task graph (FFT3)

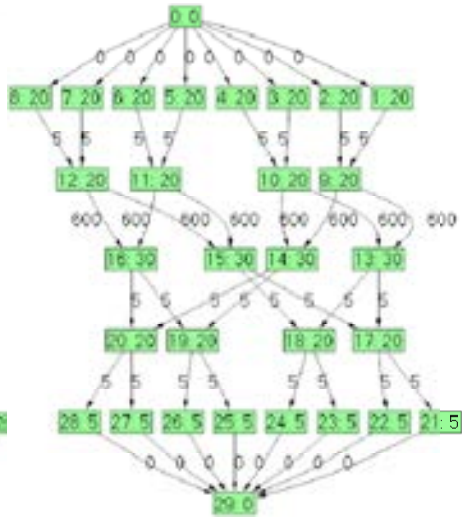


Figure9. Task graph (FFT4)

The benchmark of task graphs IRR, STD are shown in figures 10,11 [14,15,18].

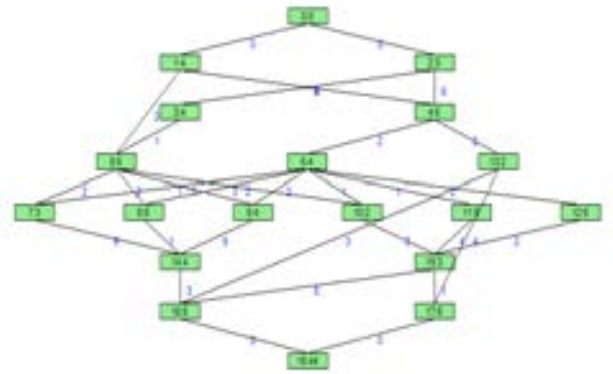


Figure10. Task graph (IRR)Figure *11. Task graph (STD)*

4.1 First step of PSSGT (processors cost)

Figure 2 shows the GJ algorithm. The numbers of tasks we choose in GJ algorithm is straighter forward. Layers L0, L1, L2, L3, L4, L5, L6, L7 contain 3, 6, 10, 15, 21, 28, and 36 tasks. Also figure 3 shows the LU algorithm. Each layer starts with one task followed by a number of tasks. The LU diagram, the number of task increasing from 9 into 14, 20, 27, and 35, respectively.

Table 3.The effect of applying PSSGT to estimate the appropriate number of processors for 8 task graphs

Task Graph	Completion time with Max. number of processors are available	Completion Time when the number of processors are computed by PSSGT	Percentage of reduction in the number of processors	Percentage of increase in execution time of the task graph
GJ_15	440, 5(cpu)	460, 4(cpu)	0.20	0.043
GJ_21	540, 6(cpu)	540, 5(cpu)	0.17	0
GJ_28	680, 7(cpu)	680, 5(cpu)	0.29	0
GJ_36	760, 8(cpu)	780, 6(cpu)	0.25	0.026
LU_14	350, 4(cpu)	320, 3(cpu)	0.25	-0.086
LU_20	470, 5(cpu)	530, 3(cpu)	0.40	0.113
LU_27	650, 6(cpu)	680, 4(cpu)	0.34	0.044
LU_35	860, 7(cpu)	930, 4(cpu)	0.43	0.075

The applicability of PSSGT in predicting the appropriate number of processors for scheduling 20 random task graphs and 20 new task graphs is shown in table 4. As shown in table 4, in average the execution time of the task graphs are increased about 3.5% while the number of parallel processors is reduced by 29.5%.

Table 4. The effect of applying PSSGT to estimate the appropriate number of processors for different graphs

Task Graph	Completion Time with Max. number of processors are available	Completion Time when the number of processors are computed by PSSGT	Percentage of reduction in the number of processors	Percentage of increase execution time of the task graph
20RandomGraphs	981	1020	0.31	0.04
20New Graphs	890	920	0.28	0.03

The applicability of PSSGT in predicting the appropriate number of processors for scheduling 6 benchmark task graphs is shown in table 5. As shown in table 5, in average the execution time of the task graphs are increased about 12% while the number of parallel processors is reduced by 27.2%.

Table 5. The effect of applying PSSGT to estimate the appropriate number of processors for 6 benchmarks

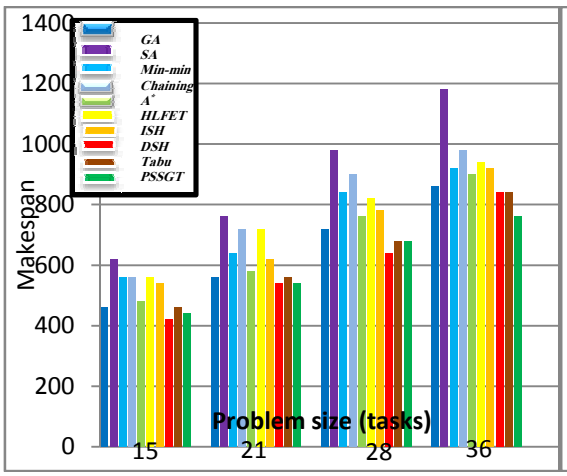
Task Graph	Completion time with Max. number of processors are available	Completion Time when the number of processors are computed by PSSGT	Percentage of reduction in the number of processors	Percentage of increase in execution time of the task graph
FFT1	120, 8(cpu)	144, 6(cpu)	0.25	0.16
FFT2	180, 8(cpu)	235, 6(cpu)	0.25	0.23
FFT3	1640, 8(cpu)	1924, 6(cpu)	0.25	0.15
FFT4	125, 8(cpu)	145, 6(cpu)	0.25	0.14
IRR	525, 7(cpu)	536, 5(cpu)	0.29	0.02
STD	38, 6(cpu)	39, 4(cpu)	0.34	0.02

4.2 PSSGT vs. other algorithms

The quality of results is measured by two metrics: makes pan and computational cost. The makes pan is represented by the time required to execute all tasks, the computational cost is the execution time of an algorithm. A good scheduling algorithm should yield short makes pan and low computational cost. In our simulation, we evaluate Gauss–Jordan elimination and LU factorization algorithms based on different number of tasks, with the former increasing from 15, 21, 28 to 36 and the latter from14, 20, 27 to 35. Other parameters such as task computation time and intercommunication delays are given in table 1. An ideal algorithm must perform well on different problems with different sizes.

The makes pan of the obtained solutions are represented on figure 12 for the Gauss–Jordan elimination and on figure 13 for the LU factorization.

Applying PSSGT algorithm to schedule five benchmark task graphs, the parallel execution times of the resultant scheduling were compared with the execution times resulted by applying 7 known scheduling algorithms. These scheduling algorithms are CLANS[9], DSC[10], MCP[11], LC[12], LAST[13], SR[14] and Genetic[17] and the benchmark task graphs are FFT1-4[15] and IRR[14]. In table 6 the comparison results are presented.



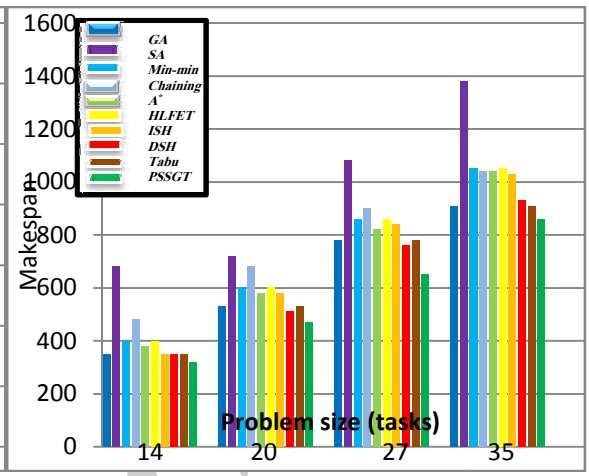


Figure12.Makespan for Gauss Jordan elimination for variable task sizes and the scheduling algorithms(left side)

Figure13.Makespan for LU factorization for variable task sizes and the scheduling algorithms (right side)

Table 6.The effect of PSSGT on preparing 5 benchmark task graphs for optimal scheduling

Graph	CLANS	DSC	MCP	LC	LAST	SR	GA	PSSGT
FFT1	124 (4)	124 (4)	148 (8)	127 (80	146 (1)	146	173 (3)	**120 (6)**
FFT2	200 (8)	205 (8)	205 (8)	225 (8)	240 (8)	215	225 (20)	**180 (6)**
FFT3	1860 (4)	1860 (4)	2350 (8)	2838 (8)	2220 (2)	--	1992 (2)	**1640 (6)**
FFT4	405 (2)	710 (12)	710 (8)	710 (8)	170 (8)	160	160 (8)	**125 (6)**
IRR	725 (7)	605 (12)	605 (7)	710 (8)	840 (3)	680	730 (3)	**525 (5)**

5. Conclusion

Multiprocessors systems were start of a revolution in computation with high utility and caused remarkable change in computation. Recently, researchers and workmen despite of multiprocessor systems can reach to higher goals. Therefore the purpose of pre-scheduling that is our suggested solution according to games theory and idea Nash equilibrium is minimizing schedule length and determining number of processors for minimizing idle processors.

Task graphs can be prepared for scheduling by minimizing their critical path length and maximizing their parallelism before the scheduling. To achieve this, a task merging and clustering algorithm can be applied. In this paper suggests merging each task with a subset of its parents for better merging, provided that the earliest start time of the task reduces and the start time of the siblings of the task does not increase after the merge. To avoid any increase in the start time of the siblings those parent nodes which are supposed to be merged with the child node, are duplicated. When duplicating a parent node, the number of tasks apparently increases. Therefore, there should be enough number of available processors to execute the parallel tasks, otherwise the duplication and thereby the merge will not be beneficial.

Problem is to achieve the trade-off between two parameters energy consumption and time (makespan). Our experimental result on a number of known benchmark graphs demonstrates the effect of our proposed algorithm and it is to achieve the trade-off between these two parameters.

6. References

[1] [1] Jin S, Schiavone G, Turgut D (2008) A performance study of multiprocessor task scheduling algorithms. DOI10.1007/ s11227-007-0139-z, J Supercomput 43, pp 77–97.

[2] [2]ChoonLee Y, Zomaya A, Siegel H (2010) Robust task scheduling for volunteer computing systems. DOI10.1007/ s11227-009-0326-1, J Supercomput 53, pp 163–181

[3] [3]Afgan E, Bangalore P, Skala T (2012) Scheduling and planning job execution of loosely coupled applications. DOI10.1007/ s11227-011-0555-y, J Supercomput. 59, pp 1431–1454

[4] [4]Ababneh I, Bani-Mohammad S, Ould-Khaoua M (2010) An adaptive job scheduling scheme for mesh-connected multi computers. DOI10.1007/ s11227-009-0333-2, J Supercomput. 53, pp 5–25

[5] [5]Niemi T, HameriA (2012) Memory-based scheduling of scientific computing clusters. DOI10.1007/ s11227-011-0612-6, J Supercomput. 61, pp 520–544

[6] [6]Qureshi K, Majeed B, Kazmi J.H, Madani S.A (2012) Task partitioning, scheduling and load balancing strategy for mixed nature of tasks. DOI10.1007/ s11227-010-0539-3, J Supercomput. 59, pp 1348–1359

[7] [7]Aronsson P, Fritzson P (2005) A Task Merging Technique for Parallelization of Modelica Models. In : 4th International Modelica Conference Hamburg

[8] [8] Aronsson P, Fritzson P (2003) Task Merging and Replication using Graph Rewriting. In : 2nd International Modelica Conferenc Germany

[9] [9] Aronsson P, Fritzson P (2002) Multiprocessor Scheduling of Simulation Code from Modelica Models.

[10] [10. Parsa S, Lotfi S, Lotfi N (2007) An Evolutionary Approach to Task Graph Scheduling. Lecture notes in computer scince, 4431, pp 110

[11] [11] Kim S, Browne J (1988) A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures. In : International Conference on Parallel Processing

[12] [12] McCreary C, Gill H (1989) Automatic determination of grain size for efficient parallel processing. Communications of the ACM, 32(9), pp 1073-1078

[13] [13] Yang T, Gerasoulis A (1994) DSC: scheduling parallel tasks on an unbounded number of processors. Parallel and Distributed Systems, IEEE Transactions on, 5(9), pp 951-967

[14] [14] Wu M, Gajski D (1990) Hypertool : A programming aid for message-passing systems. Parallel and Distributed Systems, IEEE Transactions on, 1(3), pp 330-343

[15] [15] BAXTER J, PATEL J (1989) The LAST algorithm- A heuristic-based static task allocation algorithm. In : International Conference on Parallel Processing

[16] [16]Parsa S, Reza Soltan N, Shariati S (2010) Task Merging for Better Scheduling. Lecture Notes in Computer Science, pp 311-316

[17] [17]Abdeyazdan M, Rahmani A.M (2008) Multiprocessor Task Scheduling using a new Prioritizing Genetic Algorithm based on Number of Task Children. Distributed and parallel system, pp 105-114

[18] [18] Kwok Y. K, Ahmad I (1999) Benchmarking and Comparison of the Task Graph Scheduling Algorithms. Journal of Parallel and Distributed Computing 59, pp 381-422
[19] [19]Ahmad I, Ranka S (2008) Using Game Theory for Scheduling Tasks on Multi-Core Processors for Simultaneous Optimization of Performance and Energy. 978-1-4244-1694-3/08 IEEE, Volume 27, Number 2, pp 177-194
[20] [20]ChouLai K, Yang C.T (2008) A dominant predecessor duplication scheduling algorithm for heterogeneous systems. DOI10.1007/ s11227-007-0152-2, J Supercomput. 44, pp 126–145
[21] [21]Gairing M, Lücking T, Mavronicolas M, Monien B (2010) Computing NashEquilibria for Scheduling on Restricted Parallel Links. Theory of Computing Systems, Volume 47, Number 2, pp 405-432
[22] [22]M.R.Garey, D.S.Johnson (1979), "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman and Company
[23] [23]Baskiyar S Abdel-Kader R (2010) Energy aware DAG scheduling on heterogeneous systems. Cluster Computing, Volume 13, Number 4, pp 373-383
[24] [24]Wang W, Mok A.K, Fohler G (2005) Pre-Scheduling. Real-Time Systems, Volume 30, Numbers 1-2, pp 83-103
[25] [25]Bonyadi M.R, Moghaddam M.E (2009) A Bipartite Genetic Algorithm for Multi-processor Task Scheduling. International Journal of Parallel Programming, Volume 37, Number 5, pp 462-487
[26] [26]Li K (2012) Energy efficient scheduling of parallel tasks on multiprocessor computers. J Supercomput, Volume 60, Number 2, pp 223-247
[27] [27]Thang N.K (2010) NP-hardness of pure Nash equilibrium in Scheduling and Connection Games.In : Proc. of the 35th Int. Conf. on Current, Trends in Theory and Practice of Computer Science (SOFSEM), 2009, Preprint submitted to Elsevier
[28] [28]Tosun S (2011) Energy- and reliability-aware task scheduling onto heterogeneous MPSoC architectures. J Supercomput.The Journal of Supercomputing, Online First™
[29] [29] Nisan N, Roughgarden T, Tardos E, Vazirani V (2007) Algorithmic Game Theory. Cambridge University, NY 10012-2473, Chapter 12 in Hand book, pp 301–330
[30] [30]Sinnen O, Sousa L (2004) On Task Scheduling Accuracy: Evaluation Methodology and Results. J Supercomput.,Volume 27, Number 2, pp 177-194
[31] [31]McKelvey R.D, McLennan A (1996) Computation of Equilibria in Finite Games. Chapter 2 in Hand book of Computational Economics, Volume 1, pp 87–142
[32] [32] Nash J.F (1951) Non-Cooperative Games. Annals of Mathematics, Vol. 54, No. 2, pp 286–295
[33] [33] Papadimitriou C.H (2001) Algorithms, Games and the Internet. Proceedings of the 33rd Annual ACM Symposiumon Theory of Computing, pp 749–753
[34] [34]Aydin H, Melhem R, Moss D, Meja-Alvarez P (2004) Power-Aware Scheduling for Periodic Real-Time Tasks. IEEE Trans on Computers, 53(5), pp 584-600
[35] [35] Deb K (2001) Multi-Objective Optimization using Evolutionary Algorithms. Wiley
[36] [36] Kang J, Ranka S (2008) Dynamic Algorithms for Energy Minimization on Parallel Machines. Euromicro Int'l Conf. on Parallel, Distributed and Network-based Processing
[37] [37] Khan S.U, Ahmad I (2006) Non-cooperative, Semi-cooperative and Cooperative Games-based Grid Resource Allocation. Int'l Parallel and Distributed Processing Symposium
[38] [38] Schmitz M.T, Al-Hashimi B. M (2001) Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. Int'l Symposium on System Synthesis, pp 250-255
[39] [39]Weichen L, Zonghua G, Jiang X, Xiaowen W, Yaoyao Y (2010) Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems. Digital Object Identifier 10.1109/TPDS.2010.204, IEEE
[40] [40]Agarwal A, Kumar P (2009) Economical Duplication Based Task Scheduling for Heterogeneous and Homogeneous Computing Systems. WEE International Advance Computing Conference (IACC)