Online Scheduling of Dynamic Task Graphs with Communication and Contention for Multiprocessors

Pravanjan Choudhury, P.P. Chakrabarti, *Senior Member*, *IEEE*, and Rajeev Kumar, *Senior Member*, *IEEE*

Abstract—This paper presents an online scheduling methodology for task graphs with communication edges for multiprocessor embedded systems. The proposed methodology is designed for task graphs which are dynamic in nature either due to the presence of conditional paths or due to presence of tasks whose execution times vary. We have assumed homogeneous processors with broadcast and point-to-point communication models and have presented online algorithms for them. We show that this technique adapts better to variation in task graphs at runtime and provides better schedule length compared to a static scheduling methodology. Experimental results indicate up to 21.5 percent average improvement over purely static schedulers. The effects of model parameters like number of processors, memory, and other task graph parameters on performance are investigated in this paper.

Index Terms—Multiprocessor scheduling, task graphs, static and online scheduling, edge scheduling.

1 Introduction and Related Work

The increasing requirement of computing power has shifted embedded system designs from a single processor to multiple-processor-on-a-chip solutions. A typical multiprocessor consists of a set of processors connected via a communication subsystem for exchanging data. A *directed acyclic graph* (DAG) is used to describe an application at a high level. Scheduling of the nodes to the processors (*task scheduling)* and edges to the communication channels (*edge scheduling*) with the objective of minimizing overall execution time of the DAG is called the general Multiprocessor Scheduling problem.

This general scheduling problem is known to be NP-hard. A survey by Kwok and Ahmed [1] provides a compilation of majority of heuristics which are largely static in nature and are based on *List Scheduling*. The scheduling algorithms typically address a particular task and processor model, e.g., arbitrary communication/computation execution cost [2], [3], conditional nodes [4], probabilistic execution cost [5], heterogeneous [6], and arbitrarily connected processor models [7]. Among the above models, this work focuses on the scheduling problem of task graphs which are dynamic in nature due to the fact that they are comprised of either conditional tasks which are evaluated at runtime or tasks whose exact execution times are not known in advance. It proposes a new contention aware online scheduling methodology for dynamic task graphs for multiprocessors with limited channel capacity.

In the scheduling problem of task graphs with communication, several methodologies appear in the literature. *Task Duplication Based* (TDB) scheduling algorithms, e.g., Critical Path Fast Duplication (CPFD) by Ahmed and Kwok [8], limited duplication by Bansal et al. [9], and duplication minimization algorithm by Shin et al. [10], duplicate tasks across multiple processors to reduce the amount of communication. Similarly, an optimal scheduling algorithm on arbitrary number of processors for low-communication task graphs has been proposed by Park and Choe [11]. *Clustering-based* solutions cluster a set of nodes involved in large communications and schedule the clustered nodes together in a single processor [12]. The above methods try to minimize the actual communication on the communication channels in the schedule. However, they do not take into account the contention in the communication channel. Sinnen and Sousa [13] have proposed a *contention aware* modified list scheduling scheme for realistic communication subsystem models. The algorithm statically resolves contention by appropriate communication routing and, thus, improves on the classical list scheduling algorithms.

Majority of the above algorithms are based on static list scheduling algorithms guided by the graph structure, execution cost, and communication cost. However, in the presence of conditional tasks, the graph structure varies from one execution to the other depending on the conditions evaluated. Xie and Wolf proposed a static mutual exclusion detection algorithm [4] for efficient processor resource sharing. However, it may be noted that two tasks which are not statically mutually exclusive may not coexist under certain execution conditions at runtime, which cannot be used for processor resource sharing by the above scheme. Moreover, within a macro task, parameters like conditional paths, loops, cache hits, and interrupts lead

- *P. Choudhury is at the B 002, Bharati Towers, Forest Park, Bhubaneswar 751009, Orissa, India. E-mail: pravanjan@gmail.com.*
- *P.P. Chakrabarti and R. Kumar are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, West Bengal, India. E-mail: {ppchak, rkumar}@cse.iitkgp.ernet.in.*

Manuscript received 25 Apr. 2010; revised 23 Oct. 2010; accepted 24 Jan. 2011; published online 17 Mar. 2011.
Recommended for acceptance by F. Petrini.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-04-0248.
Digital Object Identifier no. 10.1109/TPDS.2011.104.

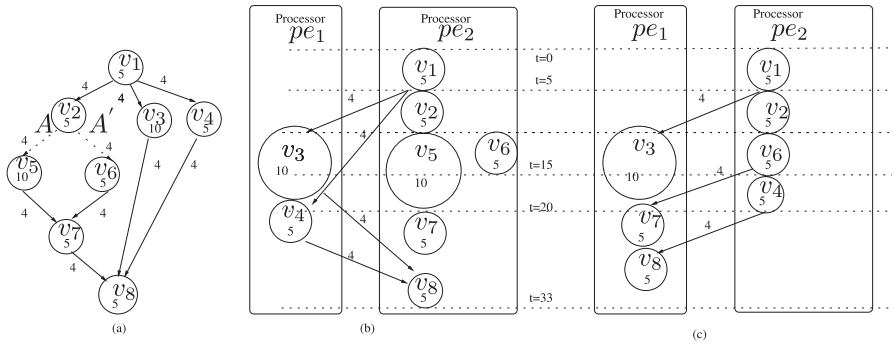


Fig. 1. (a) An example of conditional task graph, (b) static schedule on two processors, and (c) online remapping of v_4.

to unpredictable execution behavior at runtime. Static schedulers often use worst case execution time which may be multiple times greater than the actual execution time [14]. Among the few research works on such variable tasks, Yang et al. [15] have modeled the tasks and communications as stochastic variables and proposed a genetic algorithm to optimize schedule length. Satish et al. [16] have introduced the concept of Statistical Static Level to prioritize tasks and they have presented a Simulated Annealing method for optimizing the deterministic schedule. Wang et al. [17] have modified the list scheduling algorithm to incorporate statistical timing analysis and have shown considerable improvements.

The above static schedules, whether generated by the worst case, average case, or statistical behavior of the tasks, run into the problem of precommitting the schedule to the processors. A task then can only be executed on the assigned processors even if a favorable rescheduling is possible on another processor. Though online scheduling schemes do not have this constraint, they have not been studied extensively due to negligible or marginal improvements for deterministic tasks as well as for their runtime overhead. Among few classically studied methodologies, Feldmann et al. presented a technique for online scheduling of parallel jobs assuming that the task execution time is not known a priori [18]. For reducing the scheduling overhead, low-overhead techniques have been proposed, e.g., the scheduling algorithms proposed by Anderson et al. [19] and Gupta et al. [20]. Intel TBB scheduler [21] uses "breadth-first theft and depth-first work" strategy to maximize utilization of threads. However, the scheduler does not take static priority, communication, and contention into consideration and it is designed for fine-grained tasks with low overhead. Choudhury et al. [22] proposed a mixed static and online approach for such dynamic task graphs without any communication cost which shows improvement over pure static schedulers.

This work focuses on a contention aware online strategy for the scheduling problem of dynamic task graphs onto a multiprocessor system with limited communication capacity. The contribution of this paper is mainly twofold. First, the scheduling problem model is more realistic compared to conventional schemes because the model considers task graphs which are subject to variation at runtime and interprocessor communications of fixed number of channels. Second, we have proposed an online scheduling scheme as an alternative to overcome the above-mentioned limitations of conventional static schedulers for two types communication schemes, namely, broadcast and point-to-point communication schemes. We have compared our results with the contention aware static scheduling presented by Sinnen and Sousa [13] and Xie and Wolf [4]. Experimental results indicate that the proposed methodology shows up to 21.5 percent average schedule length improvement over static schedulers. The effects of graph and processor model parameters like edge density, number of conditional paths, and channel capacity are studied with various experiments.

We explain the motivation of the work in Section 2. The task graph, processor, and scheduling models are included in Section 3. Section 4 presents the online scheduling algorithms for the two types of assumed communication models. Experimental results and comparison with static schedulers are presented in Section 5. Section 6 concludes the work.

2 Motivational Example

Traditionally, static scheduling is preferred over online scheduling due to the assumption of deterministic nature of the tasks. In the setting of dynamic task graphs, a task graph has either conditional execution paths or it is comprised of tasks with variable execution times. In the following sections, we illustrate the motivation of this work by taking two examples of dynamic task graphs considering the above aspects separately. In the first example (Fig. 1a), we consider a task graph with eight nodes comprising of a conditional node v_2 with fixed execution times. We schedule it optimally for the worst case behavior by a static scheduler proposed by Xie and Wolf [4] (Fig. 1b). It can be observed that by making an online remapping of nodes at time $t = 10$ (when v_2 is evaluated), the schedule length can be improved by four units for the case when v_2 is evaluated to A' (Fig. 1c). In the second example (Fig. 2a), we consider a static schedule (Fig. 2b) which is optimal for a task graph with deterministic execution time and with no conditional tasks. We examine two cases: 1) v_3 finishing early (Fig. 2c), and 2) v_2 finishing early (Fig. 2d) and extract the new schedule lengths. It can be observed from Fig. 2e that by an online remapping of v_4, the schedule length improves by three time units. We also observed that for both the examples, no static mapping will be able to provide schedule length improvement for all the cases (Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.104). Thus, an online scheduler capable of remapping of nodes at runtime is required.

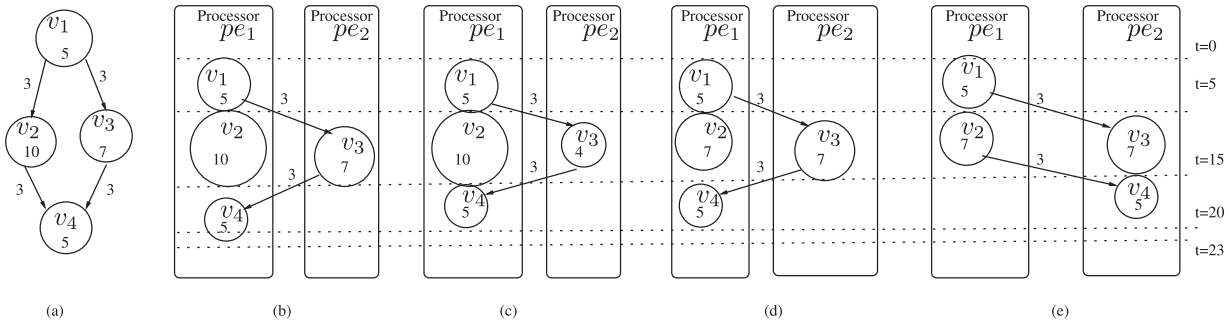


Fig. 2. (a) An example task graph with variable execution times, (b) schedule on two processors, (c) optimal static schedule with v_3 is finishing early, (d) static schedule with v_2 finishing early, and (e) online remapping with v_2 finishing early.

3 Models, Notations, and Definitions

3.1 Task Graph Model

The application task graph model used in this work is called a Conditional Task Graph (CTG). A CTG is a special DAG, $G <V, E>$, where $V = V_S \cup V_C$ represents set of non-preemptive tasks and E represents the communication edges between the tasks. V_S and V_C represent the set of simple and conditional tasks, respectively. When a conditional task is executed, only one path among its outgoing edges is selected for further execution. Each node v_i ($v_i \in V(G)$) can execute between its worst case and best case execution values which are known a priori. The precedence and communication between v_i and v_j is expressed as a directed edge e_{ij}.

- $\mathbf{WCET_{v_i}}$, $\mathbf{BCET_{v_i}}$. *Worst Case and Best Case Execution Time* of v_i.
- $\mathbf{ex_{v_i}}$. Actual execution time of the node v_i. It takes a value between $BCET_{v_i}$ and $WCET_{v_i}$.
- $c(e_{ij})$. Communication cost on edge e_{ij}.
- *ccr*. Average communication to average computation ratio in a task graph

$$\frac{\frac{\sum_{e_{ij} \in E} c(e_{ij})}{|E|}}{\frac{\sum_{v_i \in V} ex_{v_i}}{|V|}}.$$

  If *ccr* of a task graph is low, then it is more computation heavy; otherwise, it is more communication heavy.
- **source**, **sink**. The start and end node, respectively, of a DAG. We assume a single source and a single sink node in a task graph.
- $\mathbf{parent_{v_i}}$, $\mathbf{child_{v_i}}$. Set of immediate parent and children nodes of v_i, respectively.
- **edgedensity**. $|E|$ expressed as a fraction of maximum possible edges with $|G|$ vertices of a task graph. This indicates the density of precedence constraints in the graph.

3.2 Processor Model

Several processor architectures appear in the literature, including homogeneous/heterogeneous processor models [6], and different interprocessor communication models like buses, switches, half/full duplex links, multihop (partially connected) networks [13]. We have chosen a model consisting of homogeneous processors connected by a shared bus with limited channels because they are most common. The processor model is represented by a set of homogeneous processing elements $PE = \{pe_1, pe_2, pe_3, \ldots, pe_{|PE|}\}$ connected via a bus with few channels $B = \{C_1, C_2, \ldots, C_{z(B)}\}$ for interprocessor communication, where $z(B)$ represents the number of independent channels and C_i is the ith communication channel. All the channels can carry only one communication at any point in time and, hence, are subjected to channel contention. They are connected to all the processors, and hence, there is no end-point contention. Every channel has an independent buffer to store the data and initiates the communication from that buffer when it finishes the ongoing communication. We have assumed following two types of communication model in the shared bus.

- Broadcast communication model. In this type of communication model, a channel carrying a communication delivers the data to all the processors.
- Point-to-point communication model. In this type of communication model, a channel carrying a communication delivers the data to a processor specified by the processor initiating the communication.

3.3 Scheduling Notations

A task v is said to be *scheduled* when its start time (ST_v) and processor mapping (pe_v) are allocated.

- $\mathbf{SU_v}$. Static urgency of a node v. It is the maximum length from the start of v to the sink.
- $\emptyset(V)$. An ordering of nodes for the node set V, such that if $SU_{v_i} > SU_{v_j}$ then v_i precedes v_j in the order.
- $\mathbf{drt(v, pe)}$. Data Ready Time. Time t at which v can start execution on a processor pe.
- $\mathbf{PE\_Available\_time(v, pe)}$. Time unit t at which v can start its execution on pe.

4 Proposed Online Scheduling

4.1 Methodology

We present online scheduling algorithms for the two types of communication models described in the processor model in Section 3.2, namely, broadcast and point-to-point communication models. In both the cases, the proposed methodology uses a *Global Scheduler*, which schedules the tasks on processors and edges on the channels.

4.1.1 Global Scheduler Modeling

The design of the global scheduler is essentially the core of the proposed methodology which either schedules a task or

a communication at runtime. The global scheduler is notified by an event when there is a change in any of the processor states. The global scheduler's functionality can be modeled by the following two events.

1. Task completed event. A task completed event is generated by a processor when it completes execution of the assigned task on it. At this event, the global scheduler assigns the outgoing communications of the finished task to channels. Additionally, in the point-to-point communication model, it schedules the child tasks for a future time in a processor.
2. Free processor event. A free event is generated by a processor to notify the global scheduler when it is free and it has received a new communication. At this event, the global scheduler analyzes the processor state and it may assign a task to the processor for execution.

At the start, the global scheduler schedules the source node to one of the processors and then waits for any processor state change event. When the $source$ node finishes execution, it schedules the child tasks of the $source$, the required communications to appropriate resources, and again waits for the events to be generated from the processors. The scheduling continues till the $sink$ node finishes its execution. At any given instant of time, the global scheduler maintains the partial schedule state to make a best possible scheduling decision during handling an event.

The procedures for handling the above two types of events sufficiently explain the proposed online methodology. An illustrative example is given in Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.104, for the proposed methodology.

4.1.2 Online Scheduling under Broadcast Communication Model (Algorithm 1)

In this communication model, when a task is completed, the global scheduler needs to schedule the edges on the channels. Since all the processors receive the communication by the channel, the scheduling decision of the child tasks (targets of the above-mentioned communications) can be deferred to a later time. The edge scheduling is done by selecting the edge that has highest priority based on the priority SU of the child tasks. If the completed task is a conditional task, then only one among the outgoing edges is considered based on the condition evaluated. The selected edge is then allocated to the channel that gives the earliest start time. At the free processor event, the global scheduler analyzes all the communications that have arrived on the processor and selects the task with highest priority for execution, hence confirming to the methodology of ETF-based static scheduling. The event handling details are given in the description of Algorithm 1.

Algorithm 1. Global scheduler for broadcast communication model

Event - Task Completed {v_i completes on p}
Let V_r be $Child_{v_i}$ if $v_i \in V_s$ else $Child_{v_i}$ with exception of nodes based on the conditions evaluated at v_i
while V_r is not empty **do**
Let v_j be the first element in the ordering $\emptyset(V_r)$
Let $C \leftarrow$ the channel in bus B that provides minimum start time (based on on-going and queued communications)
Schedule e_{ij} on C, Remove v_j from V_r
end while
End Event

EVENT - Free Processor Event {A processor pe is free and a communication has arrived}
Let V_r be set of tasks that are ready on processor pe and not scheduled on any other processor
Let v be the first element in the ordering $\emptyset(V_r)$
$ST_v \leftarrow t\{$ Schedule and Start v on $pe\}$
End Event

4.1.3 Online Scheduling under Point-to-Point Communication Model (Algorithm 2)

In this communication model, every edge that is scheduled on a channel must have a destination processor, which in turn means that the target task of that communication has to be tentatively scheduled on that destination processor. Hence, when a task is completed, the global scheduler not only decides to schedule the edges on the channels, but also decides which processor the child task will be executed on for better schedule length, thus making a joint scheduling decision. Therefore, the global scheduler schedules the edges on channels and makes a future scheduling of all the child tasks of the finished task. The actual start time of the task, however, is decided at runtime. To make a proper scheduling decision of a task v_j onto a processor, the global scheduler needs to know the expected time at which the task can start on any given processor. This value depends on the tentative schedules (which again depends on the drt of those tasks) of the already scheduled tasks on that processor. The value also depends on the data ready time drt of the current task (v_j) on that processor. Hence, an accurate estimation of drt is needed to choose a processor for scheduling. For this, the scheduler uses the partial schedule (actual execution time or start time of tasks executed till now) to either accurately calculate or estimate the drt of any task at any given time. The value of $drt(v_j, pe)$ can be calculated as a maximum of edge arrival time $eat(e_{ij}, pe)$ (1) from all the incoming edges of v_j from $parent_{v_j}$

$$eat(e_{ij}, pe) = c(e_{ij}) + ST_{v_i} + ex_{v_i}, \tag{1}$$

$$drt(v_j, pe) = max(eat(e_{ij}, pe)) \forall v_i \in parent_{v_j}. \tag{2}$$

$drt(v_j, pe)$ can be calculated with the equation given in (2). If v_i ($v_i \in parent_{v_j}$) is already started or scheduled in one of the processors, ST_{v_i} is the actual start time of v_i. If v_i is not scheduled, then ST_{v_i} is the as-soon-as-possible (ASAP) time of v_i. Similarly, if v_i is already completed, then ex_{v_i} is the exact execution time of v_i, else it is the expected execution time of v_i. If v_i has already been scheduled and started in the same processor pe, then $c(e_{ij})$ is taken as 0, else it is the communication weight on the edge.

At the free processor event, the global scheduler behaves exactly as before. The event handling details are given in the description of Algorithm 2.

Algorithm 2. Global scheduler for point-to-point communication model
Event - Task Completed {v_i completes on p}
Let V_r be $Child_{v_i}$ if $v_i \in V_s$ else $Child_{v_i}$ with exception of nodes based on the conditions evaluated at v_i
while V_r is not empty **do**
Let v_j be the first element in the ordering $\emptyset(V_r)$
Let $C \leftarrow$ the channel in the bus B that provides minimum start time
if v_j is already scheduled on a processor $pe(v_j)$ **then**
Schedule $e_{ij}, pe(v_j)$ on C {schedule e_{ij} on Channel C intended for processor $pe(v_j)$}
else
Let $pe_k \leftarrow min(PE\_Available\_time(v_j, pe)),\ \forall pe \in PE$ {Select Channel for edge scheduling}
Schedule e_{ij} on C from p to pe_k {schedule e_{ij} on Channel C intended for processor pe_k}
Schedule v_j on pe_k
end if
Remove v_j from V_r
end while
End Event

EVENT - Free Processor Event {A processor pe is free and a communication has arrived}
Let V_r be all the tasks that are ready on processor pe
Let v be the first element in the ordering $\emptyset(V_r)$
$ST_v \leftarrow t$ { Schedule and Start v on pe}
End Event

PROCEDURE - $PE\_Available\_time(v, pe)$ {Calculates the tentative schedule time of Task v on pe}
Calculate Expected data arrival time $drt(v, pe)$
Tentatively schedule v on pe after the time $drt(v, pe)$ such that for any scheduled task v_s after v, $SU_v < SU_{v_s}$
Return expected start time of v on pe
End PROCEDURE

4.2 Complexity and Overhead

The overall complexity of the broadcast communication scheduler for the task completed event and free processor event are $O(|E|z(B))$ and $O(|V||PE|log(|V|))$, respectively. Similarly, the overall complexity of the point-to-point communication scheduler for the task completed event and free processor event are $O(|V|^2)$ and $O(|V|log(|V|))$, respectively. The local processor complexity for storing the ready task list is $O(|V|log(|V|)$. The complexity derivation details can be found in Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.104.

5 Experimental Results

We evaluate the proposed online scheduling algorithm for 1) conditional task graphs with fixed execution time, and 2) unconditional task graphs with nodes of unpredictable execution time, separately.

5.1 Experiments on Random Conditional Task Graphs with Fixed Execution Nodes

5.1.1 Setup

We have modified the random task graph generation model defined by Johnsonbaugh and Kalin [23] to generate random conditional task graphs with communication. The number of nodes ($|V|$) was varied from 100 to 500, and *edge density* from 5 to 50 percent. The communication to computation ratio (ccr) was varied from 0.1 to 10.0. For each combination of parameters, 100 random DAGs were generated. Random worst case execution cost was assigned between 5 and 50 units on each node. The number of conditions $|C|$ (expressed as a fraction of $|V|$) was varied from 2 to 6 percent. This way, we generated 10,000 wide range of random task graphs producing different structures. The effects of different parameters were studied experimentally on the generated task graph set.

5.1.2 Experiments

The schedule length generated by the proposed strategy was compared to the contention-aware static scheduler proposed by Sinnen and Sousa [13]. Since this contention aware scheduler algorithm is not clearly defined for conditional task graphs, we made certain modifications to the static scheduler with the concepts of resource sharing given by Xie and Wolf [4] for comparison. If L_i is the schedule length produced by Sinnen and Sousa [13] with resource reclaim and L'_i is the schedule length produced by the proposed strategy for the ith instance of total of I instances of a task graph G execution, the average case schedule improvement ($Av\_Improv_G(\%)$) is calculated as

$$\frac{\sum_{i=1}^{I} \frac{(L_i - L'_i) \times 100}{L_i}}{I}.$$

The average improvement ($Average\ \%\ Improvement$) for a task graph set of N task graphs is calculated as

$$\frac{\sum_{i=1}^{N} Av\_Improv_{G_i}(\%)}{N}.$$

For all the experiments, we have used the value of I to be 500 or $2^{|C|}$, whichever is lower. In the first experiment, we varied the edge density of the task graph and noted the performance improvements varying the number of conditional nodes ($|C|$ from 2 to 6 percent) (Fig. 3). Then, we varied the ccr to test the schedule performance for task graphs with low communication delay as well as task graphs with high communication delay. For this experiment, we chose task graphs with suitable edge density and limited number of processors to produce a large number of communications at runtime. On the same experiment, we also varied the number of channels ($z(B)$) to obtain the effect of online edge scheduling (Fig. 4). The best case/worst case improvement/degradation was also noted for the entire simulation (Fig. 5).

5.1.3 Results

Fig. 3 shows the improvement on schedule length of the proposed online method over the static scheduler with

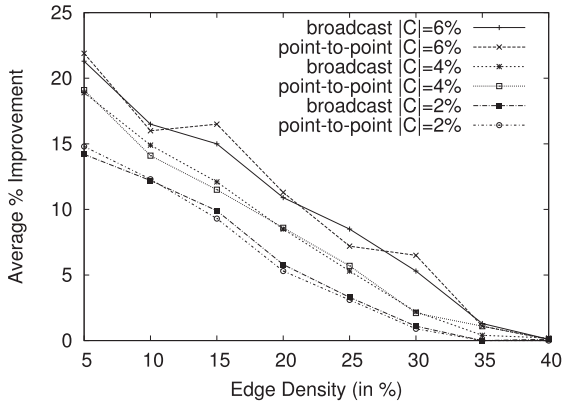


Fig. 3. Schedule length improvement (online over static scheduling in percent) versus edge density (percent of *max edge*) for different number of conditional nodes $|C|$ ($|PE| = 3, z(B) = 3, |V| = 100,\ ccr = 1.0$).

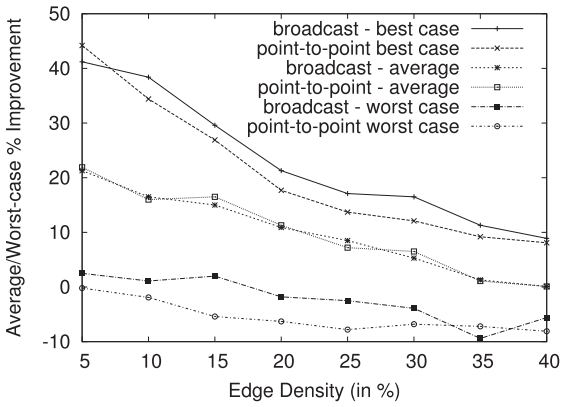


Fig. 5. Average/worst case improvement (online over static scheduling in percent) versus edge density (percent of *max edge*) $|PE| = 3, |V| = 100, |C| = 4\%$.

varying edge density for different $|C|$. We also observed that the proposed scheduler provides identical schedules for both broadcast and point-to-point communication method in about 60 percent of the simulations. However, the point-to-point communication methodology results in a less number of communication (on actual channels) than broadcast-based communication methodology. With an increase in a number of conditional nodes, we observed that the proposed scheduler performance also increases due to the reason that the task graphs becomes more dynamic in nature at runtime. Fig. 3 also shows that the average improvement steadily decreases with an increase in edge density. This is due to the fact that with an increase in edge density, the parallelism in the task graph decreases forcing the tasks to execute in a specific order. This limits the scope of improvement.

Fig. 4 shows the improvement of schedule length versus ccr for different numbers of channel availability. We observed that the point-to-point communication scheduler provides slightly better schedule length compared to the broadcast scheduler when a number of channels are less ($z(B) = 2$). This is because the number of communications are less in the case of the point-to-point communication scheduler. We also observed that the proposed methodology degrades for ccr values beyond 6, giving negligible benefits

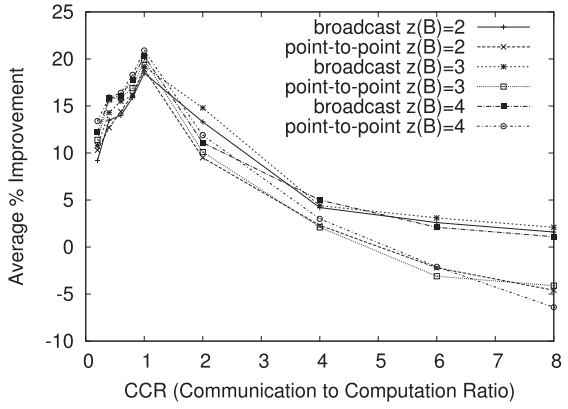


Fig. 4. Schedule length improvement (online over static scheduling in percent) versus ccr for different channel numbers $z(B)$ ($|V| = 200$, $|c| = 2\%$ and edge $\text{density} = 5\%$).

over static scheduler. We presume that, for task graphs with communication cost significantly higher than average computation cost, a cluster-based solution similar to [12] is a better choice than the proposed method (Appendix E, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.104). Additional results in Appendix D, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.104, indicate that the proposed methodology is suitable for the limited number of processors 2-6. Finally, Fig. 5 presents the online scheduler average/worst case/best case performance improvements over static schedule for the whole simulation run for task graphs of different sizes.

5.2 Experiments on Random Unconditional Task Graphs with Variable Execution Nodes

5.2.1 Setup

The randomly generated execution time for each node earlier is considered as worst-case execution time in this setup. We define a parameter $\alpha(0 < \alpha < 1)$ as $\alpha = \frac{BCET_{v_i}}{WCET_{v_i}} \forall v_i \in V$. For each value of α, we calculated the value of BCET for each node. Then, at runtime we assigned a random value between BCET and WCET to each node. It can be noted that if a task graph has a smaller α, the tasks are likely to be more dynamic at runtime.

5.2.2 Experiments

We performed similar experiments as explained in the earlier section with varying α from 0.2 to 0.6. The static scheduler in this case uses worst case execution time of nodes for the static schedule generation with online resource reclaim (since tasks finish earlier). For each graph, we have taken 500 random instances for the calculation of average improvement.

5.2.3 Results

Fig. 6 shows the improvement on schedule length versus edge density for different α. With higher α, the task graph behaves more like a task graph with deterministic execution nodes and, hence, limits the scope of improvement. Fig. 7 shows the improvement of schedule length versus ccr for different numbers of channel availability. Finally, Fig. 8

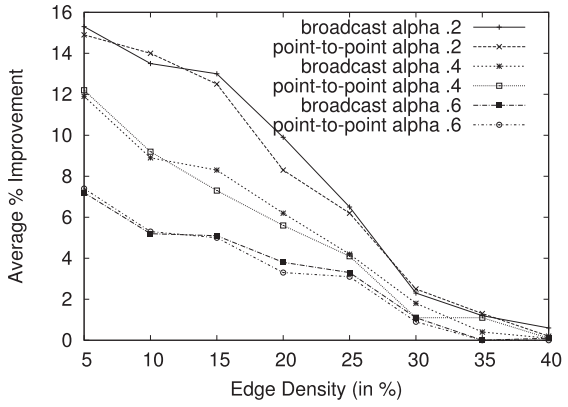


Fig. 6. Schedule length improvement (online over static scheduling in percent) versus edge density (percent of *max edge*) for different alpha α ($|PE| = 3, z(B) = 3, |V| = 100$, $ccr = 1.0$).

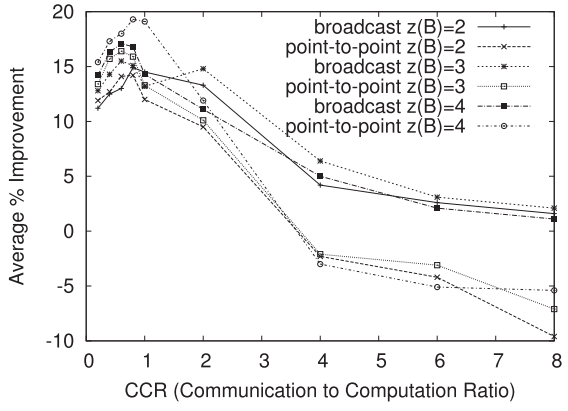


Fig. 7. Schedule length improvement (online over static scheduling in percent) versus ccr for different channel numbers $z(B)$ ($|V| = 200$, $\alpha = 0.5$, and edge $\text{density} = 5\%$).

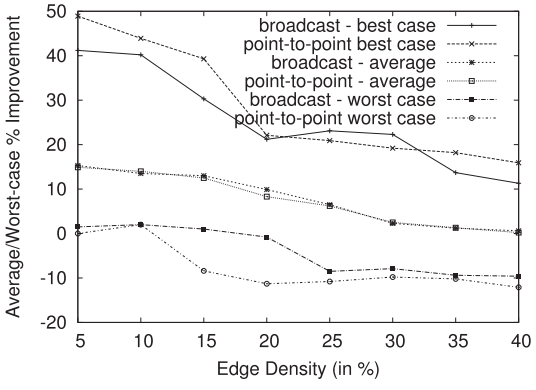


Fig. 8. Average/worst case improvement (online over static scheduling in percent) versus edge density (percent of *max edge*) $|PE| = 3$, $|V| = 100, \alpha = 0.4$.

presents the online scheduler average/worst case/best case performance improvements over static schedule for the whole simulation run for task graphs of different sizes.

5.3 Experiments on Benchmark Task Graphs

In the absence of any publicly available benchmark suites for stochastic task graphs with communications, we evaluated the proposed strategy by modifying the benchmark task graphs given in [24]. We chose five graphs of different structures and assigned different α to those tasks. We found out the optimal static schedule for these five task graphs by searching the whole solution space. Then, we simulated the online scheduler and the schedule generated by the optimal static scheduler for 100 random instances for each of the five graphs. The result of this experiment is tabulated in Table 1.

TABLE 1
Benchmark Applications

Name	$\|V\|$	$\|E\|$	α	Av. Improvement (in %)	
				Broadcast	Point-to-Point
B1	10	10%	0.2	6.2	6.7
B2	20	15%	0.4	7.8	8.6
B3	30	7%	0.6	5.3	3.2
B4	40	20%	0.8	2.3	2.1
B5	50	21%	0.2	2.2	0.6

TABLE 2
Overhead versus Task Graph Size, Edge $\text{Density} = 5\%$

$\|V\|$	Computation Time (Static) (milli-seconds)	Online Cumulative Overhead	
		(Broadcast) (milli-seconds)	(Point-to-Point) (milli-seconds)
100	56	37	68
200	201	126	389
300	364	167	508
400	670	234	641
500	765	450	796

5.4 Computation Time and Overhead Results

The schedulers were simulated on a JAVA virtual Machine running on a 1.4 GHz Intel Centrino Processor with 1 GB RAM. The overall schedule times for the static as well as the online schedulers are tabulated in Table 2. The overhead of the online scheduler in columns 3 and 4 indicates the global scheduler event processing time for the entire graph schedule. It can be noted that the point-to-point communication scheduler has higher cumulative overhead than the broadcast communication scheduler as expected. The cumulative overhead increases as expected with number of nodes; however, overhead per node typically stays in the range of microseconds. This overhead is low and acceptable for coarse-grain tasks, where the execution times are typically significantly higher. The proposed methodology may not be suitable for very fine grain tasks.

6 Conclusions

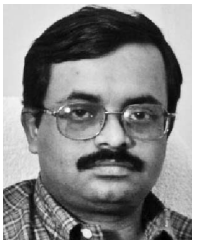
This paper presents a novel online strategy for mapping and scheduling of task graphs comprising tasks with unpredictable execution behavior. The proposed scheduling strategy schedules tasks to processors at runtime based on the execution behavior and performs the edge scheduling to handle contention. Experimental results show that the proposed solution provides better average-schedule-length at runtime over the static schedulers for task graphs with unpredictable execution behavior. The algorithm presented in this paper may be suitably extended for heterogeneous processor models or models with communication uncertainty.

References

[1] Y.-K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computing Surveys,* vol. 31, no. 4, pp. 406-471, 1999.

[2] J.-J. Hwang, Y.-C. Chow, F.D. Anger, and C.-Y. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM J. Computing,* vol. 18, no. 2, pp. 244-257, 1989.
[3] Y.-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors," *IEEE Trans. Parallel and Distributed Systems,* vol. 7, no. 5, pp. 506-521, May 1996.
[4] Y. Xie and W. Wolf, "Allocation and Scheduling of Conditional Task Graph in Hardware/Software Co-Synthesis," *Proc. Conf. Design Automation and Test in Europe (DATE),* pp. 620-625, 2001.
[5] K.M. Chandy and P.F. Reynolds, "Scheduling Partially Ordered Tasks with Probabilistic Execution Times," *Proc. Fifth ACM Symp. Operating Systems Principles (SOSP),* pp. 169-177, 1975.
[6] D.L. Rhodes and W. Wolf, "Co-Synthesis of Heterogeneous Multiprocessor Systems Using Arbitrated Communication," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD),* pp. 339-342, 1999.
[7] H. El-Rewini and T.G. Lewis, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines," *J. Parallel and Distributed Computing,* vol. 9, no. 2, pp. 138-153, 1990.
[8] I. Ahmad and Y.-K. Kwok, "On Exploiting Task Duplication in Parallel Program Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 9, pp. 872-892, Sept. 1998.
[9] S. Bansal, P. Kumar, and K. Singh, "Dealing with Heterogeneity through Limited Duplication for Scheduling Precedence Constrained Task Graphs," *J. Parallel and Distributed Computing,* vol. 65, no. 6, pp. 479-491, 2005.
[10] K. Shin, M. Cha, M. Jang, J. Jung, W. Yoon, and S. Choi, "Task Scheduling Algorithm Using Minimized Duplications in Homogeneous Systems," *J. Parallel Distributed Computing,* vol. 68, no. 8, pp. 1146-1156, 2008.
[11] C.I. Park and T.Y. Choe, "An Optimal Scheduling Algorithm Based on Task Duplication," *IEEE Trans. Computers,* vol. 51, no. 4, pp. 444-448, Apr. 2002.
[12] V. Kianzad and S.S. Bhattacharyya, "Efficient Techniques for Clustering and Scheduling onto Embedded Multiprocessors," *IEEE Trans. Parallel Distributed System,* vol. 17, no. 7, pp. 667-680, July 2006.
[13] O. Sinnen and L.A. Sousa, "Communication Contention in Task Scheduling," *IEEE Trans. Parallel Distributed Systems,* vol. 16, no. 6, pp. 503-515, June 2005.
[14] R. Ernst and W. Ye, "Embedded Program Timing Analysis Based on Path Clustering and Architecture Classification," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '97),* pp. 598-604, 1997.
[15] J. Yang, X. Ma, C. Hou, and Z. Yao, "A Static Multiprocessor Scheduling Algorithm for Arbitrary Directed Task Graphs in Uncertain Environments," *Proc. Eighth Int'l Conf. Algorithms and Architectures for Parallel Processing (ICA3PP '08),* pp. 18-29, 2008.
[16] N.R. Satish, K. Ravindran, and K. Keutzer, "Scheduling Task Dependence Graphs with Variable Task Execution Times onto Heterogeneous Multiprocessors," *Proc. Eighth ACM Int'l Conf. Embedded Software (EMSOFT '08),* pp. 149-158, 2008.
[17] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-Aware Task Allocation and Scheduling for MPSoC," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design,* pp. 598-603, 2007.
[18] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng, "Optimal Online Scheduling of Parallel Jobs with Dependencies," *Proc. 25th Ann. ACM Symp. Theory of Computing (STOC '93)* pp. 642-651, 1993.
[19] R. Anderson, P. Beame, and W. Ruzzo, "Low Overhead Parallel Schedules for Task Graphs," *Proc. Second Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '90),* pp. 66-75, 1990.
[20] R. Gupta, D. Mosse, and R. Suchoza, "Real-Time Scheduling Using Compact Task Graphs," *Proc. 16th Int'l Conf. Distributed Computing Systems (ICDCS '96),* pp. 55-62, 1996.
[21] "The TBB Task Scheduler," http://www.intel.com/technology/itj/2007/v11i4/5-foundations/4-tbb.htm, 2011.
[22] P. Choudhury, R. Kumar, and P.P. Chakrabarti, "Hybrid Scheduling of Dynamic Task Graphs with Selective Duplication for Multiprocessors under Memory and Time Constraints," *IEEE Trans. Parallel and Distributed Systems,* pp. 967-980, 2008.
[23] R. Johnsonbaugh and M. Kalin, "A Graph Generation Software Package," *Proc. 22nd SIGCSE Technical Symp. Computer Science Education (SIGCSE),* pp. 151-154, 1991.
[24] "Task Graphs for Free 3.0," http://ziyang.eecs.umich.edu/~dickrp/tgff/download.html, 2011.

Pravanjan Choudhury received the BTech degree in instrumentation engineering from the Indian Institute of Technology (IIT) Kharagpur, Kharagpur, India, in 2002. Currently, he is working toward the PhD degree in computer science and engineering at IIT Kharagpur. He is also working as a software architect at Minekey, Inc. Previously, he has worked as a research consultant for CAD research groups at IIT Kharagpur and as a senior R&D engineer at Himachal Futuristic Communication Limited. His research interests include multiprocessor systems, scheduling, CAD, and embedded systems.



P.P. Chakrabarti received the BTech and PhD degrees in computer science and engineering from the Indian Institute of Technology (IIT) Kharagpur, Kharagpur, India, in 1985 and 1988, respectively. He joined the Department of Computer Science and Engineering, IIT, as a faculty member in 1988 and is currently a professor of the Computer Science and Engineering Department. He is the fellow of Indian National Science Academy, New Delhi, Indian Academy of Science, Bangalore, and the West Bengal Academy of Science and Technology. Currently, he holds the position of dean (Sponsored Research and Industrial Consultancy). His research interests include artificial intelligence, CAD for VLSI, and algorithm design. He has received the President of India Gold Medal, the Swarnajayanti Fellowship, and the Shanti Swarup Bhatnagar Prize from the Government of India, for his contributions. He is a senior member of the IEEE.

Rajeev Kumar received the MTech degree from the University of Roorkee (now the Indian Institute of Technology Roorkee) in 1992 and the PhD degree from the University of Sheffield in 1997, both in computer science and engineering. He is a professor of computer science and engineering at the Indian Institute of Technology (IIT) Kharagpur, Kharagpur, India. Prior to joining IIT, he worked for the Birla Institute of Technology & Science (BITS), Pilani and Defence Research & Development Organization (DRDO). His areas of interest include multimedia and embedded systems, programming languages and software engineering, and multiobjective combinatorial optimization. He is a senior member of the IEEE.