Edge Scheduling Algorithms in Parallel and Distributed Systems

Jian-Jun Han *Duo-Qiang Wang*
(School of Computer, Huazhong University of Science & Technology, China)
han_j_j@163.com

Abstract

Many research efforts have been done in the domain of static scheduling algorithms based on DAG. However, most of these literatures assume that all processors are fully connected and receive communication data concurrently, while ignoring the contentions and delays on network links in real applications, which leads to low efficiency. This paper focuses on the issue of edge scheduling for dependent task set in parallel and distributed environment. Combined with conventionally efficient heuristics, two contention-aware scheduling algorithms are proposed in the paper: OIHSA (Optimal Insertion Hybrid Scheduling Algorithm) and BBSA (Bandwidth Based Scheduling Algorithm). Both the proposed algorithms start from the inherent characteristic of the edge scheduling problem, and select route paths with relatively low network workload to transfer communication data by modified routing algorithm. OISHA optimizes the start time of communication data transferred on links in form of theorems. BBSA exploits bandwidth of network links fully to transfer communication data as soon as possible. Therefore, the makespan yielded by our algorithms can be reduced efficiently. Moreover, the proposed algorithms adapt to not only homogeneous systems but also heterogeneous systems. The experiment results indicate that the proposed algorithms obviously outperform other algorithms so far in terms of makespan.

1. Introduction

Efficient scheduling of tasks is a critical issue for achieving high performance in parallel and distributed systems. The goal of scheduling algorithm is to map tasks to processors and order their executions to satisfy task precedence relationship and make minimum schedule length (*makespan*). Since the general scheduling algorithm is NP-complete [1], many research efforts have been made in this rich field, i.e., list scheduling [2,3], cluster-based algorithms [4], genetic algorithms [5], simulated annealing algorithms [6], duplication-based algorithms [7], fuzzy scheduling [8], branch and bound algorithm [9], *etc.*. Among these scheduling algorithms, list-scheduling algorithm has been shown to have a good trade-off between performance and cost, and static scheduling outperforms dynamic scheduling in most cases when static conditions are known *a priori*. Therefore, many efforts have been done in the domain of static scheduling algorithms using list scheduling. Nevertheless, most of these literatures have in common that they employ an idealized model of the target system. In this model, each processor is a dedicated communication subsystem, the processors are fully connected, and all communications between tasks can be dealt with concurrently. However, intuition suggests that these assumptions are not met on real parallel and distributed systems [10]. In real applications, when a resource is occupied by one communication, any other communication requiring the same resource has to wait until it is released. Thus, conflicts and contentions among communication generally result in a higher execution time. Since contentions and delays on network links play great impact on the scheduling performance, contention-aware scheduling algorithms need to be researched further in order to meet currently complicated network computing environment. Oliver Sinnen *et al*. propose a new contention-aware scheduling model and a corresponding algorithm. Moreover, that paper proves that the edge scheduling problem is NP-Completeness [11].

The algorithms proposed in this paper are based on the scheduling model presented by [11], and focus on the efficient heuristics of edge scheduling oriented to contentions and delays of network communication. Since the communication transfer is involved in edge scheduling, contention-aware scheduling algorithms should be coupled with network communication scheme and be dynamic. This paper proposes two

dynamically contention-aware algorithms, OIHSA (Optimal-Insertion Hybrid Scheduling Algorithm) and BBSA (Bandwidth-Based Scheduling Algorithm). Both two algorithms start from the inherent characteristics of edge scheduling problem, and select route paths with relatively low network workload to transfer communication data by modified routing algorithm. OISHA optimizes the start time of communication data transferred on links in form of theorems. BBSA utilizes bandwidth of links on network fully to transfer communication data as fast as possible. Thus, the *makespan* can be reduced efficiently by our algorithms.

2. Scheduling model

Before describing the scheduling model, some notations and symbols used throughout this paper are listed in Table 1.

Table 1. The symbols and notations

V	Task set in graph G
n_i	Task i in V
$pred(n_i)$	Predecessor set of n_i
$succ(n_i)$	Successor set of n_i
$e_{i,j}$	Edge from n_i to n_j in G
$w(n_i)$	The computation cost of n_i
$c(e_{i,j})$	The communication cost of $e_{i,j}$
P_i	The *ith* processor in processor set P
$s(P_i)$	The processing speed of P_i
L_i	The *ith* link of route path R
$s(L_i)$	The communication data transfer speed of L_i
MLS	The average transfer speed of network links
$t_s(n_i,P_j)$	The earliest start time of n_i on P_j
$t_f(n_i,P_j)$	The finish time of n_i on P_j
$t_{dr}(n_i,P_j)$	The data ready time of n_i on P_j
$t_f(P_i)$	The current finish time of P_i
$TS_{i,j}$	The *jth* occupied time slot on L_i
$t_s(TS_{i,j})$	The start time of $TS_{i,j}$
$t_f(TS_{i,j})$	The finish time of $TS_{i,j}$
$rbr(TS_{i,j})$	The remaining bandwidth rate on $TS_{i,j}$
$br(e_{x,y},TS_{i,j})$	The bandwidth rate used by $e_{x,y}$ on time slot $TS_{i,j}$.
$occupy(TS_{i,j})$	The edge occupying $TS_{i,j}$
$fs(L_i)$	The first occupied time slot on L_i
$ls(L_i)$	The last occupied time slot on L_i
$t_s(e_{i,j})$	The start time of $e_{i,j}$
$t_{es}(e_{i,j},L_m)$	The earliest start time of $e_{i,j}$ on L_m
$t_s(e_{i,j},L_m)$	The virtual start time of $e_{i,j}$ on L_m
$t_f(e_{i,j},L_m)$	The finish time of $e_{i,j}$ on L_m
$\mathrm{int}(e_{i,j},L_m)$	The execution time of $e_{i,j}$ on L_m, that is, $\frac{c(e_{i,j})}{s(L_m)}$
$proc(n_i)$	The processor to which n_i is allocated
$sl(e_{i,j})$	The first link by which $e_{i,j}$ passes
$dl(e_{i,j})$	The last link by which $e_{i,j}$ passes
$PL(e_{i,j},L_m)$	The previous route link before $e_{i,j}$ traverses by L_m ($L_m \neq sl(e_{i,j})$)
$NL(e_{i,j},L_m)$	The next route link after $e_{i,j}$ traverses by L_m ($L_m \neq dl(e_{i,j})$)
$\|M\|$	The element number of set M

2.1. The graph of tasks

The topology of dependent tasks is represented by a DAG (Directed Acyclic Graph) $G=(V,E,w,c)$, where the task set and edge set are denoted by V and E, respectively. The predecessor set and successor set of task n_i with computation cost $w(n_i)$ in V are denoted by $pred(n_i)$ and $succ(n_i)$, respectively. There exists an edge $n_i \to n_j \in E$ with the communication cost $c(e_{i,j})$, if and only if $n_i \in pred(n_j)$ and $n_j \in succ(n_i)$. Only all predecessors of n_i finish their executions and the processor to which task n_i is assigned completes receiving all communication data from $pred(n_i)$, can n_i start to execute. The data ready time $t_{dr}(n_i,P_j)$ of node n_i on processor P_j is defined as the time when the last data from its parent nodes arrives, that is, $t_{dr}(n_i,P_j)=\max\{t_f(e_{k,i})\}$ for $\forall e_{k,i}\in E$, where $t_f(e_{k,i})$ is the time P_j completes receiving data from n_k. For each n_i and P_j, the earliest start time of n_i on P_j, $t_s(n_i,P_j)=\max\{t_{dr}(n_i,P_j),t_f(P_j)\}$. Assume that if two tasks are assigned to the same processor, the communication time between them is ignored. In this model, no task can preempt other tasks' executions,

that is, the following inequalities hold: $proc(n_i) = proc(n_j) = P_k \Rightarrow t_f(n_i, P_k) \le t_s(n_j, P_k)$ or $t_f(n_j, P_k) \le t_s(n_i, P)$.

The static priority of tasks is presented by bottom-level *bl*, which is the length of the longest path leaving the task. Recursively defined, it is $bl(n_i) = w(n_i) + \max\{c(e_{i,j}) + bl(n_j)\}$ for $\forall n_j \in succ(n_i)$.

2.2. The topology graph of network

For traditional scheduling algorithm, it is assumed that if $proc(n_i) = proc(n_j) = P_k$ where $n_j \in succ(n_i)$, then $t_f(e_{i,j}) = t_f(n_i, P_k)$, otherwise, $t_f(e_{i,j}) = t_f(n_i, proc(n_i)) + \frac{c(e_{i,j})}{s(proc(n_i), proc(n_j))}$ where $s(proc(n_i), proc(n_j))$ is transfer speed of direct link between $proc(n_i)$ and $proc(n_j)$. However, we point out that this model is not well suited to modern network in Section 1. For this reason, the model proposed in [11] is described in a more reasonable way.

The topology of a communication network is modeled as a graph $TG = \{N, P, D, H\}$, where *N* is the node set of network including processor and switch, *P* is the processor set, *D* is the directed communication link set, and *H* is the set of hyperedges (multidirectional communication link). Let $L = D \bigcup H$ be the communication link set of network.

Cut-through routing is assumed in [11] rather than stored-and-forward. Since BA (Basic Algorithm) [11] called in this paper does not consider the possible division of communication into packets, circuit switching is assumed. In cut-through routing, a station immediately forwards the data to the next station-the message "cuts through" the station. With every hop that a message or packet takes along its route through the network, a delay might occur. This delay is typically very small, usually it takes only a few network cycles. For this reason, the hop delay is neglected in edge scheduling for simplicity, but it can be included if necessary. Assume that each communication execution does not preempt the others, that is, $t_f(e_{i,j}, L_k) \le t_s(e_{m,n}, L_k)$ or $t_f(e_{m,n}, L_k) \le t_s(e_{i,j}, L_k)$ holds for $\forall e_{i,j}, e_{m,n}, L_k$. For any an edge $e_{i,j}$ and its route path $R = < L_1, L_2, ..., L_l >$, *link causality condition* should be met to approximate real packet-based communication. Namely, $t_{es}(e_{i,j}, L_{k-1}) \le t_{es}(e_{i,j}, L_k)$ and $t_f(e_{i,j}, L_{k-1}) \le t_f(e_{i,j}, L_k)$ hold for $\forall L_{k-1} \ne dl(e_{i,j})$.

Lemma 1. Assume that $e_{i,j}$ traverses route link L_m and the next route link is L_{m+1}. If the earliest start time of $e_{i,j}$ on L_{m+1} is $t_{es}(e_{i,j}, L_{m+1})$, then $t_f(e_{i,j}, L_{m+1}) = \max\{t_f(e_{i,j}, L_m), t_{es}(e_{i,j}, L_{m+1}) + \text{int}(e_{i,j}, L_{m+1})\}$ (1), where $\text{int}(e_{i,j}, L_{m+1})$ is the execution time of $e_{i,j}$ on L_{m+1}.

Proof. It can be easily derived from link causality condition. □

It can be found that $t_{es}(e_{i,j}, L_{m+1})$ is exactly the start time of $e_{i,j}$ on L_{m+1}. Nevertheless, the execution of $e_{i,j}$ usually can not utilize the bandwidth of L_{m+1} fully in heterogeneous systems. To approximate real packet-based communication in heterogeneous systems, we define the start time of $e_{i,j}$ on L_{m+1} by $t_s(e_{i,j}, L_{m+1})$, which is $\max\{t_{es}(e_{i,j}, L_{m+1}), t_f(e_{i,j}, L_{m+1}) - \text{int}(e_{i,j}, L_{m+1})\}$. That is, $t_s(e_{i,j}, L_{m+1})$ is the virtual start time of $e_{i,j}$ on L_{m+1}. It can be found that the execution of $e_{i,j}$ can fully utilize bandwidth of L_{m+1} from the time $t_s(e_{i,j}, L_{m+1})$ without contradicting the link causality condition because $t_s(e_{i,j}, L_{m+1})$ is always no smaller than $t_{es}(e_{i,j}, L_{m+1})$ by their definitions.

3. Basic Algorithm (BA)

Based on the scheduling model, Sinnen *et al.* propose Basic Algorithm (BA) [11].

Algorithm 1.

1: Sort tasks into list *L*, according to static priority scheme and precedence constraints.

2: **for** each $n_i \in L$ **do**

3: Find processor $P_k \in P$ that allows earliest finish time of n_i.

4: Schedule n_i on P_k.

BA employs minimal routing, which means it chooses the shortest possible path, in terms of number of edges, through the network for every communication. Given the graph-based representation of the network, finding a shortest path can be accomplished with a Breadth First Search (BFS) algorithm. Once the route path *R* for an edge $e_{i,j}$ has been found, BA searches idle time interval on each route link without contradicting link causality condition.

We call this algorithm as basic insertion algorithm in this paper.

4. OIHSA

This paper proposes two edge scheduling algorithms. The first algorithm proposed is Optimal Insertion Hybrid Scheduling Algorithm. The core idea of OIHSA comprises of four aspects: the choice of processor for ready task, the determination of the priorities of edges, the modified routing algorithm, the edge scheduling on links.

4.1. The choice of processor for ready task

Generally, the edge scheduling algorithm falls into the scope of dynamic scheduling. The reason is that static choice of a processor for a task could not reflect the real network situations and the contentions on network links play a critical role in scheduling performance as described in Section 1. From Algorithm 1, the start time of the communication data from predecessors to the ready task is all the same, that is, the finish time of the predecessor which finishes latest at runtime. BA chooses the processor for ready task by whether the processor can provide the earliest finish time of the task or not, while ignoring the effect of edge communication on scheduling performance. OIHSA uses the heuristic similar to static scheduling algorithm to assign the ready task n_i to the processor P_k, which satisfies the following criterion:

$$\max_{\forall n_j \in pred(n_i)} \{t_f(n_j, proc(n_j)) + \frac{c(e_{j,i})}{MLS}, t_f(P_k)\} + \frac{w(n_i)}{s(P_k)} =$$

$$\min_{\forall P_m \in P} \{\max_{\forall n_j \in pred(n_i)} \{t_f(n_j, proc(n_j)) + \frac{c(e_{j,i})}{MLS}, t_f(P_m)\} + \frac{w(n_i)}{s(P_m)}\}$$

4.2. The choice of priority of edges

As described in Section 4.1, for any a ready task, each edge from all of its predecessors has the same start time. Obviously, the different scheduling sequence of edges leads to different scheduling quality. Intuitively, the edge with a larger cost dominates the start time of the ready task due to the fact that the edge with small cost still has opportunity to search an earlier idle time interval on the same link even if the edge with large cost occupies the idle time slot in advance, but not vice versa. For an edge $e_{i,j}$, after the route path $R = < L_1, L_2, ..., L_l >$ has been determined (see Section 4.3), a probing message can be sent to traverse the route path to determine and reserve time slot for $e_{i,j}$ on each route link by optimal insertion algorithm (see Section 4.4). The edge transfer sequence is strictly in accordance with the cost of edges. Thus, the edge with higher priority determines its route path and reserve time slot earlier than the edge with lower priority on the same link, which possibly shortens the start time of successive tasks.

4.3. Modified routing algorithm

BA adopts classical BFS to search route path according to the criterion of smallest number of links between source processor and target processor. However, generally, the shortest physical distance does not mean the most suitable route path because BFS neglects the real workload of network. OIHSA utilizes modified Dijkstra algorithm to find suitable route path. The minimal criterion is the finish time of edge on each link by basic insertion algorithm (see Section 3).

4.4. Optimal insertion algorithm

Unlike BA, OIHSA uses the optimal insertion policy for the edges when they are allocated with idle time slots on route links. As described in Section 4.2, each edge reserves time slot to ensure that the edge with higher communication cost has a higher priority to choose earlier idle time interval. Thus, for any a scheduled edge $e_{i,j}$ stalled on L_m, because its start time on link $NL(e_{i,j}, L_m)$ has been known and $t_{es}(e_{i,j}, NL(e_{i,j}, L_m))$ is no sooner than $t_{es}(e_{i,j}, L_m)$, $e_{i,j}$ can defer its communication transfer to produce a longer idle time interval without contradicting link causality condition on link L_m and link $NL(e_{i,j}, L_m)$ (see Lemma 2). Thereby, the task scheduler based on OIHSA should be assisted by some kinds of packet delay schemes (e,g., all-optical packet switching [12]). Note that the duration that can be deferred is also constrained by the edges with a later start time than $e_{i,j}$ on the same link because of the non-preemption of edge executions on network links.

Lemma 2. The longest deferrable time of $e_{i,j}$ on link L_m can be $\min\{t_{es}(e_{i,j}, NL(e_{i,j}, L_m)) - t_{es}(e_{i,j}, L_m), t_f(e_{i,j}, NL(e_{i,j}, L_m)) - t_f(e_{i,j}, L_m)\}$.

Proof. It is feasible that $t_{es}(e_{i,j}, L_m) = t_{es}(e_{i,j}, NL(e_{i,j}, L_m))$ by the first requirement of link causality condition. Meanwhile it is also feasible that $t_f(e_{i,j}, L_m) = t_f(e_{i,j}, NL(e_{i,j}, L_m))$ by the second requirement of link causality condition. Thus, the conclusion of Lemma 2 is true. □

Proposition 1. Each edge scheduling algorithm tries to allocate the earliest idle time intervals for edges.

Namely, each edge allocated with time slot can not start its communication earlier.

In light of Proposition 1, the start time of the edges that have occupied time slots can only be postponed. Thus, OIHSA searches the largest idle time interval from tail to head on the network link queue, which makes the edge start as early as possible. Only if the appropriate position has been found, can relevant time slots be adjusted.

Suppose that the edge $e_{x,y}$ is to be scheduled on link L_m. The variable, *dt*, is denoted as the longest deferrable time of edge $e_{i,j}$ scheduled on L_m only taking link causality condition into consideration. Namely, $dt = \min\{t_{es}(e_{i,j}, NL(e_{i,j}, L_m)) - t_{es}(e_{i,j}, L_m), t_f(e_{i,j}, NL(e_{i,j}, L_m)) - t_f(e_{i,j}, L_m)\}$.

It is easy to obtain that $dt=0$ when $dl(e_{i,j}) = L_m$ or $t_{es}(e_{i,j}, L_m) = t_{es}(e_{i,j}, NI(e_{i,j}, L_m))$ or $t_f(e_{i,j}, L_m) = t_f(e_{i,j}, NI(e_{i,j}, L_m))$.

The variable, *symbol*, records the time slot $TS_{m,n}$ provided that there is enough time to execute $e_{x,y}$ between $TS_{m,n}$ and $TS_{m,n-1}$. The initial value of *symbol* is null. We define *accum* as the largest accumulated deferrable time so far. We can obtain $accum = \min\{dt, accum + t_s(TS_{m,n+1}) - t_f(TS_{m,n})\}$ (2) when testing occupied time slot $TS_{m,n}$. The term $accum + t_s(TS_{m,n+1}) - t_f(TS_{m,n})$ in (2) means the longest accumulated deferrable time if the edges occupying $TS_{m,n+1}, \ldots, ls(L_m)$ delay their executions. The edge $e_{x,y}$ can find idle time interval to insert before $TS_{m,n}$, only and only if $\max\{t_f(e_{x,y}, PL(e_{x,y}, L_m)), \max\{t_f(TS_{m,n-1}), t_{es}(e_{x,y}, PL(e_{x,y}, L_m))\} + \text{int}(e_{x,y}, L_m)\} \le accum + t_s(TS_{m,n})$ (3), where $t_{es}(e_{x,y}, PL(e_{x,y}, L_m)) = t_s(e_{x,y})$ if $sl(e_{x,y}) = L_m$, $t_f(TS_{m,n-1}) = 0$ if $TS_{m,n} = fs(L_m)$, and $t_s(TS_{m,n+1}) = +\infty$ if $TS_{m,n} = ls(L_m)$.

An array, *symbol*1, records the time slots that the current variable *accum* is zero when they are being tested. If all occupied time slots on link L_m have been tested and $symbol = null$, $e_{x,y}$ is append to the tail of queue of L_m.

According to our algorithm, maybe there are more than one idle time interval suitable for edge $e_{x,y}$. OIHSA provides idle time interval for $e_{x,y}$ before the time slot recorded by *symbol* since the variable *symbol* keeps track of the newest appropriate idle time interval, which ensures that $e_{x,y}$ can be scheduled on L_m as soon as possible.

Property 1. The idle time interval available to the edge $e_{x,y}$ must be between two adjacent occupied time slots.

Proof. It can be derived directly from the non-preemption of edge execution on route link assumed in scheduling model. □

Theorem 1. The OIHSA proposed is the optimal insertion algorithm under the assumptions in this paper.

Proof. It is easy to derive that the optimal insertion algorithm should meet two requirements: finding the largest idle time slot and the earliest start time for edge's execution. The proof of first requirement proceeds by induction. By Property 1, the available idle time slot should be within two adjacent occupied time slots. Of course, the largest idle time slot is unexceptional. By Proposition 1, the longest idle time interval between $TS_{m,n}$ and $TS_{m,n-1}$ is determined by deferrable time produced by $TS_{m,n}$, that is the value of *accum*. It is easy to derive that the value of *accum* is optimal for $ls(L_m)$. Assume that the value of *accum* is optimal when searching $TS_{m,k} (1 < k < ls(L_m))$. Thus the largest idle time interval between $TS_{m,k}$ and $TS_{m,k-1}$ is also maximal between the deferrable time of $TS_{m,k}$ constrained by link causality condition and the spare time after $TS_{m,k}$ is postponed, which is the formula (2). Since the value of *accum* for $TS_{m,k}$ is optimal, the value of *accum* for $TS_{m,k-1}$ is obviously optimal.

The variable *symbol* keeps track of the newest appropriate idle slot, so that the appropriate idle time slot has the earliest start time if there exists such an idle time interval. □

From another point of view, we can obtain that the Proposition 1 is reasonable by Theorem 1 because each edge is allocated with the earliest time slot.

Once the optimal insertion algorithm has found the idle time interval before $TS_{m,n}$ for edge $e_{x,y}$ on link L_m, the corresponding occupied time slots should adjust their start time and finish time.

We define the time slot $near(TS_{m,n}, symbol1)$, whose start time is later than that of $TS_{m,n}$, as the time slot nearest to $TS_{m,n}$ that makes *accum* be 0. Since variable *symbol*1 records the time slot that can not be deferred, the time slot $TS_{m,k}$ called affected time slot, where *k* is smaller than the label of $near(TS_{m,n}, symbol1)$ and is larger than or equal to the label of $TS_{m,n}$, will be adjusted accordingly. And the labels of remaining time slots keep unchanged. Therefore, the scheduling cost can also be reduced.

The adjustment of affected time slots is described as bellows. First, the labels of these effected time slots are modified accordingly. Next, the start time and the finish time of the effected time slots should be adjusted. Assume the edge $e_{x,y}$ is being scheduled and the variable *symbol* is equal to $TS_{m,n}$, we can obtain

$t_{es}(e_{x,y}, L_m) = \max\{t_f(TS_{m,n-1}), t_{es}(e_{x,y}, PL(e_{x,y}, L_m))\}$,

$t_f(e_{x,y}, L_m) = \max\{t_f(e_{x,y}, PL(e_{x,y}, L_m)), t_{es}(e_{x,y}, L_m) + \text{int}(e_{x,y}, L_m))\}$,

$t_s(e_{x,y}, L_m) = t_f(e_{x,y}, L_m) - \text{int}(e_{i,j}, L_m)$,

$t_s(TS_{m,n}) = t_s(e_{i,j}, L_m)$, and $t_f(TS_{m,n}) = t_f(e_{i,j}, L_m)$.

For $\forall k > n$ and $t_s(TS_{m,k}) \le t_s(near(TS_{m,n}, symbol1))$,

$x = t_f(TS_{m,k}) - t_s(TS_{m,k})$,

$t_s(TS_{m,k}) = \max(t_f(TS_{m,k-1}), t_s(TS_{m,k}))\}$,

$t_f(TS_{m,k}) = t_f(TS_{m,k}) + x$.

Finally, the earliest start time, the start time, and the finish time of edges occupying these effected time slots should be adjusted accordingly by their definitions.

5. BBSA

BBSA starts from another point of network communication, that is, bandwidth. In this paper, we assume that the link bandwidth is linearly related to the data transfer speed of the link. BBSA proposed tries to transfer edge communication as early as possible by fully exploiting the bandwidth of network links without contradicting the link causality condition. We define the remaining bandwidth rate provided by $TS_{i,j}$ as $rbr(TS_{i,j})$, the bandwidth rate used by $e_{x,y}$ on time slot $TS_{i,j}$ as $br(e_{x,y}, TS_{i,j})$. Since any edge communication is generally distributed to diverse time slots with various bandwidth utilization rates, the idle time interval can be treated as the same as the occupied time slot with the exception that the remaining bandwidth rate of the idle time interval is 100 percent.

Lemma 2. Assume that edge $e_{x,y}$ traverses link L_m and L_{m+1}, and one time slot occupied by $e_{x,y}$ on L_m is $TS_{m,n}$ with the bandwidth utilization rate $br(e_{x,y}, TS_{m,n})$. For any time slot $TS_{m+1,k}$ where $TS_{m+1,k}(t_f(TS_{m+1,k}) > t_s(TS_{m,n}))$, the $br(e_{x,y}, TS_{m+1,k})$ is $\min\{rbr(TS_{m+1,k}), \frac{br(e_{x,y}, TS_{m,n})}{s(L_{m+1})/s(L_m)}\}$ (4).

Proof. First, we do not consider $rbr(TS_{m+1,k})$. If $s(L_{m+1}) \le s(L_m)$, $br(e_{x,y}, TS_{m+1,k})$ is $\max\{1, \frac{br(e_{x,y}, TS_{m,n})}{s(L_{m+1})/s(L_m)}\}$. Otherwise, $br(e_{x,y}, TS_{m+1,k})$ is $\frac{br(e_{x,y}, TS_{m,n})}{s(L_{m+1})/s(L_m)}$. Taking $rbr(TS_{m+1,k})$ into account and $rbr(TS_{m+1,k}) \le 1$, formula (4) obviously holds true. □

Theorem 3. The presumption is the same as Lemma 2. The bandwidth utilization rate $br(e_{x,y}, TS_{m+1,k})$ computed by (4) does not contradict link causality condition.

Proof. For the communication of $e_{x,y}$ on $TS_{m,n}$ with x time unites, the communication volume is $x \cdot s(L_m) \cdot br(e_{x,y}, TS_{m,n})$. Thus, in order to finish the same communication volume on $TS_{m+1,k}$, the time spent is $y = \frac{x \cdot s(L_m) \cdot br(e_{x,y}, TS_{m,n})}{br(e_{x,y}, TS_{m+1,k}) \cdot s(L_{m+1})}$. By using (4), we can obtain $y \ge \frac{x \cdot s(L_m) \cdot br(e_{x,y}, TS_{m,n})}{\frac{br(e_{x,y}, TS_{m,n})}{s(L_{m+1})/s(L_m)} \cdot s(L_{m+1})}$

$\Rightarrow y \ge x$. □

Theorem 4. The presumption is the same as Lemma 2, the execution time of $e_{x,y}$ on $TS_{m+1,k}$ is $\frac{\min\{\text{int}(e_{x,y}, TS_{m,n}) \cdot s(L_m) \cdot br(e_{x,y}, TS_{m,n}), (t_f(TS_{m+1,k})}{s(L_{m+1}) \cdot br(e_{x,y}, TS_{m+1,k})} - \frac{\max\{t_s(TS_{m,n}), t_s(TS_{m+1,k})\}) \cdot s(L_{m+1}) \cdot br(e_{x,y}, TS_{m+1,k})\}}{s(L_{m+1}) \cdot br(e_{x,y}, TS_{m+1,k})}$ (5).

Proof. It can be easily found that the communication volume of $e_{x,y}$ on time slot $TS_{m,n}$ is $\text{int}(e_{x,y}, TS_{m,n}) \cdot s(L_m) \cdot br(e_{x,y}, TS_{m,n})$. By link causality condition, the start time of $e_{x,y}$ on $TS_{m+1,k}$ is $\max\{t_s(TS_{m,n}), t_s(TS_{m+1,k})\}$, so the capacity provided by $TS_{m+1,k}$ for $e_{x,y}$ is $(t_f(TS_{m+1,k}) - \max\{t_s(TS_{m,n}), t_s(TS_{m+1,k})\}) \cdot s(L_{m+1}) \cdot br(e_{x,y}, TS_{m+1,k})$. Therefore, (5) obviously holds true. □

When scheduling $e_{x,y}$ on $TS_{m+1,k}$, the remaining bandwidth rate of $TS_{m+1,k}$ should be adjusted accordingly. Provided that $TS_{m+1,k}$ can not afford enough communication capability for the communication volume of $e_{x,y}$ on $TS_{m,n}$, the remaining communication volume should be transferred by successive time slots on L_{m+1}. Otherwise, $TS_{m,n}$ should be divided into several time slots, each of which has diverse remaining bandwidth rate. The edge scheduling of $e_{x,y}$ stops until all communication volume distributed in the time slots of source link completes on the last link of route path.

6. Experiment results

Simulation experiments are used in the paper to compare our algorithms with relevant algorithms. Let U(i,j) be an uniformly distributed integer in the range of [i, j]. Let the number of processors be 2, 4, 8, 16, 32, 64, 128, the number of tasks in task set U(40, 1000). The speed of processors and link speed in network are U(1, 10), and the computation cost of tasks and communication cost between tasks are U(1, 1000). CCR (Communication– Computation–Ratio) is defined as 0.1-10.0. The construction of task graph is subject to literature [3]. We assume that each switch connects with U[4, 16] processors and there exists a path between any a pair of switches. The switches are connected randomly to simulate real wide area network. The performance metric in the experiments is *makespan*. The simulation program is written in GNU C, running on Linux 2.4.18-3, 1G RAM, 2.0GHz CPU.

6.1 Simulation experiment in homogeneous systems

In homogeneous systems, we assume that the processing speed of processors and communication transfer speed on links are all 1. Fig.2 gives the performance comparison between OIHSA, BBSA to BA.

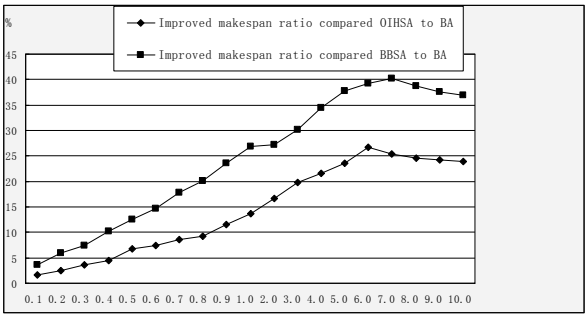

Figure 1. Performance comparison between OIHSA, BBSA and BA.

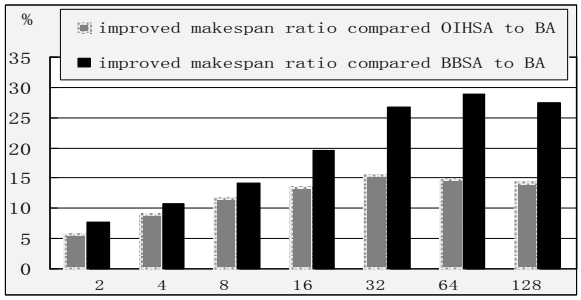

Figure 2. The influence of the number of processors on *makespan*.

The results in Figure 1 are average value under different number of processors when CCR is 0.1-10. The x-axis represents CCR and the y-axis represents percentage improvement in reduced *makespan* compared OIHSA and BBAS to BA.

As can be seen from the figure, with the increase of CCR, the reduced *makespan* ratio increases gradually. One reason is that our algorithm chooses the modified route to balance the workload of network. Another reason is that our algorithms utilizes efficient heuristics to schedule edges on links as soon as possible, which leads to earlier start time of successive tasks, thus the *makespan* is reduced efficiently. However, when the CCR is very large (e.g., CCR>6), the improved performance is not as good as the case when CCR is relatively small. It owes to the fact that overdue communication data makes network workload very heavy, thus, all algorithms face with the critical problem of communication contention and delay on links. It can be found that BBSA is generally better than OIHSA since spare bandwidth can be fully used by BBSA to improve scheduling performance.

Figure 2 shows the influence of the number of processors on *makespa*n. With the increase of the number of the processors, the number of available communication links increases accordingly in the simulation experiment. Thus the modified routing algorithm will choose better route path for edge communication. Meanwhile the optimal insertion and full use of bandwidth reduce *makespan* more efficiently because of more even network workload. But when the number of processors exceeds 64, the improved scheduling performance degrades. The reason is that improved scheduling performance is constrained by the number of tasks and degree of parallelism in the simulation experiment.

6.2 Simulation experiment in heterogeneous systems

Figure 3 and Figure 4 give the performance comparison between our algorithms and BA in heterogeneous systems. The benchmark is the same as the experiment in homogeneous system except the various speeds of processors and network links. Our algorithms still outperform BA in heterogeneous systems except that our algorithms show more superiority in heterogeneous systems. The reason is that the modified routing algorithm reflects the network situation much better than classical routing algorithm in heterogeneous systems. Meanwhile, BBSA can utilize more spare bandwidth on high speed network links to reduce *makespan*. The influence of

the number of processors on *makespan* in heterogeneous systems is quite similar to the case in homogeneous systems, thus we do not give details further.

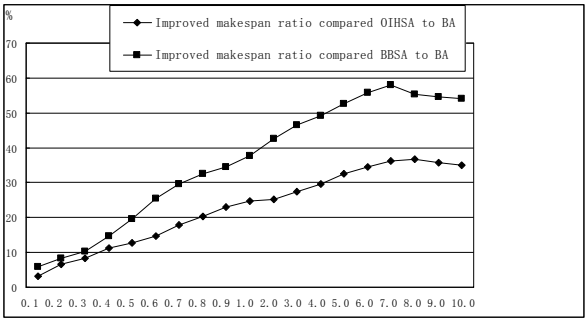

Figure 3. Performance comparison between OIHSA, BBSA and BA.

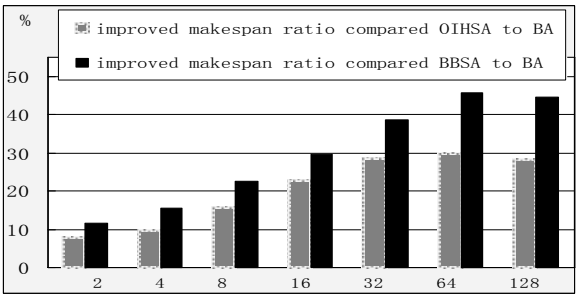

Figure 4. The influence of the number of processors on *makespan*.

7. Conclusion

This paper focuses on the issue of edge scheduling for dependent tasks in parallel and distributed environment, while most of recently relative works ignore the contentions and delays of communication data transferred on network links. Combined with classically efficient heuristics, the paper proposes two contention-aware algorithms. Both of two algorithms start from the basic characteristic of the scheduling problem, and select route paths with relatively low network workload to transfer communication data by modified routing algorithm. OISHA optimizes the start time of communication data transferred on links in form of theorems. BBSA utilizes bandwidth of links on network fully to transfer communication data as fast as possible. Thus, the schedule length is reduced by our algorithms efficiently.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant No. 60503048.

Reference

[1] M. R. Garey and D. S. Johnson, "Computers and Intractability: A guide to the Theory of NP-Completeness", *W. H. Freeman and Co.*, 1979

[2] Jian-Jun Han and Qing-Hua Li, "A Novel Static Task Scheduling Algorithm in Distributed Computing Environment", *18th International Parallel and Distributed Processing Symposium*, 2004

[3] Rashmi Bajaj and Dharma P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment", *IEEE trans. on Parallel and Distributed Systems*, 2004, 15(2): 107-118

[4] Andrei Radulescu and Arjan J. C. van Gemund, "Low-Cost Task Scheduling for Distributed Memory Machines", *IEEE Trans. Parallel and Distributed Systems*, 2002, 13(6): 648-658

[5] Kwan Woo Kim, Mitsuo Gen, and Genji Yamazaki, "Hybrid genetic algorithm with fuzzy logic for resource-constrained projecting scheduling", *Applied Soft Computing*, 2003, 3: 174-188

[6] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version", *European Journal of Operational Research*, 2003, 149: 268-281

[7] Chan-Ik Park and Tae-Young Choe, "An Optimal Scheduling Algorithm Based on Task Duplication", *IEEE Trans. Computers*, 2002, 51(4): 44-448

[8] Hong Jin et al., "Dynamic fuzzy preemptive scheduling algorithm", *China Journal of Computer*, 2004, 27(6): 812-818

[9] G. Viswanathkumar and G. Srinivasan, "A branch and bound algorithm to minimize completion time variance on a single processor", *Computer&Operations Research*, 2003, 30: 1135-1150

[10] B.S. Macey and A.Y. Zomaya, "A Performance Evaluation of CP List Scheduling Heuristics for Communication Intensive Task Graphs", *Proc. Int'l Parallel and Distributed Processing*, 1998, 538-541

[11] Oliver Sinnen and Leonel A.Sousa, "Communication Contention in Task Scheduling", *IEEE Transaction on Parallel and Distributed System*, June 2005, 16(6):503-515

[12] Soung Y. Liew, Gang Hu, and H. Jonathan Chao, "Scheduling Algorithms for Shared Fiber-Delay-Line Optical Packet Switches — Part I: The Single-Stage Case", *Journal Of Lightwave Technology*, 2005, 23(4):1586-160