Toward Optimizing Latency under Throughput Constraints for Application Workflows on Clusters*

Nagavijayalakshmi Vydyanathan[1], Umit V. Catalyurek[2], Tahsin M. Kurc[2], Ponnuswamy Sadayappan[1], and Joel H. Saltz[2]

[1] Dept. of Computer Science and Engineering
{vydyanat, saday}@cse.ohio-state.edu
[2] Dept. of Biomedical Informatics
{umit, kurc, saltz}@bmi.osu.edu
The Ohio State University

Abstract. In many application domains, it is desirable to meet some user-defined performance requirement while minimizing resource usage and optimizing additional performance parameters. For example, application workflows with real-time constraints may have strict throughput requirements and desire a low latency or response-time. The structure of these workflows can be represented as directed acyclic graphs of coarse-grained application tasks with data dependences. In this paper, we develop a novel mapping and scheduling algorithm that minimizes the latency of workflows that act on a stream of input data, while satisfying throughput requirements. The algorithm employs pipelined parallelism and intelligent clustering and replication of tasks to meet throughput requirements. Latency is minimized by exploiting task parallelism and reducing communication overheads. Evaluation using synthetic benchmarks and application task graphs shows that our algorithm 1) consistently meets throughput requirements even when other existing schemes fail, 2) produces lower-latency schedules, and 3) results in lesser resource usage.

1 Introduction

Complex application workflows can often be modeled as directed acyclic graphs (DAGs) of coarse-grained application components with data dependences. The quality of execution of these workflows is often gauged by two metrics: latency and throughput. Latency is the time to process an individual data item through the workflow, while throughput is a measure of the aggregate rate of processing of data. It is often desirable or necessary to meet a user-defined requirement in one metric, while achieving higher performance value in the other metric and minimizing resource usage. Workflows with real-time constraints, for example, can have strict throughput requirements, while interactive query processing may have strict latency constraints. To be able to meet requirements and minimize resource usage is also important in settings such as Supercomputer centers, where resources (e.g., a compute cluster) have an associated cost and are contended for by multiple clients.

Workflows in domains such as image processing, computer vision, signal processing, parallel query processing, and scientific computing often act on a stream of input

* This research was supported in part by the National Science Foundation under Grants #CCF-0342615 and #CNS-0403342.

data [1, 2]. Each task in the workflow repeatedly receives input data items from its predecessor tasks, computes on them, and writes the output to its successors. Multiple data items can be processed in a parallel or pipelined manner and independent tasks can be executed concurrently. In this paper, we present a novel approach for the scheduling of such workflows on clusters of homogeneous processors. Our algorithm employs pipelined, task and data parallelism in an integrated manner to meet strict throughput constraints and minimize latency. *Pipelined parallelism* is the concurrent execution of dependent tasks in the workflow on different instances of the input data stream, *data parallelism* is the concurrent processing of multiple data items by replicas of a task, and *task parallelism* is the concurrent execution of independent tasks on the same instance of the data stream.

We compare our approach against two existing schemes: Filter Copy Pipeline (FCP) [3] and EXPERT (EXploiting Pipeline Execution undeR Time constraints) [2]. Evaluations are done using synthetic benchmarks and application task graphs in the domains of Image Analysis, Video Processing and Computer Vision [1, 2, 4]. We show that our algorithm is able to 1) consistently meet throughput requirements even when the other schemes fail, 2) generate schedules with lower latency, and 3) reduce resource usage.

2 Related Work

Several researchers have addressed the problem of minimizing the parallel completion time (latency) of applications modeled as DAGs. As this problem is NP-complete [5], heuristics have been proposed and a survey of these can be found in [6]. Researchers have also proposed the use of pipelined scheduling for maximizing the throughput of applications. Hary and Ozguner [7] discussed heuristics for maximizing the throughput of application DAGs, while Yang [8] presented an approach for resource optimization under throughput constraints. Benoit and Robert [9] have addressed the problem of maximizing the throughput of pipeline skeletons of linear chains of tasks on heterogeneous systems. These techniques, however, do not consider replication of tasks.

Though many papers focus on optimizing latency or throughput in isolation, very few address both. Subhlok and Vondran [10] have proposed a dynamic programming solution for optimizing latency under throughput constraints for applications composed of a chain of data-parallel tasks. Benoit and Robert [11] study the theoretical complexity of latency and throughput optimization of pipeline and fork graphs with replication and data-parallelism under the assumptions of linear clustering and round-robin processing of input data items. In [3], Spencer et al. presented the Filter Copy Pipeline (FCP) scheduling algorithm for optimizing latency and throughput of data analysis application DAGs on heterogeneous resources. FCP computes the number of copies of each task that is necessary to meet the aggregate production rate of its predecessors and maps the copies to processors that yield their least completion time. Another closely related work is [2], where Guirado et al. have proposed a task mapping algorithm called EXPERT (EXploiting Pipeline Execution undeR Time constraints) that minimizes latency of streaming applications, while satisfying a given throughput constraint. EXPERT identifies maximal clusters of tasks that can form synchronous stages that meet the throughput constraint and maps tasks in each cluster to the same processor so as to reduce communication overheads and minimize latency.

3 Task Graph and Execution Model

A workflow can be modeled as a connected, weighted DAG $G = (V, E)$, where V, the set of vertices, represents non-homogeneous sequential tasks and E, the set of edges, represents data dependences. The task graph G acts on a stream of data, where each task repeatedly receives input data items from its predecessors, computes on them, and writes the output to its successors. The weight of a vertex (task) $t_i \in V$, is its execution time to process a single data item, $et(t_i)$. The weight of an edge $e_{i,j} \in E$, $wt(e_{i,j})$, is the communication cost measured as the time taken to transfer a single data item of size $d_{i,j}$ between t_i and t_j. The length of a path in G is the sum of the weights of the tasks and edges along that path. The *critical path* of G, denoted by $CP(G)$, is the longest path in G. The *bottom level* of a task t in G, $bottomL(t)$, is defined as the length of the longest path from t to the exit task, including the weight of t.

In this paper, we target homogeneous compute clusters for execution of the task graph G. Our algorithm assumes that the execution behavior of the tasks in G is not strongly dependent on the properties of the input data items and that profiling G on several representative data sets gives a reasonable measure of the task execution times. The system model assumes overlap of computation and communication.

The latency of a schedule of task graph G on P processors is the time taken to process a single data item through G. G', the DAG that represents the dependences in the schedule, can be constructed from G by adding zero-weight *pseudo-edges* between concurrent tasks in G that are mapped to the same processor. These pseudo-edges denote induced dependences. The latency is defined to be the critical path length of G'.

Let a task-cluster denote the group of all tasks that are mapped to the same processor. The time taken by a task-cluster C_i to process a single data item is given by the sum of the execution times of its constituent tasks, i.e $et(C_i) = \sum_{\forall t \in C_i} et(t)$. If the workflow is assumed to act on a stream of independent data items (i.e processing of each data item is independent of the processing of other data items), replicas of a task/task-cluster can be executed concurrently. If $nr(C_i)$ denotes the number of replicas of task-cluster C_i, the aggregate processing rate of C_i, $pr(C_i)$ is given by $\frac{nr(C_i)}{et(C_i)}$ data items per unit time. Each replica of a task-cluster is assumed to be executed on a separate processor. For example, assume that tasks t_1 and t_2 are mapped to task-cluster C and $bottomL(t_1) > bottomL(t_2)$ in G'. Let $nr(C)$ be 2, the replicas be mapped to processors P_1 and P_2, $et(t_1) = 10$, and $et(t_2) = 20$. Then, on each of these processors, t_1 processes a data item followed by t_2. The processing rate of C is $\frac{2}{(10+20)}$.

The data transfer rate of an edge $e_{i,j}$, $dr(e_{i,j})$, is $\frac{1}{\frac{d_{i,j}}{bw_{i,j}}}$ data items per unit time, where $bw_{i,j} = \min(nr(t_i), nr(t_j)) \times bandwidth$. Here, bandwidth corresponds to the minimum of disk or memory bandwidth of the system depending on the location of data and the network bandwidth. $nr(t_i)$ denotes the number of replicas of task t_i. As we assume that computation and communication can overlap, the overall processing rate or throughput of the workflow is determined by the slowest task-cluster or edge, and is given by $\min(\min_{\forall C_i} pr(C_i), \min_{\forall e_{i,j}} dr(e_{i,j}))$.

4 Workflow Mapping and Scheduling Heuristic

Given a workflow-DAG G, P homogeneous processors and a throughput constraint T, our workflow mapping and scheduling heuristic (WMSH) generates a mapping and schedule of G on P that minimizes the latency while satisfying T. The algorithm consists of three main heuristics, which are executed in sequence: the *Satisfy Throughput Heuristic* (STH) to meet the user-defined throughput requirements, the *Processor Reduction Heuristic* (PRH) to ensure that the resulting schedule does not require more processors than available, and the *Latency Minimization Heuristic* (LMH) to minimize the workflow latency. In this section, we describe each of these heuristics. Details on the proofs for theorems can be found in the technical report [12].

Theorem 1 *Given a workflow-DAG $G = (V, E)$ that acts on a stream of independent data items, the maximum achievable throughput T_{max}, on P homogeneous processors is given by $\frac{P}{\sum_{t \in V}(et(t))}$, where $et(t)$ is the time taken by t to process a single data item.*

T_{max} can be achieved by mapping all tasks in G to a single task-cluster and making P replicas, each mapped to a unique processor. However, this mapping suffers from a large latency as it fails to exploit parallelism between concurrent tasks in G. For the sake of presentation, the rest of this section assumes that G acts on a stream of independent data items and hence all tasks can be replicated. However, the heuristics described here can be applied when processing of a data item is dependent on the processing of certain other data items (i.e replication of tasks is not allowed), by enforcing the weight of every task-cluster to be $\leq \frac{1}{T}$, for a given throughput constraint $T \leq T_{max}$. T_{max} in this case, is the reciprocal of the weight of the largest task in G.

Given a throughput constraint $T \leq T_{max}$, STH verifies whether a non-pipelined low latency schedule, generated by priority-based list-scheduling [6], meets the throughput requirement. The tasks in G are prioritized in the decreasing order of their bottom-levels and scheduled in priority order to processors that yield their least completion time. If the throughput of this schedule (which is the reciprocal of the latency) is $\geq T$, STH returns this schedule. Otherwise, the following steps are executed to obtain a low-latency pipelined schedule that satisfies T. To generate a pipelined schedule, each task $t_i \in V$ is mapped to a separate task-cluster C_i. Let M denote the set of all the task-clusters. The number of replicas of C_i, $nr(C_i)$, required to satisfy T is computed as $nr(C_i) = T \times et(C_i)$. When there is no throughput constraint, $nr(C_i) = 1$. For all edges $e_{i,j} \in E$, whose data transfer rate is $< T$, STH avoids the communication overhead by merging the task-clusters containing the incident tasks. When two task-clusters are merged, the DAG G' representing the dependences in the schedule is constructed from G by adding zero weight *pseudo-edges* between concurrent tasks in G that are mapped to the same task-cluster. The pseudo-edges originate from the task with the larger bottom-level. Edges between tasks mapped to the same task-cluster have zero weight in G'. An example to illustrate this is given in the technical report [12].

Following STH, PRH is executed. The total number of processors required to execute $nr(C_i)$ copies of each task-cluster C_i, where each copy is mapped to a unique processor, is $P' = \sum_{C_i \in M} \lceil nr(C_i) \rceil$. If $P' > P$, PRH merges certain task-clusters and obtains a schedule that uses $\leq P$ processors. Once a feasible schedule is obtained, LMH is called to optimize the latency. PRH and LMH output a set of task-clusters and the pipelined schedule is obtained by mapping each replica of a task-cluster to a unique processor. Tasks within a task-cluster are run in the decreasing order of their bottom-levels and iterate over the instances of the data stream. We now present PRH and LMH in greater detail.

Algorithm 1 PRH: Processor Reduction Heuristic

```
 1: function PRH($G'$, $M$)    ▷ $G' \leftarrow$ schedule DAG returned by STH, $M \leftarrow$ set of task-clusters returned by STH
 2:     $P' = \sum_{C_i \in M}(\lceil nr(C_i) \rceil)$
 3:     repeat
 4:         $\mathcal{C}' \leftarrow \{(C_i, C_j) \mid C_i \in M \wedge C_j \in M \wedge \lceil nr(C_i) + nr(C_j) \rceil < (\lceil nr(C_i) \rceil + \lceil nr(C_j) \rceil)\}$
 5:         while $\mathcal{C}'$ not empty $\wedge (P' > P)$ do
 6:             Pick the task-cluster pair $(C_i, C_j)$ from $\mathcal{C}'$ that yields the largest decrease in latency when merged.
                    Preference is given to task-clusters that are connected, not concurrent and which produce the
                    largest resource wastage when merged.
 7:             For all task-pairs $(t_a, t_b) \in C_i \times C_j \mid t_a$ concurrent to $t_b$ in $G$, add a pseudo-edge in $G'$ originating
                    from the task with the larger bottom-level.
 8:             For all edges $e_{a,b} \in G \mid (t_a, t_b) \in C_i \times C_j, wt(e_{a,b}) \leftarrow 0$ in $G'$
 9:             Merge $C_i$ and $C_j$ and update $M$
10:             $P' \leftarrow P' - 1$
11:             Update $\mathcal{C}'$
12:         if $P' > P$ then
13:             Pick the task-cluster pair $(C_i, C_j)$ that yields the maximum value of $\lceil (nr(C_i) + nr(C_j)) \rceil -$
                    $(nr(C_i) + nr(C_j))$ and the largest decrease in latency when $C_i$ and $C_j$ are merged.
14:             For all task-pairs $(t_a, t_b) \in C_i \times C_j \mid t_a$ concurrent to $t_b$ in $G$, add a pseudo-edge in $G'$ originating
                    from the task with the larger bottom-level.
15:             For all edges $e_{a,b} \in G \mid (t_a, t_b) \in C_i \times C_j, wt(e_{a,b}) \leftarrow 0$ in $G'$
16:             Merge $C_i$ and $C_j$ and update $M$
17:     until $P' \leq P$
18:     return $< G', M >$
```

4.1 Processor Reduction Heuristic (PRH)

PRH recursively merges pairs of task-clusters based on some metric until we get a mapping that uses $\leq P$ processors.

Theorem 2 *If task-clusters C_i and C_j are merged and P_i and P_j are the number of processors required to run the replicas of C_i and C_j respectively, i.e $P_i = \lceil nr(C_i) \rceil$ and $P_j = \lceil nr(C_j) \rceil$, the number of processors required to run the replicas of the new task-cluster formed that meets the throughput constraint is either $P_i + P_j$ or $P_i + P_j - 1$.*

The pseudo code of PRH is illustrated in Algorithm 1. Step 4 of the algorithm considers all pairs of task-clusters that when merged would reduce the number of processors used by 1. Among these, PRH picks the task-cluster pair that yields the largest decrease in latency when merged. To break ties, preference is given to task-clusters that are connected, not concurrent, and which produce the largest resource wastage, in that order (step 6). Task-clusters C_i and C_j are "connected" if there exists some task t_a in C_i and some task t_b in C_j such that $e_{a,b}$ is an edge in G. Task-clusters C_i and C_j are "not concurrent" if for all pairs of tasks (t_a, t_b), $t_a \in C_i$ and $t_b \in C_j$, t_a is not concurrent to t_b in G. Resource wastage of a task-cluster C is defined as $\lceil nr(C) \rceil - nr(C)$. Giving preference to task-cluster pairs that yield a larger resource wastage reduces the possibility of fragmentation. Steps 5-11 are repeated as long as there are task-cluster pairs that reduce the processor count and $P' > P$. After all possible task clusterings, if the resource usage is still greater than P at step 12, defragmentation is done in steps 13-16 where the task-clusters that produce the largest resource wastage are merged. To break ties, the one that causes the largest decrease in latency is chosen. The outer-loop (steps 3-17) are repeated until the resource usage is lesser than or equal to P. At the end of the processor reduction phase, a mapping M and schedule G' is obtained that meets the throughput constraint and uses $\leq P$ processors.

Algorithm 2 LMH: Latency Minimization Heuristic

```
1: function LMH($G'$, $M$)        ▷ $G' \leftarrow$ schedule DAG returned by PRH, $M \leftarrow$ mapping returned by PRH.
2:     $E^* \leftarrow$ set of all edges in $G'$ where it is optimal to merge the incident tasks (theorem 3)
3:     repeat
4:         while $E^*$ not empty do
5:             $e_{i,j}$ is an edge in $E^*$
6:             For all task-pairs $(t_a, t_b) \in$ clusterOf($t_i$)×clusterOf($t_j$) | $t_a$ concurrent to $t_b$ in $G$, add a
                   pseudo-edge in $G'$ originating from the task with the larger bottom-level.  ▷ clusterOf($t_i$) is
                   the task-cluster that contains task $t_i$.
7:             For all edges $e_{a,b} \in G$ | $(t_a, t_b) \in$ clusterOf($t_i$)×clusterOf($t_j$), $wt(e_{a,b}) \leftarrow 0$ in $G'$
8:             Merge clusterOf($t_i$) and clusterOf($t_j$), update $M$
9:             Update $E^*$
10:        Pick edge $e_{i,j}$ in $CP(G')$ that does not increase the latency when clusterOf($t_i$) and clusterOf($t_j$)
               are merged and has maximum value of $\min(wt(e_{i,j}), CPL(G') - LBL(G))$ and minimum
               value of $(|\text{critical-edges}(t_i)| + |\text{critical-edges}(t_j)|)$   ▷ $CPL(G') \leftarrow$ Critical Path Length of $G'$,
               $LBL(G) \leftarrow$ Lower Bound on Latency of $G$
11:        For all task-pairs $(t_a, t_b) \in$ clusterOf($t_i$)× clusterOf($t_j$) | $t_a$ concurrent to $t_b$ in $G$, add a
               pseudo-edge in $G'$ originating from the task with the larger bottom-level.
12:        For all edges $e_{a,b} \in G$ | $(t_a, t_b) \in$ clusterOf($t_i$)×clusterOf($t_j$), $wt(e_{a,b}) \leftarrow 0$ in $G'$
13:        Merge clusterOf($t_i$) and clusterOf($t_j$) and update $M$
14:        Update $E^*$
15:    until For all edges $e_{i,j}$ in $CP(G')$, latency increases when clusterOf($t_i$) and clusterOf($t_j$) are merged
16:    return $< G', M >$
```

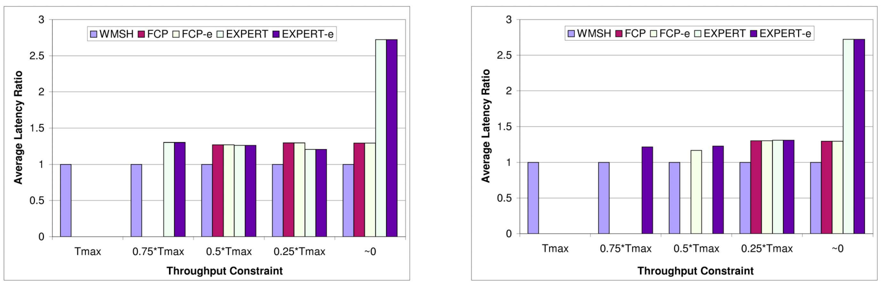
4.2 Latency Minimization Heuristic (LMH)

LMH is called to refine the mapping obtained by PRH to further optimize the latency by reducing communication overheads. The task-clusters in M are considered by LMH as indivisible macro-tasks. A macro-task therefore, may contain one or more tasks. The incoming and outgoing edges of a macro-task is the union of the incoming and outgoing edges, respectively, of the tasks that it contains, without considering edges between tasks belonging to the macro-task. Hence, the term task in Theorem 3 is the same as macro-task in the case where multiple tasks are mapped to same task-cluster by PRH.

Theorem 3 *Let G' and M denote a schedule and mapping of G that meets the throughput constraint and uses $\leq P$ processors. Let $e_{i,j}$ be an edge in G' from task/macro-task t_i to t_j such that the in-degree(t_i) = in-degree$(t_j) = 1$ and the out-degree(t_i) = out-degree$(t_j) = 1$ (i.e. t_i and t_j are connected along a linear chain in that order). Let t_k be the parent of t_i and t_l be the child of t_j. If $wt(e_{i,j}) > wt(e_{k,i}) + wt(e_{j,l})$, it is optimal to merge t_i and t_j to a single task-cluster, assuming that all tasks can be replicated. If replication is not allowed, t_i and t_j can be merged to a single task-cluster only if $et(t_i) + et(t_j) \leq \frac{1}{T}$ and $e_{i,j}$ satisfies the above condition.*

This theorem can be extended to the case where the tasks/macro-tasks are not connected in a linear chain. Details of this can be found in the technical report [12].

Algorithm 2 describes LMH. LMH identifies the set E^* of edges where it is optimal to merge the incident tasks (theorem 3 and its extensions) (step 2) and merges the task-clusters of the incident tasks (steps 6-8). After merging, E^* is updated (step 9). Steps 4-9 are repeated until E^* is empty. In steps 10-14, among the edges along $CP(G')$ that do not cause an increase in latency when zeroed-in, LMH zeroes-in the edge with the largest maximum possible decrease in latency. To break ties, the edge $e_{i,j}$ with the minimum value of the sum of number of critical edges to t_i and number of critical edges to t_j is chosen. The outer-loop of steps 3-15 is repeated until all edges in $CP(G')$



(a) (b)

Fig. 1. Performance on Benchmark-I on (a) 32 processors, (b) 64 processors.(The missing bars indicate that the corresponding algorithm could not meet the throughput requirement.)

cause an increase in latency when zeroed-in. Details regarding the order of complexity of WMSH can be found in [12].

5 Performance Analysis

This section evaluates the performance of WMSH against previously proposed schemes: Filter Copy Pipeline (FCP) [3] and EXPERT (EXploiting Pipeline Execution undeR Time constraints) [2], and FCP-e and EXPERT-e, their modified versions. When FCP fails to utilize all processors and does not meet the throughput requirement T, FCP-e recursively calls FCP on the remaining processors until T is satisfied or all processors are used. EXPERT-e replicates the task-clusters by dividing the remaining processors among them in the ratio of their weights. The performance of these algorithms is evaluated using both synthetic task graphs and those derived from applications, using simulations.

5.1 Synthetic Task Graphs

Two sets of synthetic benchmarks were used in the evaluations: 1) Benchmark-I: randomly generated task graphs with communication delays [13], and 2) Benchmark-II: synthetic graphs generated using the DAG generation tool in [14]. More details on the benchmarks can be found in the technical report [12]. Figure 1 plots the performance on benchmark-I on 32 and 64 processors. The x-axis is the throughput constraint, which is decreased from the maximum achievable throughput (T_{max}) in steps of 0.25. The symbol ≈ 0 denotes the case when there is no throughput constraint (or negligibly small). The y-axis is the average latency ratio. Latency ratio is the ratio of the latency of the schedule generated by an algorithm to that of WMSH. The results show that WMSH consistently generates schedules that meet the throughput constraint, while FCP and EXPERT fail at large throughput requirements ((T_{max} and 0.75*T_{max}). Though FCP replicates tasks, it computes the number of replicas independent of the number of processors and fails to refine the number of replicas when it maps multiple tasks to the same processor. EXPERT does not replicate tasks. The modified versions are designed

T	WMSH	FCP	FCP-e	EXPERT	EXPERT-e
T_{max}	1.00	0.31	0.35	0.40	0.68
0.75*T_{max}	1.00	0.41	0.55	0.53	1.00
0.50*T_{max}	1.00	0.59	1.00	0.80	1.00

(a)

T	WMSH	FCP	FCP-e	EXPERT	EXPERT-e
T_{max}	1.00	-	-	-	-
0.75*T_{max}	0.91	-	-	-	0.94
0.50*T_{max}	0.73	-	1.00	-	0.94

(b)

Table 1. Performance on Benchmark-I on 64 processors (a) Average Throughput Ratio, (b) Average Utilization Ratio.(The missing values in (b) indicate that the corresponding algorithm could not meet the throughput requirement.)

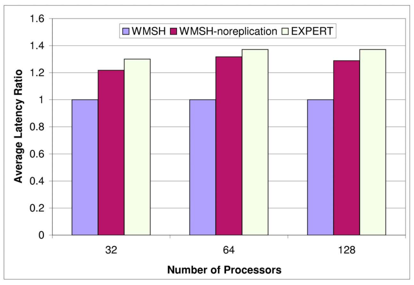
to overcome some of these limitations and hence, meet the constraint in some of the cases where FCP or EXPERT fail.

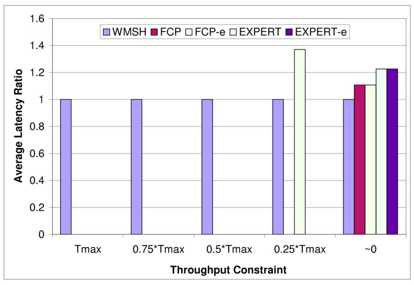
With respect to latencies, we find that WMSH generates lower latency schedules. On 32 processors, FCP generates 27%-29% longer latencies than WMSH, while EXPERT generates 20%-30% longer latencies when throughput constraint is relaxed upto 0.25*T_{max}. As EXPERT creates maximal task-clusters with weights $\leq \frac{1}{T}$, for negligible throughput constraint, it groups all tasks to a single task-cluster resulting in large latencies. For FCP-e, we used the smallest of the latencies of all the workflow instances it creates and hence it is similar to that of FCP. Latency in EXPERT-e is similar to EXPERT, since EXPERT-e only replicates tasks; this improves the throughput but does not alter the latency. As P is increased, T_{max} increases, and hence, there are more instances where FCP and EXPERT do not satisfy T.

Table 1(a) shows the average throughput ratio for the schemes for Benchmark-I on 64 processors. The throughput ratio is the ratio of the throughput achieved by an algorithm to the throughput constraint. If the achieved throughput is greater than the constraint, the ratio is taken to be 1. Beyond 0.5*T_{max}, all schemes meet the constraint. When FCP and EXPERT fail, they generate schedules with throughput atleast 40% and 20% less than the constraint, respectively. Table 1(b) shows the average utilization ratio for the schemes. The utilization ratio is given by the ratio of the number of processors used by an algorithm to the total number of available processors. Among schemes that satisfy T, WMSH produces lower-latency schedules while using fewer processors. For example, when T is 0.5*T_{max}, utilization of WMSH is 27% lower than that of FCP-e and 19% lower than EXPERT-e, and it produces latencies 15% and 19% shorter than FCP-e and EXPERT-e respectively.

As EXPERT does not replicate tasks, we compared its performance with that of WMSH with replication disabled (Fig. 2(a)). Even with no replication, WMSH produces lower latencies than EXPERT. WMSH with replication shows the least latency as tasks connected by edges with heavy communication cost can be mapped to the same task-cluster and replicated to meet the throughput constraint. Thus replication not only helps in improving throughput but also minimizing the latency.

To study the impact of communication costs, we evaluated the schemes using Benchmark-II by varying the communication to computation ratio (CCR) as 0.1, 1 and 10. Figure 2(b) shows the performance when CCR=10. Due to space constraints, we have not included results for CCR=0.1,1. These can be found in the technical report [12]. For larger CCR values, we find more instances where FCP, EXPERT and their modified versions do not meet the throughput constraint, while WMSH always does. WMSH intelligently zeroes-in heavy edges by mapping the incident tasks to the same task-cluster and replicating this cluster to meet the throughput constraint. Though FCP minimizes communication costs in some capacity by mapping replicas to processors that yield their least completion time, it still incurs the cost when the processor to which the parent task is mapped is heavily loaded (as mapping the task to this processor would cause a larger





(a) (b)

Fig. 2. (a) Relative Performance of WMSH, WMSH with replication disabled and EXPERT when throughput constraint is $\frac{1}{\max_{t \in V}(et(t))}$, (b) Performance on Benchmark-II on 32 processors and CCR=10.(The missing bars in (b) indicate that the corresponding algorithm could not meet the throughput requirement.)

T	WMSH	FCP	FCP-e	EXPERT	EXPERT-e
T_{max}	1.00	-	-	-	-
0.75*T_{max}	1.00	-	-	-	-
0.50*T_{max}	1.00	-	-	-	-
0.25*T_{max}	1.00	-	1.04	-	-
≈0	1.00	1.04	1.04	1.15	1.15

(a)

T	WMSH	FCP	FCP-e	EXPERT	EXPERT-e
T_{max}	1.00	-	-	-	-
0.75*T_{max}	0.75	-	-	-	-
0.50*T_{max}	0.53	-	-	-	-
0.25*T_{max}	0.31	-	1.00	-	-
≈0	0.25	0.47	0.47	0.03	1.00

(b)

Table 2. Performance of Darpa Vision Benchmark on 32 processors (a) Latency Ratio, (b) Utilization Ratio.(The missing values indicate that the corresponding algorithm could not meet the throughput requirement)

completion time). EXPERT does not replicate and hence cannot cluster heavy tasks that also have a huge communication cost. The modified versions of the schemes do not completely avoid the communication overheads as they only replicate tasks. As for Benchmark-I, WMSH generates the lowest latency schedules that use lesser resources.

5.2 Application Task Graphs

Evaluations were done using task graphs from computer vision, multimedia and imaging domains. Due to space limitations, we present results for only two applications; detailed evaluation can be found in the technical report [12]. Table 2 shows the performance for the Darpa Vision Benchmark (DVB) [4], which performs model-based object recognition of a hypothetical object. We find that FCP, EXPERT and their modified versions do not meet the throughput requirement T, in many instances. In cases where they satisfy T, WMSH produces schedules with shorter latencies and lower resource utilization than FCP. When T is negligible, the schedule generated by WMSH uses 22% fewer processors than that of FCP and has 4% lower latency. WMSH also produces latencies 15% lower than that of EXPERT.

Table 3 shows results for an MPEG video compression application [2]. Due to frame encoding dependences, MPEG frames have to be processed in-order and hence task replication is not allowed. However, the input frames can be divided into N segments, which can be processed in parallel. We assumed T_{max} to be the reciprocal of the weight of the largest task and varied N from 2 to 16. We find that FCP and WMSH generate

Segments	WMSH	FCP	EXPERT
2	1.00	1.00	1.21
4	1.00	1.00	1.36
8	1.00	1.00	1.41
16	1.00	1.00	1.24

(a)

Segments	WMSH	FCP	EXPERT
2	0.13	0.13	0.09
4	0.25	0.41	0.22
8	0.50	0.78	0.47
16	1.00	1.00	1.00

(b)

Table 3. Performance of MPEG video compression on 32 processors (a) Latency Ratio, (b) Utilization Ratio.(The missing values indicate that the corresponding algorithm could not meet the throughput requirement)

schedules with similar latencies, but WMSH has upto 28% lower utilization. Though EXPERT shows lower utilization, it generates schedules with 21%-41% longer latencies than WMSH or FCP. The scheduling times in these experiments were less than a second suggesting that scheduling is not a time critical operation for these applications.

6 Conclusions

This paper presents a mapping and scheduling heuristic that minimizes the latency of workflows that operate on a stream of data, while satisfying strict throughput requirements. Our algorithm meets the throughput constraints through pipelined parallelism and replication of tasks. Latency is minimized by exploiting task parallelism and reducing communication overheads. Evaluation using synthetic and application task graphs indicate that our heuristic is always guaranteed to meet the throughput requirement and hence can be deployed for scheduling workflows with real-time constraints. Further, it produces lower latency schedules and utilizes lesser resources.

Acknowledgments We would like to thank Dr. Yves Robert and Dr. Anne Benoit for their valuable discussions and constructive reviews on the paper.

References

1. Kumar, V.S., Rutt, B., Kurc, T., Catalyurek, U., Saltz, J., Chow, S., Lamont, S., Martone, M.: Large image correction and warping in a cluster environment. In: Supercomputing Conf. (2006) 79
2. Guirado, F., A.Ripoll, Roig, C., Luque, E.: Optimizing latency under throughput requirements for streaming applications on cluster execution. In: Cluster Computing Conf. (2005)
3. Spencer, M., Ferreira, R., Beynon, M., Kurc, T., Catalyurek, U., Sussman, A., Saltz, J.: Executing multiple pipelined data analysis operations in the grid. In: Supercomputing Conf. (2002) 1–18
4. Shukla, S.B., Agrawal, D.P.: Scheduling pipelined communication in distributed memory multiprocessors for real-time applications. SIGARCH Comput. Archit. News **19**(3) (1991)
5. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
6. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput. Surv. **31**(4) (1999) 406–471
7. Hary, S.L., Ozguner, F.: Precedence-constrained task allocation onto point-to-point networks for pipelined execution. IEEE Trans. Par. Distrib. Syst. **10**(8) (1999) 838–851
8. Yang, M.T., Kasturi, R., Sivasubramaniam, A.: A pipeline-based approach for scheduling video processing algorithms on now. IEEE Trans. Par. Distrib. Syst. **14**(2) (2003) 119–130
9. Benoit, A., Robert, Y.: Mapping pipeline skeletons onto heterogeneous platforms. Technical Report LIP RR-2006-40 (2006)

10. Subhlok, J., Vondran, G.: Optimal latency-throughput tradeoffs for data parallel pipelines. In: 8th ACM Symp. on Parallel Algorithms and Arch. (1996) 62–71
11. Benoit, A., Robert, Y.: Complexity results for throughput and latency optimization of replicated and data-parallel workflows. Technical Report LIP RR-2007-12 (2007)
12. Vydyanathan, N., Catalyurek, U., Kurc, T., Sadayappan, P., Saltz, J.: An approach for optimizing latency under throughput constraints for application workflows on clusters. Technical Report OSU-CISRC-1/07-TR03, The Ohio State University (2007)
13. Davidovic, T., Crainic, T.G.: Benchmark-problem instances for static scheduling of task graphs with communication delays on homogeneous multiprocessor systems. Computers & OR **33**(8) (2006) 2155–2177
14. Vallerio, K.: Task graphs for free. (http://ziyang.ece.northwestern.edu/tgff/maindoc.pdf)