Novel Neighborhood Search for Multiprocessor Scheduling with Pipelining

K. K. Leung, N. H. C. Yung & P. Y. S. Cheung
Department of Electrical & Electronic Engineering, The University of Hong Kong
kkleung@hkueee.hku.hk, nyung@hkueee.hku.hk, cheung@hkueee.hku.hk

Abstract

This paper presents a neighborhood search algorithm for heterogeneous multiprocessor scheduling in which loop pipelining is used to exploit parallelism between iterations. The method adopts a realistic model for inter-processor communication where resource contention is taken into consideration. The schedule representation scheme is flexible so that communication scheduling can be performed in a generic manner. Based on a general time formulation of the schedule performance, the algorithm improves an initial schedule in an efficient way. Experimental results show that significant improvement over existing methods can be obtained. Using the scheduling results, a parallel software video encoder was implemented and real time performance was achieved.

1. Introduction

Given a program modelled by a task graph, finding an optimal multiprocessor schedule is a well-known NP-complete problem [1]. Taking into account inter-processor communication (IPC), optimal solution has been found under restrictive assumptions on the task graph [2,3] and unbound number of processors connected by a contention free network. These assumptions are rather ideal for real applications and platforms.

More realistic approaches try to model IPC resource contention [4,6]. For example, the Mapping Heuristic (MH) proposed in [4] estimates an additional contention delay for each message with respect to the system state. Unfortunately, no actual implementation was given based on the model. In [5,6], the Ordered Transaction model was proposed and implemented on a board containing four DSP96002 processors and a memory access controller. The shared memory access pattern is determined at compile time, so that run-time resource contention is eliminated. For a 1024 point complex FFT, a speedup of 3 is obtained. Based on a similar IPC model, the Dynamic Level Scheduling (DLS) [7] performs list scheduling where in each step, the best matched task processor pair is found based on the system state. Similarly, the genetic algorithm (*GA*) proposed in [8] represents a schedule by matching and scheduling strings. Both algorithms have implicit restrictions in that the input data transfers for each task are scheduled only when the task is being considered.

For iterative applications, rotation operation was proposed in [9] for loop pipelining without consideration of IPC. In [10], although IPC is included in the model, its scheduling has similar restriction as that of [7,8]. Moreover, both [9,10] assume synchronous control steps and so are unsuitable for asynchronous processors that are common in most distributed or shared memory systems.

As discussed, optimal solution has been found only under restricted problem instances and ideal platforms such as contention free network. When IPC contention is considered, there are often unnecessary restrictions to the IPC scheduling. Therefore, one of our objectives is to develop a realistic and general model for computation and IPC scheduling. Based on this model, we developed a novel neighborhood search algorithm with pipelining to exploit inter-iteration parallelism. Experimental results show that significant improvement can be obtained over existing methods. Using the resulting schedules, a parallel video encoder was implemented, which achieved over 30 frames/sec at 352×240 resolution using 24 processors, which is about 2 times that of the *GA* tested and 37% better than a manually optimized video coding algorithm.

This paper is organized as follows: Section 2 states the model for scheduling. Section 3 presents the method of neighborhood search. Section 4 gives experimental results and discussions. This paper is concluded in Section 5.

2. Problem modelling

In order to obtain true overall performance, the scheduling model should take into account IPC resource contention. For example, ignoring IPC contention, the task graph in Fig. 1(a) has an optimal schedule in Fig. 1(b). In the presence of link contention, the schedule is no longer optimal as shown in Fig. 1(c). As illustrated, the resource contention and the flow of data should be emphasized, which can be represented by a data flow graph (DFG).

In general, the DFG model consists of a number of non-preemptive computation tasks and a number of data objects connected according to $G(V_T \cup V_D, E_{TD} \cup E_{DT})$ with

0-7695-0589-2/00 $10.00 © 2000 IEEE

definition of notations given in the APPENDIX. Each task takes some data objects as input and produces some data objects as output. Fig. 2 depicts an example DFG.

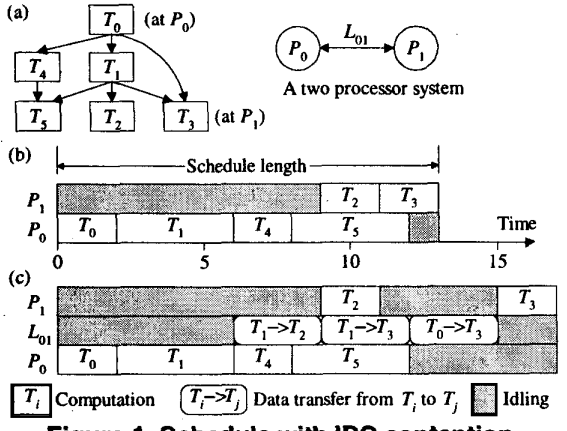

Figure 1. Schedule with IPC contention

For iterative DFG G, an iteration is the execution of all the tasks in G once. For tasks T_i , T_j in V_T and D in V_D, if $(T_i,D)\in E_{TD}$ and $(D,T_j)\in E_{TD}$, then T_j is dependent on the instance of T_i in the $d(D)$ past iteration. In order to maintain precedence constraint, all the cycles in the DFG should include at least one data object, such as D_3 and D_4 in the cycles in Fig. 2, with positive dependence distance.

In the parallel platform model adopted, each data transfer is scheduled to channel resources by dedicating them throughout the duration of transfer [7,8]. For each ordered pair of processors, there is a channel that contains the resources involved. For each computation task, the execution time is assumed to be known a priori and it can be different on different processors. The data transmission time may be modelled with a channel setup time plus the product of data size and an effective bandwidth.

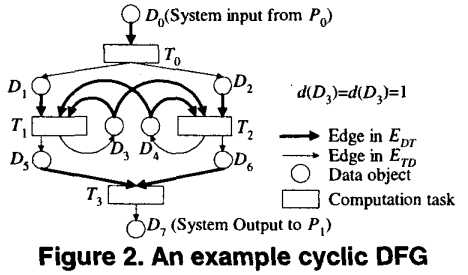

Figure 2. An example cyclic DFG

3. Proposed method

Schedules generated by heuristic and non-deterministic approaches are often sub-optimal. There is obviously opportunity for improving them with neighborhood search in which a solution undergoes modification to obtain neighbor solution which is adopted if it is better. The optimization criteria should be the overall schedule length, rather than the task start time [11]. For example, Fig. 3(a) shows the modified schedule of Fig. 1(c) with T_4 migrated to an earlier time slot in P_1, resulting in a longer schedule. Moreover, the scheduler should not impose unnecessary restriction to the IPC scheduling as in [7,8,10]. For instance, the data transfer $(T_0\rightarrow T_3)$ in Fig. 1(c) can be moved before $(T_1\rightarrow T_2)$, giving a better schedule in Fig. 3(c). We also consider overlapping of successive iterations to exploit inter-iteration parallelism. Fig. 3(b) shows an example of overlapped iterations in which significant improvement is obtained over Fig. 1(c).

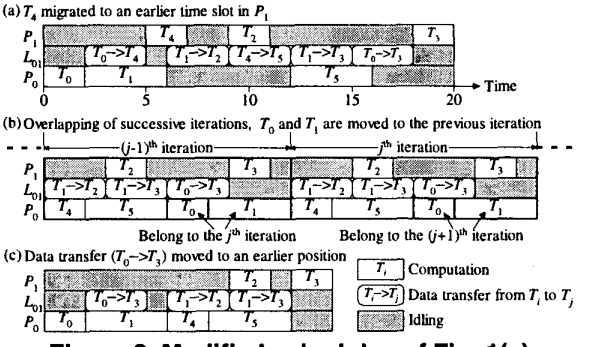

Figure 3. Modified schedules of Fig. 1(c)

At this juncture, we introduce a set of communication tasks, which is formed from the input DFG according to Fig. 4. For case 1, data object D may be used by a number of computation tasks. Each outlet edge (D,T_j) is associated with a communication task T_c for transfering the instance of D in the $d(D)$ past iteration. In case 2, if D is a system output, its inlet edge is associated with a communication task. The G_C of the example DFG is shown in Fig. 5. We collectively call any computation or communication task a task, and $V_T\cup V_{CT}$ is then the set of all tasks.

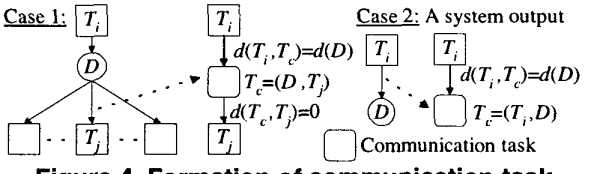

Figure 4. Formation of communication task

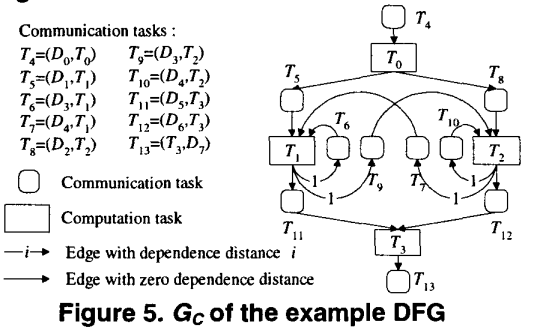

Figure 5. G_C of the example DFG

3.1. Schedule characterization and evaluation

For iterative program, all the loops execute according to the same static schedule (RI,Map,DS,Seq). Table 1

shows an example schedule for the DFG of Fig. 2. The modelled schedule performance is evaluated with several intermediate graphs, as depicted in Fig. 6. First, the DFG G is transformed into G_C. Second, the precedence relations between the tasks are determined with respect to their relative iteration indices (RI). Then, G_P' is derived according to the platform resource constraints.

Table 1. An example schedule on 4 processors

Task	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}
RI	1	0	0	-1	1	0	0	0	0	1	0	0	0	-1
Map	P_0	P_3	P_2	P_1	-	-	-	-	-	-	-	-	-	-
DS	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Seq	T_3	T_{13}	T_{10}	T_7	T_8	T_2	T_5	T_{12}	T_6	T_1	T_{11}	T_4	T_0	T_9

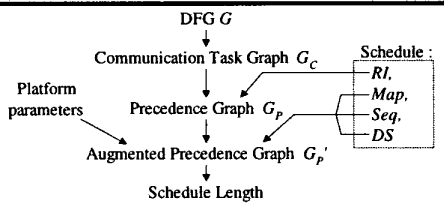

Figure 6. Evaluation of modelled performance

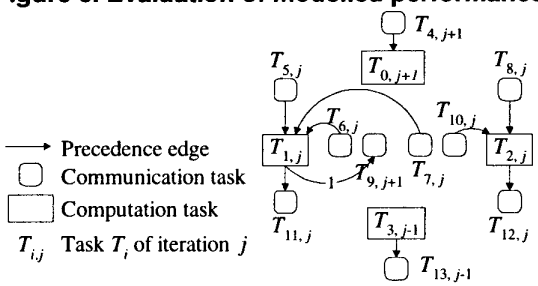

Figure 7. G_P derived from the example G_C and RI

3.1.1. Relative iteration index (RI). The tasks in the schedule may belong to different iterations. $RI(T)$ is the relative iteration index of task T. As indicated by RI in the example schedule, 3 successive iterations are overlapped.

3.1.2. Precedence graph (G_P). The precedence relation of the tasks in the schedule, as represented by $G_P(V_T \cup V_{CT}, E_P)$, is derived from G_C and RI. For $(T_i,T_j) \in E_C$, $(T_i,T_j) \in E_P$ if $(T_i,T_j) \in E_C$ and $RI(T_i)=RI(T_j)-d(T_i,T_j)$. Obviously, E_P contains a subset of the edges of E_C. Fig. 7 depicts the G_P obtained from the example schedule and the G_C.

3.1.3. *Map*, *Seq* and *DS*. Given a schedule, the execution time line is formed by traversing and scheduling the tasks in the order of *Seq*, which is a topological ordered sequence that satisfies the precedence relation of G_P. Each resource has a task list to guide its execution. During the scheduling, computation task T is appended to the task list of processor $Map(T)$. For communication task T, it is appended to the channel resources between the source processor, $Map(Producer(T))$, and the destination processor, $Map(Consumer(T))$. If an alternative data source (*Data Forwarder* [8]) is specified by $DS(T)$, the source is the destination of $DS(T)$. If the data object is already present in the destination processor, T is not scheduled and $ET(T)$ is set to zero. Fig. 8 shows the time line of the example schedule. The resources involved in each scheduling step are tabulated in Table 2.

3.1.4. Augmented precedence graph (G_P'). The schedule length is the critical path of the augmented precedence graph G_P' $(V_T \cup V_{CT}, E_P')$. For tasks T_i and T_j, $(T_i,T_j) \in E_P'$ if $(T_i,T_j) \in E_P$, or $T_i=DS(T_j)$, or T_i is scheduled just before T_j in some resource. The schedule length can be obtained using (1). Using (2), the *tlevel* can be obtained by traversing the tasks in the order of *Seq* since *Seq* satisfies the precedence of G_P'. Similarly, *blevel* can be found uisng (3) by traversing the tasks in reverse order of *Seq*.

$$Sch.length = \max\{tlevel(T) + blevel(T) \mid T \in V_T \cup V_{CT}\}, \quad (1)$$

$$tlevel(T) = \max\{0, tlevel(T_p) + ET(T_p) \mid (T_p,T) \in E_P'\}, \quad (2)$$

$$blevel(T) = \max\{0, blevel(T_s) \mid (T,T_s) \in E_P'\} + ET(T) \quad (3)$$

Assume that the number of input and output data objects for each computation task and the number of resources in each channel are bounded by constants, it takes a constant time for finding *tlevel* and *blevel* for each task. As there are at most $e+v$ tasks, it takes $O(e+v)$ time to find the schedule length.

Table 2. Scheduling steps

Step, i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$T=Seq(i)$	T_3	T_{13}	T_{10}	T_7	T_8	T_2	T_5	T_{12}	T_6	T_1	T_{11}	T_4	T_0	T_9
Source	-	P_1	P_2	P_2	P_0	-	P_0	P_2	P_3	-	P_3	P_0	-	P_3
Destination	-	P_1	P_2	P_3	P_2	-	P_3	P_1	P_3	-	P_1	P_0	-	P_2
Resources involved	P_1	-	-	L_{23}	L_{01} L_{12}	P_2	L_{30}	L_{12}	-	P_3	L_{30} L_{01}	-	P_0	L_{23}

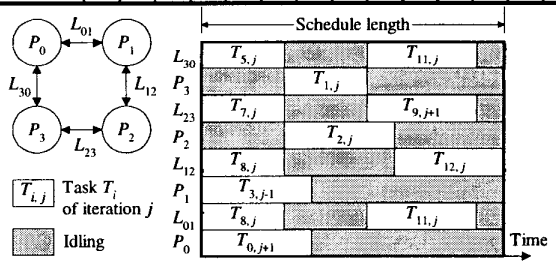

Figure 8. Execution time line

3.2. Neighborhood search

Below are the three phases of search employed.

3.2.1. Phase *NSP-MAP*. In this phase, neighbor solutions are obtained by changing the processor mapping for some computation task, while keeping the processor mapping of the other tasks fixed. The algorithm cycles through all the computation tasks for evaluation upon different processor mappings. It terminates if no improvement is found for all the tasks. In the best and worst cases, it requires p and pv evaluations per improvement respectively. Thus, the time complexity for finding an improvement is $O[pv(e+v)]$.

3.2.2. Phase *NSP-SEQ*. This phase searches, for each task T, a new position in *Seq* that gives the shortest schedule

length. Due to precedence constraints, the search starts by shifting T backward from its original position until reaching a predecessor task. If this shifting reaches the head of *Seq*, T is wrapped around to the end of *Seq* with $RI(T)$ increased by 1 and G_P updated. Then the shifting continues from the end of *Seq*. In this way, T is effectively shifted to the previous loop while the instance of T in the next loop is shifted in. After backward shifting, T is forward shifted from its original position until reaching a successor task. Similarly, when T reaches the end of *Seq*, it is wrapped around to the head with $RI(T)$ decreased by 1 and G_P updated. Wrap around in both directions is not performed if the maximum latency exceeds L. After the shifting, the T is moved to the best position.

For each task, there are at most $e+v$ possible positions in *Seq* and L-1 times of wrap around in both directions. The algorithm terminates when no improvement is found after inspection for all the $e+v$ tasks. As each evaluation takes O(e+v) time, an improvement takes $O[L(e+v)^3]$ time.

This time complexity can be reduced. The idea is to first remove T, then use the levels functions of the remaining tasks with the absence of T to find the levels for T in each insertion position in a constant time. After removing T, if $SL_T < SL\times(1-\varepsilon)$, the algorithm proceeds by shifting T. At each new position of T, SL_{NEW} is given by

$$SL_{NEW} = \max\{tlevel(T) + blevel(T), SL_T\}. \quad (4)$$

In (4), $tlevel(T)$ and $blevel(T)$ are obtained by (2) and (3). During the shifting, pointers are used to keep the positions of T in the resources where it is scheduled. When T is exchanged with its left (right) neighbor T', those pointers corresponding to common resources with T' are decreased (increased) by 1. Then the immediate predecessors and successors of T in G_P' can be identified and the levels for T can be obtained in a constant time. For each task, evaluation of SL_T, updating of G_P during wrap around and inspection of all positions take $O[L(v+e)]$ time. So, an improvement now takes $O[L(v+e)^2]$ time.

3.2.3. Phase *NSP-SEQ-DF*. *NSP-SEQ* may get stuck at a local optimum. We use the degree of freedom as a heuristic function to guide the search, which is defined as

$$DF(T) = SL - tlevel(T) - blevel(T) \quad (5)$$

The search method is similar to *NSP-SEQ*, but without the checking that $SL_T < SL\times(1-\varepsilon)$. During the search, SL may be reduced or unchanged at the new position for T. In order to limit the search time, the algorithm terminates if there is no improvement to SL after cycling all the tasks for k_{DF} (fixed at 2 in all the tests) times. Similar to *NSP-SEQ*, it takes $O[L(v+e)^2]$ time to find an improvement.

3.2.4. The overall *NSP* algorithm. In the overall *NSP* algorithm, the above 3 phases are cycled repeatedly until all of them fail to reduce SL. At worst, it takes the sum of worst case times of the 3 phases to find an improvement. The overall time complexity is $O\{n_I\times[pv(e+v)+L(e+v)^2]\}$. As $n_I \le (SL_i / SL^* - 1)/\varepsilon$, this time complexity becomes $O\{(SL_i/SL^*-1)\times[pv(e+v)+L(e+v)^2]/\varepsilon\}$. Fine improvement is ignored with a large ε. With a small ε, the search is likely to give better result using a longer search time. In all the tests done, the value of ε is 10^{-7}, which can be considered as typical value.

4. Experimental results and discussions

4.1. Comparison by random DFG

Comparisons were made with *DLS* [7] and the *GA* of [8] since they have a similar model of IPC as our approach. For acyclic DFGs, five tests were done with variation in the parameters as shown in Table 3. In each random DFG, v data objects are added to v computation tasks with the producer and consumer tasks selected randomly. Each computation task has an expected unit execution time selected from the range 0.001 to 1.999 with uniform distribution. Each data object has a size also from this range but post-scaled by the desired *CCR*. In each test, 50 DFGs were used, i.e. a total of 250 DFGs in the 5 tests.

Table 3. Test parameter setting

Test	v	CP	N_{SHARE}	p	CCR
1	100-1000	5	5	10	0.2
2	300	10-55	5	10	0.2
3	300	5	1-19	10	0.2
4	300	5	5	2-20	0.2
5	300	5	5	10	0.025-12.8

We adopted the platform model of the *GA*, which has a central crossbar switch to which each processor is connected through an input and an output link. In the following, *NSP/GA* and *NSP/DLS* represent the cases of applying *NSP* to the results of *GA* and *DLS* respectively. In all the tests, L is fixed at 3.

4.1.1. Test 1. From the speedup comparison in Fig. 9, *NSP* has a speedup close to 10, which is ideal at p=10. It also outperforms *DLS* and *GA* consistently by about 30% and 7.5% respectively. One reason is that *DLS* and *GA* have implicit restriction to the scheduling of data transfers.

4.1.2. Test 2. As this test was done with v=300 and p=10, the ideal schedule length has a mean of 300/10 or 30 tu. In Fig. 10, the schedule lengths of *GA* and *DLS* are well above 30. On the contrary, *NSP* allows iterations to overlap such that SL can be shorter than CP.

4.1.3. Test 3. In practice, data objects are often shared by several computation tasks, which introduces more IPC. In Fig. 11, both *DLS* and *GA* give an increasing schedule length with N_{SHARE}. *NSP* is superior, giving a steady value around the ideal value of 30 for N_{SHARE}<13. For larger N_{SHARE}, the schedule length is increased only slightly and is about 35% better than *GA* at N_{SHARE}=19.

4.1.4. Test 4. From Fig. 12, the speedup of *GA* and *DLS* start to deviate from linearity for p>8 and 4 respectively They also tend to level at 14.2 and 11 respectively for p>16, while *NSP* is close to linear for p up to 20.

4.1.5. Test 5. In Fig. 13, the improvement of *NSP* over *GA* and *DLS* starts to increase, reaches about 32% and 45% respectively at *CCR*=0.4. In fact, *NSP* gives a better IPC scheduling as reflected from this substantial improvement.

4.1.6. Execution time. On a Pentium® II 350MHz system, the execution time of the algorithms was measured. For the case of *CP*=55, *NSP* starts from an initial solution from *DLS*, which takes 0.16 second for scheduling. As depicted in Fig. 14, *GA* attains a steady value after about 10 minutes while the *NSP* phases show stepwise drops and stop at about 3.5 minutes. This shows that *NSP* gives a substantially better schedule in a comparably short time.

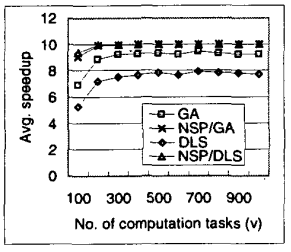

Figure 9. Test 1

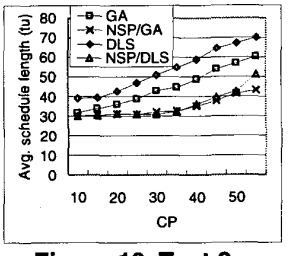

Figure 10. Test 2

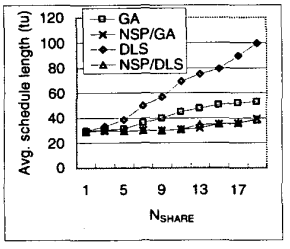

Figure 11. Test 3

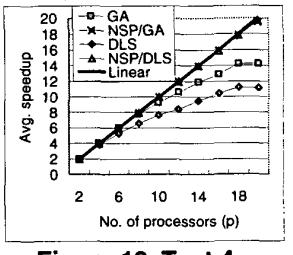

Figure 12. Test 4

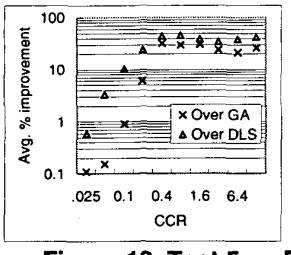

Figure 13. Test 5

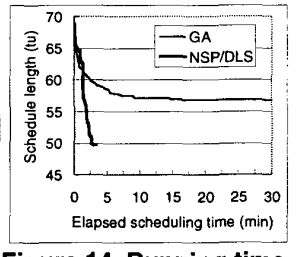

Figure 14. Running time

4.2. Application to video encoding

An H.261 [12] video encoder was implemented and comparisons were made with *GA* and the *Multiple Master Multiple Slave* (*MMMS*) solution [13]. *DLS* was not compared since it cannot be applied to cyclic DFG. The encoding algorithm is represented by the cyclic iterative DFG of Fig. 15 where macroblock (*MB*) is the basic unit of data decomposition. The platform used is the IBM-SP2 in the University of Hong Kong, which is composed of 48 160MHz IBM P2SC RISC processors connected by the High Performance Switch (HPS) with point-to-point bandwidth of 105MBytes/s and latency of 27.5μsec. Blocking send and receive operations of the Message Passing Library were used. All the task execution times were measured using *gettimeofday*. For simplicity, we assume that the HPS is a completely connected switch such that each IPC channel is composed of the source and destination processors only. The video tested consists of 50 frames of 352×240-pixel resolution. It shows a table tennis game that involves a zooming view. Each test was repeated 8 times and the average frame rates were taken.

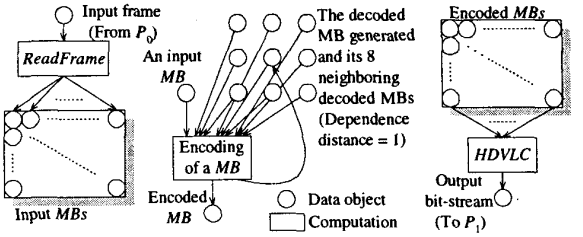

Figure 15. DFG for video coding

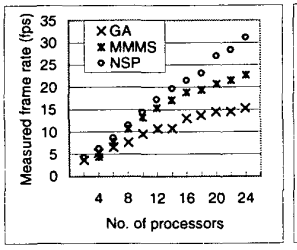

Figure 16. Frame rate

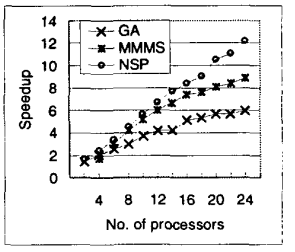

Figure 17. Speedup

4.2.1. Results & discussions. As depicted in Fig. 16, *NSP* gives about 31 frames/sec at p=24, which is about 2 times that of *GA* and 37% better than *MMMS*. From Fig. 17, the speedup of *GA* tends to level at about 6 for p over 20 while both *MMMS* and *NSP* show an increasing trend and reach about 9 and 12 respectively at p=24. *NSP* shows a curve closer to linear than *MMMS*. In fact, *MMMS* is the product of manual optimization by experience while *NSP* is an automatic scheduler for arbitrary DFG and platform.

Owing to variation in message transfer time, deviation from the predicted performance is observed. Firstly, the HPS has message buffers so that the sender can complete earlier. Secondly, network congestion may cause the transfer time to be longer. In our case, the first effect dominates and the message transfer time is generally shorter than the predicted. Furthermore, there is variation in the *MB* encoding time depending on the video content. Since the schedules were generated based on a frame with above average encoding time, the result is better than the predicted. Moreover, the latency from input frame to output bit-stream is only 3 frames encoding time, which suits on-line applications such as video conferencing.

5. Conclusions

In this paper, a scheduling algorithm for heterogeneous multiprocessor systems is presented. First, a flexible

representation scheme is used so that communication scheduling can be done in a generic way. Second, loop pipelining is used to exploit parallelism between iterations. Third, an efficient technique is incorporated into the search that reduced the time complexity by an order of magnitude. Fourth, experimental comparisons were made with *DLS* and a *GA* algorithm using different suites of random DFGs with variations in different parameters including the effect of data sharing. Finally, the method is verified by actual implementation of a video encoder in which over 30 frames/sec is obtained using 24 processors.

APPENDIX. Definition of symbols

DFG $G(V_T \cup V_D, E_{TD} \cup E_{DT})$:	
V_T, V_D	The sets of computation tasks and data objects.
E_{TD}, E_{DT}	The sets of directed edges from V_T to V_D and vice versa.
$d(D)$	Dependency distance of data object D.
v, e	The number of computation tasks and edges in G.
Communication task graph $G_C(V_T \cup V_{CT}, E_C)$:	
V_{CT}	The set of communication tasks.
E_C	The set of directed edges in G_C.
$d(T_i,T_j)$	The dependence distance from T_i to T_j, (T_j is dependent on T_i)
Producer(T)	The computation task producing the data object for communication task T.
Consumer(T)	The computation task using the data object delivered by communication task T.
Platform model :	
S_R, S_P	The sets of resources and processors, $S_P \subset S_R$.
p	The number of processors.
Solution characterization :	
$RI(T)$	The relative iteration index of T.
L	Max. latency from input to output in terms of number of loops.
$Map(T)$	The processor mapping of computation task T.
$Seq(i)$	The i^{th} task in the topological ordered sequence satisfying precedence of G_P.
$DS(T)$	The data source of communication task T.
Precedence graph $G_P(V_T \cup V_{CT}, E_P)$:	
E_P	The set of directed edges in G_P.
Augmented precedence graph G_P' ($V_T \cup V_{CT}$, E_P') :	
E_P'	The set of directed edges in G_P'.
$ET(T)$	The execution time of T.
$tlevel(T)$	Longest path length in G_P' from an entry task to T, excluding $ET(T)$.
$blevel(T)$	Longest path length in G_P' from T to an exit task.
Neighborhood search :	
SL	The current schedule length.
SL_T, SL_{NEW}	The schedule length when T is removed and re-inserted in a new position, respectively.
SL^*, SL_i	The optimal and the initial schedule lengths.
$DF(T)$	Degree of freedom of T.
k_{DF}	No. of times that all the tasks are cycled since the last improvement before the search stops.
ε	Percentage decrease in SL that is counted as an improvement step.
n_I	Total number of improvement steps.
CCR	Mean ratio of communication to computation time.
CP	Critical path length of G ignoring IPC.
N_{SHARE}	Max. no. of consumer tasks sharing each data object.

ACKNOWLEDGMENT. The authors would like to express their gratitude to the Computer Center at the University of Hong Kong for their support of the IBM SP2 system.

REFERENCES

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

[2] T. Yang, A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors", IEEE Trans. Parallel Distrib. Syst. 5, No. 9 (Nov. 1994), 951-967.

[3] S. Darbha, D. P. Agrawal, "Optimal Scheduling Algorithm for Distributed-Memory Machines", IEEE Trans. Parallel Distrib. Syst. 9, No. 1 (Jan. 1998), 87-95.

[4] H. El-Rewini, "Scheduling Parallel Program Tasks onto Arbitrary Target Machines", J. Parallel Distrib. Comput. 9, No. 2 (June 1990), 138-153.

[5] S. Sriram, E. A. Lee, "Statically Scheduling Communication Resources in Multiprocessor DSP Architectures", Conf. Rec. of 28th Asilomar Conf. on Signals, Systems and Computers 2, 1994, 1046-1051.

[6] S. Sriram, E. A. Lee, "Design and Implementation of an Ordered Memory Access Architecture", Proc. of the International Conference on Acoustics Speech and Signal Processing, Apr. 1993, pp. I345-I348.

[7] G. C. Sih, E. A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures", IEEE Trans. Parallel Distrib. Syst. 4, No. 2 (Feb. 1993), 175-187.

[8] L. Wang, H. J. Siegel, V. P. Roychowdhury, A. A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach", Journal of Parallel & Distributed Computing 47, No. 1 (Nov. 1997), 8-22.

[9] L. F. Chao, A. S. LaPaugh, E. H. M. Sha, "Rotation Scheduling: A Loop Pipelining Algorithm", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 16, No. 3 (Mar 1997), 229-239.

[10] S. Tongsima, E. H. M. Sha, N. L. Passos, "Communication-Sensitive Loop Scheduling for DSP Applications", IEEE Trans. on Signal Processing 45, No. 5 (May 1997), 1309-1322.

[11] Y. K. Kwok, I. Ahmad, "Bubble Scheduling: A Quasi Dynamic Algorithm for Static Allocation of Tasks to Parallel Architectures", Proc of 7th Symp. on Parallel & Dist. Proc., Oct. 1995, pp. 36-43.

[12] "ITU-T recommendation H.261: video codec for audiovisual services at px64 kbits", ITU, 1990.

[13] N. H. C. Yung, K. K. Leung, "Parallelization of the H.261 video coding algorithm on the IBM SP2 multiprocessor system", Proc. of 3rd ICA3PP-97, Dec. 1997, 571-578.