논문 2005-42SD-6-7

멀티프로세서 설계를 위한 Linear Data-Flow Graph의 최적화 ILP 알고리즘

(An Optimal ILP Scheduling Algorithm on Linear Data-Flow Graph for Multiprocessor Design)

김 기 복*, 인 치 호**

(Ki-Bog Kim and Chi-Ho Lin)

요 약

본 논문에서는 멀티프로세서 설계를 위한 동질적인 동기 데이터에 의해서 표현되는 LDFG(Linear Data-Flow Graph)의 최적화 ILP (Integer Linear Program)알고리즘을 제안하였다. 이 논문에서 제안된 연산들은 데이터의 종속적 의미를 담고 있지 않으며, 그러한 알고리즘을 위한 스케줄링은 시간 컴파일에 의해서 결정되어지며 충분한 정적 중첩 스케줄링이 고려된다. 제안된 중첩 스케줄링에서는 모두 선형의 동일한 스케줄링과 동일한 처리장치 할당한 것을 의미한다. 본 논문에서는 자원의 제약하에서 스케줄링을 하였으며, 멀티프로세서 설계를 위한 LDFG의 최적화를 위하여 문제를 ILP 공식화하여 해법을 제공하였다. 벤치 마크 실험 결과들은 제안된 스케줄링 방법의 효율성을 검증 하였다.

Abstract

In this paper, we propose an optimal ILP scheduling algorithm for multiprocessor design on LDFG(Linear Data-Flow Graph) that can be represented by homogeneous synchronous data-flow. The proposed computation in this paper does not contain data-dependent, all scheduling decisions for such algorithms can be taken at compile time, only fully static overlapped schedules are considered. It means that all linear have the same schedule and the same processor assignment. In this paper, the resource-constrained problem is addressed, for the LDFG optimization for multiprocessor design problem, formulating ILP solution available to provide optimal solution. The results show that the scheduling method is able to find good quality schedules in reasonable time.

Keywords : ILP, LDFG, Scheduling, Multiprocessor, Overlapped, Constraints

I. Introduction

The basic scheduling techniques are ASAP(As Soon As Possible) scheduling with no constraints of usable Hardware resources' number or action-speed and ALAP(As Late As Possible) scheduling.

Many algorithms in the field of DSP(Digital Signal Processing) belong to this class. An algorithm that contains computations that can be executed simultaneously, offers possibilities of exploiting the parallelism present by implementing it on appropriate hardware such as a multiprocessor system. A schedule is called non overlapped if all operations belonging to the same linear have to finish before the operations of the next linear can start; if operations belonging to different linear can execute simultaneously, the schedule is called overlapped. Overlapped scheduling is sometimes also called loop

* 학생회원, ** 평생회원, 세명대학교 컴퓨터학과
(Department of Computer Science, Semyung University)
접수일자: 2005년3월18일, 수정완료일: 2005년6월14일

folding.[1-3]

The multiprocessor scheduling problem for iterative algorithms has received quite some attention in the last years. However, most of the publications have considered the simplified problem in which the time to communicate data from one processor to another was negligible. When communication delays are taken into account, often target architectures are considered in which setting up a communication or performing the transfer of a single data word requires tens or even hundreds of system clock cycles. In such architecture, it only makes sense to have parallel implementations if the basic tasks performed by a single processor, require a similar number of cycles.[4-7]

In this paper, however, fine-grain parallelism is considered. The basic tasks are primitive operations such as additions and multiplications and the processors in the target architecture are supposed to transfer a data word in just a few cycles. This work presented here extends the one reported in based on integer linear programming. Using ILP has several advantages. First of all, it leads to exact solutions rather than approximate ones. Second, being a general-purpose optimization method, general-purpose tools, so-called ILP solvers, are available to provide the solution once the ILP formulation of a problem is available. An important disadvantage of ILP is its NP-completeness. Computation times associated to ILP usually become too large for cases that are slightly more complex than trivial. However, the ILP formulation of is such that many large benchmarks are solved in a reasonable time.[7-10]

In the rest of this paper, first the scheduling problem will be defined more precisely. Then, the ILP constraints used to solve the problem will be explained. Finally, the experimental results obtained will be presented and compared to other results.

II. Problem Definition

The LDFG consists of a node set V and an edge set E. The node set contains computational nodes

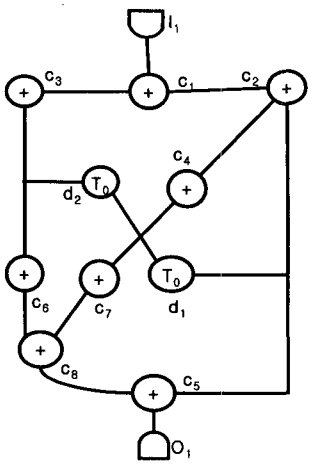


그림 1. 2차 디지털 필터의 LDFG
Fig. 1. LDFG of a second-order digital filter section.

(such as additions or multiplications) collected in the subset N, input nodes contained in the subset IN, output nodes contained in the subset OUT, and delay nodes (a delay node stores data received at its input during the current linear and makes this data available at its output in the next linear).

An example of an LDFG is shown in Figure 1 that represents a second-order digital filter section. In the figure, delay nodes are indicated by T_0. Data-flow computation is best understood by considering tokens that carry data along edges of the LDFG. A computational node starts its computation (one says that it fires) when all its incoming edges have received a token; it consumes these tokens, and produces a token carrying the result of the computation on its outgoing edges when it has finished computing. Input nodes only produce tokens and output nodes only consume tokens.

Although delay nodes help to better visualize a computation an LDFG, it is better to replace paths with delay nodes between computational or I/O nodes by direct edges with delay buffers, one buffer for each delay in the path.

The buffers are supposed to contain initial tokens such that the computation can start. In the rest of this text, it will be assumed that all LDFG have been modified in this fashion. The existence of an edge

from a node $i \in V$ to a node $i_p \in V$ in the modified LDFG will be denoted by i⇒ip. The number of buffers (delays) on the edge will be denoted by Di, ip. Each edge i ⇒ ip indicates a precedence constraint, meaning that i and ip should be scheduled sufficiently apart.

The target architecture consists of a given network of processors or functional units. Each processor can execute all or a subset of the computations contained in the LDFG. It is assumed that a processor can perform a single operation at a time (no internal parallelism). The time necessary to execute a specific operation (e.g. an addition) is the same for all processors. For a computational node $i \in N$, this time is expressed in multiples of the system clock time and denoted by C_i. The hardware is allowed to be pipeline; the pipeline period for i ∈ N is denoted by L_i ($L_i = C_i$) when there is no pipelining). The interconnection network can also be described as a graph with a vertex set P representing functional units (processors) and an edge set W representing unidirectional or bidirectional links. Each link is able to transfer only a single data word at a time. The time to transfer a data word through a link is the same for all links and equals δ cycles of the system clock. Parallel links can be used to model the possibility of transferring multiple data words simultaneously between a pair of functional units. Figure 2 shows some examples of typical architectures that have been used for benchmarking. Note that some of the processors carry the labels IN and OUT: these labels indicate at which processor the inputs to the computations arrive and at which the outputs have to be delivered.

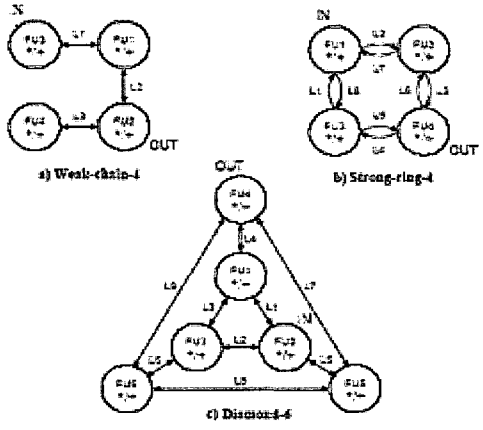


그림 2. 벤치마크를 위해 사용된 멀티프로세서 구성도
Fig. 2. Multiprocessor configurations as used for the benchmarks.

The problem formulation allows a separate processor assignment for each have the input and output nodes in the LDFG as part of the specification. The assigned processor is indicated by FIXki with i ∈ IN ∪ OUT. Also the times at which inputs are available and outputs should become available can be specified separately for each input and output. These times are indicated by FIXj i with i ∈ IN ∪ OUT.

Now that the main aspects of the problem have been presented, the problem to be solved can be formulated. Given are: an LDFG, a target architecture, a linear period and the times and processors for the inputs and outputs.

The goal of an ILP formulation is to check whether a mapping of the LDFG on the target architecture exists that respects the linear period and the timing constraints on inputs and outputs. If it exists, the mapping should be reported. For each processor, it will detail which computation is executed at each clock cycle (if any) and for each link it will detail which data is transported at each clock cycle (if any). Besides, the result should not contain spurious data transports. The main goal is to solve the resource-constrained scheduling problem. Only the LDFG, the target architecture, the input times and the processor assignment for the inputs and outputs is considered given. The primary objective is to find the solution with the fastest linear period T_0. The secondary objective is to minimize the latency for the minimal linear period. When the LDFG has a single input and a single output, the latency is the time difference between the consumption of the input and the production of the output. Otherwise, latency can be defined between

each of the input-output pairs. Values for the output times FIXji with i ∈ OUT can be derived from latencies and the input times. In the rest of this paper, it will be assumed that there is a single input and a single output. The latency associated with this pair will be denoted by L_t. With T_0 and L_t fixed, the scheduling problem becomes a decision problem: there either exists a solution or not.

Each new value combination of the parameters T0 and L_t will result in a different ILP. In order to achieve the main goal, the ILP solver is called in the body of two nested loops: the outer loop increments T_0 until a solution is found and the inner loop varies L_t. The next section provides more information on the values that the parameters can assume.

III. Preparatory Calculations

Although one could directly define the constraints of anILP formulation from the problem description, it is worthwhile to analyze the description first. In this way, the numbers of variables and inequalities in which the variables occur can be reduced significantly. This leads to smaller search spaces and shorter solution times.

In order to reduce the search space, one should consider the precedence constraints. The fact that the input and output times as well as the linear period are known, and that a certain execution order has to be observed, makes that a scheduling rangeman be computed for each computational node. This is an interval consisting of the earliest and latest clock cycles in which the computation can start. The earliest time is called the as soon as possible (ASAP) time while the latest time is called the as late as possible (ALAP) time. For a given LDFG, a T0, input and output times, the scheduling ranges can be computed using longest-path algorithms for graphs, such as the Bellman-Ford algorithm. For each computational node $i \in N$ the ASAP and ALAP times will be denoted by $ASAP_i$ respectively $ALAP_i$. The range will be denoted by $Rx_i : Rx_i = [ASAP_i, ALAP_i]$.

ASAP and ALAP times may have negative values (due to the delay elements that lead to negative offsets in precedence constraints). For reasons of convenience, a constant value is added to all ASAP and ALAP values such that the minimum ASAP value becomes 1. The maximum ALAP value determines the universe of time values that have to be taken into consideration when mapping computations on time instants. ALAPmax will denote this maximum. The scheduling process should also route the transports of data between processors. Note that the mapping of data transports is a one-to-many mapping as opposed to the one-to-one mapping of operations. A transport will in general claim different links at different time instants in order to make sure that data produced by some computation reaches all processors that consume these data. On the other hand, just a single processor needs to be found on which to execute a computational node. Transports also have scheduling ranges. The earliest, start time for the transport of the result of computational node i ∈ N will be denoted by ASAPyi and can easily computed from the ASAP time of the computation itself: $ASAPyi = ASAPi + C_i$

An expression for the latest start time for the same transport denoted by ALAPyi is somewhat more complex and considers all edges i ⇒ ip outgoing from I:

$$ALAP_{yi} = \max_{ip} \{Di, ip\,To + ALAPip - \delta\} \quad (1)$$

The maximum is required here because the transport of data along one edge may in principle be independent from the transport of the same data along other edges. In order to reduce the search space, meaningful values for T_0 and Lt should be tried. There are several factors that contribute to the lower bound of T_0, called the LPB (Linear Period Bound). It is well known that a lower bound on the linear period can be derived from the topology of the LDFG when the LDFG contains loops (each loop

contains at least one delay element in order for the LDFG to be computable). The bound follows from the fact that an algorithm should be executed sufficiently slowly such that data produced in one of the nodes in the loop can return to the node in time for the computation of the next linear. Assuming that all loops in the LDFG are contained in the set L, the bound is given by:

$$LPB_1 = \max_{l \in L} \left(\frac{\sum_{i \in \gamma(l)} G_i}{\sum_{i \to ip \in \eta(l)} D_{i,ip}} \right) \quad (2)$$

The loop that leads to the maximum value in the expression is called the critical loop. It is not necessary to enumerate all loops. Many efficient algorithms exist that can compute the bound.

Another cause for the existence of a lower bound on the linear period lies in the resource-constrained nature of the problem. A processor can at most be utilized for 100%:

$$LPB_2 = \left(\frac{\sum_{i \in N} C_i}{|P|} \right) \quad (3)$$

Yet another trivial lower bound is given by the duration of the IPC time or by the longest pipeline period of a computation on a processor. In a fully static schedule the linear period should be larger that this duration. Combining all bounds, one gets the following expression for the LPB:

$$LPB = \max\left(LPB_1, LPB_2, \delta, \max_{i \in N} Li \right) \quad (4)$$

The maximum of the three lower bound computations gives a minimum value for T_0. This value has been used in the implementation reported in this paper. aking the delay time δ for IPC into account can choose the lower bound for the linear period even tighter.

This will be illustrated for the LDFG of Figure 1 assuming that an addition lasts one clock cycle, a multiplication two and a data transports two. The critical loop is $c4 - c2 - d1$. This loop implies a linear period of 3 (a total duration of computations of 3 to be executed within a single linear) and needs to be executed in a single processor in order not to loose time on IPC. The LDFG has a second loop that shares node $c2$ with the first: c3-c1-c2-d1-d2. The processor executing the critical loop when T_0 = 3 is fully occupied, implying that computations c3 and c1 should be executed on a second processor and that communication to and from c2 on the other processor is necessary. The total time to execute the second loop then becomes 8 (2 additions, 1 multiplication and 2 IPCs) for which two linear are available (2 delay nodes in the loop). The conclusion is that no schedule with T_0 = 3 can exist and that the tighter lower bound of T_0 = 4 should be used.

Another reasoning to come to the same conclusion starts with the observation that the loop c3-c1-c2-d1-d2 contains two delay nodes. The total duration of computations in this loop is 4. Because of the fully-static scheduling requirement, any schedule with $T_0 < 4$ will necessitate that the execution of the loop is distributed across two or more processors. This immediately means that at least two IPC delays have to be added to the total duration of the loop resulting in a total loop execution time of 8 for two linear or 4 one linear. Finding a tighter lower bound in this way is quite complex for the general case. T_0 the knowledge of the authors, no general procedure is known for computing a lower bound for T_0 that takes IPC into account. A lower bound on L_t is quite straightforward to find.

One simply computes the ALAP time of the output. An upper bound on L_t is given by the sum of the durations of all computations in the LDFG plus δ times the number of links in the shortest path from the input to the output processor.

This is an upper bound because all computations can take place on a single processor and sufficient time is available to transport the data from input to output. processor transporting the input from an

incoming link.

$$Minimize: \sum_i \sum_j \sum_l Yi,j,i$$

$$\forall i \in IN \cup N, \forall j \in SLOTS_{yi}, \forall l \in W \quad (5)$$

Ⅳ. ILP Constraints

The variables of the scheduling problem are: $X_{i,j,k}; X_{i,j,k} = 1$ implies that computation $i \in N$ starts at time j on processor k.

$Y_{i,j,l}; Y_{i,j,l} = 1$ implies that data produced by computation $i \in IN \cup N$ is sent at time j through data transportation link i.

The constraints for the scheduling problem have been collected in Figure 3. A short explanation of them will be given here.

Equation 6 expresses the fact that at most one operation is executed at the same time on a processor. For every processor, the entire universe of time instants (1 ≤ j ≤ALAPmax) has to be scanned for conflicts. Note that this universe is never smaller than T0. This means that if a computation i1 has been mapped to start on some time instant j on a processor k, no other computations i2 can start for the duration of the pipeline period Li2 before time instant j or any time instant that is separated a multiple of T_0 from j. For this reason, the formulation not only considers the scheduling range Rxi of an operation, but its iterated version with a period of T_0. The set of time instants that is obtained in this way, is called SLOTSxi. One of the advantages of using this formulation is that no variables Xi;j;k are created for values of j that do not make sense.

Equation 7 states that every operation i has to start on exactly one processor k at exactly one time j.

Equation 8 is the counterpart of Equation 6 for data transports. Some extensions with respect to have been made for handling bidirectional links and IPC times larger than 1. The set Wb contains all bidirectional paired links $lb_u = \{l_u, l_v\}$ and all unidirectional links $lb_w = \{l_w\}$.

Wherever l_u is tested for data conflicts, l_v has to be tested for the same conflicts. The set SLOTSyi is the counterpart of SLOTSxi for the universe of time instants available for transportation.

The requirement that data produced by an operation i passes a link between two processors at most once is expressed by Equation 9. Equation 10 states that a data transport can only leave a processor at some time if it had arrived at the processor at an earlier time or was produced on that processor itself.

In order to capture the topology of the target architecture the following notation is used: frl indicates the processor where a link l originates from and tol the processor that the link connects to.

The correct transportation of input data is controlled by Equation 11. A binary constant is needed to express the availability of the input i on its input processor $FIX_{j,i}$. When input is available after the defined time of input, $FIX_{j,i}$, becomes one, allowing input transportations from $FIX_{j,i}$. A transportation of input i is still possible with a= 0, because the processor can be a non-input processor transporting the input from an incoming link.

Precedence relations between computations and transports, where a computation ip depends on a computation i or on a result of i from an earlier iteration, are handled by Equation 8. The constraint is, in a sense, a counterpart for Equation 10 and states that the execution of a computational node can only start if its inputs have been delivered through a link in time or produced earlier on the same processor. A similar situation for data arriving from an input rather than from another computation is controlled by Equation 13.

Equation 14, finally, takes care that an output is available in time at its destination processor. It is

$$\sum_{i \in N} \{ \sum_{q=0}^{L_i - 1} \{ \sum_{\substack{p=0 \\ (j + p^* T_0 - q) \in SLOTS_x}}^{(ALAP_{\max} - j + q)/T_0} X_{i, j + p^* T_0 - q, k} \} \} \leq 1 \qquad 1 \leq j \leq T_0, \forall k \in P \tag{6}$$

$$\sum_{j \in SLOPSx_i} \sum_{k \in P} X_{i,j,k} = 1 \qquad \forall i \in N \tag{7}$$

$$\sum_{i \in IN \cup N} \sum_{q=0}^{B-1} \sum_{l \in lb} \sum_{\substack{p=0 \\ (j + p^* T_0 - q) \in SLOTSyi}}^{(ALAPy_{\max} - j + q)/T_0} Y_{i, j + p^* T_0 0 q, l} \leq 1 \quad 1 \leq j \leq T_0, \quad \forall lb \in Wb \tag{8}$$

$$\sum_{j \in SLOTSyi} Y_{i,j,l} \leqq 1 \qquad \forall i \in\in \cup N, \ \forall l \in W \tag{9}$$

$$Y_{i,j,l} \leqq \sum_{\substack{ip < j \\ jp \in Ryi \ T_{0lp} = frl}} \sum_{lp} Y_{i,jp,lp} + \alpha \qquad \forall i \in N, \forall j \in Ry_i, \forall l \in W \tag{10}$$

$$Y_{i,j,l} \leqq \{ \sum_{\substack{jp < j \\ jp \in Ry_i}} \sum_{\substack{lp \\ to_{ip} = fr_l}} Y_{i,jp,lp} + \alpha \} \qquad \forall i \in IN, \forall j \in Ry_i, \forall l \in W \tag{11}$$

where :
$$\alpha = \begin{cases} 1 & \text{if } j \geq FIX_{j_i}, \text{and} fr_l = FLXk_i \\ 0 & otherwise \end{cases}$$

$$X_{ip,j,k} \leqq \sum_{\substack{jp < j + D_{i,j}^* T_0 \\ jp \in Ry_i}} \sum_{\substack{l \\ to_{lp} \in k}} Y_{i,jp,l} + \sum_{\substack{jp \leq j + D_{i,j}^* T_0 - C_i \\ jp \in Rx_i}} X_{i,jp,k} \ (\forall i, ip, i \neq ip), \forall j \in Rx_{ip}, \forall k \in P \tag{12}$$

$$X_{ip,j,k} \leqq \sum_{\substack{jp < j + D_{i,ip}^* T_0 \\ jp \in Ry_i}} \sum_{\substack{l \\ to_l = k}} Y_{i.jp,l} + \alpha \quad (\forall i \in IN, \forall ip \in N \mid i \rightarrow ip), \forall j \in Rx_{ip}, \forall k \in P \tag{13}$$

where :
$$\alpha = \begin{cases} 1 & \text{if } j \geq FIX_{j_i}, \text{and } k = FLXk_i \\ 0 & otherwise \end{cases}$$

$$1 \leqq \sum_{\substack{jp < j + D_{i,j}^* T_0 \\ jp \in Ry_i}} \sum_{\substack{l \\ to_{lp} \in k}} Y_{i,jp,l} + \sum_{\substack{jp \leq FIXj_{ip} + D_{i,j}^* T_0 - C_i \\ jp \in Rx_i}} X_{i,jp,k}$$
$$(\forall i \in N, \forall ip \in OUT \mid i \rightarrow ip), k = FIXk_{ip} \tag{14}$$

그림 3. 스케줄링의 ILP 공식을 위한 제약
Fig. 3. The constraints for the ILP formulation of the scheduling problem.

either produced on the destination or has to arrive through an incoming link.

It was stated earlier that the ILP formulation should in principle check whether a solution is possible for a given T_0-L_t combination. As such, no objective function is necessary.

However, the constraints do not prevent solutions in which data is transported through unnecessarily long paths or even to processors that do not need the data. These spurious transports can be prevented by counting all data transports in the objective function of the ILP problem:

V. Experimental Results

The ILP formulation has been applied to the combinations of LDFG and target-architecture benchmarks presented earlier in and listed in Table I. The LDFGs were already given in and are not included in this paper except for second (Figure 1) and elliptic (Figure 4). The target architectures are displayed in Figure 2.

The table 1 gives the durations of additions, multiplications and IPC for each combination. No pipelining is assumed for the execution of the computations and the input time is set to 1 in all benchmarks. The ILP solver BonsaiG was used for the experiments. It can be freely downloaded from NEOS or the problems can be submitted to a remote server running the software.

The Benchmarks A through E were run remotely; the remaining ones locally on an HP 9000/785/C3000 server.

The results are presented in Table I in which they are compared to the results of which were obtained by an approach based on genetic algorithms (GAs). From the results, it can be seen that the T_0 could not be improved in any of the cases. The column labeled IPB contains the value computed according to Equation 1. It can be proved in most cases, following a reasoning as given earlier in this paper, that the T_0 > IPB that was found, is optimal in the presence of IPC. As far as the latency is concerned, several improvements with respect to the GA approach can be reported. The situation is even better than can be directly observed from the table because the GA approach did not route inputs and outputs to specified processors.

The comparison of the CPU times seems to be in favor of the GA approach. One should, however, take into account that the run times reported for GA isaverages of 50 runs and that not all runs resulted in the best results reported in the table. The CPU times for ILP are cumulative counting the times spent on all T0-Lt combinations until a solution is found (the GA approach also dynamically increases these parameters). It has to be observed that Benchmarks M and N were too complex to be solved for ILP.

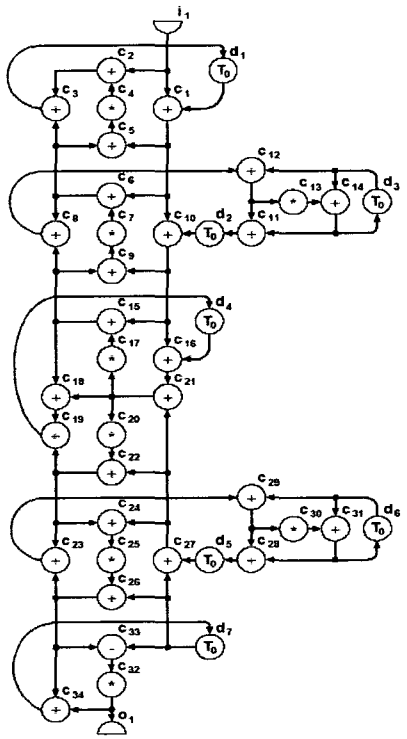


그림 4. 5차 웨이브 디지털 타원형 필터의 LDFG
Fig. 4. The LDFG of a fifth-order wave digital elliptic filter, called elliptic.

표 1. 벤치마크의 결과
Table 1. The results for the benchmarks.

No	ILP			GA			IPB
	T_0	L_t	CPU	T_0	L_t	CPU	
1	4	5	1619s	4	6	11s	3
2	4	5	-	4	9	12s	3
3	3	7	244s	3	7	12s	3
4	5	4	4s	5	5	24s	5
5	6	4	75s	6	4	25s	6
6	3	8	88249s	3	9	27s	3
7	16	9	1s	16	9	33s	16
8	18	10	4s	18	13	43s	16
9	16	30	41s	16	30	27s	14
10	18	31	16s	18	31	33s	14
11	18	15	1062s	18	16	134s	16
12	21	14	720s	21	18	129s	16
13	18	-	-	18	20	111s	16
14	21	-	-	21	14	111s	16

Notes: (1) ILP-solver time-outs(>10,000s)
(2) First evaluation with local BonsaiG
(3) $T_{0,intial}$ = 18 ≠ IPB
(4) $T_{0,intial}$ = 16 ≠ IPB
(5) $T_{0,intial}$ = 21 ≠ IPB
(6) evaluation time > 48 hours

VI. Conclusions

This paper has presented an ILP approach to the overlapped scheduling problem for target architectures with non-negligible IPC times. The approach improves some aspects of the formulation given in. By its nature, ILP provides optimal solutions and has exponentially growing computation times. Optimal results to problems of reasonable size were found after acceptable computation times.

It turns out that it is feasible to apply ILP to benchmarks of reasonable size. Improvements can be expected when the ILP formulation is split into sub problems that are solved separately.

References

[1] E. A. Lee and D.G. Messerschmitt, *"Synchronous data flow, "Proceedings of the IEEE,* vol.75, no.9, pp. 1235-1245, September1987.

[2] S.M. Heemstra de Groot, S.H. Gerez, and O.E. Herrmann,*"Range-chart-guided iterative data-flow-graph scheduling,"* IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 39, pp. 35-{364, May 1992.

[3] K. K. Parhi and D.G. Messerschmitt, *"Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding,* "IEEE Transactions on Computers, vol. 40, no. 2, pp. 178-195, February 1991. PROGRESS 2000 Workshop on Embedded Systems, Utrecht, The Netherlands, October 2000.8

[4] S. H. Gerez, S.M. Heemstra de Groot, E.R. Bonsma, and M.J.M. Heijligers, *"Overlapped scheduling techniques for high-level synthesis and multiprocessor realizations of DSP algorithms,"* in Advanced Techniques for Embedded System Design and Test, J.C. Lopez, R. Hermida, and W. Geisselhardt, Eds., pp. 125-150.

[5] G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man, *"An efficient microcode compiler for application specific DSP processors,"* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 9, no. 9, pp. 925-937, September 1990.

[6] T.F. Lee, A.C.H. Wu, Y.L. Lin, and D.D. Gajski, *"A transformation-based method for loop folding,"* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 13, no. 4, pp. 439-450, April 1994.

[7] V. K. Madisetti, *VLSI Digital Signal Processors, An Introduction to Rapid Prototyping and Design Synthesis,* IEEE Press and Butterworth Heinemann, Boston, 1995.

[8] J. Sanchez and H. Barral, *"Multiprocessor implementation models for adaptive algorithms,"* IEEE Transactions on Signal Processing, vol. 44, no. 9, pp. 2319-2331, September 1996.

[9] S. Sriram and S.S. Bhattacharyya, *Embedded Multiprocessors, Scheduling and Synchronization,* Marcel Dekker, New York, 2000.

[10] D.C. Chen and J.M. Rabaey, *"A reconfigurable multiprocessor IC for rapid prototyping of algorithmic-specific high-speed DSP data paths,"* IEEE Journal of Solid-State Circuits, vol. 27, no.12, pp. 1895-1904, December 1992.

저 자 소 개

인 치 호(평생회원)

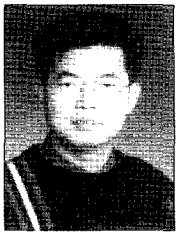
1985년 한양대학교 전자공학과 공학사

1987년 한양대학교 대학원 공학 석사(CAD 전공)

1996년 한양대학교 대학원 공학 박사(CAD 전공)

1992년~현재 세명대학교 컴퓨터학과 부교수

<주관심분야 : VLSI CAD, ASIC 설계, CAD 알고리즘, SOC 설계, RTOS 및 내장형 시스템>



김 기 복(학생회원)

1987년 충북대학교 학사

2001년 세명대학교 대학원 수학 석사(SAS 전공)

2002년 현재 세명대학교 대학원 전산정보학과 박사과정 (CAD 전공)

2002년~현재 (주) 인라인정보통신 전무이사

(주) 이모션 21 이사

<주관심분야 : VLSI CAD, ASIC 설계, CAD 알고리즘>