Fully Static Processor-Optimal Assignment of Data-Flow Graphs

Yeh-Chin Ho and Jong-Chuang Tsay

***Abstract*—The data-flow graph (DFG) is an important graph-theoretic model for multiprocessor implementation of real-time digital signal processing (DSP) algorithms. Given a time schedule for a DFG, we consider the problem of the processor-optimal assignment for a fully static schedule. Previously, the solution of this problem was found by solving an integer programming problem. In this letter, we propose a linear programming approach to solving the problem.**

I. Introduction

THE DATA-FLOW GRAPH (DFG) is an important graph-theoretic model for multiprocessor implementation of real-time digital signal processing (DSP) algorithms [1]–[5]. A DFG is expressed as $G = (V, E, t, d)$, where V is the set of vertices, E is the set of directed edges, $t(v)$ for $v \in V$ is the integral execution time for vertex v, and $d(e)$ for $e \in E$ is the delay count for edge e. The ith iteration of vertex $v \in V$ is denoted by v^i, which we call a *computation*. If there is a directed edge $e = (u, v)$ with delay count $d(e) = j - i$ in G, then there is a precedence relationship from computation u^i to computation v^j.

A *schedule* for a DFG is defined as $\mathcal{S} = (\mathcal{T}, \mathcal{P})$ where $\mathcal{T}$ is the *time schedule* and $\mathcal{P}$ is the *processor assignment*. The starting time step and the processor to execute the computation v^i are denoted by $\mathcal{T}_v^i$ and $\mathcal{P}_v^i$, respectively. A schedule $\mathcal{S} = (\mathcal{T}, \mathcal{P})$ is said to be *fully-static* with *unfolding factor* r and time displacement p if $\mathcal{T}_v^i = \mathcal{T}_v^{i-r} + p$ and $\mathcal{P}_v^i = \mathcal{P}_v^{i-r}$. For this time schedule $\mathcal{T}$, the *iteration period* is $\frac{p}{r}$. It is obvious that the complete schedule can be built by repeating a partial schedule of r consecutive iterations.

Given a time schedule for a DFG, finding a processor-optimal assignment for a fully static schedule is an important problem in multiprocessor implementation of DSP algorithms. Previously, finding such a schedule required solving an integer programming problem [6], [7]. In this paper, we propose a linear programming approach to solving the problem. We first build an *assignment graph* according to the given time schedule, then find a processor-optimal assignment for the assignment graph using a binary search for the minimum number of processors required. For each minimum-number candidate, we propose a linear programming model to find a processor assignment for the assignment graph. Then, simple unfolding can be used to obtain a fully static schedule from this processor assignment.

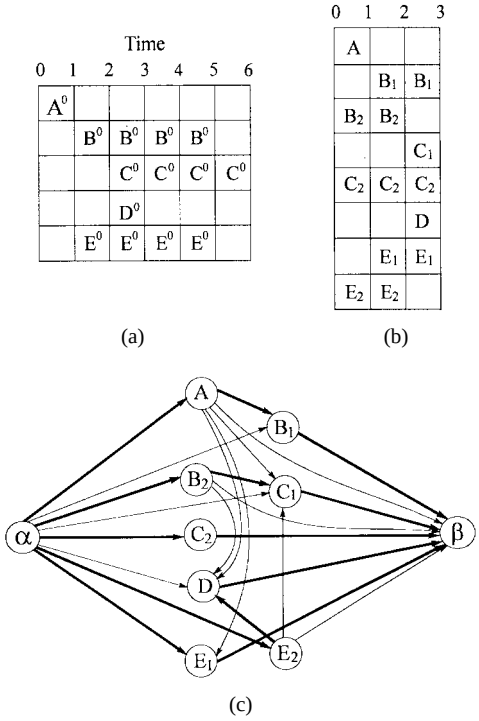

Fig. 1. (a) Time schedule for the 0th iteration of a DFG with five vertices. (b) Folded time schedule for (a). (c) Assignment graph $G^a = (V^a, E^a)$ for (b).

II. The Assignment Graph and Its Processor Assignment

Given a time schedule $\mathcal{T} = \{\mathcal{T}_v^i\}$ for a DFG $G = (V, E, t, d)$ with iteration period $\mathcal{I}_p$, we build an *assignment graph* $G^a = (V^a, E^a)$. We first fold the execution interval for each vertex into $[0, \mathcal{I}_p - 1]$ time slots. This folds each execution interval into segments within the $\mathcal{I}_p$ time slots. We then build an assignment graph $G^a = (V^a, E^a)$ from these segments. Vertex set V^a consists of a vertex for each segment, and two specific vertices α, β. Edge set E^a consists of an edge (u, v) when the time slots occupied by $u, v \in V^a$ do not overlap and u's time slots precede v's time slots, and an edge from α and an edge to β for every vertex in $V^a - \alpha - \beta$. For example, we are given a time schedule $\mathcal{T}_v^i = \mathcal{T}_v^{i-1} + 3$ (iteration period $\mathcal{I}_p = 3$) for a DFG with five vertices A, B, C, D, E. Fig. 1(a) shows the 0th iteration time

Manuscript received August 7, 1996. This work was supported by the National Science Council of Taiwan, R.O.C. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. G. E. Sobelman.

The authors are with the Institute of Computer Science and Information Engineering, College of Electrical Engineering and Computer Science, National Chiao Tung University, Hsinchu 30050, Taiwan, R.O.C. (e-mail: jctsay@csie.nctu.edu.tw).

Publisher Item Identifier S 1070-9908(97)03578-5.

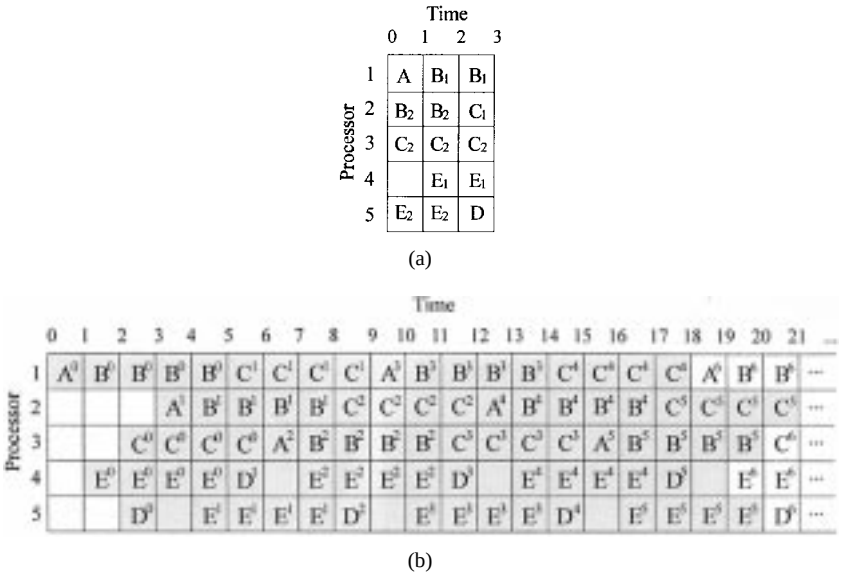

Fig. 2. (a) Processor-optimal assignment for Fig. 1(c). (b). Fully static schedule with iteration period $\mathcal{I}_p = 3$ and unfolding factor 6.

schedule, Fig. 1(b) is its corresponding folded time schedule, and Fig. 1(c) is its corresponding assignment graph (edges in bold lines are described later). From the definition of the assignment, we know that all vertices on any path from α to β can be assigned to the same processor.

Given an assignment graph $G^a = (V^a, E^a)$ and number of processors P, a processor assignment for G^a is a feasible solution under the following constraints:

$$\sum_{\{v|(u,v)\in E^a\}} x_{uv} = 1, \qquad \sum_{\{v|(v,u)\in E^a\}} (-x_{vu}) = -1 \quad u \in V^a - \alpha - \beta \tag{1}$$

$$\sum_{u\in V^a-\alpha-\beta} x_{\alpha u} = P, \qquad \sum_{u\in V^a-\alpha-\beta} (-x_{u\beta}) = -P \tag{2}$$

$$x_{uv} = 0 \text{ or } 1 \qquad (u,v) \in E^a. \tag{3}$$

Here, x_{uv} denotes the flow quantity on the edge (u, v). Constraints in (1) guarantee that exactly one unit of flow enters and leaves every vertex $v \in V^a - \alpha - \beta$, respectively. Constraints in (2) guarantee that there are exactly P units of flow from the source vertex α and to the destination vertex β, respectively. Constraint (3) ensures that the flow on every edge $(u, v) \in E^a$ is zero or one unit. We can obviously find a feasible solution (if it exists) by solving an integer program: any linear objective function of $x'_{uv}s$ under constraints (1)–(3).

Note that each variable x_{uv} appears in constraints (1)–(2) exactly twice, once with a $(+1)$ coefficient and once with a (-1) coefficient. Since these constraints establish a totally unimodular constraint matrix [8], constraint (3) can be relaxed to $0 \le x_{uv} \le 1$ and the integer programming can be reduced to a linear programming. Thus, we can find a feasible processor assignment by solving a polynomial-time linear programming problem rather than a nondeterministic polynomial-complete (NP-complete) integer programming problem [8]. The linear programming model can be used to find a feasible processor assignment for a given number of processors P—if there is one; otherwise, more processors are required. A processor-optimal assignment can be found using binary search for the processor number P within the range $[1, |V^a| - 2]$, which requires solving $\mathcal{O}(\log|V^a|)$ linear programs. The processor-optimal assignment for Fig. 1(c) found by this procedure is shown in Fig. 2(a). We call it a *basic pattern* $\mathcal{B}$. The assignment for processor i, denoted by $\mathcal{B}(i)$, in Fig. 2(a) corresponds to a path (e.g., bold lines) from α to β in Fig. 1(c).

III. The Fully Static Processor-Optimal Assignment

We now consider obtaining a fully static processor-optimal assignment for the DFG $G = (V, E)$ from its basic pattern. In the folding step, a vertex $v \in V$ may be sequentially divided into d vertices (or segments) $v_1, v_2, \cdots, v_d \in V^a$, where vertex v_1 is called the *head*, vertex v_d the *tail*, and vertices $v_2, v_3, \cdots, v_{d-1}$ the *body* of the vertex v. To obtain a fully static schedule, we first find independent processor lists (*independent lists*, for short) using the following procedure.

Step 1: Let all processors be unprocessed and the number of independent list $n = 0$.

Step 2: *If* there is an unprocessed processor *then* increment n; *else stop*.

Step 3: *If* there is an unprocessed processor p without any tail or body, *then* select it; otherwise, select an unprocessed processor p at random.

Step 4: Add processor p to the nth independent list, and set processor p as processed. *If* processor p does not contain any head or body, *then goto* Step 2.

Step 5: Select the processor p' containing the body or tail following the head or body in processor p. *If* p' is unprocessed, *then* let $p = p'$ and *goto* Step 4; otherwise, *goto* Step 2.

Since the above procedure processes every processor exactly once, the time complexity is $\mathcal{O}(P)$ for P processors. We show operation of the above procedure by the example in Fig. 2(a). In Fig. 2(a), vertices B_1, C_1, and E_1 are the heads in processors 1, 2, and 4, respectively. And vertices B_2, C_2, and

E_2 are the tails in processors 2, 3, and 5, respectively. First, we select processor 1, which does not contain any tail or body, and add it to independent list 1. Next, we add processor 2, which contains the tail B_2 following the head B_1 in processor 1, to independent list 1. Because processor 3 has the tail C_2 following the head C_1 in processor 2, we add it to independent list 1. Since processor 3 does not contain any head or body, we complete independent list 1 with processors 1, 2, and 3. In similar fashion, we complete independent list 2 with processors 4 and 5.

For each independent list n with x_n processors $i_1, i_2, \cdots, i_{x_n}$, we can build an *assignment pattern*, covering $x_n\mathcal{I}_p$ time steps, by concatenating sequentially the assignments $\mathcal{B}(i_1), \mathcal{B}(i_2), \cdots, \mathcal{B}(i_{x_n})$. Then the assignment pattern is repeated every $x_n\mathcal{I}_p$ time steps to form a *schedule pattern*. To obtain a schedule for these x_n processors, each processor is assigned by the same schedule pattern except for a time lag of $\mathcal{I}_p$ time steps between consecutive processors. According to this procedure, if there are m independent lists, then the obtained schedule is fully static with an unfolding factor $lcm(x_1, x_2, \ldots, x_m)$. In this assignment procedure, since the number of processors $x_1 + x_2 + \cdots + x_n$ assigned is equal to the number of processors assigned in the basic pattern, the fully static schedule obtained is processor optimal. In Fig. 2(a), for example, there are two independent lists with $x_1 = 3$ and $x_2 = 2$ processors, respectively. The assignment patterns are ABBBBCCCC and ⊔EEEED for independent lists 1 and 2, respectively. Thus, the obtained schedule is fully static and processor optimal with an unfolding factor $lcm(3, 2) = 6$. The complete schedule can be built by repeating a partial schedule of six consecutive iterations [shaded region in Fig. 2(b)] every $18 = \mathcal{I}_p * 6$ time steps.

IV. Conclusion

Given a time schedule for a DFG, we have proposed an efficient method for finding a fully static schedule using a minimum number of processors. The processor-optimal assignment is obtained by solving $\mathcal{O}(\log|V^a|)$ polynomial-time linear programming problems rather than NP-complete integer programming problems as in the past.

References

[1] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, pp. 178–195, Feb. 1991.

[2] S. M. Heemstra de Groot, S. H. Gerez, and O. E. Herrmann, "Range-chart-guided iterative data-flow graph scheduling," *IEEE Trans. Circuits Syst. I*, vol. 39, pp. 351–364, May 1992.

[3] L. Jeng and L. Chen, "Rate-optimal DSP synthesis by pipeline and minimum unfolding," *IEEE Trans. VLSI Syst.*, vol. 2, pp. 81–88, Mar. 1994.

[4] K. K. Parhi, "High-level algorithm and architecture transformations for DSP synthesis," *J. VLSI Signal Processing*, vol. 9, no. 1–2, pp. 121–143, 1995.

[5] D. Wang and Y. H. Hu, "Multiprocessor implementation of real-time DSP algorithms," *IEEE Trans. VLSI Syst.*, vol. 3, pp. 393–403, Sept. 1995.

[6] L. E. Lucke and K. K. Parhi, "Generalized ILP scheduling and allocation for high-level DSP synthesis," in *Proc. IEEE 1993 Custom Integrated Circuits Conf.*, 1993, pp. 5.4.1–5.4.4.

[7] K. Ito, L. E. Lucke, and K. K. Parhi, "Module selection and data format converson for cost-optimal DSP synthesis," in *Proc. 1994 IEEE/ACM Int. Conf. Computer-Aided Design*, 1994, pp. 322–329.

[8] A. Schrijver, *Theory of Linear and Integer Programming*. New York: Wiley, 1986.