Vol. 40 No. 5 Transactions of Information Processing Society of Japan May 1999

*Regular Paper*

# An Optimal Two-Processor Scheduling for a Class of Program Nets via a Hybrid Priority List

QI-WEI GE† and NAOMI YOSHIOKA††

This paper deals with two-processor scheduling for acyclic SWITCH-less program nets that is a graph representation of data-flow programs. A task graph is a special case of acyclic SWITCH-less program net and the important difference between a program net and a task graph is that a program net allows the nodes to fire more than once while a task graph require each of its nodes to fire exactly once. Hence the multiprocessor scheduling problem for general acyclic SWITCH-less program nets is $NP$-hard in a strong sense. In this paper, we require the program nets to satisfy: (i) all the nodes have the same firing time and (ii) all the AND-nodes possess single input edge. For such a class of program nets, we first propose a scheduling method using hybrid priority list that consists of both dynamic and static lists, and then prove optimality of the schedules generated by the hybrid priority list.

## 1. Introduction

Multiprocessor systems have been widely used in a variety of computer applications, such as information processing, control of robots and high-speed simulation of dynamic systems[1]; for their potential effectiveness in decreasing the computation times of programs. To maximally achieve the advantage of a multiprocessor system, it is desirable to find out an efficient way to schedule the processors in executing the tasks of programs in order to attain the minimum execution time.

Usually the problem of multiprocessor scheduling is, given with processors and a program that is represented as a task graph (an acyclic directed graph) with its nodes and edges representing tasks and precedence relations between the tasks respectively, to determine the order of tasks' (called node hereafter) execution assigned to the processors to minimize the total execution time for the task graph. However, this problem is extremely difficult and generally intractable[2], which has been known as NP-hard problem[3),4]. Only for two special cases polynomial algorithms were found: (i) the first is proposed by Hu[5] and to schedule execution of rooted task graphs with same node execution time by using arbitrary processors; (ii) the second is by Coffman and Graham[6] and to schedule execution of general task graphs also with same node execution time but by two processors. For this reason multiprocessor scheduling is usually approached by heuristic methods[7].

Till now in dealing with multiprocessor scheduling, task graphs, as a program representation by looking at the control flows, have such a limited characteristic that each node is allowed to be executed only once. However for parallel computers, such as data-flow computers[8)~10], data-flow of the programs becomes ever important in analysis and evaluation of program executions and therefore the programs are usually represented as data-flow program nets[11)~14] (program nets or nets for short) by taking notice of data flows. So that each node of a program net is generally executed more than once. Generally a program net is a variation of Petri nets[15] and consists of three types of nodes: AND-node, OR-node and SWITCH-node, that respectively represent arithmetic/logical, data merge and context switch operations. In this paper, we are interested in list scheduling for a class of acyclic SWITCH-less program nets (of no SWITCH-nodes) as to be detailedly stated later.

Task graph is a special case of acyclic SWITCH-less nets consisting of only AND-nodes and hence the scheduling problem for acyclic SWITCH-less nets is $NP$-hard in strong sense as well. As list scheduling for task graphs, $CP$ (Critical Path) method was successively applied in the optimal schedulings[5),6] as stated just now and has also been shown generally effective for other cases[16]. Later, its improved $CP/MISF$ (Most Immediate Successors First) was proposed[17] to avoid occurring of worse schedule when multiple nodes have the same level from sink nodes of the task graphs. For the

† Faculty of Education, Yamaguchi University
†† Fujitsu Ten Limited

case of node executions more than once, static list scheduling and GA scheduling have been studied for program nets with arbitrary processors[18], and also timed Petri nets' schedulings have been studied theoretically and experimentally for repetitive executions[19),20]. However all these methods, except for the cases as dealt with in Refs. 5) and 6), are heuristic and give only the approximate results. The purpose of this paper is to propose an optimal scheduling for program nets

In this paper, we deal with nonpreemptive two-processor scheduling problem for a class of acyclic SWITCH-less program nets with same node execution (called firing thereafter) time, of which each AND-node is allowed to possess at most one input edge. First we investigate the bottlenecks when scheduling is carried out by using a static priority list constructed according to longest distances (that is $CP$ method for task graphs). Then to dissolve the bottlenecks, we add a dynamic priority list, prior to the static one, to propose a hybrid priority list. Finally we show how such a hybrid priority list gives optimal schedules of the program nets.

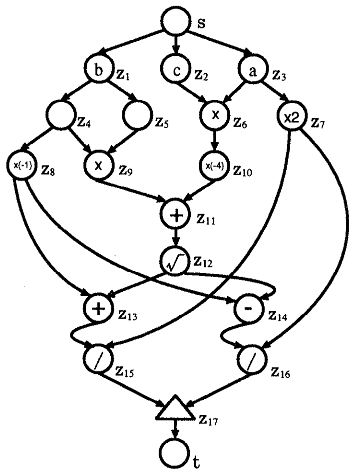



**Fig. 1** A program net to solve quadratic equation: "$ax^2+bx+c=0$".

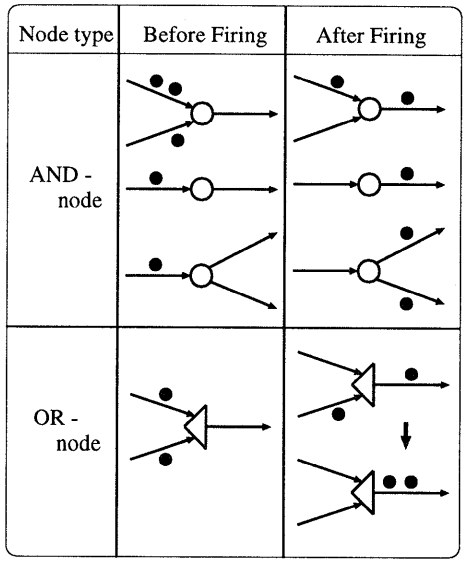



**Fig. 2** The firing rules of the nodes of a SWITCH-less program net.

## 2. Preliminary

In a *program net* $PN=(N,E,\alpha,\beta)$, a *node* $z\in N$ is one of three types: *AND-node* (○), *OR-node* (△) and *SWITCH-node* (▽°), representing *arithmetic/logical*, *data merge* and *context switch* operations respectively. An *edge* $e\in E$ represents a *FIFO token transmission channel* and possesses two *thresholds*, $\alpha_e$ and $\beta_e$ that represent numbers of tokens taken off from and deposited on itself by a firing of its output and input nodes respectively. A *token* (•) represents a *single datum* and *token distribution* $d^\tau=(d^\tau_{e_1},\cdots,d^\tau_{e_{|E|}})$ expresses token numbers on each $e_i$ at time $\tau$.

There are two special AND-nodes, *start node* $s$ (source) firing exactly once and *termination node* $t$ (sink). For each $z$, there are two directed paths from $s$ to $z$ and from $z$ to $t$. A *firing* of node $z_i$ occurs at integer time epochs, $\tau=0,1,\cdots$, and takes $\gamma_i$ unit times, called *(node) firing time* and supposed to be integer. Once a node fired, it can not be fired any more before finishing its current firing. Generally each node has at most two input and two output edges. In this paper, program nets are considered as usual as *acyclic SWITCH-less net* (of no SWITCH-nodes) with *initial token distribution* $d^0=0$, unity node firing time and unity thresholds $\alpha_e\equiv\beta_e\equiv 1$. Further we require AND-nodes (except $s$) to possess single input edge. **Figures 1** and **2** show an SWITCH-less net and the node firing rules respectively. For the detailed description of program nets, the reader is referred to Refs. 13) and 14).

The following basic definitions are given for general SWITCH-less program nets.

**Definition 1:** Let $PN$ be a program net.

**(i)** Node $z$ is called *firable* and denoted $d^\tau$-*firable* with respect to $d^\tau$, iff (a) for AND-node $z$, each its input edge $e$ satisfies $d^\tau_e\geq 1$;

(b) for OR-node $z$, one of its input edges $e$ satisfies $d_e^{\tau} \geq 1$.

**(ii)** A firable AND-node fires to take off one token from each of its input edges and deposit one token on each of its output edges; while a firable OR-node fires to take one token from any one of its input edges and deposit one token on each of its output edge.

**(iii)** A node sequence $\sigma = z_1 z_2 \cdots z_k$ of $PN$ is called *firing sequence* iff $PN$ can be fired with single processor in the order of $\sigma$ so that $z_i$ is $d^{\tau_{i-1}}$-firable and $d^{\tau_i}$ is resulted from $d^{\tau_{i-1}}$ by firing $z_i$.

**(iv)** $\sigma$ is called *terminating* iff no node is $d^{\tau_k}$-firable. $PN$ is called *terminating* iff all of its firing sequences are of finite length $k < \infty$. □

Since the program nets dealt with in this paper are acyclic, they are always terminating.

**Definition 2:** Let $\sigma$ and $f(z)$ be a terminating firing sequence and the firing number of node $z$ in $\sigma$, respectively. $f(z)$ is called the *maximum firing number* of $z$, denoted as $\overline{f}(z)$, iff there is no $\sigma'$ such that $f'(z)$ satisfies $f'(z) < f(z)$. □

It has been known[13] that, for any two terminating firing sequences of a SWITCH-less net, $\sigma'$ and $\sigma$, $f'(z) = f(z) = \overline{f}(z)$ holds. That is as only to fire each node $z$ of a SWITCH-less net $\overline{f}(z)$ times we need not especially pay attention to the firing orders of the nodes. The problem of scheduling a program net $PN$ in this paper is to fire nodes of $PN$ with two processors so that firing rules are obeyed and all the nodes ($\{z_i\}$) are fired maximum firings ($\{\overline{f}(z_i)\}$) individually in shortest possible time. The time costed is called *firing completion time*.

The following definitions are given for the use in this paper.

**Definition 3:** Let $z_1$ and $z_2$ be two nodes of a program net $PN = (N, E)$.

**(i)** $z_1$ is called *predecessor* of $z_2$ or $z_2$ is called *successor* of $z_1$ iff there exists a path (directed path) from $z_1$ to $z_2$. The sets of predecessors and successors of a node $z$, except start node $s$ and termination node $t$ respectively, are denoted as $Pre(z)$ and $Suc(z)$ respectively;

**(ii)** $z_1$ ($z_2$) is called *immediate predecessor* (*successor*) of $z_2$ ($z_1$) iff $(z_1, z_2) \in E$ is satisfied. The sets of immediate predecessors and successors of a node $z$, except $s$ and $t$ respectively, are denoted as $IP(z)$ and $IS(z)$ respectively;

**(iii)** $z_1$ (or $z_2$) is called *irrelative* node of $z_2$ (or $z_1$) iff there exist no any paths from $z_1$ to $z_2$ and from $z_2$ to $z_1$. The set of irrelative nodes of $z$ is denoted as $Ir(z)$. □

**Definition 4:** Let $F(\tau)$ and $Dis(z)$ respectively denote the set of firable nodes at time $\tau$ and the maximum distance from $z$ to termination node $t$ by taking into account the edge numbers. □

## 3. A hybrid priority list

In this section, we first apply a static priority list to program nets to investigate the bottlenecks. Then to dissolve the bottlenecks we add a dynamic priority list, that is prior to the static one, to give a hybrid priority list.

The static priority list is concretely constructed by arranging the nodes in descending order of $Dis(z)$ and is denoted as $L_{all}$ (including all the nodes), which is in fact adopted in critical path method for task graphs. Let's show an example by applying $L_{all}$ to a program net shown in **Fig. 3**. For this net, the priority list is $L_{all} = s z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8 z_9 z_{10} z_{11} z_{12} t$. By assigning node firings to two processors according to $L_{all}$, we can obtain the schedule with firing completion time $T_1 = 30$ as also shown in Fig. 3. In this schedule, the processor $P_2$ is idle during the time interval between $\tau = 6$ (including $\tau = 6$) and $\tau = 7$ (denoted as $[6, 7)$).

Looking precisely at the schedule as well as the situation of the net, we find that

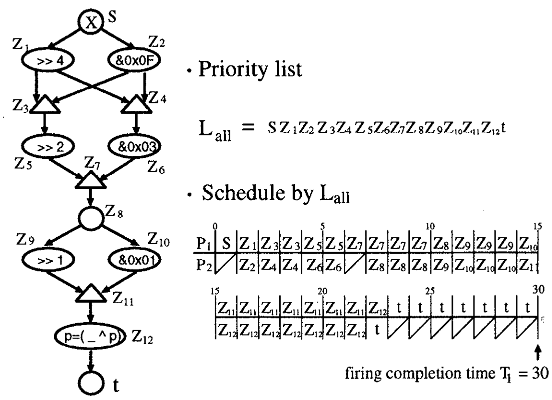
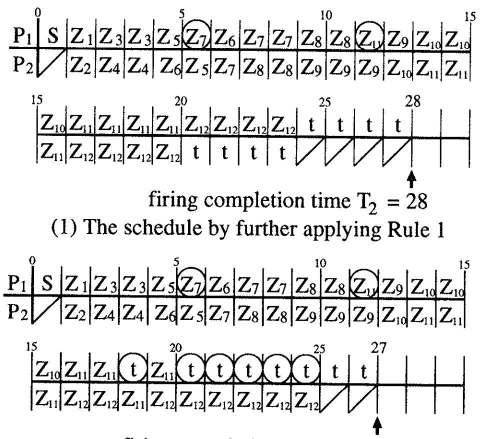



**Fig. 3** A program net to create parity bit for a binary data with 8 bits, $X = x_7 x_6 \ldots x_0$, and the schedule by $L_{all}$, where (i) "$>> n$", "&" and "^" show logical operations of right shift for $n$ times, AND and XOR respectively; (ii) inside of node $z_{12}$, "_" shows its 1 bit input data and the initial value of "$p$" is 1; (iii) the 8th output data of node $z_{12}$ is the parity bit.



firing completion time $T_2 = 28$
(1) The schedule by further applying Rule 1

firing completion time $T_3 = 27$
(2) The schedule by further applying Rule 2.

**Fig. 4** The schedules by further applying Rules 1, 2.

at time $\tau=5$ there are three firable nodes, $F(5)=\{z_5, z_6, z_7\}$, and nodes $z_5$ and $z_6$ are selected to fire at that time other than the OR-node $z_7$ due to that $z_5$ and $z_6$ are input nodes of $z_7$ and are prior to $z_7$ from $L_{all}$. However we might have a choice to select the OR-node $z_7$ instead of $z_6$, so that at time $\tau=6$ there would be another firable node $z_6$ besides $z_7$. Thus $P_2$ would not be idle during $[6, 7)$, which means such OR-node should be selected to fire priorly. Hence we add a priority rule as follows:

**Rule 1:** An OR-node $o_i$ is prior to its two input nodes, $x_i$ and $y_i$, at a moment $\tau$ if (i) $x_i$ and $y_i$ have $o_i$ as their unique immediate predecessor; (ii) $o_i$, $x_i$ and $y_i$ are the only firable nodes at $\tau$.

Applying Rule 1 and $L_{all}$ to the net of Fig. 3, the schedule becomes one as shown in **Fig. 4** (1), in which the nodes enclosed with a circle are selected by Rule 1. Obviously the firing completion time $T_2=28$ is shorter than $T_1=30$. However even in this schedule the processor $P_2$ keeps idle for four unit times from the time $\tau=24$. This situation arises due to the accumulation of too many tokens on the input edge of $t$ and hence we need to dissolve this by firing $t$ earlier as following rule:

**Rule 2:** To priorly fire $t$ at a moment $\tau$ if there are two tokens on the input edge of $t$ at $\tau$.

By further applying Rule 2, the schedule becomes one as shown in Fig. 4 (2), in which node $t$ enclosed with a circle is selected by Rule 2. Surely the schedule in Fig. 4 (2) gives the optimal firing completion time $T_3=27$.

Generalizing the above discussions, we propose a hybrid priority list including both dynamic and static lists as follows:

**Definition 5:** A hybrid priority list is a node list concatenated from 3 priority lists, $L^*=L_t \cdot L_o \cdot L_{all}$, where $L_t$, $L_o$ and $L_{all}$ are called *t-priority list*, *OR-priority list* and *all-priority list* respectively and defined as follows:

**(i)** $L_t=\psi_t(\tau)$ and $\psi_t(\tau)$ satisfies:
$$\psi_t(\tau)=\begin{cases} t: & \text{if } d_e^\tau \geq 2 \\ \phi: & \text{otherwise} \end{cases}$$
where $e$ is the input edge of $t$;

**(ii)** $L_o=\psi_{o_1}(\tau)\psi_{o_2}(\tau)\cdots\psi_{o_k}(\tau)$ and $o_i$ is such an OR-node that its 2 input nodes, $x_i$ and $y_i$, satisfy $IS(x_i)=IS(y_i)=\{o_i\}$. And the order of the OR-nodes satisfies $Dis(o_i)\geq Dis(o_j)$ if $i<j$ and $\psi_{o_i}(\tau)$ satisfies:
$$\psi_{o_i}(\tau)=\begin{cases} o_i: & \text{if } F(\tau)=\{o_i, x_i, y_i\} \\ \phi: & \text{otherwise;} \end{cases}$$

**(iii)** $L_{all}=z_1 z_2 \cdots z_n$ includes all the nodes satisfying $Dis(z_i)\geq Dis(z_j)$ if $i<j$. □

**Definition 6:** Let $S_{L^*}$ denote a schedule generated by a priority list $L^*$.

**(i)** The firings of $t$ and OR-node $z$ in $S_{L^*}$ are called *t-priority firing* and *OR-priority firing* respectively, iff $t$ and $z$ are selected to fire from $L_t$ and $L_o$ respectively;

**(ii)** Node $z_1$ is prior to node $z_2$ at time $\tau$, iff $z_1, z_2 \in F(\tau)$ and the first appearance of $z_1$ is before $z_2$ in $L^*$. □

It is not difficult to verify that the time complexity in scheduling a program net by the hybrid priority list $L^*$ is $O(|N|^2 \bar{f}_m)$, where $\bar{f}_{max}= \max\{\bar{f}(z_i)\}$ is an invariant element for a given net.

## 4. Optimality of list scheduling by $L^*$

We are to show the schedule generated by $L^*$ gives optimal firing completion time.

Generally a schedule generated by any method is expressed as **Fig. 5**, in which a program net begins its firing at time $\tau_1=0$ and ends at $\tau_{\kappa+1}$. The notions of this schedule are defined in the following.

**Definition 7:** Let $S_L$ be a schedule generated by a priority list $L$ as shown in Fig. 5.

**(i)** $P_1$ is priorly assigned and when $P_2$ is idle it is denoted by "/" in $S_L$. $\tau_2$ is first time that $P_2$ is not idle, $\tau_3$ is first time that $P_2$ becomes again idle after $\tau_2$ and so on in $S_L$;

**(ii)** A part of $S_L$ during time from $\tau_j$ till $\tau_{j+1}$ (denoted as $[\tau_j, \tau_{j+1})$) is called *j-th span* and the time interval, $l_j=\tau_{j+1}-\tau_j$, is called *j-th span time*. Number of total spans is

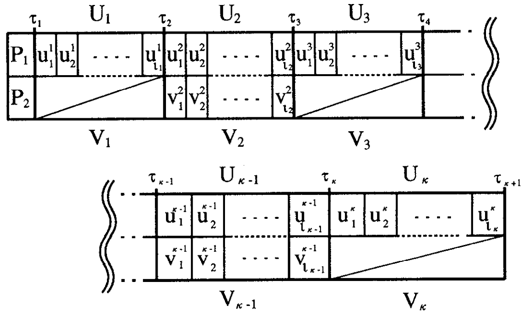


**Fig. 5** General expression of a schedule.

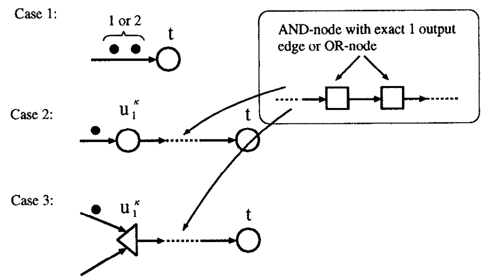



**Fig. 6** The structure of nodes in $U_\kappa$ and the token distribution at $\tau_\kappa$.

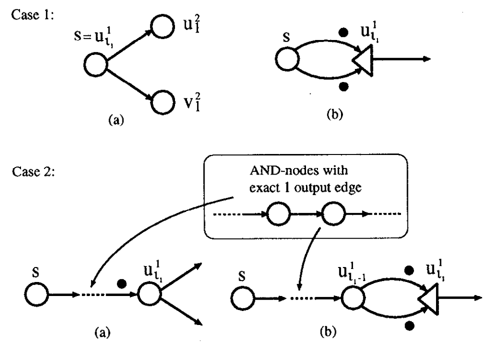



**Fig. 7** The structure of nodes in $U_1$ and the token distribution at $\tau_2-1$.

odd, $\kappa=2i+1$;

**(iii)** $U_j$ and $V_j$ show nodes of $j$-th span, $u_1^j\cdots u_{l_j}^j$ and $v_1^j\cdots v_{l_j}^j$, that are assigned to processors $P_1$ and $P_2$ respectively during $[\tau_j, \tau_{j+1})$, and $u_i^j$ and $v_i^j$ indicate individual nodes.

**(iv)** $S_L$ is optimal iff $S_L$ gives minimum firing completion time of $PN$. □

Since each node $z$ appears $\overline{f}(z)$ times in a schedule $S_L$, $u_i^j$ (or $v_i^j$) may probably denote $h$-th firing of a node $z$ ($1\leq h\leq \overline{f}(z)$). Hence hereafter when we say firing of $u_i^j$, we mean such $h$-th firing of node $z$. In the following discussions, we suppose the schedule shown in Fig. 5 is $S_{L^*}$ generated by $L^*$ for program net $PN$.

**Lemma 1:** If $u_1^\kappa$ of $U_\kappa$ can not be fired before $\tau_\kappa$ for any scheduling, then $S_{L^*}$ is optimal. □

**Proof:** It is obvious that at time $\tau_\kappa$ tokens appear only on the input edge of $u_1^\kappa$ and the structure related to the nodes in $U_\kappa$ of $\kappa$-th span is one of three cases shown in **Fig. 6**.

**Case 1:** $u_1^\kappa$ is termination node $t$ and possesses one or two input tokens, and $l_\kappa=2$.

**Case 2:** $u_1^\kappa$ is AND-node with one output edge and possesses one input token. The nodes following $u_1^\kappa$ are AND-nodes with one output edge or OR-nodes.

**Case 3:** $u_1^\kappa$ is OR-node and possesses one input token. The nodes following $u_1^\kappa$ are as same as Case 2.

It is obvious that this lemma holds for Case 1. For Cases 2 and 3, $u_i^\kappa$ must be fired after firing of $u_{i-1}^\kappa$ and hence $u_{l_\kappa}^\kappa$ can not be fired before $\tau_{\kappa+1}-1$ if $u_1^\kappa$ can not be fired before $\tau_\kappa$ for any scheduling. Therefore $S_{L^*}$ is optimal. **Q.E.D**

From Lemma 1, to show $S_{L^*}$ is optimal, we need to prove that $u_1^\kappa$ can not be fired before $\tau_\kappa$ for any schedulings. At first we see the firings of first span.

**Lemma 2:** Each node $u_i^1$ of $U_1=u_1^1\cdots u_{l_1}^1$ of first span in $S_{L^*}$ can not be fired before $\tau_1+i-1$ for any schedulings. □

**Proof:** The structure related to nodes in $U_1$ is one of the following cases as shown in **Fig. 7**.

**Case 1:** If $u_1^1$ (start node $s$) possesses two output edges, then the structures are as follows:

**(a)** $l_1=1$ and $|IS(u_1^1)|=2$; or

**(b)** $l_1=2$, $IS(u_1^1)=\{u_{l_1}^1\}$ and $u_{l_1}^1$ is OR-node.

**Case 2:** If there is only one output edge of $u_1^1$, then the structures are as follows:

**(a)** $l_1\geq 2$ and all the nodes are AND-node satisfying $IS(u_i^1)=\{u_{i+1}^1\}$ ($1\leq i\leq l_1-1$) and $|IS(u_{l_1}^1)|=2$; or

**(b)** $l_1\geq 3$, nodes $u_1^1,\cdots,u_{l_1-1}^1$ are AND-node and $u_{l_1}^1$ is OR-node; and these nodes satisfy $IS(u_i^1)=\{u_{i+1}^1\}$ ($1\leq i\leq l_1-1$).

Obviously, this lemma holds for any one of the cases. **Q.E.D**

Now let us to see $u_1^3$'s firing of the third span. We have the following theorem.

**Theorem 1:** $S_{L^*}$ is optimal if $Dis(u_1^3)\leq 1$. □

We need the following lemma to prove the above theorem.

**Lemma 3:** If processor $P_2$ is idle for only once or twice from time $\tau_2$, $S_{L^*}$ is optimal. □

**Proof:** The first span of $S_{L^*}$ is optimal according to Lemma 2 and further during the firing of $u^{\kappa}_{l_\kappa}$, the last firing in $S_{L^*}$, $P_2$ has to be idle. Then if $P_2$ is not idle during $[\tau_2, \tau_\kappa+l_\kappa-1)$ $S_{L^*}$ is obviously optimal. When $P_2$ is idle only at a time $\tau$ during $[\tau_2, \tau_\kappa+l_\kappa-1)$, then the total firing number of the nodes during this period of time is odd, which means $P_2$ has to be idle at some time. Therefore $S_{L^*}$ is optimal. **Q.E.D**

**Proof of Theorem 1:** We are to prove this theorem by dividing the cases into $Dis(u^3_1)=0$ and $Dis(u^3_1)=1$.

**Case 1:** Let $Dis(u^3_1)=0$, which means $u^3_1=t$. In this case, $u^3_1$ must have at most two input tokens at $\tau_3$, since $t$-priority firing must occur if it has two input tokens. Thus $\kappa=3$ and $l_\kappa\leq 2$, i.e. $S_{L^*}$ is optimal from Lemma 3.

**Case 2:** Let $Dis(u^3_1)=1$, which means $IS(u^3_1)=\{t\}$. Note that in the following, we use the fact that node $u^3_1$ never fires at $\tau_3-1$; otherwise at $\tau_3$ nodes $u^3_1$ and $t$ are firable.

If $u^3_1$ has only one input token at $\tau_3$, then $l_\kappa=2$ and hence $S_{L^*}$ is optimal from Lemma 3. So we need to see when $u^3_1$ has more than one input token at $\tau_3$.

**(a)** It is impossible that tokens only exist on the same input edge of $u^3_1$; otherwise $P_2$ is not idle at $\tau_3$. The reason is that: (i) if the tokens only exist on the same input edge, $u^3_1$ must be firable and but is not selected to fire at $\tau_3-1$; and then (ii) at least one node fired at $\tau_3-1$ must be $t$ selected by $L_t$ or node $z\notin IP(u^3_1)$; which means $t$ or the node in $IS(z)$ is firable at $\tau_3$ besides $u^3_1$.

**(b)** Let $u^3_1$ be an OR-node with tokens appearing on both of its two input edges. In this case, there is exact one token on each its input edge; otherwise $P_2$ is not idle at $\tau_3$. The reason is almost the same as (a) that: (i) $u^3_1$ must be firable and but is not selected to fire at $\tau_3-1$; and then (ii) at least one node fired at $\tau_3-1$ must be $t$ selected by $L_t$ or node $z\notin IP(u^3_1)$ or $z\in IP(u^3_1)$ with two output nodes (i.e. no $OR$-priority firing occurs for $u^3_1$); which means $t$ or a node in $IS(z)$ (except $u^3_1$) is firable at $\tau_3$. Therefore $\kappa=5$, $l_3=1$ and $l_5=1$ hold. This means $S_{L^*}$ is optimal from Lemma 3. **Q.E.D**

The following theorem plays an important role in proving optimality of $S_{L^*}$ for the case of $Dis(u^3_1)\geq 2$, which has been proved satisfied in Ref. 21).

**Theorem 2**[21]: The first firing of third span of $S_{L^*}$ ($u^3_1$'s firing) can not be done before $\tau_3$ for any scheduling if the maximum distance of the related node is longer than 1 ($Dis(u^3_1)\geq 2$). □

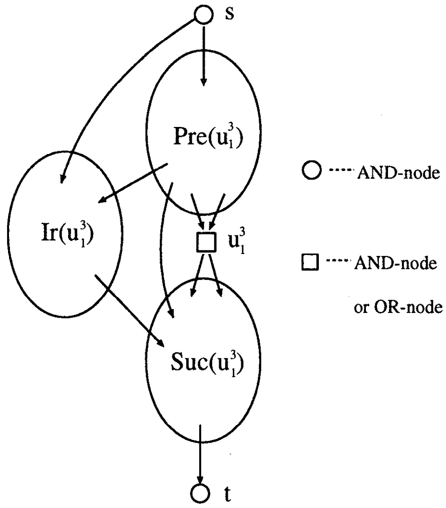



**Fig. 8** Structural relations between $u^3_1$ and all the other nodes.

Following result is immediate from Lemmas 1, 2 and Theorems 1, 2.

**Corollary 1:** If $\kappa=1, 3$ or $Dis(u^3_1)\leq 1$, $S_{L^*}$ is optimal. □

Hereafter we need only to show the optimality for $\kappa\geq 5$ and $Dis(u^3_1)\geq 2$. **Figure 8** shows connection of nodes by taking notice of $u^3_1$. It is always true no matter $\kappa=3$ or $\kappa\geq 5$, that (i) at $\tau_3$ there is no token on any edges except the input edge(s) of $u^3_1$; and (ii) the nodes in $Pre(u^3_1)\cup Ir(u^3_1)\cup\{s\}$ have finished all their firings and will never fire after time $\tau_3$. As shown in **Fig. 9**, $u^3_1$ may be an AND-node or an OR-node and may have at most one token on each of its input edges, which is just generated by the firing of its immediate predecessor(s) at $\tau_3-1$. Note that if the token number is more than one, or the token is not generated at $\tau_3-1$, then $P_2$ is not idle at time $\tau_3$ as similar as has been stated just now in Case 2 of the proof of Theorem 1. So for a $PN$ whose schedule by $L^*$ has $\kappa\geq 5$ and $Dis(u^3_1)\geq 2$, we can transform it into a new one $PN'$ by following operations:

**(1)** Delete all nodes in $Pre(u^3_1)\cup Ir(u^3_1)\cup\{s\}$;

**(2)** Replace the OR-node possessing exact one input edge with AND-node;

**(3)** Add a new start node $s$ and take off the tokens as shown in Fig. 9.

The following result is immediate from the above operations and Theorems 1, 2.

**Lemma 4:** Let $PN$ and $PN'$ be the original

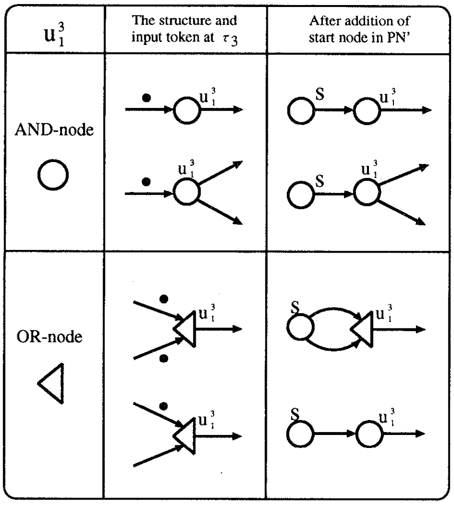



**Fig. 9** The structure and input token of $u_1^3$ at $\tau_3$ and addition of new start node $s$.

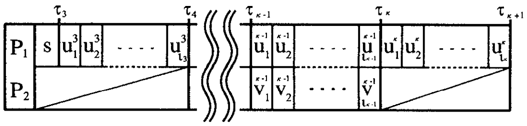



**Fig. 10** The schedule $S'_{L^*}$ generated by $L^*$ for $PN'$.

net whose $S_{L^*}$ has $\kappa \geq 5$ and $Dis(u_1^3) \geq 2$, and the transformed net from $PN$ by the above operations respectively. If we start firing $s$ at $\tau_3 - 1$ for $PN'$, then (i) from time $\tau_3$, schedule $S'_{L^*}$ of $PN'$ as shown in **Fig. 10** is exactly the same as $S_{L^*}$; (ii) if $Dis(u_1^5) \geq 2$ then $u_1^5$ can not be fired before $\tau_5$ for any scheduling; otherwise $S'_{L^*}$ is optimal. □

From Lemma 4, it is obvious that the number of spans of $S'_{L^*}$ is 2 shorter than $S_{L^*}$. Recursively by applying operations (1)–(3) and Lemma 4, we can get a final transformed net $PN^f$ and its schedule $S^f_{L^*}$ whose number of spans is 3 or 5.

**Lemma 5:** $S_{L^*}$ is optimal if the following conditions are satisfied: (i) $Dis(u_1^{2i+1}) \geq 2$ and $u_1^{2i+1}$ can not be fired before $\tau_{2i+1}$; (ii) the part of schedule $S_{L^*}$ from time $\tau_{2i+1}$ is optimal; where, $i \geq 2$. □

The above lemma holds because: (1) the final two nodes of $2i$-th span are $u_{l_{2i}}^{2i} \in IP(u_1^{2i+1})$, $v_{l_{2i}}^{2i} = t$ or $u_{l_{2i}}^{2i}, v_{l_{2i}}^{2i} \in IP(u_1^{2i+1})$, and thus $u_{l_{2i}}^{2i}$ or both $u_{l_{2i}}^{2i}$ and $v_{l_{2i}}^{2i}$ can not be fired before $\tau_{2i+1} - 1$; (2) all the firings of the nodes in the spans from $2i+1$-th are dependent on the firings of $u_{l_{2i}}^{2i}$ or both $u_{l_{2i}}^{2i}$ and $v_{l_{2i}}^{2i}$.

Now we give a theorem showing that $S_{L^*}$ is optimal.

**Theorem 3:** For any given acyclic SWITCH-less $PN$, of which each AND-node possesses single input edge, schedule $S_{L^*}$ is optimal. □

**Proof:** Obviously, this theorem holds individually for $\kappa = 1, 3$ and $Dis(u_1^3) \leq 1$ (in this case $\kappa = 3$ or 5 as can be seen in the proof of Theorem 1). For $\kappa \geq 5$ and $Dis(u_1^3) \geq 2$, recursively applying operations (1)–(3) and Lemma 4, we have

**Case 1:** $Dis(u_1^{\kappa-2}) \geq 2$ and $u_1^5, \cdots, u_1^{\kappa-2}$ can not be fired before $\tau_5, \cdots, \tau_{\kappa-2}$ respectively; or

**Case 2:** $Dis(u_1^{\kappa-2}) \leq 1$ and $u_1^5, \cdots, u_1^{\kappa-4}$ can not be fired before $\tau_5, \cdots, \tau_{\kappa-4}$ respectively.

For Case 1, if $Dis(u_1^{\kappa}) \geq 2$ then $u_1^{\kappa}$ can not be fired before $\tau_\kappa$ from Lemma 4 and Theorem 2; and thus $S_{L^*}$ is optimal according to Lemma 1. Even when $Dis(u_1^{\kappa}) \leq 1$, $S_{L^*}$ is optimal from Lemmas 4, 5. Similarly $S_{L^*}$ is optimal from Lemmas 4, 5 for Case 2. **Q.E.D**

## 5. Concluding Remarks

We have proposed a method of non-preemptive two-processor scheduling for a class of program nets by list scheduling. The characteristics of our method are that the priority list is hybrid, which consists of both dynamic and static parts, and the schedules generated by the hybrid priority list are optimal.

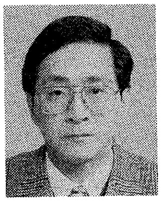
Among multiprocessor scheduling problems, few optimal solutions have been found till now. Compared with the concerned researches by Hu[5] and Coffman-Graham[6], structural complexity of our program nets is between Hu's and Coffman-Graham's; however activity of the nets during execution is not so simple due to that the nodes are executed generally more than once. Therefore as a result of proposing an optimal scheduling method, this paper gives a contribution to multiprocessor scheduling.

Nevertheless we have to point out that our method is not applicable directly to most practical applications because of the assumption that each AND-node doesn't possess two or more input edges. Hence the most important issue for future researchers is to remove this assumption. Besides, future researchers on multiprocessor schedulings of program nets should aim to: (i) To investigate if optimal two-processor scheduling exists for general acyclic SWITCH-less program nets; and (ii) To find efficient heuristic scheduling that allows to use arbitrary processors.

## References


1) Kasahara, H. and Narita, S.: Parallel processing for real-time control and simulation of DCCS, *Proc. 4th IFAC Workshop on Distributed Computer Control Systems*, pp.103–113 (1982).
2) Garey, M.R. and Johson, D.S.: *Computers and Intractability, A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York (1991).
3) Coffman, E.G.: *Computer and Job-Shop Scheduling Theory*, John Wiley, New York (1976).
4) Lenstra, J.K. and Kan, A.H.G.R.: Complexity of scheduling under precedence constraints, *Oper. Res.*, Vol.26, pp.22–35 (1978).
5) Hu, T.C.: Parallel sequencing and assembly line problems, *Oper. Res.*, Vol.9, pp.841–848 (1961).
6) Coffman, E.G. and Graham, R.L.: Optimal scheduling for two-processors systems, *Acta Inf.*, Vol.1, pp.200–213 (1972).
7) Morton, T.E. and Pentico, D.W.: *Heuristic Scheduling Systems*, John Wiley, New York (1993).
8) Rumbaugh, J.: A data flow multiprocessor, *IEEE Trans. Comput.*, Vol.C-26, pp.138–146 (1977).
9) Veen, A.H.: Dataflow machine architecture, *ACM Computing Survey*, Vol.18, pp.365–396 (1986).
10) Yuba, T.: Dataflow Parallel Computers, *Syst., Cont. and Info.*, Vol.33, pp.220–229 (1989).
11) Dennis, J.B.: First version of data flow procedure language, *Lecture Notes in Computer Science*, Vol.19, pp.362–376 (1974).
12) Dennis, J.B.: The MIT Data Flow Engineering Model, *Proc. IFIP Congress 83*, pp.553–560 (1983).
13) Ge, Q.W., Watanabe, T. and Onaga, K.: Topological analysis of firing activities of data-flow program nets, *IEICE Trans. Fundamentals*, Vol.E73, No.7, pp.1215–1224 (1990).
14) Ge, Q.W., Watanabe, T. and Onaga, K.: Execution termination and computation determinacy of data-flow program nets, *J. Franklin Inst.*, Vol.328, No.1, pp.123–141 (1991).
15) Peterson, J.L.: *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, N.J. (1981).
16) Adam, T.L., Chandy, K.M. and Dickson, J.R.: A comparison of list scheduling for parallel processing systems, *Comm. ACM*, Vol.17, No.12, pp.685–690 (1974).
17) Kasahara, H. and Narita, S.: Practical multiprocessor scheduling algorithms for efficient parallel processing, *IEEE Trans. Comput.*, Vol.C-33, No.11, pp.1023–1029 (1984).
18) Ge, Q.W.: On multiprocessor scheduling of acyclic SWITCH-less program nets, *The 11th Workshop on Circuits and Systems in Karuizawa*, pp.499–504 (1998).
19) Onaga, K., Silva, M. and Watanabe, T.: Qualitative analysis of periodic schedules for deterministically timed petri net systems, *IEICE Trans. Fundamentals*, Vol.E76-A, No.4, pp.580–592 (1993).
20) Tanida, T., Watanabe, T., Yamauchi, M. and Onaga, K.: Priority-list scheduling in timed Petri nets, *IEICE Trans. Fundamentals*, Vol.E75-A, No.10, pp.1394–1406 (1992).
21) Ge, Q.W. and Yoshioka, N.: Properties of a two-processor list scheduling for acyclic SWITCH-less program nets, Technical Report of IEICE, Vol.98, No.220, pp.47–54, CST98-17 (1998).


(Received August 7, 1998)
(Accepted February 8, 1999)



**Qi-Wei Ge** received the B.E. from Fudan University, the People's Republic of China, in 1983, M.E. and Ph.D. from Hiroshima University, Japan, in 1987 and 1991, respectively. He was with Fujitsu Ten Limited from 1991 to 1993. Since 1993 he has been an Associate Professor at Yamaguchi University, Japan. His research interest includes Petri net, program net theory and combinatorics. He is a member of the Institute of Information Processing Society of Japan (IPSJ), the Institute of Electronics, Information and Communication Engineers (IEICE) and the Institute of Electrical and Electronics Engineers (IEEE).

**Naomi Yoshioka** received B.E. and M.E. from Faculty of Education, Yamaguchi University, Japan, in 1996 and 1998 respectively. She has been with Fujitsu Ten Limited since 1998. Her research interest includes program net and graph theory. She is a member of the Institute of Electronics, Information and Communication Engineers (IEICE).