論文 84-21-3-3



# Local Microprogram의 竝列 遂行의 最大化
# (Maximal Parallelism in Local Microprogram)

趙 榮 一*, 林 寅 七*
(Young Il Cho and In Chil Lim)

**要 約**

Horizontal microprogram에서 MO' S(microoperations)의 concurrency와 resource의 割當을 考慮하여 MO' S을 竝列로 遂行할 수 있는 알고리즘을 提案한다.

本 알고리즘은 SLM(straight line microprogram) 上에 있는 각 MO에 weight를 附與함으로써 竝列 實行될 수 있는 MO' S을 1개의 MI(microinstruction)으로 結合하여 전체 MI數를 最小化시킴으로써 實行 時間과 microprogramed 디지탈 시스템의 制御記憶裝置 크기(space)를 減少시키는 結果를 얻을 수 있다.

**Abstract**

This paper suggests an algorithm which can perform microoperations (MO'S) in parallel by considering concurrency of MO'S and resource allocation in horizontal microprograms.

The algorithm can be obtained the result which reduces execution time and the space of control memory in microprogrammed digital systems by minimizing the total number of microinctructions by combining MO'S, which can be performed in parallel by assigning a weight to each MO in the SLM (straight line microprogram), into a microinstruction.

## I. 序 論

現在 大規模 計算을 處理하는 데에 超高速 micro-programmed computer 시스템의 必要性이 增加하고 있다. 또 LSI/VLSI등 半導體 技術의 集積度가 向上됨으로서 microprogrammed digital system 設計에 있어서 制御記憶裝置의 幅(size)에 큰 制約을 받지 않게 됨에 따라 制御 시스템을 vertical format 보다는 horizontal format으로 設計하는 趨勢에 있다. 따라서 시스템을 制御하는 microprogram을 좀 더 迅速하게 遂行시키기 위해 microcode compaction에 關한 硏究가 活潑히 行해지고 있다.[1]

그러나, Ramamoorthy와 Tsuchiya의 Critical path (C-Path)[5]와 Wood의 list scheduling 알고리즘[8]은 最適의 microcode를 生成하지 못하고 Yau, Showe의 branch and bound(BAB) 알고리즘[6]은 最適의 microcode를 生成하기는 하나 알고리즘 遂行時間이 入力 MO' S數에 따라 指數函數的으로 增加하므로 많은 時間이 要求된다는 短點을 갖는다. 또한 Dasgupta와 Tartar의 first-come first-served(FCFS) 알고리즘[4]은 MO' S을 ordering하는 境遇에 따라 最適의 microcode를 生成할 수 있는데 最適의 microcode를 찾기 위해서는 入力 MO'S이 n個이면 n!個의 境遇가 發生하므로 알고리즘을 n!番 遂行해야 最適의 microcode를 찾을 수 있게 되어 많은 時間이 要求된다. 그러나 FCFS 알고리즘은 한번 遂行하는데 必要한 時間이 다른 알고리즘(C-path, list scheduling, BAB) 보다 빠르다는 長點을 갖는다.

*正會員, 漢陽大學校 工科大學 電子工學科
(Dept. of Electronics Eng., Hanyang Univ.)
接受日字 : 1984年 1月 12日

本 論文에서는 MO'S의 concurrency와 microprogram resource의 割當을 考慮하여 microprogram 遂行 時間과 制御記憶容量을 減少시킬 수 있는 MO'S의 竝列 遂行 알고리즘을 提案한다. 즉 straight line microprogram(SLM) 上에서 入力 MO'S가 n個일 때 最適의 microcode를 生成하기 위해 FCFS와 같이 n! 번을 反復 遂行하지 않고 각 MO'S에 weight를 주어 再番號를 붙여서 그 順序대로 MO'S을 處理함으로써 단 한번에 항상 最適의 microcode를 生成할 수 있게 한다.

## Ⅱ. Microoperation의 6-Tuple 表現方式 및 相互關係

Local microcode compaction은 SLM 上에서 MO'S의 竝列 遂行과 microprogram resources의 割當을 考慮해 줌으로써 high-level language로 된 horizontal microprograms을 MO'S이 竝列로 遂行될 수 있는 microinstruction(MI)으로 再構成하는 方法이다. 즉, microprogram이 遂行하는데 必要한 時間과 space를 減少시키기 위해 竝列로 遂行할 수 있는 MO'S을 한個의 MI로 結合하는 過程이다.

SLM이란 MO의 sequence로 構成되어 있으며 다음과 같이 表示했을 때

$S=\langle MO_1, MO_2, \cdots\cdots, MO_t\rangle$

S는 1個의 始點 $MO_1$과 1個의 終點 $MO_t$을 가지며 그 사이에 어떤 branch MO을 갖지 않으며 內部 loop가 microcode compaction 알고리즘에서 可能하면 竝行하여 遂行될 수 있는 MO을 檢出하고 conflict 關係를 調査하기 위해 MO을 遂行하는데 要求되는 resource를 6-tuple 表現法으로 나타내기로 한다.

MO=〈name, I, O, U, T, F〉

1) 이름(name) : 遂行되는 MO에 대한 名稱, 어떠한 動作을 하느냐에 따라 이름이 달라진다.

2) 入力(I) : MO의 入力으로서 要求되는 모든 resources의 集合.

3) 出力(O) : MO의 出力으로서 要求되는 모든 resources의 集合.

4) 機能(U) : MO이 遂行되는 동안에 要求되는 모든 機能素子(functional unit)의 集合.

5) 時間(T) : MO 遂行에 必要한 clock phases의 集合.

6) 필드(F) : MO이 遂行되는 동안에 要求되는 制御部의 性質을 이 部分에 明示한다.

두 MO'S tuple

MOi=〈name i, Ii, Oi, Ui, Ti, Fi〉

MOj=〈name j, Ii, Oj, Uj, Tj, Fj〉

에서 i〈j는 SLM에서 MOi가 MOj보다 먼저 나타난다는 것을 意味하며 MOi〈ip〉 MOj로 表示한다. 그러나 MOi가 MOj 보다 먼저 遂行된다는 것을 意味하지는 않는다.

Poly phase MI에 대하여 MOi가 遂行되는 clock phase가 MOj가 遂行되는 clock phase 보다 앞선다면 MOi는 MOj보다 時間上으로 先行(precede)한다고 말하며 MOi〈tp〉 MOj로 表示한다.

두個의 MO'S間의 關係를 다음과 같이 定義한다.

〈定義1〉 MOi〈ip〉 MOj일 때 다음 條件 1),2),3) 中 어느 하나를 滿足한다면 MOi와 MOj사이에 data interaction이 있다고 말한다.

〈條件〉 1) $Oi \cap Ij \neq \phi$

2) $Ii \cap Oj \neq \phi$

3) $Oi \cap Oj \neq \phi$

이 境遇에 MOi와 MOj의 實行 順序를 變更시킬 수 없으며 compaction 알고리즘에 의해 生成되는 microcode는 SLM에서의 順序대로 나타나야 한다. 즉 서로 다른 MI에 存在하게 된다.

〈定義2〉 MOi〈ip〉 MOj이고 MOi와 MOj사이에 data interaction이 成立하고 MOi와 MOj사이에 어떤 MO도 존재하지 않는다면 MOj는 MOi에 직접 데이타종속관계(directly data dependent : ddd)가 있다고 정의하고 MOi〈ddd〉 MOj로 表示한다.

〈條件1〉 MOi〈ddd〉 MOj라면 $MOi\langle ddd\rangle MO_{k_1}$, $MO_{k_1}\langle ddd\rangle MO_{k_2}$, ……, $MO_{k(n-1)}\langle ddd\rangle MO_{kn}$, $MO_{kn}\langle ddd\rangle$ MOj (n≧1)를 滿足하는 어떤 $MO_{k_1}$, $MO_{k_2}$, ……, $MO_{kn}$의 sequence도 存在하지 않는다.

〈定義3〉 條件1에서 $MO_{k_i}$ (i=1, 2, ……, n)가 하나 以上 存在한다면 MOi와 MOj사이에는 데이타從屬關係(data dependent : dd)가 있다고 말하고 MOi〈dd〉 MOj로 表示한다.

〈定義4〉 두 MOi와 MOj가 데이타從屬關係가 없으면 데이타獨立(data independent : di)이라 말하고 MOi〈di〉 MOj로 表示한다.

〈定義5〉 두 MOi와 MOj가 데이타獨立關係가 있거나, MOi가 MOj 보다 時間上으로 先行한다면 (MOi〈tp〉 MOj) 두 MO'S은 데이타兩立(data compatible : dc)하다고 말하며 MOi〈dc〉 MOj로 表示한다.

MOj가 MOi에 데이타從屬關係가 있는 境遇라도 MOi의 遂行이 끝나는 瞬間이 MOj가 遂行을 始作하는 瞬間보다 앞서면 두 MO'S은 같은 MI에서 實行될 수 있다.

〈定義6〉 두 個의 MO'S MOi와 MOj에서 MOi

〈tp〉 MOj 하면서 MOi〈ddd〉 MOj하다면 두·MO'S은 弱從屬關係(weakly dependent : wd)가 있다고 말하고 MOi〈wd〉 MOj로 表示한다.

弱從屬關係가 있는 두 MO'S은 같은 MI에 놓여질 수 있다.

〈定義7〉 弱從屬關係가 아니면서 MOi〈ddd〉 MOj인 MO'S을 强從屬關係(strongly dependent : sd)가 있다고 말하고 MOi〈sd〉 MOj로 表示한다.

Field는 현재 MO이 수행되는 동안 요구되는 제어부의 성질을 나타내는 부분으로서 ALU control, BUS control, immediate data 등이 있으며 1개의 MO은 1개 이상의 field을 가질 수 있다. 예를들어 jump MO은 jump control과 branch할 address인 immediate data를 가질 수 있다. 따라서 동일한 MI에 있는 MOi, MOj가 동일한 field을 점유할 때 MOi와 MOj는 field conflict라 한다.

〈定義8〉 어떤 field conflict도 없는 두 MO'S MOi, MOj를 field compatible하다고 말하고 MOi〈fc〉 MOj로 表示한다.

두 MO'S이 同時에 같은 functional unit를 共有한다면 unit conflict하다고 말한다.

〈定義9〉 Unit conflict하지 않는 두 MO'S MOi, MOj를 unit compatible하다고 말하고 MOi〈uc〉 MOj로 表示한다.

〈定義10〉 두 MO'S MOi, MOj가 MOi〈dc〉 MOj이 $MO_i<UC>MO_j$이고 $MO_i<fc>MO_j$이면 이 MO'S을 compitable하다고 말하고 MOi〈cp〉MOj로 表示한다.

〈定義11〉 Data dependent graph(DDG)에 있는 어떤 MO의 weight는 그 MO의 descendents의 數로 정해진다.

## Ⅲ. Weight를 考慮한 Local Compaction 알고리즘

MO'S間의 데이타從屬與否을 알기 위해 定義 1~7을 바탕으로 制御흐름 狀況을 나타내는 데이타從屬圖를 構成하는 節次는 다음과 같다.

〈節次1〉 現在 그래프에 追加하려는 MO이 그래프上에서 特定 MO과 直接 데이타 從屬關係가 있다면 두 MO'S사이에 線을 連結시키고 節次4로 간다.

〈節次2〉 特定 MO에 대한 parent가 있다면 節次1로 간다.

〈節次3〉 節次1과 節次2에서 調査한 path 오른쪽에 나타나는 leaf가 있다면 節次1로 간다.

〈節次4〉 그래프에 追加할 MO이 SLM에 남아 있으면 節次1로 간다.

求해진 데이타從屬圖와 定義 8~10에 의한 conflict 關係, 그리고 定義11과 같이 weight을 設定하여 weight가 큰 順序대로 MO을 適用시켜 順序(ordering)가 달라지는 어떠한 境遇라도 항상 最適의 microcode를 生成하도록 한다. 즉, 데이타從屬圖에 있는 MO'S에 대해서 weight가 가장 큰 것부터 順序대로 데이타從屬關係와 conflict 與否에 의해 現在의 MI에 附加된다. 現在 考慮中인 MO이 가장 最近의 MI에 있는 어떤 MO'S과도 데이타 從屬關係가 없으면 그 MI 바로 前의 MI으로 올라가서 調査를 계속한다. 만약 現在의 MO이 i번째 MI에 있는 어떤 MO'S과 데이타從屬關係가 있다면 現在의 MO은 i보다 先行하는 어떤 MI에도 놓여질 수 없다. 즉 그러한 MI'S에 있는 MO'S이 實行을 끝낼 때까지는 現在의 MO이 實行될 수 없음을 意味한다. 이러한 調査의 目的은 새로 附加될 MO이 데이타從屬關係를 破壞시키지 않고 놓여질 수 있는 가장 빠른 MI을 求하는 것이다. 現在의 MO이 어떤 MI에 附加되기 위한 또 하나의 條件은 그 MI에 있는 어떤 MO과도 conflic 關係가 없어야 한다. 그러한 MI이 發見되지 않으면 새로운 MI에 그 MO이 附加된다.

本 알고리즘의 節次는 다음과 같다.

〈節次1〉 DDG에 있는 각 MO에 대한 weight을 定義11에 따라 求한다.

〈節次2〉 DDG에서 weight가 가장 큰 MO을 1로 하여 weight가 큰 順序대로 MO에 再番號를 붙인다. 但, weight가 같은 境遇에는 ddd나 dd關係가 많은 MO에 빠른 番號를 붙이되 그것도 같은 境遇에는 原來의 順序가 빠른 MO을 擇한다.

〈節次3〉 Conflict 關係가 있는 MO을 再番號가 붙여진 MO으로 바꾼다.

〈節次4〉 새로 附加될 MO이 調査되는 i번째 MI에 있는 MO'S과 데이타從屬關係가 있다면 다음 過程을 遂行한다(i번째 MI은 현재 조사하려는 MI, 즉 현재까지 가장 나중에 만들어진 MI를 의미함).

〈過程1〉 새 MO이 i번째 MI에 있는 어떤 MO과도 conflict 關係가 없으면 過程3으로 간다.

〈過程2〉 i를 하나 增加시키고 i번째 MI에 MO이 存在하면 過程1을 다시 遂行하고 그렇지 않으면 節次9로 간다.

〈過程3〉 새 MO이 데이타從屬關係가 있는 MO과 弱從屬關係가 있다면 節次9로 가고 그렇지 않으면 過程2를 다시 遂行한다.

〈節次5〉 i를 하나 減少시키고 i가 1보다 크거나 같으면 節次4로 간다.

〈節次6〉 i를 하나 增加시키고 i번째 MI에 MO이 存在하면 節次8로 간다.

〈節次7〉 存在하는 모든 MI'S의 番號를 하나씩 增加시키고 i를 1로 놓고 節次9로 간다.

〈節次8〉 새 MO이 i번째 MI에 있는 MO'S과 conflict關係가 있다면 節次6으로 간다.

〈節次9〉 i번째 MI에 새 MO을 附加하고 SLM 上에 남아 있는 MO이 있으면 i를 가장 큰 MI의 番號로 놓고 節次4로 간다.

위의 節次에 대한 flow chart는 그림1에 나타나 있다.

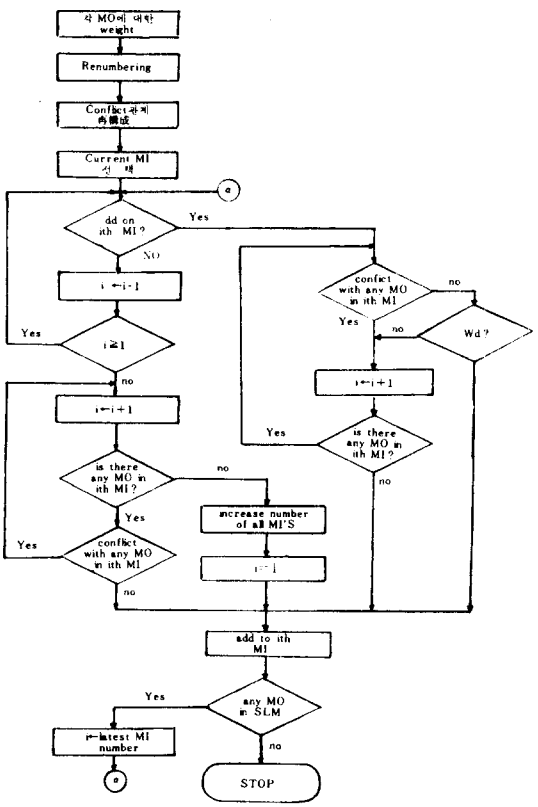



그림 1. 알고리즘의 狀態흐름圖
Fig. 1. A flow chart of the algorithm.

## Ⅳ. 適用例 및 從來의 알고리즘과의 比較檢討

그림2와 같은 모델에 FCFS 알고리즘을 適用시킨 結果를 그림3에 나타내었다.

그림 2에서 wid는 약 종속관계를 의미한다.

그림2에 本 알고리즘을 適用시켰을 때 weight는 그림4(a)와 같고 變形된 DDG와 MO'S間의 conflict는 각각 그림4(b), (c)와 같고, 그 結果는 그림5에 나타내었다.

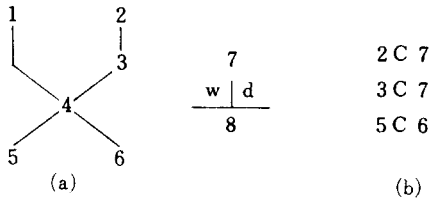



그림 2. (a) 데이타 從屬圖
(b) MO'S간의 conflict
Fig. 2. (a) DDG.
(b) Conflict between MO'S.

| MI \ 反復回數 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1,2 | 1,2 | 1,2 | 1,2 | 1,2 | 1,2 | 1,2 |
| 2 | | | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | | | | 4 | 4 | 4 | 4,7 | 4,7,8 |
| 4 | | | | | 5 | 5 | 5 | 5 |
| 5 | | | | | | 6 | 6 | 6 |

그림 3. 그림1에 대한 FCFS 알고리즘에 의한 compaction 結果
Fig. 3. Compaction result of FCFS algorithm for Fig. 1.

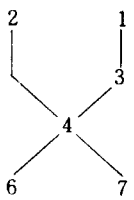
그림 2의 모델에 대하여 FCFS 알고리즘을 適用시켰을 때와 本 알고리즘의 遂行 結果는 同一함을 알 수 있다. 즉 그림2에 대한 microprogram은 5個의 MI으로 이루어진다. 그림2의 모델에서 가장 긴 經路는 2-3-4-5에 該當하므로 直接 데이타關係만을 考慮할 때 서로 다른 MI에서 MO'S을 區分해 줄 수있는 最小의 MI은 理論的으로 4個이다. 實際로는 MO5와 MO6가 conflict하기 때문에 5個의 MI이 必要하며 最適의 값이 된다. 그러나 FCFS 알고리즘의 境遇, 항상 最適의 microcode를 生成하는 것은 아니다. 그림2의 MO'S에 대한 順序를 달리하여 그림6에 나타내었다. 이 모델에 FCFS 알고리즘을 適用시킨 結果는 그림7과 같으며 MI이 하나 더 增加하였다. 그림6을 本 알고리즘에 適用시켰을 때 weight는 그림8과 같으나 變形된 DDG와 MO'S間의 conflict는 結局 그림4(b), (c)와 同一하게 된다. 또한 그림6에 대한 本 알고리즘 遂行 結果 역시 그림5와 同一하게 된다.

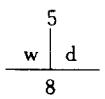
以上의 結果로부터 FCFS 알고리즘은 SLM上에 있는 MO'S의 順序를 달리할 때마다 結果가 다르게 나올 수 있으므로 항상 最適 값을 保障할 수 없었으나

本 論文의 알고리즘에서는 weight에 따라 順序가 定해지므로 SLM上에서의 MO'S이 어떤 順序를 가진다고 하더라도 항상 最適의 microcode를 保障할 수 있다.

| MO | weight | 順序 |
|---|---|---|
| 1 | 3 | 2 |
| 2 | 4 | 1 |
| 3 | 3 | 3 |
| 4 | 2 | 4 |
| 5 | 0 | 6 |
| 6 | 0 | 7 |
| 7 | 1 | 5 |
| 8 | 0 | 8 |

(a)




(b)

1 C 5
3 C 5
6 C 7

(c)

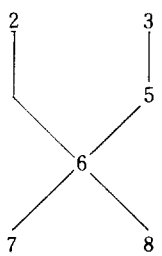
그림 4. (a) 각 MO'S의 weight와 再番號 (b) DDG (c) MO'S간의 conflict

Fig. 4. (a) Weight and renumber of each MO. (b) DDG. (c) Conflict between MO'S.

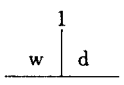
| MI \ 反復回數 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1,2 | 1,2 | 1,2 | 1,2 | 1,2 | 1,2 | 1,2 |
| 2 | | | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | | | | 4 | 4,5 | 4,5 | 4,5 | 4,5,8 |
| 4 | | | | | | 6 | 6 | 6 |
| 5 | | | | | | | 7 | 7 |

그림 5. 그림2에 대한 本 알고리즘의 compaction 結果

Fig. 5. Compaction result of our algorithm for Fig. 2.




(a)




1 C 3
1 C 5
7 C 8

(b)

그림 6. (a) DDG
(b) MO'S간의 conflict

Fig. 6. (a) DDG.
(b) Conflict between MO'S.

| MI \ 反復回數 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2 | | | 1,2 | 1,2,4 | 1,2,4 | 1,2,4 | 1,2,4 | 1,2,4 |
| 3 | | | | | 5 | 5 | 5 | 5 |
| 4 | | | | | | 6 | 6 | 6 |
| 5 | | | | | | | 7 | 7 |
| 6 | | | | | | | | 8 |

그림 7. 그림6에 대한 FCFS 알고리즘의 compaction 結果

Fig. 7. Compaction result of FCFS algorithm for Fig. 6.

| MO | weight | 順序 |
|---|---|---|
| 1 | 1 | 5 |
| 2 | 3 | 2 |
| 3 | 4 | 1 |
| 4 | 0 | 8 |
| 5 | 3 | 3 |
| 6 | 2 | 4 |
| 7 | 0 | 6 |
| 8 | 0 | 7 |

그림 8. 그림6에 대한 각 MO'S의 weight와 再番號

Fig 8. Weight and renumber of each MO for Fig. 6.

## V. 結 論

本 論文에서는 local microprogram에서 각 MO'S에 대한 데이타 從屬關係를 考慮한 데이타從屬圖上에서 각 MO'S에 weight를 주어 높은 weight를 갖는 MO부터 차례로 MI에 適用시킴으로써 항상 最適의 microcode를 生成할 수 있는 알고리즘을 提案하였다. 즉, SLM上에서 MO'S이 어떤 順序를 갖더라도 weight에 따라 順序가 定해지므로 항상 最適의 microcode를 求할 수 있었고 또한 最適의 microcode를 生成함으로써 MI의 數를 減少시켜서 microprogram의 遂行 時間을 短縮시킬 수 있고 制御記憶裝置의 크기(space)를 減少시킬 수 있다.

## 參 考 文 獻

[1] R.J. Sheraga and J.L. Gieser, "Experiments in automatic microcode generation," *IEEE Trans. Comput.*, vol. C-32, pp. 557-569, June 1983.

[2] T. Agerwala, "Microprogram optimization: a survey," *IEEE Trans. Comput.*, vol. C-25, pp.962-973, Oct. 1976.

[3] S. Davidson et al., "Some experiments in local microcode compaction for horizontal machines," *IEEE Trans. Comput.*, vol. C-30, pp.460-477, July 1981.

[4] S. Dasgupta and J. Tartar, "The identification of maximal parallelism in straight line microprograms," *IEEE Trans. Comput.*, vol. C-25, pp.986-991, Oct. 1976.

[5] C.V. Ramamoorthy and M. Tsuchiya, "A high level language for horizontal microprogramming," *IEEE Trans. Comput.*, vol. C-23, pp.791-802, Aug. 1974.

[6] D. Landskov, et al., "Local microcode compaction techniques," *ACM Comput. Surveys*, vol.12, pp.261-294. Sep. 1980.

[7] G. Wood, "On the packing of micro-operations into microinstruction words," *in Proc. 11th Annu.* Microprogramming Workshop, pp.51-56.

[8] T.L. Adam, K.M. Chandy, and J.R. Dickson, "A comparison of list schedules for parallel processing systems," *Commun. Ass. Comput. Mach.*, vol. 17, pp. 685-690, Dec. 1974.

[9] S. Dasgupta, "Parallelism in loop-free microprograms," *in Proc. IFIP Congress*, pp.745-750, 1977.

[10] R.L. Kleir and C.V. Ramamoorthy, "Optimization strategies for microprograms," *IEEE Trans. Comput.*, vol. C-20, pp.783-794, July 1971.

[11] Sadahiro Isoda and Yoshizumi Kobayash, "Global compaction of horizontal microprograms based on the generalized data dependency graph," *IEEE Trans. Comput.*, vol. C-32, pp.922-933, Oct. 1983.