C-06

# 分割スケジューリングによるクラスタ型 VLIW DSP のコード生成

## Partitioned Scheduling for Clustered VLIW DSPs

益井　勇気 †　　石浦 菜岐佐 †
Yuuki Masui　Nagisa Ishiura

### 1　はじめに

DSP (Digital Signal Processor) はディジタル信号処理を高速に低消費電力で実行することを目的に最適化されたプロセッサであり, 特に VLIW (Very Long Instruction Word) 型 DSP は静的スケジューリングに基づく並列演算により優れた電力性能比を達成する. しかしその性能を引き出すためには, 効率のよい命令スケジューリングを行うことが必須となる.

[1] ではクラスタ型 VLIW DSP の TMS320C62x (以下 C62x) に対して Simulated Annealing を利用した高速近似解法を提案しているが, ユニットの非対称性やレジスタ制約を考慮していないという問題がある. [2] では C62x のデータパスの詳細やレジスタ制約まで考慮したコードスケジューリング問題を定式化し, PBSAT (pseudo Boolean satisfiability) [3] により厳密解を求めているが, 計算量の増大により大規模なプログラムを扱えないという問題がある.

そこで本研究では, [2] に基づいて一定サイクルずつスケジューリングを行うことにより計算時間の増加を抑制する, 分割スケジューリング手法を提案する. この手法を C62x に適用した結果, [2] では解けなかった規模の問題を解くことが出来た.

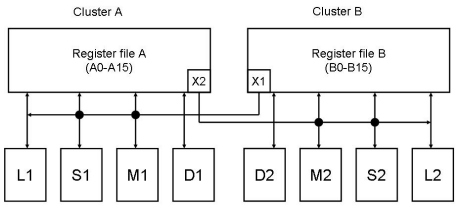


図 1: TMS320C62x のデータパス

### 2　TMS320C62x

図 1 に C62x のデータパスを示す. C62x は 2 つのクラスタ (A, B) を持ち, 各クラスタはレジスタファイル (RF) と 4 つのユニット (L, S, M, D) から成る. 各ユニットのオペランドには基本的に同じクラスタの RF を指定するが, 「クロスパス」を使用すれば反対側のクラスタの RF をオペランドとして指定出来る. クロスパスには RF B から RF A に転送する X1, RF A から RF B に転送する X2 の 2 つがあり, それぞれ 1 サイクルに 1 回ずつ使用出来る.

†関西学院大学

### 3　分割スケジューリングとコード生成問題

#### 3.1　最適コードスケジューリング [2]

本稿で扱うコード生成問題は, 「依存グラフ」と, 「命令パターン集合」に対し, 実行サイクル数が最小となるように各演算の実行開始サイクルと資源割り当てを決定する問題である. ここで「依存グラフ」は分岐演算を含まない基本ブロック (以下 BB) を表し, 「命令パターン」はプロセッサで実行可能な命令の情報を表す. 必要に応じて, 転送演算を挿入することもある. BB 中の演算と値を表わす節点の集合をそれぞれ $F$, $V$, データ依存枝の集合を $E$ とする. 依存グラフ $G = (V \cup F, E)$ は非巡回の有向 2 部グラフ ($E \subseteq V \times F \cup F \times V$) である. 各枝 $e \in E$ にはオペランド番号 $o(e)$ と整数値の遅延 $d(e)$ が定義されている. データを格納する記憶要素の集合を $M$ とし, $m \in M$ の容量を $C(m)$ とする. C62x では $M = \{A, B, MM\}$ (それぞれ RF A, RF B, メモリを表す) である. 各節点には $i(v)$, $o(v)$ が定義されており, $m \in i(v)$ は初期に節点 $v$ の値が記憶要素 $m$ に存在することを示し, $m \in o(v)$ はスケジューリング終了時節点 $v$ が記憶要素 $m$ に存在することを示す.

命令パターン集合 $P$ の要素 $p$ には, $p$ が使用するユニットの集合 $U(p)$, パスの集合 $X(p)$, 遅延 $d(p)$ が定義されている. 演算 $f \in F$ の結果を記憶要素 $m$ に格納する命令パターンの集合を $P_{f,m}$, 命令パターン $p$ において $i$ 番目のオペランドの値を格納する記憶要素を $m(p, i)$ とする. $S$ を転送演算の集合, $d(s)$ を転送演算 $s$ の遅延とする. $P_{s,n,m}$ を転送演算 $s$ で記憶要素 $n$ から記憶要素 $m$ へデータを転送する命令パターンの集合とする. 変数 $\alpha_{t,v,m}$ は, サイクル $t$ に節点 $v$ が記憶要素 $m$ に存在すれば 1, そうでなければ 0 をとる. $\xi_{t,f,p}$ はサイクル $t$ に節点 $f$ の演算を命令パターン $p$ で実行すれば 1, そうでなければ 0 をとる. $\tau_{t,v,p}$ はサイクル $t$ に節点 $v$ を命令パターン $p$ で転送すれば 1, そうでなければ 0 をとる. $t_{max}$ は実行サイクル数の上限, $T = \{1, 2, \cdots, t_{max}\}$ は実行サイクルの集合を表す.

最適スケジューリングの制約条件を以下に示す.

(1) 入力制約:

$$\begin{cases} \forall v \in V: & \alpha_{0,v,m} = 1 \quad \text{iff} \quad m \in i(v). \\ \forall v \in V: & \alpha_{0,v,m} = 0 \quad \text{iff} \quad m \notin i(v). \end{cases}$$

(2) 依存制約:

1. $\forall t \in T, \forall (f, v) \in E, \forall s \in S, \forall n \in M, \forall m \in M:$

$$\alpha_{t,v,m} \rightarrow \alpha_{t-1,v,m} \vee \Big( \bigwedge_{(f,v) \in E} \Big( \bigvee_{p \in P_{f,m}} \xi_{t-d(f,v),f,p} \Big) \Big)$$

$$\vee(\bigvee_{p\in P_{s,n,m}} \tau_{t-d(s),v,p}).$$

2. $\forall t \in T, \forall f \in F, \forall p \in P_{f,m}$ :

$$\xi_{t,f,p} \rightarrow \bigwedge_{(v,f)\in E} \alpha_{t-d(v,f),v,m(p,o(e))}.$$

3. $\forall t \in T, \forall v \in V, \forall s \in S, \forall m,n \in M, \forall p \in P_{s,n,m}$ :

$$\tau_{t,v,p} \rightarrow \alpha_{t,v,n}.$$

(3) 資源制約: サイクル $t$ でユニットまたはパス $r$ を使用する変数の集合を $B_r^t = \{\xi_{t,f,p}, \tau_{t,v,p} \mid t \in T, f \in F, U(p) = r \vee X(p) = r\}$ とする.

$$\forall t \in T, \forall r \in U(p) \cup X(p) : \quad \sum_{\beta \in B_r^t} \beta \leq 1.$$

(4) 記憶容量制約:

$$\forall t \in T, \forall m \in M : \quad \sum_{v\in V} \alpha_{t,v,m} \leq C(m).$$

(5) 代入制約:

$$\forall f \in F : \quad \sum_{t\in T, p\in P} \xi_{t,f,p} \leq 1.$$

(6) 目的関数:

$$\text{maximize} \sum_{t\in T}(\bigwedge_{v\in V_O}(\bigvee_{m\in M} \alpha_{t,v,m}))$$

### 3.2 分割スケジューリング

前節の最適スケジューリング問題は NP 完全であるため, 依存グラフの規模が大きいと現実時間で解けないという問題がある. そこで本稿では, コード全体を一度にスケジューリングするのではなく, コードを先頭から計算が可能なサイクル数分づつスケジューリングしていく, 分割スケジューリング手法を提案する. まず 1 から $t_{max}$ サイクル目までのスケジューリングを行う. これで全体のスケジューリングが完了しないときは, スケジューリングする演算数の最大化を目指す. 以後 $t_{max}+1$ から $2t_{max}$ サイクル, $2t_{max}+1$ から $3t_{max}$ サイクル, $\cdots$ と, $t_{max}$ サイクルずつのスケジューリングを全体のスケジューリングが完了するまで繰り返す. 各スケジューリングにおける制約式は最適コードスケジューリングの定式化と同じである. 異なる部分を以下に示す.

(1') 入力制約: 各節点には $\hat{i}(v) = \{(t,m) \mid 1 \leq t \leq t_{max}, m \in M\}$ が定義されており, $(t,m) \in \hat{i}(v)$ は節点 $v$ の値がサイクル $t$ に記憶要素 $m$ に存在することを意味する. $\hat{i}(v)$ は 1 回目のスケジューリングでは $\hat{i}(v) = \{(0,m) \mid m \in i(v)\}$ である. 2 回目以降のスケジューリングでは 1 回前のスケジューリングの変数 $\alpha$ を元に決まり $\hat{i}(v) = \{(t,m) \mid \alpha_{t_{max}+t,v,m} = 1\}$ である. 各スケジューリングの入力となる $i(v)$ によって各値の存在変数 $\alpha$ は以下の制約を満たさなければならない.

$$\begin{cases} \forall v \in V : & \alpha_{t,v,m} = 1 \quad \text{iff} \quad (t,m) \in i(v). \\ \forall v \in V : & \alpha_{0,v,m} = 0 \quad \text{iff} \quad (0,m) \notin i(v). \end{cases}$$

(6') 目的関数:

$$\text{maximize } \beta \sum_{t\in T}(\bigwedge_{v\in V_O}(\bigvee_{m\in M} \alpha_{t,v,m})) \quad + \sum_{t\in T, f\in F, p\in P} \xi_{t,f,p}.$$

ここで $\beta$ は十分大きい数とする. 第 1 項により $t_{max}$ サイクルで実行可能の実行サイクル数を最小化する. 第 2 項により $t_{max}$ サイクルで実行不可能のときに演算の実行数を最大化する.

## 4 実験結果

前節の定式化に基づき, TMS320C62x のコードスケジューリングプログラムを実装した. ソルバーには PBSAT (PBS Ver2.1 for Win) [3] を用いた. 3 つの依存グラフ BB1 〜 BB3 に対して, [2] と提案手法の比較を行った実験の結果を表 1 に示す. 演算数は各依存グラフの演算節点の数, サイクル数は生成されたコードの実行に要するサイクル数, CPU は解を得るのに要した時間 (Cygwin ver.2.05b.0, Intel Celeron 1.83GHz, メモリ 1GB) である。「>1000」は 1000 秒で解が得られなかったことを表わす. 分割スケジューリングでは $t_{max} = 6$ としてスケジューリングを行った. 最適スケジューリングでは 1000 秒で解を得られなかった問題を分割スケジューリングでは 10 数秒で解くことが出来た. また BB1 と BB2 では最適スケジューリングと同じサイクル数の解を求めることが出来ている.

表 1: TMS320C62x のコードスケジューリング

| | 演算数 | 最適解 [2] | | 提案手法 | |
|---|---|---|---|---|---|
| | | サイクル数 | CPU[s] | サイクル数 | CPU[s] |
| BB1 | 10 | 13 | 1.4 | 13 | 2.6 |
| BB2 | 70 | 24 | 7.2 | 24 | 5.8 |
| BB3 | 76 | NA | >1000 | 68 | 12.9 |

## 5 むすび

本稿では VLIW 型 DSP のコード生成のための分割スケジューリング手法を提案した. 詳細な評価実験を行うこと, およびその結果に基づいて効率の改善を試みることが今後の課題である.

## 参考文献


[1] R. Leupers: "Instruction Scheduling for Clustered VLIW DSPs," in *Proc. 2000 International Conference on Parallel Architectures and Compilation Techniques,* pp. 291–300 (Oct. 2000).

[2] 小林, 石浦, 益井: "準ブール充足可能性判定によるクラスタ型 VLIWDSP の最適コードスケジューリング," 情処関西支部大会, C–4 (Oct. 2006).

[3] F. A. Aloul, A. Ramani, l. L. Markov, and K. A. Sakallah: "Generic ILP versus Specialized 0-1 ILP: An Update," in *Proc. International Conference on Computer-Aided Design*, pp. 450–457 (Nov. 2002).