修 士 論 文 概 要 書

2010 年 2 月提出

専攻名（専門分野）	情報理工学専攻	氏 名	橋本 識弘	指 導 教 員	戸川 望 印
研究指導名	情報システム設計研究	学籍番号	5108B100 CD - 4		
研 究 題 目	部分マッチングを考慮し MISO(Multiple Input, Single Output) 構造に対応したプロセッシングユニット最適化手法に関する研究				

1 はじめに

近年，アプリケーションに特化した専用プロセッサの需要が伸びており，携帯端末や車載システムなどにおいて利用されている．専用プロセッサは汎用プロセッサと比較して小面積であり，特定のアプリケーションに対して高性能かつ低消費電力であるという利点がある．

我々が構築中であるプロセッサ合成システム SPADES(System for Processor Architecture Design with Estimation - type SIMD) は，アプリケーションと実行時間制約を与えることで，プロセッサを自動合成できる．また，専用演算器であるプロセッシングユニット (以降 PU と呼ぶ) の付加を考慮している．

プロセッサに付加される専用演算器の生成に際し，満たすべき要件が大きく 2 つ挙げられる．1 つはアプリケーションに柔軟に対応するために，専用演算器が MISO(Multiple Input, Single Output) 構造を持っていることである．MISO 構造の専用演算器の例を図 1(a) に示す．もう 1 つは専用演算器がより多く使用されるために，専用演算器の構成とアプリケーションの CDFG のサブグラフが部分的に一致している場合 (以降，部分マッチングと呼ぶ) にも専用演算器で実行させることである．例えば図 1 において，(a) に示した専用演算器と (b) に示した DFG のマッチングを行う場合，点線で囲った演算ノードが (c) のように専用演算器で実行される．専用演算器内の不要な演算に対して演算をさせないことで，必要な出力結果を得ることができる．

しかし，これまでに専用演算器が満たすべき 2 つの要件の両方を満たした手法が提案されていなかった．

そこで，本論文では，部分マッチングを考慮した MISO 構造を持つ専用演算器の合成手法を提案する．SPADES を用いて，提案手法により PU を生成した．

2 提案手法

PU の最適化問題について，以下のように定義する．

PU 最適化問題 アプリケーションの CDFG，プロセッサの構成情報，実行時間制約 T_c が与えられたとき，アプリケーションの実行時間 T_{app} が $T_{app} \leq T_c$ を満たす PU の構成を出力する．PU の面積 A_{pu} の最小化を目的とする．

提案手法は，大きく**生成系**と**再構成系**に分かれる．生成系において，PU の面積を考慮せずに T_{app} が最小となる PU の構成を得る．再構成系にて PU の粒度を小さくしていき，T_c に違反したら，違反する前の PU の構成を解として出力する．生成系は，複数の演算から PU を 1 個生成する**クラスタリング**，アプリケーションに対して新しく生成された PU を割付ける**バインディング**，生成された PU の内部構造に対してパイプラインレジスタを割付ける **PU 内パイプライン化**および CDFG に対してステップ数の計算およびレジスタを割付ける**スケジューリング**により成り立つ．再構成系は，現在生成されている PU を削除し，より粒度の小さい PU を生成する**リクラスタリング**，削除された PU が割付けられている CDFG 内の演算ノードに対してバインディングを解除し，新たに生成された PU を割付ける**リバインディング**，**PU 内パイプライン化**および**スケジューリング**により成り立つ．

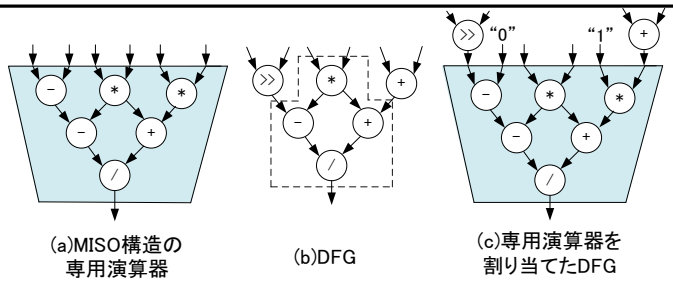

図 1: 部分マッチング.

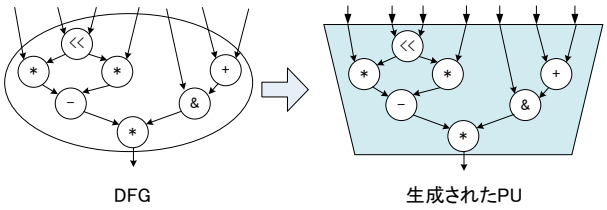

図 2: クラスタリングの様子.

2.1 生成系

クラスタリング

クラスタリングでは，アプリケーションの CDFG から，複数の演算ノードをまとめて実行する PU を生成する．PU 内部のグラフ構造は MISO 構造をとり，根のノードからのみ演算結果を出力する．クラスタリングの様子を図 2 に示す．

バインディング

バインディングでは，MISO 構造の PU とアプリケーションの CDFG を入力として，PU が割付けられた CDFG を出力する．まず，バインディングのマッチングタイプを定義する．マッチングタイプは，PU と CDFG のサブグラフの構成が同じである完全マッチングと，構成が部分的に一致している部分マッチングに分けられる．図 2 で生成された PU と完全マッチングしている DFG のサブグラフを図 3 に示す．また，部分マッチングしている DFG のサブグラフを図 4 に示す．図 4 では，乗算ノード以外は PU と DFG のサブグラフの構成が一致しているので，加算ノードの片方に定数 0 を入力することで，PU で実行できる．

バインディングの様子を図 5 に示す．斜線で囲まれたノード集合が，PU で実行される．

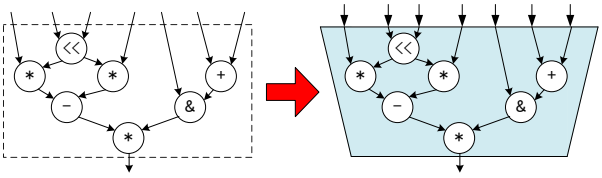

図 3: 完全マッチングの例.

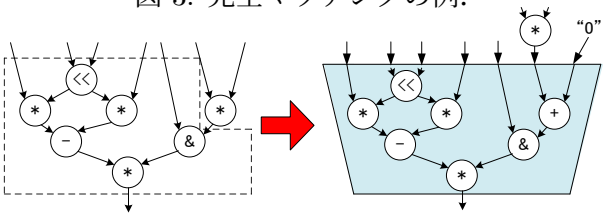

図 4: 部分マッチングの例.

PU 内パイプライン化

PU 内部にパイプラインレジスタを挿入し，PU のクリティカルパスを分割する．各演算を最小モジュールに分割することで，演算ノード単位ではなく最小モジュール単位でパイプライン化を行い，PU のクリティカルパスを分割する．パイプライン化された PU を図 6 に示す.

スケジューリング

アプリケーションの CDFG に対しリストスケジューリングを行い，PU を CDFG に割付けた場合のアプリケーションの実行にかかるコントロールステップ数を得る．また，汎用レジスタを割付ける.

2.2 再構成系

再構成系の構成要素の内，PU 内パイプライン化とスケジューリングは生成系と同じである.

リクラスタリング

リクラスタリングでは，現在生成されている PU を削除する．そして，削除された PU のノード梱包数から 1 引いた値を次に生成する PU のノード梱包数の上限とし，クラスタリング条件に追加する．その後，クラスタリングを行う.

リバインディング

リバインディングでは，アプリケーションの CDFG の中で削除された PU が割付けられているノードに対して，バインディングを解除し元の CDFG に戻す．その後，新たに生成された PU のバインディングを行う.

3 実験

入力アプリケーションとしてアルファブレンドを与えた場合において，MISO 構造で部分マッチングを考慮した場合，MISO 構造で部分マッチングを考慮しない場合，木構造で部分マッチングを考慮した場合でそれぞれ PU を 1 個生成し，PU の面積およびアプリケーションの実行時間を比較した．実験では特に実行時間制約を与えず，生成系で MISO 構造の場合はノード梱包数 6，木構造の場合はノード梱包数 5 の PU を生成した．それより粒度の小さい PU に関しては，再構成系で生成した．ノード梱包数 n，PU の面積 A_{pu}，アプリケーション実行時間 T_{app}，実行時間削減率 $\left(\frac{T_b - T_a}{T_a}\right)$

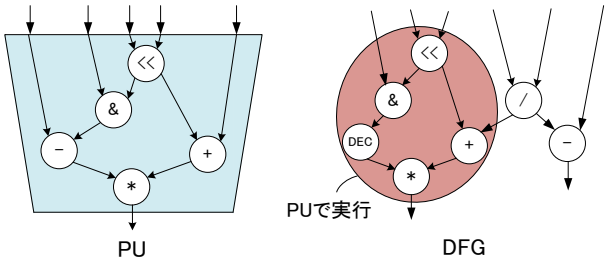

図 5: バインディングの様子.

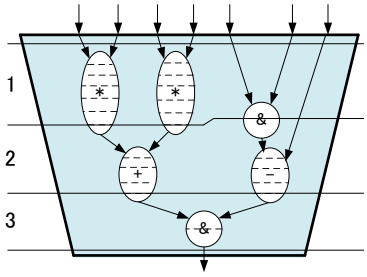

図 6: パイプライン化された PU.

(T_a：PU 付加前のアプリケーション実行時間，T_b：PU 付加後のアプリケーション実行時間) を表 1 に示す.

提案手法では，最大 29.4%の実行時間削減率を達成した．また，MISO 構造で部分マッチングを考慮しなかった場合 (15.8%)，木構造で部分マッチングを考慮した場合 (10%) と比較して，実行時間削減率がそれぞれ 86%，194%改善された.

4 おわりに

本論文では，部分マッチングを考慮し MISO 構造に対応した PU 最適化手法を提案した．提案手法で生成した PU を付加することで，アプリケーション実行時間が最大 29.4%削減された.

表 1: 実験結果.

	MISO/ 部分マッチング○			MISO/ 部分マッチング×		
n	A_{pu} [μm^2]	T_{app} [ms]	T_e (%)	A_{pu} [μm^2]	T_{app} [ms]	T_e (%)
6	24401	6.27	29.4	24401	7.00	15.8
5	24345	6.64	22.2	24345	7.37	10.0
4	24289	7.00	15.8	24289	7.37	10.0
3	24232	7.74	4.8	24232	7.37	10.0
2	12144	7.74	4.8	12144	7.74	4.8

	tree/部分マッチング○		
n	A_{pu} [μm^2]	T_{app} [ms]	T_e (%)
6	–	–	–
5	12313	7.37	10
4	12257	7.74	4.8
3	13400	7.74	4.8
2	113	7.74	4.8