4J-1

OpenMP ハードウェア動作合成におけるコード生成手法の改良と検証

西川諒†　孟林†　山崎勝弘†
立命館大学大学院　理工学研究科†

1. はじめに

我々は LSI の設計時間短縮のため、OpenMP ハードウェア動作合成システムの研究を行なっている。本研究では、コードジェネレータを状態遷移数の削減と演算器生成モジュールの実装の２点で改良を行う。改良前、改良後のコードジェネレータの生成回路を比較して評価する。

2. OpenMP ハードウェア動作合成システム

2.1 システムの構成

OpenMP ハードウェア動作合成システムは、図１の様にアルゴリズム評価とハードウェア合成からなる。アルゴリズム評価ではターゲットハードウェアの動作を OpenMP を用いて記述し、並列性を検証する。その後、OpenMP 記述をハードウェア合成に入力し、ターゲットハードウェアの HDL 記述を得る。

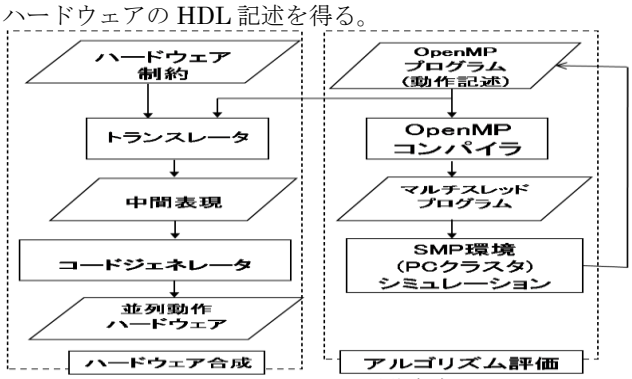

図１：OpenMP ハードウェア動作合成システム

2.2 コードジェネレータ

コードジェネレータは中間表現から HDL コードを生成する。現在、高性能計算研究室には一状態一演算のみの S ジェネレータ[3]、一状態複数演算の M ジェネレータ[4]、及び今回改良した L ジェネレータがある。図２にコード生成の例を示す。

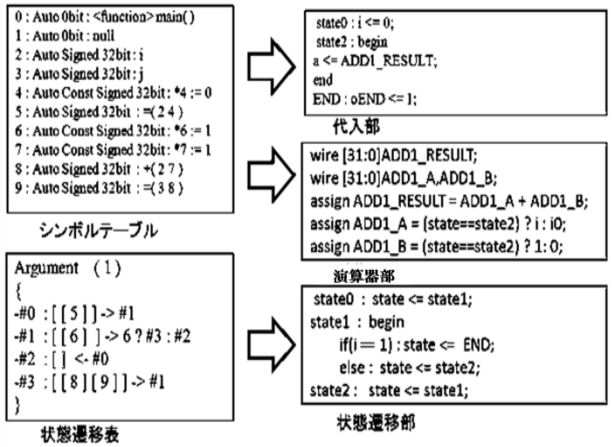

図２：コード生成の例

Improving code generation for an OpenMP hardware behavior synthesis system, Ryo Nishikawa, Lin Meng and Katsuhiro Yamazaki, Graduate School of Science and Engineering, Ritsumeikan University

中間表現はシンボルテーブルと状態遷移表の２部からなり、シンボルテーブルには演算の情報、状態遷移表には状態遷移の情報が記述されている。コードジェネレータはシンボルテーブルから HDL 記述された代入部と演算器部を生成する。また状態遷移表からは状態遷移部(ステートマシン)を生成する。

3. コードジェネレータの改良点

L ジェネレータでは状態遷移数の削減と演算器生成モジュールの実装を行う。各演算の依存関係を計算し、依存関係のない演算を全て同時に行うことで実行時間の短縮を行う。また増加した一状態の演算を行うのに最適な演算器を演算器部にて実装する。図３に L ジェネレータのモジュール構成を示す。

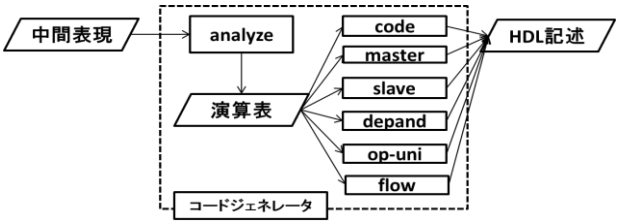

図３：L ジェネレータの構成

3.1 状態遷移数の削減

状態遷移数の削減では、一状態で行える演算を増やすために、各演算の依存関係を調べる必要がある。L ジェネレータでは、まず図４のような、分岐するまでの連続した STATE（セグメント）の演算の依存関係を調べる。セグメント内で、依存関係のない演算を同時に行い、状態遷移数を削減する。S、M ジェネレータでは中間表現のシンボルテーブルの行数、状態遷移表の行だけステートマシンを生成していたため、各演算の依存関係の計算が不十分であった。L ジェネレータでは、初めに図３の analyze モジュールで中間表現から全ての演算情報を読み込み、演算表を作成する。その後にステートマシンを生成して、セグメント内の演算の依存関係を計算する。

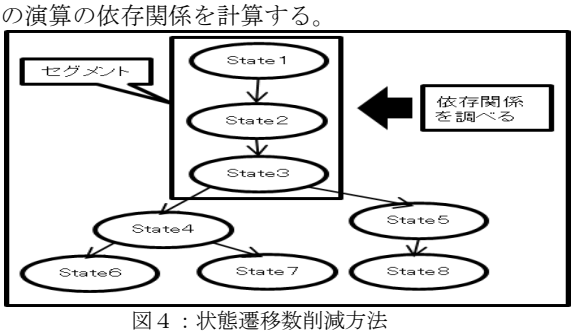

図４：状態遷移数削減方法

3.2 演算器生成モジュールの実装

演算器部は、増えた演算を実際に同時に行うために最適な演算器を生成する。生成は演算表をもとに op-unit にて全演算数だけ行われ、演算結果をレジスタに直接書き込むことで一時保存レジスタを削減した。不必要なレジスタを削減して配線規模を縮小し、遅延を減らすことができる。

演算器部により生成される演算器の例を、図５に示す。

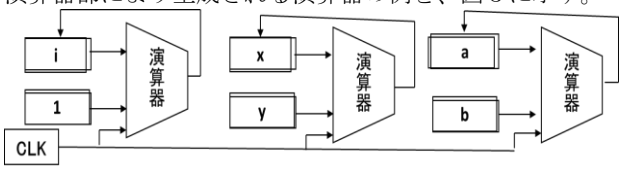

図５：生成される演算器の例

4. 生成回路のアルゴリズム

4.1 マンデルブロ集合

マンデルブロ集合とは、$Z_{n+1} = Z_n^2 + C$, $Z_0 = 0$ で定義される複素数列が“n → ∞”の極限で、無限大に発散しない集合のことである。複素数を使わない場合は Zn を点(Xn, Yn) に、C を点(A, B) にそれぞれ置き代え,$X_{n+1} = X_n^2 - Y_n^2 + A$、$Y_{n+1} = 2X_nY_n + B$ とする。

4.2 バイトニックソート

バイトニック列とは、上昇し下降する列、あるいは並べ替えるとそのようになる列である。バイトニックソートでは、まず各スレーブに値を渡して昇順、降順と交互にソートする。そしてソートできた列を昇順と降順で組み合わせて，複数のバイトニック列を作る。その列を値の大きいもの、小さいものに分け、再びソート、バイトニック列の作成を行う。これを全スレーブのデータが組み合わせるまで行い、最後に各スレーブで昇順ソートを行うとバイトニックソートが完了する。

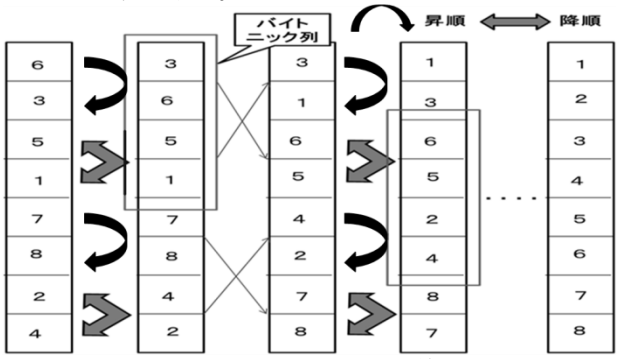

図６：バイトニックソートの並列化

5. 各回路による性能比較

S ジェネレータ、M ジェネレータ、L ジェネレータでマンデルブロ集合、バイトニックソート回路を生成し、比較を行う。実行クロック数、動作周波数、実行時間、及び演算器数の観点から比較する。

5.1 実行クロック数, 動作周波数, 実行時間

表１に各回路の実行クロック数、動作周波数、実行時間を示す。

表１：各回路の実行時間

	Generator	実行クロック数	動作周波数 (MHz)	実行時間
Mandel Brot	S	788.5(k)	44.2	17.84(ns)
	M	197(k)	48.5	4.06(ns)
	L	166.1(k)	48.5	3.42(ns)
Bitonic sort	S	671	49.8	13.47(ps)
	M	251	51.1	4.91(ps)
	L	137	51.1	2.68(ps)

実行クロック数を比較すると、マンデルブロ集合では L ジェネレータの実行クロック数が、S ジェネレータと比べて約 79%、M ジェネレータと比べて約 16%削減できた。バイトニックソートでは L ジェネレータの実行クロック数が、S ジェネレータと比べて約 80%、M ジェネレータと比べて約 45%削減できた。この理由は、依存関係のない演算を同時に行うことで、状態遷移数を減らすことができたためである。また動作周波数では S ジェネレータが１番低く、M ジェネレータと L ジェネレータは同じとなった。実行時間を比較すると、マンデルブロ集合では L ジェネレータが、S ジェネレータと比べて約 81%、M ジェネレータと比べて約 16%削減できた。またバイトニックソートでは L ジェネレータが、S ジェネレータと比べて約 80%、M ジェネレータと比べて約 45%削減できた。動作周波数では、M ジェネレータと L ジェネレータは同じ周波数であったが、実行クロック数で L ジェネレータが優秀なため、実行時間は L ジェネレータの方が短くなったと考えられる。

5.2 演算器数

表２に各回路の演算器数を示す。

表２：演算器数(個)

	S generator	M generator	L generator
Mandel Brot	4	32	42
Bitonic sort	2	12	20

マンデルブロ集合、バイトニックソートともに L ジェネレータが一番多いという結果になった。これは演算器部を実装することで、増加した演算に対応できるだけの演算器が実装されたためである。

5.3 考察

今回の改良の目的は依存関係の無い演算を同時に行うことで状態遷移数を減らし、実行速度を向上させるのと、演算器部を実装し、増加した演算に対応できるだけの演算器を実装することである。改良の結果、マンデルブロ集合、バイトニックソートの両回路で実行速度を向上させることができた。また、表２の演算器数からもわかるように、必要な演算器も実装できており、当初の目標を達成できた。

6. おわりに

今回は OpenMP ハードウェア動作合成システムのコードジェネレータを状態遷移数の削減と演算基部の実装の 2 点で改良した。これらにより、実行時間を最大 80%削減することができた。今後は速度重視、回路規模重視など、ユーザの希望に合わせた回路を生成できるようにしたい。

参考文献

[1] 中谷嵩之,“OpenMP によるハードウェア動作合成システムの設計と検証”，立命館大学大学院理工学研究科修士論文, 2006.

[2] 松崎裕樹, “OpenMP によるハードウェア動作合成システム:コードジェネレータの実装と画像処理による評価”，立命館大学院理工学研究科修士論文, 2008.

[3] 松崎裕樹, 中谷嵩之, 山崎勝弘“OpenMP によるハードウェア動作合成システム:コードジェネレータの実装と画像処理による評価”, 第6回情報科学技術フォーラム論文集 FIT2008, C-008, 2008.

[4]住井大介, “OpenMP ハードウェア動作合成のためのコード生成手法の改良”，立命館理工学部電子情報デザイン学科卒業論文, 2010.