C-006

OpenMP によるハードウェア動作合成システムの設計と検証

Design of a Hardware Behavioral Synthesis System using OpenMP

中谷 嵩之† 松崎 裕樹† 山崎 勝弘†
Takayuki Nakatani Hiroki Matsuzaki Katsuhiro Yamazaki

1. はじめに

LSI の回路規模の増加と設計期間の短縮に伴い，回路の動作など抽象的な記述からハードウェアを自動合成する動作合成技術が実際の設計に適用されている．しかし，現在の動作合成技術では設計者の意図する並列化を行った回路を自動生成することは難しく，設計者による手動での並列化に頼っており，設計者の負担が大きい．また，並列化したハードウェアの検証においては，主に RTL レベルのシミュレータなどを用いるため，検証に時間がかかるという問題もある．本研究では動作レベル記述に並列プログラミング言語 OpenMP を用いた動作合成手法を提案する．回路設計において並列動作の記述を容易にすることにより，ハードウェアの動作合成において設計者の意図した並列化手法を実現する．本論文では OpenMP を用いた動作合成システムの構成と実装，データ分割/タスク分割によるハードウェア並列化，及び FIR フィルタとウェーブレット変換の並列化を対象とした本提案手法の評価について述べる．

2. OpenMP による動作合成システム

2.1 OpenMPによる動作合成システムの構成

OpenMP とは共有メモリ環境や SMP 環境における並列処理用 API である．C，Fortran のプログラミング言語にプラグマを追加することにより，繰り返し処理を並列化する並列リージョン，及び複数の異なる処理を並列化する並列セクションの範囲を指示することができる．逐次プログラムに対し，共有変数や並列動作するノード数などの並列化構造を容易に追記可能である．

図 1 に OpenMP を用いた動作合成システムの構成を示す．本システムでは OpenMP を用いて，逐次プログラムの並列化を段階的に行い，図 1 のアルゴリズム評価系において，並列化手法の有効性や性能を評価する．SMP 環境で，ソースコードのテスト，アルゴリズムの評価を設計の早期にかつ高速にシミュレーションできることが，本システムの最大の特徴である．

並列処理の性能を検証後，図 1 左側のハードウェア動作合成系において動作合成を行う．既存の動作合成において，並列化は主に演算単位で局所的な処理に対して行われ[1]，データ構造やアルゴリズムに依存した大規模な並列化やパイプライン化を自動で検出することは非常に難しい．本システムでは設計者が OpenMP の並列化指示文を用いることによって，明示的に逐次処理を並列実行モデルへ変換する．その際，同期制御や排他処理などを記述する必要がないため，設計者の負担が軽減される．また，並列化の制御ではなく構造を記述するため，メモリアクセスやデータの割り当てなど，並列化構造を考慮したハードウェアの自動生成が可能である．

† 立命館大学大学院 理工学研究科, Graduate School of Science and Engineering, Ritsumeikan University

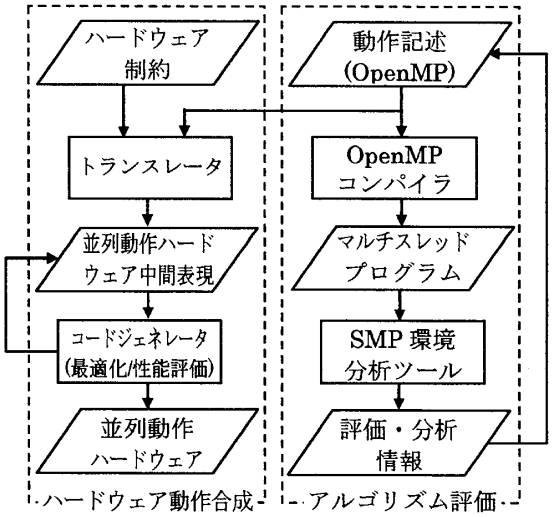

図 1：OpenMP による動作合成システムの構成

2.2 OpenMPの実行モデルと生成ハードウェア

OpenMP の共有メモリモデル環境における実行モデルを図 2 に示す．常に逐次処理から実行を開始し，並列処理の開始(fork)時にスレッドを動的に生成する．処理を複数のプロセッサに割り当てて並列実行した後，複数スレッドの結果を join で回収して，逐次処理へ戻る． SMP 環境実行での fork ではスレッドの生成に時間的コストが必要であるため，処理の粒度がある程度大きくなければ並列効果が得られないが，並列処理中は通信や排他制御など，複雑な制御は必要ない．

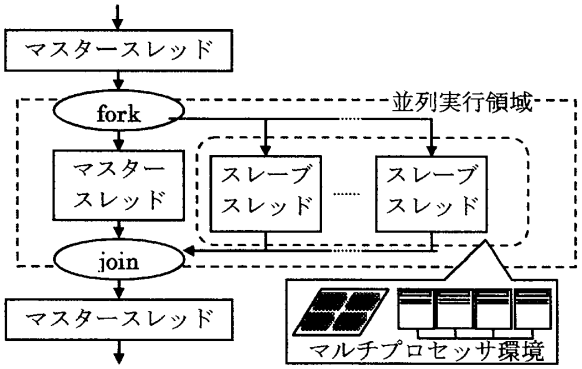

図 2：OpenMP の実行モデル

動作合成によって生成されるハードウェアは，図 3 のように逐次処理ハードウェアと並列動作ハードウェアによって構成される．逐次処理ハードウェアは状態を制御する有限状態機械(FSM)とデータパスによって構成され，逐次処理や並列動作ハードウェアの制御を行う．並列動作，ハードウェアは同時に動作可能な複数の処理ノードと，

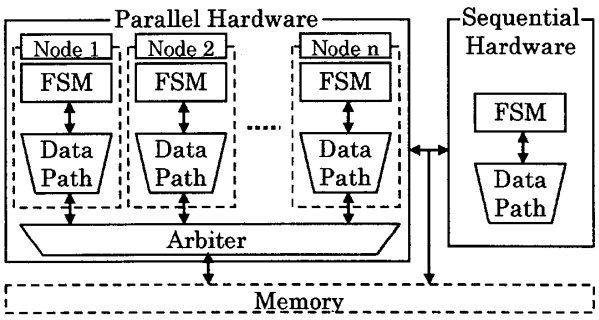

図 3：並列動作ハードウェアの構成

それらのメモリアクセスを調停するアービタで構成され動作合成時に並列化構造を考慮して生成される．並列動作ハードウェア内部の処理ノードは，それぞれが実行モデルの処理ノードに対応した，逐次処理データパスであり，内部に自分のみが使用するプライベート変数を持つ．他の処理ノードと共有して使用する変数はメモリへ格納され，複数の処理ノードがアクセスした場合，アービタによって調停が行われ，許可された処理ノード以外は停止する．並列処理開始時は，fork 処理で並列処理に必要な情報が並列動作ハードウェアを構成する各ノードに渡され，動作を開始する．OpenMP の SMP 環境での実行モデルと比べ，スレッド生成や引数のコストが必要ないため，処理の粒度が小さくても高い並列化効果が期待できる．

3. 共有メモリモデルによるハードウェア並列化

3.1 データ分割による並列化

データ分割による並列リージョンでは，大量のデータに対する繰り返し処理を，複数のノードに分配する．動作を開始すると，制御回路は繰り返し処理を分割し，ノードに割り当てる．100 回の繰り返し処理を 4 ノードに割り当てた場合の実行モデルを図 4 に示す．

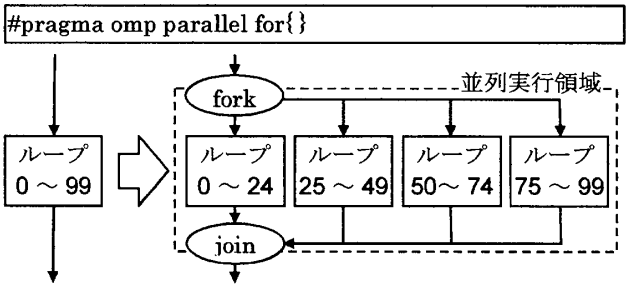

図 4　データ分割の実行モデル

このモデルで実行するためには 0 から 99 までの和のように，各繰り返しに依存性がなく，全く独立して実行できること(データ並列性)が必要である．

図 4 の実行モデルを実現する並列動作ハードウェアの構成を図 5 に示す．並列動作ハードウェアの各処理ノードは実行モデルの各処理ノードに対応し，全て同じ FSM とデータパスで構成される．このため,fork 処理において繰り返し処理の範囲や必要な変数のコピーなどが行われる．データ分割による並列化は，大量のデータに対し，同一の処理を行う画像処理や信号処理などに適している．

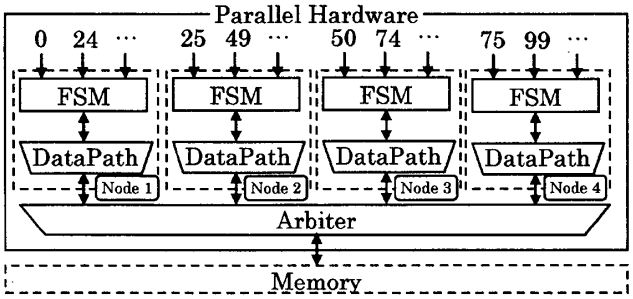

図 5　データ分割の並列動作ハードウェア構成

3.2 タスク分割による並列化

タスク分割では，並列セクションにおいて同時実行可能な処理を指定し，複数のノードに分配する．並列処理中は全ての分割処理が並列に動作し，各分割処理の終了時に他の分割処理と同期を取る．異なる処理 A，B，C を並列化した実行モデルを図 6 に示す．

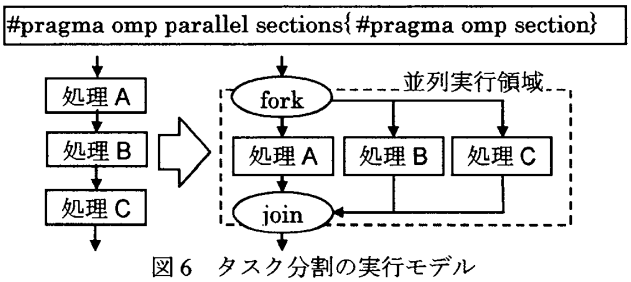

図 6　タスク分割の実行モデル

このモデルを実行するためには，異なる処理 A,B,C の間に依存関係が無く，同時実行できる必要がある．また，A,B,C の負荷が均衡していることが望ましい．

タスク分割の並列ハードウェア構成を図 7 に示す．各処理ノードは分割した処理に対応し，異なる FSM とデータパスが含まれる．fork 処理では各処理に応じて必要なデータのコピーが行われ，処理ノードは動作を開始する．

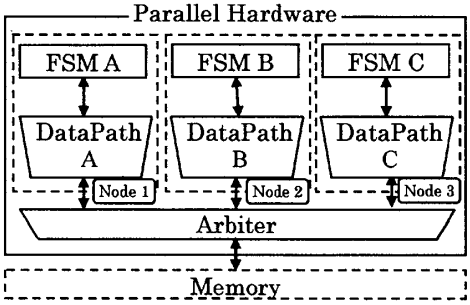

図 7　タスク分割の並列動作ハードウェア構成

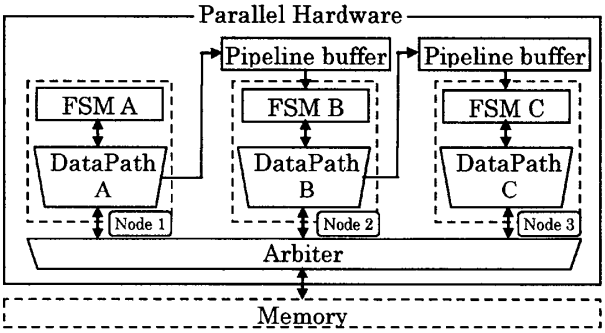

図 8　タスク分割によるパイプライン化

タスク分割の別の使用法として，依存性がある連続した処理を時系列に従って分割し，各分割処理の結果を次の分割処理の入力とすることで，パイプライン処理可能である．パイプライン処理を行う処理ノード間に直前の処理ノードの結果を保存するバッファを置き，各処理ノードの並列処理を可能とする．パイプライン処理の構成を図 8 に示す．

4. FIR フィルタとウェーブレット変換の並列化

4.1 対象のアルゴリズムと並列化構造

FIR フィルタ，及びウェーブレット変換の並列動作ハードウェアを対象とし，動作合成システムの性能評価を行った．動作合成においては OpenMP 記述を入力とし，RTL 中間表現までを動作合成システムによって自動で生成し，中間表現から手作業で HDL 記述へ変換した．

(1) FIR フィルタ

FIR フィルタはデジタル信号処理における非常に一般的なフィルタであり，主に特定の周波数信号の除去や，抽出を行う場合に用いられる．本研究の評価に用いた直接形 FIR フィルタの構成を図 9 に示す．FIR フィルタは過去の入力の保持，それらへの重みの乗算，乗算結果の足し合わせの 3 つの処理で構成されており，タスク分割では 3 つの処理を並列化し，パイプライン構成で並列化した．

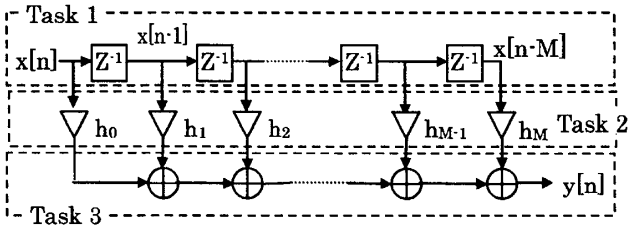

図 9 FIR フィルタの構成

(2) ウェーブレット変換

時間－周波数変換処理を行うウェーブレット変換は周波数変換処理を行うフーリエ変換と同じように，スペクトル解析に用いられる信号処理の一つであり，画像や音声の特性の解析に用いられる．本研究で用いた Haar のウェーブレット変換の構成を図 10 に示す．Haar のウェーブレット変換は信号列の隣接する要素に対する加算と減算，右シフトで構成可能であり，信号列の長さを半分にして繰り返す．タスク分割では加算と減算，右シフトを 2 つの処理に並列化した．

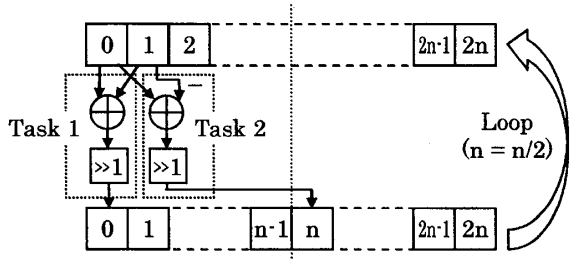

図 10 Haar のウェーブレット変換の構成

4.2 並列メモリアクセス

FIR フィルタとウェーブレット変換のデータ分割におけるメモリアクセスのタイミングを図 11 に示す．データ分割では，全ての処理ノードが基本的に同じ動作を行うため，全ての処理ノードから同時にメモリアクセスが要求される．メモリアクセスが競合したノードはストールし，動作が遅れる．今回の検証ではメモリアドレスの計算とメモリアクセス間で実行される演算が 12 サイクルあり，ノード数よりも多かったため，以降はノード間でメモリアクセスが重複せず，ストールが発生しなかった．

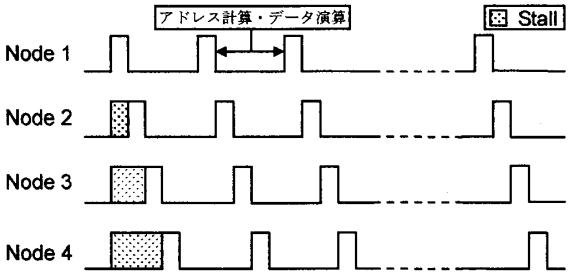

図 11 データ分割における同時メモリアクセス

すなわち，メモリアクセス間に十分な演算量がある場合，ストールを発生させずに処理できることがわかった．このため，並列メモリアクセスの環境に合わせて，メモリアクセス間の演算量を調整し，メモリアクセスの頻度を最適化することが可能である．また，演算量が固定の場合，演算に必要なサイクル数を増やすことで，回路面積の節約が可能である．

さらに，メモリバンクやキャッシュなどを用いると，1 サイクル中に同時にアクセスが可能なノードが増える．これにより，メモリアクセスの頻度が高くてもストールを発生させずに実行が可能であり，回路面積よりも処理速度が重要となる場合に有効である．

4.3 動作合成系による生成ハードウェアの評価

OpenMP 記述のアルゴリズム評価と動作合成系によるハードウェアそれぞれにおいて処理に必要な時間を計測した．また，ハードウェアでは必要回路面積も計測した．

FIR フィルタは次数 16，ウェーブレット変換は 256 点 1 次元ウェーブレット変換を対象とし，10000 点のサンプルデータを入力とした．アルゴリズム評価において OpenMP 記述に用いたコンパイラは Intel C/C++ Compiler 9.0，実行した SMP 環境は，Intel 社 Dual Core Xeon 2.8G[Hz]を 2 個搭載した Quad Core マルチプロセッサ PC である．動作合成系によるハードウェアの評価では，実行時間の計測にサイクルレベルシミュレータである Menter Graphics 社の ModelSim SE 5.8c を用い，回路面積の計測には論理合成ツールとして Xilinx 社の ISE7.1 を用いた．計測結果を表 1,2,3 に示す．表 1,2,3 のタスク分割では，FIR フィルタは 3 タスクを，ウェーブレット変換は 2 タスクを並列実行した結果を示す．

表 1　SMP 環境での実行時間(単位：ms)

	FIR	Wavelet
逐次	12.8(1.00)	87.5(1.00)
タスク分割	29.42(0.43)	1803(0.04)
データ分割(2 CPU)	9.06(1.41)	57.1(1.53)
データ分割(4 CPU)	7.48(1.71)	32.8(2.67)

※　()内の値は逐次との速度向上比

表 2　動作合成ハードウェアの実行時間(単位：10^6 cyle)

	FIR	Wavelet
逐次	31.7(1.00)	100.69(1.00)
タスク分割	20.6(1.54)	115.99(0.87)
データ分割(2 Node)	15.9(1.99)	50.38(1.99)
データ分割(4 Node)	8.0(3.96)	25.21(3.99)

※　()内の値は逐次との速度向上比

表 3　動作合成ハードウェアの回路面積(単位：Slice)

	FIR	Wavelet
逐次	1256(1.00)	21004(1.00)
タスク分割	2595(2.06)	29508(1.40)
データ分割(2 Node)	2655(2.11)	52875(2.51)
データ分割(4 Node)	5248(4.17)	104646(4.98)

※　()内の値は逐次との回路面積比

表 1 の SMP 環境での実行では，データ分割では 4 ノードで約 2.67 倍の速度向上が得られたが，タスク分割は速度が低下した．アプリケーションを比較すると，処理負荷が高いウェーブレット変換の速度向上が高かった．

表 2 の動作合成 HW による実行時間では，データ分割において，両アプリケーション共に，2 ノードで約 2 倍，4 ノードで約 4 倍と理想的な速度向上比が得られた．表 2,3 を比較すると，増加した回路面積も，最大で速度向上比の 1.25 倍程度であった．理想的な速度向上比が得られた理由として，図 11 で示したように，ノード間におけるメモリアクセス要求の衝突が殆ど発生しなかったことが上げられる．Score を用いた PC クラスタでは，低レベル通信ライブラリやゼロコピー通信を用いて，共有変数のアクセスのオーバーヘッドを極力低減させている．本システムでは，アービタとアクセスの工夫により，高速化が期待できる．

表 1,2 の速度向上比を比較すると，動作合成ハードウェアは SMP 環境での実行と比べ，高い速度向上が得られた．これは SMP 環境でのアルゴリズムの改善などにより並列化効果が得られる場合，動作合成ハードウェアにおいて同等以上の速度向上が得られることを保証し，本提案手法による動作合成システムが回路設計において有効であることを示している．

SMP 環境での実行と比べ，ハードウェアで速度向上が得られやすい原因は，動作合成時に並列ノードを生成しておくため，fork などのスレッド生成の時間が必要ないためである．しかし，静的にハードウェアを生成しておく必要があり，アービタなどの追加の回路も必要なことから，回路面積が並列化効果以上に増加している．従って実行速度と回路面積のトレードオフを考慮することが重要である．

5. 動作合成システムの評価

動作合成システムを用いることで，データ分割により理想的な速度向上が得られるハードウェアを生成することができた．一方，タスク分割ではデータ分割ほどの速度向上が得られなかった．これは FIR フィルタとウェーブレット変換の処理の粒度に違いがあり，ウェーブレット変換相当の処理負荷がなければ，並列化効果が fork-join 処理のコストを吸収できないためである．OpenMP は大規模な処理を並列化する API であり，生成されたハードウェアも粒度の大きな処理に有効であることがわかった．また，画像・音声処理など均等で負荷が大きい処理に対し，データ分割による並列化が非常に有効であり，本手法を用いることで効果の高い並列化が容易に可能となる．

SMP 環境での実行では，非常に高速な検証とアルゴリズムの評価を行うことが可能であった．FIR フィルタとウェーブレット変換の，動作合成ハードウェアを用いたサイクルレベルシミュレータでの検証時間を表 4 に示す．

表 4　サイクルレベルシミュレータの検証時間(単位：ms)

	FIR	Wavelet
逐次	3968(1.00)	652719(1.00)
タスク分割	4375(1.10)	842047(1.29)
データ分割(2 Node)	4672(1.18)	2333859(3.57)
データ分割(4 Node)	4938(1.24)	4239344(6.49)

※　()内の値は逐次との時間比

表 1 の検証は，サイクルレベルシミュレータでの検証と比べ数十から十万倍程度高速に実行できる．また，シミュレータの検証では並列化を行えば行うほど検証に必要な時間が増加したのに対し，SMP 環境では並列化により必要な時間が減少した．すなわち，並列動作部分が増加するハードウェアにおいて，SMP 環境は，アルゴリズムや並列化手法の高速な検証に適していると考えられる．

6. おわりに

本研究では OpenMP を用いた動作合成システムにおける並列化手法，及び，FIR フィルタとウェーブレット変換に対して，データ分割によるハードウェア並列化が有効であることを示した．今後の課題として，パイプライン化の有効性の検証，各種アプリケーションの評価，及びコードジェネレータの作成によるシステム全体の統合があげられる．

参考文献

[1] 森江善之，冨山宏行，村上和彰：“動作合成の効率化を指向した動作レベル記述・トランスフォーメーション”，情報処理学会研究報告 Vol.2003 No.120, 2003.

[2] 鈴木，他：“C 言語によるサイクル精度でのハードウェア/ソフトウェア協調検証手法”，シャープ技法第 92 号，pp.78-83，2005.

[3] 中谷，山崎：“OpenMP によるハードウェア動作合成システムの検討”,FIT2006,2006.