計算機アーキテクチャ 110−14
設 計 自 動 化 73−14
（1995. 1. 20）

マルチプロセッサによる分散処理を意識した専用プロセッサ設計支援システムSYARDSの構築

樋渡 仁　　白井 克彦

早稲田大学　理工学部

東京都新宿区大久保3-4-1

あらまし

近年、高位合成を利用した設計支援システムが実用化されつつあるが、信号処理などのリアルタイム性を必要とする応用例には力不足であることが否めない。そこで、あらかじめ分散処理アーキテクチャに割り付けることを想定してシステム設計を行なう分散合成を設計支援システムSYARDSに導入し、HMM音声認識を含む幾つかの信号処理アルゴリズムに対して適用を試みた。また、それぞれに関して実行時間やハードウェア資源の評価を行ない、新たに導入した設計法がこれらの信号処理アルゴリズムに対して有効に動作することを示した。

SYARDS: A Design System for Special Purpose Processors using Distributed Processing with a Multiprocessor

Jin HIWATASHI　Katsuhiko SHIRAI

School of Science and Engineering, Waseda University

3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169, Japan

Abstract

In recent years, although design systems using high-level synthesis are becoming practicable, yet they are not powerful enough to design the digital signal processing (DSP) applications which are in need of real-time processing. In this paper, we introduce the system synthesis on assumption that a behavioral description is allocated into an architecture for distributed processing, and design several DSP algorithms which include the Forward algorithm for HMM speech recognition. We also evaluate execution time and hardware resources of those applications, and indicate that the new design methodology is available for those DSP algorithms.

1 まえがき

近年のVLSI製造技術の進歩にともない、さまざまなVLSIを実現することが可能となり、特定のアルゴリズムを高速に実行する特定用途向け集積回路(ASIC)の応用分野も広がっている。一方、規模の増大にともない、システム設計もより複雑化、多様化し、設計期間の長期化とコストの増加が指摘されている。今後、VLSIの多様な分野への応用が進む中で、専用プロセッサへの要求は増加していくと考えられるが、そのためにはVLSIをより容易により短時間で設計できる設計支援システムが必要不可欠である。このような背景から、われわれはハードウェア設計に関する専門知識を持たないユーザを対象とし、アルゴリズムによる動作記述を入力とした専用プロセッサ設計支援システムSYARDSの提案と研究を行なってきた[1], [2], [3]。

さらに、システム設計をとりまく状況としては、VHDLを中心とした高位合成による設計環境の充実やハードウェア/ソフトウェア協調設計の重要性などが指摘されている。これは、研究の中心がシステムにおいて構成要素の設計を支援する段階から、どのような構造を採用すれば効率良い設計ができるかに移行していることを意味する。このような状況を背景にして、本稿においては分散処理アーキテクチャを想定した新たな設計法として分散合成を提案し、さまざまな信号処理アルゴリズムに対して適用と評価を行なった。

2 設計手法

2.1 分散合成

通常の高位合成では、特別な場合を除いてターゲットアーキテクチャとして単一プロセッサ構成を採用するが、リアルタイム処理を必要とする応用、特に画像や音声などの大量の情報を扱う処理を必要とするシステムを設計する場合には、このような設計法では性能的に要求条件を満たさない可能性がある。この問題を解決するひとつの方法として、本稿ではターゲットアーキテクチャとして分散処理アーキテクチャを採用して、あらかじめ密結合マルチプロセッサに割り付けることを想定した分散合成を提案する(図1参照)。このシステムにおいては、演算は各処理部で実行しデータ転送は共有記憶を用いるが、この実装形態にとらわれるものではない。

従来の高位合成システムにおいても特定の最適化により、このような分散処理アーキテクチャを持つシステムを合成することが可能であれば問題は生じない。しかし、実際には高位合成システムでこのような最適化を行なうものはないために、信号処理のような応用を考えた場合、分散合成の重要度は増加する。

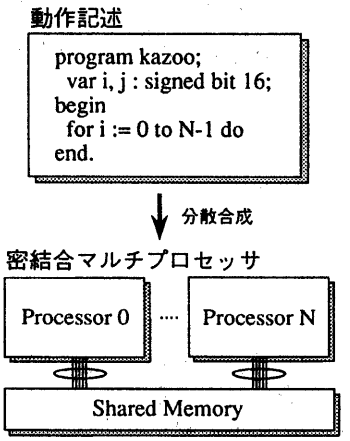

図 1: 分散合成の概念

2.2 並列記述の導入

高位合成において並列性を利用して性能向上を獲得するには、大別して2種類の戦略がある。ひとつは細粒度の並列性を自動で抽出する戦略であり、もうひとつは粗粒度の並列性を手動で抽出する戦略である。細粒度の並列性を自動で抽出する戦略に関しては、多くの高位合成システムが主要な最適化手法として利用しているが、粗粒度の並列性を手動で抽出する戦略に関しては、pbegin～pend のような並列実行記述を用意しているが十分に利用されているとはいえない状況である。

そこで分散合成においては、積極的に並列記述を導入して、粗粒度の並列性を最大限に利用することを考える。ここでは、並列プログラミング[4]における共有記憶, 排他制御, バリア同期などの機構を関数呼び出しの形式で提供した。新たに導入された関数を図2に示し、以下で各関数の機能を解説する。

関数 process_fork(nproc) が親プロセッサから呼び出されると、nproc-1個の子プロセッサが動作を開始する。各プロセッサは固有のプロセッサ識別子 process_id の値により自分が実行すべき命令列を検出することができる。子プロセッサが手続き process_join を呼び出すと動作を終了し、親プロセッサが呼び出すとすべての子プロセッサの動作が終了するまで待機する。

スピンロックは、排他制御されている領域を保護する機構で、保護領域は同時に1個のプロセッサしか実

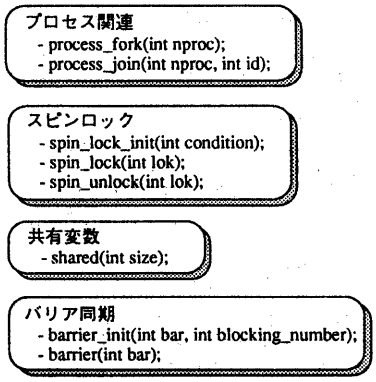

図 2: 新たに導入した関数

行できない。関数 spin_lock_init はスピンロック機構を初期化して、手続き spin_lock, spin_unlock によって領域の保護と解放を行なう。手続き shared は、すべてのプロセッサで共有可能な変数を宣言する。ここで、子プロセッサは親プロセッサの正確な複写であり、共有変数以外のすべての変数を局所的に保持する。

バリア同期は、特定のプロセッサが先の命令列を実行しないように各プロセッサの同期を取る機構である。関数 barrier_init によりバリア同期機構を初期化して、手続き barrier により各プロセッサがその実行点に到着するまで待機させることができる。

3 設計支援システム SYARDS

3.1 システム構成

新たに構築したシステム (SYARDS) のシステム構成を図 3に示す。動作記述を入力とするシステム設計においては、あらかじめ実現すべきアルゴリズムが高級言語で記述された形式で存在する場合が多い。

ここでは、プロトタイピングという段階を導入することにより、これらの動作記述に新たに導入した関数による並列情報を付加し、既存のコンパイラを用いることで記述検証や動作確認を行なえるようにした。このためには、新たに導入した各関数をなんらかの手段で実装する必要が生じるが、ここでは各関数を UNIX のシステムコールをもちいて実現し、最終的に実行時にライブラリとして結合する形式を採用した。

次に、この結果を利用して SYARDS Pascal による動作記述を行なう。この段階では、プロトタイピングにおいて C で記述された動作記述を SYARDS が処理できる SYARDS Pascal に変換する必要がある。

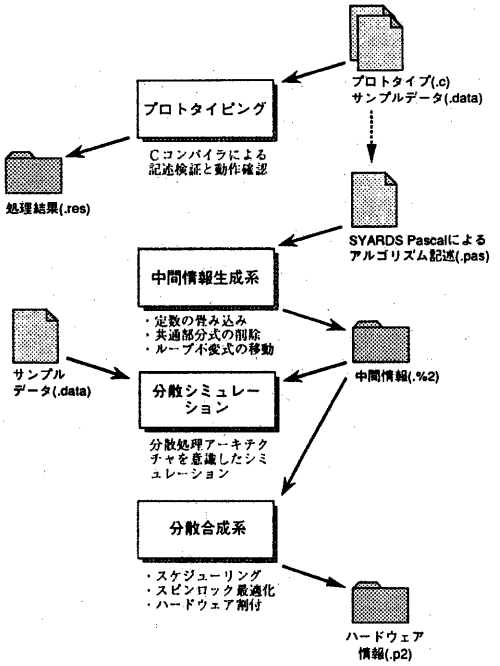

図 3: システム構成図

さらに、前段階で記述を行なった動作記述を中間情報生成系で処理することにより、並列情報を含んだ中間情報を獲得できる。分散シミュレータは、この中間情報を利用してこの段階での動作確認を可能にするものである。分散シミュレーションを利用することにより、動作記述を実行するのに必要なステップ数を調査できるため、分散アーキテクチャで実行した場合の分散合成を効果を計測することが可能になる。

最終的に、分散合成系でスケジューリングやスピンロック最適化, ハードウェア割付などを行ないハードウェア情報を得る。これにより、動作記述の実行に必要なハードウェア資源を見積もること可能になる。なお、スケジューリングには資源に制約を課さない ASAP(As Soon As Possible) スケジューリングを、レジスタ割付には左端アルゴリズム (left-edge algorithm) を利用した [5]。

3.2 プロセッサ最適化手法

この実装においては、並列記述をユーザが行なうことにより各処理部を並列実行させるため粗粒度のスケジューリングは、記述の段階ですでに終了している。しかし、各処理部に対して最適化手法を適用することで、

各処理部を高速動作させることが可能になる。ここで、プロセッサ最適化とは、プロセッサ構成が決定した後に各処理部に与えられた命令列をより一層高速に実行するように最適化を行なうことを意味する。

当初は、各処理部の実行する命令列の違いを利用して、さまざまな命令セットを持った処理部からなるシステムを設計する方針であったが、システムが扱う並列性が粗粒度であることを考慮すると各処理部は同質になることが予想されるため、本実装においては各処理部が同質に設計される手法を採用した。具体的には各処理部を高位合成によって合成して、最終的にはハードウェア記述言語で出力することを目標としている。また、最終的な論理合成に関しては、論理合成システム PARTHENON[6] を利用することを想定する。

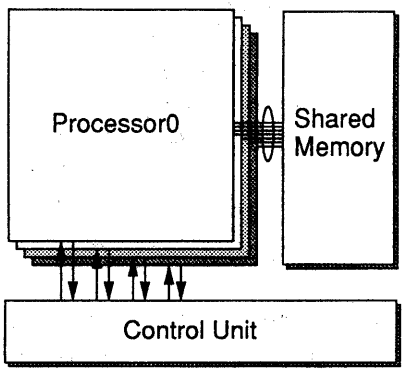

図 4: 対象とするアーキテクチャ

3.3 実現手法

ここでは、最終的にシステムを実現するために必要なハードウェア記述言語による記述をどのようにして獲得するかについて述べる。本システムが対象とするターゲットアーキテクチャを図 4に示す。各処理部に関しては、ハードウェア割付等が終了したハードウェア情報が出力されるので、これを処理することによりハードウェア記述言語による記述を獲得することができる。

次に、各処理部の動作を制御する制御部は、分散合成において導入した関数 (spin_lock, spin_unlock, barrier 等) の要求が生じた場合、然るべき動作を各処理部に指示するように設計する必要がある。この制御部を実現するためには、ハードウェア記述言語で記述された制御部の雛型を用意して、任意のアルゴリズムが必要な機能だけを合成するという処理が必要になる。このようにして得られたプロセッサ部と制御部のハードウェア記述言語による記述を既存の論理合成ツール (PARTHENON) などを利用することで論理合成し、ハードウェアとしての実現や実現に必要な資源の見積り等を行なうことができる。

○画像アルゴリズム
- 離散コサイン変換 DCT (規模：192)
- テンプレートマッチング TEMP (規模：313)
- メディアンフィルタ MEDIAN (規模：264)

○音声アルゴリズム
- ピッチ抽出 PITCH (規模：142)
- パーコール格子型フィルタ PARCOR (規模：81)
- 自己相関関数 AUCOR (規模：42)
- HMM 音声認識 FORWARD (規模：348)
- ハミング窓 HAMM (規模：50)

ただし、規模とは中間情報段階での静的ステップ数を示す

図 5: 適用した信号処理アルゴリズム

4 適用例

4.1 信号処理アルゴリズム

本システムを利用して設計を行なったアルゴリズムは、信号処理アルゴリズムを中心に 8 種類である (図 5 参照)。その内訳は画像処理アルゴリズムが 3 種類と音声アルゴリズムが 5 種類であり、概ね実行する際にリアルタイム性を必要とするものを選択している。

また、図 5において、規模というのは中間情報段階での静的なステップ数を意味しており、アルゴリズムの規模を大まかに表現している。ここでは、これらの信号処理アルゴリズムに対して本システムの適用を試み、分散シミュレータを利用することにより分散合成の効果を計測する。さらに、分散合成系を用いることによって、各処理部に対するプロセッサ最適化の効果とアルゴリズム実行に必要なハードウェア資源 (レジスタ, 機能ユニット等) を測定することが可能になる。

4.2 HMM 音声認識

以上で述べた信号処理アルゴリズムの中でも比較的成功した例として、ここでは HMM(Hidden Markov Model) 音声認識アルゴリズム [7] に関して、詳しく取り上げて本システムにおいてどのように設計が行なわれるかについて解説する。

4.2.1 動作記述

まず、設計を行なう際にはプロトタイピングの段階でC言語による動作記述を利用して記述や動作について問題がないか検証する。次に、プロトタイピングの結果を用いて、SYARDS Pascal によりHMM音声認識アルゴリズムを記述する(図6参照)。ここに図示したものはアルゴリズムの主要な部分であり、実際にはある単語のVQコードが入力されると、あらかじめ格納されている遷移確率や出現確率を利用して各単語との尤度を計算する。1単語の認識を行なうためにはあらかじめ格納されているnwordの単語との尤度を計算する必要がある。

通常の単一プロセッサ構成であれば、逐次的に各単語との尤度を計算することになるが、われわれが提供する分散合成システムでは自分の任意のプロセッサ数を設定して尤度の計算を並列的に行なうことが可能になる。これを具体化したのが、図6に示した動作記述であり、実際には並列プログラミングにおけるループ分割という手法を用いている。

```
{ process forking }
id := process_fork(nproc);
w := id;
while w < nword do
 begin ...
  for t := 1 to num do
   begin
    spin_lock(lok);
    alpha[t] := alpha[t-1] * a[0] * b[code[t-1]-1];
    spin_unlock(lok);
    for i := 1 to nstate-1 do
     begin
      spin_lock(lok);
      bb  := b[i*vq_level+code[t-1]-1];
      aa1 := a[(i-1)*nstate+i];
      aa  := a[i    *nstate+i];
      spin_unlock(lok);
      alpha[i*max_nsymbol+t]
            := alpha[(i-1)*max_nsymbol+t-1] * aa1 * bb
             + alpha[i    *max_nsymbol+t-1] * aa  * bb;
     end; ...
  tmp := ln(alpha[(nstate-1)*max_nsymbol+num]) - sum_c;
  spin_lock(lok); likeli[w] := tmp; spin_unlock(lok);
  w := w + nproc;
 end;
{ process joining }
process_join(nproc, id);
```

図 6: HMM 音声認識の動作記述

変数 id には各処理部固有の値が格納されており、ループの最後でプロセッサ数 nproc を変数 w に加えることにより、例えばプロセッサ数が4であれば単語0, 4, 8番との尤度の計算がプロセッサ0に割り当てられるということになる。このような方法により、単一プロセッサ構成で実行する場合よりも4倍の認識性能を獲得することができる。これは、速度を要求される応用分野では重要であり、特にリアルタイム処理が必要な場合には絶大な効果をもたらす。

4.2.2 分散シミュレーション

中間情報生成系をもちいることで動作記述から中間情報を生成できるが、この中間情報を分散シミュレータに入力してシミュレーションを行なうと、分散アーキテクチャにおける動作記述の実行ステップを計測することができる。この結果を利用すれば本実装において導入された分散合成の効果を計測することが可能になる。HMM音声認識の場合にはプロセッサ数が4程度まではほぼ線形に分散処理の効果が現れるが、それ以降ではその効果はほぼ飽和していくという結果が得られた。

次に、スケジューリング前とスケジューリング後の中間情報をそれぞれ分散シミュレータでシミュレーションすることにより、プロセッサ最適化手法としてのスケジューリングの効果を計測することができる。詳しい結果については以降で言及するが、おおむね6割程度のステップ数減少がみられ、HMM音声認識はこれ以外の適用例よりも本プロセッサ最適化手法が効果的であることが示された。

```
(MODULE FORWARD
 (INPUT
  (HMM_AA FLOAT) (HMM_BB FLOAT) (VQ_CODE INTEGER))
 (OUTPUT
  (LIKELIP FLOAT))
 (EXT-RAM
  (A FLOAT 100) (B FLOAT 2560) (LIKELI FLOAT 8))
 (EXT-REG
  (LOK INTEGER))
 (UNIT
  (MUL 2) (FMUL 2) (ALU 2) (FALU 1) (FDIV 1))
 (RAM
  (ALPHA FLOAT 1500) (C FLOAT 150) (CODE INTEGER 150))
 (REG
  (R1 INTEGER) ... (R20 INTEGER) (F1 FLOAT) ... (F6 FLOAT))
 (STATE ...
  (B0038
   (79 (MUL R4 R9 256) (LOAD R3 CODE[R11]) (SUB R2 R9 1)
       (MUL R1 R9 10))
   (80 (ADD R3 R4 R3) (MUL R2 R2 10) (ADD R1 R1 R9))
   (81 (SPIN_LOCK LOK))
   (82 (SUB R2 R3 1) (ADD R1 R2 R9) (LOAD F1 A[R1]))
   (83 (LOAD F2 B[R2]) (LOAD F3 A[R1]))
   (84 (SPIN_UNLOCK LOK))
   (85 (SUB R2 R9 1) (MUL R1 R9 150))
   (86 (MUL R2 R2 150) (ADD R1 R1 R6) (ADD R9 R9 1))
   (87 (ADD R3 R2 R6) (SUB R2 R1 1))
   (88 (SUB R2 R3 1) (LOAD F4 ALPHA[R2]))
   (89 (LOAD F4 ALPHA[R2]) (FMUL F1 F4 F1))
   (90 (FMUL F3 F4 F3) (FMUL F1 F1 F2))
   (91 (FMUL F2 F3 F2))
   (92 (FADD F1 F2 F1))
   (93 (STORE ALPHA[R1] F1))
   (94 (JMP B0037)))
```

図 7: HMM 音声認識のハードウェア情報

4.2.3 分散合成

分散合成系においては中間情報をスケジューリングにより並列実行可能な命令列を抽出し、レジスタ割付, 機能ユニット割付等を行ない実行に必要なハードウェ

ア資源を見積もることが可能になる。HMM 音声認識についてスケジューリング、ハードウェア割付を行なった結果を図 7に示す。

さて、基本ブロック B0038 は HMM 音声認識において中心的な役割を果たす箇所であるが、例えばステップ 79 においては 2 個の乗算, 1 個の減算, 1 個の配列読込みが並列に実行可能であることがわかる。このように、HMM 音声認識はアルゴリズム自体が比較的多量の細粒度の並列性を含んでおり、プロセッサ最適化手法としてもちいたスケジューリングが有効な例のひとつであるといえる。

次に、各変数に対してレジスタ割付が適用されると、基本ブロック内に閉じた局所変数は他の局所変数とレジスタを共有するように割り付けられ、それ以外の大域変数には完全なレジスタが割り付けられる。さらに、機能ユニット割付においては、同一ステップ内では各演算器を共有できないため、同一ステップで使用される各演算器の最大数を求めることにより実現に必要な機能ユニット数を算出する。

実際に HMM 音声認識にレジスタ割付, 機能ユニット割付を行なうと、プロセッサ 1 個当たり整数レジスタ 20 本, 浮動小数点レジスタ 6 本, 整数の ALU 2 個, 乗算器 2 個, 浮動小数点の ALU 1 個, 乗算器 2 個, 除算器 1 個が必要であることがわかる (図 7参照)。

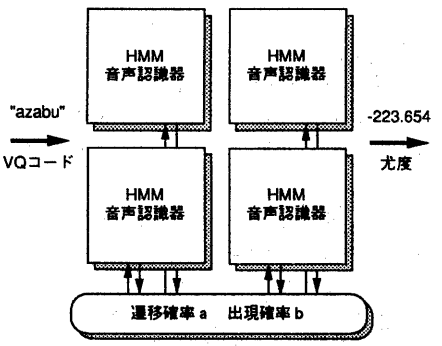

図 8: HMM 音声認識のシステムイメージ

4.2.4 システムイメージ

最終的には設計されたシステムの情報をハードウェア記述言語として出力するが、現時点では完成していない。そこで、ここでは 4 個のプロセッサ部を持つように設計されたシステムイメージを図 8に図示し、実現された場合の処理概要について解説する。

まず、システムに認識すべき単語の VQ コードが入力されると、HMM 音声認識部が認識すべき単語とあらかじめ格納された単語との尤度を計算する。ここで各音声認識部は次々と異なる単語との尤度を計算するために、完全に並列に認識処理を実行されることになる。最終的には、各音声処理部が計算した尤度の中で、最大の尤度を持つ単語が認識結果ということになる。

従来の高位合成においては、このような粗粒度の並列性を抽出して分散アーキテクチャに割り付ける最適化は行なわれないが、本設計法では記述に改良を加えてこれを行なうので、信号処理アルゴリズムのようにリアルタイム性を要求される応用に対して非常に効果的であることがわかる。特に HMM 音声認識は、適用例の中でも非常に成功した例である。

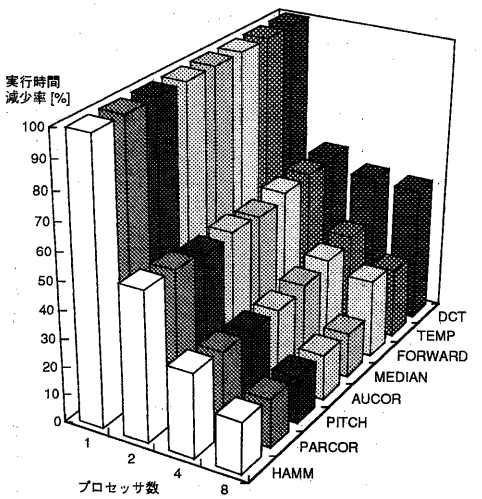

図 9: 実行時間減少率とプロセッサ数の関係

5 評価

前章で述べた 8 種類の信号処理アルゴリズム (図 5 参照) に対して、本システムを適用して分散合成の効果, プロセッサ最適化手法の効果, 実現に必要なハードウェア資源に関する評価を行なった。以下の節ではこれらの結果を提示して、結果に関する考察を行なう。

5.1 分散合成の効果

信号処理アルゴリズムに関してプロセッサ数 (1, 2, 4, 8) の場合について分散合成の効果を調査した。プロセッサ最適化を行なう以前の単一プロセッサによる実行ステップ数を 100%として、それ以外のプロセッサ数の実行ステップを換算した場合の実行時間減少率とプロセッサ数の関係を図 9に示す。

ここでは、アルゴリズムが本設計法に比較的適していたために、概ね理想的な結果が得られた。しかしながら、離散コサイン変換 (DCT) に関してはバリア同期が頻繁に利用されるために、プロセッサ数を増加させても期待するほどの効果が得られていない。さらに、HMM 音声認識 (FORWARD) やテンプレートマッチング (TEMP) に関しても、共有記憶を保護するスピンロック機構の影響で分散処理の効果は飽和している。

5.2 プロセッサ最適化の効果

次に、信号処理アルゴリズムに対して、最適化前と最適化後の実行ステップ数を調査して、最適化効果 (最適化後/最適化前 × 100) を算出した結果を図 10に示す。

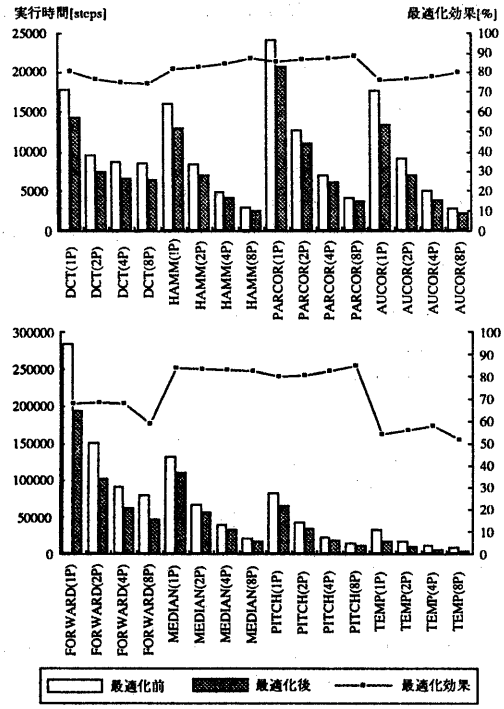

図 10: 信号処理アルゴリズムにおける最適化効果

結果としては、どの適用例に関しても概ね 8 割程度の最適化効果がみられた。特に、HMM 音声認識 (FORWARD) とテンプレートマッチング (TEMP) に関しては、アルゴリズムが細粒度の並列性を多量に含んでいるため、それぞれ 5〜6 割程度の最適化効果がみられた。

この結果からわかるように信号処理アルゴリズムに含まれる並列性は一般にそれほど多量ではないために、従来の高位合成が行なう細粒度の並列性を利用した最適化は、最大でも高々5 割程度の効果しか持たない。したがって、この評価からも信号処理アルゴリズムなどにおいては分散合成がいかに重要であるかが分かる。

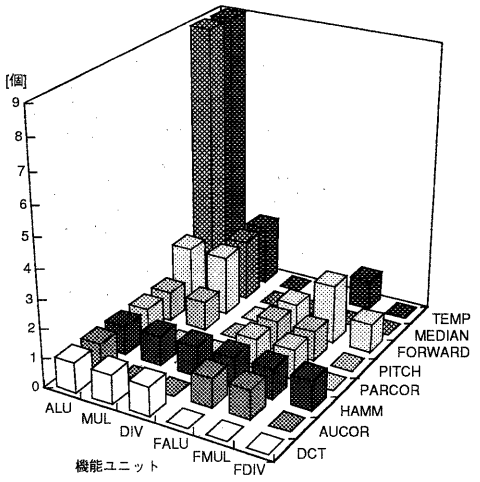

図 11: 単一プロセッサ当たり必要な機能ユニット数

5.3 ハードウェア資源に関する評価

さらに、単一プロセッサ当たり実現に必要な機能ユニット数を図 11に示す。ここで、MUL, DIV はそれぞれ乗算器, 除算器を、さらに先頭に F が付加しているものは浮動小数点用であることを示す。また、この評価においては加減算を ALU に割り付けたが、割付情報を変更することにより任意の演算を任意の機能ユニットに割り付けることが可能である。

さて、結果としては各機能ユニットとも概ね 1, 2 個必要になるが、画像アルゴリズム 2 例 (MEDIAN, TEMP) については ALU が 9 個必要という結果になった。これは、スケジューリング戦略として ASAP スケジューリングを利用したためであり、結果として 9 画素の演算を並列に行なうために 9 個の ALU が必要になる。

次に、単一プロセッサ当たり必要なレジスタ数を図 12に示す。ここでは、整数レジスタと浮動小数点レジスタのそれぞれについてレジスタ割付を行なったが、画像アルゴリズム 2 例 (MEDIAN, TEMP) や離散コサイン変換 (DCT) や HMM 音声認識 (FORWARD) もが大量の整数レジスタを必要とするという結果になった。これは、分散合成時に大域変数が大量に発生するが、本実装においては大域変数を単純に 1 レジスタに割り付けるので、それにともないレジスタ数が増大するため

である。この問題を解決するためには大域的なレジスタ割付が必要であり、今後考慮すべき問題であろう。

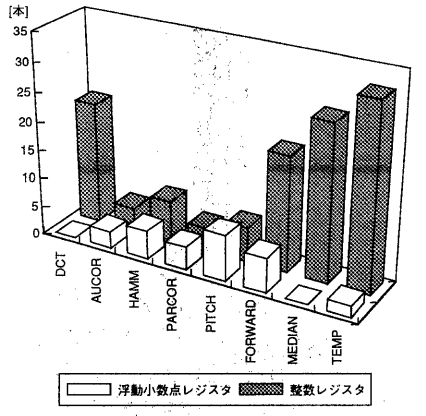

図 12: 単一プロセッサ当たり必要なレジスタ数

5.4 最適なアルゴリズム

本評価では、適用例として本システムに最適なアルゴリズムを取り上げたために、比較的良好な結果となったが、すべての信号処理アルゴリズムに関して本評価で示したような結果になるとは限らない。以下では、分散合成がどのような信号処理アルゴリズムに対して良好に動作するかについて述べる。

- 各処理部の処理内容が同質 … 分散合成において各処理部は同質に合成されるため記述の段階で同質になるように記述する必要がある
- 共有記憶とのデータ転送が少量 … 共有記憶は同時に 1 個の処理部しか操作できないため共有記憶とのデータ転送はなるべく少量である方が良い
- 単一プロセッサが実行する計算量が膨大 … 各処理部の実行する計算量がある程度多量でないと分散合成を行なう意味がない

以上のような条件に当てはまるアルゴリズムにおいて分散合成は成功するということができる。このように考えると、将来的には入力として動作記述が与えられると、その記述に最適なアーキテクチャを選択して、そのアーキテクチャに割付を行なう設計法が必要になることが予測される。また、このためには協調設計や分散合成以外にも多様な設計法が提案され、それを選択する目的で動作記述の特徴を抽出する技術が必要になる。

6 むすび

本稿では、設計支援システム SYARDS に分散合成という設計法を導入して、さまざまな信号処理アルゴリズムに関して設計を行ない、分散合成やプロセッサ最適化手法の効果について調査した。この調査を通じて、新たに導入した分散合成が HMM 音声認識等の信号処理アルゴリズムに対しては極めて有効に動作し、従来からの最適化手法と組み合わせることにより、システムをより高速に動作させることが可能であることが示された。

また、将来的には多様な選択肢から適切なアーキテクチャを選択してシステムの構造的な設計支援を行なうシステムが必要であることを指摘したが、そのひとつの選択肢として分散合成が有効であることが確認された。

参考文献

[1] 池永 剛, 白井 克彦: "高級言語により記述されたアルゴリズムを実現する専用プロセッサ設計支援システム," 情報処理学会論文誌, vol.32, pp.1445–1456 (1991).

[2] Hironobu Kitabatake, Katsuhiko Shirai: "Functional Design of a Special Purpose Processor Based on High Level Specification Description," IEICE Trans. Fundamentals, vol.E75-A, no.10 (1992).

[3] 雨坪 考尚, 上田 穣, 吉田 裕, 白井 克彦: "ビット幅を考慮した大規模システム処理系の設計手法について," 情報処理学会設計自動化研究会,67–2 (1993-6)

[4] Steven Brawer: "Introduction to Parallel Programming," Academic Press, Inc. (1989).

[5] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin: "HIGH-LEVEL SYNTHESIS," Kluwer Academic Publishers (1992).

[6] 中村 行宏, 小栗 清, 野村 亮: "RTL 動作記述言語 SFL," 信学論 (A), J72–A, 10, pp.1579–1593 (1989-10).

[7] 小林 哲則: "隠れマルコフモデルに基づく音声認識," 電気学会論文誌 (電子・情報・システム部門), Vol. 113-C, No.5 (1993-5).