社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－ＳＬＤＭ－112 (29)
2003／11／28

能動関数モジュールを再配置可能なアクティブソフトウェア向けアーキテクチャの提案

伴野 充† 中西 正樹† 山下 茂† 渡邉 勝正†

†奈良先端科学技術大学院大学 〒630-0101 奈良県生駒市高山町 8916 番地の 5

E-mail: †{mitsu-to, m-naka, ger, watanabe}@is.aist-nara.ac.jp

あらまし アクティブソフトウェアとは自分の計算状況を監視して，自分の状態や計算手順を調整する機能をもつソフトウェアである．アクティブソフトウェアは能動関数と呼ばれる内部に起動条件をもつ関数で記述され，自身の起動条件が成立すると起動する．そのため，アクティブソフトウェアでは能動関数の起動条件の判定，能動関数の起動・実行といったことが並列かつ頻繁に行われるという性質をもっている．そこで，本論文では起動条件の並列判定，能動関数モジュールの再配置処理により，効率的に能動関数を処理するためのアクティブソフトウェア向けアーキテクチャを提案する．

キーワード アクティブソフトウェア，能動関数，再構成可能アーキテクチャ

Architecture for active software that can rearrange active functions

Mitsuru TOMONO† Masaki NAKANISHI†

Shigeru YAMASHITA† and Katsumasa WATANABE†

† Graduate School of Information Science, Nara Institute of Science and Technology

8916-5 Takayama, Ikoma-shi, Nara, 630-0101 Japan

E-mail: †{mitsu-to, m-naka, ger, watanabe}@is.aist-nara.ac.jp

Abstract Active Software is software that has abilities to observe its own computational situations and to regulate its own state and its computational procedures. It is described with Active Functions that have Activation Condition in them. They are activated when their Activation Conditions are satisfied. Therefore, Active Software has nature that judgment of Activation Condition of Active Functions and Activation and Execution of Active Functions take place in parallel and frequently. Hence, we propose architecture for Active Software that can process Active Functions with the use of parallel judgment and rearrangement process of Activation Condition.

Keyword Active Software, Active Function, Reconfigurable Architecture

1. はじめに

近年，ソフトウェアの大規模化，複雑化に伴い，より頑健で安全なソフトウェアが求められるようになってきている．そのような，頑健な(robust)ソフトウェアを構築するための方法としてアクティブソフトウェア[1],[2]が提案されている．

多くの場合，プログラムは関数の集まりとして構築され，実行の形式は関数を陽に呼び出す形となっている．しかし，アクティブソフトウェアは計算の状況に応じて自立的な振る舞いをするソフトウェアである．つまり，何かの条件が成立したり，事象が発生したりしたときに，対応する関数を実行する構成になっている．

通常アクティブソフトウェアは，C プログラムに変換することによって実行される．しかし，アクティブソフトウェアでは条件の判定，関数の起動・実行といったことが並列に行われるため，C プログラムによる逐次実行よりも，効率よく起動条件を評価する実行形式が望ましい．そこで，本論文ではアクティブソフトウェアをハードウェアで効率よく実行するためのアーキテクチャを提案する．

関連研究として，ある特定の言語専用のハードウェアとしては，Lisp マシン[3]，Prolog マシン[4]，また，Java Processor[5]等が挙げられる．これらの研究では，命令系列の工夫によって高速化を図るというアプローチだが，提案する手法はアクティブソフトウェアに内在する並列性に着目して処理の効率化を図っている．

また，近年，回路の構成を変えることにより，一つ

の回路で様々な機能を実現することが出来る再構成可能ハードウェア[6]の研究が盛んに行われている．例えば，仮想パイプラインを実現した PipeRench[7]や，多数のプロセッサエレメントを配列上に配置し，コンテクスト切り替えによって様々な機能を実現する Dynamically Reconfigurable Processor (DRP)，また，再構成可能な演算器マトリックスをもつ DAP/DNA[8]等が挙げられる．

提案手法では，能動関数の条件判定機構に再構成可能ハードウェアを用いることで柔軟かつ並列性の高い条件判定機構を実現している．

以下では2章でアクティブソフトウェアの概要を説明し，3章で提案するアーキテクチャについて述べる．最後にまとめと今後の課題を述べる．

2. アクティブソフトウェア

2.1. アクティブソフトウェアの特徴

アクティブソフトウェアとは，ソフトウェアが「強靭さ(robustness)と複雑さの管理に能動的な責任」をもち，次の特徴を備えているものであるとしている，

1．自分の状況を知る(self awareness)
2．自分を調整できる(self regulating)
3．自分の誤りを訂正する(self correcting)
4．遊牧的に動き回る(nomadic or mobility)

これらは，エージェントプログラムを始めとする自立性をもつソフトウェアの基本となる共通の特徴であり，つまり「自分の計算状況を監視して，自分の状態や計算手順を調整する」機能をもつソフトウェアである．

2.2. アクティブソフトウェアの動作

ここではアクティブソフトウェアの動作を簡単な例を用いて説明する．アクティブソフトウェアは能動関数[9]と呼ばれる，自立的に起動する関数で記述される．能動関数は，それぞれが自身の起動を判断するための起動条件を内部にもっている．能動関数の起動は起動文と呼ばれる構文によって指示され，起動文は以下のようになっている．

@p 変数名(引数);

起動文において@は能動関数をこの時点において起動するという意味を表し，p は並列に能動関数を起動することを表す．一つだけ能動関数を起動する場合は@1になり，順次起動する場合は@2となる．その後に続く変数名は，この変数を起動条件にもつ能動関数の起動が判定されることを意味する．

例えば図 1 において，main プログラム中の起動文@p a( );においてもしaが70であれば，図1の能動関数の中で(2)と(3)が条件に合致するのでこの二つの能動関数が起動され，両方の能動関数の処理が終わると制御が main プログラムに戻る．同様に，@p b( );においてbが100ならば，図1の(5)と(6)が条件に合致するのでこれらの能動関数が起動され，能動関数の終了の同期をとって main プログラムが再開される．

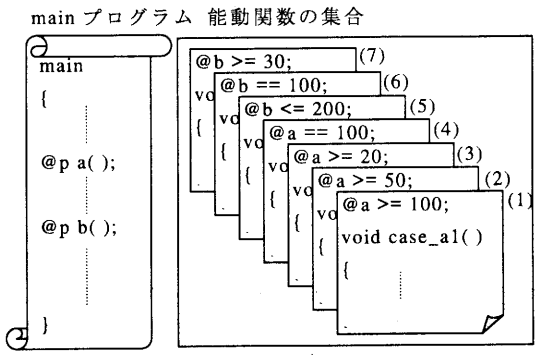

図 1 能動関数

3. アーキテクチャ

3.1. アーキテクチャの概要

本節では提案するアーキテクチャについて説明する．ブロック図は図2のようになっており，大きく分けてメインプロセッサ，条件判定用変数レジスタ，条件判定部，インストラクションメモリ，能動関数コントローラ，プロセッサエレメントアレイ，データキャッシュ，データキャッシュコントローラ，データメモリの9つのモジュールから構成される．

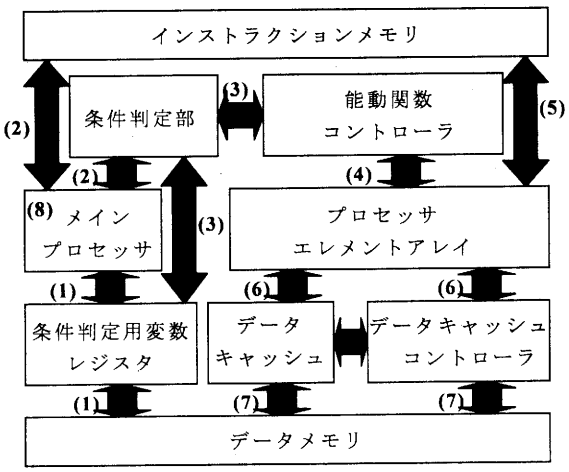

図 2 ブロック図

処理の流れは図3のようになっている．メインプロセッサはアクティブソフトウェアの main 関数の処理を行い，起動文に先立って必要な変数を条件判定用変

数レジスタに読み出す(1).

起動文において，条件判定部の再構成を行いメインプロセッサをストールする(2).

条件判定部は条件判定用レジスタの値を参照してどの能動関数が起動するかを判断する(3).

条件判定部からの出力を受け取った能動関数コントローラは，各能動関数の処理をプロセッサエレメントに割り振る(4).

プロセッサエレメントアレイは能動関数を処理するためのプロセッサを n 個搭載したモジュールである．能動関数の処理を割り振られた各プロセッサエレメントはインストラクションメモリから能動関数を読み出して処理を開始する(5).

アクティブソフトウェアの仕様上データの取り扱いには注意を要する．能動関数は起動された時点でのデータを参照するため，能動関数が任意にグローバル変数および条件判定用変数を書き替えることは許されない．これらのデータのストアが必要な場合は，データキャッシュとデータキャッシュコントローラに一時的に書き込みを行う(6).

そして，能動関数の終了の同期を取りデータをデータメモリに書き込む(7).

データの書き込みが完了次第，main 関数の処理を再開する(8).

なお，図 2 中の数字は図 3 の対応するステップにおけるデータの流れを示している．

以下の節では，条件判定部における条件判定機構，能動関数コントローラによる処理の割り振り，プロセッサエレメントアレイの能動関数へのアクセス，データキャッシュコントローラの機構をそれぞれ詳細に説明する．

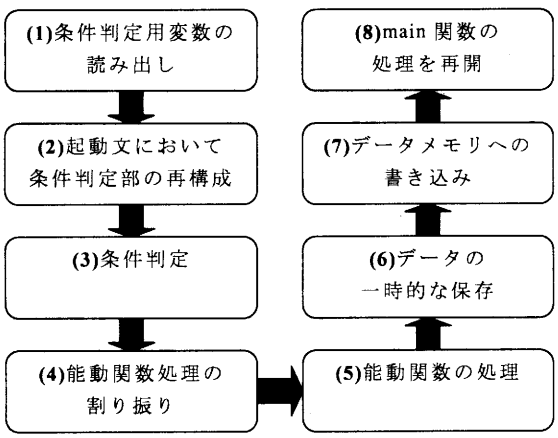

図 3 処理の流れ

3.2.条件判定機構

条件判定部を図 4 に示す．条件判定部は再構成可能ハードウェアで構成され，起動文ごとにその構成を変える．メインプロセッサからの起動文は構成情報になっており，起動文をデコードすることによって構成を決定する．条件判定用変数を参照して，どの能動関数を起動するかを判定し出力する．一度に参照できる変数には限りがあるため，コンパイル時に条件文の分割を行うことで対応する．

条件判定出力には起動すべき能動関数の ID が出力され，その際起動する能動関数 ID の最上位 bit は 1 に，起動する必要のない能動関数 ID の最上位 bit は 0 になっている．

条件判定出力以下の部分では，出力された能動関数 ID を木状に比較を行う．図 5 に木状の比較器の一部を示す．各木の頂点には能動関数の ID を格納するためのレジスタが用意してある．能動関数の ID の最上位 bit を比較し，1 になっている能動関数の ID を次へ伝播させる．比較の際，両方の ID の最上位 bit が 1 であった場合は能動関数 ID の若い方を優先させる．順次比較を行い，最終的に出力された ID をキューレジスタへ格納し能動関数コントローラへと渡す．

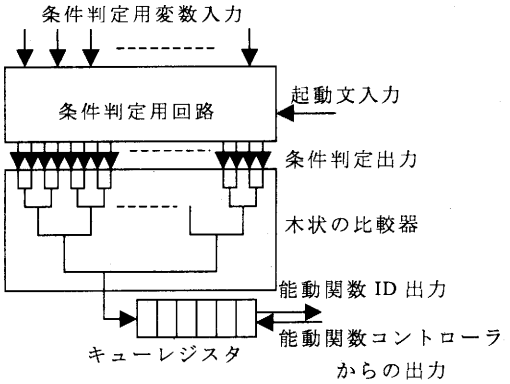

図 4 条件判定部

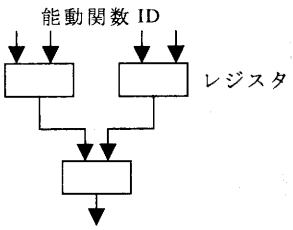

図 5 木状の比較器の一部

3.3.能動関数コントローラ

能動関数コントローラとプロセッサエレメントの関係を図5に示す．各プロセッサエレメントは自身の状態を表す状態レジスタと，処理する能動関数の IDを保持する ID レジスタを内部に持っている．

能動関数コントローラは，状態レジスタを参照してプロセッサエレメントがアイドル状態であれば，キューレジスタの先頭にある能動関数の ID を渡して処理を開始させる．その後，条件判定部へ出力を返し，それを受け取った条件判定部はキューレジスタに次に起動すべき能動関数の ID を追加する．

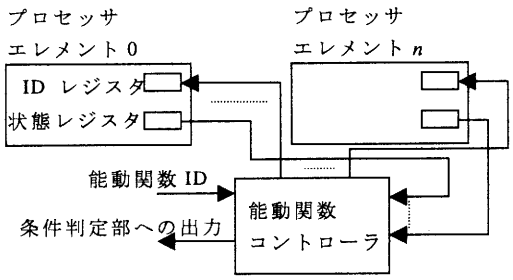

図5 能動関数コントローラと
プロセッサエレメント

3.4.能動関数へのアクセス

能動関数コントローラより、処理する能動関数の IDを受け取ったプロセッサエレメントが、どのように能動関数の命令系列へアクセスするかを説明する．

図6はインストラクションメモリの構造を表す．図中の act は個々の能動関数の命令系列を現す．インストラクションメモリ中に，能動関数の ID とその能動関数のコードが格納してあるアドレスを示す一覧表を保持している．各プロセッサエレメントはそれぞれが同様のインストラクションメモリをもっており，能動関数の ID を受け取ったプロセッサエレメントは一覧表からその能動関数のコードが格納されているアドレスを算出し，命令コードを読み出す．

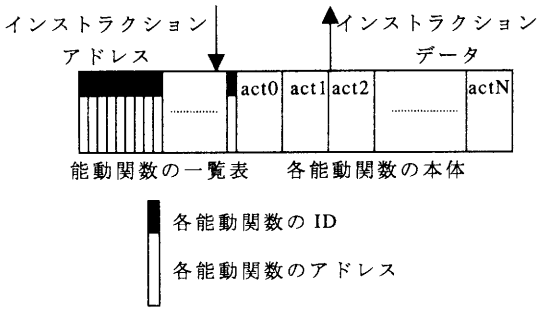

図6 能動関数へのアクセス

3.5.データキャッシュコントローラの機構

各プロセッサエレメントは，ローカル変数用データメモリとグローバル変数および条件判定用変数用のデータキャッシュを保持している．データキャッシュコントローラはグローバル変数および条件判定用変数のロードとストアの際の制御を行う。

これらの変数のロード時に、キャッシュミスが起きた場合，プロセッサエレメントはストールし，データキャッシュコントローラによってキャッシュが書き替えられるのを待ち，データにアクセスする．

ストアの際は，アクティブソフトウェアの仕様により，すべての能動関数の処理が終了してからデータの更新を行う必要がある．このとき，複数の能動関数が同一の変数の値を更新する場合に，どの能動関数が優先されるかは処理系によるとされている．そのため，プロセッサエレメントはこれらの変数のデータストア時に，データキャッシュコントローラ中にあるデータストア用のキューレジスタへ一旦書き込みすべき内容を保存しておく．そして，すべてのプロセッサエレメントがアイドル状態になると，データキャッシュコントローラは能動関数の処理が終了したと判断して，キューレジスタの内容をデータメモリに書き込む．

キューレジスタの内容の書き込みが完了すると，それをメインプロセッサに伝え，それを受け取ったメインプロセッサは起動文の次の命令から処理を再開する．

4. まとめと今後の課題

本論文では，アクティブソフトウェアを効率的に処理するための専用ハードウェアのアーキテクチャを提案した．提案アーキテクチャでは，再構成可能な論理素子を用いた条件判定部により，アプリケーション毎に最適化された形で，並列に条件判定をすることが出来る．また，各プロセッサエレメントがそれぞれインストラクションメモリをもち，能動関数に即座にアクセスすることが可能である．それにより，複数のプロセッサエレメントによる能動関数の同時実行を可能とし，処理の効率化を狙っている．

今後の課題は本アーキテクチャの Verilog HDL による実装を行い性能と面積を評価することである．

また，「能動関数の配列」と呼ばれる，異なるデータに対して同時に同一の処理を行う能動関数がある．本アーキテクチャでは，能動関数の配列への対応が不十分であると考えられる．そこで，能動関数の配列にも対応できるようにアーキテクチャの拡張を行う．

また，能動関数の配列では複数の能動関数が同時にデータにアクセスするため，データロードがボトルネックとなる可能性がある．本アーキテクチャも含めて，能動関数処理のためのより効率的なデータアクセス機

構を提案する．

謝　辞

本研究は，一部，日本学術振興会科学研究費補助金(課題番号：15500023) ならびに大川情報通信基金研究助成による．

文　献

[1] Robert Laddage, "Active Software, " In Self-Adaptive Software, First Internatinal Workshop, IWSAS 2000, LNCS 1936, pp.11-19, Spronger 2000.

[2] 渡辺勝正，小林郁典，木村晋二，堀山貴史，中西正樹，"アクティブソフトウェアの構成と表現について，" 2001 日本ソフトウェア科学会第 18 回大会，論文集，4D-2.

[3] 服部彰，"Lisp マシン，" 1987 人工知能学会誌，Vol.2, No.4, 特集「AI マシン」．

[4] 横田実，"逐次型 Prolog マシン，" 1987 人工知能学会誌，Vol.2, No.4, 特集「AI マシン」．

[5] Shinji Kimura, Hiroyuki Kida, Kazuyoshi Takagi, Tatsumori Abematsu, Katsumasa Watanabe, "An Application Specific Java Processor with Reconfigurabilities," IEEE 2000.

[6] Ranga R. Vemuri, Randolph E. Harr, "Configurable Computing: Technology and Applications," IEEE Computer, pp.39-40, April 2000.

[7] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, R. Reed Taylor, "PipeRench: A Reconfigurable Architecture and Compiler," IEEE Computer, pp.39-40, April 2000.

[8] 佐藤友美，"10ns で演算器間の構成を書き替えるダイナミック・リコンフィギュアラブル技術を開発，" 日経エレクトロニクス，2003 年 1 月 6 日号，pp.111-122, Jan.2003.

[9] 渡辺勝正，木村晋二，相良かおる，高木一義，"能動形プログラミング Active Software，" In PLL'99, 日本ソフトウェア科学会第，1999, pp.91-100.