大域ディペンダブル情報基盤プロジェクト

～ディペンダブルアーキテクチャグループ　田中・坂井研究室～

田中 英彦　坂井 修一

情報理工学系研究科電子情報学専攻

あらまし　コンピュータシステムにとって処理速度･消費電力とともに重要なことがディペンダビリティである。ここではマイクロプロセッサレベルでのディペンダビリティを対象として、システムＬＳＩに再構成可能ユニットを組み込み、ＣＰＵと機能分散することで、性能とディペンダビリティの両面を向上させるプロセッサアーキテクチャ（コデザインを含む）の研究を行う。その初年度として、再構成可能ユニットの基本構成と処理のわりつけ方式の検討を行い、ベンチマークを実装してその初期的評価を行った。提案する方式を高性能・高信頼性が不可欠である応用分野である暗号、フィルター、画像圧縮の分野に適用しその有効性を部分的に検証することができた。

1. はじめに

コンピュータシステムにとって処理速度・消費電力とともに重要なことがディペンダビリティである。本研究ではマイクロプロセッサレベルでのディペンダビリティを対象とし、システムＬＳＩに再構成可能ユニットを組み込み、ＣＰＵと機能分散することで、性能とディペンダビリティの両面を向上させるプロセッサアーキテクチャ（コデザインを含む）の研究を行う。

本年度はその初年として、再構成ユニットの基本構成と処理のわりつけ方の検討を行い、ベンチマークを実装してその初期的評価を行った[8]。

2. 対象とするマイクロプロセッサ

ここでは、図１に示すような近未来のシステムＬＳＩを考える。本システムＬＳＩのアーキテクチャは、ＰＤＡからＰＣ，サーバまで広範な用途を考えている。

本アーキテクチャはチップマルチプロセッサと再構成可能ユニット（Reconfigurable Unit)から成り、PU 多重利用による性能とディペンダビリティの向上、ＲＵによるアプリケーション適応型のディペンダリティ提供とＰＵとの機能分散、ディペンダブル・共有メモリプロトコルの実現、アプリケーション適応コード生成、多重実行コード生成、段階的デグレーデーションなどがこの上で実現される。

ここでは、本ＬＳＩの中でＲＵについて、すなわち、再構成可能ユニットの内部構成法について検討する。

3. ＲＵにおける動的資源割り当ての目的

RU は、プログラム可能デバイス（Programmable Device, PD）[2][4]とそのコントローラなどで構成される。RU では回路書き換えを行いアプリケーションをハードウェアとして動作させるのでハードウェアに近い性能を実現でき、一方構成情報(Context)はデータとして所持するので、ソフトウェアのように書き換え、保守が容易である。さらに、動的な回路の書き換えを行うことによって RU の柔軟性を高めることができる。

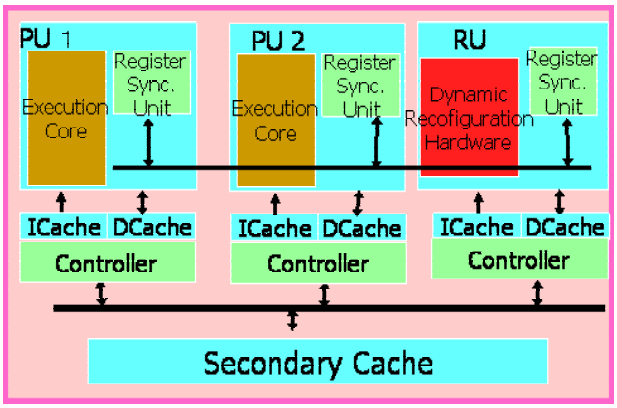

図１．ディペンダブル・システムＬＳＩ

例えば RU にハードウェアの規模を超えた大きな回路を割り当てたい場合、その回路を割り当て可能な大きさに分割し、分割したものを切り替えつつ実行する、時分割を行う[3][5]ことにより、RU を大きく見せることができる。

一方、短時間しか使わない回路を複数任意のタイミングで割り当てる場合、空間分割を行う[1]ことによって RU が複数存在するように見せることが可能である。また、空間分割には、PD の一部分に障害が発生しても該当する部分を切り離して使うことができるという、耐障害性という面での利点も存在する。

従来はハードウェア規模の上限が厳しく、時分割がハードウェア仮想化の主な手段であった。しかし、半導体集積度の向上、信頼性の要求などにより、時分割のみならず柔軟な空間分割が可能なアーキテクチャが求められるようになっている。

空間分割を行う際に重要となるのは、柔軟な割り当てを可能にする、自由度のある構成情報の生成である。一般に構成情報は、用いるハードウェア資源に論理回路を対応付ける、いわゆる配置配線を行うことによって生成する。ここで、構成情報に自由度を持たせるということは、配置配線の一部分を実行時に行うことに他ならない。しかも、実行時の配置配線時間は可能な限り短くしなければならない。

そこで本研究では、Configuration Block(CB)と呼ばれる粗粒度な書き換え単位で構成された PD を用い、CB に対応する塊をノ

ードとしたグラフで構成情報を与える。そして、この構成情報を、動的に PD に割り当て、実行を行うようなアーキテクチャを考える。

今年度の研究目的は、このアーキテクチャ上で、配置配線に要する時間を可能な限り削減しつつ、必要とする配線資源を(時間をかけた場合と比べて)許容できる範囲に抑えることである。

4. 動的資源割り当ての手法

回路の動的割り当てに必要とされる配置配線は、ASIC における回路設計、PD における静的配置配線[6]でも要求されるが、動的割り当てを行う際には時間を大幅に短縮しなければならない。回路の動作と並行して計算を行えば計算時間の隠蔽が可能であるが、それでも Simulated Annealing などといった手法を行うのは現実的でない。特に、割り当て要求が来てから割り当てるまでに要する時間はそのまま実行レイテンシとなってしまうため、これは可能な限り削減したい。

4.1 ＲＵアーキテクチャ

（１） ＰＤの構造

本研究で扱う PD(図 1)は、Logic 書き換えの最小単位となる Configuration Unit(CB)、近接した特定の CB 間を配線するための専用配線である Local Wire(LW)、PD 部分を縦あるいは横に貫き、自由に配線ができる Global Wire(GW)、縦と横の GW を接続するための Switch で構成されている。各部分は以下のようなモデルになっている。

CB 本研究では CB の内部については詳細を定めず、整数の乗算程度までをサポートする 8bit-ALU が 16 個含まれており、それらの間は自由に配線できると理想化した。

Local Wire 特定の CB 間のみを接続することが出来る配線資源である。本アーキテクチャでは、CB の位置を XY 座標で表現したときに座標の差が (x, y) = (1, 0), (2, 0), (3, 0), (0, 1), (0, 2), (0, 3)となるような CB の組に対して、LW が用意されている。各 CB 間の LW の本数は十分に多く、配置配線に影響が出ないと理想化した。

Global Wire 8 本(8bit)を単位として、各座標の x または y 方向に有限な数だけ用意されている。この GW は、縦方向の場合には x 座標が、横方向の場合には y 座標が対応する任意の CB に接続することができる。

Switch 縦 GW と横 GW を、任意の組み合わせで接続することができると理想化した。

（２） Module とその実行モデル

本研究では、RU に割り当てる機能を Module と呼ぶ。Module は、ネットリストの分割[7]を行うことによって、あらかじめ、CB にそのまま割り当てられる塊をノードとしたグラフになっている。各ノードの内部は静的に決定されているが、各ノードの位置関係や配線方法についてはこの段階では決まっていないため、動的に柔軟な割り当てを行うことが可能である。

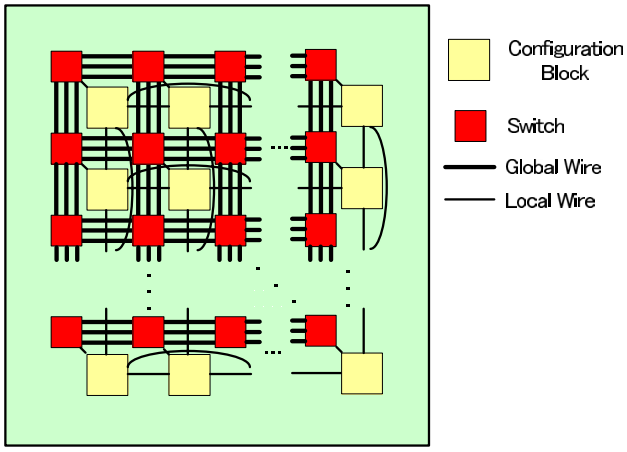

図２． ＰＤの構成

RU で Module を実行する際に必要となる動作を以下に示す。

1. 実行する Module の Context を RH に転送する。
2. Module が用いる CB、配線資源を決定する。
3. 該当する CB へ、該当する Context を転送する。
4. 実際にデータを流し込み、回路を動作させる。
5. 占有していた CB、配線資源を解放する。

本研究では、2. の配置手法と、4. と並行しての再配置を扱う。

（３）資源割り当て

資源割り当てとは、Module を PD の一部に割り当てることである。Context Controller は、Module の各ノードを CB に、各枝を LW あるいは GW に割り当てる。割り当てに失敗するケースは 2 つ存在する。1 つは、空き CB の数が足りない場合で、もう 1 つは CB 間を結ぶための GW が足りない場合である。前者は割り当ての手法の良し悪しに関わらない。一方、後者は割り当ての手法によって回避できる場合がある。そのため、以降では後者を左右する GW 数に着目する。

4.2 提案する資源割り当て手法

ここでは資源割り当ての手法として、No-Backtrack Placing による配置配線と、Force Directed アルゴリズム[6]を用いた再配置を提案する。前者は割り当て要求が来てから回路を実行するまでに必要とする時間の削減、後者は配置配線の計算にかかる時間を隠蔽しつつ、配線資源を抑えるために有効であると考えられる。

4.2.1 No-Backtrack による配置配線

No-Backtrack による配置場所決定の手順は、下のようにまとめられる。Backtrack を行わないので、配置配線にかかる時間は回路規模に比例した時間、つまり $O(n)$となる。

1. 割り当てを行う Module のうち、未配置のノードを 1 つ選ぶ。このノードに接続されるノードのうち、未配置のものの数を p とする。
2. このノードを以下のルールに従って空き CB に配置する。そして、既に配置されているノードと現在配置されたノ

ードとの間の配線を行う。

- 既に配置したノードと接続するために必要な GW の数が最も小さくなる場所を選ぶ。
- そのような候補が複数存在する場合には、各 CB に対して LW で接続できる先の CB のうち使われていないものの数 q を求め、$q \geqq p$ となるものを選ぶ。そのような CB が存在しない場合には、q が最も大きいものを選ぶ。
- 全てのノードの配置が終わるまで、2. を繰り返す。

上記の配置 CB 選択部分をハードウェアで実現するために必要となるのは、「各 CB に対して、ある接続すべき CB が与えられたときにそこへ接続するために
必要となる GW の数を求め、今までに計算した GW 必要数に加算する回路」である。このような回路を用いることによって、配置する CB を決めることができる(この動作を実現するハードウェアについては、文献[8]で述べた)。

4.2.2 Force Directed による配置配線

No-Backtrack による配置配線だけでは、他の時間をかける配置配線手法に比べ、余分に GW を消費してしまうことが考えられる。そこで、割り当てられているノードの場所を入れ替えることによって再配置を行う。この手法の一つである Force Directed による再配置の手法は、以下のようにまとめられる。

1. ノードが配置されている CB のうち、1 つを選ぶ。ここでは、ノード``X″に使用されたものを選んだとする。
2. 1. で選んだ CB を別の CB に移動したときに生じる、全体での使用 GW 数の増減を調べる。ただし、移動先の CB があるノード``Y″に使用されている場合、使用 GW 数の増減は、``X″を``Y″のあった場所に移動した場合の使用 GW 数の増減と、``Y″を``X″のあった場所に移動した場合の使用 GW 数の増減の和となる。
3. 使用 GW 数を最も減らすことができる移動先と、場所の入れ替えを行う。もし、入れ替えた後の再配線が不可能な場合には、元の場所を保つ。

5. 評価

本研究では、座標あたりの GW 数を変化させた場合の割り当て失敗率の変化、および、割り当て失敗率を抑えるために必要となる座標あたりの GW 数で評価を行った。座標あたりの GW 数とは、特定の座標の CB が、縦方向、横方向でそれぞれ用いることのできる GW の
数のことである。例えば図では座標あたりの GW 数は 3 本である。

5.1 評価環境

（1）評価ベンチマーク

本研究では、CB の使用率を特定の値付近で前後させつつ、ランダムな順番で割り当て、解放の要求を行うようなベンチマークを用意し、評価を行った。割り当て、解放を行う Module として用意したのは以下の 3 つである。

DCT(Discrete Cosine Transform) 1 次元、8 点の 8bit 値に対しての離散コサイン変換を行う。12 個のノードと 66 本の枝で構成される。

FIR(Finite Impulse Response filter) 16bit 値、8 次の FIR フィルタ回路である。9 個のノードと 37 本の枝で構成される。

IDEA(International Data Encryption Algorithm) IDEA 暗号化/復号化回路の 1 ステージである。6 個のノードと 20 本の枝で構成される。

これらを等確率に選び、割り当て/解放を行った。どちらも、全 CB のうち使われている CB の率の時間平均は約 86%となった。

（2）評価対象

次の 3 つの方式を評価対象とすることとした。

① **No-Backtrack による配置**
② **Force-Directed による配置**
③ **Simulated Annealing による配置**

5.2 実験および結果

5.2.1 再配置を行わない場合に必要となる座標あたりの GW 数

No-Backtrack, Force Directed, Simulated Annealing の各配置手法について、座標あたりの GW 数を変化させた場合の割り当て失敗率を調べた。Force Directed については、繰り返し回数の上限を 1 回、2 回、4 回、8 回の 4 通りに設定して、測定を行った。その結果を図 3 に示す。1%程度の割り当て失敗を許容した場合、Simulated Annealing では 10 本、Force Directed では 18 本、No-Backtrack では 22 本の GW が必要となった。
つまり、最善の場合に比べ、倍程度の GW が必要であることがわかる。また、Force Directed に関しては、4 回以上の繰り返しはほとんど意味がないこともわかる。

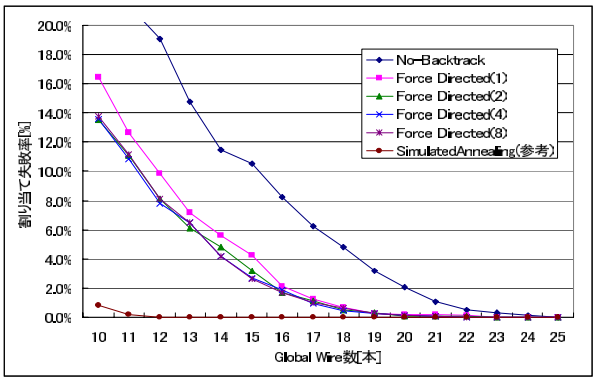

図 3．座標あたりの GW 数と失敗率の相関- Force Directed(x) の x は繰り返し回数の上限

5.2.2 割り当てにかかる時間

No-Backtrack による割り当て、Force Directed による割り当てそれぞれについて、割り当てに要した時間の概算と、割り当て失敗を 1%以下に抑えるために必要となる座標あたりの GW 数の関係を調べた。

計算にかかる時間は、No-Backtrack による割り当てでは 1 回

の割り当てあたりに行われた CB 選択回路への座標入力の平均回数、つまり、Module 内の枝の数を用いた。また、Force Directed では、ノードに接続されている枝の最大数と Module 内ノード数と繰り返し回数の積を計算に要したサイクル数とした。

その結果を図 4 に示す。Force Directed による配置では、繰り返し回数を 1 回に抑えてもおよそ 3 倍の時間がかかってしまう。一方、移動が行われず、繰り返し回数の上限まで達せずに終了するケースが増えるため、上限を 4 回にしても計算時間はそれほど増えないことが分かる。

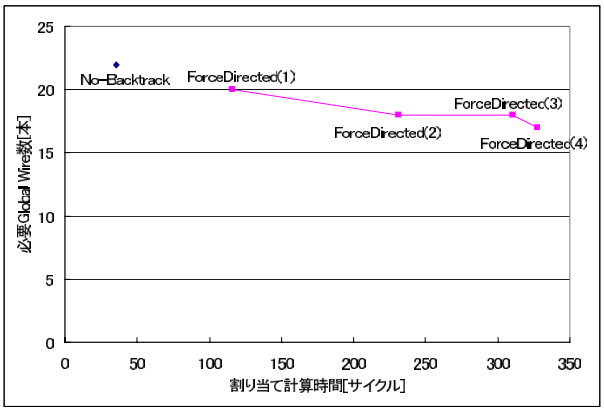

図 4．割り当てに要する時間と失敗を 1%以下に抑えるために必要な座標あたりの GW 数の関係

5.2.3 再配置の効果

再配置による効果を調べるために、再配置を行った場合について、割り当て失敗を 1%以下に抑えるために必要となる座標あたりの GW 数の変化を調べた。この実験に際し、再配置の実行は各割り当て/解放要求の間にn回だけ行えると仮定し、n= 0, 1, 2, 4, 8, 16, 32, 64, 128 と変化させて実験を行った。

その結果を図 5 に示す。8 回程度の再配置が行えるのであれば、割り当てに用いる手法の良し悪しをほぼ隠蔽できるということがわかる。ここで、再配置回数 8 というのは 8 つのノードで構成された Module を Force Directed(繰り返し上限 1 回)で割り当てた場合とほぼ同じである。つまり、図 4 からわかるように、No-Backtrack による割り当ての 3 倍程度の時間が各割り当て/解放の間に存在すれば、再配置による効果を得られるということになる。

6． まとめ

今年度の目的は、動的な資源割り当てを行うような RU において、割り当てに要する時間を可能な限り削減しつつ、時間をかけた場合の品質に近づけることであった。そのための割り当て手法として No-Backtrack による配置を提案し、時間を十分にかけた場合の倍程度の配線資源で同程度の性能が得られることを示した。さらに、一度割り当てを行った後に回路の動作と並行して再配置を行うことにより、割り当て時間を短く保ちつつ必要となる配線資源を再配置を行わない場合の 3/4 程度に削減できることを示した。

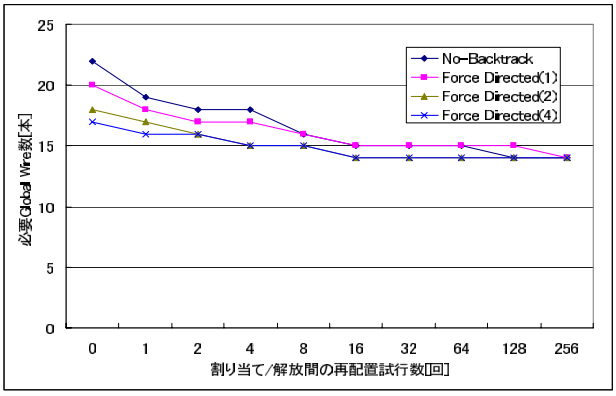

図 5．再配置の回数と、割り当て失敗を 1%に抑えるための座標あたりの GW 数の関係

今後は、本研究では詳しく検討しなかった、各機能をハードウェアで実装した場合の面積や所要時間などといった評価を行う必要がある。特に、再配置の際に入れ替える組を決定する機構について詳細を検討する必要がある。一方、実行・制御を行うモデルについても、具体的に検討されなければならない。そのためには、プロセッサがソフトウェアの実行を行いつつ、必要に応じて RU に指示を出すような実行機構、コンパイラ、およびそれらの評価環境を設計・試作することが課題となる。

参考文献

[1] Eylon Caspi, Michael Chu, Randy Huang, Joseph Yeh, John Wawzynek and Andre Dehon, Stream Computations Organized for Reconfigurable Execution (SCORE), *Field Programming Logic and Applications*, pp.605-614, 2000.

[2] Katherine Compton and Scott Hauck, Reconfigurable Computing: A Survey of Systems and Software, *ACM Computing Surveys*, vol.34, No.2, pp.171-210, 2002.

[3] Seth Copen Goldstein, Herman Schmit, Mihai Bidiu, Srihari Cadambi, Matt Moe and R. Reed Taylor, PipeRench: A Reconfigurable Architecture and Compiler, *Computer*, Vol.33, No.4, pp.70-77, 2000.

[4] Yuichiro Shibata, Hideharu Amano and Masaki Uno, Reconfigurable Systems: New Activities in Asia, *Field Programming Logic and Applications*, pp.585-594, Aug.2000.

[5] X.-P. Ling, Y. Shibata, H. Miyazaki, H. Amano and K. Higure, Total System Image of the Reconfigurable Machine WASMII, *International Conference on Parallel Distributed Processing Techniques and Applications*, pp.1092-1096, June 1997.

[6] K. Shahookar and P. Mazumder, VLSI Cell Placement Techniques, *ACM Computing Surveys*, Vol.23, No.2, pp.143-220, June 1991.

[7] 上村明、リコンフィギャラブルコンピューティングにおける回路分割機構の研究、修士論文、東京大学大学院情報理工学系研究科、2003.

[8] 高田正法、再構成可能ハードウェアにおける動的資源割り当て手法の提案及び評価、卒業論文、東京大学工学部電子情報工学科、2003.