2003－ARC－153 (3)
2003／5／8

空間分割を行う再構成可能ハードウェアにおける動的資源割り当て

高田 正法* 上村 明* 森 晃平* 岡部 淳* 坂井 修一* 田中 英彦*

概要

Reconfigurable Computing において、時分割及び空間分割による Programmable Device の仮想化は、柔軟性や信頼性の向上に不可欠である。そのためには、空き資源に対し動的に回路を割り当てるようなコントローラが必要となる。このコントローラに要求されるのは、割り当て要求に対し十分に短い時間で割り当て、言い換えれば配置配線を完了させることである。そこで我々は、粗粒度な単位で構成された部分再構成が可能なハードウェアに対し、バックトラックを行わない $O(n)$ のアルゴリズムによる配置、及び Force Directed アルゴリズムを用いた再配置を提案する。本稿ではこれらのアルゴリズムについて、計算時間、及び必要となる配線資源を、シミュレータを用いて Simulated Annealing や既存の Force Directed アルゴリズムによる配置と比較し、検討を行った。

Dynamic Resource Allocation on Space Division Reconfigurable Hardware

Masanori Takada† Akira Uemura† Kohei Mori†
Jun Okabe† Shuichi Sakai† Hidehiko Tanaka†

Abstract

In Reconfigurable Computing, virtualization of Programmable Device using time and space division is important to improve computation's flexibility and reliability. To realize this virtualization under time restriction, a hardware controller which dynamically allocate and assign available computing resources is needed. In this paper, we propose a non-backtracking allocation algorithm for programmable device whose configuration block granularity is relatively large. We also propose Force Directed algorithm for resource relocation. The algorithms are examined in terms of computation time and required resources, and a comparison is described by replacing the non-backtrack allocation algorithm with a conventional Force Directed algorithm or Simulated Annealing.

1 はじめに

近年、回路を書き換えることのできるハードウェアである Programmable Device(PD) の性能向上に伴い、Reconfigurable Computing(RC) と呼ばれる技術が注目を集めている [3]。その中でも、動的な回路の書き換えは、RC の柔軟性を高める重要な要素である。例えば RC に用いられる再構成可能ハードウェア (Reconfigurable Hardware, RH) 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、時分割を行うことにより、RH を大きく見せることができる [4, 6]。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、空間分割を行うことによってある機能の一部分の書き換えや、複数の機能単位の同時割り当てが可能である [1, 5]。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、規模を大きく見せる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、信頼性を上げる多重実行への対応、デバイスの耐障害性向上などといった要求から、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、自由度を残した構成情報の生成である [2]。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。つまり、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。しかし、実行時の配置配線時間は可能な限り削減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想定する。

本研究の目的は、このアーキテクチャ上で、配置配

*東京大学大学院情報理工学系研究科
†Graduate School of Information Science and Technology, The University of Tokyo

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。

2 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 [8]、PD における静的配置配線でも要求されるが、動的割り当てを行う際には計算時間の大幅な短縮が必要となる。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 などといった手法の適用は難しい。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。

そのため本研究では、質では劣るものの十分に短い時間で配置、配線を行い、その後回路の動作と平行して再配置、再配線を行う手法を提案する。

2.1 想定するアーキテクチャ

2.1.1 Programmable Device(PD) の構造

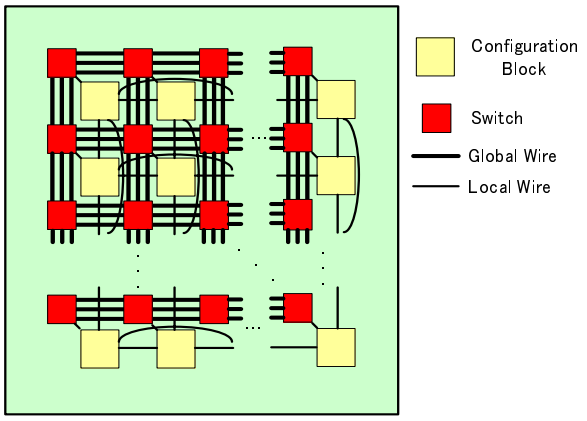

図 1: PD の内部

本研究で扱う PD(図 1) は、Logic 書き換えの最小単位となる 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。

Configuration Block(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(LW) 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

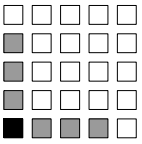

図 2: LW で配線できる CB - 黒の CB は上下左右方向に 3 つ離れた CB までの接続が可能である。

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 (図 2)。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(GW) 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。

2.1.2 Module とその実行モデル

本研究では、RH に割り当てる機能を Module と呼ぶ。Module は、ネットリストの分割を行うことによって、あらかじめ、例えば図 3 のような、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。

RH 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。

2.1.3 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 (図 4)。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。

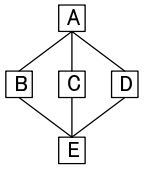

図 3: 配置する Module の例。ノードが CB に、枝が Wire に割り当てられる。

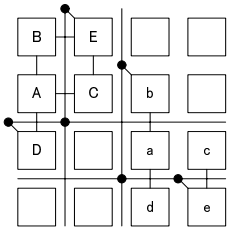

図 4: PD に Module が割り当てられた例

2.2 提案する資源割り当て手法

本論文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 [8] を用いた再配置を提案する。前者は計算時間が $O(n)$ であるため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に、後者は再配置との親和性及びそれ自体の性能の良さ [7] から、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。

なお、本節での例についてはわかりやすさのために、CB は縦横 4 個の計 16 個存在し、LW は隣接した CB に対してのみ用意されているモデルを考える。

2.2.1 No-Backtrack による配置配線

No-Backtrack による配置場所決定の手順は、以下のようにまとめられる。また、図 3 の Module を配置する過程を例として図 5 に示す。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。例えば、図 5(e) で、B, C, D に接続する E の配置場所を決定する場合を考える。この場合では、C に対してのみ、GW を XY 方向にそれぞれ 1 本計 2 本用いる、$(x, y) = (1, 0)$ が計 2 本で最小となる。
   - 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、$q \geq p$ 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。例えば図 5(a) では、今後 3 つのノードに接続しなければいけないという情報を元に、$(0, 0)$ ではなく $(0, 1)$ を選ぶ。
3. 全てのノードの配置が終わるまで、2. を繰り返す。

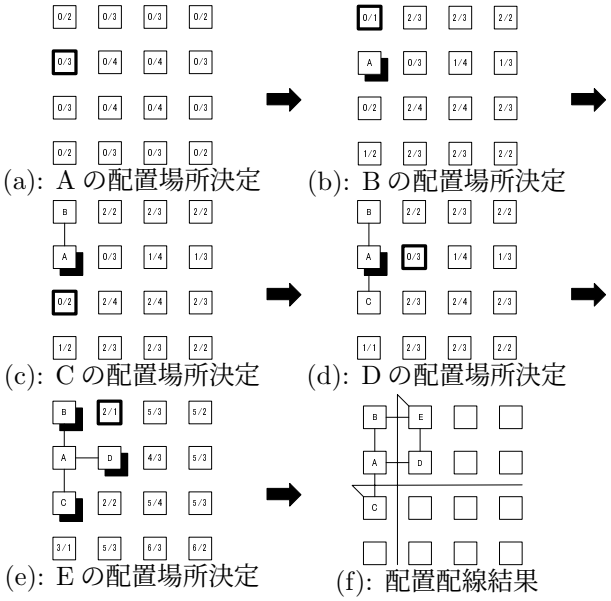

図 5: No-Backtrack による配置配線。“m/n”の m は、影のついた CB へ接続する際に必要となる GW の数であり、n は、その CB が LW で接続できる空き CB の最大値である。太枠の CB が配置先として選ばれる。

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、図 6 の過程を経て配置する CB を決めることができる。

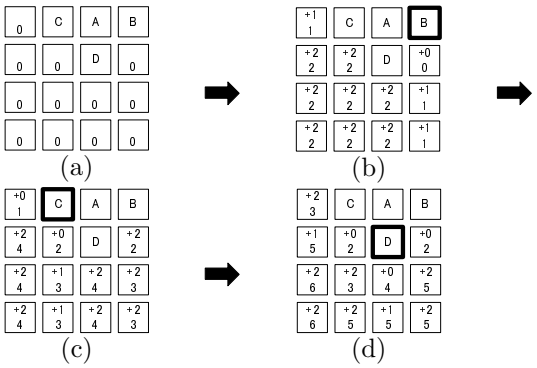

図 6: 配置 CB を選択する際に行われる動作 - 上段は太枠の CB に接続するために必要となる使用 GW 数で、下段はそれまでの合計である。(a) のように配置されている場合、(b), (c), (d) の段階でそれぞれ B, C, D に接続する際の使用 GW 数を計算し、最も合計が少なくなる (1, 1) を選ぶ。

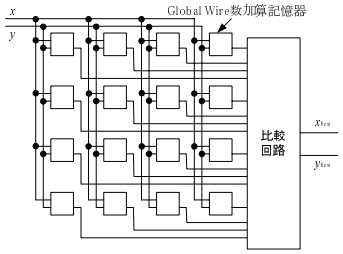

図 7: 使用 GW 数が少なくなるように CB 選択を行うハードウェア

この動作は、図 7 のハードウェアで実現できる。ここで、(a, b) の CB に対応する GW 数加算記憶器は、図 8 の構成で実現できる。ただし、f は、

$$f(a,b,x,y) = \begin{cases} 0 & (a,b) \text{ と } (x,y) \text{ の間に LW が存在} \\ 1 & (a,b) \text{ と } (x,y) \text{ の間に LW が存在せず、} a = x \text{ または } b = y \\ 2 & \text{それ以外} \end{cases} \quad (1)$$

である。この回路に対してクロックごとに接続先 CB の座標を順次流し込むことによって、対応する CB に配置を行った際に必要となる GW の本数が計算できる。その際の時間は、GW 数加算記憶器にデータが流し込まれる回数で見積もることができる。

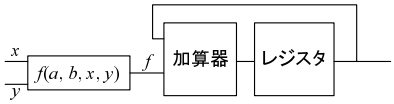

図 8: GW 数加算記憶器

2.2.2 Force Directed による再配置

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。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。

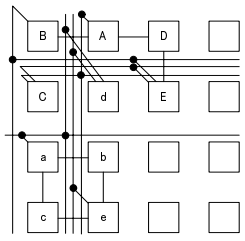

図 9: 再配置例 - $(0, 1)$ に配置されているノードを最適な場所に移動する。

例えば、図 9 のように、図 3 の Module が 2 つ割り当てられており、$(x, y) = (0, 1)$ にあるノードの再配置を行う場合を考える。この場合、他の各 CB に移動した場合の使用 GW 数の増減 (図 10) から、$(1, 1)$ に移動した場合に最も使用 GW 数を減らせることが分かる。そこで、両者の CB に配置されているノードの入れ替えを行い、影響する配線を改めて行う (図 11)。この再配置は以下のようなタイミングで行われる。

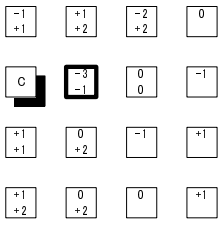

図 10: 移動先の決定 - 上段は $(0, 1)$ のノードをそこに、下段はそこのノードを $(0, 1)$ に移動した場合の使用 GW 数の増加である。

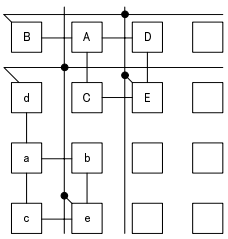

図 11: 再配置後 - 使用 GW 数が 4 減少している。

1. 割り当て/解放要求を処理する。
2. Module が割り当てられている CB を 1 つ選び、移動によって使用 GW 数が最も減らせる場所を探す。
3. 使用 GW 数が減少するのであれば、入れ替えを実行する。入れ替えた後の配線に失敗した場合には元の状態に戻す。
4. まだ次の割り当て/解放要求が来ていなければ、2. に戻る。

3 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 1 では座標あたりの GW 数は 3 本である。

3.1 評価環境

3.1.1 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 本の枝で構成されている。

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 本の枝で構成されている。

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 本の枝で構成されている。

これらを等確率に選び、割り当て/解放を行った。ただし、空き CB の数が足りなくなるような割り当てが要求されることはないようにベンチマークを生成した。なお、CB 使用率を特定の値付近で保つために、使用 CB 数が 50 よりも小さい場合には次の要求は必ず割り当て、使用 CB 数が 60 よりも大きい場合には次の要求は必ず解放になるようにして、ベンチマークを生成した。なお、ベンチマークの長さは割り当て、解放を合わせて 10,000 回で、半分の 5,000 回が割り当てである。ただし、Simulated Annealing を用いた割り当てに関しては、計算時間の都合から、長さ 1,000、割り当ては 500 回のデータを用いた。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。

3.1.2 比較対象

No-Backtrack による配置配線と比較するために、以下の 2 つの配置アルゴリズムを用意した。

Force Directed による配置 再配置に用いた Force Directed を配置の際に用いる。乱数によって Module のノードを空き CB に対して初期配置とする。その後、Module のノードを 1 つ選び、他の Module が使用していない CB を対象に、入れ替え先を決定し、入れ替えを行う。全てのノードに対してこの操作を繰り返す。その後、もし入れ替えが行われており、しかも繰り返しの上限回数を超えていなければ、繰り返し回数を 1 加え、再度 Module の全てのノードに対し、同じ操作を繰り返す。繰り返し回数の上限を変えることによって、質と計算時間のバランスを取ることが可能である。

Simulated Annealing による配置 Simulated Annealing とは、評価関数が良くなる方向へ配置状態を変化させていく山登り法に、最初のうちは悪い方向へも進めるようにしたものである [8]。これを用いて配置を行う。乱数で初期状態を決定し、使用 GW 数を評価関数とした。この手法は、計算時間が長いものの良い解を得られることが知られているので、時間をかけた場合の理想値として比較することができる。

3.2 実験及び結果

本稿では再配置を行わない場合について、GW 数を変化させた場合の失敗率及び計算に要する時間を調べ、再配置を行った場合に必要となる配線資源がどの程度減少するかを評価した。以下では各実験及び結果について述べる。

3.2.1 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 12 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。

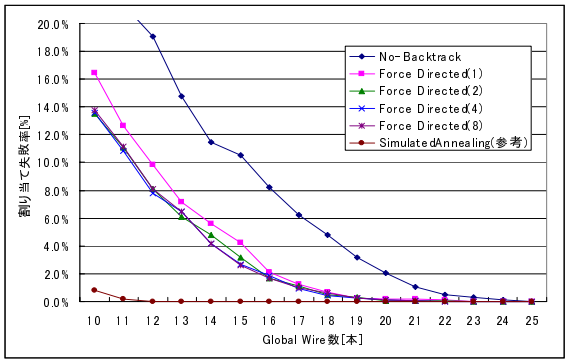

図 12: 座標あたりの GW 数と失敗率の相関 - Force Directed(x) の x は繰り返し回数の上限

3.2.2 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 (図 7) 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 13 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。

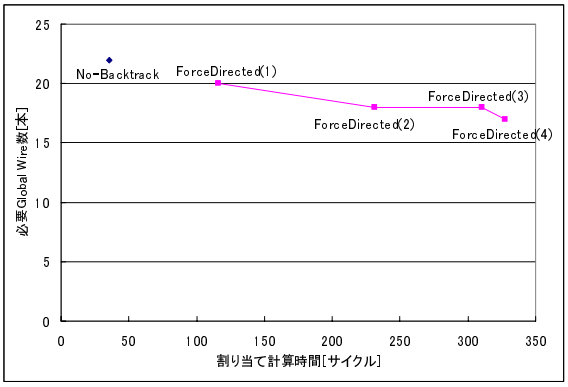

図 13: 割り当てに要する時間と失敗を 1%以下に抑えるために必要な座標あたりの GW 数の関係 - Force Directed(x) の x は繰り返し回数の上限

3.2.3 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 14 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 13 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。

4 おわりに

本研究では動的な資源割り当ての手法としてバックトラックを行わない $O(n)$ のアルゴリズム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 Force Directed アルゴリズムによる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。

今後の課題としては、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が挙げられる。特に、再配置の

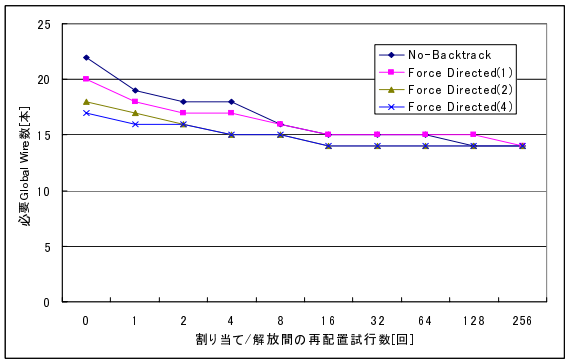

図 14: 再配置の回数と、割り当て失敗を 1%に抑えるための座標あたりの GW 数の関係。Force Directed(x) の x は繰り返し回数の上限

際に入れ替える組を決定する機構については、詳細を検討する必要がある。一方、実行を行うモデルについても、具体的なアプリケーションにおいての検討が必要となる。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 Reconfigurable Hardware に指示を出すようなアプリケーション及びその評価環境を用意する必要がある。

謝辞

本論文の研究は、一部、JST CREST ディペンダブル情報処理基盤、21 世紀 COE 情報技術戦略コア研究費によります。また、本研究はルネサステクノロジ社の技術協力を得て行われました。

参考文献

[1] Eylon Caspi, Michael Chu, Randy Huang, Joseph Yeh, John Wawrzynek, and Andre DeHon. Stream Computations Organized for Reconfigurable Execution (SCORE). In *Field Programmable Logic and Applications*, pp. 605–614, 2000.

[2] K. Compton, J. Cooley, S. Knol, and S. Hauck. Configuration Relocation and Defragmentation for FPGAs. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000.

[3] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 2000.

[4] Seth Copen Goldstein, Herman Schmit, Mihai Budiu, Srihari Cadambi, Matt Moe, and R. Reed Taylor. PipeRench: A reconfigurable architecture and compiler. *Computer*, Vol. 33, No. 4, pp. 70–77, 2000.

[5] Zhiyuan Li, Katherine Compton, and Scott Hauck. Configuration Caching Techniques for FPGA. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000.

[6] X.-P. Ling and H. Amano. WASMII: a data driven computer on a virtual hardwrae. In *Proceedings of the IEEE Internatilnal Symposium on FPGAs for Custom Computing Machines*, pp. 33–42, Apr. 1993.

[7] Chandra Mulpuri and Scott Hauck. Runtime and quality tradeoffs in FPGA placement and routing. In *FPGA*, pp. 29–36, 2001.

[8] K. Shahookar and P. Mazumder. VLSI cell placement techniques. *ACM Computing Sureys*, Vol. 23, No. 2, pp. 143–220, Jun. 1991.