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

2006－ARC－167（18）
2006－HPC－105（18）
2006／2／28

# 制約充足問題を解く正当な並列プログラムの生成について

斉藤雄介[†]　棟朝雅晴[††]　赤間清[††]

本論文では、等価変換による計算モデルに基づく、並列プログラム生成の理論を提案する。この理論では、与えられた問題を表現する確定節集合を変換するための等価変換ルールの集合から、正当な並列プログラムが生成される。この手法の正当性について説明し、また、ある制約充足問題を解く並列プログラムを使用して簡単な実験を行い、その有効性を示す。

# Generation of Correct Parallel Programs for Solving Constraint Satisfaction Problems

Yusuke Saito[†] , Masaharu Munetomo[††] and Kiyoshi Akama[††]

Based on Equivalent Transformation (ET) computation model, a theory of parallel program generation is proposed, where correct parallel programs are generated from a set of ET-rules that transform a set of definite clauses that represents a given problem. Correctness of the method is explained. Usefulness of the method is partly shown by experiments of a parallel program generated for a constraint solving problem.

## 1. はじめに

並列計算に関する研究としてさまざまな試みがなされているが，並列性能を出すことは一般に難しい問題であるとされている．並列性能がどれだけ得られるかといった問題は，その計算に必要となる処理全体のうち，並列に実行可能な領域をどれだけの割合見い出せるか，という点にかかっている．この並列実行可能領域の抽出という観点において，現在の主な並列化のアプローチは，各処理が参照する変数空間同士の関係に着目し，参照する変数空間が互いに独立な処理ブロックの組を見つけ，それらのブロック同士を並列処理可能な領域として抽出することで並列化を行っている．

この変数空間の分解可能性に関する証明は，人の手で場当たり的に行われているのが現状であり，ある程度以上の複雑さを持つ問題を扱うプログラムの場合，変数依存関係の解析が困難となるため，問題が複雑であるほど十分な並列性能が得ることは難しくなる．

また，逐次プログラムを入力とする並列化手法は多数存在するが，逐次プログラムは計算内容を逐次処理によって記述したものである以上，並列化を行おうとする始めの段階で，すでにいくらかの処理並列性が排除されてしまっているという可能性が考えられる．

このような背景を踏まえると，並列プログラムの正当性を，場当たり的でない，統一的な方法で保証することができる枠組みの必要性は大きいと考えられる．また，並列性の抽出を究極的に行い，より高い並列性能を目指すためには，問題の仕様をより正確に表現した内容を入力とすることが有効であろうと予想される．与えられた計算資源をいかにして利用するかといった問題も，並列性能を考える上でのポイントとなってくるが，適切なジョブ割り当てについて評価を行うためには，多様なプログラム（アルゴリズム）を生み出すことのできる枠組みも重要となる．

これに関連して，正当な逐次プログラム生成のための統一的な枠組みとして，赤間らにより，等価変換計算モデル [1] に基づくプログラム生成理論 [2] が提案されている．この枠組みでは，問題仕様の等価変換ルールの集合から，多様な逐次プログラムを生成することが可能であるとしている．この考え方に基づくならば，解くべき問題の仕様が節で表現され，その節の等価変換ルールからなる集合が与えられれば，そのルール集合をソースとして任意の逐次プログラムを生成可能であるといえる．しかし，これは等価変換計算モデルの

† 北海道大学大学院 情報科学研究科 複合情報学専攻
Division of Synergetic Information Science, Graduate School of Information Science and Technology, Hokkaido University
†† 北海道大学 情報基盤センター 大規模計算システム研究部門
Division of Large-Scale Computational Systems, Information Initiative Center, Hokkaido University

中で定義された等価変換操作の逐次適用によってなされる計算の正当性・非決定性に基づいたものであり，並列プログラム生成については考慮されていなかった．

本論文では，逐次計算の正当性を統一的に扱うことが可能な枠組みである等価変換計算モデルに基づく，並列プログラムの生成について理論的検討を行い，実験によってその妥当性を示す．

## 2. 等価変換計算モデルに基づく正当なプログラム生成

本研究が前提とするプログラム生成理論は，等価変換理論において定義された，等価変換による逐次計算モデルに基づいている．この理論は，逐次的等価変換による計算の正当性・非決定性に基づき，正当な逐次プログラム生成のための枠組みとして提案されている．

### 2.1 等 価 変 換

等価変換とは，対象の意味を保存した変換操作であり，ある 1 つの等価変換操作は 1 つの等価変換ルールとして表現される．ある表現 $A$ を変換した結果として表現 $B$ が得られたとき，

$$Meaning(A) = Meaning(B)$$

が成り立つならば，その変換は等価変換である．

### 2.2 等価変換理論

等価変換理論は，等価変換に基づく計算理論である．等価変換理論における計算モデルでは，計算は等価変換であり，等価変換ルールに対し，その適用順序に関する情報を付加することによって，さまざまな計算を作り出すことができる．

計算の内容は，使用する等価変換ルールとその適用順序によって決定される．同時に適用する等価変換ルールの数を 1 つに限定し，適用順序を逐次的にすることによって得られる計算が逐次計算に相当する．逆に，複数のルールを互いに同時性をもたせて適用させるような場合，得られる計算は並列計算となる．

等価変換が意味を保存した変換であることから，等価変換を逐次的に適用することによってなされる計算は，明らかに正当であるといえる．しかし，並列計算の場合には，等価変換操作が完結する前に別の等価変換操作によって特殊化がなされる可能性があり，かならずしも正当であるとは限らない．

等価変換ルールには，N ルール，D ルール，B ルール の 3 種類が存在するが，このうち，その適用に非決定性を持った N ルールのみが，その計算に並列性を有する可能性がある (**図 1**)．

### 2.3 等価変換計算モデルに基づくプログラム生成

この逐次プログラム生成理論におけるプログラム生成の流れは大きく 3 つのフェーズに分かれ，それぞれ，1) 節で表現された任意の問題をインプットとして与え，2) その節の等価変換ルール集合を作成，3) ルール集合から正当性を保証された任意のプログラムを生成する，である (**図 2**)．

$$\begin{aligned}\langle Head\rangle, \{\langle Cond\rangle\} &\Rightarrow \{\langle Exec\rangle\}, \langle Body\rangle;\\ &\Rightarrow \{\langle Exec2\rangle\}, \langle Body2\rangle;\\ &\vdots\\ &\Rightarrow \{\langle ExecN\rangle\}, \langle BodyN\rangle.\end{aligned}$$

**図 1** ET N ルール

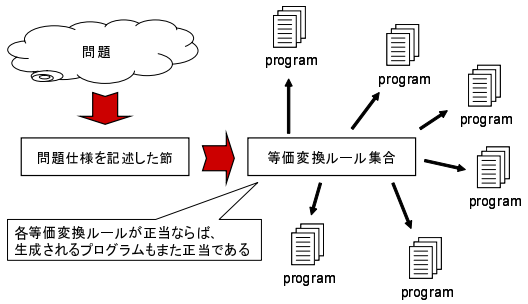


**図 2** 等価変換モデルに基づくプログラム生成

すなわち，ある問題を表現した節の等価変換ルール集合は，その問題を解くためのプログラムの部品であると捉えられる．等価変換ルール集合から任意のルールを選択して等価変換ルールの部分集合を作り，それらの適用順序情報を付加することにより，はじめてそのルール集合は 1 つの (抽象的な) 計算を構成する．ここで得られる計算は抽象的な表現であり，この「計算のテンプレート」をプログラム言語の記述に対応させることで，任意のプログラムを生成することができる．

逐次的な等価変換による計算の持つ正当性，非決定性より，集合中のルールを任意の順序で逐次的に組み合わせて得られる等価変換列が与える計算は，その正当性を保証されている．そのため，この枠組みで作成される任意の逐次プログラムもまた正当である．

## 3. 等価変換ルールからの並列プログラム生成に関する検討

ここでは，等価変換による逐次計算モデルを拡張し，等価変換計算における並列処理可能性について検討を行い，等価変換計算モデルに基づく正当な並列プログラム生成について考察する．

### 3.1 ルール適用における正当性

2.2 で述べたように，等価変換ルールにはその適用順序に関する情報は含まれず，集合として与えれたルー

$$A, \{\langle Cond \rangle\} \Rightarrow \{\langle Exec\_without\_Specialization \rangle\}, A, Specialization.$$

**図 3** 特殊化遅延ルール

ルの 1 つ 1 つはそれぞれ独立である．ルール集合中のどのルールを使用し，適用順序をどのように設定するかを指定することで，全体としての計算が決定される．ここで，等価変換ルールからの計算の生成という観点から，計算を次の 3 通りに分類する．

(1) **時間的分離**
各等価変換ルールの適用操作を時間的に分離させたものである．この計算は，等価変換ルールを逐次的な形に組み合わせることによって得られる．1 つの等価変換操作の終了後に次の等価変換操作に移るので，同時に 1 回の特殊化しか起こらないことが保証され，正当性の保証が最も容易な制御方法である．

(2) **空間的分離**
変数空間的に分離可能な等価変換操作同士を並列的に組み合わせる方法である．変数空間を共有しない処理同士ならば，明らかに同時実行可能であり，この組み合わせから得られる並列計算は，正当である．また，多くの並列化アプローチで抽出を行っている並列処理可能領域は，このタイプの処理部である．

(3) **時間的，空間的に非分離**
等価変換ルール適用の際，複数の処理が時間的・変数空間的に接点を持つような場合である．変数の依存関係が問題となるため，並列処理は難しい．

本研究は，(3) のドメインに分類されるようなルールの組み合わせ方の可能性について検討し，等価変換計算モデルに基づく正当なプログラム生成の枠組みを並列プログラム生成に適用することが目的である．

### 3.2 特殊化遅延ルール

3.1 で述べたように，扱う変数空間に重なりを持った複数の等価変換計算を並列的組み合わせについて検討する．理論的には，その適用に非決定性を有する N ルールのすべてに対し，以下と同様の議論が可能だが，本論文では，提案手法の妥当性を示すための理論的基礎を構築することを目的とし，**図 3** の N ルールのみを対象とする． このルールは Exec と Body に特徴を持ち，以後，特殊化遅延ルールと呼ぶ．

通常のルールでは，Exec において特殊化計算とその結果の特殊化が行われるのに対し，この特殊化遅延ルールの Exec 中のアトム `Exec_without_Specialization` では，特殊化計算のみが行われ，その計算結果の特殊化処理は，Body の `Specialization` アトムで行われる形になっている．つまり，このルール適用によってなされた計算結果をすぐに特殊化せず，一度 N アトムとして展開しておくことにより，この計算の特殊化が任意の時間遅延されることになる．また，特殊化遅延ルールは自己再帰的な性質を持つ．Head アトムである $A$ が Body にも存在するという構造をしており，実際の計算において適用頻度の高いルールであると考えられる．そのため，このルールから並列処理を生成できれば，それは有効である可能性が高い．

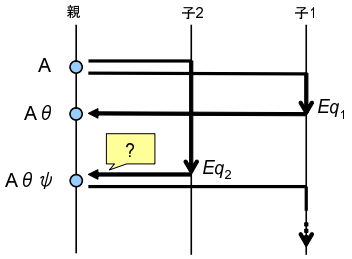


**図 4** Master-Slave 方式の並列計算

### 3.3 正当な並列プログラムの生成

ここでは，並列に適用した場合にも正当な計算が可能な特殊化遅延ルールについて検討するとともに，そのルールを使用した並列プログラムの生成について考える．以下の議論は，並列計算として親子（Master-Slave）方式での並列計算（4.1.3 参照）を考える．また，簡単のため，特殊化遅延ルールは $A \Rightarrow A, Eq.$ と表現し，Cond や Exec は省略した．

**図 4** は特殊化遅延ルールによる等価変換操作を並列的に適用する場合の様子を表したものである． 親プロセスの節の中で起こる変換のプロセスを考えると，

(1) アトム $A$ が存在する状態から
(2) $A$ を子プロセス 1, 2 に送信する
(3) $Eq_1$ を子プロセス 1 から受信する
(4) $A$ に対して特殊化 $\theta$ を実行し，$A\theta$ にする
(5) $Eq_2$ を子プロセス 2 から受信する
(6) $A\theta$ に対して特殊化 $\psi$ を実行し，$A\theta\psi$ にする

となる．

ここで，子プロセス 1 は $A$ の情報を受け取って $Eq_1$ を計算し，親にその結果を返信する．子プロセス 2 も同様に，$A$ の情報を受け取って $Eq_2$ を計算し，親にその結果を返信する．とした場合，

$A \Rightarrow A, Eq_1.$
$A \Rightarrow A, Eq_2.$

が等価変換となるように，Eq が計算される．また，$A \Rightarrow A, Eq_2.$ より，

$A\theta \Rightarrow A\theta, Eq_2\theta.$

も等価変換であることがわかる．

$$A\,\theta, \{\langle Cond \rangle\} \Rightarrow \{\langle Exec\_without\_Specialization \rangle\}, A\,\theta,\ Specialization\,\theta.$$

**図 5** 計算中の外部特殊化を許容する特殊化遅延ルール

さらに,

$\theta = spec(Eq_1)$

$\psi = spec(Eq_2\theta)$

とおく. (ただし $spec(Eq)$ は, $Eq$ の内容の特殊化を実行することを表す) そうすると, 親プロセスでの変換プロセスは,

$A \to A, Eq_1 \to A\theta \to A\theta, Eq_2\theta \to A\theta\psi$

となるが, これは等価変換プロセスであり, 正しい計算であるといえる.

以上より, 次の結論に達する.

特殊化遅延ルール $A \Rightarrow A, Eq.$ の並列的適用によってなされる計算は, 以下の条件

「親プロセスがある $Eq$ の特殊化計算を子プロセスに依頼した時点から, その計算結果を受信するまでの間に起こったすべての特殊化が $\delta$ であるならば, 親は $Eq$ の特殊化操作を,

$Eq\delta$

として実行する.」

を満たすならば, 正当である.

また, この計算は, 一般には難しい, 変数空間を共有した処理同士の並列計算になっていることがわかる. よって, 並列プログラムを作成するにあたって, 特殊化遅延ルール $A \Rightarrow A, Eq.$ をいかに多く作れるか, ということが並列性能を得る上での鍵となる.

## 4. 実験および考察

### 4.1 実 験 条 件

提案手法の妥当性・有効性を調査・予測するため, 制約充足問題の例としてペンシルパズルお絵かきロジックを解く並列プログラムを作成し, 以下の実験によってその性能を調べた.

(1) 提案手法で作成した並列プログラムの並列性能を調査

(2) 並列処理タスク量を $n$ 倍した場合の並列性能を調査

実験 (1) では, お絵かきロジックパズルを解く直列プログラム, 並列プログラムをそれぞれ作成し, それらのプログラムを使って実際に問題を解くのに要した時間から, 並列プログラムの並列性能を調べた.

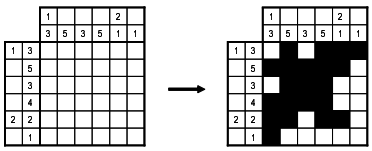

**図 6** お絵かきロジックパズルの例

実験 (2) はより大規模な問題を解いた場合の性能予測を目的とし, 並列実行部の処理量を 2 倍, 4 倍した場合の計算所要時間を実験 (1) 同様に調べた. 並列計算時においては, 問題の計算に加え, プロセス間の通信処理がコストとして発生する. 並列実行部の処理量を大きくし, 全体の所要時間に対する通信コスト率を小さくすることで, より純粋な並列性能データが得られることを期待している.

実験における各種条件設定は以下の通り.

**使用した問題数と盤面サイズ** 盤面サイズが $25 \times 25$ の問題 4 題を用意して実験を行った.

**並列環境** 以下のスペックのマシン 3 台からなる PC クラスタを使用した.

- CPU: Pentium 4 2.60GHz
- Memory: 1GB
- OS: WindowsXP Professional SP2

#### 4.1.1 お絵かきロジック

実験に使用するお絵かきロジックパズルは, 制約充足問題の一種である. これは, 盤面の各行各列に与えられた「塗りつぶし制約」をすべて満たすように, 盤面を黒または白に塗り分けるパズルである.

**図 6** のように, すべての行・列には, 塗りつぶし制約が数字の列で与えられており, それぞれ対応する行・列は, その制約の通りに塗りつぶされなければならない. また, 列に含まれる各数字は, 盤面を黒く塗りつぶす長さを表している. このような問題では, 各変数が互いに依存性を持っており, 一般に並列処理は難しいと考えられる.

提案手法に基づき, このパズルを解くプログラムを生成するための準備として, 問題を節で表現し, その節の等価変換ルール集合を構築する. このパズル問題の節表現として, 今回の実験では, **図 7** のような表現を用いた. 盤面の各行各列がそれぞれ 1 つの pat アトムに対応し, 上側の 6 つのアトムが行方向, 下側の残りの 6 つが列方向を表している. 節表現の方法は 1 通りではないが, 今回はパズルの解き方の方針として, 4.1.2 にて述べられる共通特殊化方策をとることを考慮した上で, このような表現方法にとっている.

#### 4.1.2 共通部分特殊化ルール

お絵かきロジックを解くためのテクニックはさまざ


**図 7** 図 6 の問題の節表現

$$(pat\ {}^{*}Pat\ {}^{*}Cell), \{(includesPVar\ {}^{*}Cell)\}$$
$$\Rightarrow \{(intersects\ {}^{*}Pat\ {}^{*}Cell\ {}^{*}Intersection)\},$$
$$(pat\ {}^{*}Pat\ {}^{*}Cell),$$
$$(specialize\ {}^{*}Cell\ {}^{*}Intersection).$$

**図 8** 共通特殊化ルール pat

$$(pat\ {}^{*}Pat\ {}^{*}Cell)\theta, \{(includesPVar\ {}^{*}Cell)\}$$
$$\Rightarrow \{(intersects\ {}^{*}Pat\ {}^{*}Cell\ {}^{*}Intersection)\},$$
$$(pat\ {}^{*}Pat\ {}^{*}Cell)\theta,$$
$$(specialize\ {}^{*}Cell\ {}^{*}Intersection)\theta.$$

**図 9** 共通特殊化ルールは外部からの特殊化を許容する

まに存在するが，今回の実験で採用した解き方は，制約から考えられるすべての塗りつぶしパターンの共通部分のみを特殊化する，というものである．

具体例として，ある行の塗りつぶし制約が「5」，行の長さが「6」の場合を考える．この行の塗りつぶし方には [白黒黒黒黒黒] と [黒黒黒黒黒白] の 2 通りがあり，このとき，両者に共通である中央 4 マスを黒に確定させるという方法である．

このような共通部分の特殊化計算は，等価変換ルールで表現すると，**図 8** のように記述できる．このルールは Head，Body 共に `pat` アトムを含み，`intersects` で計算した塗りつぶしパターンの共通部分を `specialize` で特殊化を行うという形になっており，3.2 の特殊化遅延ルールの 1 つだということがわかる．

また，共通部分特殊化という方策の持つ性質から，`intersects` による特殊化計算中に起こる任意の特殊化 $\theta$ に対し，明らかに**図 9** が成り立ち，`pat` ルールは並列処理を生成可能なルールであるといえる．

#### 4.1.3 プログラムの作成

実験に使用するプログラムを以下の手順で作成した．

(1) お絵かきロジックパズルの問題を節で表現

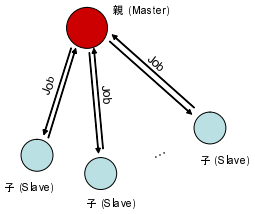


**図 10** 親子 (Master-Slave) モデル

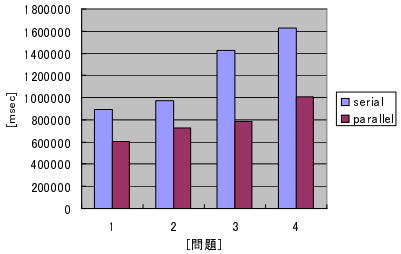


**図 11** 直列/並列計算時間の比較

(2) (1) で作成した節を変換し，計算するための等価変換ルールの集合を作成

(3) (2) で作成した等価変換ルール集合から，直列プログラム，並列プログラムをそれぞれ生成する

等価変換ルール集合のルールに適用順序に関する制御情報を付加して計算を作成し，これを何らかのプログラムに落とすことにより，その計算を具体化する．並列プログラムの場合，同じ並列計算のテンプレートから，さまざまなアーキテクチャ用のプログラムを生成することが可能である．

今回の実験では，PC クラスタを構成する 3 台のうち，1 台を親プロセス，残り 2 台を子プロセスとするマスタースレーブ方式で並列計算を行った．(**図 10**) 並列計算の方法としては，親プロセスは問題全体の管理と子プロセスへのジョブ割り当て処理を行い，子プロセスは親から送られてきたジョブの受信 ⇒ 計算 ⇒ 結果の返信を繰り返す．

### 4.2 実験結果

各問題を直列計算・並列計算のそれぞれの方法で解き，その所要時間を比較することにより，並列プログラムの並列性能を調べた．実験の結果はそれぞれ以下のようになった．

#### 4.2.1 並列性能

実験 (1) の結果を**図 11** に示す．いずれの問題でも，並列計算で解いた場合の所要時間は直列計算のそれよりも短い時間となっており，平均して，並列処理時には直列計算時の 64.8% の時間で問題を解くことができるという結果が得られた．

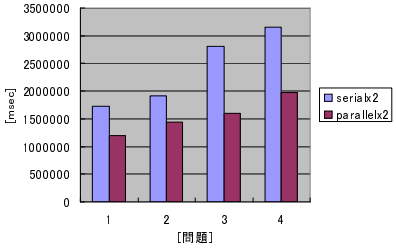


**図 12**　処理量 2 倍での計算時間比較

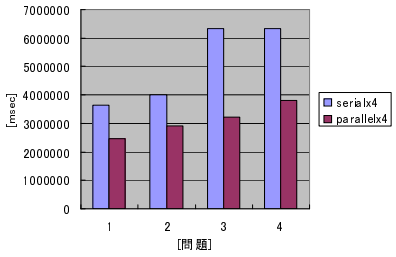


**図 13**　処理量 4 倍での計算時間比較

#### 4.2.2　問題の規模に応じた性能予測

実験 (2) の結果を**図 12**，**図 13** に示す．　実験 (1) との違いは，Exec の計算量である．具体的には，述語 `intersects` を 2 回，あるいは 4 回の複数回行い，Exec の計算量を実験 (1) の 2 倍 (4 倍) にしている．使用した問題は実験 (1) と同じで，各問題を直列，並列の両プログラムを使用して所要時間を計測し，5 回の平均をとった．

直列計算時との所要時間比率はほとんど変化せず，負荷 2 倍では 65.9%，4 倍では 62.9% であった．並列計算時に発生するコスト $Cost$ は，$Cost + Exec =$ 所要時間 と $Cost + Exec \times 2 =$ 所要時間 など 2 つの式から概算することは可能である．問題にもよるが，$Cost = 10$ [sec] 程度であり，全体の計算時間に対する割合がさほど大きくないため，負荷を増やしてもその影響は小さかったと考えられる．

## 5. おわりに

### 5.1　ま と め

本論文では，等価変換計算モデルに基づく，正当な並列プログラムの生成について提案した．等価変換計算モデルに基づくプログラム生成理論では，問題仕様を表現した節の等価変換ルールの集合から，多様なプログラムを生成可能であるとしている．この枠組みに基づき，並列計算プログラムを生成可能な等価変換ルールについて検討を行った結果，特殊化遅延ルール $A, \{\langle Cond \rangle\} \Rightarrow \{\langle Exec\_without\_Sp \rangle\}, A, Sp.$ が，任意の外部からの特殊化 $\theta$ に対し，$A\theta, \{\langle Cond \rangle\} \Rightarrow \{\langle Exec\_without\_Sp \rangle\}, A\theta, Sp\theta.$ を満たすとき，そのルールを使うことによって並列計算を生成可能であることを示した．

これを基に，制約充足問題としてお絵かきロジックパズルを解く並列プログラムを生成，PC 3 台からなるクラスタ環境を使用して並列計算を行い，直列プログラムでの処理時間を比較したところ，並列計算する処理量に応じてそれぞれ，64.8%，65.9%，62.9%という結果が得られた．

### 5.2　今後の課題

今回は，並列プログラムの性能を調べるという形で実験を行ったが，実験の目的としては，変数を共有した処理同士での並列実行の効果を見ることであり，どれだけの並列性能が出せるかというよりは，直列計算よりも計算時間が改善するかどうかを確かめることが目的である．

今回は，マスタースレーブ方式を採用し，ジョブを単純に子プロセスに均等に振り分けて計算を行わせたが，スケジューリングの調整などによって，計算効率を改善することは十分に考えられる．これと併せて，今後の課題としては，スーパーコンピュータやグリッド環境など，より多くのプロセッサ数を使用し，また，より大規模な問題を解くことで，提案手法の実用性についてより詳細な検証を行う必要性があると考えられる．

## 参　考　文　献

1) Kiyoshi Akama and Ekawit Nantajeewarawat: Formalization Of The Equivalent Transformation Computations Model, Proc. of the Fifth International Conference on Intelligent Technologies (InTech'04), pp. 190–199, Houston USA (2004.12).
2) Kiyoshi Akama, Ekawit Nantajeewarawat, and Hidekatsu Koike: Program Synthesis Based on the Equivalent Transformation Computation Model, Proc. 12th International Workshop on Logic Based Program Development and Transformation (LOPSTR 2002), Madrid, Spain, pp. 285–304 (2002).