ハイパフォーマンス
コンピューティング 81－8
（2000．6．2）

NaraViewを利用した実アプリケーションの並列化事例

羽田 昌代* 笹倉 万里子† 城 和貴‡

* 奈良女子大学人間文化研究科 † 岡山大学工学部情報工学科
‡ 奈良女子大学理学部情報科学科

並列計算機を効果的に使用するためには，これまで逐次計算機で使用されてきたアプリケーションを並列プログラムに再構築する必要があるが，その作業は煩雑で容易ではない．アプリケーションの並列化を自動で行う自動並列化コンパイラが開発されているが，これを用いても正確で効果的な並列化を行うことは極めて難しい．このため自動並列化コンパイラには，その使用を支援するツールが開発されていることが多い．我々はこれまでにNaraViewという並列化支援ツールを開発している．NaraViewは並列化コンパイラParafrase-2から得られる中間表現の情報を抽出し，与えられたプログラムの視覚化を行う．本稿では実アプリケーションの並列化を通してこのNaraViewの有用性を評価する．

Parallelization of Real Applications with NaraView: A Case Study

Masayo Haneda* Mariko Sasakura† Kazuki Joe*

* Nara Women's University † Okayama University

To use parallel computer systems effectively, users need to reconstruct their sequential appreciation into parallel programs, and such parallelization is not an easy work for general users. To solve such a problem, parallelizing compilers are developed, but it is still quite difficult to parallelize applications correctly, efficiently, and automatically. For this reason, some parallelization support tools are desired. NaraView, as such a support tool, visualizes the given programs by extracting internal information from a parallelizing compiler Parafrase-2. In this paper, we validate the ability of NaraView by parallelizing a real application.

1 はじめに

近年，並列コンピュータの普及が進み，これまで主流であった高価なベクトルコンピュータとその地位が入れ替わりつつある．しかし，ベクトルコンピュータで使用されていたアプリケーションをスカラ並列計算機上で使用できるように再構築することは難しい．このため，プログラムを自動的に並列化する自動並列化コンパイラは長年に渡り開発が進められ，実用可能な物もいくつか存在している．自動並列化コンパイラは，ユーザーの代わりに多くの解析とプログラム変換を逐次プログラムに対して行う．この時に行う解析や変換の手法は，コンパイラ自身が決めるのではなく，ユーザーの指定が必要である．これらの最も効果的な選択は，並列化を行うプログラムによって異なるために，自動並列化コンパイラを用いてもプログラムの最適な並列化は困難である．このような理由から自動並列化コンパイラとともに，その操作を支援するツールも同時に開発されることが多い．我々は自動並列化コンパイラの一つであるParafrase-2[2]に対応する並列化支援ツールNaraView[3]を開発している．NaraViewは，Parafrase-2が出力する並列化の情報のひとつであるhierarchical task graph(HTG)[1]を視覚化することによって，プログラムの構造を表示し，その中でユーザーが指定した部分のデータ依存等の情報を表示する．この視覚化を，コンパイラの並列化の処理とインタラクティブに行うことで，ユーザと自動並列化コンパイラの対話形式の情報のやり取りが可能になる．これにより，ユーザは最適化手法の具体的な効果を直感的に理解し，より有効な並列化を行うことが可能になる．またユーザは，得られる情報からコンパイラが抽出することができない並列性を見つけ出すことも可能である．

Parafrase-2はイリノイ大学で開発されている自動並列化コンパイラの一つで，主に大学での研究目的に使用されている．Parafrase-2は入力として逐次プログラムとパスファイルを用いる．パスファイルでは，付属するいくつもの最適化手法(パス)の中から，適用するものを任意の順序で指定できる．このためユーザーの並列化の戦略が，生成される並列プログラムの実行性能を左右するようになった．

本稿では実アプリケーションの並列化を，NaraViewを全く用いない手法，NaraViewの情報を用いてパスファイルを修正する手法，修正されたパスファイルの適用後にNaraViewを用いてアプリケーションのソースコードを直接操作する手法の3つの手法で行い，比較を行う．その結果から視覚化されたプログラム情報の並列化における有効性と，並列化支援ツールNaraViewの性能を示す．本稿は次のように構成される．2章ではNaraViewの概要と並

列化を行う実アプリケーションについての説明，3章では Parafrase-2 の適用によって並列化されたプログラムと，その情報を基に NaraView によって出力された画像を利用した並列化を行う．4章では3章の結果の検討を行い，5章ではまとめと今後の課題について述べる．

2 アプリケーションの説明

2.1 NaraView

NaraView はソースレベルでのデータ依存関係と制御フロー情報を視覚化することで，プログラムの並列化を支援するシステムである．NaraView は，自動並列化コンパイラ Parafrase-2 の内部表現を視覚化する．NaraView は，プログラム情報を4つのビューで表示する．

1. プログラム構造ビュー
   与えられたプログラム全体の流れ，並列度，ループ構造，文の種類を3次元の木構造で表す．
2. ソースコードビュー
   プログラム構造ビューで指定した部分のソースコードを表示する．
3. CFG ビュー
   ループの階層構造にしたがって各階層の制御フローグラフ (CFG) を2次元表示する．
4. データ依存ビュー
   プログラム中の，あるループ内での変数に関するデータ依存関係を3次元空間に表示する．

これらのビューは，ユーザが行う最適な並列化の戦略の決定を支援する．NaraView を用いた並列化は，次のような手順で表される．

1. プログラム構造ビューを表示させる．ここで与えられたプログラムの構成を認識する．
2. プログラム構造ビューより，注目するループを特定する．NaraView は与えられたソースプログラムに対応するソースコード，データ依存ビューを呼び出すことができる．
3. ビューから得られる情報を基に，並列化手法をプログラムに適用して並列化を進める．ここで，直接ソースコードを修正することもできる．

2.2 並列化を行うアプリケーション (拡張ヒュッケル法) について

田島らは分子構造を高速に最適化する，拡張ヒュッケル法のプログラムを開発した [4]．このプログラムは，化学計算で広く使われているが厳しい制約のある拡張ヒュッケル法を修正したものである．炭化水素の分子に，このプログラムを用いて得られた計算値と観測値の相違は 10 パーセント以下である．

拡張ヒュッケル法の計算時間は $O(n^3)$ に比例する．ここで n は原子の数を表す．この手法の 99 パーセント以上の計算時間は行列の対角化に費やされるため，大規模な拡張ヒュッケル法の計算を行う際には，並列処理は効果的である．

本稿では，並列化の対象として Fortran77 で書かれた拡張ヒュッケル法を用いる．このアプリケーションは8個のサブルーチンで構成される．この中から，計算の主要な部分を含むサブルーチン hoqrv2.f を並列化の対象とする．これは，他のサブルーチンが極めて単純な構成であるためである．

3 NaraView を用いた並列化

本章では，Parafrase-2 と NaraView を用いた実アプリケーションの並列化を示す．

3.1 デフォルトパスファイルによる並列化

はじめに，Parafrase-2 に付属されているデフォルトパスファイルのみを用いて並列化を行う．Parafrase-2 に付属するデフォルトパスファイルの内容は，以下の通りである．

fixup	構文木とデータ構造を統合する
callgraph	コールグラフの作成
sumfcn	関数の参照情報の作成
libsum	ライブラリ関数の参照情報を読む
param_alias	引数のエイリアス関係を解析
donest	ループの入れ子関係を解析
depend	変数の read/write リストを作成
flow	フローグラフの作成
constant	定数伝播を行う
induction	誘導変数の検出
builddep	データ依存グラフの作成
hier	HTG の作成
dotodoall	並列化可能なループの検出
codegen	変換後のソースコードの作成

このパスファイルを用いることで得られる並列化されたプログラム hoqrv2.f(14(a) 参照) は，NaraView のプログラム構造ビューでは図 1 のようになる．NaraView のプログラム構造ビューでは，グラフの X 軸が制御フロー，Y 軸が並列度，Z 軸がループの深さを表現する．本論文中の全てのプログラム構造ビューにおいて，フローは画面の左から右に向かって進む．図 1 を観察することで，デフォルトパスファイルを用いた並列化によって2つのループが並列化されていることが分かる．

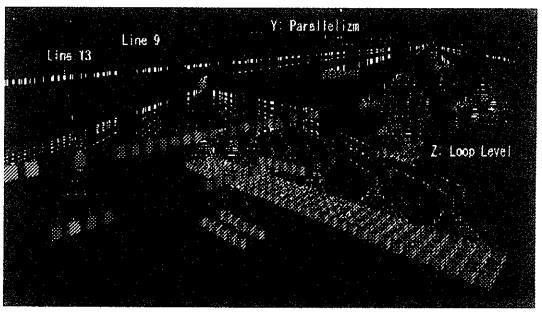


図 1: プログラム構造ビュー

3.2 パスファイルの修正による並列化

本節では NaraView から得られる情報に基づいて，デフォルトパスファイルを修正する．並列化されて

いないループは図 1 より簡単に検出することができる．並列化されていない各ループを観察することで，より効果的な並列化手法を見つけ出すことができる．

まず，並列化されていないループの中で制御フロー上で最初にあたる，9 行目のループに注目する．このループはその下に多数の子キューブを持つ．このため，その子キューブの中で最初にあたる 13 行目のループに注目する．このループは 3 つの文からなり，そのひとつは並列化可能で，後の 2 つが並列化できないと解析されている．キューブを選択すると，そのノードに対応するソースコードビューとデータ依存ビューを見ることができる．データ依存ビューでは，変数をキューブで表現し，そのキューブを xy 平面に配置することで各変数を表現する．変数へのアクセスが起こった時間を表すために，z 軸を用いる．この時間の単位には，文単位に設定された時間とイタレーション単位に設定された時間の 2 種類があるが，ここではイタレーション単位に設定された時間を使用する．これは Parafrase-2 が，ループを並列化する自動並列化コンパイラであるためである．図 2

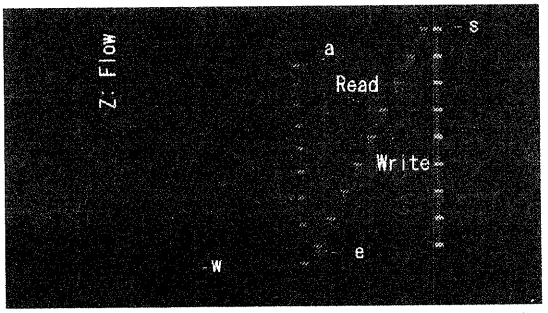


図 2: データ依存ビュー (line13)

より，変数 s と e に依存があることが分かる．s は xy 平面に垂直に配置されているので，スカラである．s の各キューブはポールで接続されている．このポールは，ループのイタレーション間で依存が起こっていることを表す．このポール状に配置されたキューブが，階段状の配置に変換されることで依存は解消することができる．このようなプログラム変換をスカラ拡張と呼ぶ．e を示すキューブは階段状に配置されているため，配列である．この配列ではループのイタレーションの中で起こるアクセスの間に依存があることがポールによって示されている．このような依存を解消するためにはループを分割することが挙げられる．ループを分割する変換手法には，ループ分割がある．

次に，プログラムのフロー上，13 行目のループの次にあたる 26 行目のループに注目する．このループのデータ依存ビュー (図 3 参照) では，ループ間にわたる依存があることが分かる．これは，依存がある配列 w の要素についてスカラ拡張を行うことで，依存を解消できる．また，図 3 の依存構造のパターンは，72 行目，76 行目のものと同じであることが，プログラム構造ビューとソースコードビューを使うことにより分かる．

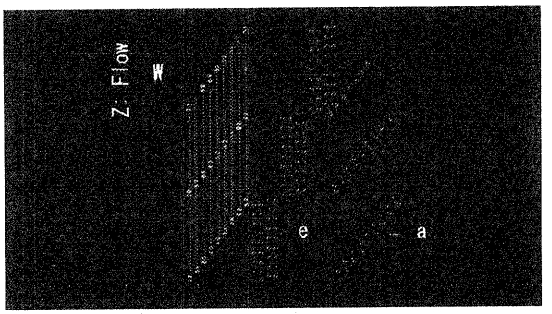


図 3: データ依存ビュー (line26)

同様の検討を他のループにも繰り返した結果，ループ分割とスカラ拡張を行うことで，より高い並列度が得られると予想されたため，これらの 2 つの変換手法をデフォルトパスファイルに追加する．

Parafrase-2 では，最も外側のループのみを分割の対象とするもの (outermost)，最も内側のループのみを分割の対象とするもの (innermost) と，内側から探索を行い，分割可能な時点で分割を行うもの (inside-out) の，3 種類の手法が実装されている．これらの 3 種類の中から各ループに対して適用する手法を変える適用法と，全てのループに一律に同じ手法のループ分割を行う適用法が存在する．ここでは，3 つの中でデフォルトである inside-out を一律に全てのループに適用する方法をとった．この修正されたパスファイルで並列化されたプログラム (図 14(b) 参照) に再度 NaraView で視覚化を行うと，図 4 のようになる．

視覚化された画像より，13 行目と 69 行目ののループが並列化されていることが分かる．ソースコードより，これらのループはループ分割によって並列化されたことが分かる．ここでは，スカラ拡張による効果は見られなかった．残りのループを並列化するためには，異なる変換手法を適用する必要がある．

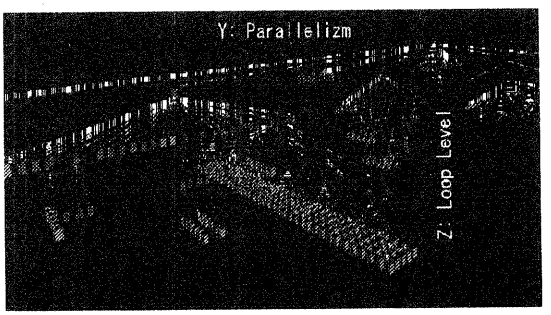


図 4: デフォルトパスファイルによる並列化

3.3 ソースコード修正による並列化

並列化されたプログラム (図 14(b)) において，更に並列化できる部分を探すと，プログラムを修正することで並列化が可能になるループをいくつか見つけることができる．しかし，Parafrase-2 を使ってそれらを並列化することは，難しいと判断されたため，プログラムを直接操作することによって依存を解消する手法をとる．

新しく出力された並列プログラムの 21 行目のループに注目する．このループは 3.2 章で最初に検討されたループである．この部分のデータ依存ビューは図 5 のようになる．図 5 より配列 e で，正の依存がある

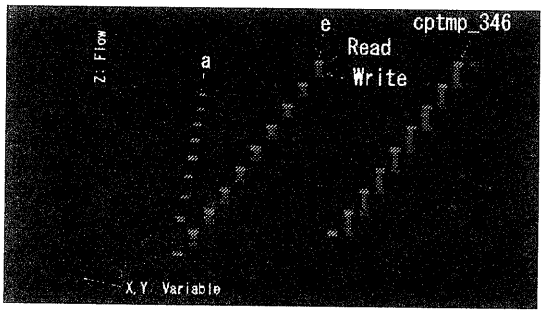


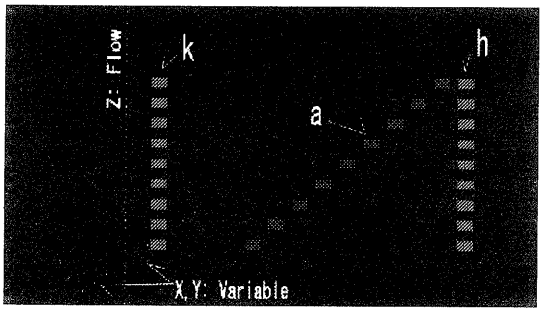
図 5: データ依存ビュー (line21)

ことが分かる．ここでソースコードを見ると，ループ内の各ステイトメントが異なるループに属するように，このループを分割を行うことで e の依存を解消することが予測される．Parafrase-2 のループ分割ではこの分割が行われなかったため，ソースコードを直接操作して分割を行い，並列化を行う．2 つに分割したループの 1 つは，そのまま並列化が可能であるが，もうひとつのステイトメントを含むループはイタレーション間に依存があるために，そのままでは並列化が不可能である．

並列化ができないループでは，配列 e の 2 乗の総和を取ることを行っている．図 14(a) に示されるソースコードでは，ループ内に 1 つ前のイタレーションで求められた値 s を含むステイトメントがあることが分かる．3.2 節で並列化されたソースコード (図 14(b) 参照) では，s はスカラ拡張によって配列に変換されていて，各イタレーションで求められる結果は異なる変数に格納される．このようなプログラムは，修正を加えることで，意味を変えることなく並列化することができる．

ただし，このようにプログラムの修正を行うことで，新しいループが必要となる．この部分の並列化は，実行時に与えられるデータセットによって，有効ではない場合もあるため，ここでは修正を行っていない．

また，84，103，118 行目のループは，明らかに依存がないことが，データ依存ビュー (図 6 参照) でも示されるが，Parafrase-2 には並列化可能であることが




```
      DO 348 i = k + 1,n                    84
        a(i,k) = a(k,i) * h                 85
348   CONTINUE                              86
```

図 6: データ依存ビュー (line84)

認識されないため，プログラマが直接並列コードに修正することで，並列化を行う．同様の修正をソース上に適用することで，並列化を進めていく．最終的に出力されるソースコードは図 14(c) のようになる．

4 結果の検討

本章では，3 章で行った並列化の評価を行う．

はじめに，最終的に並列化されたプログラム構造ビュー (図 7 参照) と，デフォルトパスのみを使った並列化の出力結果であるプログラム構造ビュー (図 1 参照) の比較を行う．図 1 と図 7 を表 1 を用いて比

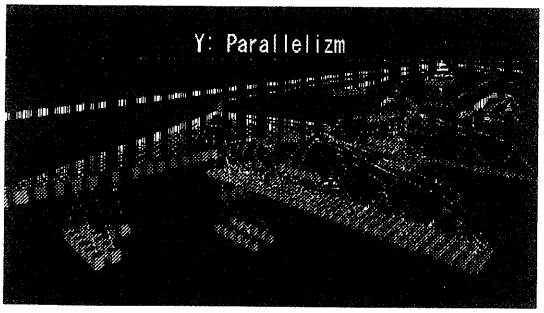
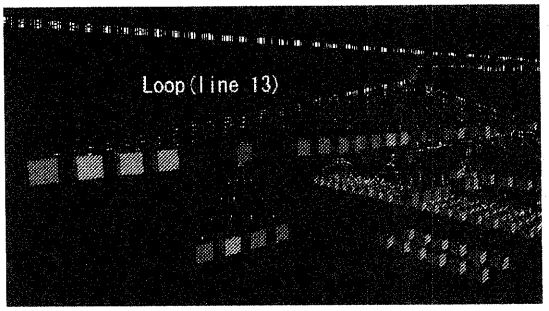


図 7: ソースコード修正による並列化

	Section 3.1	Section 3.2	Section 3.3
line-13	$\frac{3n^2-21n-6}{2}$	n^2+2n-8	$5(n-2)$
line-69	n^2-n-1	$\frac{n^2-n+1}{2}$	2

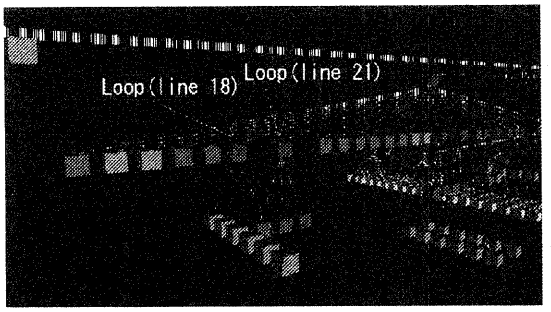
表 1: ループのイタレーション数

較すると，並列度が上がっていることが認識できる．3 章では，デフォルトパスファイルのみでの並列化，デフォルトパスファイルに変換手法のループ分割とスカラ拡張を加えたものによる並列化，修正後のパスファイルでの並列化の後にプログラムを直接修正する並列化の 3 種類を行った．これらの出力結果で




```
   DO 10 j = k + 1,n      13
    w(j,1) = 0.d0         14
    e(j) = a(k,j)         15
10  s = e(j) * e(j) + s   16
```

図 8: デフォルトパスファイルによる並列化




```
    CDOALL 10 j = k + 1,n          18
     w(j,1) = 0.d0                 19
10  CONTINUE                      20
    DO 347 j = k + 1,n            21
     e(j) = a(k,j)                22
     cptmp_346(j) =e(j)*e(j)
                  +cptmp_346(j-1) 23
347 CONTINUE                      24
```

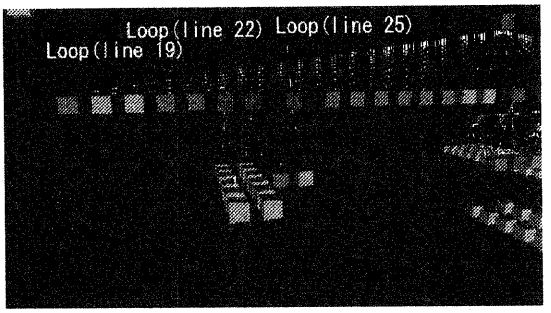
図 9: 修正を加えたパスファイルによる並列化

ある並列プログラムの並列度の違いを検討するために，それぞれのプログラム構造ビューの並列化が行われている部分を拡大して示す．

ソースコードの 13 行目に対応するプログラム構造ビューを各手法ごとに拡大すると，図 8(並列化されていない)，図 9(部分的に並列化されている)，図 10(完全に並列化されている) のようになる．これらの図は 13 行目のループが分割され並列化される過程を示している．

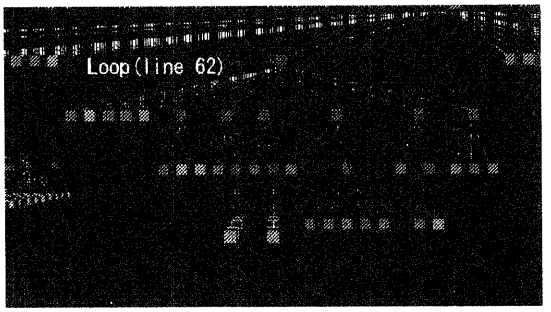
次に 62 行目のループを拡大した図と対応するコードを，図 11，図 12，図 13 に示す．ここで図 11 に示されるループが，図 12 で示されるように並列化され，また図 12 で示されるループも図 13 に示されるように並列化されていることが認識できる．

ここまでの比較の結果，視覚化された情報が並列化を支援することが示された．この結果は，NaraView によって並列化に必要な情報が，ユーザにより明確に伝わるようになることを示していると考えられる．




```
    CDOALL 10 j = k + 1,n          19
     w(j,1) = 0.d0                 20
10  CONTINUE                      21
    CDOALL 347 j = k + 1,n        22
     e(j) = a(k,j)                23
347 CONTINUE                      24
    DO 390 j = k + 1,n2           25
     cptmp_346(j)=e(j)*e(j)
                  +cptmp_346(j-1) 26
390 CONTINUE                      27
```

図 10: ソースコード修正による並列化




```
   DO 130 i = k + 1,n    69
    w(i,2) = 0.d0        70
130 a(i,k) = a(k,i) * h  71
```

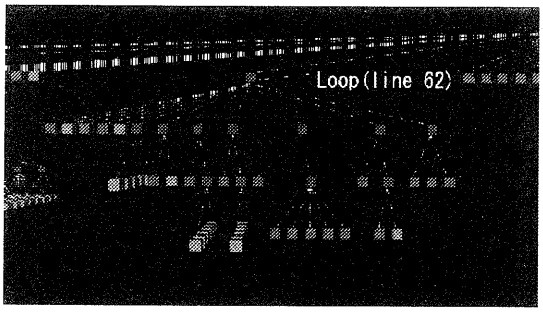
図 11: デフォルトパスファイルによる並列化

5 まとめ

本稿では，視覚化による並列化支援ツールの有効性の評価を行うために，拡張ヒュッケル法と呼ばれる化学計算の実アプリケーションに対して，3 つの手法で並列化を行った．

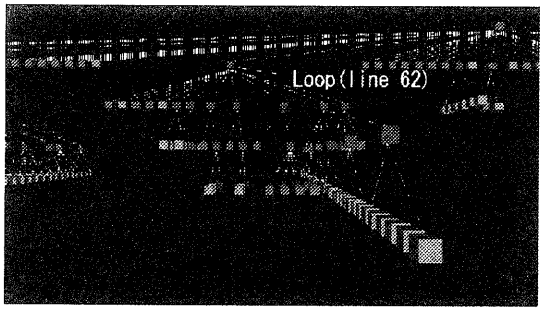
この 3 つの手法の比較によって，ユーザと自動並列化コンパイラが相互に情報のやりとりを行うことで，並列化が進む過程が認識された．このことより，情報の伝達の手法としてのプログラムの視覚化が，並列化の支援ツールとして有効であることが示された．

謝辞
並列化を行う対象となった実アプリケーションである，拡張ヒュッケル法のプログラムを提供してくださった産業技術融合領域研究所の長嶋雲兵氏に感謝します．




```
    CDOALL 130 i = k + 1,n 81
     w(i,2) = 0.d0         82
130  CONTINUE              83
     DO 348 i = k + 1,n    84
      a(i,k) = a(k,i) * h  85
348  CONTINUE              86
```

図 12: 修正を加えたパスファイルによる並列化




```
    CDOALL 130 i = k + 1,n                    84
     w(i,2) = 0.d0d3                          85
130  CONTINUE                                 86
     CDOALL 348 i = k + 1,n                   87
      a(i,k) = a(k,i) * h                     88
348  CONTINUE                                 89
```

図 13: ソースコード修正による並列化

参考文献

[1] M. Girkar and C. D. Polychronopoulos. The hierarchical task graph as a universal intermediate representation. *International Journal of Parallel Programming*, 22(5):519–551, 1994.

[2] C. D. Polychronopoulos, M. Girkar, M. R. Haghighat, C. L. Lee, B. Leung, and D. Schouten. Parafrase-2: An environment for parallelizing, partitioning, synchronizing, and scheduling programs on multiprocessors. In *International Conference on Parallel Processing, Vol. 2*, pages 39–48, 1989.

[3] M. Sasakura, K. Joe, Y. Kunieda, and K. Araki. NaraView: An interactive 3D visualization system for parallelization of programs. *International Journal of Parallel Programming*, 27(2):111–129, 1999.

[4] S. Tajima, T. Katagiri, Y. Kanada, and U. Nagashima. Parallel processing of molecular geometry parameter optimization by extended huckel method- an attempt at simple fast generation of molecular structure -. *J. Chem. Software.,2000,in press.*


```
      :
    DO 120 k = 1,n0 - 2                    9
      :
    DO 10 j = k + 1,n                     13
     w(j,1) = 0.d0                        14
     e(j) = a(k,j)                        15
10   s = e(j) * e(j) + s                  16
      :
    DO 30 j = k + 1,jm,4                  26
     CDOALL 20 i = k + 1,n                27
20     w(i,1) = w(i,1) +...
1             + e(j) * a(i,j)             28
30  CONTINUE                              29
      :
    DO 60 i = k + 1,n                     35
60   r = e(i) * w(i,1) + r                36
      :
    DO 240 k = n - 2,1, -1                62
      :
     DO 130 i = k + 1,n                   69
      w(i,2) = 0.d0                       70
130   a(i,k) = a(k,i) * h                 71
     DO 150 i = k + 1,im,4                72
      CDOALL 140 j = k + 1,n              73
140    w(i,2) = w(j,2) + ...              74
150  CONTINUE                             75
     DO 170 i = im + 1,n                  76
      CDOALL 160 j = k + 1,n              77
160    ...                               78
      :
```

(a)Default Passfile

```
      :
     cptmp_346(k) = s                     17
     CDOALL 10 j = k + 1,n                18
      w(j,1) = 0.d0                       19
10   CONTINUE                             20
     DO 347 j = k + 1,n                   21
      e(j) = a(k,j)                       22
      cptmp_346(j) = e(j) * e(j)
1          +cptmp_346(j-1)                23
347  CONTINUE                             24

     DO 30 j = k + 1,jm,4                 35
      CDOALL 20 i = k + 1,n               36
       w(i,1) = w(i,1) +...
1            + e(j) * a(i,j)             37
20    CONTINUE                           38
30   CONTINUE                            39

     CDOALL 130 i = k + 1,n               81
      w(i,2) = 0.d0                       82
130  CONTINUE                             83
     DO 348 i = k + 1,n                   84
      a(i,k) = a(k,i) * h                 85
348  CONTINUE                             86

     DO 190 j = k + 1,im,4                96
      DO 180 i = k + 1,n                  97
       a(i,j) = a(i,j)
1          + w(j,2) * a(k,i)             98
      :
180   :                                  101
190  CONTINUE                            102
     DO 210 j = im + 1,n                 103
      DO 200 i = k + 1,n                 104
200    a(i,j) = a(i,j)
1          + w(j,2) * a(k,i)             105
210  CONTINUE                            106
220  DO 230 i = k + 1,n                  107
      a(i,k) = 0.d0                      108
230   a(k,i) = 0.d0                      109
240  CONTINUE                            110
250  gn = dabs(e(1))+dabs(w(1,1))       111
      :
     DO 260 i = 1,n10000 - 1             118
      w(i + 1,2) = w(i,1)                119
      gn = dmaxl(dabs(w(i,1))+...        120
260  CONTINUE                            121
      :
```

(b)Modified Passfile

```
      :
     CDOALL 10 j = k + 1,n                19
      w(j,1) = 0.d0                       20
10   CONTINUE                             21
     CDOALL 347 j = k + 1,n               22
      e(j) = a(k,j)                       23
347  CONTINUE                             24
     DO 390 j = k + 1,n2                  25
      cptmp_346(j) = e(j) * e(j)
1          + cptmp_346(j - 1)             26
390  CONTINUE                             27
      :
     CDOALL 130 i = k + 1,n               84
      w(i,2) = 0.d0d3                     85
130  CONTINUE                             86
     CDOALL 348 i = k + 1,n               87
      a(i,k) = a(k,i) * h                 88
348  CONTINUE                             89
      :
     CDOALL 210 j = im + 1,n             106
      CDOALL 200 i = k + 1,n             107
200    ...                              108
210  CONTINUE                            109
220  CDOALL 230 i = k + 1,n              110
      a(i,k) = 0.d0                      111
230   a(k,i) = 0.d0                      112
      :
     CDOALL 260 i = 1,n100000 - 1 122
      :
260  CONTINUE                            125
```

(c)Modificated Passfile and Hand Modification

図 14: Result of Parallelization