ハイパフォーマンス
コンピューティング 71－3
（1998. 5. 15）

共有メモリワークステーション向けCコンパイラの並列化機能による並列化

草野和寛† 佐藤三久†

共有メモリワークステーションにおいて、Cコンパイラの自動並列化機能、およびpragmaを用いたプログラムの並列化を行った。その結果、プログラムによってはCもFortranと同程度の性能向上が可能であることが確認できた。ポインタの依存関係が解析できないために並列化できない場合などで、pragmaを用いた並列化指示も有効である。しかし、コンパイラによってコンパイラへのヒントとして扱われるpragmaでは、ユーザ指示を用いても並列化されないことがあった。このため、pragmaによる並列化でもあまり性能が向上しない場合があることも確認できた。

Parallelization by compiler directives for the shared memory multi-processor

KAZUHIRO KUSANO † and MITSUHISA SATO†

This paper reports the result of some parallelization by automatic parallelizing of the C compiler for the shared memory multi-processor. In some benchmarks, the C compiler achives good speedup by the automatic parallelization. The pragma directive is effective when the compiler cannot recognize parallelism because of the undecidable dependencies. In some compilers, the pragma directives are not effective enough to control various kind of parallelization.

1. はじめに

近年のワークステーションやPCの上位機種は、共有メモリのマルチプロセッサ構成のものが一般になってきている。このため、商用コンパイラも並列化に関する機能を利用できるようになってきた。

これまで、並列化の研究は比較的解析しやすく、並列性を引き出しやすいアプリケーションが多いFortranを中心に行われてきた。C言語は、ポインタや手続き呼び出しが多いなどのために並列化の研究が遅れていたものの、並列化コンパイラが使えるようになってきている。そこで、まず本稿では、現在ワークステーション上で利用可能なコンパイラを用いて自動並列化の能力について調べることにする。

プログラムの並列化は、自動並列化で効率的な並列実行できるのが理想であるが、静的に解析するには限界がある。そこで、並列プログラミング言語や指示行(pragma)などの形でユーザが並列化に関するヒントや指示を行うことによる並列化が一般的になると考えられる。このような並列化に関するユーザ指示では、Fortranを中心にして分散メモリ向けのHPF[1]や共有メモリ向けのOpenMP[2]など標準化の動きがある。OpenMPでは、Fortran以外にCやC++に関してpragmaの標準化を検討しており、今年には最初の仕様が公開される予定になっている。

我々は、現在のCコンパイラのサポートしている並列化指示(pragma)の仕様を調査し、そしてそれを用いてベンチマークプログラムの自動並列化、およびpragmaを用いた並列化を行った。

今回調査したコンパイラとそのバージョンは以下の通りである。ただし、SGIのコンパイラに関しては仕様の調査のみで並列化のテストは行っていない。

(1) SUNWspro C Compiler SC4.2[3]
(2) Apogee Rel. 4.0 with KAP/C for Sparc 3.1[4]
(3) SGI Power C User's Guide(12/96)[5]

次章では自動並列化機能の評価を述べ、3章でC言語の並列化指示について述べた後、4章でその並列化指示を用いた並列化について述べる。そして、5章で考察した後、6章でまとめを行う。

2. 自動並列化機能の評価

2.1 プラットフォーム

並列実行のプラットフォームには、8プロセッサのSparc1000(Solaris 5.3)を使用した。コンパイラは前述のSUNとApogeeを用い、比較対象として示すFortranの性能には'SUN FORTRAN 77 3.0.1'を用いた。

また、今回のテストプログラムには数値計算のプログラムである、clinpack(n=100&1000)、および

† RWCPつくば研究センタ
RWCP Tsukuba Research Center

SPECfp95[6] の tomcatv と swim を用いた。Fortran で書かれている SPECfp95 は、f2c により自動変換したものと、手で C 言語に変換したものを用いている。

逐次実行 (S) と並列実行 (P) はそれぞれ表 1のコンパイルオプションを指定して実行した。

Fortran	S	-fast -O4
	P	-fast -O4 -parallel -reduction
C/SUN	S	-fast -xO5 -xdepend
	P	-fast -xO4 -xdepend -xparallel -xreduction
C/Apogee	S	-fast -O4 -Xkap
	P	-fast -O4 -Xkap=mp

表 1 コンパイルオプション

このオプションに加え、clinpack では n=100 において SUN コンパイラで -xunroll=4 を、Apogee コンパイラで -Wk,'-unroll=4' を指定した。n=1000 では、SUN コンパイラで -xunroll=16 を、Apogee コンパイラで -Wk,'-unroll=16' を指定した。

2.2 clinpack(n=100)

オリジナルは Fortran で書かれた浮動小数点演算のベンチマーク[7] で、このプログラムはその C 版である。以降で示す clinpack の性能は、サイズに関わらず全て最適化オプションの他に'-DDP -DROLL -DGTODay' を指定している。

このプログラムを自動並列化した場合の性能を、オリジナル (Fortran) の Linpack を f77 で自動並列化した結果と共に表 2に示す。これ以降の表で PE 欄に'S' とあるのが逐次実行の性能である。性能は全て Mflops であり、効率は並列版を 1 台で実行した性能を基準にした並列化効率 (スピードアップ) を表す。

	Fortran	C/SUN	C/Apogee
PE	性能 (効率)	性能 (効率)	性能 (効率)
S	21.94(-)	19.53(-)	12.84(-)
1	13.58(1.00)	13.33(1.00)	12.44(1.00)
2	13.58(1.00)	13.39(1.00)	12.33(0.99)
4	13.52(1.00)	12.94(0.97)	12.33(0.99)
8	13.57(1.00)	13.09(0.98)	12.05(0.97)

表 2 clinpack(n=100) の自動並列化

この結果自動並列化ではほとんど速度が向上していない。性能が向上していないのは、実行時間がかかる部分が並列化されていないことと、サイズが小さいためにループの繰り返し回数や計算処理が少ないことが原因であると思われる。

2.3 clinpack(n=1000)

次に、サイズの異なる clinpack(n=1000) を自動並列化した性能を表 3に示す。このサイズのベンチマークはソースの変更が許されているが、ここではソースに手は加えていない。

この結果、Fortran では 8 台で 2 倍弱の性能向上が見られるが、C の場合は台数を増やしても性能が向上していない。このサイズでは、ループの実行回数は十分に多いので、性能が向上しない原因は、実行時間がかかる部分の並列化ができていないためと考えられる。

	Fortran	C/SUN	C/Apogee
PE	性能 (効率)	性能 (効率)	性能 (効率)
S	6.10(-)	6.11(-)	6.44(-)
1	5.42(1.00)	5.30(1.00)	5.61(1.00)
2	7.04(1.30)	5.32(1.00)	5.63(1.00)
4	9.35(1.73)	5.28(1.00)	5.67(1.01)
8	10.17(1.88)	5.19(0.98)	5.60(1.00)

表 3 clinpack 1000 の自動並列化

2.4 tomcatv(SPECfp95)

このベンチマークは一つの手続きのみから成り、コモン変数もなく、並列化は容易なプログラムである。

まず、オリジナル (Fortran) プログラムと f2c を用いて変換した C プログラムの逐次実行とそれを自動並列化した性能を表 4に示す。SPECfp95(tomcatv,swim) の実行問題サイズは全て reference を用いている。

	Fortran	C/SUN	C/Apogee
PE	時間 (効率)	時間 (効率)	時間 (効率)
S	1093(-)	2349(-)	2376(-)
1	1141(1.00)	2360(1.00)	2468(1.00)
2	658(1.73)	1741(1.36)	1694(1.46)
4	429(2.66)	1445(1.63)	1359(1.82)
8	355(3.21)	1335(1.77)	1257(1.96)

表 4 tomcatv(Fortran と f2c) の自動並列化

f2c により変換した C プログラムは、逐次実行で Fortran の倍以上の実行時間がかかっている。また、台数を増やした時の速度向上も Fortran ほど出ておらず、8 台の並列実行でも Fortran の逐次実行より性能が悪い結果となった。この性能の差は、並列実行と判定されたループが Fortran より C プログラムの方が少ないためである。

次に、Fortran を手で C プログラムに変換して実行した結果を表 5に示す。この変換では、主要配列は大域変数として静的に確保し、多次元配列表記を用いた。

	C/SUN	C/Apogee
PE	時間 (効率)	時間 (効率)
S	2154(-)	2147(-)
1	2132(1.00)	2314(1.00)
2	1171(1.82)	1449(1.60)
4	710(3.00)	951(2.43)
8	552(3.86)	761(3.04)

表 5 tomcatv(手で C に変換) の自動並列化

この結果、SUN コンパイラでは逐次実行の性能が f2c により変換した場合よりもさらに低下して、Fortran の 2.8 倍程度の実行時間を要しているが、並列実行による速度向上は 8 台で 5 倍以上となっている。これに対して、Apogee コンパイラを用いた場合、逐次実行の性能は SUN コンパイラより良いが、並列化による性能向上

は逆に低くなっている。

各々の診断メッセージで並列実行ループを確認したところ、SUN コンパイラで並列実行と判定されたループは Fortran と同じであった。一方、Apogee コンパイラは SUN コンパイラで並列実行されているループの一部が、逐次実行用の最適化や、ループ分割といったによる変形が行われていた。この違いが並列化した時の性能向上の差になっていると思われる。

2.5 swim(SPECfp95)

このプログラムは気象予測で使われるアルゴリズムに基づくベンチマークで、配列の隣接要素を用いて値を更新していくプログラムである。

オリジナル (Fortran) と f2c で C に変換したプログラムを自動並列化した性能を表 6示す。ここで示している性能は入力ファイルに'swim.in' を用いている。

	Fortran	C/SUN	C/Apogee
PE	時間 (効率)	時間 (効率)	時間 (効率)
S	1009(–)	1421(–)	1299(–)
1	1024(1.00)	1425(–)	1323(–)
2	528(1.94)	–(–)	–(–)
4	273(3.74)	–(–)	–(–)
8	171(5.99)	–(–)	–(–)

表 6 swim(Fortran, f2c) の自動並列化

この結果、Fortran は自動並列化で非常に良い速度向上を得られている。これと対照的に、f2c で変換した C プログラムでは、自動並列化では並列実行可能な部分がないと判定され、並列実行されなかった。このため、f2c により変換した C プログラムは 1 台で実行した性能のみ示している。

並列化されていない原因を診断メッセージなどで見ると、依存関係があると判定されていることがわかる。この依存関係は、Fortran のコモンブロックがコモン変数を要素とする構造体となっているためであった。

次に、このプログラムを手で C プログラムに変換する際に、f2c において問題となったコモン変数は以下のようにした。

まずプログラム中のコモンブロックは、全ての手続きで同じ宣言がされており、equivalence や領域を越えるアクセスなどもない。したがって、このプログラムではコモン変数をそれぞれ別の変数として宣言しても問題はない。そこで、コモン変数はそれぞれ個別に大域変数として Fortran と同様に多次元配列として宣言した。

このようにした C プログラムの逐次、自動並列による性能を表 7に示す。この結果を見ると、逐次実行の性能は f2c で変換したものとほとんど同じ、あるいは性能が向上し Fortran とほぼ同じとなっている。しかし、コモン変数の宣言を変えたことにより、依存関係を判定できるようになり、自動並列化で Fortran と同程度の性能向上を達成している。

自動並列化の判定結果を診断メッセージで比較して

	C/SUN	C/Apogee
PE	時間 (効率)	時間 (効率)
S	1085(–)	1261(–)
1	1044(1.00)	1307(1.00)
2	548(1.91)	674(1.94)
4	305(3.43)	372(3.51)
8	201(5.20)	233(5.61)

表 7 swim(手で C に変換) の自動並列化

みると、SUN コンパイラは Fortran と同じ判定結果となっている。一方、Apogee コンパイラは tomcatv と同様に、一部の並列実行可能なループが逐次実行用に最適化されていた。

3. C 言語の並列化指示

前記コンパイラの並列化関連の pragma で指定できる機能を表 8に示す。

機能	C/SUN	C/Apogee	SGI
逐次実行	○	○	○
並列実行ループ	○	○	○
変数の属性	○	–	○
配列の重複	○	○	○
手続き呼び出し	○	○	○
reduction	○	–	–
手続き間解析	○	○	○
診断メッセージ	○	○	○
インライン展開	–	○	○
リスト構造の並列実行	–	○	○
ユーザ定義タスク	–	–	○
実行制御	–	–	○
同期	–	–	○
排他制御	–	–	○
中間ファイル出力	–	–	○

表 8 各コンパイラの機能

3.1 SUN C コンパイラ

- 並列化の対象
  並列化の対象としてループ以外を指定することはできない。自動並列化で解析するループには、goto 文で構成されるループ構造も含まれている。
- 手続き呼び出しを含むループ
  オプションなどの指定がなくても手続き間解析を行い、依存解析や最適化を行う。このため、ユーザ定義の手続きを含んでいても自動並列化が可能である。また、ソース全てを一括コンパイルする場合と、分割コンパイルを行う場合に自動並列化や最適化の結果が異なることがある。しかし、組み込み関数 (abs など) の場合、Fortran では特に宣言をしなくても並列化できるのに対して、pragma を用いて宣言しておかないと並列化阻害要因となる。
- reduction 演算
  reduction 演算は、自動並列化の他にユーザが並列化を指示することができる。しかし、演算の種類の

指定はなく、総和演算しか並列化できない。

- 診断メッセージ
  自動並列化やpragmaによる並列化を行う際には、オプションを指定することで各ループに対する診断メッセージを出力できる。この出力には、ファイル名、ループの行番号、そのループに対する実行形態や依存関係などの情報が含まれている。この診断メッセージは、入力プログラムと対応させた形での出力機能がないため、ユーザが行番号を見て行う必要がある。また、同じループに対する複数のメッセージがまとめて出力されない上に、矛盾するメッセージが出力されることもあるため、やや使いにくいものとなっている。
- その他
  並列実行ループ内の条件文中で定義される変数がある場合、その値が逐次実行と同じことが保証されない。これは、条件文中の変数の値が、最後のループindexを実行したプロセッサの値となる仕様によるものである。しかし、変数に別の指示(pragma)を指定することで、値を保証する仕組みが欲しいところである。

3.2 Apogee C コンパイラ

Apogeeコンパイラの並列化はKAP/C[8]が行っている。KAP/Cは、KAI(Kuck & Associates, Inc.)が開発、製品化している、並列化と最適化を行うCのプリプロセッサであり、多くのプラットフォーム上で動作している。また、KAIはOpenMPの検討にも参加しており、Fortranはいちはやく OpenMPに対応した製品を発表している。本稿ではKAP/Cを含めてApogeeコンパイラと呼んでいる。

- 並列化の対象
  SUNコンパイラと同じくループ構造のみを並列化の対象としている。変数の属性(sharedなど)の指定を行うことはできない。
- 手続き呼び出しを含むループの並列化
  手続きに副作用がないことを指定する以外に、ループ単位に依存関係がないことを指定することができる。また、コマンドライン引数やpragmaの指定がなければ手続き間解析は行わない。
- reduction演算
  リストなどの特殊な並列性が指定できるにもかかわらず、reduction演算の並列化の指定を行うことができない。このため、reduction演算は処理系が認識して並列化する場合にのみ並列実行となる。
- 診断メッセージ
  診断メッセージはオプションの指定により、各手続き単位に出力する。出力するメッセージの種類もオプションによって指定する。このメッセージはcppによって変換した後のソースに対応しているため、オリジナルソース中にない変数に関するものも多く、メッセージの量が多く、見にくくなっている。さらに、ループに対する診断メッセージの表示フォーマットが通常の画面サイズ(80カラム)より幅が広くなる点もあり見にくい。しかし、変換結果など詳細な情報を表示することができる。

3.3 SGI C コンパイラ

SGIのCコンパイラでは、前に述べたApogeeコンパイラで利用できた並列化の指示がそのまま指定できる。それに加え、並列実行の制御などで新たなpragmaを使用することができるようになっている。追加された仕様は、ほぼOpenMP(Fortran)に対応する形になっている。

しかし、新たに加えられた仕様は並列化を行うプリプロセッサ部が認識しないため、プリプロセッサが出力したソースに指定しなければならない。入力ソースに新たに追加されたpragmaが指定された場合には、現在は警告メッセージが出力されて無視される。このため、これらのpragmaを用いて細かい並列実行制御を行うには、二種類の異なるソースに対して、それぞれ別のpragmaを指定する必要がある。

- 並列化の対象
  ループ以外に対して並列実行部分であることを指定できる仕様が加えられている。また、並列実行部分の変数に属性(sharedなど)を指定できる。
- reduction演算
  reduction演算の指定は行えない。
- その他
  同期や排他制御など並列実行を制御する指示が加えられている。post-wait形式の同期も指定可能となっている。この他、プリプロセッサ部で並列化したソースを出力することができる。

3.4 OpenMP との比較

SUN/Apogee/SGIのコンパイラそれぞれの並列化の仕様を、OpenMPの指示行を比較してみると、以下のような違いがある。

- reduction演算
  ApogeeコンパイラとSGIコンパイラはreductionに関する指定ができない。一方、SUNコンパイラではreductionの指定が可能であるが、演算の種類に関する指定はない。
- 並列実行部の変数に指定できる属性が少ない。
- sectionsやparallell sectionsに対応するものがない。
- SUNコンパイラはループに対してしか並列実行の指定ができない。
- SUNコンパイラにはsingleなど実行プロセッサ制御の指定ができない。

4. pragmaを用いた並列化

次に、テストプログラムにpragmaを加えて並列化を行った場合の性能を示す。

4.1 clinpack(n=100)

このプログラムで最も実行時間の長い手続きが'dgefa'であることがprofile情報からわかる。そして、この手続きとここから呼び出される手続きを含めると、プログラム実行時間の大半(8割程度)を占めることがわかる。しかし、この手続きのループは、手続き呼び出し(daxpy)があるために自動並列化では逐次実行と判定されている。しかし、実際にはこのループは依存関係がなく、並列実行可能であるので、これを並列化するため、以下に示すpragmaを加えた。SUNコンパイラでは、手続き呼び出しに副作用がないこと、およびループを並列実行することを指示するpragmaを加える(図1)。

```
#pragma no_side_effect (daxpy)
#pragma MP taskloop
  for(j = k+1 ; j < n ; j++){
    t = a[lda*j+l];
    if ( l != k ){
      a[lda*j+l] = a[lda*j+k];
      a[lda*j+k] = t;
    }
    daxpy(n-(k+1),t,&a[lda*k+k+1],1,
          &a[lda*j+k+1],1);
  }
```

図1 SUNコンパイラでのpragmaによる並列化(clinpack)

Apogeeコンパイラでは、手続き呼び出しを含むループを並列実行するpragmaを加える(図2)。

```
#pragma _KAP concurrent call
  for(j = k+1 ; j < n ; j++){
    t = a[lda*j+l];
    ....
  }
```

図2 Apogeeコンパイラでのpragmaによる並列化(clinpack)

表9がpragmaを入れて並列化したプログラムの性能である。

	C/SUN	C/Apogee
PE	性能(効率)	性能(効率)
1	9.58(1.00)	12.07(1.00)
2	15.37(1.60)	11.62(0.96)
4	22.28(2.33)	11.33(0.94)
8	23.60(2.46)	11.49(0.95)

表9 clinpack(n=100)のpragmaによる並列化性能

この結果、SUNコンパイラでは並列実行により速度が向上している。しかし、並列実行の1台での性能が逐次実行の半分以下になっているため、4台以上で実行した場合にのみFortranの逐次実行を上回る結果となる。一方、Apogeeコンパイラでは、pragmaを用いて並列化した結果、自動並列化の場合より性能が低下した。これは、診断メッセージを見たところ、pragmaを用いて並列化の指定を行ったループが、ユーザの意図したように並列化されず、ループ分割や逐次処理用の最適化が行われていたためである。

4.2 clinpack(n=1000)

n=100の場合と同じpragmaを指定して並列化したプログラムの性能を表10に示す。

	C/SUN	C/Apogee
PE	性能(効率)	性能(効率)
1	5.41(1.00)	5.64(1.00)
2	10.77(1.99)	5.65(0.96)
4	24.21(4.48)	5.66(0.94)
8	68.65(12.69)	5.62(0.95)

表10 clinpack 1000のpragmaによる並列化

この結果、SUNコンパイラは非常に良い性能向上を示しているが、Apogeeコンパイラの性能は逆にやや低下している。このような結果になったのは、前に述べたように、並列化の指示を行ったループが逐次処理用に最適化されており、pragmaで意図した並列化がされていないためであると考えられる。

4.3 swim(SPECfp95)

Fortranをf2cで変換したCプログラムにpragmaを加えて並列化することを考える。このためには、コモン変数に対するアクセスに依存関係がないことを指示しなければならない。SUNコンパイラにおける指定例を図3に示す。SUNコンパイラでは図に示すようにループに対して依存関係がないことを指示できるが、Apogeeコンパイラでは変数に重なりがないことを列記しなければならない。

```
#define p_ref(a1,a2) _BLNK1.p[a2*513+a1-514]
  ...
#pragma nomemorydepend
#pragma ML taskloop
  for(j = k+1 ; j < n ; j++){
    cv_ref(i1,j+1)=(p_ref(i1,j+1)
     +p_ref(i1,j))*(float)0.5*v_ref(i1,j+1);
    ....
  }
```

図3 pragmaによるの並列化指示(SUNコンパイラ)

このようなpragmaを加えて並列化を行ったが、コンパイラの診断メッセージをみると、やはり逐次実行となっていた。このような判定がされた原因の一つは、やはりコモン変数が構造体の要素になっているためであると考えられる。これに加えて、Fortranの二次元配列を一次元化してアクセスしているため、アクセス領域の重なりを判定することが難しくなっていることも考えられる。

5. 考　　察

SUNコンパイラとApogeeコンパイラそれぞれを用いて得られた速度向上のグラフを図4と図5にそれぞれ示す。

次に、Cプログラムの並列化において感じた問題点について述べる。

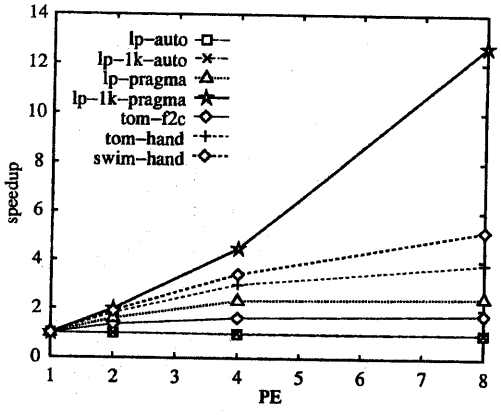

図4 Sunの速度向上

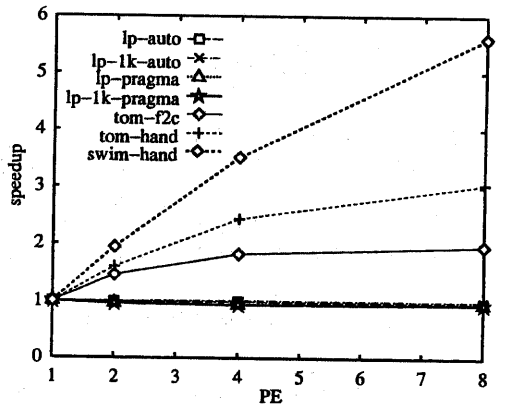

図5 Apogeeの速度向上

- ポインタや配列の依存関係があるループの並列化
  swimで示したように、pragmaで指示をしても並列化されないことがある。これは、コンパイラがpragmaをプログラムの解析における付加的情報として利用するため、指示通りの最適化を行うとは限らないためである。ユーザが意図した並列化を常に行うには、pragmaを他のプログラムと同様に扱うことが必要になるが、そうなると従来のpragmaの位置づけと異なってしまう。並列化指示の方法やpragmaの扱いに関しては、今後さらに研究が必要である。
- reduction演算
  Apogeeコンパイラではreduction演算の指定ができず、SUNコンパイラも演算の種類は指定できない。しかし、静的な解析で並列化できるreduction演算は限られているので、ユーザによる指示は並列化で性能を向上させるために非常に重要である。
- 診断メッセージ
  並列化作業において、診断メッセージは非常に重要である。特に、ループに対する変形や最適化に関する情報、また並列化を阻害している要因などは、性能を向上させるためにpragmaを入れる際に不可欠である。しかし、現在の診断メッセージはコンパイラによって異なっていたり、入力ソースと対応はユーザが取らなければならないことなど、まだ使いにくい点が多い。

6. おわりに

本稿では、Cコンパイラの自動並列化機能、および並列化指示(pragma)の仕様の比較と、それを用いたプログラムの並列化に関する評価に関して述べた。SUNワークステーション上の評価で、自動並列化のみによる性能向上は難しかった。また、ユーザ指示(pragma)を加えた場合でも、意図した通りに並列実行されずに、性能が向上しないことがあることがわかった。

最近リリースされたCコンパイラ(PGI)では、本稿で使用したclinpack(n=100)の自動並列化で、PentiumProのSMP(4CPU)で2倍強の性能向上が得られることが報告されている。

今後は、ここで調査した並列化に関するpragmaの仕様を元に、現在我々が開発しているSMPクラスタシステムCOMPaS[9]をターゲットにした並列化指示の仕様を決定し、それを入力とする共有メモリ用並列化Cコンパイラを開発していく予定である。

参考文献

1) Koelbel, C. H., Loveman, D. B., Schreiber, R. S., Steel Jr., G. L. and Zosel, M. E.: *The High Performance Fortran handbook*, The MIT Press, Cambridge, MA, USA (1994).
2) *OpenMP Fortran Application Program Interface Ver 1.0* (1997).
3) Sun Microsystems, Inc.: *C User's Guide* (1995).
4) Apogee Software Inc.: *User's Manual for Apogee compilers Rel. 4.0* (1996).
5) Silicon Graphics, Inc.: *SGI Power C User's Guide(12/96)* (1996).
6) : http://www.specbench/org/.
7) Dongarra, J. J.: The Complete Linpack Report, Technical report, University of Tennesse (1994).
8) Kuck and Associates, Inc.: *KAP for Apogee-C User's Guide Ver. 3.1* (1995).
9) Tanaka, Y., Matsuda, M., Ando, M., Kubota, K. and Sato, M.: COMPaS: A Pentium Pro PC-based SMP Cluster and its Experience, *IPPS Workshop on Personal Computer Based Networks of Workstations*, pp. 486–497 (1998).