ハイパフォーマンス
コンピューティング 76－2
（1999．5．14）

OpenMP コンパイラの試作と評価

草野和寛† 佐藤三久† 佐藤茂久†

共有メモリのSMP構成の計算機が一般に利用可能になってきつつある。加えて、共有メモリの計算機で並列処理を行うためにOpenMPが提案され、注目を集めている。そこで我々は並列化指示インタフェースとしてOpenMPを受け付け、Cプログラムの並列化を行うOpenMPコンパイラと実行時ライブラリを試作した。OpenMPを用いていくつかのベンチマークプログラムの並列化を行い、それを用いて我々が開発したOpenMPコンパイラの評価を行った。その結果、8CPUを用いた時にNPB CGでマルチスレッドで記述したプログラムとほぼ同様の最大6倍の性能向上を得ることができた。

RWC OpenMP Compiler Prototype and Preliminary Evaluation

KAZUHIRO KUSANO ,† MITSUHISA SATO † and SHIGEHISA SATOH†

We are developing RWC OpenMP C compiler and runtime libraries. OpenMP is a proposed standard interface which parallelize a program for shared memory multi-processor. This paper describes the RWC OpenMP compiler and preliminary performance of some benchmark programs, using OpenMP directives. The results show the parallel programs using OpenMP directives achieves as good speedup as its multi-threaded version.

1. はじめに

近年の計算機技術の向上は目覚ましく、個人で利用するPCやWSも複数の高性能なCPUと大容量のメモリを搭載するようになっている。高性能な計算機も、従来からの大型機に加えて、多数のプロセッサを結合したMPPや、安価なPCを結合したPCクラスタなどが多数作られるようになっている。サーバタイプのPCやWSは、複数のプロセッサを搭載した共有メモリのSMPが一般的になっており、並列処理を行うハードウェア的なプラットフォームは整ってきている。しかし、並列処理を行うために必要な、既存の逐次プログラムの並列化や、並列プログラミング言語を用いたプログラムの開発は困難な作業であるため、並列化されたソフトウェアは依然として少ない。

近年、SMPを対象にしたプログラムの並列化インタフェースとして提案されたOpenMP[1)]が注目を集めている。OpenMPは、データ並列を利用した並列化が可能なループや同期が必要な位置をコンパイラに指示するためのフォーマットやその内容を共通化し、可搬性の高い並列プログラムを開発することを目的としている。仕様の検討には多数のベンダやユーザが参加しており、これまでそれぞれのコンパイラで異っていた並列化指示が統一されることを期待されている。

† 新情報処理開発機構つくば研究センタ
RWCP Tsukuba Research Center

現在我々は、このOpenMPを並列化指示インタフェースとした並列処理環境をSMPクラスタ上での環境として構築することを考え、まずSMPマシン上で動作するOpenMPコンパイラと実行時ライブラリを研究開発している。本稿では、そのOpenMPコンパイラの概要とその評価について述べる。

以下、2章ではOpenMPとそれを取り巻く環境について述べる。3章では、我々が開発しているOpenMPコンパイラの構成と各部の概要を述べる。4章ではOpenMPコンパイラの性能を測定するためにOpenMPで並列化したベンチマークとその結果に関して述べる。そして、最後の5章でまとめと今後の課題をあげる。

2. OpenMP

OpenMPは共有メモリ計算機を対象にした、逐次プログラムの並列化に関する指示を行うための並列化指示インタフェースである。OpenMPは1997年のSC'97においてFortran版の仕様[1)]が公開され、さらに翌年のSC'98でC/C++版の仕様[2)]が公開された。

OpenMPの指示は、データ並列を利用した並列実行を行うループの指定と並列実行を制御する指示から成っている。この他、ユーザが定義したプログラム部分の並列実行の指定も可能である。これらの指示は、FortranやCの文法の拡張ではなく、コメントあるいはプラグマの一部として定義されているため、並列化指示を加えた後でも、逐次実行するようにコンパイル可能である。コ

ンパイラに対する並列化指示以外に、実行時ライブラリと環境変数に関して定義されている。

OpenMP と同様の並列化指示の仕様として、分散メモリマシンを対象にプログラムの並列化を行うために提案された HPF[7] がある。HPF はプログラム中のデータを分散メモリ上に配置する方法を指定し、並列ループの検出などは処理系にまかされている。一方、OpenMP は共有メモリを対象としていることもあり、データ配置に関する指定はなく、並列ループなどの並列実行の指定が中心となっている。HPF は Fortran のみの仕様であるのに対して、OpenMP は Fortran に加えて C/C++ にも対応している。

2.1 OpenMP を用いた並列化の特徴

OpenMP の並列実行は fork-join 型の実行モデルを採用している。このため、プログラム実行中に並列実行部分の指定があった時点で、複数のスレッドを生成し、並列実行部分が終了した時点でマスタ以外のスレッドは消滅して元の 1 スレッドとなる。

OpenMP では、プログラムの整合性や並列化可能性などは、全てユーザが保証しなければならない。このため、OpenMP コンパイラはループの依存関係やデッドロック等のチェックを行う必要がなく、プログラム中に指示した通りにプログラムの並列化を行うことが可能となっている。OpenMP の指示は、コンパイラに対するヒントとして扱われていた並列化指示とは異り、並列実行を記述するプログラミング言語であるとも言える。

OpenMP では、並列実行の開始と並列実行ループの指示を別にすることが可能である上に、それらが並列実行される部分から呼び出される手続きにあってもよい仕様 (orphan directive) となっている。このような場合、並列実行ループの指示は、並列実行している手続きから呼ばれた時にのみ有効となる。

2.2 関連研究

Fortran 版の OpenMP は、その仕様が確定した後、仕様検討に参加していた企業から製品化のアナウンスがあり、いくつかは既に発売されている。C と C++ に関しても、仕様の発表と同時に各企業から製品化のアナウンスはあったものの、仕様の決定が遅かったこともあり、一般に利用可能な処理系はまだ存在しない。

Fortran 版の OpenMP をソフトウェア分散メモリ環境で動作させる研究が Rice 大で行われている[3]。OpenMP と HPF を統合したプログラミングの検討が行われている[6]。OpenMP と MPI を用いて、SMP クラスタのメモリ階層を効率的に利用するプログラミング手法の研究がある[4]。Fortran に OpenMP 指示を付加したプログラムを並列化するフリーのトランスレータ形式の処理系も開発されている[5]。

3. RWC OpenMP コンパイラ

3.1 概　　要

OpenMP を並列化インタフェースとした並列処理環境として、SMP マシンをターゲットにした OpenMP コンパイラとその実行時ライブラリを開発している (図 1)。

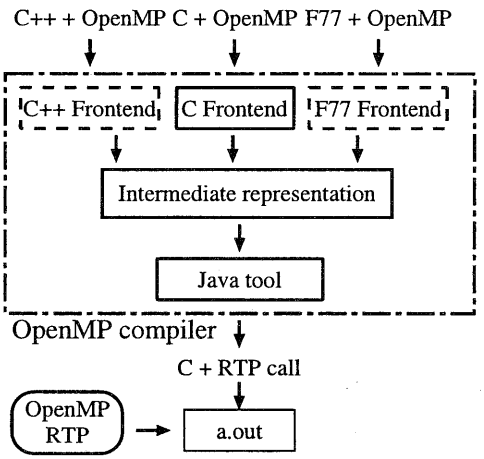

図 1　OpenMP コンパイラの構成

OpenMP コンパイラは、OpenMP 指示を含む C プログラムを入力として、それをライブラリ呼び出しを含む並列プログラムに変換する。変換されたプログラムは、実行時ライブラリとリンクして SMP マシン上で並列実行される。OpenMP コンパイラは、移植性を高くすること、また将来並列化に関する機能拡張を容易に行うことができることを考慮して開発した。

3.2 動作環境

現在の開発プラットフォームは、SUN Enterprize450(4CPU) Solaris 2.6 と Pentium II Xeon SMP 上の Linux 2.2.5 を用いている。並列実行のスレッド生成は、実行時ライブラリのオプションで Solaris スレッドおよび POSIX スレッドの両方に対応可能としている。また、スレッド生成時にプロセッサを固定して生成することができる Solaris スレッドの場合には、その機能を利用してスレッド生成を行っている。

3.3 フロントエンド

RWC OpenMP コンパイラのフロントエンドは、現在 OpenMP 指示を含む C プログラムを入力として受け付ける。対応している並列化指示は、C/C++ 版 OpenMP 仕様[2] で定められている仕様である。フロントエンドの入力ソースは、C プリプロセッサによる処理が済んでいることが前提となっている。フロントエンドは、OpenMP 指示を含む C プログラムを、等価な中間

表現に変換して、テキスト形式でファイルに出力する。フロントエンドでは、OpenMP の指示は記述フォーマットのチェックのみを行い、C の実行文と同様の構文木を生成している。中間表現のファイルは、OpenMP の指示に基づいた並列化を行うコンパイラツールキットに対する入力となる。

OpenMP コンパイラの中間表現は、変数の型情報、グローバルな宣言情報、および実行文を保持する構文木の 3 つの部分から成っている。この中間表現は、ファイルにテキスト形式で入出力できること、独自の拡張機能や C 以外の言語への拡張を容易に行えることを考慮して設計したものである。現在の C フロントエンド以外に、同じ中間表現を生成する Fortran77 および C++ のフロントエンドを現在開発中である。

3.4 コンパイラツールキット

フロントエンドで生成した中間表現ファイルを入力として、OpenMP 指示の解釈、およびその解釈に基づいたプログラムの変換、およびプログラムの解析機能を持つコンパイラツールキットを開発している。このツールキットは、可搬性の高さなどを考慮して Java を記述言語として採用している。

コンパイラツールキットは、プログラムの中間表現をファイルから内部データに読み込んだ後、そこに含まれている OpenMP 指示の解釈を行う。そして、解釈した OpenMP の指示に基づいて、入力プログラムを実行時ライブラリ呼び出しを含む並列プログラムに変換する。図 2に OpenMP で並列実行 (parallel) を指定した部分を変換したプログラム例を示す。プログラムの変換は、ツールが保持している中間表現のデータ構造を変更することで行う。この変換には、並列実行部分を新たな手続きとすることや、新たに生成した手続きを呼び出すコードの挿入、ライブラリ呼び出しの生成などが含まれている。最後に C ソースの出力部を用いて、変換した中間表現からライブラリ呼び出しを含む C プログラムが生成される。

コンパイラツールキットの解析機能では、フロー解析、変数の参照関係である UD-chain(SSA) の生成、およびアクセス範囲情報の解析を行っている。これら解析情報は、プログラムの変換時に利用される他、以下にあげるプログラムの最適化で利用する予定である。

- 不要な同期の削除
- 不要な終値保証コードの削除
- private 変数の認識

3.5 実行時ライブラリ

OpenMP コンパイラの実行時ライブラリは、OpenMP 指示に基づいて変換した並列プログラムで呼び出す実行時ライブラリと、OpenMP で定義されている実行時ライブラリから成っている。変換したプログラムで用いるライブラリには、並列実行ループの担当範囲を得るライブラリやバリア同期を行うライブラリ、そして OpenMP での並列実行環境を初期設定するライブラリがある。

```
void __ompc_func_6(__ompc_args)
  void  **__ompc_args;
{
  auto double **_pp_ptz;
  auto double **_pp_pty;
  auto double **_pp_ptx;
  auto double **_pp_ptolrsd;
  auto double *_pp_dt;
  (_pp_ptz) = ((double **) *(__ompc_args+0));
  (_pp_pty) = ((double **) *(__ompc_args+1));
  (_pp_ptx) = ((double **) *(__ompc_args+2));
  (_pp_ptolrsd) = ((double **) *(__ompc_args+3));
  (_pp_dt) = ((double *) *(__ompc_args+4));
  {
    /* 並列実行コード */
  }
}
main(){
  ...
  {/* parallel の指定個所 */
    auto void *__ompc_argv[5];
    (*(__ompc_argv+0)) = ((void *) &ptz);
    (*(__ompc_argv+1)) = ((void *) &pty);
    (*(__ompc_argv+2)) = ((void *) &ptx);
    (*(__ompc_argv+3)) = ((void *) &ptolrsd);
    (*(__ompc_argv+4)) = ((void *) &dt);
    _ompc_do_parallel(__ompc_func_6,__ompc_argv);
  }
  ...
}
```

図 2 OpenMP コンパイラによるプログラムの変換例

並列実行環境の設定の一部として、並列実行に用いるスレッドの生成を行う。スレッド生成は、プログラムの先頭でただ一度、利用可能なプロセッサ数と同じ数生成する。スレッドを生成した後は、マスタ以外のスレッドは並列実行の開始まで待機状態となる。

実行時ライブラリは、利用するスレッドライブラリとバリア同期の種類をコンパイル時に選択することができる。スレッドの生成には Solaris スレッドと POSIX スレッド、排他制御にはシステムが提供している mutex_lock 関数と、我々が作成した排他制御関数を選択できる。バリア同期には、1-read/n-write のビジーウェイトを用いている。Solaris スレッドを用いてる時には、スレッドを CPU に bind する機能を利用している。

4. 評　　価

本章では、OpenMP コンパイラと実行時ライブラリの性能をベンチマークを用いて評価した結果について述べる。

4.1 プラットフォームとベンチマーク

ベンチマークの実行は SUN SparcCenter1000(8CPU) Solaris2.5.1 を用いた。また、C コ

ンパイラはSUNWspro4.2のCコンパイラを用い、最適化オプションには全て'-fast -xO4'を指定している。実行時ライブラリは、Solarisスレッドおよび我々が作成した排他制御を用いるオプションの指定を行っている。

OpenMPコンパイラの評価に用いたベンチマークプログラムは、NPB1[8]のCG,BT,SPの3つを用いた。これらのオリジナルはFortranであるが、BTとSPはWisconsin大でC言語に書き換えたバージョンを使用し、CGは筑波大で作成したNPB1のバージョン[9]を使用した。CGは、OpenMPを用いた並列化に加えて、マルチスレッドで並列化したバージョンを作成し、OpenMPを用いた並列化との比較を行った。これらのプログラムの特徴は以下の通りである。

CG CG法を用いて大規模粗行列の最小固有値を求めるプログラムである。このプログラムは、粗行列ベクトル積を計算する部分が計算の主要部分を占めている。

BT,SP BTとSPは、いずれもmulti-partitionアルゴリズムを利用してNavier Stokes方程式を解くプログラムである。

4.2 OpenMPによる並列化

OpenMPを用いてベンチマークプログラムを並列化する際に、逐次プログラムのアルゴリズムを変更するようなことは行わず、OpenMP指示(pragma)を加えていくことで、プログラムの並列化を行った。また、その際にOpenMPの並列実行部分を表す'parallel'の指定はできるだけ少なくした。このため、並列実行ループの指定はOpenMPで'orphan'と呼ばれている指定方法となっている。このような方針で並列化したCGの一部を図3に示す。BTとSPでも、CGと同様の方針で並列化作業を行った結果、メインループを含む手続きに対して'parallel'を指定し、各手続きで並列実行可能なループに対して並列ループ'for'の指定を行った。

マルチスレッドを用いて並列化したCGのプログラムでは、各スレッドのIDを用いて担当範囲をプログラム中で計算するSPMD形式の並列実行を行っている(図4)。

4.3 性　　能

OpenMPを用いて並列化したベンチマークの性能を以下に示す。

表1にOpenMPで並列化したCG, BT, SPの性能(class A)を示す。表のPE欄に'S'とあるのは、逐次実行したプログラムの性能を表しており、値は全て実行時間で単位は秒である。図5に、並列版の1台を基準にして並列実行のスピードアップを表したグラフを示す。この結果を見ると、OpenMPによる並列化により8台で4.7倍から6倍近い台数効果が得られている。ただし、逐次版と並列実行版の1台を比べると、並列化によって手続きが増えていることや、多くのライブラリ呼び出しによる影響のためか、2割程度性能が低下している。こ

```
main(){
  ...
#pragma omp parallel private(resid,it,..)
  for(it=0; it<NITER; it++){
    resid = cgsol(...);
#pragma omp master
    printf(..,resid);
  }
 ...
}

double cgsol(...){
  ...
  for( it = 0; it < NITCG; it++ ){
    ...
#pragma omp for
    for(i = 0; i < cols; i++)
      z[i]=...;
    ...
    matvec(...);
    ...
  }
}

matvec(...){
  ...
#pragma omp for private(i,j,...)
  for( i = 0; i < nn; i++ ){
    ...
    for(j = start; j < end; j++)
      ...
  }
}
```

図3 OpenMPによるCGの並列化

PE	S	1	2	4	8
CG	281	310	158	83	52
BT	50466	54939	27575	15664	10661
SP	46452	58206	30332	18145	12342

表1 OpenMPで並列化したNPB1(classA)の実行時間(秒)

こので用いたBTおよびSPは、配列や構造体の複雑な依存関係のため、Cコンパイラの自動並列化では並列化可能と判定されるループがほとんどなく、性能向上も得られないプログラムである。

次に、OpenMPで並列化したCGと、マルチスレッドで並列化したCGを、SparcCenter1000で並列実行した場合の実行時間(秒)を表2に示す。表では、OpenMPで並列化したものを'OpenMP'、マルチスレッドで並列化したものを'MT'と表している。

PE	1	2	4	8
OpenMP	310.28	158.21	83.28	51.76
MT	290.98	148.85	78.92	47.55

表2 OpenMPとマルチスレッドの実行時間(秒)

OpenMPの実行時間を1にした場合のマルチスレッ

```
main(){
  ...
  for(t = 1; t < n_thd; t++){
    r = thr_create(...,cg_main,...);
    ...
  }
  ...
}

void *cg_main(id){
  ...
  for(it=0; it<NITER; it++){
    resid = cgsol_par(...);
    if(id == 0) printf(...,resid);
  }
  ...
}

double cgsol_par(...){
  ...
  s = id*N_blk;
  e=(id+1)*N_blk;
  if(e > n) e = n;
  b = e-s;
  cols = n;
  for(i = s; i < e; i++){
  ...
```

図4 マルチスレッドを用いたCGの並列化

ド、およびマルチスレッドのスレッド生成をOpenMPで行った場合の実行時間を図6に示す。この結果を見ると、マルチスレッドで並列化したプログラムの方が良い結果となっているが、OpenMPで並列化したプログラムでも、マルチスレッドと大きな違いはなく、ほぼ同じ速度向上が得られている。

4.4 考　　察

4.4.1 OpenMPによる並列化

OpenMPを用いてプログラムの並列化を行う作業は、基本的にCのpragmaを用いて並列化指示を行う場合とほとんど同じである。ただし、従来のコンパイラに対する並列化指示の多くは、コンパイラに対する解析のヒントであるため、必ずしもユーザの指示通りに並列化されるとは限らないのに対して、OpenMPの場合には指示した通りに並列化されるという違いがある。これは、エラーチェックや正当性の保証がユーザに委ねられているためであるが、その分ユーザの負担が増えているともいえる。この負担を軽減するためには、依存関係のチェック機能などを備えたツールなどを利用できる方が望ましい。しかし、この負担によって、ユーザの指示通りに並列化することができ、さらにインクリメンタルに並列化を進めることもできるという特徴を利用できるようになっている。特に、ユーザの指示通りに並列化できることは、複雑な依存関係がある部分を並列化する場合には非常に有用である。

OpenMPで並列実行(parallel)を指定された部分に並列実行ループ以外が含まれる場合、共有変数に対する

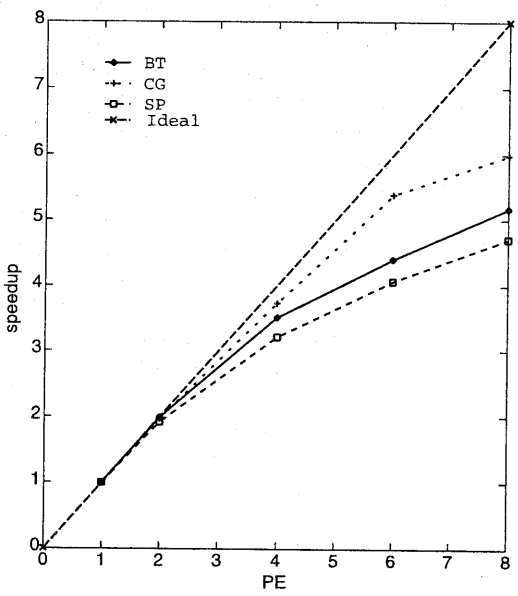

図5 OpenMPによるNPB1の台数効果

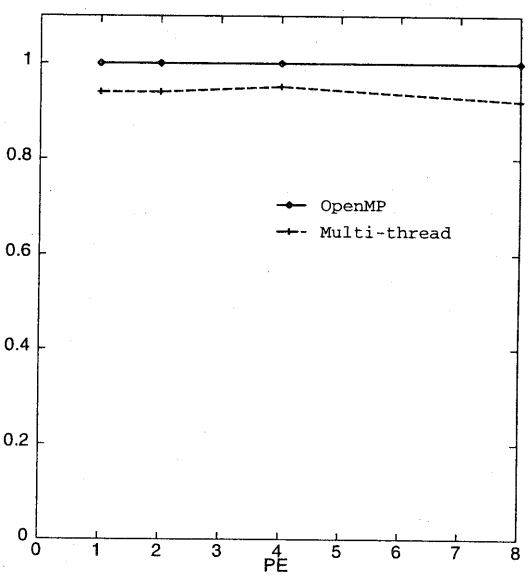

図6 OpenMPを基準にした実行時間比

書き込みでは'single'などを用いた適切な実行制御が必要となる。特に、OpenMPでは'orphan'と呼ばれる指定が可能であるため、並列実行部分から呼ばれることがある関数では注意が必要である。

OpenMPで並列実行の指定を行った場合、処理系により暗黙的な同期が挿入されている。このため、並列実行ループの後に同期をしなくても良いことがわかって

いる場合には、並列ループで'nowait' を指定した方がよい。特に、並列実行ループの粒度が小さい場合には、同期によるオーバヘッドが無視できないことがあるので、この指定によって性能が向上する可能性がある。しかし、適切な実行制御を行っていないと、正しい結果を得ることができないので、注意が必要である。

4.4.2 OpenMP 仕様の問題点と課題

数値計算プログラムでは reduction 演算の結果を配列に代入するコードが一般に用いられるが、現在の OpenMP の仕様ではこのようなループを並列化することができない。これは、reduction 演算の指定ができるのはスカラ変数のみであるという制限のためである。したがって、配列に対する reduction 演算を並列化するには、プログラムの書き換えが必要となる。また、頻繁に利用される reduction 演算として、最大値や最小値があるが、これも現在の仕様では並列化できない。

OpenMP の並列実行ループでは、ループの開始などの変化に関わらず特定の繰り返しを常に同じスレッドに割り付けることはできないために、並列ループ間で同期を行う必要が生じる場合がある。プログラムの動作としては同期を加えれば正しく動作するが、実行の効率を考えると、同期を加えなくても済む方が望ましい。コンパイラが不要な同期を削除する最適化を行う場合でも、ループの割り付け方法を言語仕様で指定することができれば、性能のチューニング手段として有効である。

同様の例として、シミュレーションの境界部分で異る処理を行うために、連続したループの初期値が異る場合がある。この場合には、同じ静的スケジューリングを適用しても、スレッドへの割り当てが変化する。しかし、この場合でも連続するループで、同じ index の範囲をスレッドに対して割り当てることができれば、同期を省略できる場合がある。このような場合には、コンパイラの最適化で同期を削除することは難しく、ユーザが何らかの指示を行う必要がある。

5. まとめと今後の課題

以上、本稿では RWC で開発している OpenMP コンパイラの概要とその性能評価を述べた。RWC OpenMP コンパイラは、フロントエンド、プログラム解析および OpenMP 指示変換部、実行時ライブラリから成っている。現在は C プログラムのフロントエンドのみ動作しているが、この他に Fortran77 と C++ のフロントエンドも開発中である。OpenMP コンパイラの評価として、ベンチマークプログラムを OpenMP で並列化した結果、概ね良好な速度向上を得られることが確認できた。また、NPB1 の CG について、OpenMP によって並列化した場合とユーザがマルチスレッドで並列化した場合、両者の性能にほとんど差がないことも確認できた。

マルチスレッドで並列化する場合と比較すると、OpenMP による並列化作業は容易であった。これは、逐次プログラムに対して指示を追加するのみであり書き換えは行わないこと、およびインクリメンタルに並列化していくことが可能であるためである。しかし、OpenMP を用いた並列化はユーザの責任で行うので、並列化の正当性はユーザが行う必要がある。このため、これまでのコンパイラに対する指示と比べて、並列化におけるユーザの負担が大きくなるので、その負担を軽減するためのツールが利用できることが望ましい。また、OpenMP の仕様で並列化できない場合があることや、性能のチューニングを考えた場合に欠ている部分があることもわかった。

今後は、他のベンチマークでも評価を行うとともに、性能を向上させる最適化手法の検討と、組み込みを行っていく。また、現在開発している分散メモリ対応機能を組み合わせ、OpenMP コンパイラを透過的に SMP クラスタシステム上で実行する並列処理環境の研究開発を予定している。さらに、並列化作業を支援するツールと組み合わせて、並列化作業を容易に行うことができるシステムの検討を行っていく。

謝辞 本研究に関する議論に参加して頂き、有益なアドバイスをして頂きました TEA グループ、ならびに並列分散システムパフォーマンス研究室の皆様に感謝致します。

参 考 文 献

1) http://www.openmp.org/
2) OpenMP Consortium, "OpenMP C and C++ Application Program Interface Ver 1.0", Oct, 1998.
3) H. Lu, Y. C. Hu and W. Zwaenepoel, "OpenMP on Networks of Workstations", SC'98, Orlando, FL, 1998.
4) http://www.wes.hpc.mil/news/SC98/HPCchallenge4a.htm
5) http://www.it.lth.se/ d92jh/odin.html
6) B. Chapman and P. Mehrotra, "OpenMP and HPF: Integrating Two Paradigms", Euro-Par'98, 650-658, 1998.
7) C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr. and M. Zosel, "The High Performance Fortran handbook", The MIT Press, Cambridge, MA, USA, 1994.
8) http://science.nas.nasa.gov/Software/NPB/
9) David Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, and H. Simon, "The NAS Parallel Benchmarks", RNR-94-007, NAS, 1994.