# Omni OpenMP コンパイラの性能評価

草野和寛† 佐藤茂久† 佐藤三久†

我々は共有メモリ向けの並列化インタフェースである OpenMP に対応した，Omni OpenMP コンパイラと実行時ライブラリを開発し，公開している．本稿は，Omni を SMP クラスタ向けの並列処理環境のベースとして利用するのに十分な性能であることを確認することを目的として行った，Omni OpenMP コンパイラの実行時ライブラリに対する評価とその評価結果について述べる．評価には，OpenMP 指示個別のオーバヘッドを測定するマイクロベンチマークと，計算機の基本性能を測定することを目的に開発された ParkBench の Low Level ベンチマークを用いた．マイクロベンチマークを用いた評価では，OpenMP の parallel によるオーバヘッドが，SUN（4 CPU）上の KAI コンパイラが 2 割程度，COMPaS-II（4 CPU）上の PGI コンパイラが約 2 倍，それぞれ Omni の性能を上回っていた．一方，ParkBench を用いた OpenMP の性能向上に関する評価では，SUN では KAI とほぼ同等の性能向上を得ることができた．この評価結果より，我々の開発した Omni OpenMP コンパイラは十分実用的であり，SMP クラスタ上の並列処理環境を構築するベースとしても十分な性能を有していることが確認できた．

# The Performance Evaluation of Omni OpenMP Compiler

Kazuhiro Kusano,† Shigehisa Satoh† and Mitsuhisa Sato†

OpenMP is a proposed standard interface which parallelize a program for a shared memory multi-processor. We are developing Omni OpenMP compiler and runtime libraries on the SMP machine. This paper describes evaluation of the Omni OpenMP compiler and its run-time library using some benchmark programs. The results show that the Omni achieves competitive performance with the KAI guidec, except the OpenMP parallel overhead. The parallelization overhead of the Omni is bigger than the one of the KAI about 20% on a four processor SUN450. On a COMPaS-II which is a PC with four Pentium II Xeon processors running Linux, we find the parallelization overhead of the Omni is twice as the PGI OpenMP compiler. The results using ParkBench benchmark program show the Omni can achieves almost the same performance compared to the KAI guidec compiler on the SUN450. The results show the Omni OpenMP compiler is effective to parallelize programs on an SMP machine.

## 1. はじめに

計算機ハードウェア，特に CPU の性能向上にともない，PC の計算能力も著しく向上している．さらに PC を多数結合した PC クラスタを高価な計算機に匹敵する高性能計算機として利用する例も増えている．また，SMP 構成の計算機が PC でも一般的になっており，様々な分野で並列計算機が利用可能になってきている．並列計算機を利用するには，既存の逐次プログラムを並列化するか，または新たに並列プログラムを開発する必要がある．しかし，プログラムの並列化作業，特に並列計算機の性能を十分に引き出す作業は，並列計算機に関する知識を必要とするうえに，非常に困難な作業である．このため，並列化されたプログラムは依然として少なく，並列計算機の利用を妨げる要因となっている．

プログラムの並列化インタフェースであるOpenMP[1)]が注目されている．OpenMP は，これまで並列計算機それぞれで異なっていた並列化に関する機能や指示フォーマットを統一し，移植性の高い並列プログラム開発を可能にすることを目的としている．仕様検討には多数のベンダが参加して，将来的な対応を表明しており，共有メモリ向け並列プログラムの標準として期待されている．この仕様は，データ並列を利用して並列実行可能なループ，同期位置などをコンパイラに指示するフォーマットや実行時ライブラリ，環境変数に関して定義をしている．OpenMP を用いて並列化したプログラムは，その結果の整合性，および並列化可能性はユーザが保証する必要がある．OpenMP は

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

Fortran 版の仕様書がまず 1997 年の SC'97 において公開され，翌年の SC'98 で C/C++版[2]の仕様書が公開された．Fortran 版はバージョン 1.1 で細かい修正がされた後，さらに新たな仕様を追加したバージョン 2.0 が SC2000 で公開されている．

我々は，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 クラスタ上で構築することを目標とし，まず SMP マシン上で動作する Omni OpenMP コンパイラとその実行時ライブラリを開発した[3),4)]．本研究は，この Omni OpenMP コンパイラが SMP クラスタ上で並列処理環境を構築するベースとして十分な性能と機能を有していることを確認することを目的として行った Omni の評価に関するものである．本稿では，Omni OpenMP コンパイラの実行時ライブラリに対する評価方法，およびその評価結果について述べる．ここで一般の科学技術計算では Fortran が利用されることが多かったが，近年では C 言語で書かれたプログラムも増えてきている．これに対して，並列化などにおいて C 言語での評価は少ないため，本稿の評価では C 言語を評価対象としている．

以下，2 章で Omni OpenMP コンパイラの概要，および Omni によって変換されるプログラムの特徴に関して述べる．3 章では，OpenMP 指示のオーバヘッドを測定するマイクロベンチマークを用いた Omni の基本性能の評価結果を述べる．そして，4 章では OpenMP を用いて並列化したループの性能向上に関して，Parkbench を用いて行った評価とその結果について述べる．5 章で関連研究を述べた後，最後の 6 章でまとめを行う．

## 2. Omni OpenMP コンパイラ

### 2.1 概　　要

我々は 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 Omni OpenMP コンパイラとその実行時ライブラリを開発した[3),4)]．Omni の動作条件は，Solaris（Sparc, x86），Linux（2.2.x）など，POSIX thread をサポートしており，かつ Java の実行環境が動作する環境である．

Omni OpenMP コンパイラは，**図 1** に示すとおり，フロントエンド部，プログラム解析および OpenMP 指示変換部，そして実行時ライブラリの 3 つの部分から成る．そして，入力である OpenMP 指示を含むプログラムを，ライブラリ呼び出しを含むマルチスレッド C プログラムに変換する．変換したマルチスレッドプログラムは，Omni の実行時ライブラリとリンクして SMP マシン上で並列実行する．

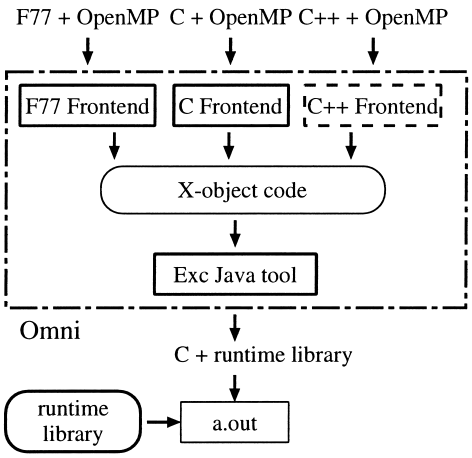


**図 1** Omni OpenMP コンパイラの構成
Fig. 1 Omni OpenMP Compiler.

Omni のフロントエンドは，仕様書第 1 版[2]に基づいた OpenMP 指示を含む C または FORTRAN77 プログラムを入力として受け付ける．C と FORTRAN77 のほかに，C++に対応したフロントエンドを開発中である．フロントエンドは，入力プログラムを変数の型情報，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 つの部分から成る，等価な中間表現（X-object）に変換して，テキスト形式でファイルに出力する．OpenMP の指示もフォーマットチェックのみを行い，実行文と同様の構文木などの情報に変換する．

次に，中間表現（X-object）を入力として，Exc Java tool が OpenMP 指示などの解析，および並列プログラムへの変換を行う．Exc Java tool では，ファイルから中間表現を読み込んだ後，コンパイラで一般に行われるフロー解析や，変数の参照関係である UD-chain（SSA）の生成，アクセス範囲情報の解析を行う．OpenMP の指示も，これら解析情報を利用して解釈を行い，内部の解析情報へ変換する．この解析結果に基づいて，入力プログラムと等価な中間表現を，ライブラリ呼び出しを含むマルチスレッド C プログラムへ，中間表現レベルで変換する．最後に，内部データである中間表現から C ソースの生成を行う．**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．

Omni OpenMP コンパイラの実行時ライブラリは，変換したマルチスレッドプログラムの制御などに利用する実行時ライブラリ，および OpenMP の仕様書で定義されている実行時ライブラリから成っている．実行時ライブラリでは，利用するスレッドライブラリ

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

**図 2** Omni により並列化したプログラム例
Fig. 2 Program fragment parallelized by using Omni.

（Solaris スレッドまたは POSIX スレッド）と排他制御で利用する関数（mutex_lock またはスピンウェイト）をコンパイル時に選択することができる．並列実行スレッドのバリア同期には，1-read/n-write のビジーウェイトのアルゴリズムを用いている[5),6)]．

Omni では，並列実行に用いるスレッドの生成は並列実行環境の設定の一部としてプログラムの先頭で 1 度だけ行い，プログラム実行中のスレッド管理は実行時ライブラリで行っている．この際に，最初にスレッドが生成された後，および並列実行部分以外では，マスタ以外のスレッドは待機状態となる．生成するスレッド数を CPU 数以下とすることも，我々が定義した環境変数の設定により可能であるが，CPU 数を超える数のスレッド生成は行うことはできない．また，プログラム実行の途中でスレッドを動的に生成する機能は現在サポートしていない．

## 3. Omni の基本性能評価

本章では，エジンバラ大学が開発，公開しているマイクロベンチマークを用いて評価した，Omni OpenMP コンパイラと実行時ライブラリの基本性能について述べる．

### 3.1 マイクロベンチマーク

マイクロベンチマーク[7)] は，エジンバラ大学が開発，公開している OpenMP のベンチマークで，OpenMP の指示それぞれに要するオーバヘッドを測定するものである．この測定では，あるループに OpenMP 指示を加えた場合と加えない場合の実行時間を測定し，その差分の OpenMP のオーバヘッドとしている．測定項目は OpenMP の指示である parallel や for，barrier など 10 項目，および並列実行ループのスケジューリング方法（static，dynamic，guided）と chunk サイズを指定した場合のオーバヘッドである．

### 3.2 評価環境

今回の評価は，以下にあげる計算機を用いて行った．

- SUN Enterprise450（4 CPU），Solaris2.6，SUNWspro4.2 C compiler，JDK1.2
- COMPaS-II（COMPAQ ProLiant 6500）（4 CPU），RedHat Linux 6.0 + kernel 2.2.12，egcs-1.1.2，JDK1.1.7

評価した OpenMP C コンパイラは，Omni 1.2 および，その比較対象として，上記計算機で利用できる以下の商用の OpenMP 処理系を用いた．

- KAI guidec[8)]（SUN）
  複数プラットフォームに対応した OpenMP 処理系である．OpenMP の並列化は，Omni と同様にいったん C へ変換するトランスレータ形式である．
- PGI pgcc[9)]（Linux）
  Intel プロセッサ上の Solaris/Linux/WinNT で動作する OpenMP 対応の C コンパイラである．今回の評価で用いた Linux 版のほかに，Solaris86/WinNT 版がある．

ここで，SUN 上の Omni では Solaris スレッドとビジーウェイトによる排他制御を，COMPaS-II 上の Omni は POSIX スレッドとビジーウェイトによる排他制御を利用している．このように商用の OpenMP 処理系と比較する形としたのは，Omni の性能を評価するのにも比較対象が必要であること，そして SMP マシン上において Omni を用いた場合に，商用のシステムと比較してどの程度の性能向上が得られるかを確認するためである．

### 3.3 マイクロベンチマークの評価結果

#### 3.3.1 SUN での評価

マイクロベンチマークで測定した，Omni 1.2 と KAI guidec の SUN 上における OpenMP 指示（10 種類）のオーバヘッド（単位：$\mu$s）を，それぞれ**図 3** と**図 4** に示す．ここで，バックエンドの C コンパイラには両方とも SUNWspro4.2 を用い，かつ同じ最適化オプション（-fast）を指定した．また，Omni の実行時ライブラリには，既定値である Solaris スレッドライブラリを利用し，排他制御にスピンウェイトを用いている．なお，本稿では示さないが実行時ライブラリに POSIX スレッドを用いた場合や，排他制御に mutex_lock を用いた場合もほぼ同等の性能であった．

図 3 と図 4 から，バリアなどにおいて Omni のオー

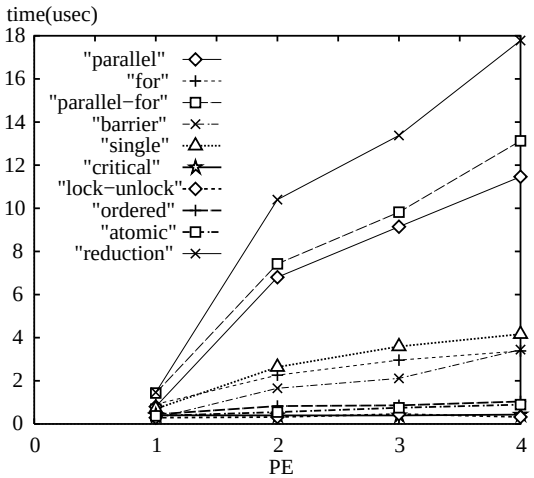


**図 3** Omni の基本性能（SUN）
Fig. 3 Overhead of Omni on SUN.

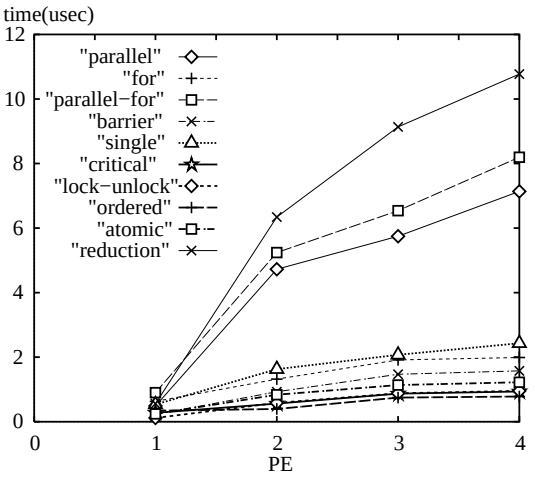


**図 5** Omni の基本性能（COMPaS-II）
Fig. 5 Overhead of Omni on COMPaS-II.

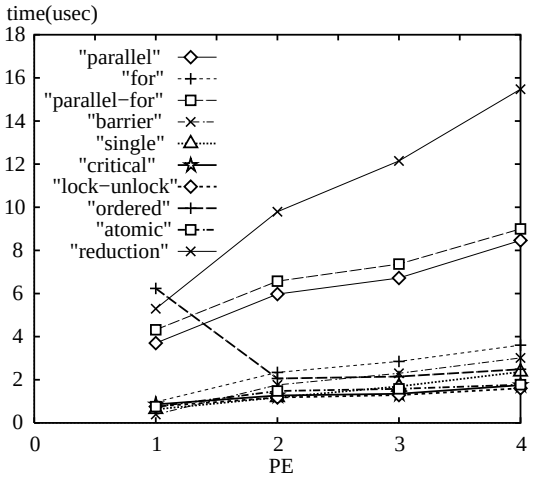


**図 4** KAI guidec の基本性能（SUN）
Fig. 4 Overhead of KAI guidec on SUN.

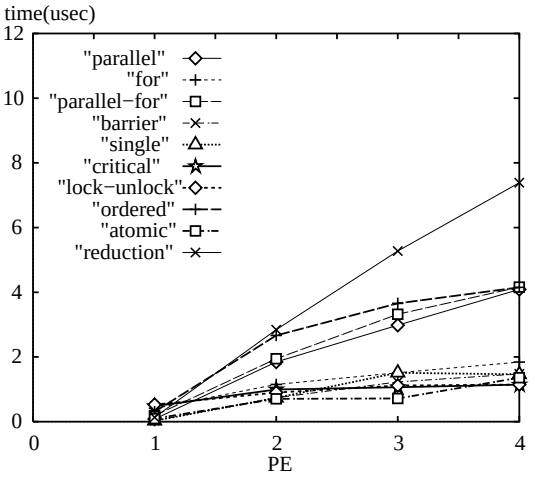


**図 6** PGI pgcc の基本性能（COMPaS-II）
Fig. 6 Overhead of PGI pgcc on COMPaS-II.

バヘッドは商用の OpenMP コンパイラである KAI の処理系と同等であることが確認できる．しかしながら，並列実行部分の指定である ‘parallel’ と ‘parallel-for’，それに ‘reduction’ 演算（parallel 含む）のオーバヘッドは KAI と比較して大きくなっている．これらすべてに含まれる ‘parallel’ オーバヘッドは，4 CPU で Omni は KAI の 1.5 倍程度であるのに対し，1 CPU では Omni が KAI の 1/4 程度である．このような結果となったのは，Omni の実行時ライブラリにおいて並列実行に利用するスレッドの操作でオーバヘッドが大きくなっているためと思われる．具体的には，実行時ライブラリのスレッド管理ではリスト構造を利用しており，スレッドの割当てや解放を行う操作に排他制御が必要となっている．このため，スレッドの処理が逐次化され，CPU 台数に比例したオーバヘッドを要しているためと思われる．Omni は，‘parallel’ のオーバヘッドはやや大きいものの，ほかは商用 OpenMP コンパイラと同程度の性能を達成しており，並列実行に用いるのに十分実用的であるといえる．

### 3.3.2 COMPaS-II での評価

COMPaS-II において，Omni と PGI pgcc を用いた場合の OpenMP 指示のオーバヘッドをそれぞれ**図 5** と**図 6** に示す．Omni の実行時ライブラリは，SUN と同じく既定値である POSIX スレッドとスピンウェイトによる排他制御を用いている．Omni のオーバヘッドは，前の SUN における結果と比較して 3 割程度速くなっており，CPU の速度差からいって妥当な値であるといえる．PGI の測定結果では，Omni で他より

**表 1** OpenMP parallel のオーバヘッド内訳（$\mu$s (%)）
Table 1 Breakdown of the OpenMP parallel overhead ($\mu$s (%)).

| PE | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 割当て | 0.02 (23) | 1.9 (30) | 3.6 (35) | 5.0 (36) |
| 解放 | 0.28 (30) | 2.2 (36) | 4.4 (43) | 7.0 (50) |

オーバヘッドが大きい 'parallel' などの項目に関しても半分程度と，非常にオーバヘッドが小さいことが分かる．この原因の 1 つとして，PGI の OpenMP コンパイラは，Omni とは異なりソースコードへの変換を行うことなく，マシンコードを生成する構成となっていることが考えられる．しかし，それを考慮しても PGI コンパイラのオーバヘッドは非常に小さい値である．PGI と比較するとオーバヘッドが大きいとはいえ，Omni のオーバヘッドも極端に大きいものではなく，プログラムの並列実行に利用することを考えると十分に実用的であると考える．

ここで，SUN と COMPaS-II の両方で大きなオーバヘッドを要しており，OpenMP で最も使用頻度が高いと思われる 'parallel' に関して，COMPaS-II において Omni の処理時間の内訳を調べた結果を**表 1** に示す．ここで，管理データの割当てに含まれているのは，実行時ライブラリで空きスレッドを割り当てて，最低限のデータに関して初期化を行う時間であり，管理データの解放に含まれているのはスレッドを解放し，そのすべてを空きスレッドとして登録するまでの時間である．表 1 では，各々の操作に要した時間（$\mu$s）とそれがオーバヘッド全体に占める割合（%）を示している．この結果，並列実行を管理するデータ構造の取得とその設定，および並列実行後に管理データをクリアする処理と同期待ちにかなりの時間を費やしていることが分かった．

Omni では，並列実行に利用するスレッドはプログラム実行開始時に生成した後，実行時ライブラリにおいて OpenMP プログラムへの割当てなどを行う．空きスレッドに対応した管理データはフリーリストでつながれており，並列実行の開始と終了にそのリスト構造からスレッドの割当てと解放操作を行う．このデータは共有資源であるため，その操作には排他制御が必要であり，スレッドの割当てや解放は逐次化される．また，この操作はネストした並列実行を考慮に入れて設計しているため，ネストした並列実行の指定を逐次化している現在の状況では冗長な操作を含む結果となっている．このようなオーバヘッドが，排他制御により処理が逐次化されることに加わった結果，表 1 に示すように CPU 台数にほぼ比例したオーバヘッドになったと考えられる．

このオーバヘッドを削減して性能を向上させるためには，ネストした並列化をサポートしない場合に，冗長となる処理を省略するように変更することが考えられる．さらに，データ構造もネスト並列化などを考慮したために複雑になっている部分もあるので，これらの最適化により性能が向上することが期待できる．これにより，並列実行の開始と終了においてスレッドの割当てと解放を行う部分のオーバヘッドが削減される可能性があると考える．

## 4. 問題サイズを変化させた場合の最大性能の比較

この章では，Parkbench [10] の Low level ベンチマークの 1 つである "rinf1" を利用して評価した OpenMP による性能向上についての評価結果を述べる．

### 4.1 Parkbench

この評価で用いた Parkbench は，並列計算機の性能指標となることを考慮し，フリーなベンチマークとして開発された．この中には大きく分けて 3 種類（application，kernel，low-level）のベンチマークがあるが，今回用いたのは基本的な計算機性能を測定する low-level に含まれるプログラム（rinf1）の一部である．このプログラムは，長さの異なる配列に対して様々な基本的な配列計算を行うループの実行時間から，最小自乗法を用いた計算により $r_\infty$ と $n_{1/2}$ を求める．$r_\infty$ は配列の要素数（ループ長）を変化させた場合に得られる最大性能であり，$n_{1/2}$ は $r_\infty$ の半分の性能を得られるループ長である．これらの値は，キャッシュ容量以下におけるものと，キャッシュ容量以上でのものの 2 種類の値が示される．この計算に用いるために，プログラムではループ長を変化させて，それぞれのループ長での性能（Rn）を測定する．今回の評価では，内側の計算ループを OpenMP 指示により並列化して，1/2/4 PE それぞれの実行時間から，逐次の場合と同様にして求めた性能（Rn）を示す．なお，このプログラムは Fortran で書かれているため，評価には同等の C プログラムに変換したものを用いている．また，評価に用いた計算機環境は 3.2 節に示したものである．

今回の評価には，rinf1 に含まれるカーネルループより，ループ長（n）を変化させ配列演算を行う kernel 3 と 6 の 2 つを用いた．**図 7** にこれらのカーネルループ部分および追加した OpenMP 指示を示す．

### 4.2 SUN における評価

SUN 上の Omni を用いて図 7 に示した kernel 3 と

```
/* kernel 3 */
 for( jt = 0 ; jt < ntim ; jt++ ){
   dummy(jt);
#pragma omp parallel for
   for( i = 0 ; i < n ; i++ )
     a[i] = b[i] * c[i] + d[i];
 }
/* kernel 6 */
 ....
#pragma omp parallel for
   for( i = 0 ; i < n ; i++ )
     a[i] = b[i] * c[i] + d[i] * e[i] + f[i];
...
```

**図 7** Rinf1 kernel 3 と 6 のループ
Fig. 7 Rinf1 program fragment of kernel 3 and kernel 6.

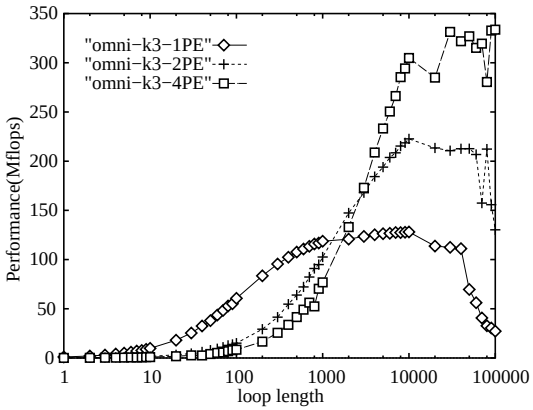


**図 8** Omni による kernel 3 の性能（SUN）
Fig. 8 Omni performance of kernel 3 on SUN.

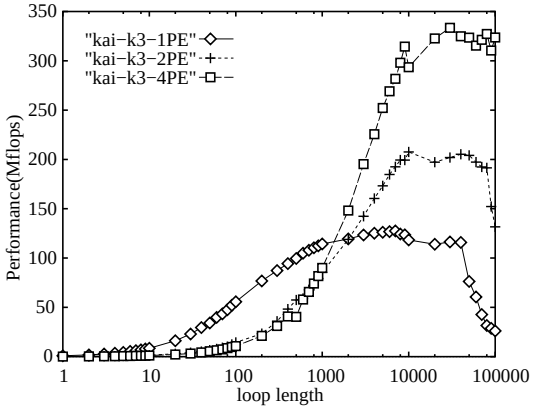


**図 9** KAI による kernel 3 の性能（SUN）
Fig. 9 KAI performance of kernel 3 on SUN.

kernel 6 の評価結果（性能：Mflops）を**図 8** と**図 9** に，KAI guidec を用いて評価した結果を**図 10** と**図 11** に示す．この結果から，どちらのループも OpenMP による並列化によりピーク性能は向上しており，十分なループ長があればループ本体の計算量が少なくても並列化による性能向上が期待できることが分かる．次

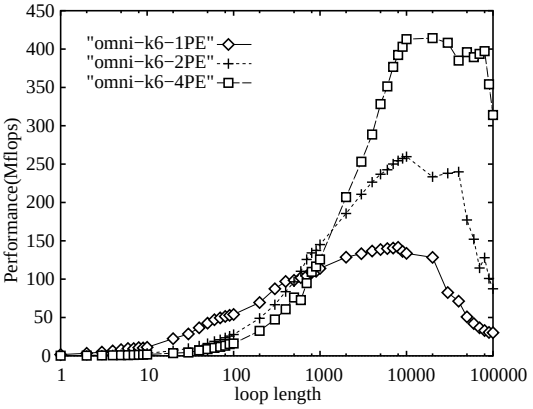


**図 10** Omni による kernel 6 の性能（SUN）
Fig. 10 Omni performance of kernel 6 on SUN.

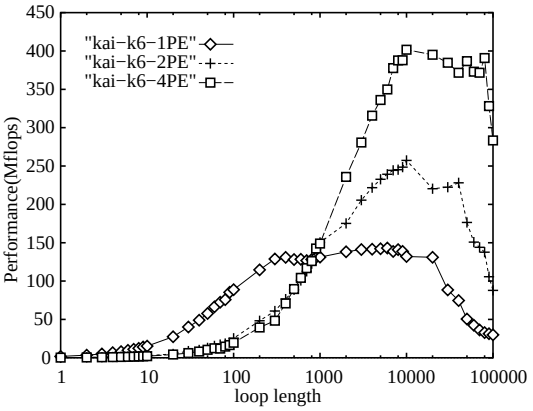


**図 11** KAI による kernel 6 の性能（SUN）
Fig. 11 KAI performance of kernel 6 on SUN.

に，並列化による性能向上は台数が増えるほど，立ち上がりが鈍くなっており，並列実行台数の増加以上にループ長，つまりループの計算量が必要であることが分かる．このループを並列化した場合に性能が向上する分岐点，つまり 1 PE のピーク性能を超えるループ長は，2 PE と 4 PE ともに kernel 3 ではおよそ 2000 であった．kernel 6 の場合でも同様に分岐点を見ると，ループ長が 800 から 1000 と短くなっているが，これはループに含まれている計算量が増加しているためである．

次に，Omni と KAI それぞれの SUN 上の評価で得られた $r_\infty$ と $n_{1/2}$ のキャッシュサイズ以下における値を**表 2** と**表 3** に示す．ここで示した値は，プログラム中で最小自乗法を用いて計算して出力される値であり，図 8 や図 9 の結果から読み取れる値とは異なるように見えるものもある．この結果からも，PE 数を増やした場合の性能（$r_\infty$）の向上は良好であることが分かる．$n_{1/2}$ の値から，十分な性能を引き出すため

**表 2** $r_\infty$ と $n_{1/2}$（Omni on SUN）
Table 2 $r_\infty$ and $n_{1/2}$ (Omni on SUN).

| | PE | 1 | 2 | 4 |
|---|---|---|---|---|
| Kernel 3 | $r_\infty$ | 129 | 191 | 332 |
| | $n_{1/2}$ | 117 | 549 | 2739 |
| Kernel 6 | $r_\infty$ | 141 | 249 | 366 |
| | $n_{1/2}$ | 144 | 630 | 811 |

**表 3** $r_\infty$ と $n_{1/2}$（KAI on SUN）
Table 3 $r_\infty$ and $n_{1/2}$ (KAI on SUN).

| | PE | 1 | 2 | 4 |
|---|---|---|---|---|
| Kernel 3 | $r_\infty$ | 119 | 206 | 330 |
| | $n_{1/2}$ | 48 | 1030 | 2080 |
| Kernel 6 | $r_\infty$ | 142 | 246 | 383 |
| | $n_{1/2}$ | 66 | 673 | 813 |

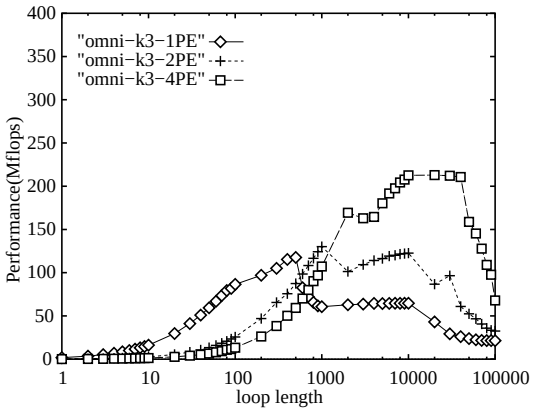


**図 12** Omni による kernel 3 の性能（COMPaS-II）
Fig. 12 Omni performance of kernel 3 on COMPaS-II.

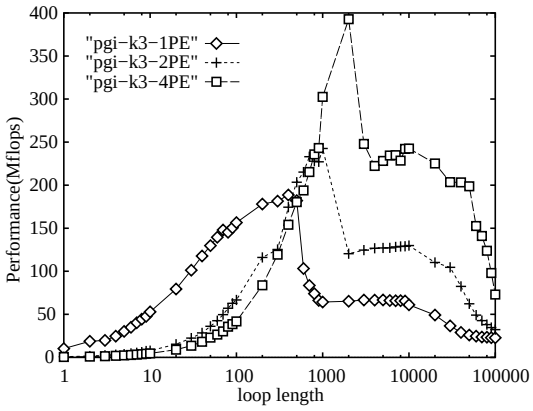


**図 13** PGI による kernel 3 の性能（COMPaS-II）
Fig. 13 PGI performance of kernel 3 on COMPaS-II.

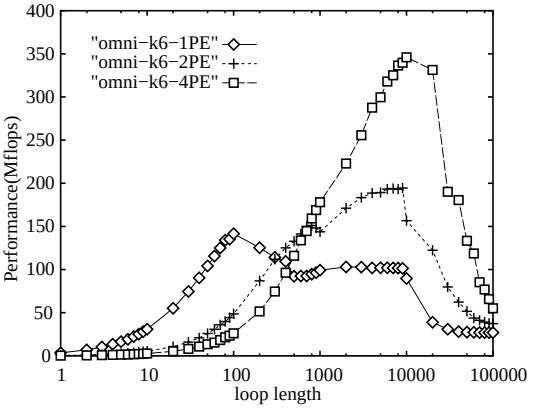


**図 14** Omni による kernel 6 の性能（COMPaS-II）
Fig. 14 Omni performance of kernel 6 on COMPaS-II.

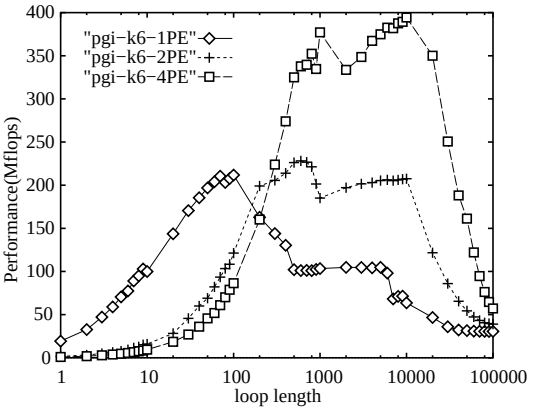


**図 15** PGI による kernel 6 の性能（COMPaS-II）
Fig. 15 PGI performance of kernel 6 on COMPaS-II.

に必要なループ長は，ループに含まれる計算量により変わるものの十分に長い必要があることが分かる．また，ここで使用したループに関しては，Omni と KAI でほぼ同等の性能が得られていることが確認できる．

### 4.3 COMPaS-II における評価

次に，同様にして kernel 3 と kernel 6 に関して COMPaS-II [11]で測定した Omni の性能を**図 12** と**図 13** に，PGI OpenMP コンパイラの性能を**図 14** と**図 15** にそれぞれ示す．この結果，COMPaS-II では SUN と比較して，並列化による性能向上は短いループ長で現れるが，その性能が低下するループ長も同様に短くなっており，ピーク性能を引き出すのが難しいことが分かる．また，1 PE のピーク性能は SUN とほとんど変わらないのに対して，台数を増やすことによるピーク性能の向上は SUN より低かった．特に 2 PE での性能向上が低く，kernel 3 では 1 PE とほぼ同じピーク性能しか得られていない．この結果から，COMPaS-II では SUN よりもピーク性能を得るには，

**表 4** $r_\infty$ と $n_{1/2}$ (Omni on COMPaS-II)
Table 4 $r_\infty$ and $n_{1/2}$ (Omni on COMPaS-II).

| | PE | 1 | 2 | 4 |
|---|---|---|---|---|
| Kernel 3 | $r_\infty$ | 83 | 110 | 225 |
| | $n_{1/2}$ | 21 | 217 | 1360 |
| Kernel 6 | $r_\infty$ | 102 | 166 | 385 |
| | $n_{1/2}$ | 21 | 129 | 1280 |

**表 5** $r_\infty$ と $n_{1/2}$ (PGI on COMPaS-II)
Table 5 $r_\infty$ and $n_{1/2}$ (PGI on COMPaS-II).

| | PE | 1 | 2 | 4 |
|---|---|---|---|---|
| Kernel 3 | $r_\infty$ | 200 | 359 | 249 |
| | $n_{1/2}$ | 27 | 435 | 312 |
| Kernel 6 | $r_\infty$ | 240 | 307 | 400 |
| | $n_{1/2}$ | 12 | 52 | 293 |

並列化するループの処理量が最も性能を得られるように分割するなど，慎重な並列化を行う必要があるといえる．

**表 4** と**表 5** に，COMPaS-II において Omni と PGI で得られた $r_\infty$ と $n_{1/2}$ の値を示す．これもプログラムが出力した値で，キャッシュサイズ以下での値である．これより，PGI コンパイラは Omni と比較して短いループ長で高い性能が得られることが分かる．

## 5. 関 連 研 究

OpenMP の処理系では，我々と同様に OpenMP プログラムを C プログラムへ変換するアプローチをとる OpenMP トランスレータ Odin/CCp[12] が Lund 大学で開発，公開されている．Odin/CCp も，我々の開発している OpenMP コンパイラ Omni と同様に，プログラム変換部分の開発に Java を用いている．対応言語は，Odin/CCp が C のみであるのに対して，Omni では C と FORTRAN77 に対応している．また，Omni は通常の OpenMP の変換に加え，コンパイラでソフトウェア分散共有メモリ機能を実現するために必要なフロー解析などの機能を備えている[4]．商用コンパイラでは，仕様の発表後間もなく対応した KAI guidec[8] のほか，最近は多くの製品が OpenMP 対応となってきている．

SMP クラスタ環境での並列化インタフェースとして OpenMP を利用する研究として，Fortran 版の OpenMP をソフトウェア分散メモリ環境で動作させる Rice 大学の研究[13]や，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[14]が行われている．特に，OpenMP と MPI を組み合わせるプログラミング手法は，SMP クラスタでの標準的なプログラム並列化手法として期待されている．我々は，SMP クラスタを OpenMP のみで透過的に利用するため，ページベースの分散共有メモリを用いずに，コンパイラによって一貫性制御と通信コードを最適化するコンパイラの研究を行っている[4),15)]．

OpenMP のベンチマークでは，エジンバラ大学が OpenMP を用いた場合のオーバヘッドを測定するマイクロベンチマークを開発，公開している[7]．OpenMP の ARB もベンチマークの開発を進めていることを表明しているが，具体的なことは明らかになっていない．

## 6. ま と め

以上，本稿では RWC で開発している Omni OpenMP コンパイラの概要とその性能評価を述べた．

Omni OpenMP コンパイラは，フロントエンド部，プログラム解析および OpenMP 指示変換部，そして実行時ライブラリの 3 つの部分で構成されており，OpenMP 指示を含むプログラムを，マルチスレッド C プログラムに変換する．マイクロベンチマークを用いた評価では，SUN において商用コンパイラである KAI の方が 2 割程度，COMPaS-II においては PGI が 2 倍程度，それぞれ Omni の性能を上回っていた．この性能の違いの原因であった OpenMP の ‘parallel’ を指定した際の Omni のオーバヘッドについて内訳を調べた結果，その半分以上は実行時ライブラリでの制御データの操作に費やしていることが分かった．この管理方法をオーバヘッドの少ないものにすることにより，Omni の性能を向上させることが可能であると考える．

次に，Parkbench を用いて積和演算ループを OpenMP を用いて並列化した場合に得られる最大性能を測定した結果，SUN において Omni は KAI guidec とほぼ同等の性能を得られることが分かった．COMPaS-II でも，十分な計算量がある場合には PGI コンパイラの 8 割程度の性能を得られることが確認できた．また，COMPaS-II でピーク性能を得るためには，並列実行ループの処理量を考慮して並列化する必要があることなど，SUN と比較すると最高性能を得ることが難しいことも分かった．

以上の評価結果から，Omni を SMP クラスタで並列処理環境を構築するベースとすることを考えてみる．マイクロベンチマークを用いた評価により，Omni が OpenMP の全機能をサポートしており，SMP マシンで商用の OpenMP コンパイラに近い性能が得られることが確認できた．また，Parkbench を用いた評価から，簡単なループの並列化でも商用の OpenMP コン

パイラと遜色ない最大性能を得られることが分かった．さらに，Omni はフリーソースであり，ソースコードが一般に入手可能であり，様々な並列計算機への移植や修正，最適化などを容易に行うことが可能である．以上のことから，我々が開発した Omni は SMP クラスタの並列処理環境を構築するベースとして十分な性能と機能を持っているといえる．

今後は，COMPaS-II でオーバヘッドが大きい原因となっていた並列実行の管理データの獲得と解放処理部分の最適化を行うとともに，他のベンチマークも用いて Omni の評価，および性能チューニングを進めていく．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．

**謝辞** 本研究に関する議論に参加して有益なアドバイスをしていただきました 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いたします．

## 参 考 文 献

1) OpenMP ARB: http://www.openmp.org/ (1997).
2) OpenMP ARB: OpenMP C and C++ Application Program Interface Ver 1.0 (1998).
3) 草野和寛，佐藤茂久，佐藤三久：Omni OpenMP コンパイラと実行時ライブラリの性能評価，並列処理シンポジウム JSPP2000 論文集，pp.229–236 (2000).
4) Sato, M., Satoh, S., Kusano, K. and Tanaka, Y.: Design of OpenMP Compiler for an SMP Cluster, *Proc. European Workshop on OpenMP EWOMP '99*, pp.32–39 (1999).
5) 田中良夫，久保田和人，佐藤三久，関口智嗣：並列アルゴリズムにおける Collective 通信の性能比較，*IPSJ 96-HPC-62*, pp.19–26 (1996).
6) Mellor-Crummey, J.M. and Scott, M.L.: Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors, *ACM Trans. Computer Systems* (*TOCS*), Vol.9, No.1, pp.21–65 (1991).
7) Bull, J.M.: Measuring Synchronisation and Scheduling Overheads in OpenMP, *Proc. European Workshop on OpenMP EWOMP '99*, pp.99–105 (1999).
8) Kuck and Associates.: http://www.kai.com/.
9) The Portland Group.: http://www.pgi.com/.
10) PARKBENCH Committee Report-1, assembled by Hockney, R. (chairman) and Berry, M.: Public International Benchmarks for Parallel Computers, Technical Report UT-CS-93-213, Department of Computer Science, University of Tennessee (1993).
11) 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*, LNCS, Vol.1388, pp.486–497, Springer (1997).
12) Brunschen, C. and Brorsson, M.: OdinMP/CCp – A portable implementation of OpenMP for C, *Proc. European Workshop on OpenMP EWOMP '99*, pp.21–26 (1999).
13) Lu, H., Hu, Y.C. and Zwaenepoel, W.: OpenMP on Networks of Workstations, *Proc. Supercomputing'98* (CD-ROM) (1998).
14) Cappello, F. and Richard, O.: Performance Characteristics of a Network of Commodity Multiprocessors for the NAS Benchmarks Using a Hybrid Memory Model, *Proc. 1999 International Conference on Parallel Architectures and Compilation Techniques* (*PACT '99*), pp.108–116, IEEE Computer Society Press (1999).
15) 佐藤茂久，草野和寛，佐藤三久：OpenMP コンパイラにおけるメモリ一貫性制御の最適化，並列処理シンポジウム JSPP2000 論文集，pp.221–228 (2000).

(平成 12 年 9 月 13 日受付)
(平成 13 年 2 月 1 日採録)

**草野 和寛**（正会員）

昭和 40 年生．平成元年九州大学工学部情報工学科卒業．平成 3 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．同年，日本電気（株）入社．平成 9 年より新情報処理開発機構つくば研究センターに出向．並列化コンパイラ，並列化支援ツール等の研究に従事．

**佐藤　茂久**（正会員）

昭和 41 年生．平成元年東京理科大学理学部応用数学科卒業．平成 3 年東京理科大学大学院理学研究科数学専攻修士課程修了．同年（株）日立製作所入社．システム開発研究所にて最適化/並列化コンパイラの研究開発に従事．平成 10 年より新情報処理開発機構に出向．OpenMP コンパイラ，ソフトウェア DSM，SMP クラスタを用いた高性能計算の研究に従事．コンパイラ（プログラム解析，コード最適化），並列処理，マイクロプロセッサ・アーキテクチャ等に興味を持つ．IEEE，ACM 各会員．

**佐藤　三久**（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年より，新情報処理開発機構つくば研究センターに出向．現在，同機構並列分散システムパフォーマンス研究室室長．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グローバルコンピューテイング等の研究に従事．日本応用数理学会会員．