計算機アーキテクチャ 137−31
ハイパフォーマンス
コンピューティング 80−31
（2000. 3. 3）

Omni OpenMP コンパイラの性能評価

草野和寛† 佐藤茂久† 佐藤三久†

我々は並列化指示インタフェースとしてOpenMPを採用したSMPクラスタ用並列処理環境を構築しており、その一部としてOmni OpenMPコンパイラと実行時ライブラリを試作した。現在OmniはCに対応したバージョンを公開している。本稿では、ベンチマークを用いて行ったOmniの性能評価について述べる。4プロセッサのSUNで行った評価の結果、OmniはKAIのOpenMPコンパイラと比較して、OpenMPのオーバヘッドはほぼ同等であることがわかった。一方、Pentium II Xeonを用いたSMP(4CPU)上のLinuxでは、当初非常に大きなオーバヘッドを要していたが、スレッドの待機状態における'sched_yield()'の呼び出しをなくすことで、オーバヘッドを5分の1以下にすることができた。

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 C compiler and runtime libraries on the SMP machine. This paper describes evaluation of the Omni OpenMP compiler and its runtime library using some benchmark programs. The result shows the overhead of OpenMP parallelization of the Omni is almost the same compared with the KAI guidec OpenMP compiler on a four processor SUN450. On a PC with four Pentium II Xeon processors running Linux, we find a function call to a library 'sched_yield()' makes the OpenMP parallelization overhead several times.

1. はじめに

計算機ハードウェア、特にCPUの性能向上に伴い、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上に、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]が注目を集めている。OpenMPは、これまで並列計算機それぞれで異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。この仕様では、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を定義している。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。現在Fortran版の仕様は1.1となっており、さらに今年中に2.0が公開される予定である。

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[9][10]。本稿では、前述のマイクロベンチマークなどを用いて行ったOmni OpenMPコンパイラと実行時ライブラリの評価方法とその結果について述べる。

OpenMP関係の研究としては、Fortran版のOpenMPプログラムをソフトウェア分散共有メモリを用いて分散環境で実行する研究がRice大で行われている[3]。また、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[4]に対して大きな期待が寄せられている。OpenMP処理系の研究として、Omniと同様にトランスレータ形式であるフリーの処理系がLund大で開発されている[5]。さらに、共有メモリ用のOpenMP

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

と分散メモリ用の HPF を統合する検討も行われている[6]。OpenMP のベンチマークでは、エジンバラ大が 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[7] を開発、公開[8] している。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。

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。そして、最後の 5 章でまとめと今後の課題をあげる。

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[9][10]。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に示す通り、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三つの部分から成る。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。

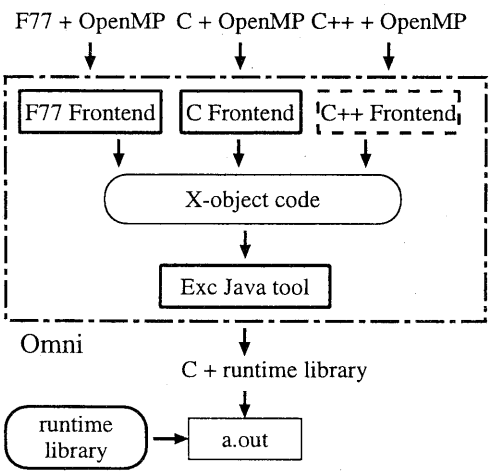

図 1　Omni OpenMP コンパイラ

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 の他に、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のチェックのみを行い、実行文と同様の構文木などに変換される。

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 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。そして、これらの解析情報を利用して OpenMP 指示の解釈を行い、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。

```
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

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。実行時ライブラリは、利用するスレッドライブラリ (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。

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一度だけ行い、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。

3. Omniの基本性能評価

本章では、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][8]は、エジンバラ大が開発し、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、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、その差分をOpenMPによる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にスケジューリング方法と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

今回の評価には、以下にあげる計算機と最新バージョンのOmniを用いた。

- SUN Enterprise450(4CPU), Solaris2.6, SUNWspro4.2 C compiler, JDK1.2
- COMPaS-II(COMPAQ ProLiant6500, 4CPU), RedHat Linux 6.0+kernel 2.2.12, egcs-1.1.2, JDK1.1.7

また、OpenMPコンパイラの比較対象として、上記計算機で利用できるOpenMP処理系である以下のものを用いた。

- KAI guidec[11](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と同様に一旦Cへ変換するトランスレータ形式である。
- PGI pgcc[12](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。今回の評価ではLinux版を用いたが、他にSolaris86/WinNT版もある。

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、SUNにおけるOmniのOpenMP指示10種類のオーバヘッドを図3に、KAI guidecのオーバヘッドを図??に示す。ここで、バックエンドのCコンパイラには両方供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は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、実行時ライブラリでmutex_lockやPOSIXスレッドを用いた場合もほぼ同等の性能であった。

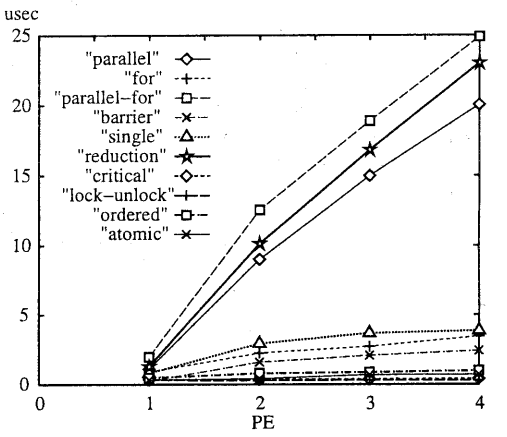

図3 Omniの基本性能(SUN)

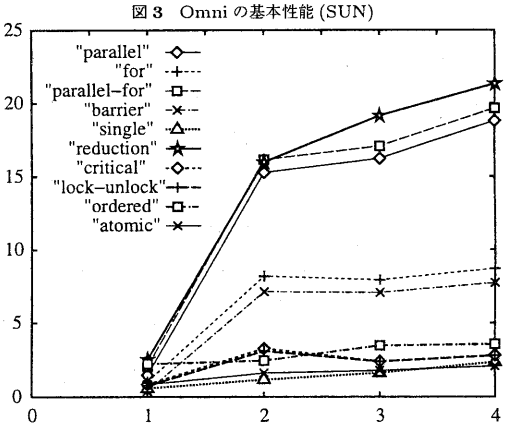

図4 KAI guidecの基本性能(SUN)

この結果、Omniのオーバヘッドは商用のOpenMPコンパイラであるKAIの処理系と同等であった。また、並列実行部分の指定である'parallel'と'parallel-for'、それに'reduction'演算のオーバヘッドが大きい。したがって、OpenMPを用いた並列化を行う場合には、できるだけ並列実行部分を大きくとることが並列化で性能をあげるために重要であること、そして、要素数の少ない'reduction'演算は逐次実行した方が有利であると言える。

3.3.2 COMPaS-IIでの評価

COMPaS-IIにおけるOmniのOpenMP指示のオーバヘッドを図5に、PGI pgccのオーバヘッドを図6に示す。Omniでは、実行時ライブラリにPOSIXスレッドを用い、排他制御にスピンウェイトを用いている。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

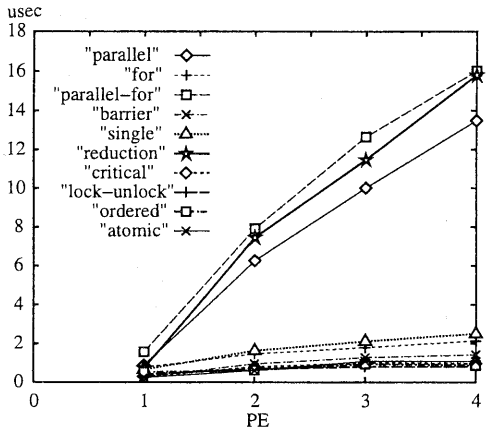

図 5 Omni の基本性能 (COMPaS-II)

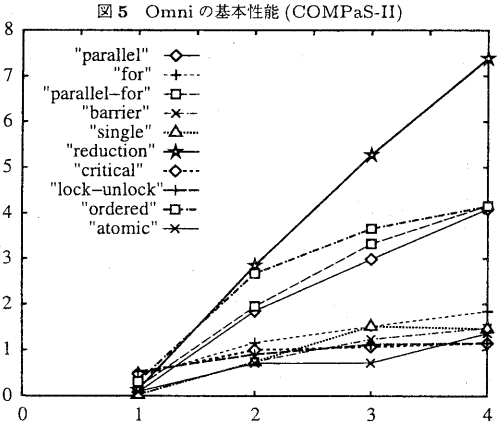

図 6 PGI pgcc の基本性能 (COMPaS-II)

当な値であると言える。一方、PGI の測定結果では、Omni で他より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、非常に小さなオーバヘッドとなっている。この原因の一つとして考えられるのは、PGI のコンパイラは Omni とは異り、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である。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り、Omni の実行時ライブラリにはまだチューニングの余地があると思われる。

そこで、大きなオーバヘッドを示している OpenMP で並列実行部分を指示する'parallel' に関して、その処理時間の内訳を調べた。この結果、オーバヘッドが大きいことがわかった、並列実行を管理するデータ構造の取得とその設定、および並列実行の後で前記管理データのクリア処理と同期待ちの時間 (usec) とオーバヘッド全体に占める割合 (%) を表 1に示す。 Omni では、並列実行の管理データはフリーリストから割り当て、並列実行終了後にフリーリストに戻しており、この操作では排他制御を行っている。この排他制御自体のオーバヘッド

PE	1	2	3	4
管理データ割り当て	0.02(23)	1.9(30)	3.6(35)	5.0(36)
管理データ解放	0.28(30)	2.2(36)	4.4(43)	7.0(50)

表 1 主要な OpenMP オーバヘッド (usec(%))

に加え、処理が逐次化されることによるオーバヘッドがあるため、台数にほぼ比例したオーバヘッドとなっていた。

3.3.3 COMPaS-II での評価 (2)

ここで、昨年リリースした Omni 1.0 で得られた結果を、図 7に示す。図 5と比べると、'parallel' などで 10

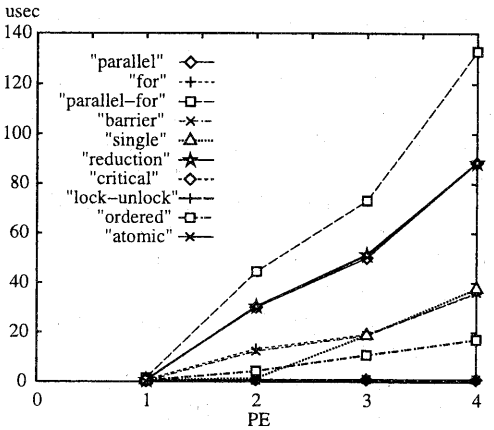

図 7 Omni 1.0 の基本性能 (COMPaS-II)

倍近いオーバヘッドを要していることがわかった。この原因は、プログラム開始時にスレッドを生成した後、マスタスレッド以外のスレッドが待機している部分で、ライブラリ関数 "sched_yield()" を呼び出しているためであった。

また、排他制御に'mutex_lock()' を用いた場合には、ライブラリ関数 "sched_yield()" の呼び出しでオーバヘッドに大きな違いはなかった (図 8)。

4. OpenMP による性能向上の評価

この章では、Parkbench[13] の Low level ベンチマークの一つである "rinf1" を利用して評価した OpenMP による性能向上について結果を述べる。

4.1 Parkbench

この評価で用いた Parkbench[13] は、並列計算機の性能指標となることを考慮しつつ、フリーなベンチマークとして開発されたベンチマークである。この中には大きく分けて三種類 (application, kernel, low-level) のベンチマークがあるが、今回用いたのは最も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

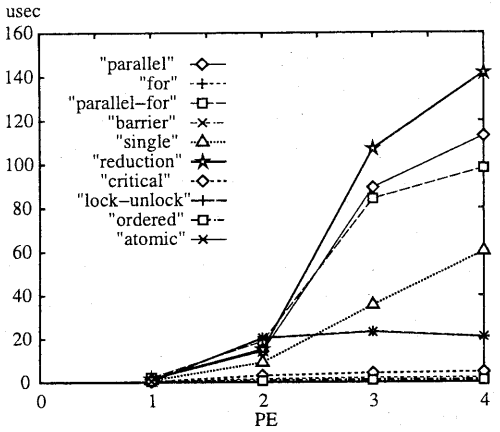

図 8 Omni 1.0 の基本性能 (COMPaS-II)-mutex_lock

を用いて、R-infinity と N-half を求める。R-infinity は配列のベクトル長を変化させた場合に得られる最大性能で、N-half は、R-infinity の半分の性能を得られるベクトル長である。今回の評価では、内側計算ループを OpenMP 指示により並列化して、1/2/4PE それぞれの実行時間から、逐次の場合と同様に R-infinity と N-half を求めた。

4.2 測定結果

今回の評価には、rinf1 に含まれる複数のループから、ベクトル長 (n) を変化させ配列演算を行う kernel 3 と 6 の二つを用いた (図 9)。

```
/* 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 9 Parkbench のループ

図 10と図 11に kernel 3 と 6 に OpenMP の並列化指示を加え、SUN 上の Omni を用いて並列化した結果を示す。また、R-infinity(Ri) と N-half(Nf) は、この図よりおよそ以下の表 2の値であることがわかる。この結

PE	1	2	4
Kernel3-Ri	128	220	340
Kernel3-Nf	150	1400	3200
Kernel6-Ri	145	265	405
Kernel6-Nf	70	680	1900

表 2 OpenMP 版 Parkbench の R-infinity と N-half

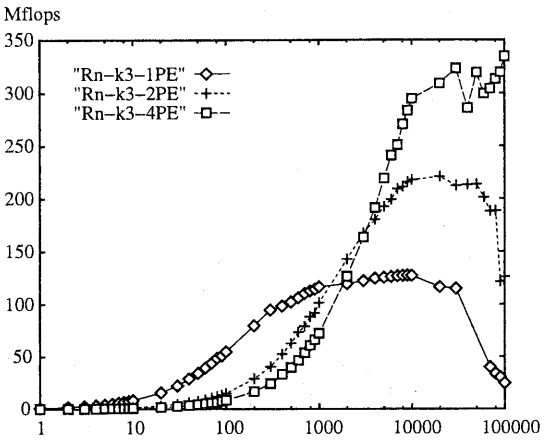

図 10 rinf1-kernel3 の性能 (Omni on SUN)

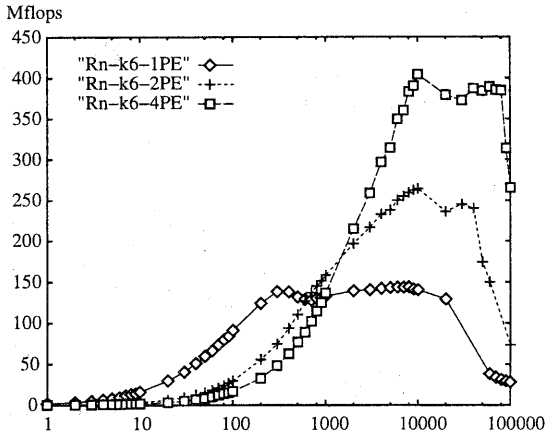

図 11 rinf1-kernel6 の性能 (Omni on SUN)

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、十分なベクトル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わかる。次に、N-half となるベクトル長から、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ベクトル長、つまりループの計算量が必要であることがわかる。並列実行時に 1PE のピーク性能を超えるベクトル長は、並列実行 PE 数で大きな違いはなく、kernel 3 ではおよそベクトル長 2000 であった。

次に、同様に COMPaS-II で測定した結果を図 12と図 13に示す。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ベクトル長で現れるが、その性能が低下するベクトル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わかる。また、1PE のピーク性能は SUN とほとんど変らないのに対して、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 2PE での性能向上が

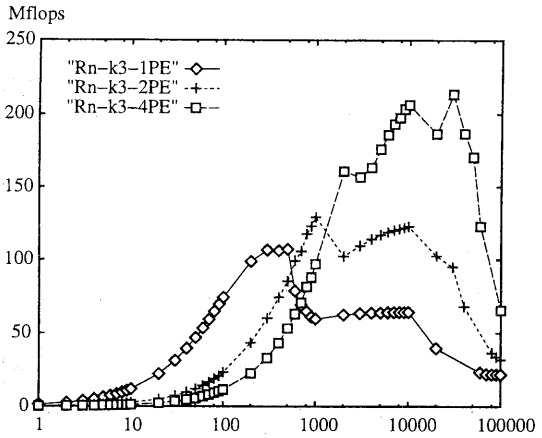

図 12 rinf1-kernel3 の性能 (Omni on COMPaS-II)

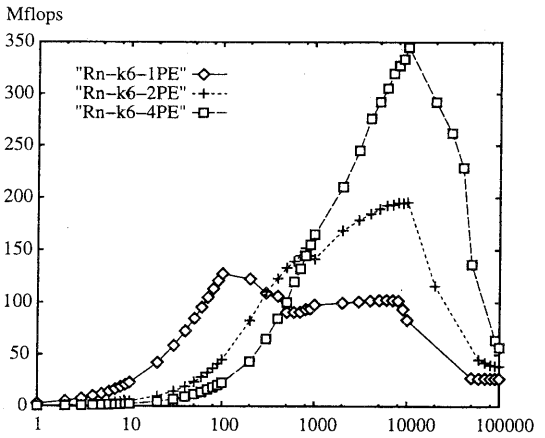

図 13 rinf1-kernel6 の性能 (Omni on COMPaS-II)

低く、kernel 3 では 1PE とほとんど変っていない。この結果から、COMPaS-II では SUN よりもピーク性能を得るには注意深い並列化が必要であることがわかる。

5. まとめと今後の課題

以上、本稿では RWC で開発している 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三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。

Omni の性能を、マイクロベンチマークと Parkbench を用いて評価した結果、SUN において商用コンパイラである KAI と同等の性能であることを示した。COMPaS-II 上の評価では、ライブラリ関数 'sched_yield()' の呼び出した場合や、実行時ライブラリの排他制御に mutex_lock() を使用した場合に、オーバヘッドが大きいことがわかった。このオーバヘッド時間の内訳を調べた結果、その半分以上は実行時ライブラリでのスレッド管理データの獲得と解放であることがわかった。Parkbench を用いた評価では、積和演算のみのループを OpenMP で並列化した場合の性能向上に関して、R-infinity と N-half の値を示した。この結果、ループ本体の計算量が少ないこともあり、並列化した場合の N-half の立ち上がりが鈍かった。

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/odinmp/
6) B. Chapman and P. Mehrotra, "OpenMP and HPF: Integrating Two Paradigms", Euro-Par'98, 650-658, 1998.
7) J. M. Bull, "Measuring Synchronisation and Scheduling Overheads in OpenMP", EWOMP '99, Lund, Sep., 1999.
8) http://www.epcc.ed.ac.uk/research/openmpbench
9) 草野, 佐藤 (三), 佐藤 (茂), "OpenMP コンパイラの試作と評価", 96-HPC-76, pp.7-12, May, 1999.
10) M. Sato, S. Satoh, K. Kusano and Y. Tanaka, "Design of OpenMP Compiler for an SMP Cluster"; EWOMP '99, pp.32-39, Lund, Sep., 1999.
11) http://www.kai.com/
12) http://www.pgroup.com/
13) http://www.netlib.org/parkbench/html/