

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 のオーバー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ー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、オーバーヘッドを 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 MITSUHIKA 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 OpenMP¹⁾ 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ー
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ー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 指示のオーバー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。

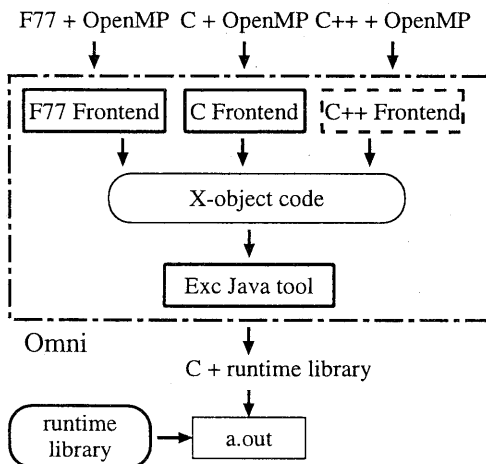


図 1 Omni OpenMP コンパイラ
Omni のフロントエンドは、仕様書第 1 版²⁾に基づい

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_ptolrstd;
    _pp_ptx = (double **)(__ompc_args+0);
    _pp_ptolrstd = (double **)(__ompc_args+1);
    {
        /* 並列実行コード */
    }
}

main(){
    ...
    /* parallel 指定箇所 */
    auto void **__ompc_argv[5];
    *(__ompc_argv+0) = (void *)&ptx;
    *(__ompc_argv+1) = (void *)&ptolrstd;
    _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ー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 によるオーバーヘッドとしている。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ーヘッドである。

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, SUN-Wspro4.2 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 guidec¹¹⁾(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 pgcc¹²⁾(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 種類のオーバーヘッドを図 3 に、KAI guidec のオーバーヘッドを図 4 に示す。ここで、バックエンドの C コンパイラには両方供 SUN-Wspro4.2 を使い、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。

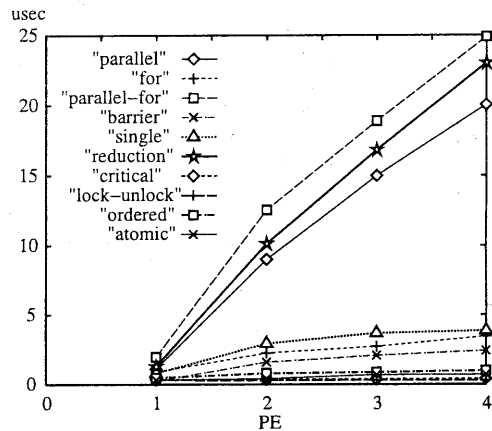


図 3 Omni の基本性能 (SUN)

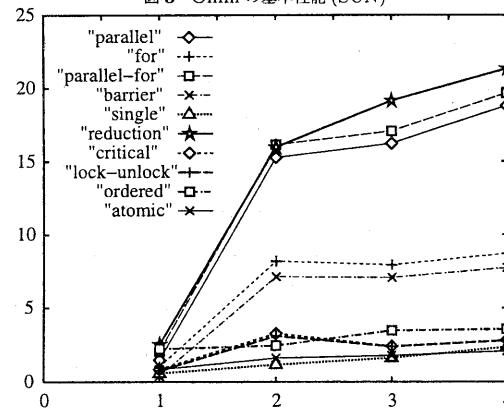


図 4 KAI guidec の基本性能 (SUN)

この結果、Omni のオーバーヘッドは商用の 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' 演算のオーバーヘッドが大きい。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云える。

3.3.2 COMPaS-II での評価

COMPaS-II における Omni の OpenMP 指示のオーバーヘッドを図 5 に、PGI pgcc のオーバーヘッドを図 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 のオーバーヘッドは、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

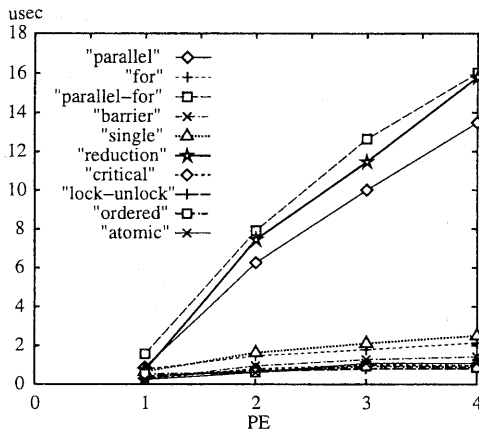


図5 Omniの基本性能 (COMPaaS-II)

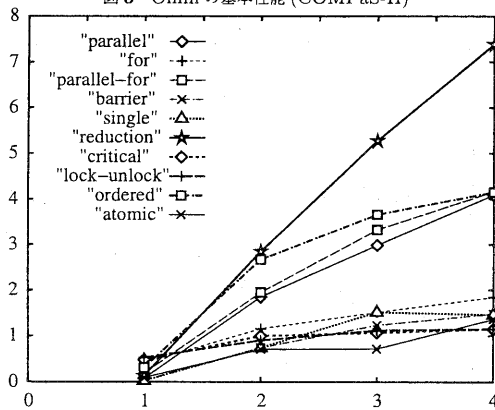


図6 PGI pgccの基本性能 (COMPaaS-II)

当な値であると言える。一方、PGIの測定結果では、Omniで他よりオーバーヘッドが大きい'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ー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ー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、大きなオーバー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、オーバー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) とオーバー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ーヘッド

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オーバーヘッド (usec(%))

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

3.3.3 COMPaaS-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

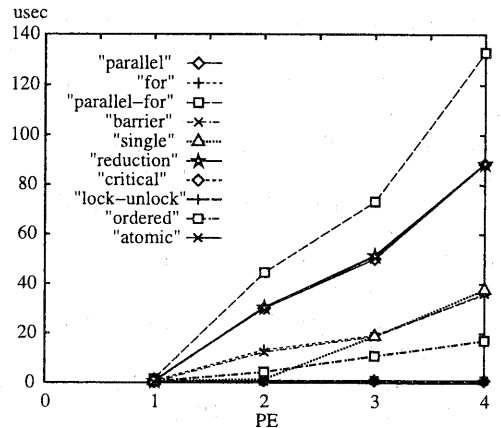


図7 Omni 1.0の基本性能 (COMPaaS-II)

倍近いオーバー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ー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、Parkbench¹³⁾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Parkbench¹³⁾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

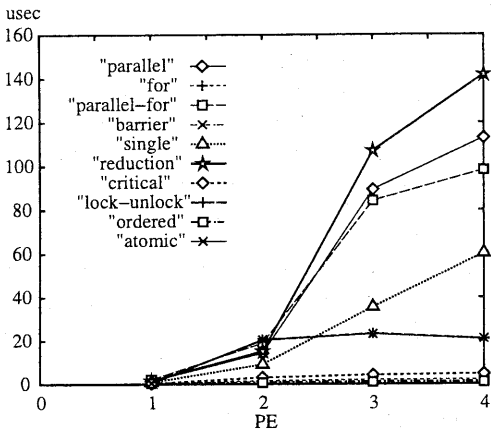


図8 Omni 1.0の基本性能 (COMPAS-II)-mutex_lock

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、逐次的の場合と同様にR-infinityとN-halfを求めた。

4.2 測定結果

今回の評価には、rinfl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

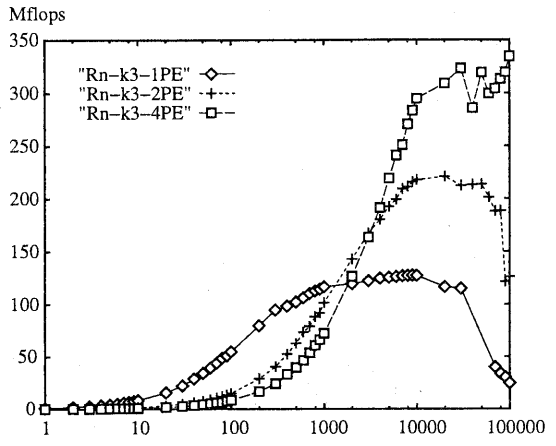


図10 rinfl-kernel3の性能 (Omni on SUN)

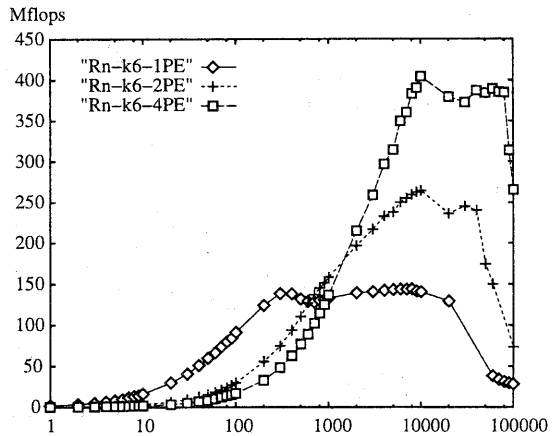


図11 rinfl-kernel6の性能 (Omni on SUN)

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、立ち上がりは鈍くなっており、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2PEでの性能向上が

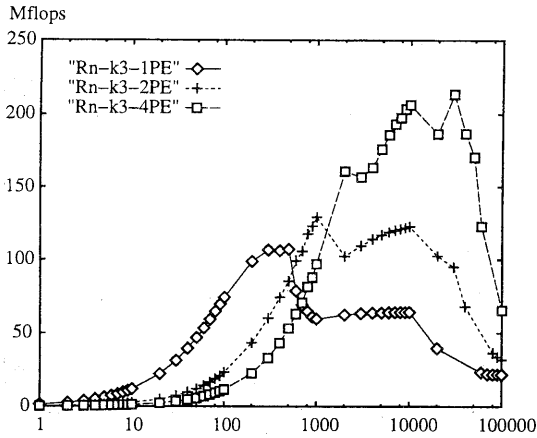


図 12 rinfl-kernel3 の性能 (Omni on COMPaS-II)

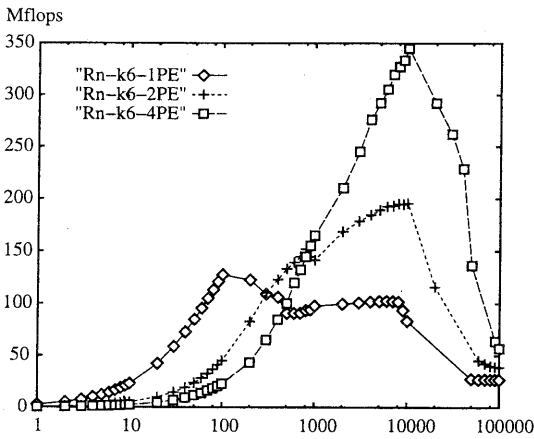


図 13 rinfl-kernel6 の性能 (Omni on COMPaS-II)

低く、kernel 3 では 1PE とほとんど変わっていない。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 mu-

tex.lock() を使用した場合に、オーバーヘッドが大きいことがわかった。このオーバー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 でオーバー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", EuroPar'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/>