社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

# λコンピューティング環境におけるOpenMPアプリケーションによる共有メモリシステムの評価

井本　舞†　合田　圭吾†　藤本　典幸†　馬場　健一††　村田　正幸†

† 大阪大学 大学院情報科学研究科　〒565–0871 大阪府吹田市山田丘 1–5
†† 大阪大学 サイバーメディアセンター　〒567–0047 大阪府茨木市美穂ヶ丘 5–1
E-mail: †{m-imoto,k-gouda,fujimoto,murata}@ist.osaka-u.ac.jp, ††baba@cmc.osaka-u.ac.jp

**あらまし**　我々の研究グループでは，各ノード計算機間に光ファイバを直結して各ノード計算機上に存在する共有メモリを波長パスで結ぶことにより，高速計算を可能とするλコンピューティング環境を提案している．本稿では，並列計算用プログラミング API である OpenMP アプリケーションをλコンピューティング環境上で動作させることによって環境の性能評価を行った．その結果，並列化による効果は得られるものの，共有メモリへのアクセス速度が性能に大きな影響を及ぼしていることがわかった．

**キーワード**　λコンピューティング環境，分散並列計算，AWG-STAR システム，OpenMP，分散共有メモリ

# Evaluation of Shared Memory System with OpenMP Applications in λ Computing Environment

Mai IMOTO†, Keigo GODA†, Noriyuki FUJIMOTO†, Ken-ichi BABA††, and Masayuki MURATA†

† Graduate School of Information Science and Technology, Osaka University
1–5 Yamadaoka, Suita, Osaka 565–0871, Japan
†† Cybermedia Center, Osaka University　5–1 Mihogaoka, Ibaraki, Osaka 567–0047 Japan
E-mail: †{m-imoto,k-gouda,fujimoto,murata}@ist.osaka-u.ac.jp, ††baba@cmc.osaka-u.ac.jp

**Abstract**　Our research group has proposed a new high performance computing architecture, which we call the λ computing environment. In the λ computing environment, every computing node has a shared memory connected with the optical wavelength path. In this paper, we execute an application of OpenMP, Parallel Programming API, and evaluate the performance of the λ computing environment. The results show an advantage of parallel computing and effect of the access speed to the shared memory on the performance.

**Key words**　λ Computing Environment, Distributed Parallel Computing, AWG-STAR system, OpenMP, Distributed Shared Memory

## 1. はじめに

近年，グリッドコンピューティングに対する期待が高まり，活発に研究開発が行われている．グリッドコンピューティングとは，広域に分散した計算機やストレージ，様々なデバイスなどの多くの資源を，ネットワークで接続することでひとつの大規模な仮想計算機として機能させる技術である．

現在のグリッドコンピューティング環境ではデータ通信に通常 TCP/IP が用いられている．しかし，TCP/IP のようなパケット単位のデータ交換では輻輳制御による通信帯域の抑制やパケット損失によるデータの再送などのパケット処理に要するオーバヘッドが大きい．このため，大規模な計算で行われる大量のデータ共有やデータ交換を行うには十分な性能を得ることが非常に難しい。

そこで我々の研究グループでは、各ノード計算機を接続している光ファイバを専用の通信路として利用し，WDM 技術を用いた高速な通信チャネルとして活用するλコンピューティング環境を提案している [1,2]．λコンピューティング環境上では，TCP/IP ではなく波長パスを利用するため，高速高信頼な通信を実現することができる．λコンピューティング環境を構築するひとつの手段として，日本電信電話株式会社フォトニクス研究所が開発している「情報共有ネットワークシステム（AWG-STAR

```
double pi = 0.0;
#pragma omp parallel reduction(+:pi)
for (i = 0; i < N; i++) {
    double x = (i + 0.5) * w;
    pi += 4.0 / (1.0 + x * x);
}
```

図 1　OpenMP ソースコードの例

システム)」[3] を用い，その上で並列プログラミング API である OpenMP [4] を設計し，実装した [5,6].

そこで本稿では，OpenMP アプリケーションを実行することにより共有メモリシステムの性能を明らかにする．さらに，ボトルネックとなる共有メモリのアクセス速度を改善したシステムを用いる場合の性能について述べる．

## 2. AWG-STAR システムと OpenMP による分散並列計算環境の構築

### 2.1 AWG-STAR システムの概要

AWG-STAR システムは，WDM 技術によるデータ転送と AWG ルータによる波長ルーティング技術によって実現された情報共有ネットワークシステムである [3]．AWG ルータは電気信号に変換せず光信号をそのまま処理するため，高速なネットワークを構築することができる．また，AWG-STAR システム上の各ノード計算機は，共有メモリボードを搭載し，共有メモリボード上で同一のデータを保持することでメモリを共有しており，高速な光リングネットワークを利用してデータ交換をリアルタイムに行うことができる．共有メモリに書き込まれたデータは光リングネットワークに送出され，全ノード計算機の共有メモリを自動的に更新する．また，他ノード計算機が更新したデータの取得は，AWG-STAR システムを通じて共有メモリに配信され，自動的に更新されるため，自ノード計算機上の共有メモリから読み込むことにより実現できる．

### 2.2 OpenMP の概要

並列プログラミング API である OpenMP によるプログラミングは，ソースコード中に OpenMP ディレクティブと呼ばれる指示文を記述することで行う．図 1 に示したのは簡単な OpenMP プログラムのソースコードの例である．2 行目の `pragma` に続く記述が OpenMP ディレクティブであり，この例では 3 行目以降の `for` ループ処理部分が並列実行される．並列化指示のない部分は逐次実行される．OpenMP ではこの逐次実行部を行うプロセスをマスタプロセスと呼び，それ以外のプロセスをワーカプロセスと呼ぶ．

OpenMP ではプログラマはノード計算機への計算処理の割り当てやノード計算機間の通信などを考える必要がない．そのかわり，OpenMP コンパイラは対象となる並列計算環境のハードウェアや OS，通信ライブラリなどに応じて実装する必要がある．

## 3. OpenMP コンパイラの設計と実装

我々の研究グループでは既存の OpenMP コンパイラ OMPi [7] に基づいて AWG-STAR システム向けコンパイラおよびランタイムライブラリを設計し，実装した [5]．ただし，OMPi は SMP 環境のためのコンパイラであるため，クラスタ環境に移植する方法として文献 [8] で採用されている手法を使用した．

変数の共有は AWG-STAR システムが提供する共有メモリボードを用いて実現した．すなわち，変数のメモリ領域を共有メモリボード上の共有メモリに割り当てる．これにより AWG-STAR システムの機能によって変数の内容が自動的に計算機間で共有される．

また，計算中に計算機間で共有されるデータの読み書きは，データの不整合を防ぐためにロック機構によって排他制御されねばならない．さらに，依存関係のある処理の実行順序を保証するために計算機間でタイミングを同期するバリア同期機能も必要である．これらの同期プリミティブとよばれる機能を我々は AWG-STAR システムが提供する共有メモリおよびシグナリング機能を用いて独自に同期ライブラリを作成した [6]．具体的には，ロックの要求やバリア到達の通知をワーカプロセスがマスタプロセスへシグナル送信することで，マスタプロセスがそれらに応じて各プロセスを制御する．この同期プリミティブを呼び出すことで OpenMP プログラムにおける計算機間同期を実現する．

## 4. アプリケーションによる評価

本章では実装した OpenMP コンパイラおよびランタイムライブラリ，同期プリミティブを用いて OpenMP アプリケーションを動作させ，その性能を評価する．OpenMP アプリケーションはマンデルブロー集合の計算を用いることとし，アプリケーションの実行状況が視覚的にわかるように，JAVA を用いて GUI からアプリケーションが操作できるようにした．

### 4.1 実験システム環境

評価に用いた計算機の仕様を表 1 に，AWG-STAR システムの仕様を表 2 に示す．実験に使用したノード計算機の台数は 1 台から 4 台の範囲で行い，全て同じ性能のノード 計算機を用いた．今回の実験では，ノード計算機数を $N$ とすると，光リングネットワークの長さは $10N$m としている．また，比較対象として Ethernet を用いたソフトウェア分散共有メモリ環境を構築し，OpenMP を動作させた．具体的には，クラスタ環境 SCore および SCore に含まれるソフトウェア分散共有メモリ SCASH を用いた [8,9]．SCore を利用する際はネットワークとして 1Gbps の Ethernet を用いている．

### 4.2 マンデルブロー集合計算

マンデルブロー集合とは，$z_0 = 0$，$z_{n+1} = z_n^2 - c$ $(n = 0, 1, \cdots)$ で定義される複素数の数列 $z_n$ が有界である（$n \to \infty$ で $|z_n|$ が発散しない）ような複素数 $c$ の集合のことであり，2 次元座標にプロットすることでフラクタル図形が描画される．

単純な静的分割による並列化処理方式では，指定サイズの複素数平面をプロセス数分の均等固定領域に分割し，それぞれの領域の計算を各プロセスが行う．計算は独立で，複素数平面の範囲を与えるパラメータが分かれば，複素数平面の初期値が必要ないため，最後に 1 回計算結果を回収すればよい．これは共有メモリへのアクセスが少なく，λコンピューティング環境の

表 1　実験に用いた計算機の仕様

| | |
|---|---|
| CPU | Xeon 3.06 GHz |
| メインメモリ | 512MB |
| 1 次キャッシュ | 20KB |
| 2 次キャッシュ | 512KB |
| NIC | Intel PRO/1000 |
| PCI バス | 64 bit/66MHz |
| PCI 転送速度 | 533MBytes/s |
| OS | Redhat Linux 7.3 |
| コンパイラ | GCC 2.96 |

表 2　AWG–STAR システムの仕様

| | |
|---|---|
| 光インターフェースの伝送速度 | 2.152Gbps |
| ノード計算機の 1 回あたりの転送データ量 | 1KByte |
| ノード計算機でのフレーム転送処理遅延 | 500ns |
| 共有メモリへの最大書き込み速度 | 64MBytes/s |
| 共有メモリから最大読みだし速度 | 80MBytes/s |

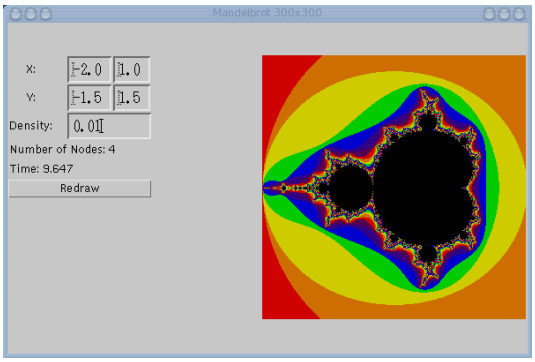


図 2　画像表示ウィンドウ

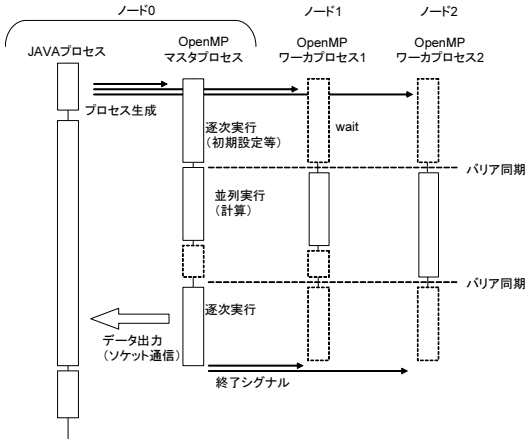


図 3　プロセスの動作順序

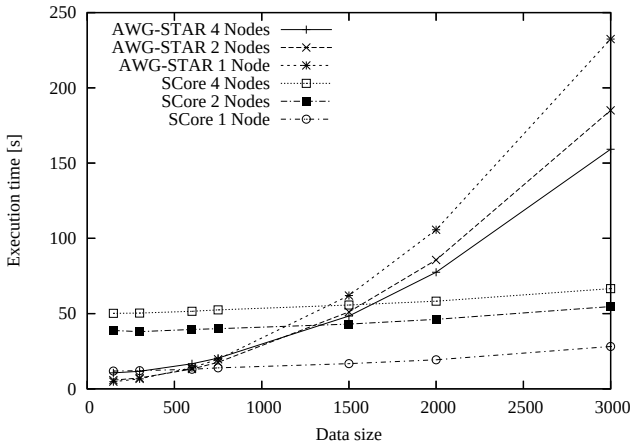


図 4　アプリケーション実行時間

みならず一般的な環境においても並列計算に向いたアプリケーションであるといえる．

### 4.3　画像表示方法

図 2 にマンデルブロー集合計算の画像を表示したウィンドウを示す．X, Y の横の値は計算範囲の最小値と最大値を示し，Density は計算密度を示す．これらの値を変更して Redraw ボタンを押すことにより，計算範囲を変えて再描画することができ，計算にかかった時間も表示する．

図 3 に JAVA および OpenMP のプロセス動作順序を示す．JAVA から OpenMP のマスタ／ワーカを含めた全ての OpenMP プロセスを生成する．OpenMP のマスタプロセスは逐次実行部を開始し，ワーカプロセスはただちにバリア同期関数を呼び出す．マスタプロセスがバリアに到達すると逐次実行部を終え，並列実行部に入る．並列実行部ではマンデルブロー集合計算を行い，並列実行部の最後にバリア同期関数が呼ばれる．全プロセスで計算が完了後逐次実行部に戻り，マスタプロセスは計算結果をソケットを通して JAVA プロセスに送信する．ただし，データサイズが大きい場合は逐次実行と並列実行を複数回繰り返して計算を行う．

### 4.4　性 能 評 価

#### 4.4.1　計 算 時 間

図 4 に AWG-STAR および SCore での実行時間を示す．計算する複素数集合のデータサイズを $x \times y$ とすると，図の横軸は $x = y$ のデータサイズで計算した $x$ の値を示している．AWG-STAR においては，データサイズが小さいときはノード数の少ない方が計算時間が短い．しかし，データサイズが大きくなるにしたがってノード数が多いほど計算時間が短くなっている．これはマンデルブロー集合計算における並列化の効果が得られることを表している．SCore においては，初期設定と思われる時間が計算ノード数に応じて 10 秒から 50 秒ほどかかっている．初期設定にかかる時間を除いた計算速度は AWG-STAR 上で動作させた OpenMP より速く，計算ノード台数の違いによる計算速度の差は少ない．

#### 4.4.2　CPU 使用率

4 台のノードでデータサイズを変更して計算をさせたときの，CPU 使用率を図 5，6 に示す．ただし，1 秒単位で CPU 使用率を計測しているため，ノード間のタイミングはずれがある．計算開始から約 7 秒間はマスタプロセスによる逐次実行部分である．ここでは共有メモリ上の画像データを読み込む配列を初期化するため，マスタプロセスが配列のサイズ分のデータを共有メモリへ書き込んでいる．初めの逐次実行が終わるとワーカプロセスを含めた並列実行部が開始される．この並列実行部ではマンデルブロー集合計算を行っているため，ワーカプロセスではほぼ 100%に近い CPU 使用率となっている．マスタプロセス

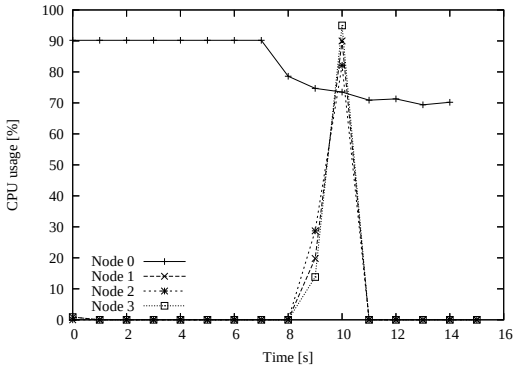


図 5　データサイズ 600x600

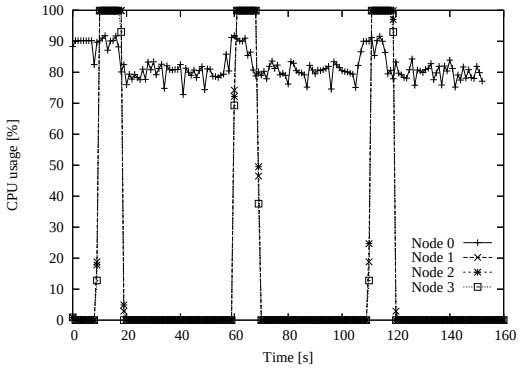


図 6　データサイズ 3000x3000

の CPU 使用率が 90%程度に留まる理由は，マンデルブロー集合計算だけでなく，JAVA プロセスや同期プリミティブのための動作を行っているためと考えられる．

この結果をみると，マスタプロセスのみが動作する逐次実行の時間が長いことがわかる．この間はマスタプロセスが共有メモリから計算結果を読み込み，それを JAVA プロセスにソケット通して送信している．また，JAVA プロセス側は OpenMP のマスタプロセスからデータを受信して画像描画している．この動作の中で特にオーバーヘッドとなっているのが共有メモリからのデータの読み込みであり，読み込み速度は実測で 1MByte/s 以下となっている．それに対して SCore はソフトウェア共有メモリにより，ローカルメモリ上に共有メモリを持つため，読み込み速度はローカルメモリとほぼ同じである．そのため，Ethernet を利用したデータ転送を行っているもののメモリアクセスのオーバーヘッドが少なくデータサイズが大きい場合でも処理遅延は低く抑えられている．

### 4.5　次期 AWG-STAR システムの性能向上に関する考察

AWG-STAR システムを利用した分散並列計算の性能向上ため，NTT フォトニクス研究所において次期 AWG-STAR-STAR システムの検討が進められている．次期 AWG-STAR システムでは共有メモリのアクセス性能の向上のため，ローカルメモリの一部分を共有メモリとして利用する構成をとる．これにより，共有メモリ上に置かれたデータをローカルメモリと同じ速度で読み取ることが可能となる．

前節にも示したように多くの並列計算アプリケーションにおいて共有メモリの読み込み性能は計算性能に大きな影響を与える．OpenMP を利用した小規模な並列計算ベンチマークである NAS Parallel Benchmark 2.3 EP について，AWG-STAR 上で実行した場合の共有メモリアクセス時間の割合を調査したところ，計算時間の大半が共有メモリからの読み込み時間で占められていることが分かった．表 3 はプログラム中での共有メモリアクセス回数と AWG-STAR システムで共有メモリの読み書きに要する平均アクセス時間とから推定した共有メモリの総アクセス時間である．このことから，次期 AWG-STAR システムでの共有メモリ読み込み性能の向上は，AWG-STAR システムを用いた分散並列計算の性能向上に大きく寄与すると考えられる．

表 3　NPB2.3 EP 共有メモリ総アクセス時間 (ms)

| ノード数 | 実行時間 | 読み込み時間 | 書き込み時間 |
|---|---|---|---|
| 1 | 337580 | 345951 | 0.03 |
| 2 | 197000 | 172975 | 0.04 |
| 4 | 89600 | 86487 | 0.08 |

## 5. おわりに

本稿では，λ コンピューティング環境の実現形態として AWG-STAR システムを用いることとし，並列プログラミング API である OpenMP のアプリケーションを動作させることで性能評価を行った．その結果，並列効果の高いアプリケーションであれば計算台数が増えるほどパフォーマンスがよくなることがわかった．しかし共有メモリへのアクセス速度が遅いため十分な性能であるとはいえない．これは現在開発中の次期 AWG-STAR システムで共有メモリのアクセス速度が改善されることにより，全体の計算性能も大幅に向上すると考えられる．

**謝辞**

本研究を進めるにあたり，日本電信電話株式会社フォトニクス研究所の岡田顕氏に多大なご支援を頂いた．深く謝意を示す．

**文　献**

[1] H. Nakamoto, K. Baba, and M. Murata, “Shared memory access method for a λ computing environment,” Proc. of IFIP OpNeTec, pp.210–217, Oct. 2004.

[2] E. Taniguchi, K. Baba, and M. Murata, “Implementation and evaluation of shared memory system for establishing λ computing environment,” Proc. of OECC2005, 5A2-3, pp.20–21, July 2005.

[3] A. Okada, H. Tanobe, and M. Matsuoka, “Dynamically reconfigurable real-time information-sharing network system based on a cyclic-frequency AWG and tunable-wavelength lasers,” Proc. of ECOC2003, Sept. 2003.

[4] OpenMP Architecture Review Board, OpenMP Application Program Interface Version 2.5, May 2005.

[5] 合田 圭吾，井本 舞，藤本 典幸，馬場 健一，村田 正幸, “λ コンピューティング環境における OpenMP ライブラリの設計と実装,” 信学技報 PN2006-40, vol.106, no.419, pp.5–8, Dec. 2006.

[6] 井本 舞，合田 圭吾，馬場 健一，村田 正幸, “λ コンピューティング環境における OpenMP ライブラリのためのデータ共有機構の設計,” 信学技報 PN2006-28, pp.19–24, Oct. 2006.

[7] V. Dimakopoulos, E. Leontiadis, and G. Tzoumas, “A portable C compiler for OpenMP V. 2.0,” Proc. of the European Workshop on OpenMP (EWOMP03), Sept. 2003.

[8] M. Sato, H. Harada, and A. Hasegawa, “Cluster-enabled OpenMP: An OpenMP compiler for SCASH software distributed shared memory system,” Scientific Programming, vol.9, no.2, pp.123–130, 2001.

[9] “SCore.” available at `http://www.pccluster.org`.