Omni OpenMP コンパイラ用並列プログラム可視化ツール

上嶋 明[†] 小畑正貴[†] 金田悠紀夫[††]

本論文では Omni OpenMP コンパイラ用の並列プログラム可視化ツールについて述べる．OpenMP を用いるとプログラマが直接スレッド操作コードをソースプログラム中に記述する必要がないため容易に並列プログラミングを行える．しかし，fork/join などのスレッド操作が隠蔽されて見えにくくなるため，並列プログラムの実行性能を解析したり改善したりすることが困難である．そこで，我々は Onmi OpenMP 向けの可視化ツールを作成した．ソフトウェア分散共有メモリシステム SCASH 上の Omni へも対応することで PC クラスタ環境でも使用可能とした．本ツールにより，OpenMP による並列プログラムの実行状態を視覚的に確認したり，実行時間や待ち時間などの各種統計情報を分析したりすることが可能となった．

A Parallel Program Visualization Tool for Omni OpenMP Compiler

Akira Uejima,[†] Masaki Kohata[†] and Yukio Kaneda[††]

In this paper, we propose a parallel program visualization tool for Omni OpenMP compiler. OpenMP is a portable model and an easy way for parallel programming on shared memory parallel computers because programmers do not have to write thread codes in the source program directly. However, thread operations (i.e. fork/join procedure calls and so on) are concealed, it is difficult for programmers to analyze and optimize the execution performance of parallel programs. We develop the visualization tool for Omni OpenMP and also adapt it for on SCASH software distributed shared memory system. Using our tool, programmers can see the visualized execution trace and statistics of OpenMP parallel programs.

1. はじめに

共有メモリ型並列プログラミングモデルである OpenMP[1] の出現により，並列化指示文（directive）をソースコード中に挿入することで並列プログラムを作成可能となった．OpenMP による並列プログラミングでは，スレッド生成などの複雑なコードをプログラマが記述する必要がない，多くのベンダが参加した統一規格なので移植性が高い，逐次コードと並列コードを一元管理できる，など多くの長所がある．しかしその反面，スレッドの生成やバリア同期などの挙動が隠蔽されてプログラマから見えにくくなり，作成した並列プログラムが期待した速度向上を得られない場合には性能改善を行うことが困難であることが多い．そこで本研究では OpenMP 処理系の 1 つとして Omni OpenMP コンパイラ[2),3)] を対象とし，並列プログラムの実行状態の可視化を可能とするツールを作成する．

OpenMP は本来共有メモリ型のプログラミングモデルであるが，ソフトウェア分散共有メモリシステム SCASH[4] 上の Omni OpenMP（Omni/SCASH）では PC クラスタ上でも使用可能となっている[5),6)]．

並列プログラムの性能チューニング用ツールとして，Intel Trace Analyzer/Collector[7)]，Intel Thread Profiler[8)]，PARAVER[9)] などがある．Intel Trace Analyzer/Collector は MPI ライブラリなどで構成される Cluster Tools の一部である．Trace Collector のライブラリをリンクした MPI プログラムを実行することで発生した通信などのイベントログを収集でき，Trace Analyzer により時刻ごとのイベントの内容を示すタイムライン表示や，通信に関する各種統計情報などの表示を行う．Intel Thread Profiler は OpenMP や POSIX などのスレッドコードに対するツールであり，統計収集用ライブラリをリンクした OpenMP プログラムを実行することで，並列実行/ 逐次実行各々の時間，バリア同期やクリティカル区間などの実行

† 岡山理科大学工学部
Faculty of Engineering, Okayama University of Science
†† 関西学院大学理工学部
School of Science and Technology, Kwansei Gakuin University

回数/待ち時間/実行時間の統計情報をリージョン単位やスレッド単位で表示できる．PARAVER は MPI, OpenMP，MPI+OpenMP ハイブリッド，Java など多くの実行モデルに対応する汎用的な可視化環境である．OMPtrace モジュールにより，OpenMP のランタイムライブラリやサブルーチンなどで実行時のログを収集し，各指示文で発生したイベントのタイムライン表示や，統計情報のグラフ表示などを行える．

Omni OpenMP には可視化ツールとして tlogview[10] が含まれる．Omni OpenMP 標準のランタイムライブラリ内に tlogview 用のログ収集を行うルーチンが用意されており，環境変数を設定して対象プログラムを実行すればログが収集できる．イベントの視覚的なタイムライン表示が可能であるが，PC クラスタ環境には対応しておらず，また，対応しないイベント（master，atomic，ordered など）も存在する．

本ツールは Omni OpenMP 用であり，tlogview と同様にランタイムライブラリ内でログ収集を行う．さらに本研究では，Omni OpenMP フロントエンドにおいて元のソースコードをスレッド API 呼び出しを含むコードに変換する際，ログ収集関数を自動挿入することで，ランタイムライブラリ内では収集不可能なログの収集も可能とした．本研究で変更を加えた Omni OpenMP コンパイラで対象となるソースコードをコンパイルして実行すれば両方法によるイベントログが収集され，より多くの情報の可視化を実現できる．

OpenMP 指示文によるイベントをタイムライン表示するのは PARAVER や tlogview と同様である．しかし，これらにない特徴として，for ループにおけるスケジューリング方法（ブロック，サイクリックなど）を視覚的に表示することや，Omni/SCASH による PC クラスタ環境での OpenMP の実行にも対応し，全体の性能に大きな影響を及ぼす SCASH バリア同期に要する時間についても他のバリア同期と区別して表示が可能であることなどがあげられる．

また，タイムライン表示においてパラレルリージョンでのスレッドの fork/join の様子を分かりやすく線で表現したり，critical 区間へ入る待ち状態やバリア同期実行中など，待ち状態を統一して破線で表示することで，並列プログラムの挙動をより直感的に理解・分析しやすいようにしている．ユーザインタフェースでは，パラレルリージョンのタイムライン表示の横に対応する統計情報やソースコードを表示するためのボタンを配置するなど，容易に操作できるよう配慮した．これらにより，Omni OpenMP による並列プログラムの実行状態の分析から，並列プログラミング初心者の学習にまで用いることができると考えられる．

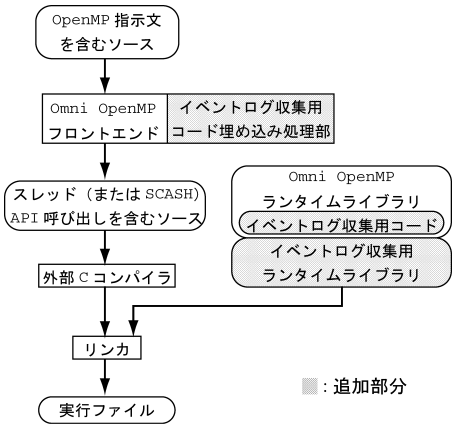

図 1 Omni OpenMP とイベントログ収集
Fig. 1 Omni OpenMP and event log collection.

本論文では，まず Omni OpenMP でコンパイルされた対象プログラムの実行時に各種イベントログ収集を行うための方法，次にそのログ情報を基に視覚的な表示を行うための可視化アプリケーションについてそれぞれ述べ，最後に PC クラスタ上で動作する並列プログラムに実際に適用した場合の結果について述べる．

2. Omni OpenMP と並列プログラム実行時イベントログ収集方法

2.1 Omni OpenMP

OpenMP の処理系の 1 つである Omni OpenMP は，フロントエンドとランタイムライブラリで構成される[2]．まず，フロントエンドにより OpenMP 指示文を含むソースコードをスレッドや SCASH の API を使用する並列 C ソースコードに変換し，次にそれを外部コンパイラ（gcc など）に処理させることで最終的な実行ファイルを生成する（**図 1**）．対象プログラム実行時のログ収集を行うため，本研究では網掛け部分を追加した．

ユーザが OpenMP 指示文を含むソースコードをコンパイルする際，後述のログ収集機能を加えられた Omni OpenMP を用いて実行ファイルを生成する．イベントログ収集用ランタイムライブラリは Omni OpenMP ランタイムライブラリとともに対象プログラムの実行ファイルにリンクされる．このプログラムを実行すれば自動的にイベントログ収集が行われ，プログラム終了時に可視化用のファイル出力が行われる．

2.2 イベントログ収集方法

対象プログラムの実行状態の可視化を行うため，発生したイベントの種類や時刻などのログ情報を記録す

```
#pragma omp critical
{
  sub1();
}
```

(a) Original source code.

```
{
  _ompc_enter_critical(&__ompc_lock_critical);
  sub1();
  _ompc_exit_critical(&__ompc_lock_critical);
}
```

(b) Translated source code.

図 2 Omni OpenMP フロントエンドによる critical 指示文の変換
Fig. 2 Translation of a critical directive by Omni OpenMP frontend.

```
void _ompc_enter_critical(_ompc_lock_t **p)
{
  _prof_log(CRITICAL_WAIT);  ← critical 待ち

     .
     . (Omni OpenMP ランタイムライブラリコード)
     .

  _prof_log(CRITICAL_START); ← critical 開始
}

void _ompc_exit_critical(_ompc_lock_t **p)
{
     .
     . (Omni OpenMP ランタイムライブラリコード)
     .

  _prof_log(CRITICAL_END);   ← critical 終了
}
```

図 3 Omni OpenMP ランタイムライブラリ内でのイベントログ収集
Fig. 3 Event log collection in Omni OpenMP runtime library.

る必要がある．Omni OpenMP では環境変数の設定により tlogview 用のログを収集することができるが，本研究ではより多くのイベントについてのログを使用するため，独自にイベントログを収集することとした．

ログは本研究で作成したイベントログ収集用ランタイムライブラリ内の関数を呼び出すことで記録される．ログ収集で生じるオーバヘッドを減らすため，本研究では対象プログラム実行前に蓄積可能なイベントの最大数を環境変数により指定しておき，ログ格納用メモリを事前に割り当てることにした．

指定を超えたイベントが発生した場合にはログの蓄積を中止する．ただし，発生したイベント数のカウントのみを最後まで続け，対象プログラム終了時にこの値を表示する．表示されたイベント数を環境変数で指定すれば，次回の対象プログラム実行時には必要となる領域を事前に確保できる．もし対象プログラムが大規模で全体のログを蓄積できない場合，ログ収集を開始，および終了するパラレルリージョンを指定しておけば，その区間のみのログの収集が可能である．

イベント発生時刻としてシステムクロックを取得しているため，精度は環境に依存する．本研究の実験環境において時刻の分解能は 1 μs，標準的なイベント 1 個のログ収集に要するオーバヘッドは 0.6〜3.1 μs であった．

対象プログラム実行時のイベントログ収集には次の 2 通りの方法を用いる．

2.2.1 ランタイムライブラリ内でのイベントログ収集

critical 指示文の含まれるソースコードが Omni OpenMP フロントエンドで変換された例を **図 2** に示す．critical 区間として実行される部分の前後に `_ompc_`から始まる関数呼び出しがあるが，これは Omni OpenMP ランタイムライブラリ関数の呼び出しである．このように Omni OpenMP ランタイムライブラリにより開始と終了の処理がされる指示文については，ランタイムライブラリ内にイベントログ収集用関数を追加すればよい．

critical 区間の前後で使用される Omni OpenMP ランタイムライブラリ関数のコードを **図 3** に示す．本研究で追加したのは `_prof_log()` というイベントログ収集関数の呼び出しコードであり，引数としてイベントの種類を与えている．

2.2.2 コード挿入によるイベントログ収集

master 指示文の含まれるソースコードが Omni OpenMP フロントエンドで変換された例を **図 4** (a), (b) に示す．2.2.1 項で述べた critical 指示文の変換と違い，このコードを実行中のスレッドがマスタスレッドであるか否かの確認のために Omni OpenMP のランタイムライブラリ関数 `_ompc_is_master()` のみが用いられ，master 実行区間は if 文のブロックに展開されている．

この 図 4 (b) のような場合，区間の終了時には Omni OpenMP ランタイムライブラリが呼び出されないため，critical 指示文のときのようにランタイムライブラリにイベントログ収集関数呼び出しを追加する方法では終了時刻を知ることができない．そこで，本研究では Omni OpenMP フロントエンドに変更を加えることで，生成される並列 C ソースコード内にイベントログ収集関数呼び出しを自動挿入するようにした．

```
#pragma omp master
{
  sub2();
}
```

(a) Original source code.

```
if(_ompc_is_master()){
  sub2();
}
```

(b) Translated source code by original Omni OpenMP.

```
if(_ompc_is_master()){
  _prof_master_start(); ← master 開始（自動挿入）
  sub2();
  _prof_master_end();  ← master 終了（自動挿入）
}
```

(c) Translated source code by modified Omni OpenMP.

図 4 Omni OpenMP フロントエンドによる master 指示文の変換
Fig. 4 Translation of a master directive by Omni OpenMP frontend.

変更を加えたフロントエンドで変換されたソースコードを 図 4 (c) に示す．_prof_で始まる行が自動的に挿入されたイベントログ収集関数呼び出しである．

2.3 PC クラスタへの対応

本ツールは SMP 環境の他，Omni/SCASH による PC クラスタ環境へも対応した．Omni/SCASH においては使用できる共有メモリ領域が少なく，またこれら領域の使用によりオーバヘッドも増加すると考えられる．そこで，ログを格納する領域として各ノードのローカルのメモリ領域を使用する．各イベントは発生したノード内に記録しておき，対象プログラム終了後に各ノードに蓄積されたログ情報をノードごとのファイルに出力する．

一般的に PC クラスタにおいては各ノードの時計が一致していないため，上記の方法では同一時刻に発生したイベントであっても各ノードごとに記録される時刻が異なってしまう．そこで，対象プログラムの実行開始前に行う SCASH バリア同期の完了時刻を基準時として記録しておき，可視化アプリケーションにおける各イベントの発生時刻として，この基準時からの相対時刻を用いる．また，実行開始後にもノード間の時刻の差が生じ，特に対象プログラムの実行時間が長い場合に問題となる可能性があるため，各パラレルリージョン実行前の SCASH バリア同期が発生したタイミングで各ノード間の時刻の差を計算し，オフセットを調整する．

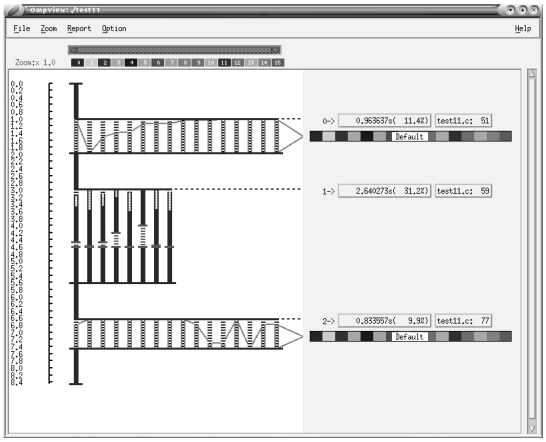

図 5 ログ可視化アプリケーションのメイン画面
Fig. 5 Main window of the log visualization application.

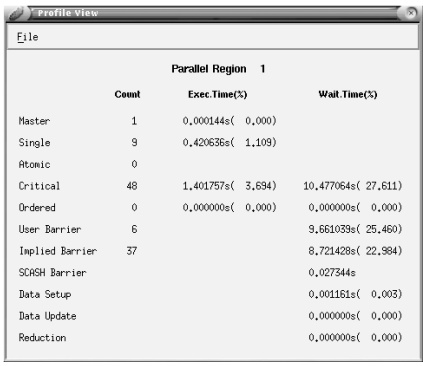

図 6 パラレルリージョン統計画面
Fig. 6 Statistics window for a parallel region.

3. ログ可視化アプリケーション

3.1 概　　要

対象プログラム実行時に収集したイベントログ情報を表示するための GUI アプリケーションを作成した．メイン画面を **図 5** に示す．画面左半分にスレッドの実行状態をタイムライン表示しており，縦軸が時間，横軸がスレッドに対応している．

アプリケーション起動時には全実行時間分を表示するが，倍率を変更して一部を拡大表示させることもできる．実行状態は同時に 16 スレッド分を表示可能であり，スレッド数がこれを超える場合には横方向にスクロールさせることで全スレッドの状態を表示する．

画面右半分には各パラレルリージョンごとにボタンを 2 つずつ配置している．左側ボタンにはパラレルリージョンの実行時間と，その時間が対象プログラム中で占める割合（%）を表示しており，押下することで詳細な統計情報を表示する（**図 6**）．この画面では

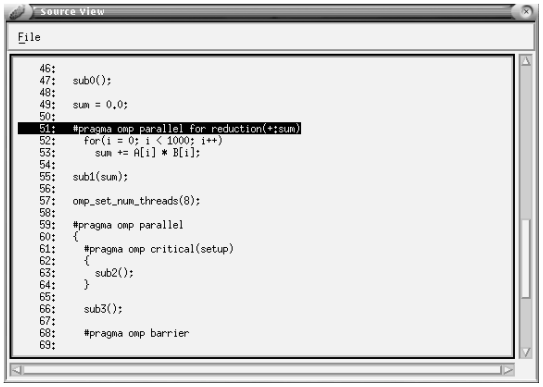

図 7 ソースコード表示画面
Fig. 7 Source code viewer window.

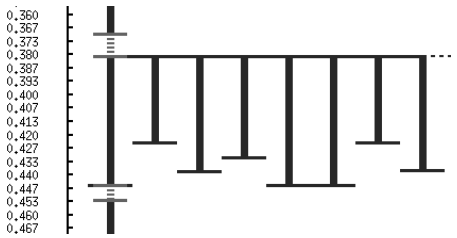

図 8 パラレルリージョン表示
Fig. 8 Visualization of a parallel region.

各指示文ごとの実行回数，実行時間，待ち時間，および各々の時間が全実行時間に占める割合（%）などを表示する．

右側ボタンにはソースファイル名と行番号を表示しており，押下することで対応するソースコードを表示する．この際，parallel 指示文を含む行を白黒反転して示す（**図 7**）ことで，ソースコード中のどの部分に対応するパラレルリージョンであるかを確認することが可能である．

3.2 スレッドの状態表示

画面左半分に表示されるスレッドの実行状態表示について述べる．パラレルリージョンの表示例を **図 8** に示す．左から，時間軸の表示に続きスレッド 0，スレッド 1，... という順に並んでおり，処理が行われている部分に縦線が引かれる．図 8 では，まずマスタスレッド（スレッド 0）のみで処理が開始され，パラレルリージョンとして 8 スレッドによる並列処理が行われ，その後再びマスタスレッドのみで処理が行われていることを表している．各スレッドの終了時刻を比較することにより負荷均衡を確認できる．

なお，Omni/SCASH の場合にはパラレルリージョンの前後で SCASH のバリア同期（SCASH barrier）が実行されるので，この処理区間についてはマスタスレッドの部分に破線で表示している．

表 1 スレッドの状態の表示方法
Table 1 Display types for thread states.

表示	処理
実行区間	master，single，section，atomic，for
待ち区間と実行区間	critical，ordered
待ち区間	barrier，暗黙 barrier，SCASH barrier，データ準備（firstprivate，copyin），データ更新（lastprivate），リダクション

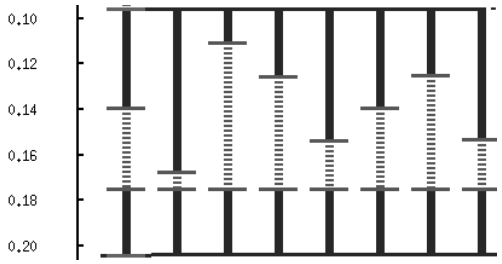

図 9 バリア同期表示
Fig. 9 Visualization of a barrier synchronization.

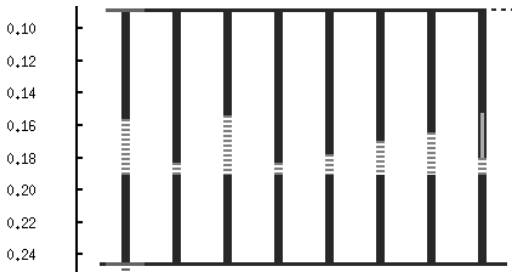

図 10 single 指示文と暗黙バリア同期表示
Fig. 10 Visualization of a single section and an implied barrier synchronization.

各スレッド内の実行状態として表示する内容を **表 1** に示す．これらの処理うち，主要なものについて表示方法を述べる．

3.2.1 barrier 指示文

barrier 指示文の表示例を **図 9** に示す．各スレッドの破線部分はバリア同期完了待ちであることを示している．バリア同期が完了すると破線を終了する．

3.2.2 single 指示文と暗黙バリア同期

single 指示文の表示例を **図 10** に示す．図 10 のスレッド 7 において，実行状態であることを表す縦線の内部に異なる色の細い実線を表示し，single 区間の実行中であることを示している．

OpenMP においては single 区間や for ループの実行にともない自動的にバリア同期（暗黙バリア同期）が行われる．このバリア同期については通常のバリアとは異なる破線で表示する．図 10 では，スレッド 7

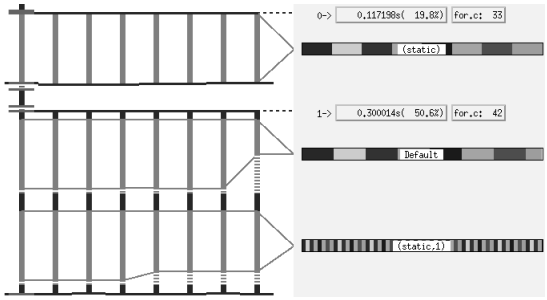

図 11 for ループ表示
Fig. 11 Visualization of for-loops.

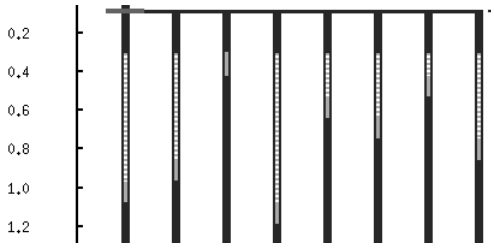

図 12 critical 区間表示
Fig. 12 Visualization of critical sections.

で実行された single 区間にともなって全スレッドで暗黙バリア同期が行われていることを示している．

3.2.3 for ループ

for ループの表示例を **図 11** に示す．for ループが実行されている区間のスレッド実行状態の色を変えるとともに，1 つの for ループについて各スレッドにおいての開始点，終了点をそれぞれ線で結んでスレッド間での対応を表示している．また，schedule 指示節によるループスケジューリング（static，dynamic，guided など）とチャンクサイズを表示し，スケジューリングが static の場合にはループの範囲が各スレッドにどのように割当てされているか（ブロックやサイクリック）を画面右側に図で表示する．

図 11 では，3 個の for ループが実行されている．最初の for ループは static スケジューリングのブロック割当てにより実行されていることが分かる．2 番目の for ループはスケジューリングの指定なし（Default）で実行され，スレッド 7 に割り当てられるブロックが小さいこと，またスレッド 7 のみ処理時間が短く，for ループの実行時間がスレッド間で不均衡になっていることが分かる．最後の for ループは static スケジューリングでチャンクサイズが 1 に指定されており，サイクリック割当てになっていることが分かる．

3.2.4 critical 指示文

critical 指示文の表示例を **図 12** に示す．各スレッドの縦線の内部に細い破線表示を行い，critical 区間へ入る条件が成立するまでの待ち状態にあることを示している．同様に内部の実線部分は critical 区間の実行中であることを示している．

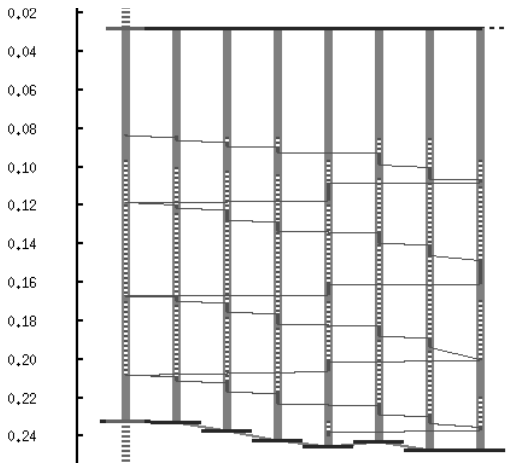

図 13 ordered 区間表示
Fig. 13 Visualization of ordered sections.

表 2 実験環境
Table 2 Experimental environment.

ノード数	8
SCore	5.4
Omni OpenMP	1.6
OS	Red Hat Linux 7.3
CPU	Intel Pentium 4 HT 2.8 GHz (Hyper Threading 無効)
メモリ	1 GB
NIC	Intel PRO/1000MT (1000BASE-T, 32 bit PCI)
スイッチ	TecnoGraphy GR2600

3.2.5 ordered 指示文

ordered 指示文は for ループ内部で指定した区間を逐次実行した場合と同じ順序に実行させるものである．表示例を **図 13** に示す．for ループの実行状態を示す太い線の中に critical と同様の細い破線と実線の表示を行うことで，for ループ内に存在する ordered の待ち状態と実行状態を示している．さらに，各スレッドに存在する ordered 区間が実際に実行された順序を線で接続して表示している．

3.3 統計情報出力

各指示文や内部処理について実行回数，実行時間，待ち時間と各々の時間が占める割合などの統計情報レポートをプログラム全体，および各パラレルリージョンごとに集計してテキストファイルに出力する．実際の出力例については 4.2 節で示す．

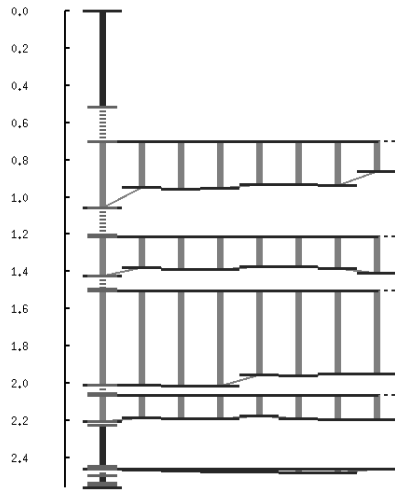

図 14 JPEG エンコーダの実行状態
Fig. 14 Timeline display of a JPEG encoder.

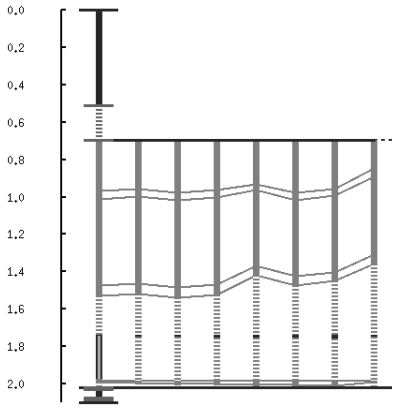

図 15 改良した JPEG エンコーダの実行状態
Fig. 15 Timeline display of a refined JPEG encoder.

4. 実験結果

表 2 の実験環境に示す PC クラスタを用い，2 種類の並列プログラムの実行状態の可視化を試みた．

4.1 JPEG エンコーダ

静止画像データ圧縮方法の 1 つである JPEG のエンコーダを対象として用いた．実行結果を **図 14** に示す．JPEG エンコーダは色空間変換，MCU 構成，DCT 変換，量子化，エントロピー符合化の各処理部からなり[11]，ファイル入力の後，各処理部それぞれに対する for ループとパラレルリージョンを 5 組構成し，最後にファイル出力を行っている．基本的に 8×8 画素からなるブロック単位で独立して計算を行えることを利用して並列化するが，エントロピー符合化のうち DC（直流）成分の差分値を計算する部分ではブロック間の依存関係があるため，逐次処理を行っている．図 14 より，並列処理部分の割合やスレッド間の負荷均衡の状態，また各処理部のうち DCT 変換に最も時間を要していることなどを確認できる．Omni/SCASH においてネットワークとして Ethernet を用いるクラスタの場合には SCASH のページ転送によるオーバヘッドが大きくなることが報告されており[6]，図 14 においてもパラレルリージョンの前後で SCASH バリア同期のオーバヘッドが生じていることが分かる．すなわち，この例のように Ethernet を使用する PC クラスタ上の OpenMP においては，単純に for ループを paralle for 指示文で並列化したのみでは良い性能が得られないことがある．

次に，並列化指示文に次の変更を加えて実験した．

- 5 個存在したパラレルリージョンを 1 個に統合
- 色空間変換，MCU 構成，DCT 変換の各 for ループに nowait 指示節を追加して終了後の暗黙バリア同期を除去
- エントロピー符合化処理のうち依存関係のある部分をパラレルリージョン内で single 指示文により逐次処理

改良した JPEG エンコーダの実行結果を **図 15** に示す．パラレルリージョンが 1 個になったことによるオーバヘッドの減少などにより性能が改善され，ファイル入出力を除く計算部分のみの比較では元のプログラムに対して速度が約 23%向上した．

なお，今回は評価を行っていないが，Omni/SCASH で拡張されたデータマッピング指示文と affinity スケジューリングを組み合わせることで，さらに性能を向上させることが可能であると考えられる．

4.2 NPB

ベンチマークの 1 つである NPB（NAS Parallel Benchmarks）の OpenMP C 版[12] のうち，FT（FFT を用いた 3 次元偏微分方程式の解法）のクラス W の実行状態を可視化した結果の前半部分を **図 16** に，また，統計情報レポートの一部を **図 17** に示す．これらより，パラレルリージョン数を 2 個のみにしていること，また，single 実行時間，critical 待ち時間，バリア同期待ち時間の割合が多いことなどが分かる．なお，このプログラムは Omni/SCASH 用には最適化されておらず，実際に本研究の Ethernet を使用する環境においては逐次処理に対する速度向上を得られなかった．

5. おわりに

Omni OpenMP コンパイラ用の並列プログラム可

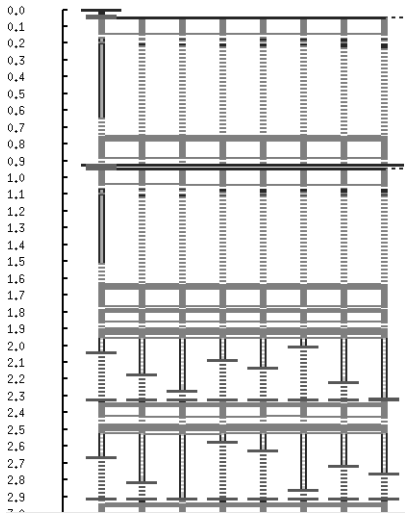

図 16 NPB FT（クラス W）の実行状態
Fig. 16 Timeline display of NPB FT (class W).

```
********** Program: ./ft.W **********
Number of Nodes           :   8
Number of Parallel Regions:   2

***************************** Total ******************************
Program  Exec. Time   :     5.729239s
Parallel Exec. Time(%):     5.617215s(  98.04%)

                  Count    Exec. Time(%)           Wait. Time(%)
        Master:     1      0.000144s(  0.000)
        Single:    11      0.870216s(  1.936)
        Atomic:     0
      Critical:    48      1.401757s(  3.119)     10.477064s( 23.315)
       Ordered:     0      0.000000s(  0.000)      0.000000s(  0.000)
   User Barrier:    6                              9.661039s( 21.499)
Implied Barrier:   43                             13.394459s( 29.807)
  SCASH Barrier:                                   0.054688s(  0.955)
     Data Setup:                                   0.001165s(  0.003)
    Data Update:                                   0.000000s(  0.000)
      Reduction:                                   0.000000s(  0.000)

************* Parallel Region    0 (ft.c: 123) **************
Region Exec. Time(%):       0.874060s(15.256)
Number of Threads   :       8
Shared Local Data   :       0B

                  Count    Exec. Time(%)           Wait. Time(%)
        Master:     0      0.000000s(  0.000)
        Single:     2      0.449580s(  6.429)
        Atomic:     0
      Critical:     0      0.000000s(  0.000)      0.000000s(  0.000)
       Ordered:     0      0.000000s(  0.000)      0.000000s(  0.000)
   User Barrier:    0                              0.000000s(  0.000)
Implied Barrier:    6                              4.673031s( 66.829)
  SCASH Barrier:                                   0.027344s
     Data Setup:                                   0.000004s(  0.000)
    Data Update:                                   0.000000s(  0.000)
      Reduction:                                   0.000000s(  0.000)
                                  .
                                  .
                                  .
```

図 17 NPB FT（クラス W）の統計情報レポート
Fig. 17 Statistics report of NPB FT (class W).

視化ツールについて述べた．Omni OpenMP ランタイムライブラリの変更とフロントエンドの変更により対象プログラム実行時の各種イベントログ収集を可能にし，その情報を表示するアプリケーションを作成した．また，Omni/SCASH に対応することで PC クラスタ環境でも使用可能とした．本ツールにより，OpenMP による並列プログラム実行状態を視覚的に確認したり，実行時間や待ち時間などの統計情報を分析したりすることが可能となった．2 種類の並列プログラムを対象として本ツールを適用し，このうち JPEG エンコーダについては並列化指示文の変更による性能向上の例を示した．

今後の課題として，次にあげることを主に検討している．

- Myrinet などの Ethernet 以外のネットワークを使用した場合の動作検証や SMP クラスタへの対応を行う．
- Omni/SCASH においてデータマッピング方法による性能の比較を可能にする．
- 大規模問題にも対応できるよう，特定条件に一致するイベントのみのログ収集や，対象プログラム中でのログ収集開始・停止の指示を可能にする．
- 大規模クラスタなどでスレッド数が多い場合のタイムライン表示の方法を改善する．

謝辞 JPEG エンコーダの並列化について有益な情報をいただいた立命館大学理工学部の山崎勝弘教授に謝意を表します．

参 考 文 献

1) OpenMP. http://www.openmp.org/
2) Omni OpenMP Compiler Project. http://phase.hpcc.jp/Omni/
3) 草野和寛，佐藤茂久，佐藤三久：Omni OpenMP コンパイラの性能評価，情報処理学会論文誌，Vol.42, No.4, pp.802–811 (2001).
4) Harada, H., Ishikawa, Y., Hori, A., Tezuka, H., Sumimoto, S. and Takahashi, T.: Dynamic Home Node Reallocation on Software Distributed Shared Memory, *Proc. HPC Asia 2000*, Beijing, China (2000).
5) 佐藤三久，原田 浩，長谷川篤史，石川 裕：Cluster-enabled OpenMP：ソフトウェア分散共有メモリシステム SCASH 上の OpenMP コンパイラ，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.42, No.SIG 9, pp.158–169 (2001).
6) 小島好紀，佐藤三久，原田 浩，石川 裕，朴 泰祐，高橋大介：Ethernet によるクラスタ上での分散共有メモリ OpenMP Omni/SCASH の性能評価，情報処理学会研究報告 2002-HPC-91-21, pp.119–124 (2002).
7) Intel Cluster Tools. http://www.intel.com/software/products/cluster/
8) Intel Thread Profiler. http://www.intel.com/software/products/threading/tp/
9) PARAVER. http://www.cepba.upc.es/paraver/

10) Omni Profile Visualization Tool. http://phase.hpcc.jp/Omni/Omni-doc/tlogview.html
11) 小野定康，鈴木純司：わかりやすい JPEG/MPEG 2 の技術，オーム社 (2001).
12) NAS Parallel Benchmarks in OpenMP. http://phase.hpcc.jp/Omni/benchmarks/NPB/

(平成 17 年 1 月 24 日受付)
(平成 17 年 5 月 14 日採録)

上嶋　明（正会員）

1968 年生．1991 年立命館大学理工学部情報工学科卒業．1993 年同大学院博士前期課程修了．博士（工学）．立命館大学理工学部助手，神戸大学大学院自然科学研究科助手を経て，2003 年岡山理科大学工学部講師．並列処理，コンピュータ・グラフィックスの研究に従事．電子情報通信学会，IEEE，ACM 各会員．

小畑　正貴（正会員）

1957 年生．1985 年神戸大学大学院博士後期課程修了．学術博士．1984 年岡山理科大学助手．1996 年同大学教授．2001 年倉敷芸術科学大学教授．2004 年岡山理科大学教授，現在に至る．計算機アーキテクチャ，並列処理，相互結合網，リコンフィギャラブルシステムに関する研究に従事．電子情報通信学会，IEEE，ACM 各会員．

金田悠紀夫（正会員）

1966 年神戸大学大学院工学研究科修士課程電気工学専攻修了．電気試験所（電総研）電子計算機部，神戸大学工学部情報知能工学科等を経て，現在，関西学院大学理工学部情報科学科教授．工学博士．計算機アーキテクチャ，高級言語指向コンピュータ，並列計算機，並列計算ソフトウェアの研究に従事．