Oracle® Solaris Studio 12.3: パフォーマンスアナライザの MPI のチュートリアル

Part No: E28031
2012 年 1 月

ORACLE

120220@25097

目次

はじめに

このドキュメントでは、MPI を使用するプログラムのパフォーマンスアナライザ機能について説明します。

サポートされるプラットフォーム

この Oracle Solaris Studio リリースは、Oracle Solaris オペレーティングシステムを実行する SPARC ファミリのプロセッサアーキテクチャーを使用するプラットフォームと、Oracle Solaris または特定の Linux システムを実行する x86 ファミリのプロセッサアーキテクチャーを使用するプラットフォームをサポートします。

このドキュメントでは、次の用語を使用して x86 プラットフォームの違いを示しています。

- 「x86」は、64 ビットおよび 32 ビットの x86 互換製品を指します。
- 「x64」は、特定の 64 ビット x86 互換 CPU を指します。
- 「32 ビット x86」は、x86 ベースシステムで特定の 32 ビット情報を指します。

Linux システムに固有の情報は、サポートされている Linux x86 プラットフォームだけに関連し、Oracle Solaris システムに固有の情報は、SPARC および x86 システムでサポートされている Oracle Solaris プラットフォームだけに関連します。

サポートされているハードウェアプラットフォームおよびオペレーティングシステムリリースの完全なリストについては、Oracle Solaris Studio 12.3 リリースノートを参照してください。

Oracle Solaris Studio ドキュメント

Oracle Solaris Studio ソフトウェアの完全なドキュメントは、次のように見つけることができます。

- 製品のドキュメントは、リリースノート、リファレンスマニュアル、ユーザーガイド、チュートリアルも含め、Oracle Solaris Studio Documentation Web サイトにあります。

- コードアナライザ、パフォーマンスアナライザ、スレッドアナライザ、dbxtool、DLight、および IDE のオンラインヘルプには、これらのツール内の「ヘルプ」メニューだけでなく、F1 キー、および多くのウィンドウやダイアログボックスにある「ヘルプ」ボタンを使用してアクセスできます。
- コマンド行ツールのマニュアルページでは、ツールのコマンドオプションが説明されています。

開発者向けのリソース

Oracle Solaris Studio を使用する開発者のための次のリソースを見つけるには、Oracle Technical Network Web サイトにアクセスしてください。

- プログラミング技術およびベストプラクティスに関する記事
- ソフトウェアの最近のリリースに関連する完全なドキュメントへのリンク
- サポートレベルに関する情報
- ユーザーディスカッションフォーラム。

Oracle サポートへのアクセス

Oracle のお客様は、My Oracle Support にアクセスして電子サポートを受けることができます。詳細は、`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` にアクセス、または、聴覚に障害がある方は、`http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` にアクセスしてください。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1　表記上の規則

字体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	`.login` ファイルを編集します。 `ls -a` を使用してすべてのファイルを表示します。 `system%`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	`system%` **`su`** `password:`

表 P–1 表記上の規則 *(続き)*

字体または記号	意味	例
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、`rm` *filename* と入力します。
『』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	`sun% grep '^#define \` `XV_VERSION_STRING'`

Oracle Solaris OS に含まれるシェルで使用する、UNIX のデフォルトのシステムプロンプトとスーパーユーザープロンプトを次に示します。コマンド例に示されるデフォルトのシステムプロンプトは、Oracle Solaris のリリースによって異なります。

- C シェル

  ```
  machine_name% command y|n [filename]
  ```

- C シェルのスーパーユーザー

  ```
  machine_name# command y|n [filename]
  ```

- Bash シェル、Korn シェル、および Bourne シェル

  ```
  $ command y|n [filename]
  ```

- Bash シェル、Korn シェル、および Bourne シェルのスーパーユーザー

  ```
  # command y|n [filename]
  ```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: `Shift` キーを押します)。ただし、キーボードによっては `Enter` キーが `Return` キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、`Ctrl-D` は `Control` キーを押したまま `D` キーを押すことを意味します。

◆ ◆ ◆ 第 1 章

1 パフォーマンスアナライザの MPI のチュートリアル

このチュートリアルは、Oracle Solaris Studio 12.3 パフォーマンスアナライザの MPI 機能の使用方法を説明するために、最初から最後まで、それに従って進むよう設計されています。このチュートリアルでは、サンプルプログラムを使って「MPI タイムライン」および「MPI チャート」タブの使用方法を説明します。

MPI とパフォーマンスアナライザについて

MPI とは Message Passing Interface であり、並列および分散コンピューティングに使用される標準化された API です。このドキュメントは、MPI を使用し、クラスタなどの分散システムで動作するアプリケーションの開発経験があることを前提としています。このドキュメントでは、分散コンピューティング環境の設定方法や MPI の使用方法については説明していません。

パフォーマンスアナライザを使って MPI アプリケーションを検査すると、次の疑問を解決できます。

- MPI コードのチューニングによってパフォーマンスが大幅に向上するか。
- MPI パフォーマンスは同期またはデータ転送を特徴とするか。
- プログラムにロードインバランスが含まれているか。
- プログラム実行の 1 回の反復にかかる時間はどれくらいか。
- プログラムパフォーマンスの均衡化にかかる時間はどれくらいか。
- プログラム実行におけるメッセージ受け渡しパターンはなにか。
- もっとも重要なのはロングまたはショートメッセージのどちらか。
- メッセージを送信するプロセスはメッセージを受信するプロセスと同期するか。

前述のリストは範囲が広すぎて 1 冊のドキュメントでは扱いきれませんが、このチュートリアルでは次を含むパフォーマンスアナライザのいくつかの新機能の使い方を案内します。

- MPI タイムライン。アプリケーションの実行中に発生した MPI アクティビティーのグラフィカル表示。

- MPI チャート。散布図とヒストグラムを生成して MPI 関数と MPI メッセージのパフォーマンスデータを図形化するツール。
- MPI データズームおよびデータフィルタリング。「MPI タイムライン」および「MPI チャート」のデータのビューを広げたり狭めたりするために使用できる 1 組のコントロール。

「MPI タイムライン」タブは、テストプログラムの実行からのデータをタイムラインとして示します。最初、タイムラインのビューにはすべての MPI 関数と MPI メッセージがグラフィカル表示された、実行の始めから終わりまでが含まれています。この表示を拡大したり、完全なビューから移行し、場合によっては 1 つの関数と同程度の粒度の細かさで、しっかりと焦点を絞り込んだビューまで狭めたりする方法を習得します。「MPI タイムライン」タブは、「MPI チャート」タブとともに、データのズーム、パン、および検査を行う多数のさまざまな方法を提供します。「MPI チャート」タブを使用すると、関数やメッセージに関する統計データをグラフィカルチャートで表示でき、実行時に何が発生しているのかを確認するのに役立ちます。

パフォーマンスアナライザについての詳細は、『Oracle Solaris Studio 12.3: パフォーマンスアナライザ』を参照してください。

チュートリアル用の設定

パフォーマンスアナライザは、Oracle Message Passing Toolkit、Oracle Sun x86 および SPARC ベースシステム用の高度に最適化された Open MPI の実装など、Message Passing Interface (MPI) 標準のいくつかの実装で動作します。

注 – Oracle Message Passing Toolkit 製品は、かつて Sun HPC ClusterTools と呼ばれていたため、Oracle の Web ページやドキュメントで両方の名称を目にすることがあります。Sun HPC ClusterTools の最終バージョンはバージョン 8.2.1c です。この製品の次のリリースは Oracle Message Passing Toolkit 9 であり、ソフトウェアパッケージリポジトリ内の Oracle Solaris 11 に使用できます。

パフォーマンスアナライザで動作する MPI バージョンの一覧を表示するには、追加の引数を指定しないでコマンド `collect` を実行します。

このチュートリアルでは、`ring_c` と呼ばれる MPI のサンプルアプリケーションでパフォーマンスアナライザを使用する方法を説明します。

このチュートリアル用にクラスタが構成され、機能している必要があります。

MPI ソフトウェアの取得

このチュートリアルでは Oracle からの MPI ソフトウェアが使用されますが、Open MPI の Web サイト (`http://www.open-mpi.org`)で入手できる Open MPI を使用することもできます。

Oracle Message Passing Toolkit ソフトウェアに関する情報は、`http://www.oracle.com/us/products/tools/message-passing-toolkit-070499.html` で入手できます。このサイトにはソフトウェアをダウンロードするためのリンクが用意されていますが、代わりに次の詳細な手順を使用してください。

Oracle Solaris 10 および Linux 用の MPI ソフトウェア

このチュートリアルでは、Oracle Solaris 10 または Linux に対して Sun HPC ClusterTools 8 とその更新を使用できます。Sun HPC ClusterTools 8.2.1c のダウンロードについては、後述の手順を参照してください。

Oracle Solaris 10 および Linux 用の Sun HPC ClusterTools をダウンロードするには:

1. My Oracle Support (`http://support.oracle.com`) にログインします。このソフトウェアをダウンロードするためには、Oracle Solaris または Oracle Solaris Studio のサポート契約が必要です。
2. 検索ボックスに「**`ClusterTools`**」と入力し、検索ボタンをクリックします。
3. ページの左側にある「検索の絞込み」領域の「Patches」をクリックします。
4. 「`HPC-CT-8.2.1C-G-F - HPC ClusterTools 8.2.1c.`」をクリックします。
5. 「ダウンロード」ボタンをクリックし、ダイアログボックス内の手順に従って、ClusterTools ソフトウェアが含まれている圧縮ファイルをダウンロードします。
6. ファイルがダウンロードされたら、それを展開し、システムの適切なプラットフォームディレクトリにナビゲートします。必要に応じて、このディレクトリ内のファイルをさらに展開します。
7. 展開されたディレクトリで PDF として使用可能な『Sun HPC ClusterTools 8.2.1c Software Installation Guide』の説明に従ってソフトウェアをインストールします。マニュアルに記載されている最上位ディレクトリの名前が展開されたディレクトリで使用されたパスと一致しないことがありますが、サブディレクトリとプログラム名は同じです。
8. ディレクトリ */Studio-installation*`/bin` および *ClusterTools-installation*`/bin` ディレクトリを自分のパスに追加します。

Oracle Solaris 11 用の MPI ソフトウェア

Oracle Solaris 11 が動作している場合は、Oracle Message Passing Toolkit がパッケージ名 `openmpi-15` のもとで Oracle Solaris 11 リリースの一部として使用可能になりま

す。Oracle Message Passing Toolkit のインストールと使用については、記事「How to Compile and Run MPI Programs on Oracle Solaris 11」を参照してください。

Oracle Solaris 11 でのソフトウェアのインストールに関する一般情報は、Oracle Solaris 11 ドキュメントライブラリ内のマニュアル『Oracle Solaris 11 ソフトウェアパッケージの追加および更新』を参照してください。

注 – Oracle Solaris 11 のパッケージ `openmpi-15` には、このチュートリアルで使用されるサンプルコードが提供されていません。このバージョンをインストールする場合は、それとは別に、12 ページの「Oracle Solaris 11 の Oracle Message Passing Toolkit 用のサンプルコード」の説明に従ってサンプルコードを取得する必要があります。Open MPI をダウンロードしてサンプルコードを入手することもできます。

サンプルソースコードの準備

サンプルソースコードの準備については、使用している MPI に適した後述のセクションを参照してください。

ClusterTools および Open MPI 用のサンプルコード

Sun HPC ClusterTools または Open MPI のどちらかをお持ちの場合は、すべてのクラスタノードからアクセスできる場所に `examples` ディレクトリの書き込み可能なコピーを作成する必要があります。

例:

- Sun HPC ClusterTools 8.2.1c をお持ちの場合は、書き込み権のある共有ディレクトリにディレクトリ `/opt/SUNWhpc/HPC8.2.1c/sun/examples` をコピーします。
- Open MPI 1.4.4, をお持ちの場合は、書き込み権のある共有ディレクトリにディレクトリ `openmpi-1.4.4/examples` をコピーします。

Oracle Solaris 11 の Oracle Message Passing Toolkit 用のサンプルコード

Oracle Solaris 11 の `openmpi-15` パッケージにはサンプルコードが含まれていません。`ring_c.c` のソースコードとそのプログラムを構築するための `Makefile` は、付録 B「チュートリアルのサンプルコード」に記載されています。それらのファイルのテキストをコピーし、自分で `ring_c.c` と `Makefile` を作成できます。

ファイルを作成するには:

1. すべてのクラスタノードからアクセスできる場所に examples というディレクトリを作成します。
2. マウスまたはキーボードを使用して、付録 B「チュートリアルのサンプルコード」から ring_c.c のソースコードと Makefile をコピーし、テキストエディタでそのテキストをペーストします。
3. それらのファイルを examples ディレクトリに保存します。

サンプルプログラムのコンパイルと実行

サンプルプログラムをコンパイルして実行するには:

1. 新しい examples ディレクトリに、ディレクトリを変更します。
2. ring_c の例を構築します。

```
% make ring_c
mpicc -g -o ring_c ring_c.c
```

このプログラムは -g オプションを使ってコンパイルされ、それによってパフォーマンスアナライザのデータコレクタが MPI イベントをソースコードにマップできるようになります。

3. mpirun で ring_c の例を実行して、それが正しく動作することを確認します。ring_c プログラムは、ring 内のプロセスからプロセスへメッセージを渡すだけで終了します。

この例は、2 ノードクラスタでプログラムで実行する方法を示しています。それらのノード名は、各ノードで使用されるスロットの数とともにホストファイル内に指定されます。このチュートリアルでは 25 個のプロセスを使用し、各ホストで 1 つのスロットを指定します。システムに適しているいくつかのプロセスとスロットを指定するようにしてください。ホストとスロットの指定方法の詳細は、mpirun(1) のマニュアルページを参照してください。クラスタの一部でないスタンドアロンのホストでこのコマンドを実行することもできますが、その結果はあまり教育的でない場合があります。

この例のホストファイルは clusterhosts と呼ばれ、次の内容が含まれています。

```
hostA slots=1
hostB slots=1
```

リモートシェル (ssh または rsh) を使用して、ホストへのログインなしに各ホストに接続する権限が必要です。デフォルトでは、mpirun は ssh を使用します。

```
% mpirun -np 25 --hostfile clusterhosts ring_c
Process 0 sending 10 to 1, tag 201 (25 processes in ring)
Process 0 sent to 1
Process 0 decremented value: 9
Process 0 decremented value: 8
```

```
Process 0 decremented value: 7

Process 0 decremented value: 6
Process 0 decremented value: 5
Process 0 decremented value: 4
Process 0 decremented value: 3
Process 0 decremented value: 2
Process 0 decremented value: 1

Process 0 decremented value: 0
Process 0 exiting
Process 1 exiting
Process 2 exiting
.
.
.
Process 24 exiting
```

このコマンドを実行して、同じような出力が得られた場合は、次のセクションに示すようにアプリケーション例に関するデータを収集する準備が整っています。

ssh を使用する mpirun で問題が発生した場合は、mpirun コマンドでオプション --mca plm_rsh_agent rsh を使用して、rsh コマンドによる接続を試してください。

```
% mpirun -np 25 --hostfile clusterhosts --mca plm_rsh_agent rsh -- ring_c
```

ring_c 例に関するデータの収集

1. example のバイナリとソースコードが置かれているディレクトリに変更します。
2. 次のコマンドを実行します。

   ```
   % collect -M OMPT mpirun -np 25 --hostfile clusterhosts  -- ring_c
   ```

   このコマンドの実行には少し時間がかかることがあり、その出力は mpirun コマンドによるテスト実行と同じになります。

   -M OMPT オプションは、MPI バージョンが Oracle Message Passing Toolkit であることを示します。サポートされている MPI バージョンの詳細は、collect(1) のマニュアルページを参照してください。

   -np 25 オプションはクラスタ上に 25 個のプロセスがあることを示し、--hostfile clusterhosts はノード名と各ノードで使用されるスロットの数が clusterhosts というファイル内に指定されていることを示します。

   このコマンドは 2 つのホストで 25 個のプロセスを使用することを指定し、各ホストで 1 つのスロットを指定します。システムに適しているいくつかのプロセスとスロットを指定するようにしてください。
3. 新しく作成した test.1.er ディレクトリの内容を一覧表示し、ファイルの日付が最新の実行を反映していることを確認します。これは、コマンドが正常に実行され、ring_c でパフォーマンスアナライザを実行する準備が整っていることを意味

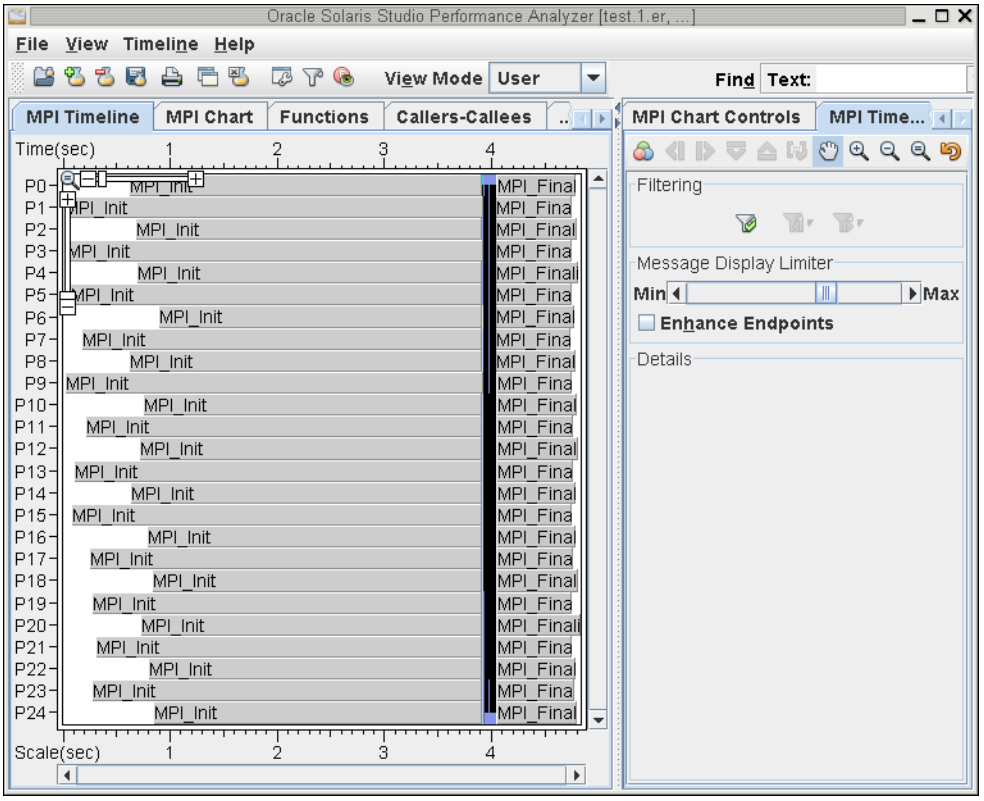
します。test.1.er 内の整数は collect コマンドを実行するごとに増分されるため、このチュートリアルの残りの部分ではこの名前を総称的に test.*.er と呼びます。

実験の開始

1. ring_c.c ソースファイル、ring_c 実行可能ファイル、および test.*.er ディレクトリが含まれているディレクトリに変更します。
2. コマンド行からパフォーマンスアナライザを開始します。

   ```
   %  analyzer
   ```

   パフォーマンスアナライザが、実験を検索して開くためのファイルブラウザを開きます。そうでない場合は、「ファイル」>「実験を開く」を選択します。
3. 作成したばかりの test.*.er 実験を検索し、それを開きます。「パフォーマンスアナライザ」ウィンドウの表示が次のようになります。



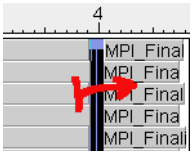
実験は「MPI タイムライン」タブで開きます。「MPI チャート」タブはその隣にあります。右側のパネルに「MPI チャートコントロール」および「MPI タイムラインコントロール」タブがあります。

「MPI タイムライン」には、コレクタを通じてプログラムが実行されたときのデータのビューが時間とともに表示されます。水平軸は経過時間を示しています。最下部にある水平軸は、画面の左端を起点とした相対時間を示しています。最上部にある水平軸は、起点がデータの開始となる絶対時間を示しています。垂直軸は MPI プロセスランクを示しています。MPI プロセスごとに水平に見ることで、プロセスが経過時間に応じて実行している内容を確認できます。

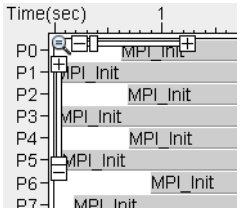
タイムラインの初期ビューは、プログラム実行のタイムスケールはなにかという疑問を解決します。このスクリーンショットでは、タイムスケールがおよそ 5 秒であることを示しています。ただし、実際の実行時間はアプリケーションプログラムの通常状態である 3.90 から 4.05 秒までの範囲です。`collect` ツールは、`MPI_Init` と `MPI_Finalize` を使用してデータ収集を設定および終了します。

「MPI タイムライン」のナビゲート

1. 「MPI タイムライン」タブがまだ選択されていない場合はそれをクリックします。
2. データを拡大表示するには、下の図の矢印で示すように、任意のプロセス行をクリックして左から右にドラッグします。マウスボタンを離すと、そのボックスの内側の領域が自動的に広がって、拡大されたビューを表示します。



クリックしてドラッグの代替方法として、タイムラインの左上にあるズームスライダコントロールを使用することもできます。



タイムスケールを変更するには、水平スライダを使用します。拡大すると、すべてのプロセスを引き続き表示しながら、時間の断片が徐々に小さくなるのがわかります。

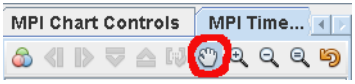
MPI プロセスを拡大表示するには、垂直スライダを使用します。

前のズームレベルに戻すには、下に示す「MPI タイムラインコントロール」タブの「前のズームに戻す」ボタンをクリックします。

最初のズームに戻すには、「前のズームに戻す」ボタンをもう一度クリックします。

3. データ全体にわたってパンするには、タイムラインの最下部と右側にあるスクロールバーをスライドさせます。

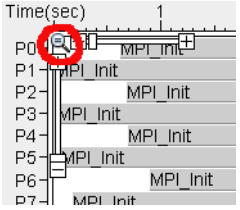
   別の方法として、「MPI タイムラインコントロール」タブにある手のアイコンをクリックして、ズームするポインタとパンするポインタを切り替えることもできます。



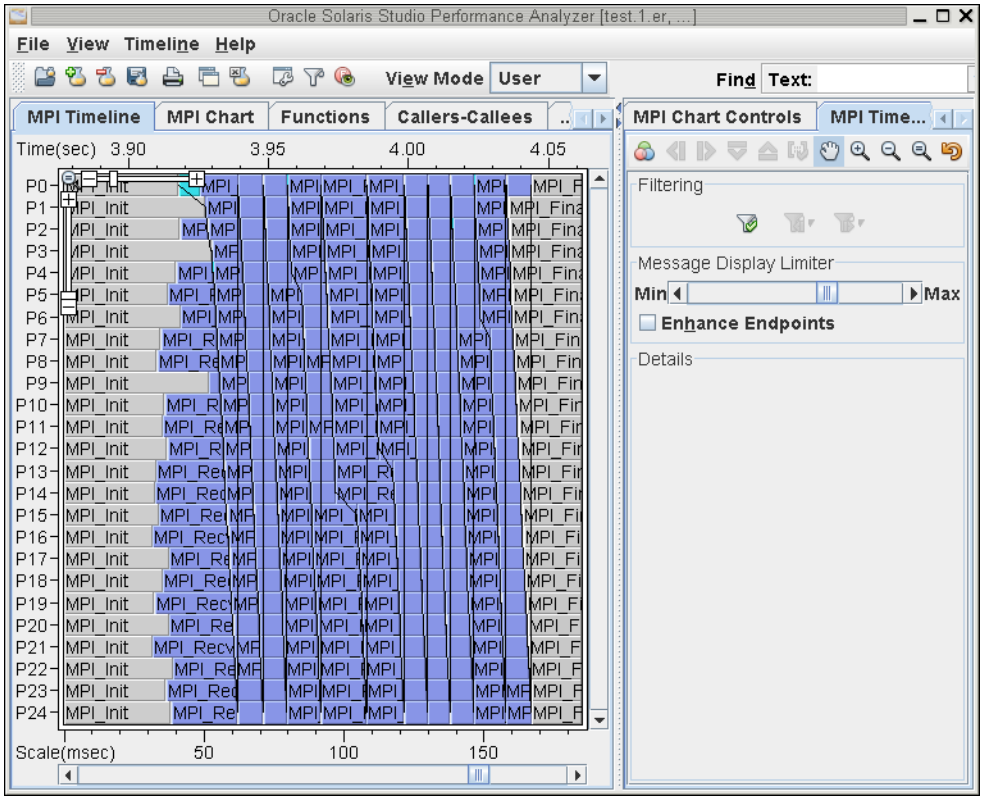
   ポインタが手の場合は、MPI タイムライン全体をドラッグして水平方向にパンできます。

メッセージの詳細の表示

1. ズームスライダの左上にある「ズームリセット」ボタンをクリックして、ビューを元の設定、最大の設定、縮小の設定にリセットします。



2. ここに表示されているものと同じように、マウスを使って領域を水平方向にドラッグすることで、アクティビティー領域を拡大表示します。



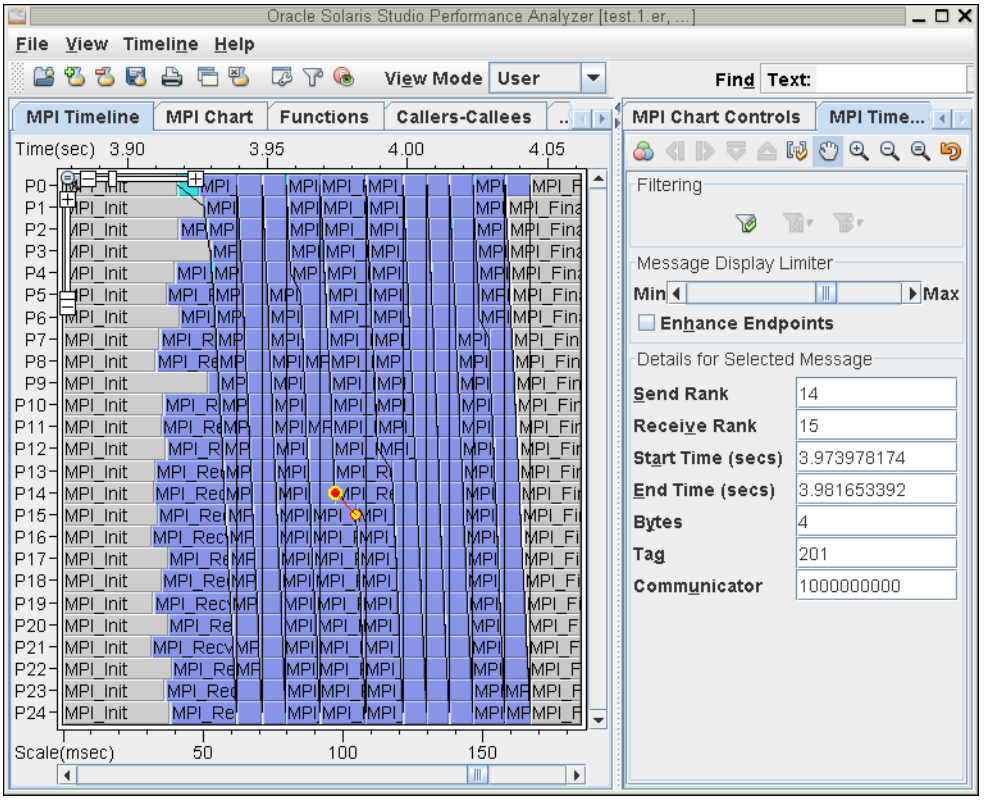
拡大表示されたタイムラインでは、プログラム実行の通常状態部分が 3.93 秒から 4.03 秒までであるように表示されることがわかります。

また、MPI 関数が色分けされていることもわかります。イベント間に描かれている黒い線は、MPI プロセスによってやりとりされた 2 点間のメッセージを表しています。

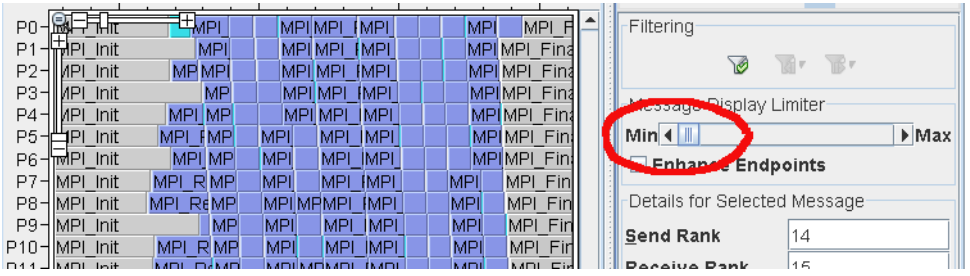
このタイムラインのビューを使用すれば、パターンが繰り返される前の 1 回の反復にかかる時間はどれくらいかという疑問を解決できます。答えはおおよそ 10 ミリ秒です。最下部の相対時間のスケールを見ると、ループがどれくらいの頻度で繰り返されるようであるかがわかります。

3. 黒いメッセージ線のいずれかをクリックします。

   その線が赤色に変わり、そのメッセージに関する詳細が右側のパネルの「MPI タイムラインコントロール」タブに表示されます。



4. 「MPI タイムラインコントロール」タブで、「メッセージ表示数上限」スライダを見つけ、図に示すように、それをクリックして「最小」までドラッグします。



「メッセージ表示数上限」スライダは、画面に表示されるメッセージ線の数を制御します。「最小」では、関数のみが「MPI タイムライン」タブに表示されます。

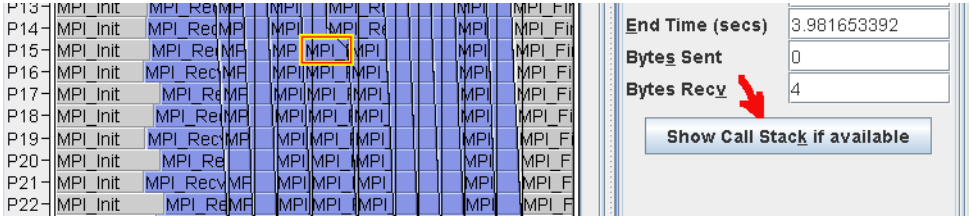
この簡単な例では、すべてのメッセージが表示されることがあります。ただし、複雑なアプリケーションのすべてのメッセージを表示すると、ツールに大きな負荷がかかり、画面が乱れて使用できなくなる可能性があります。タイムラインに表示されるメッセージの数を減らすには、下限値を選択します。100% 未満のメッセージが表示される場合、使用されたメッセージはメッセージの送信および受信関数で使用された合計時間の点からもっともコストのかかるメッセージです。

5. 「メッセージ表示数上限」スライダを「最大」に戻します。

関数の詳細とアプリケーションソースコードの表示

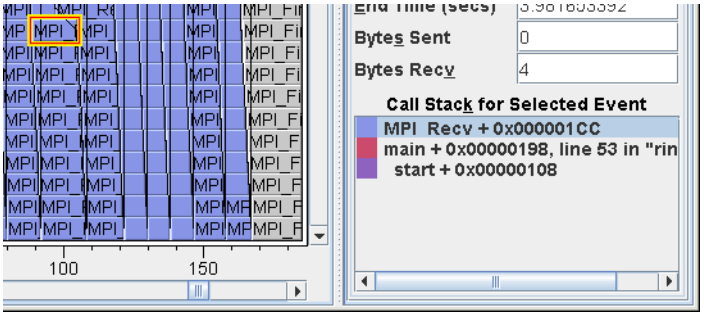
1. 「MPI タイムライン」タブで `MPI_Recv` 関数のいずれかのイベントをクリックします。

   関数が黄色で強調表示され、その関数の詳細が右側の「MPI タイムラインコントロール」タブに表示されます。



2. 「MPI タイムラインコントロール」タブで、呼び出しスタックを表示します。「使用可能な場合に呼び出しスタックを表示」というラベルの付いたボタンが表示されたら、そのボタンをクリックします。

   しばらくして、強調表示された状態の呼び出しスタックが「MPI タイムラインコントロール」タブに表示されます。

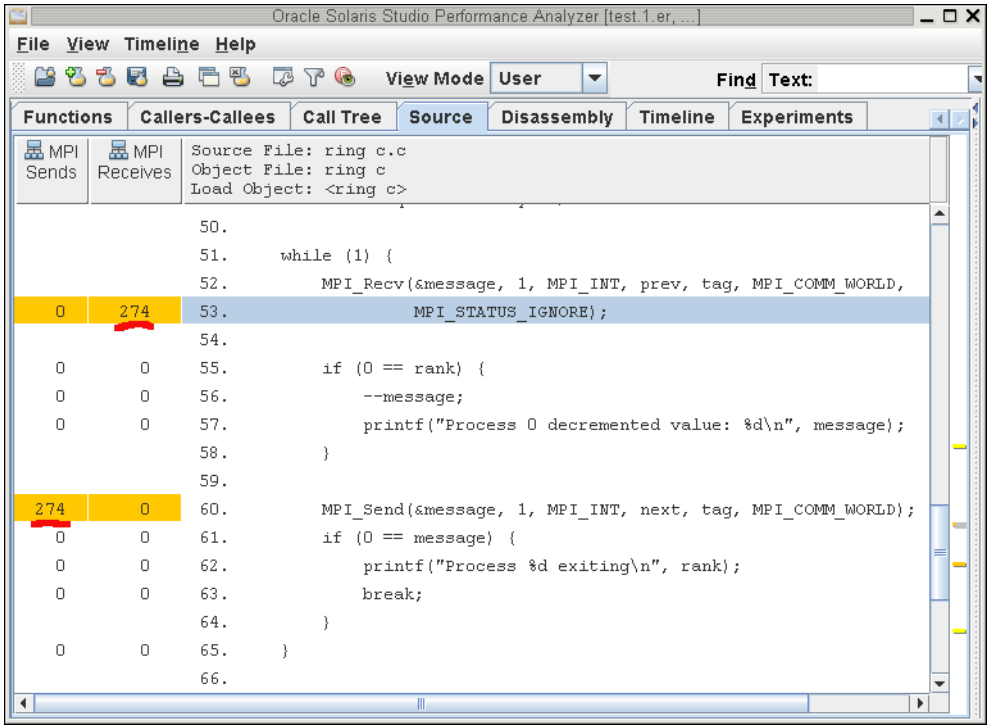


3. 「選択されたイベントの呼び出しスタック」が「MPI タイムラインコントロール」タブに表示されたら、「`main + 0x00000198, line 53 in "ring_c.c`」をクリックします。
4. パフォーマンスアナライザのメインパネルで「ソース」タブをクリックします。

   「`Object file (unknown) not readable`」のようなメッセージが表示された場合は、スタックフレーム `main + 0x00000198, line 53 in "ring_c.c` を選択していることを確認します。

注 – ソースが表示されるのは、そのソースが、プログラムがコレクタを通じて実行されたときにそれが入っていたのと同じ場所にある場合、あるいはそれが `.er.rc` または「表示」>「データ表示方法の設定」>「パスを検索」で設定された `$expts` パスにある場合に限られます。ソースは、`-g` を使用してコンパイルする必要もあります。ソースコードが表示されない場合は、`ring_c` バイナリとソースコードが含まれているディレクトリからアナライザを開始していない可能性があります。その場合は、パフォーマンスアナライザを終了し、`ring_c` が含まれているディレクトリに移動したあとで再起動します。

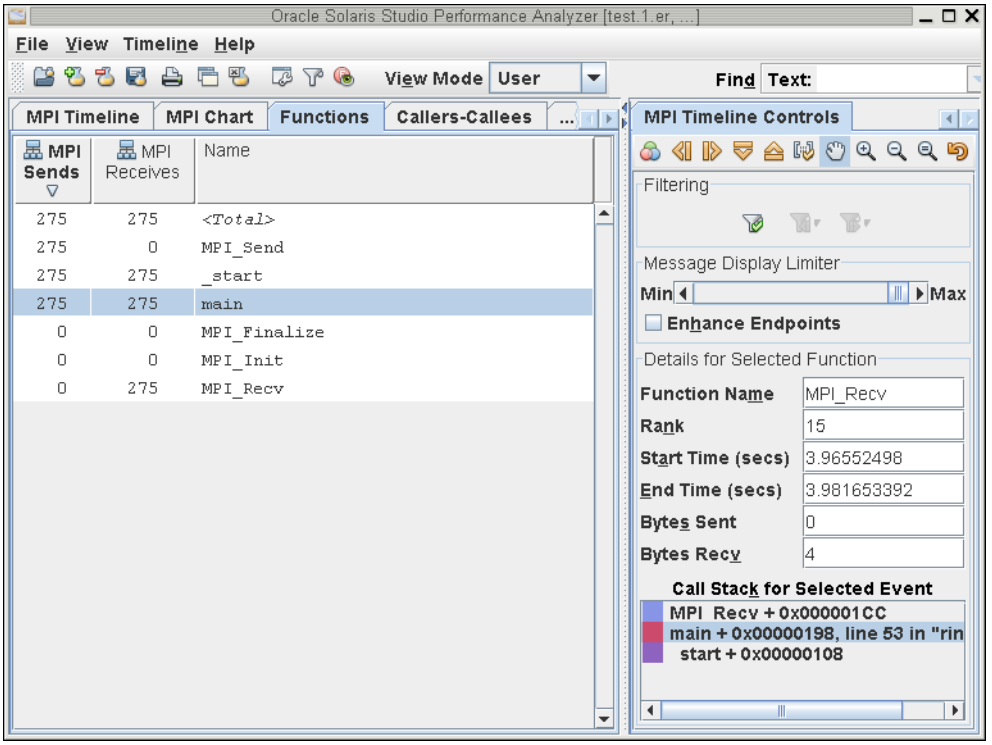
ソースが表示される場合、それは `main()` が `MPI_Recv()` 関数を呼び出す場所を示しているはずです。下に示すように、`MPI_Recv()` はソースの 53 行目から呼び出されます。高い値を持つメトリックは強調表示されます。274 の受信が 53 行目に関連付けられ、274 の送信が 60 行目の `MPI_Send()` に関連付けられています。



ヒント– このスクリーンショットでは、表示されるメトリック数が少なくされています。表示するメトリックをさらに選択するには、「表示」>「データ表示方法の設定」>「メトリック」を選択します。

5. パフォーマンスアナライザのメインパネルで「関数」タブをクリックします。

   「関数」タブでは、テーブルの左側の列に同じ MPI 送信および MPI 受信メトリックが表示されます。このテーブルをソートするには、列ヘッダー内でクリックします。



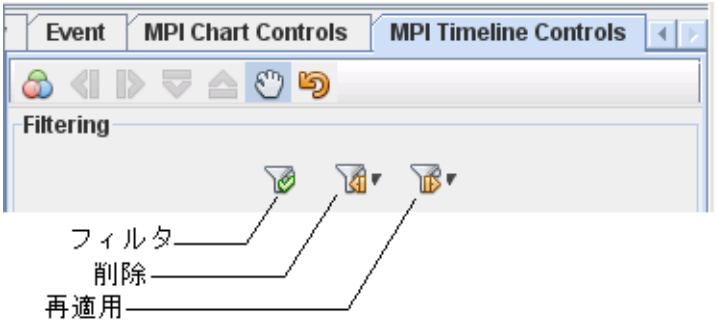
ヒント – このスクリーンショットでは、「関数」タブに表示されるメトリック数が少なくされています。表示するメトリックを選択するには、「表示」>「データ表示方法の設定」>「メトリック」を選択します。

6. 「MPI タイムライン」タブをクリックして MPI タイムラインに戻ります。

   通常の「タイムライン」タブは MPI プログラムに適用されないため、それをクリックしないでください。

MPI タブでのデータのフィルタリング

フィルタリング機能を使用すると、収集したメッセージデータのさまざまなビューを選択できます。「MPI チャートコントロール」タブまたは「MPI タイムラインコントロール」タブのどちらかで、フィルタリングコントロールを使用してフィルタを削除および再適用できます。



最初のコントロールでは、現在表示されていないすべてのものを削除することによってデータをフィルタリングします。

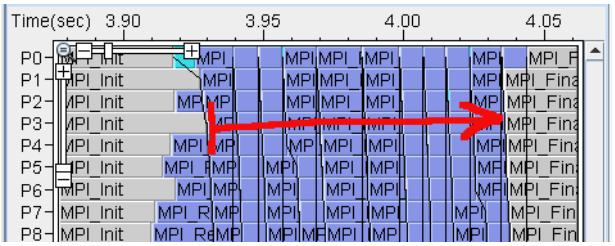
2 番目のコントロールは、フィルタを削除するための関連付けられたドロップダウンリストを提供する「削除」ボタンです。このボタンをクリックすると、最後に適用されたフィルタが削除されます。下向き矢印をクリックすると、適用されたフィルタのリストが、それらが適用された順序で表示され、リストの最上部には最新のものが示されます。このリストでフィルタを選択すると、選択したフィルタと、リスト上でそれよりも上にあるすべてのフィルタが削除されます。

3 番目のコントロールは「再適用」ボタンであり、それにはフィルタを再適用するための関連付けられたドロップダウンリストもあります。このボタンをクリックすると、削除した最後のフィルタが再適用されます。下矢印をクリックすると、削除されているすべてのフィルタのリストが、それらが削除された順序で表示されます。このリストでフィルタを選択すると、選択したフィルタと、リスト上でそれよりも上にあるすべてのフィルタが再適用されます。

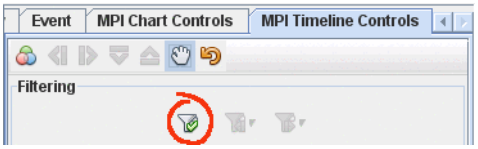
フィルタを削除および再適用するには、Web ブラウザで戻ったり進んだりするのに似た矢印を使用します。1 回のクリックで複数のフィルタを削除および再適用するには、フィルタボタンの横にある下矢印を使用します。

次の手順では、`MPI_Init` および `MPI_Finalize` 関数をフィルタで除去することにより、フィルタを使用してプログラムの通常状態部分に焦点を当てる方法について説明します。

1. タイムラインで、マウスを水平方向にドラッグして、`MPI_Init` と `MPI_Finalize` が表示されなくなるような領域を選択します。下の例では、t=3.93 から 4.03 までドラッグします。

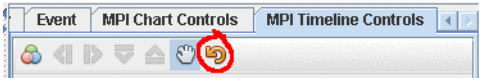


2. 「MPI タイムラインコントロール」タブで「フィルタ」ボタンをクリックします。

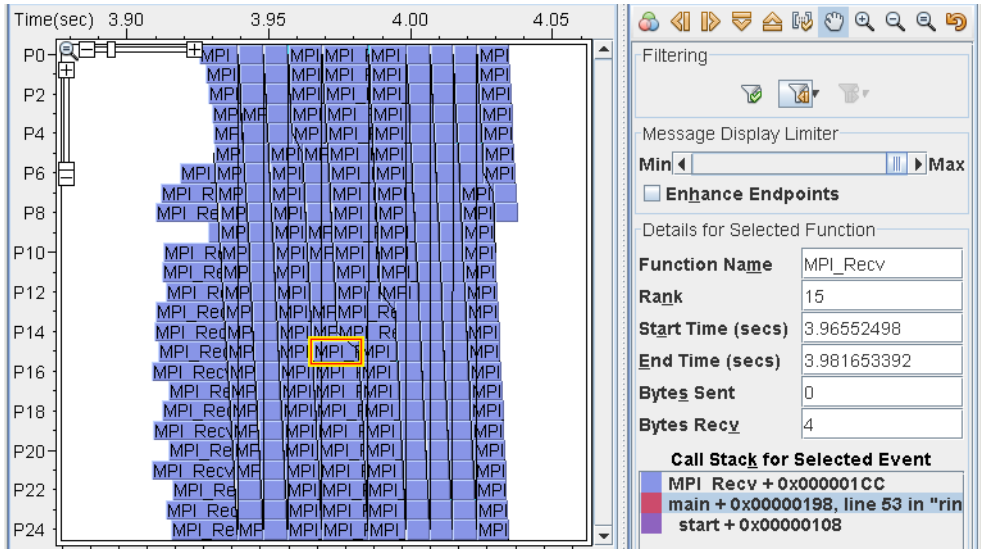


フィルタリングは縮小表示したり、チャートを表示したりすることでビューを変更するまではっきりしないため、ラインに何も起こっていないように見えることがあります。

3. 「前のズームに戻す」ボタンをクリックして、前のズームに戻します。



画面には、`MPI_Init` および `MPI_Finalize` 関数の代わりに空白の領域が表示されます。空白の領域は、その領域に対して収集された MPI データがないか、またはデータがフィルタで除去されていることを示しています。

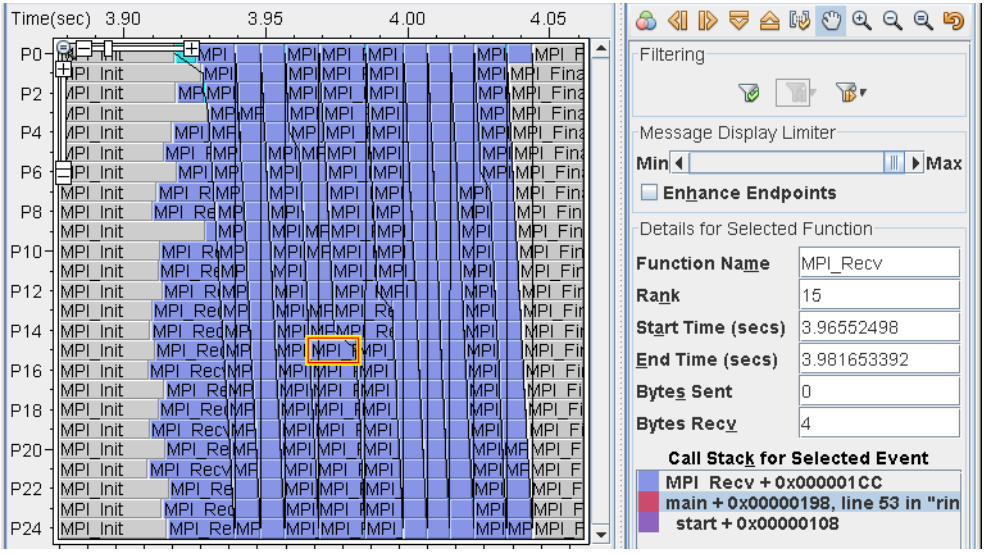


フィルタスタックの使用

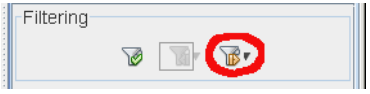
1. 「削除」ドロップダウン矢印をクリックして、適用されたフィルタのリストを表示します。

このリストを使用すると、削除するフィルタを選択できます。それはスタックのように機能します。「フィルタが適用されていません」を選択した場合は、その上にあるすべてのものが削除され、適用されるフィルタがないことを意味します。

2. 「削除」リストから「`No filters applied`」を選択します。タイムラインは次のように表示されます。



3. 「再適用」を使用して前のフィルタを復元します。

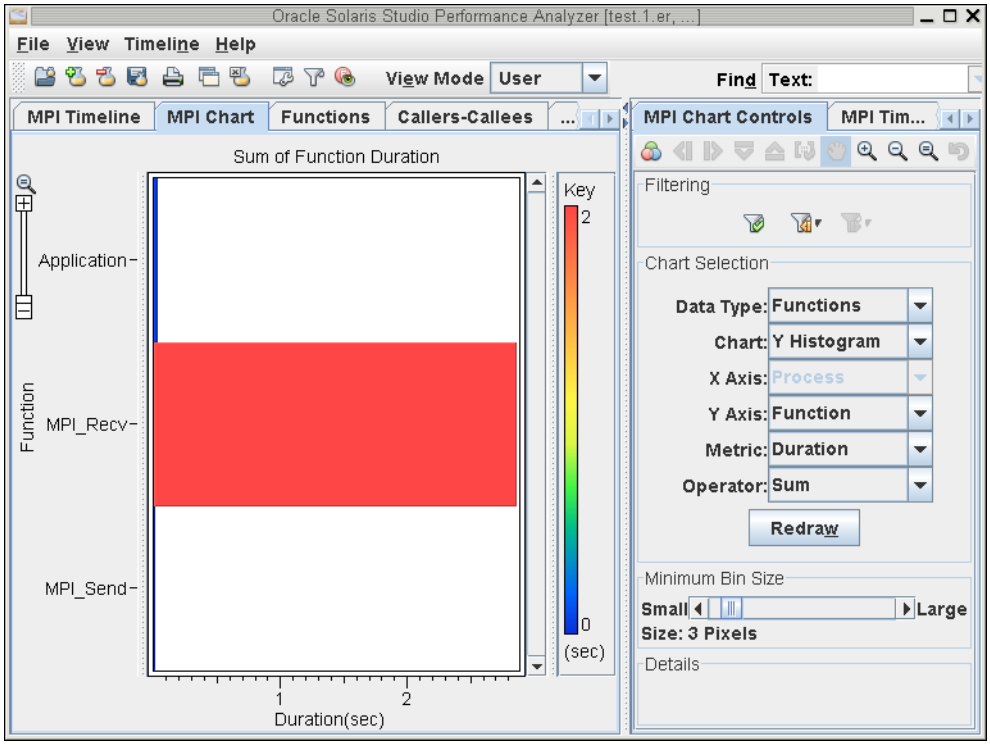


4. 次のセクションに進む前に、`MPI_Init` および `MPI_Finalize` がフィルタで除去され、以前のように空白の領域が表示されていることを確認します。

「MPI チャート」タブの使用

`MPI_Init` および `MPI_Finalize` がすでにフィルタで除去されている状態で、「MPI チャート」の機能を調べます。初期チャートには、どの関数にもっとも長い時間がかかったかが示されます。

1. 「MPI チャート」タブをクリックして、次のようなチャートを表示します。



「MPI チャート」タブが開き、関数がすべてのプロセスで実行されたときにかかった実行時間の合計を示すチャートが表示されます。チャートの右側にある垂直方向の虹色のスケールは、色と値のマッピングを示しています。この例では、`MPI_Send` および `Application` 関数にかかった時間がほとんどないのに対し、`MPI_Recv` の累積時間は 3 秒近くにもなりました。

2. 「`MPI_Recv`」関数のバーをクリックします。

   バーの正確な値が「MPI チャートコントロール」タブに表示されます。

このアプリケーションでは、どのプロセスもトークンがほかのすべてのプロセスに渡されるまで待機します。結果として、`MPI_Recv` にはかなりの時間が費やされ、MPI 関数間の時間を表す状態である `Application` にはほとんど時間が費やされません。あとで確認するとおり、あるランクへのメッセージ配信の遅れはほかのすべてのランクの進行を妨げます。

「MPI チャートコントロール」の使用

このセクションでは、さまざまな方法で「MPI チャートコントロール」タブを使用してデータを図形化する方法について説明します。解析中のプログラムによって

は、一部のチャートオプションがほかのものよりも役に立ちます。すべての MPI チャートコントロールについては、付録 A 「MPI チャートコントロールの設定」を参照してください。

メッセージの送信場所を示すチャートを作成する

1. 「MPI チャートコントロール」タブで次の選択を行なって、メッセージを調べるチャートを作成します。

   データ型: メッセージ

   チャート: 2D チャート

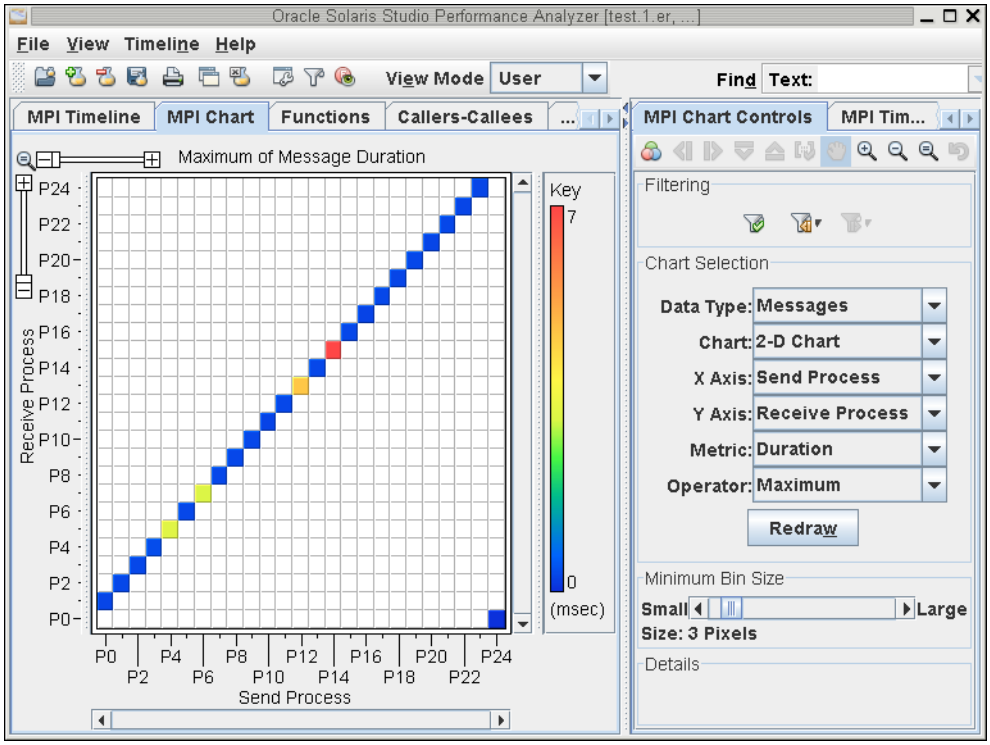
   X 軸: 送信プロセス

   Y 軸: 受信プロセス

   メトリック: 実行時間

   演算子: 最大

2. 「変更を表示更新する」をクリックすると、次のようなチャートが表示されます。



このチャートは、プロセス 0 がプロセス 1 にのみ送信することを示しています。プロセス 1 はプロセス 2 にのみ送信し、以下同様になります。各ボックスの色は選択したメトリック (「経過時間」) と演算子 (「最大」) によって設定されます。このグラフの「データ型」は「メッセージ」であるため、これはメッセージの実行時間の合計になるか、または時間寸法におけるメッセージ線の長さになります。

チャートの色はランク間のメッセージの最大実行時間を示しています。この例では、到達にかかる時間がもっとも長いメッセージが P14 から P15 に送信されました。

メッセージの受信待ち時間がもっとも長いランクを示すチャートを作成する

1. 「MPI チャートコントロール」タブで次の選択を行います。

   データ型: メッセージ

   チャート: Y ヒストグラム

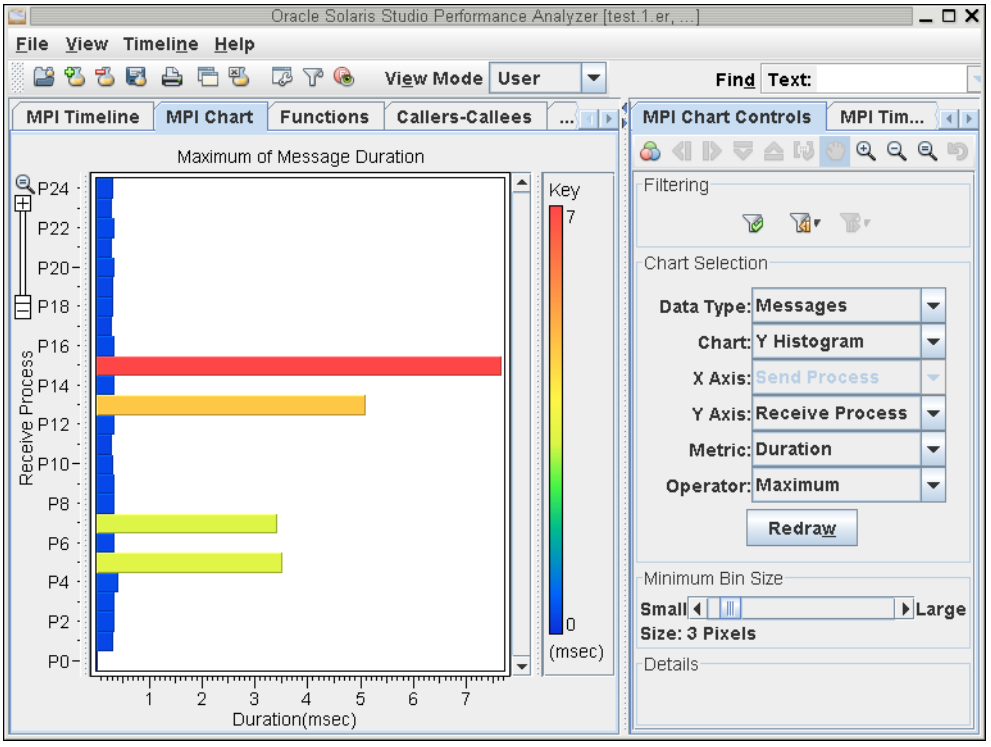
   X 軸: *N/A*

Y 軸: 受信プロセス

メトリック: 実行時間

演算子: 最大

2. 「変更を表示更新する」をクリックして、新しいチャートを表示します



この例では、チャートは P15 ランクがメッセージを受信するのにもっとも長く待機したことを示しています。

3. これらの実行時間の長いメッセージが発生した時期についてのヒストグラムを表示するには、コントロールに対して次の選択を行います。

データ型: メッセージ

チャート: X ヒストグラム

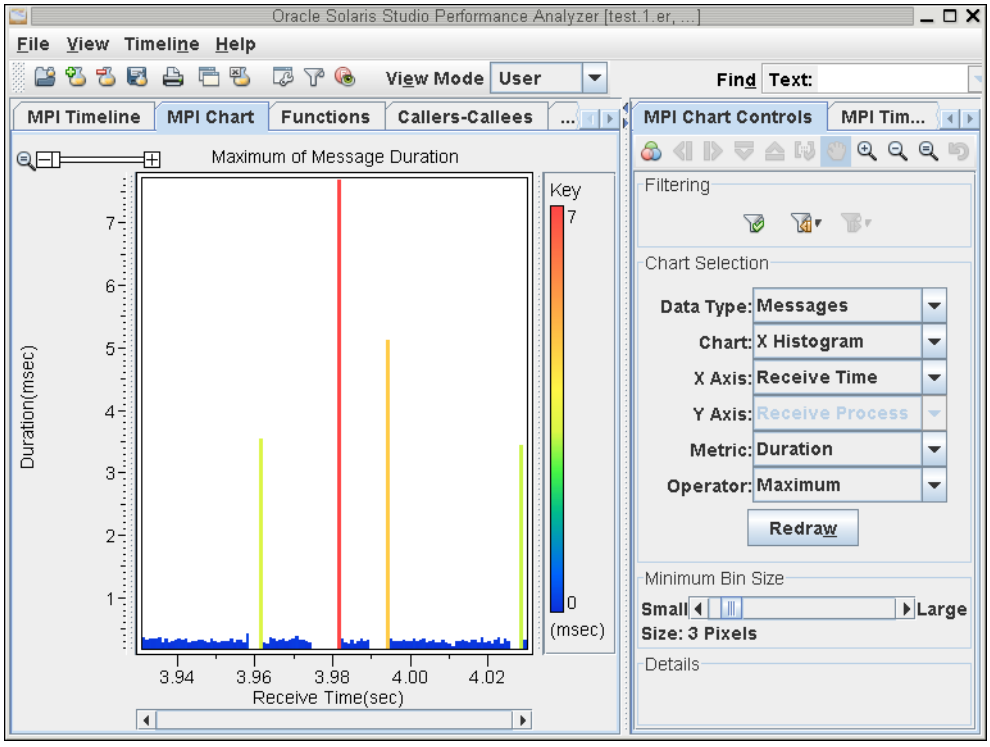
X 軸: 受信時間

Y 軸: *N/A*

メトリック: 実行時間

演算子: 最大

4. 「変更を表示更新する」をクリックして次のようなチャートを表示します。



この例では、もっとも遅いメッセージが 3.981 秒で受信されました。

遅いメッセージが MPI 関数に費やされる時間に与える影響を調べる

実行時間の長いメッセージの影響を調べるには、関数の実行時間と時間との比較を示すグラフを作成します。

1. 「MPI チャートコントロール」タブで次の選択を行なって、メッセージを調べるチャートを作成します。

   データ型: 関数

   チャート: 2D チャート

   X 軸: 終了時間

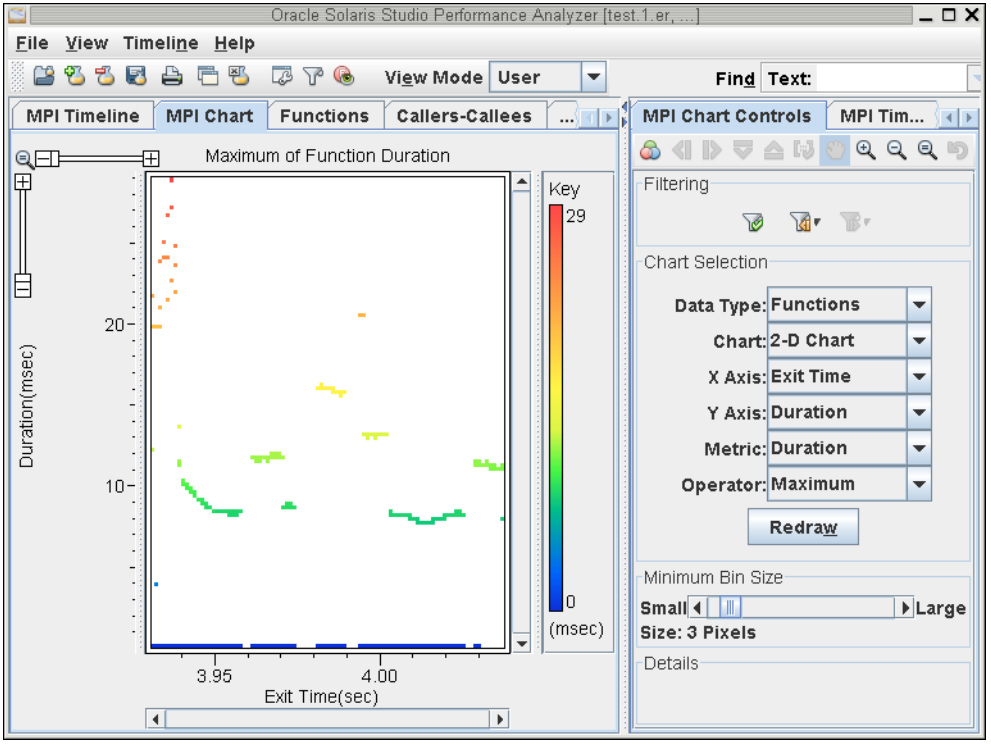
Y 軸: 実行時間

メトリック: 実行時間

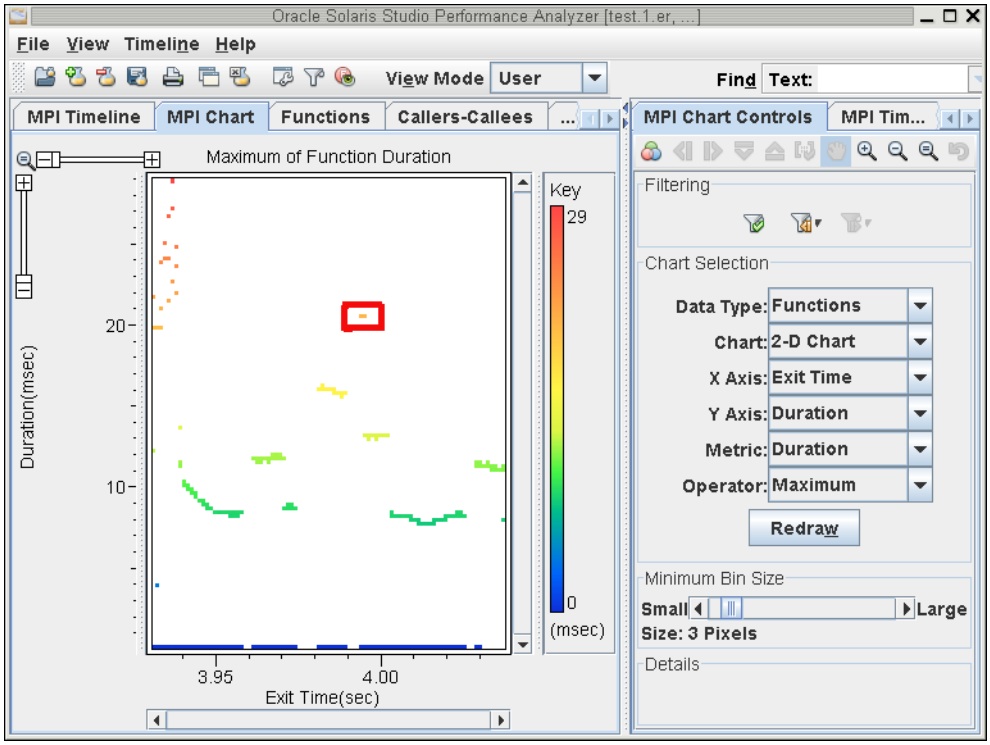
演算子: 最大

2. 「変更を表示更新する」をクリックします。

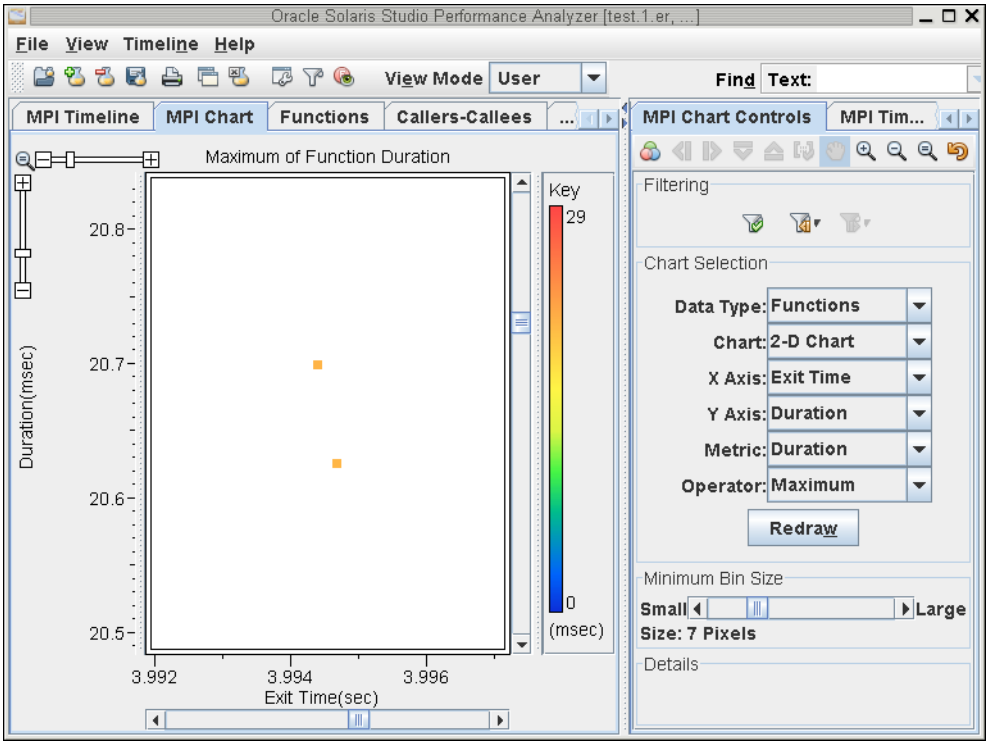
この例では、グラフは実行時間の長い関数が含まれるいくつかの時間領域を示しています。t=3.99 のところで関数の実行時間が 20 秒を超えていることに注意してください。この時間は、前のチャートに表示されている、所要時間の長いメッセージに一致しています。



3. これらの実行時間の長い関数の周囲をボックスでドラッグしてズームすることで、それらを特定します。

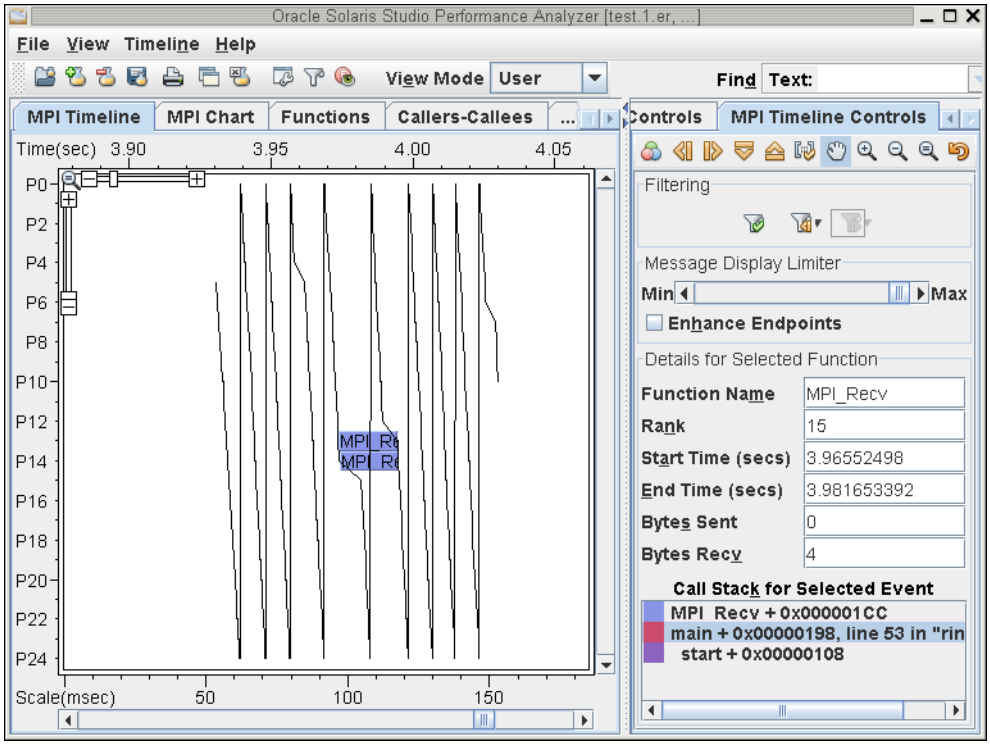


4. 「フィルタ」ボタンをクリックして、これらの関数呼び出しのみを調べます。



5. 「MPI タイムライン」タブをクリックします。

   「MPI タイムライン」で実行時間の長い関数を識別できます。それらは配信時間の遅いメッセージの結果です。



6. 「削除」および「再適用」ボタンをクリックして、実行時間の長い関数の周囲のコンテキストをトグルします。

「MPI チャートコントロール」の詳細は、付録 A 「MPI チャートコントロールの設定」を参照してください

結論

Oracle Solaris Studio パフォーマンスアナライザを使用すると、複雑なマルチスレッドアプリケーションでパフォーマンスを監視し、問題のある場所を特定できます。このチュートリアルに記載されている簡単な例は、MPI 関数と MPI メッセージの関係を調べるための基本事項を示しています。MPI タイムライン、MPI チャート、およびズームとフィルタリングの機能を使用すれば、パフォーマンスデータを収集して処理し、プログラム、関数、ソース行、および命令の各レベルでメトリックを表示して、潜在的なパフォーマンスの問題が配備上の問題になる前にそれらを特定できます。

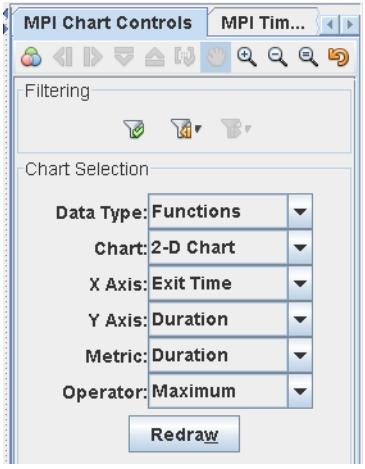
◆ ◆ ◆ 付 録 A

MPI チャートコントロールの設定

この付録では、Oracle Solaris Studio パフォーマンスアナライザの「MPI チャートコントロール」タブのすべての設定について説明します。

チャート属性

このセクションでは、MPI 実験データのチャートを作成するために「MPI チャートコントロール」タブで設定できる属性について説明します。次のスクリーンショットに、「MPI チャートコントロール」タブを示します。



- データ型 – チャートに表示されるデータの型を制御するもので、次のいずれかの値に設定できます。

- 関数 - プログラムによって使用された MPI 関数についてのデータを表示します。「関数」チャートに設定できる属性については、41 ページの「「関数」チャートの属性」を参照してください。
- メッセージ - プロセスランク間で送信された MPI メッセージについてのデータを表示します。「メッセージ」チャートに設定できる属性については、41 ページの「「メッセージ」チャートの属性」を参照してください。

- チャート – 作成されるチャートの種類を制御するもので、次のいずれかの値に設定できます。
  - Y ヒストグラム - データが水平軸上のもう 1 つのメトリックの関数として垂直軸上に表示される 1 次元チャート。Y 軸上に表示するデータの型と X 軸のメトリックを選択する必要があります。
  - X ヒストグラム - データが垂直軸上の時間の関数として水平軸上に表示される 1 次元チャート。X 軸上に表示するデータの型と Y 軸のメトリックを選択する必要があります。
  - 2 次元チャート - データが X 軸と Y 軸の両方にプロットされ、2D マトリックスまたは散布図を形成する 2 次元チャート。X および Y 軸上に表示するものとメトリックを指定する必要があります。
- X 軸 - 「X ヒストグラム」または「2 次元チャート」用に、水平軸上に表示するデータの型を選択します。使用できる型は、「データ型」リストから「関数」または「メッセージ」のどちらを選択したかで異なります。使用できる値は、41 ページの「「関数」チャートの属性」および 41 ページの「「メッセージ」チャートの属性」に記載されています。
- Y 軸 - 垂直軸上に表示するデータの型を選択します (「Y ヒストグラム」または「2 次元チャート」用)。使用できる型は、「データ型」リストから「関数」または「メッセージ」のどちらを選択したかで異なります。使用できる値は、41 ページの「「関数」チャートの属性」および 41 ページの「「メッセージ」チャートの属性」に記載されています。
- メトリック - X または Y の関数として表示されるデータを選択します。メトリック値は、チャート内で色分けして示されます。使用できる型は、「データ型」リストから「関数」または「メッセージ」のどちらを選択したかで異なります。使用できる値は、41 ページの「「関数」チャートの属性」および 41 ページの「「メッセージ」チャートの属性」に記載されています。
- 演算子 – チャート内のメトリックの結合に使用される方法を選択します。使用できる値は、42 ページの「「演算子」の設定」に記載されています。

「関数」チャートの属性

次は、「関数」のデータを表示するときに設定できるチャート属性です。これらの属性は、「X 軸」、「Y 軸」、および「メトリック」に対して選択できます。

- 時間 (範囲) - 関数の開始から終了までの時間の範囲
- エントリ時間 - 関数が呼び出される時間
- 終了時間 - 関数が呼び出し元に戻る時間
- 経過時間 - 関数の開始と関数の終了の時間差
- プロセス - 番号順による MPI のグローバルなランク。各関数呼び出しには、一意のプロセスランクが関連付けられています。
- 関数 - 呼び出された MPI 関数。
- 送信バイト - MPI 関数呼び出しで送信されたバイト数
- 受信バイト - MPI 関数呼び出しで受信されたバイト数
- 1 (「メトリック」に対してのみ) - メトリックとして 1 を指定すると、値が常に 1 の属性が指定されます。これは、データレコードを数えたり、データの有無を示したりするために使用できます。たとえば、関数ごとに関数呼び出しの回数を数えるには、「Y 軸」:「関数」、「メトリック」:「1」、「演算子」:「合計」に設定します。関数呼び出しが行われたかどうかを検出するには、「演算子」:「最大」に設定します。

「メッセージ」チャートの属性

次は、「メッセージ」のデータを表示するときに設定できるチャート属性です。これらの属性は、「X 軸」、「Y 軸」、および「メトリック」に対して選択できます。

- 時間 (範囲) - メッセージの送信から受信までの時間の範囲
- 送信時間 - メッセージが送信された時間
- 受信時間 - メッセージが受信された時間
- 経過時間 - メッセージの送信と受信の時間差
- 送信プロセス - メッセージを送信したプロセス
- 受信プロセス - メッセージを受信したプロセス
- コミュニケータ - メッセージの送信と受信に使用されるコミュニケータ (一連のプロセス) に一意のラベルを付ける任意に定義された ID
- タグ - メッセージの識別に使用される MPI タグ
- 送信関数 - メッセージを送信した関数
- 受信関数 - メッセージを受信した関数
- バイト - メッセージに含まれるバイト数

- 1 (「メトリック」に対してのみ) - メトリックとして 1 を指定すると、値が常に 1 の属性が指定されます。これは、データレコードを数えたり、データの有無を示したりするために使用できます。たとえば、関数ごとに関数呼び出しの回数を数えるには、「Y 軸」を「送信関数」、「メトリック」を「1」、および「演算子」を「合計」に設定します。関数ごとに関数呼び出しが行われたかどうかを検出するには、「演算子」を「最大」に設定します。

「演算子」の設定

「演算子」コントロールは、複数のメトリック値がチャート内でどのように結合されるかを決めるもので、次のいずれかの値に設定できます。

- 合計 - 「時間」、「経過時間」、「送信バイト」、「受信バイト」、または「1」のいずれかである必要がある、選択されたメトリックの合計を計算します
- 最大 - 選択されたメトリック (「時間」、「経過時間」、「送信バイト」、「受信バイト」、または「1」のいずれか) の最大値を計算します
- 最小 - 選択されたメトリック (「時間」、「経過時間」、「送信バイト」、「受信バイト」、または「1」のいずれか) の最小値を計算します
- 平均 - 選択されたメトリック (「時間」、「経過時間」、「送信バイト」、「受信バイト」、または「1」のいずれか) の平均値を計算します
- 標準 - 「標準」演算子は、どのタイプのメトリックにも機能します。多数のメトリック値がすべて同じチャートバイナリに割り当てられている場合、「標準」演算子はそれらの値のいずれか 1 つを「適正に」選択します。たとえば、MPI メッセージの 90% はコミュニケータ `MPI_COMM_WORLD` を使用しますが、それらの 10% はユーザー定義のコミュニケータ `mycomm` を使用するとします。メトリックに「コミュニケータ」、演算子に「標準」を使用してメッセージチャートを作成した場合、そのチャートでは時間の 90% は `MPI_COMM_WORLD` が報告されますが、時間の 10% は `mycomm` が報告されます。

◆ ◆ ◆ 付 録 B

チュートリアルのサンプルコード

この付録には、チュートリアルで使用されるサンプルコードと Makefile が記載されています。

例 B–1 ring_c.c のサンプルコード

```
/*
 * Copyright (c) 2004-2006 The Trustees of Indiana University and Indiana
 *                         University Research and Technology
 *                         Corporation.  All rights reserved.
 * Copyright (c) 2006      Cisco Systems, Inc.  All rights reserved.
 *
 * Simple ring test program
 */

#include <stdio.h>
#include "mpi.h"

int main(int argc, char *argv[])
{
    int rank, size, next, prev, message, tag = 201;

    /* Start up MPI */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* Calculate the rank of the next process in the ring.  Use the
       modulus operator so that the last process "wraps around" to
       rank zero. */

    next = (rank + 1) % size;
    prev = (rank + size - 1) % size;

    /* If we are the "master" process (i.e., MPI_COMM_WORLD rank 0),
       put the number of times to go around the ring in the
       message. */

    if (0 == rank) {
        message = 10;
```

例 B–1 ring_c.c のサンプルコード　(続き)

```
        printf("Process 0 sending %d to %d, tag %d (%d processes in ring)\n",
               message, next, tag, size);
        MPI_Send(&message, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
        printf("Process 0 sent to %d\n", next);
    }

    /* Pass the message around the ring.  The exit mechanism works as
       follows: the message (a positive integer) is passed around the
       ring.  Each time it passes rank 0, it is decremented.  When
       each processes receives a message containing a 0 value, it
       passes the message on to the next process and then quits.  By
       passing the 0 message first, every process gets the 0 message
       and can quit normally. */

    while (1) {
        MPI_Recv(&message, 1, MPI_INT, prev, tag, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);

        if (0 == rank) {
            --message;
            printf("Process 0 decremented value: %d\n", message);
        }

        MPI_Send(&message, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
        if (0 == message) {
            printf("Process %d exiting\n", rank);
            break;
        }
    }

    /* The last process does one extra send to process 0, which needs
       to be received before the program can exit */

    if (0 == rank) {
        MPI_Recv(&message, 1, MPI_INT, prev, tag, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
    }

    /* All done */

    MPI_Finalize();
    return 0;
}
```

次のコードは、サンプルコードに付属し、`ring_c.c` の構築に使用できる `Makefile` の内容を示しています。

例 B–2 ring プログラムを構築するための Makefile

```
#
# Copyright (c) 2004-2005 The Trustees of Indiana University and Indiana
#                         University Research and Technology
#                         Corporation.  All rights reserved.
```

例 B–2 ringプログラムを構築するためのMakefile (続き)

```
# Copyright (c) 2004-2005 The University of Tennessee and The University
#                         of Tennessee Research Foundation.  All rights
#                         reserved.
# Copyright (c) 2004-2005 High Performance Computing Center Stuttgart,
#                         University of Stuttgart.  All rights reserved.
# Copyright (c) 2004-2005 The Regents of the University of California.
#                         All rights reserved.
# Copyright (c) 2006-2007 Sun Microsystems, Inc.  All rights reserved.
# $COPYRIGHT$
#
# Additional copyrights may follow
#
# $HEADER$
#

# Use the Open MPI-provided wrapper compilers.  Note that gmake
# requires the CXX macro, while other versions of make (such as Sun's
# make) require the CCC macro.

CC = mpicc
CXX = mpic++
CCC = mpic++
F77 = mpif77
FC = mpif90

# Using -g is not necessary, but it is helpful for example programs,
# especially if users want to examine them with debuggers.  Note that
# gmake requires the CXXFLAGS macro, while other versions of make
# (such as Sun's make) require the CCFLAGS macro.

CFLAGS = -g
CXXFLAGS = -g
CCFLAGS = -g
F77FLAGS = -g
FCFLAGS = -g

# Example programs to build

EXAMPLES = hello_c hello_cxx hello_f77 hello_f90 \
           ring_c ring_cxx ring_f77 ring_f90 connectivity_c

# Default target.  Always build the C example.  Only build the others
# if Open MPI was build with the relevant language bindings.

all: hello_c ring_c connectivity_c
    @ if test "`ompi_info --parsable | grep bindings:cxx:yes`" != ""; then \
        $(MAKE) hello_cxx ring_cxx; \
    fi
    @ if test "`ompi_info --parsable | grep bindings:f77:yes`" != ""; then \
        $(MAKE) hello_f77 ring_f77; \
    fi
    @ if test "`ompi_info --parsable | grep bindings:f90:yes`" != ""; then \
        $(MAKE) hello_f90 ring_f90; \
    fi
```

例 B–2 ring プログラムを構築するための Makefile (続き)

```
# The usual "clean" target

clean:
    rm -f $(EXAMPLES) *~ *.o

# Don't rely on default rules for the fortran examples

hello_f77: hello_f77.f
    $(F77) $(F77FLAGS) $^ -o $@
ring_f77: ring_f77.f
    $(F77) $(F77FLAGS) $^ -o $@

hello_f90: hello_f90.f90
    $(FC) $(FCFLAGS) $^ -o $@
ring_f90: ring_f90.f90
    $(FC) $(FCFLAGS) $^ -o $@
```