# Oracle® Solaris Studio 12.3 リリースの新機能

Part No: E26448
2011 年 12 月

ORACLE®

120220@25097

# 目次

# はじめに

このガイドでは、Oracle Solaris Studio 12.3 の新機能と変更された機能、既知の問題、および制限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 ファミリー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 ファミリー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。

このマニュアルでは、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。

サポートされるハードウェアプラットフォームと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 マニュアル

Oracle Solaris Studio ソフトウェアの完全なマニュアルは、次のように見つけることができます。

- 製品のマニュアルは、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には、ツールのコマンドオプションが説明されています。

# 関連するサードパーティの Web サイトリファレンス

このマニュアルには、詳細な関連情報を提供するサードパーティの URL が記載されています。

---

注 – このマニュアルで紹介するサードパーティ Web サイトが使用可能かどうかについては、Oracle は責任を負いません。このようなサイトやリソース上、またはこれらを経由して利用できるコンテンツ、広告、製品、またはその他の資料についても、Oracle は保証しておらず、法的責任を負いません。また、このようなサイトやリソースから直接あるいは経由することで利用できるコンテンツ、商品、サービスの使用または依存が直接のあるいは関連する要因となり実際に発生した、あるいは発生するとされる損害や損失についても、Oracle は一切の法的責任を負いません。

---

# 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。

- リソースは頻繁に更新されます。
- ソフトウェアの最近のリリースに関連する完全なマニュアルへのリンク
- 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`、聴覚に障害があるお客様は `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` を参照してください。

# 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。<br>ls -a を使用してすべてのファイルを表示します。<br>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**<br>password: |
| *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。<br>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 **<br>**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 Oracle Solaris Studio 12.3 リリースの紹介

このリリースの Oracle Solaris Studio は、この『新機能』ガイドで概説する多数の新機能や変更された機能を提供します。このガイドは、以前のリリースで Sun Developer Network ポータル上で公開されていたコンポーネント README ファイルを置き換えるものです。

## Oracle Solaris Studio とは

Oracle Solaris Studio は、Oracle Solaris および Linux オペレーティング環境でアプリケーションを開発するための一連のツールから構成されています。

- 共有メモリー並列化用の OpenMP 3.1 API をネイティブで実装した、C、C++、および Fortran 向けの高性能な最適化コンパイラと実行時ライブラリ (`cc`、`CC`、および `f95`)。
- スクリプティング可能でマルチスレッドに対応した、対話型の dbx コマンド行デバッガと dbxtool デバッガ GUI。
- 高度に最適化されたマルチスレッド対応の Sun Performance Library。
- シングルスレッドおよびマルチスレッドのアプリケーションをプロファイリングしてパフォーマンス上のボトルネックや非効率性を検出するパフォーマンスアナライザと、Oracle Solaris 環境で DTrace テクノロジを使用してシステムプロファイリングを行う DLight。
- マルチスレッドアプリケーションで発生する前に実行時に潜在的で検出するのが困難なデータ競合およびデッドロックの状態を識別するスレッドアナライザ
- 静的なコードエラー、動的なメモリーアクセスエラー、およびコードカバレージデータを一緒に分析することで、ほかのエラー検出ツールでは発見できないコード内の重要なエラーを見つけるための新ツールである、コードアナライザ。
- コンポーネントのコンパイラ、デバッガ、分析ツールおよびアプリケーション構築用のコード対応エディタ、ワークフロー、プロジェクト機能で使用するように調整された IDE

Oracle Solaris Studio のすべてのドキュメントへのリンクについては、Oracle 技術ネットワークポータル (http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation) を参照してください。

# この新機能ガイドについて

このガイドは、コンパイラ、ライブラリ、パフォーマンス分析ツール、デバッグツール、IDE、およびその他の関連ツールごとに異なる章に編成されています。既知の問題、制限事項、および回避策の章では、Oracle Solaris Studio 12.3 のツールに関する追加情報について説明します。

このリリースに関する最新情報を引き続き入手するには、Oracle Technical Network の Oracle Solaris Studio ポータルにアクセスしてください。

◆ ◆ ◆ 第 2 章

# 2 コンパイラ

この章では、このリリースの Oracle Solaris Studio の C、C++、および Fortran コンパイラの新機能と変更された機能について説明します。

## 全コンパイラに共通の新機能/変更点

次に、C、C++、および Fortran コンパイラに共通する、前のリリースからの重要な変更点を一覧します。詳細は、コンパイラのマニュアルページおよびユーザーガイドを参照してください。

- 新しい SPARC T4 プラットフォームのサポート:
  `–xtarget=T4`、`–xchip=T4`、`–xarch=sparc4`
- 新しい x86 Sandy Bridge AVX プラットフォームのサポート:`–xtarget=sandybridge –xchip=sandybridge –xarch=avx`
- 新しい x86 Westmere AES プラットフォームのサポート:`–xtarget=westmere –xchip=westmere –xarch=aes`
- 新しいコンパイラオプション:`–g3` は、拡張されたデバッグシンボルテーブル情報を追加します。
- 新しいコンパイラオプション: `–Xlinker` *arg* は、arg をリンカー `ld`(1) に渡します。
- OpenMP のデフォルトのスレッド数 `OMP_NUM_THREADS` が 2 になりました (これまで 1 でした)。
- 3.1 OpenMP 共有メモリー並列化仕様のサポート。
- Sun Performance Library にリンクするには、`–library=sunperf` を使用します。これにより、`-xlic_lib=sunperf` が廃止になります。
- ユーザー提供のコンパイラオプションデフォルトのサポート。
- レガシー SPARC アーキテクチャー V7、V8、および V8a の `–xarch` サポートが削除されました。

# C コンパイラ

次に、C コンパイラに固有のバージョン 5.12 のこのリリースにおける新機能と変更された機能を列挙します。詳細は、『Oracle Solaris Studio 12.3: C ユーザーガイド』および cc のマニュアルページを参照してください。

- 新しいサブオプション –xbuiltin=%default は、errno を設定しない関数のみをインライン化します。errno の値はどの最適化レベルでも常に正確であり、高い信頼度でチェックできます。
- –xkeepframe オプションは、指定された関数のスタック関連の最適化を禁止します。
- -features=%none と -features=%all の使用はこのリリースで非推奨となりました。
- 新しい属性 vector_size と returns_twice が認識されます。
- -xcheck=init_local が、VLA (可変長配列) をその基本型に従って初期化するようになりました。
- aligned 属性の機能が、大域や静的のほかに自動も含むように拡張されました。
- -xdumpmacros は、定義済みマクロ、解除済みマクロ、実際の使用状況といった情報を提供します。
- 新しいオプション –xanalyze={code|no} はソースコードの静的分析を生成します。Oracle Solaris Code Analyzer を使用して表示できます。

# C++ コンパイラ

次に、C コンパイラに固有のバージョン 5.12 のこのリリースにおける新機能と変更された機能を列挙します。詳細は、『Oracle Solaris Studio 12.3: C++ ユーザーズガイド』および CC のマニュアルページを参照してください。

- 新規コンパイラオプション: –xivdep は ivdep プラグマの解釈を設定します。ivdep プラグマは、最適化の目的でループ内で検出された、配列参照へのループがもたらす依存関係の一部またはすべてを無視するようにコンパイラに指示します。これによってコンパイラは、マイクロベクトル化、分散、ソフトウェアパイプラインなど、それ以外の場合は不可能なさまざまなループ最適化を実行できます。これは、依存関係が重要ではない、または依存関係が実際に発生しないことをユーザーが把握している場合に使用されます。
- –compat=4 サブオプション (「互換モード」) は削除されました。デフォルトは –compat=5 になりました。さらに、Linux プラットフォーム上でのみ以前使用可能だった g++ ソースおよびバイナリ互換性のための -compat=g オプションが、Solaris/x86 にも拡張されました。
- 新規オプション –features=cplusplus_redef によって、通常は事前定義されているマクロ __cplusplus を、–D オプションによってコマンド行で再定義できるようになりました。__cplusplus をソースコード内の #define ディレクティブ経由で再定

義しようとすることは、引き続き許可されません。また、–features=%none と –features=%all の使用はこのリリースで非推奨となりました。

- 新しいサブオプション –xbuiltin=%default は、errno を設定しない関数のみをインライン化します。errno の値はどの最適化レベルでも常に正確であり、高い信頼度でチェックできます。
- C99 ヘッダー stdbool.h および C++ の同等の cstdbool が使用できます。C++ では、ヘッダーは効果がなく、C99 との互換性のために提供されています。
- 新しいオプション –xanalyze={code|no} はソースコードの静的分析を生成します。Oracle Solaris Code Analyzer を使用して表示できます。

# Fortran コンパイラ

次に、Fortran コンパイラに固有のバージョン 8.6 のこのリリースにおける新機能と変更された機能を列挙します。詳細は、『Oracle Solaris Studio 12.3: Fortran ユーザーズガイド』および f95 のマニュアルページを参照してください。

- 組み込み関数ルーチン LEADZ、POPCNT、および POPPAR には以前、引数の型として戻り型が指定されていました。このリリースでは Fortran 2008 規格に準拠するために、組み込み関数は引数型にかかわらずデフォルトで整数を返します。これによって、前のリリースとの軽微な非互換性が発生します。
- 多相性に関するオブジェクト指向の Fortran の機能がサポートされるようになりました。
  - サポートされる OOF 機能: 型拡張および多相要素: CLASS 文、無制限の多相性、SELECT TYPE 構文、ABSTRACT 構造型、EXTENDS_TYPE_OF および SAME_TYPE_AS 組み込み関数、および無制限ポインタへの連続型の割り当て。
  - サポートされない OOF 機能: 型束縛手続き: 型束縛 PROCEDURE 宣言、GENERIC、DEFERRED、NON_OVERRIDABLE、PASS、NOPASS。
- F2003/2008 のその他の新機能:
  - 拡張された構造体構成子: 成分名を使用した構造体定数の構築。
  - モジュール構造型および成分への拡張された PUBLIC/PRIVATE アクセス制御。
  - Fortran 2008 数学組み込み関数のサポートが増えました。ERFC_SCALED、NORM2、および x86 プラットフォームでの一部の REAL*16 形式を除くほとんどの Fortran 2008 数学組み込み関数がサポートされるようになりました。
  - 成分を持たない構造型。
  - KIND 引数が ICHAR、 IACHAR、 ACHAR、 SHAPE、 UBOUND、 LBOUND、 SIZE、 MINLOC、 MAXLOC、 COUNT、 LEN、 LEN_TRIM、 INDEX、 SCAN、 および VERIFY の組み込み関数に追加されました。
  - BACK 引数が MINLOC および MAXLOC 組み込み関数に追加されました。

- 新しい組み込み関数 FINDLOC および STORAGE_SIZE が追加されました。
- 新しいキーワード ERRMSG、SOURCE、および MOLD が ALLOCATE 文に追加され、ERRMSG が DEALLOCATE 文に追加されました。

# OpenMP

次に、このリリースの C、C++、および Fortran コンパイラによって実装された OpenMP 共有メモリー API の新機能と変更された機能を列挙します。詳細は、『Oracle Solaris Studio 12.3: OpenMP API ユーザーガイド』を参照してください。

- このリリースでは、3.1 OpenMP API 仕様が完全にサポートされています。
- PARALLEL および OMP_NUM_THREADS 環境変数のデフォルトが 2 に変更されました。これは、以前のリリースでは 1 でした。これは、自動並列化や明示的な OpenMP 並列化で、実行時に 2 つのスレッドの利点がデフォルトで得られることを意味します。以前はデフォルトで、これらのプログラムが 1 つのスレッド内で逐次的に実行されていました。
- SUNW_MP_PROCBIND 環境変数で、スレッドをプロセッサにバインドするための 2 つの新しいモード COMPACT と SCATTER が使用可能です。詳細は、『Oracle Solaris Studio 12.3: OpenMP API ユーザーガイド』を参照してください。

◆ ◆ ◆ 第 3 章

# 3 ライブラリ

この章では、Oracle Solaris Studio のこのリリースのライブラリに関する新機能および変更された機能について説明します。

## 数学ライブラリ

数学ライブラリ `libcx` はこのリリースで削除されました。

## Sun Performance Library

このリリースの Sun Performance Library は、Oracle Solaris オペレーティングシステムといくつかの Linux オペレーティングシステム環境で使用できます。

Sun Performance Library は、線形代数問題や非線形問題を数値的に解くための最適化された、かつ高速な数学サブルーチンを集めたものです。Sun Performance Library の基になっているのは `http://www.netlib.org/` の Netlib から入手できるパブリックドメインサブルーチンのコレクションであり、それが拡張および最適化され、Sun Performance Library としてバンドルされています。これには次のライブラリが含まれています。

- LAPACK version 3.1.1。線形代数問題解決用です。
- BLAS1 (基本的な線形台数サブプログラム)。ベクトルとベクトル演算実行用です。
- BLAS2。行列とベクトル演算実行用です。
- BLAS3。行列と行列演算実行用です。
- Netlib Sparse-BLAS。スパースベクトル演算実行用です。
- NIST Fortran Sparse BLAS version 0.5。基本的なスパース行列演算実行用です。
- SuperLU version 3.0。方程式のスパース線形システムの解決用です。
- 高速フーリエ変換 (FFT) ルーチン
- ダイレクトスパースソルバールーチン

# 互換性

Sun Performance Library の LAPACK 3.1.1 ルーチンは、1.x、2.0、3.0 などの以前のバージョンの LAPACK に含まれるユーザールーチン、および LAPACK 3.1.1 のすべてのルーチンと互換性があります。ただし、LAPACK 3.1.1 での内部変更のため、内部ルーチンとの互換性は保証できません。

互換性のない可能性がある内部ルーチンは、Netlib が提供している LAPACK ソースコードの中では auxiliary ルーチンと呼ばれています。『LAPACK User's Guide』に auxiliary ルーチンに関する情報があります。このガイドは、http://www.siam.org/ にある SIAM (Society for Industrial and Applied Mathematics) から入手できます。

LAPACK の auxiliary ルーチンのユーザーインタフェースは、LAPACK のリリースごとに変えることができるので、Sun Performane Library でも LAPACK の auxiliary ルーチンのユーザーインタフェースを変更できます。LAPACK 3.1.1 と互換性のある auxiliary ルーチンは、通常、ユーザーによる呼び出しに使用できますが、auxiliary ルーチンについてはマニュアルへの記載、テスト、およびサポートが特にされていません。LAPACK の auxiliary ルーチンのユーザーインタフェースは、Sun Performance Library の将来のリリースで変更される可能性があることに注意してください。そのため、ユーザーインタフェースは、該当バージョンの Sun Performance Library でサポートされる LAPACK のバージョンに対応します。

# ドキュメント

次に示す Sun Performance Library 関連文書が提供されています。

- マニュアルページ (セクション 3P) (英語版のみ) - ライブラリに含まれている各関数やサブルーチンに関する説明
- 『Oracle Solaris Studio Sun Performance Library User's Guide』では、Sun Performance Library ルーチンの使用方法、Fortran と C のインタフェース、最適化と並列化のオプション、SPSOLVE および SuperLU 疎ソルバーパッケージ、および FFT ルーチンについて、説明するとともに例を示します。

その他の情報については、『LAPACK User's Guide』 (第 3 版、Anderson, E. ほか著、SIAM、1999) を参照してください。SIAM (Society for Industrial and Applied Mathematics) または書店で入手できます。『LAPACK User's Guide』は、Netlib で提供している LAPACK 3.1.1 基本ルーチンに関する公式の解説書です。LAPACK 3.1.1 ルーチンについて、数学的に説明しています。

# このリリースの新機能および変更された機能

- 新しい Intel および SPARC プラットフォーム上での BLAS パフォーマンスを改善しました。

◆ ◆ ◆ 第 4 章

# 4 コード分析ツール

この章では、この Oracle Solaris Studio リリースでのコード分析ツールの新機能と変更された機能について説明します。

## Discover

このリリースで Discover メモリー分析ツールに追加された機能を、次に示します。

- 新しいオプション `-a` は、コードアナライザが使用するエラーデータを、*binary_name.analyze*`/dynamic` ディレクトリに書き込みます。
- 新しいオプション `-F` は、Discover で計測したバイナリの実行中にそのバイナリがフォークした場合の動作を決定します。デフォルトでは、Discover は親プロセスからメモリーアクセスエラーのデータを収集し続けます。Discover にフォークを追跡させ、子プロセスからメモリーアクセスデータを収集させるには、`discover` コマンドを使用してバイナリを計測する際に、`-F child` と指定します。
- 新しいオプション `-b` は、指定されたブラウザを、計測対象バイナリの実行中に自動的に起動します。
- 新しいオプション `-c` は、すべてのライブラリ内、指定されたライブラリ内、または指定されたファイル内の一連のライブラリ内でエラーをチェックするように、Discover に指示できるようにします。
- 新しいオプション `-n` は、実行可能ファイル内でエラーをチェックしないよう、Discover に指示します。

詳細は、`discover`(1) のマニュアルページと『Oracle Solaris Studio 12.3 Discover および Uncover ユーザーズガイド』を参照してください。

# Uncover

このリリースで Uncover コードカバレージツールに追加された機能を、次に示します。

- 新しいオプション `-a` は、コードアナライザが使用するエラーデータを、*binary_name.analyze*/`coverage` ディレクトリに書き込みます。
- 新しいオプション `-c` は、命令、ブロック、および関数の実行カウントの報告をオンにします。
- 新しいオプション `-o` は、指定されたファイルに計測対象バイナリファイルを書き込みます。

詳細は、`uncover`(1) のマニュアルページと『Oracle Solaris Studio 12.3 Discover および Uncover ユーザーズガイド』を参照してください。

# コードアナライザ

新しいツールであるコードアナライザを使用すれば、3 種類の分析を組み合わせることができるため、セキュリティー保護された堅牢で高品質な C および C++ アプリケーションを作成しやすくなります。コードアナライザは 3 種類のデータを表示します。

- `-xanalyze=code` オプションを指定してバイナリをビルドした場合に収集される、静的なコードの問題
- Oracle Solaris Studio のメモリーエラー発見ツールである Discover を使ってバイナリを計測および実行する場合に検出される、動的メモリーアクセスの問題
- Oracle Solaris Studio のコードカバレージツールである Uncover を使ってバイナリを計測および実行する場合に検出される、コードカバレージの問題

コードアナライザには、ソースファイル内で各問題が検出された場所のコードスニペット (該当ソース行が強調表示)、静的問題のエラーパス、および動的問題の「呼び出しスタック」 (および使用可能な場合は「スタックに割り当て」と「スタックで解放」) などの分析結果が表示されます。

エラーパス内またはスタック内のある関数呼び出しから関連するソースコード行にジャンプし、その関数のプログラム内でのすべての使用箇所を検索し、関数の宣言にジャンプし、関数のコールグラフを表示できます。

コードアナライザは、コード内の中核となる問題、つまり修正すればほかの問題も解消される可能性の高い問題を、正確に特定します。

詳細は、コードアナライザ GUI のオンラインヘルプ、『Oracle Solaris Studio 12.3 コードアナライザユーザーズガイド』、Oracle Solaris Studio 12.3 コードアナライザチュートリアル、および `code-analyzer`(1) のマニュアルページを参照してください。

◆ ◆ ◆ 第 5 章

# パフォーマンス解析ツール

この章では、Oracle Solaris Studio のこのリリースのパフォーマンス解析ツールに関する新機能および変更された機能について説明します。

## パフォーマンスアナライザ

この節では、このリリースの Oracle Solaris Studio パフォーマンスアナライザと関連ツールの、新機能と変更された機能について説明します。詳細は、『Oracle Solaris Studio 12.3: パフォーマンスアナライザ』マニュアルと、パフォーマンスアナライザ内の「ヘルプ」を参照してください。

### パフォーマンスアナライザツールの変更

パフォーマンスアナライザツールの機能は次のように拡張されています。

- 大きな実験 (特に Java 実験) を処理する際のパフォーマンスが、大幅に改善されています。
- データフィルタリングに多数の改善が施されましたが、これについては23 ページの「フィルタリングの拡張」で説明します。
- 大部分のデータタブに、タブ内で右クリックして開くコンテキストメニューが含まれるようになりました。これらのコンテキストメニューを使用すれば、フィルタリングのような、タブに固有の高度な機能を表示できます。
- 実験の比較モードで、異なる実行可能ファイルやロードオブジェクトの実験の比較がサポートされるようになりました。比較用のソースと逆アセンブリが、2 つのバージョンの分割区画内に表示されるようになりました。実験の比較モードの有効化は、「関数」および「ソース」タブのコンテキストメニューから行えます。
- タブ内でのすばやいナビゲーションの改善により、次のことが可能となりました。

  - 「関数」タブ内である関数をダブルクリックすると、その関数の「ソース」タブが開かれます。
  - 「行」タブ内である行をダブルクリックすると、その行かその行の近くで「ソース」タブが開かれます。
  - 「ソース」タブ内である行をダブルクリックすると、その行の最初の命令かその近くで「逆アセンブリ」タブが開かれます。
  - 「PC」タブ内である PC をダブルクリックすると、そのアドレスかそのアドレスの近くで「逆アセンブリ」タブが開かれます。
- 実験で参照されているソースファイルの検索に使用される方法が、変更されました。まずパスマップが試みられ、次に検索パスとパスマップを組み合わせたものが試みられ、その後、オリジナルのフルパスが試みられます。
- タイムラインに多数の改善が施されましたが、これについては22 ページの「タイムラインの拡張」で説明します。
- 「呼び出しツリー」タブでは HW サイクルがユーザー CPU 時間に変換されるため、呼び出しツリーに表示されるメトリックをコンテキストメニューから設定できます。
- 「スレッド」タブに、チャートと呼ばれる新しい表示モードが追加されました。クロックプロファイリングデータを含む実験でチャートを有効にすると、デフォルトの「ロードインバランス」チャートに、各スレッドに属する CPU 時間の合計量が表示されます。

## タイムラインの拡張

パフォーマンスアナライザの「タイムライン」タブに施された拡張を、次に示します。

- タイムラインを右クリックすると、データをフィルタリングしたり、イベントを選択したり、ズームイン/アウトを行ったり、前のビューに戻したり、タイムラインプロパティーを変更したりするためのコンテキストメニューが開きます。
- イベントの頻度を時間の関数として表示する折れ線チャートである、イベント頻度チャート。このチャートはデフォルトでは表示されず、「データ表示方法の設定」ダイアログボックスで選択する必要があります。
- さまざまな状態で費やされたアプリケーション時間の分布を時間の関数として表示する棒チャートである、イベント状態チャート。Oracle Solaris に記録されたクロックプロファイリングデータの場合は、イベント状態チャートに Oracle Solaris のマイクロステートが表示されます。このチャートはデフォルトでは表示されず、「データ表示方法の設定」ダイアログボックスで選択する必要があります。
- タイムラインでダブルクリックすると、関数カラーチューザダイアログが開かれる代わりに、ズームインするようになりました。

- 「イベント」タブが新しい「タイムラインの詳細」タブで置き換えられています。この新しいタブは、以前と同じくイベント情報を提供するほか、タイムラインのナビゲーション、ズーム、および関数カラーの変更を行うためのボタンも含んでいます。
- タイムラインでは、パフォーマンスアナライザの起動時にユーザーが指定したフォントが使用されるようになりました。
- 各実験から集約されたイベントを表示するように、タイムラインを設定することが可能です。これはデフォルトの設定ではなく、「データ表示方法の設定」ダイアログボックスの「タイムライン」タブで「データをグループ化: 実験」を選択することによって設定する必要があります。

## フィルタリングの拡張

パフォーマンスアナライザのデータフィルタリングが単純化され、拡張されました。

- 大部分のデータタブに、タブ内で右クリックして選択するコンテキストフィルタが含まれるようになりました。また、新規および既存のコンテキストフィルタの名前も理解しやすくなりました。
- コンテキストフィルタによってデータのフィルタリングが即時に行われるようになり、フィルタリングの適用をユーザーが別のダイアログボックスで行う必要はなくなりました。コンテキストメニューフィルタの選択は、すべてのタブで使用されるデータに影響を与えます。フィルタを選択するたびに、そのフィルタと既存のすべてのフィルタとの論理積が計算されるので、データのフィルタリングを段階的に行えます。
- 「データをフィルタ」ダイアログボックスが単純化され、その名前が「データフィルタの管理」に変更されました。ただし、データタブのコンテキストメニューからのフィルタリングが、データをフィルタリングするための推奨の方法です。

  「データフィルタの管理」ダイアログボックスの「カスタム」タブを使用すると、コンテキストメニューから適用されたフィルタの現在の状態を表示できます。「カスタム」タブは、現在のフィルタ式のユーザー編集を可能にします。またこれは、取り消し、再実行、およびコンテキストメニューフィルタから生成されたフィルタリング式に含めることのできるシンボルを記述したダイアログボックスである「キーワードの表示」もサポートします。
- 新しいコマンド `er_label` で追加されたラベルを使って実験をフィルタリングできます。
- オンラインヘルプに含まれるフィルタリングに関する情報が、拡張されています。

詳細は、『Oracle Solaris Studio 12.3: パフォーマンスアナライザ』の「データのフィルタリング」を参照してください。

# 新しいコマンド er_label

er_label コマンドを使用すると、実験の部分を定義し、それに名前つまりラベルを割り当てることができます。ラベルは、ユーザーが開始時刻と停止時刻のマーカーを使って定義した実験内の 1 つ以上の期間中に発生したプロファイリングイベントを取得します。

実験にラベルを割り当てるには、er_label コマンドをコマンド行で実行するか、またはそれをスクリプト内で実行します。実験へのラベルの追加が完了すると、それらのラベルを使用してフィルタリングを行えます。たとえば、実験をフィルタリングすることで、ラベルで定義される期間のプロファイリングイベントを含めたり除外したりできます。

er_label の使用方法の 1 つは、クライアントによって独立した 1 つ以上のプロセスとして駆動されるサーバープログラムのプロファイリングをサポートすることです。この使用モデルの場合、サーバーでの実験の作成を開始するため、collect コマンドを使ってサーバーを起動します。サーバーが起動され、クライアントの要求を受け入れる準備が整ったら、要求を発行してサーバーを駆動するとともに、er_label を実行してクライアント要求が発生する実験の部分にラベルを付けるようなクライアントスクリプトを実行できます。

詳細は、『Oracle Solaris Studio 12.3: パフォーマンスアナライザ』の「実験へのラベル付け」を参照してください。

# 実験の変更点

実験の形式が変更され、そのバージョン番号が 12.3 になりましたが、この番号は、Oracle Solaris Studio のバージョン番号に一致します。

Oracle Solaris Studio 12.3 のツールは、次のバージョン番号の実験を開くことができます。

- バージョン 10.1。Oracle Solaris Studio 12.2 または Sun Studio 12 update 1 で作成された実験。
- バージョン 10.2。Beta リリースなど、Oracle Solaris Studio 12.3 の早期リリースで作成された実験。
- バージョン 12.3。Oracle Solaris Studio 12.3 のリリース版で作成された実験。

以前のリリースから作成された実験を開こうとすると、以前のバージョンのツールでその実験を読み取る必要があることを示すエラーが表示されます。

# データ収集の変更点

データ収集の変更は、collect コマンド、dbx collector コマンド、および er_kernel コマンドに影響を与えます。

## collect ユーティリティー

collect ユーティリティーは、このリリースで次のように変更されています。

- collect はターゲットプログラムが ELF 実行可能ファイルであることを確認しないので、ユーザーは、環境変数を設定しなくてもスクリプトのプロファイリングを行えます。ターゲットが ELF 実行可能ファイルである場合、collect はそのターゲットが実行先マシンと互換性があるかチェックします。
- SPARC T4 チップ、および Intel の Westmere チップと Sandy Bridge チップに対するハードウェアカウンタサポートが、Oracle Solaris で追加されました。Linux の Westmere サポートは追加済みですが、Linux の Sandy Bridge サポートはまだ実装されていません。
- ハードウェアカウンタのプロファイリングで、任意の高精度ハードウェアカウンタの先頭に「+」を付けることで、任意のバイナリのメモリー領域プロファイリングを実行できるようになりました。高精度のハードウェアカウンタをサポートするプロセッサは、現時点では SPARC T4 と T3 だけです。

  どのハードウェアカウンタが高精度であるかを確認するには、collect -h コマンドの出力内でキーワード precise を探します。メモリー領域プロファイリングデータを含む実験でメモリーアクセスパターンを分析するには、「データ表示方法の設定」ダイアログボックスの「タブ」タブで、Vaddress や Vline_64b などのメモリーオブジェクトタブを選択します。すると、選択したメモリーオブジェクトタブからコンテキストフィルタを使用して、データアドレスによるフィルタリングを行えるようになります。
- 2.6.32 より大きいバージョンを持つ Linux カーネルを実行する Linux バージョンでのハードウェアカウンタサポートは PerfEvents フレームワークで実装されているため、カーネルパッチは不要ですが、以前のシステムでは、perfctr パッチが引き続き必要となります。
- collect を引数なしで実行したときに、使用方法のメッセージのみが表示されるようになりました。使用可能なハードウェアカウンタに関する情報を表示するには、collect –h を、ほかの引数を一切指定しないで実行する必要があります。
- collect -p high を使って実行される高精度のクロックプロファイリングが、Oracle Linux 6 など、それをサポートする Linux システム上で使用できるようになりました。

## dbx collector

dbx collector は、このリリースで次のように変更されています。

- SPARC T4、Westmere、および Sandy Bridge チップに対するハードウェアカウンタサポートが、Oracle Solaris で追加されました。Linux の Westmere サポートは追加済みですが、Linux の Sandy Bridge サポートはまだ実装されていません。
- 2.6.32 より大きいバージョンを持つ Linux カーネルを実行する Linux バージョンでのハードウェアカウンタサポートは `PerfEvents` フレームワークで実装されているため、カーネルパッチは不要ですが、以前のシステムでは、`perfctr` パッチが引き続き必要となります。

## er_kernel ユーティリティー

Oracle Solaris カーネルのプロファイリングを行うための `er_kernel` ユーティリティーは、次のように変更されています。

- `er_kernel` で、カーネルとアプリケーションのプロファイリングを実行できるようになりました。`-F` オプションを使用すれば、アプリケーションのプロセスを追跡し、そのデータをカーネルの実験のサブ実験として記録するかどうかを制御できます。
- `er_kernel` では、特定のプロセスをプロファイリングするための `-T` オプションがサポートされなくなりました。代わりに、`-F` オプションと正規表現を使用できます。
- `er_kernel` ユーティリティーは、DTrace の `cpc` プロバイダを使ってカーネルのハードウェアカウンタオーバーフロープロファイルを収集できますが、このプロバイダは、Oracle Solaris 11 を実行しているシステム上でのみ使用できます。カーネルのハードウェアカウンタオーバーフロープロファイリングを実行するには、`collect` コマンドの場合と同様に、`-h` オプションを `er_kernel` コマンドで使用します。ただし、`er_kernel` ではデータ領域プロファイリングはサポートされていないため、データ領域要求は無視されます。
- `er_kernel` を引数なしで実行したときに、使用方法のメッセージのみが表示されるようになりました。使用可能なハードウェアカウンタに関する情報を表示するには、`er_kernel –h` を、ほかの引数を一切指定しないで実行する必要があります。
- チップ上のハードウェアカウンタオーバーフローのメカニズムにより、どのカウンタがオーバーフローしたかをカーネルが知ることができる場合には、チップが提供するすべてのカウンタのプロファイリングを行えますが、それ以外の場合、指定できるカウンタは 1 つだけです。`er_kernel –h` の出力には、複数のカウンタを使用できるかどうかを示すために、「最大 4 つの HW カウンタを使用する HW カウンタプロファイリングを指定できます」といったメッセージが表示されます。
- `er_kernel` はターゲット負荷が ELF 実行可能ファイルであることを確認しないので、ユーザーは任意のコマンドやスクリプトのプロファイリングを行えます。

## er_print コマンド

`er_print` コマンドはこのリリースでは次のように変更されています。

- 大きな実験 (特に Java 実験) を処理する際のパフォーマンスが、大幅に改善されています。
- 実験で参照されているソースファイルの検索に使用される方法が、変更されました。まずパスマップが試みられ、次に検索パスとパスマップを組み合わせたものが試みられ、その後、オリジナルのフルパスが試みられます。
- `tlmode` サブコマンドを使用すれば、各実験から集約されたイベントがタイムラインに表示されるようにするためのデフォルトを設定できます。

## スレッドアナライザ

Oracle Solaris Studio 12.3 のスレッドアナライザで追加または変更された機能を、次に示します。

- アプリケーションが競合検出プロファイリング用に計測されていない場合に、スレッドアナライザからユーザーにその旨が通知されるようになりました。
- スレッドアナライザの「競合」タブで、最初の呼び出しスタックが自動的に開かれるようになりました。

# DLight

DLight は、Oracle Solaris Dynamic Tracing (DTrace) テクノロジを使用した対話型グラフィカル可観測性ツールです。DLight は、複数の DTrace スクリプトを同期された方法で実行しつつ、その出力をグラフィカル表示することで、ユーザーがアプリケーションの実行時の問題を追跡してその根本原因を特定するのを支援します。

Oracle Solaris Studio 12.3 の DLight で追加または変更された機能を、次に示します。

- 新しいプロセスツリーターゲットを使用すると、あるプロセスとそのプロセスによって作成されたすべてのプロセスのプロファイリングが可能となりますが、次がグラフィカル表示されます。
  - DLight がプロセスツリーターゲットでプロファイリングしたすべてのプロセスのすべてのスレッドのマイクロステートを集約したもの。
  - ターゲットプロセスとその子プロセスのスレッドごとのマイクロステートの移り変わり (タイムライン形式)。
  - プロセスとその子のロック統計。
  - 実行先となるすべての CPU にわたってプロファイリングされているすべてのターゲットプロセス内のすべてのスレッドの CPU 使用時間を集約したもの。

  - プログラムのプロセスツリー内の CPU インテンシブな領域 (プログラム内の関数、および各関数とそこから呼び出されるすべての関数で使用される CPU 時間を表示)。
- 実行中の単一プロセスのプロファイリングに使用される接続ターゲットの名前が、プロセスターゲットに変更されました。
- AMP ターゲットが削除されました。

詳細は、DLight 内のヘルプとOracle Solaris Studio 12.3: DLight チュートリアルを参照してください。

◆ ◆ ◆ 第 6 章

# 6 デバッグツール

Oracle Solaris Studio のこのリリースのデバッグツールに関する新機能です。

## dbx

### 新機能および変更された機能

Oracle Solaris Studio 12.3 dbx で追加または変更された機能は次のとおりです。

- `dbx` がマクロ展開を含むようになりました。詳細は、『Oracle Solaris Studio 12.3: dbx コマンドによるデバッグ』の付録 C「マクロ」を参照するか、`dbx` の実行時に (dbx) プロンプトで `help macros` と入力してください。
- オブジェクト指向の Fortran サポート:
  - `dbx` で型拡張と多相ポインタがサポートされるようになりました。これは、C++ サポートと一致しています。
  - `output_dynamic_type` および `output_inherited_members` dbx 環境変数が Fortran でも動作するようになりました。
  - `print` および `whatis` コマンドで `-r`、`+r`、`-d`、および `+d` オプションを使用すると、継承される (親) 型および動的型に関する情報を取得できます。
- `dbx` が、Fortran の割り当て可能配列型のほかに割り当て可能スカラー型もサポートするようになりました。

◆ ◆ ◆ 第 7 章

# Oracle Solaris Studio IDE

Oracle Solaris Studio 12.3 IDE (Integrated Development Environment) は、C、C++、または Fortran アプリケーションの作成、編集、構築、デバッグ、およびパフォーマンス分析を行うためのモジュールを提供します。この章では、Oracle Solaris Studio のこのリリース IDE についての重要な情報について説明します。

IDE を起動するコマンドは `solstudio` です。このコマンドについての詳細は、`solstudio`(1) のマニュアルページを参照してください。

IDE の完全なドキュメントについては、IDE 内のオンラインヘルプとOracle Solaris Studio 12.3 IDE クイックスタートチュートリアルを参照してください。

## 新機能および変更された機能

Oracle Solaris Studio 12.3 IDE で追加または変更された機能は次のとおりです。

- NetBeans IDE 7.0.1 が基になっています。
- 新しいプロジェクトタイプ「バイナリファイルからの C/C++ プロジェクト」を使用すれば、既存のバイナリからプロジェクトを作成するために、バイナリファイル、その構築元となるソースファイルの場所、プロジェクトに含めるファイル、および依存関係をプロジェクトに含めるかどうか、を指定できます。
- リモートホストのファイルシステム上のファイルを参照したり編集したりするなど、ローカルホストで定義されたリモートホスト上のプロジェクトを操作できるようになりました。
- リモートホストの端末ウィンドウを開くことができます。
- 新機能のテンプレート特殊化は、汎用テンプレートとテンプレート特殊化との間のナビゲーションを単純化します。このナビゲーションを使用するには、ソースエディタの余白にある注釈アイコンを右クリックするか、あるいは Ctrl+Alt キーを押してテンプレートクラスまたはテンプレートメソッドを右クリックします。

- Oracle Database アプリケーション用のプロジェクトを作成できます。これを実行するには、使用中の Oracle Solaris Studio インストールに、省略可能な Oracle Instant Client コンポーネントが含まれている必要があります。ProC サポートが IDE に含まれるようになりました。
- ソースエディタは、ユーザーの入力中にプロジェクトに対して静的なコードエラーチェックを行い、エラーが検出されると左余白にエラーアイコンを表示します。
- プロジェクトに対してメモリーアクセスエラーチェックを実行できます。
- 新しいプロジェクトプロパティー「コマンドを実行」を使用すれば、プロジェクト実行時にプロジェクトのコマンド行に提供するコマンドと引数を指定できます。実行コマンドとしては、シェルスクリプトを指定できますが、ライブラリプロジェクトの場合はバイナリを指定できます。
- gcc ツールコレクションを使ってコンパイルされたコードで `gdb` デバッガを使用するオプションが追加されました。`dbx` デバッガがデフォルトです。
- 新機能のデスクトップ配布を使用すれば、ほとんどすべてのオペレーティングシステム上で動作し、リモートサーバー上の Oracle Solaris Studio のコンパイラやツールを使用する IDE とコードアナライザの配布を含む ZIP ファイルを生成できます。デスクトップシステムで IDE を実行すると、IDE は、配布の生成元となったサーバーをリモートホストとして認識し、Oracle Solaris Studio インストール内のツールコレクションにアクセスします。

詳細は、IDE 内のオンラインヘルプと Oracle Solaris Studio 12.3 IDE クイックスタートチュートリアルを参照してください。

# ソフトウェア要件

Oracle Solaris Studio IDE には、Java SE Development Kit (JDK) 6 Update 24 以降が必要です。IDE は、必要な JDK が見つからない場合は、起動せず、エラーメッセージを表示します。

# IDE の更新

Oracle Solaris Studio 12.3 の IDE、dbxtool、DLight 可観測性ツール、およびコードアナライザに対するアップデートはすべて、IDE でデフォルトで無効になっている NetBeans オートアップデート機能経由で配布されるのではなく、Oracle Solaris Studio 製品パッチとして配布されます。

次の場合には、そのような製品パッチのインストール時にこれらのツールで競合が発生する可能性があります。

- ツール内でオートアップデート機能が有効化されており、かつ自動的なアップデートが発生した場合。

- NetBeans アップデートセンターからプラグインをインストールした場合。

競合を解決する方法:

- Solaris 10 上でパッケージインストーラを使用して、または Solaris 11 上で IPS リポジトリから、Oracle Solaris Studio ツールをインストールした場合、`ide-12.3-`*OS*`-`*architecture* (IDE または DLight 用)、`dbxtool-12.3–`*OS*`-`*architecture*、または `code-analyzer–12.3–`*OS*`-`*architecture* を、`~/.solstudio` の Oracle Solaris Studio ユーザーディレクトリから削除します。
- ダウンロード tar ファイルを使用して Oracle Solaris Studio ツールをインストールした場合は、その tar ファイルを再インストールします。

# 構成

NetBeans IDE 7.0.1 のデフォルトのヒープサイズは、システムで使用可能なメモリーの量との関係で自動的に決定されます。最大 500 個のソースおよびヘッダーファイルを含む小規模プロジェクトを開発する場合、Oracle Solaris Studio 12.3 IDE は一般にデフォルト設定で正常に動作します。

より規模の大きいプロジェクトを開発する場合は、ヒープサイズを増加する必要があります。大規模なプロジェクトの開発時に OutOfMemory 例外が発生した場合は、ヒープサイズが原因であることがあります。

NetBeans IDE の実行元となる Java Virtual Machine (JVM)* のヒープサイズは、`netbeans.conf` ファイルで設定できます。

ヒープサイズを変更するには、次の手順に従います。

- */Oracle_Solaris_Studio_installation_directory* `/lib/netbeans/etc/netbeans.conf` ファイルで、`–J-Xmx` コマンド行 Java 起動スイッチを `netbeans.conf` ファイルに追加したあと、IDE を再起動します。

  例:

```
netbeans_default_options="-J-Xms32m -J-Xmx128m –J-XX:PermSize=32m
     –J-XX:MaxPermSize=96m –J-Xverify:none –J-Dapple.laf.useScreenMenuBar=true"
```

NetBeans C/C++ Plugin での中規模および大規模のアプリケーションの推奨ヒープサイズを次に示します。

- 1G バイト以上の RAM のシステム上での中規模アプリケーション開発 (500~2000 ソースおよびヘッダーファイル) : 512M バイト
- 2G バイト以上の RAM が搭載されたシステム上で、大規模アプリケーション (2000 個を超えるソースおよびヘッダーファイル) を開発する場合:1G バイト

Oracle JVM を実行している場合は、ガベージコレクタスイッチ `-J-XX:+UseConcMarkSweepGC` (並行コレクタ) と `-J-XX:+UseParNewGC` (パラレルコレクタ)

を netbeans.conf ファイルに追加することもできます。これらのオプションによって、ガベージコレクタを主実行エンジンと並行して実行できます。これらは、Oracle 実装以外の JVM ではサポートされていない可能性があります。

NetBeans のパフォーマンスチューニングの詳細については、Tuning JVM Switches for Performance を参照してください。

注:「Java 仮想マシン」および「JVM」という用語は、Java(TM) プラットフォーム用の仮想マシンを意味します。

◆ ◆ ◆ 第 8 章

# 8 その他のツール

Oracle Solaris Studio のこのリリースの dmake およびソフトウェアインストーラに関する新機能です。

## dmake

dmake はコマンド行ツールであり、make(1) と互換性があります。dmake は、グリッド、分散、並列、または逐次モードでターゲットを構築できます。標準的な make(1) ユーティリティーを使用している場合は、dmake への切り替えに伴ってメイクファイルに変更を加える必要があるとしても、変更はわずかです。dmake は、make ユーティリティーの超集合です。make を入れ子にするときは、最上位 makefile が make を呼び出す場合に $(MAKE) を使用する必要があります。dmake はメイクファイルを解析し、並行して構築可能なターゲットを特定し、設定された多数のホストにそれらのターゲットの構築作業を分散します。

dmake は Oracle Solaris Studio IDE に統合されています。デフォルトでは、すべてのプロジェクトは dmake を使用して構築されており、並列モードで実行します。「プロジェクト」プロパティーでは、構築ジョブの最大数を指定できます。デフォルトでは、dmake は 2 個のジョブを並列実行しますが、これは、マルチ CPU システム上で多数のプロジェクトの構築速度が 2 倍になることを意味します。

dmake の使用方法については、『Oracle Solaris Studio 12.3: 分散メイク (dmake)』マニュアルを参照してください。

このリリースの dmake ユーティリティーに追加された機能を、次に示します。

dmake が、構築サーバー上でコマンドをリモート実行する際に、rsh に加えて ssh を使用できるようになりました。ssh を使用する場合は、ssh コマンドへのリモートパスを .dmakerc ファイルに指定する必要があります。

リモートシェルのパスは .dmakerc ファイルに指定できます。

例:

```
host earth { jobs = 3 }
host mars  { jobs = 5 , rsh = "/bin/ssh" }
```

rsh = が指定されない場合、dmake はデフォルトで /bin/rsh を使用します。

rsh と同様に、ssh がパスワードを必要とせずにリモートホストにログインでき、警告またはエラーを出さないようにする必要があります。

## このリリースでのソフトウェアの修正事項

- 修正されたバグ:dmake は、エスケープされていない :' を .make.state に書き込み、自身を破壊します。
- 修正されたバグ:マニュアルページの更新:新しいオプション -m grid(SGE サポート)。
- 修正されたバグ:dmake のマニュアルページの「機能説明」セクションで、「分散」モードが抜けています。
- 修正されたバグ:dmake でコマンド行オプション -x SUN_MAKE_COMPAT_MODE=GNU が無視されます。

# Oracle Solaris Studio インストーラ

インストーラの新機能および変更された機能は次のとおりです。

- 非 GUI インストーラで、インストールするコンポーネントを指定できるようになりました。
- 新しいインストーラオプション -generate-desktop-dir を使用すると、ほとんどすべてのオペレーティングシステムを持つデスクトップシステム向けに構成された IDE の配布を含む ZIP ファイルを生成できます。Oracle Solaris Studio ソフトウェアのインストールが完了したら、このファイルをデスクトップシステム上で解凍できます。デスクトップシステムで IDE を実行すると、IDE は、配布の生成元となったサーバーをリモートホストとして認識し、Oracle Solaris Studio インストール内のツールコレクションにアクセスします。
- 新しいインストーラオプション -nfs-server ではインストーラが NFS サーバーモードで実行されますが、この場合、サーバー上に必要な OS パッチが存在しているかのチェックは行われず、Oracle Solaris Studio ソフトウェアとマニュアルページへのシンボリックリンクは /usr/bin および /usr/share/man ディレクトリにインストールされません。
- 新しいインストーラオプション -ignore-architecture を使用すれば、SPARC ベースプラットフォーム用の Oracle Solaris Studio コンポーネントを x86 ベースプラットフォームにインストールしたり、x86 ベースプラットフォーム用のコンポーネントを SPARC ベースプラットフォームにインストールしたりできます。

- 新しいアンインストーラオプション `-force-uninstall` を使用すれば、NBI レジストリが破壊された場合に、Oracle Solaris Studio パッケージとインストールディレクトリを強制的に削除できます。

◆ ◆ ◆ 第 9 章

# 9 このリリースでの既知の問題、制限事項、および回避策

ここでは、このリリースの時点で確認されている問題およびそれらの問題の回避方法についての情報を説明します。その後明らかになった問題はすべて、Oracle Solaris Studio 12.3 リリースノートに記載されています。

# コンパイラ

ここでは、このリリースでのコンパイラに関する既知の問題および回避策について説明します。

## コンパイラに共通する問題

### ドキュメントの誤り

発行済みコンパイラドキュメントに含まれる誤りを、次に列挙します。

- cc(1)、CC(1)、および f95(1) のマニュアルページに、3dnow などの AMD 命令セットを SSE3 命令セットに追加する –xarch=sse3a フラグを記載し忘れています。
- C と C++ のドキュメントで、–xMF オプションと併用できるのは –xMD または –xMMD だけであり、–xM や –xM1 は併用できない点を指摘し忘れてします。これを指定すると、これらのオプションで使用されているデフォルトの .d ファイルの名前が上書きされます。

# C++

## Solaris 上の Apache 標準ライブラリの問題

Solaris 10u10 以前、および初期リリースの Solaris 11 にインストールされた Apache stdcxx ライブラリでは、ヘッダー stdcxx4/loc/_moneypunct.h で構文エラーが発生します。このエラーは、以前のコンパイラでは見られなかったものですが、Oracle Solaris Studio 12.3 C++ コンパイラでは検出されます。このエラー検出を無効にする方法はありません。

このバグの修正は、Solaris 10 のパッチ、および最初の Solaris 11 SRU として入手できます。修正が使用可能になった時点で、その修正は Solaris 10u11 と Solaris 11u1 に含められます。

## あいまいさ: コンストラクタ呼び出しまたは関数へのポインタ

C++ では、あるときは宣言と解釈されたり、またあるときは式と解釈される可能性がある文があります。C++ のあいまい排除規則では、ある文を宣言文とみなすことができる場合は、その文は宣言文とすることになっています。

従来のバージョンのコンパイラでは、次のような事例を誤って解釈していました。

```
struct S {
  S();
};
struct T {
  T( const S& );
};
T v( S() );    // ???
```

このプログラマはおそらく、最後の行で S 型の一時的な値で初期化される変数 v を定義するつもりでした。従来のバージョンのコンパイラは、この文をそのように解釈していました。

しかし、宣言コンテキスト内のコンストラクト、"S()" は、"S 型の値を戻すパラメータのない関数" を意味する抽象宣言子 (識別子のない抽象宣言子) とみなすこともできます。この事例では、関数ポインタ、"S(*)()" に自動的に変換されています。この文はまた、戻り値が T 型で、パラメータが関数ポインタ型の関数 v の宣言としても有効です。

現在ではコンパイラが正しい解釈をするようになったので、このプログラマが意図したようにならない可能性があります。

あいまいにならないようにコードを修正するには、次の 2 通りの方法があります。

```
T v1( (S()) );  // v1 is an initialized object
T v2( S(*)() ); // v2 is a function
```

1 行目の 1 対の余分な括弧は、v1 の構文が関数宣言としては不正であるので、"S 型の一時的な値で初期化される T 型のオブジェクト" という意味にしか解釈できません。

同様に、コンストラクト "S(*)()" は値とは考えられないので、関数宣言の意味にしか解釈できません。

1 行目は次のように記述することもできます。

```
T v1 = S();
```

意味は完全に明確になりますが、この初期設定の形式では、通常はそうでもないとはいえ、一時的な値として非常に大きな値が生成されることがあります。

次のようにコーディングするのはお勧めできません。その理由は、意味が不明確で、コンパイラが異なると結果が異なる可能性があるからです。

```
T v( S() ); // 推奨しない
```

## -instances=static で -xipo または -xcrossfile があると、リンクに失敗する

テンプレートオプションの -instances=static (または -pto) を -xcrossfile や -xipo オプションと組み合わせると、機能しません。この組み合わせを使用したプログラムは、リンクに失敗することがよくあります。

-xcrossfile または -xipo オプションを使用する場合は、デフォルトのテンプレートコンパイルモデルの -instances=global を使用してください。

一般に、-instances=static (および -pto) は使わないでください。使うメリットはすでになく、依然として、『C++ ユーザーズガイド』で説明しているデメリットがあります。

## リンク時の名前符号化の問題

次の場合に、リンク時に問題が発生することがあります。

- const パラメータ付きで宣言されている関数が、別の場所で const パラメータなしで宣言されている。

  例:

  ```
  void foo1(const int);
  void foo1(int);
  ```

  これらの宣言は等価ですが、コンパイラは異なる符号化名を付けます。この問題を回避するには、値のパラメータを const として宣言しないでください。たとえば、関数定義の本体などのあらゆる場所で void foo1(int); を使用します。

- 関数に同じ複合型のパラメータが 2 つあり、一方のパラメータだけ typedef で宣言されている。

  例:

  ```
  class T;
  typedef T x;
  // foo2 has composite (that is, pointer or array)
  ```

```
// parameter types
void foo2(T*, T*);
void foo2(T*, x*);
void foo2(x*, T*);
void foo2(x*, x*);
```

すべての foo2 宣言は等価で、これらは同じものを符号化する必要があります。しかし、コンパイラは一部に異なった符号化を行なっています。この問題を回避するには、一貫して typedef を使用します。

typedef を一貫して使用できない場合は、回避策として、関数を定義しているファイルに weak シンボルを使用し、宣言とその定義を等価にします。例:

```
#pragma weak "__1_undefined_name" = "__1_defined_name"
```

ターゲットアーキテクチャーによって異なる符号化名があります。たとえば、size_t は SPARC V9 アーキテクチャー (m64) では unsigned long ですが、それ以外のアーキテクチャーでは unsigned int です。これは、2 つの異なったバージョンの符号化名がそれぞれ 1 つのモデルに存在するケースです。このような場合は、2 つのプラグマを用意し、適切な #if 指令で制御する必要があります。

## 大域的ではない名前空間のオブジェクトをテンプレートから参照できない

プログラムでテンプレートと静的オブジェクトを使用していると、-instances=extern を指定してコンパイルした場合に未定義シンボルのリンク時エラーが発生します。これは、デフォルト設定の -instances=global では問題になりません。コンパイラは、大域的でない名前空間スコープのオブジェクトに対するテンプレートからの参照をサポートしません。次の例を考えてみましょう。

```
static int k;
template<class T> class C {
        T foo(T t) { ... k ... }
};
```

この例では、テンプレートクラスのメンバーは静的な名前空間スコープ変数を参照します。名前空間スコープはファイルスコープを含むことに注意してください。コンパイラは、静的な名前空間スコープ変数を参照するテンプレートクラスのメンバーをサポートしません。複数のコンパイル単位からテンプレートがインスタンス化されると、各インスタンスは異なる k を参照します。つまり、C++ 単一定義規則違反が発生し、コードは定義されていない動作を起こします。

k の使用方法やその効果に応じて、次の代替案が考えられます。2 番目のオプションは、クラスのメンバーの関数テンプレートにのみ使用できます。

1. 変数に外部リンケージを持たせる

   ```
   int k; // not static
   ```

   すべてのインスタンスは、k の同じコピーを使用します。

2. 変数をクラスの静的メンバーにする

```
template<class T> class C {
        static int k;
        T foo(T t) { ... k ... }
};
```

静的なクラスメンバーは外部リンケージを持ちます。`C<T>::foo` のインスタンスが使用する `k` はそれぞれ異なります。`C<T>::k` のインスタンスは、ほかの関数で共有することができます。通常はこのオプションが使用されます。

### 名前空間内の #pragma align と符号化名

名前空間内で `#pragma align` を使用する場合は、符号化名を使用する必要があります。たとえば、次のコードでは、`#pragma align` 文は何の働きもしません。この問題を解決するには、`#pragma align` 文の `a`、`b`、および `c` を符号化された名前に変更します。

```
namespace foo {
  #pragma align 8 (a, b, c) // has no effect
  //use mangled names: #pragma align 8 (__1cDfooBa_, __1cDfooBb_, __1cDfooBc_)
  static char a;
  static char b;
  static char c;
}
```

## Fortran

このリリースの `f95` コンパイラでは、次の問題に注意する必要があります。

- 前進なし印刷行の末尾の前にある空白が、出力の位置に影響を与えない (7087522)。

  出力文の書式の末尾に X 編集記述子を指定しても、出力レコード内の次の文字の位置がその影響を受けません。これによって違いが生まれるのは、出力文に **`ADVANCE='NO'`** が含まれていて、かつ後続の出力文によって同じレコードに転送される文字がほかに存在している場合です。

  多くの場合、これを回避するには、***n*X** 編集記述子の代わりに空文字列編集記述子を使用します。これらは厳密には同じではありませんが、それは、空文字列編集記述子の場合は実際に空文字がレコード内に挿入されるのに対し、***n*X** では次の *n* 個の文字がスキップされるだけであり、それによって通常は、それらのスキップされた位置がデフォルトで空になるからです。

- 2 つの継続アンパサンドから成る行が存在していると、有効なコードが拒否される。(7035243)。

  単一のアンパサンドを含む空の継続行は、Fortran 規格で禁止されています。しかし、同じ行に 2 つのアンパサンドが含まれていると、規格の制限の対象から外れ、空の継続行を引き続き作成できます。コンパイラではそのような場合は処理

されず、エラーが発生します。回避方法は、その行を削除することであり、それを行っても、プログラムの可読性に影響があるだけであり、意味が付加されることは一切ありません。

- BOZ 定数が切り捨てられることがある (6944225)。

  配列構成のように比較的複雑な一部のシナリオでは、BOZ 定数が、その代入先となると思われる対応する項目が 8 バイト整数エンティティーであっても、デフォルト整数サイズの 4 バイトに切り捨てられる可能性があります。回避方法は、BOZ 定数の代わりに正しい型と種別を持つ定数を、配列構成で使用することです。

以前のリリースの Fortran コンパイラで非互換性が導入されましたが、その非互換性はこのリリースのコンパイラにも引き継がれており、以前の Fortran コンパイラリリースからアップデートする際に注意する必要があります。互換性の問題とは次のとおりです。

## Fortran 77 ライブラリが削除された

ここでは、Oracle Solaris Studio 12.2 リリースで廃止対象の FORTRAN 77 ライブラリが削除されたことに注意してください。これは、レガシー Sun WorkShop `f77` コンパイラでコンパイルされた、共有ライブラリ `libF77`、`libM77`、および `libFposix` に依存する古い実行可能ファイルが動作しないことを意味しています。

## 配列組み込み関数における大域レジスタの使用

配列組み込み関数の
`ANY`、`ALL`、`COUNT`、`MAXVAL`、`MINVAL`、`SUM`、`PRODUCT`、`DOT_PRODUCT`、`MATMUL` は、各 SPARC プラットフォームアーキテクチャー用に高度に調整されています。このため、これらの関数は大域レジスタの `%g2`、`%g3`、`%g4` をスクラッチレジスタとして利用します。

上記の配列組み込み関数を呼び出す場合に、これらのレジスタが一時記憶領域として利用できることを前提にしたユーザーコードを作成しないでください。これらのレジスタ内のデータは、配列組み込み関数を呼び出したときに上書きされます。

## アーカイブライブラリ内の F95 モジュールが実行可能ファイルに含まれない

デバッガ dbx では、コンパイルに使用されたすべてのオブジェクトファイルが実行可能ファイルの中に含まれている必要があります。通常、ユーザーが追加の作業を実行しなくても、プログラムはこの要件を満たしています。例外となるのは、モジュールを含むアーカイブを使用している場合です。プログラムがモジュールを使用するが、モジュール内の手順または変数をいずれも参照しない場合は、結果として生じるオブジェクトファイルには、モジュール内で定義されるシンボルへの参照は含まれません。オブジェクトファイル内で定義されているシンボルへの参照がある場合のみ、リンカーはアーカイブから取得したオブジェクトファイルをリンクし

ます。このような参照が存在しない場合は、オブジェクトファイルは実行可能ファイルに含められません。使用されたモジュールに関連するデバッグ情報を dbx が検索しようとする場合に、dbx は警告を生成します。デバッグ情報が見つからないシンボルに関する情報は提供できません。

この問題を回避するためには、-u リンカーオプションを使用します。このオプションは、1 つのシンボルをそのオプション引数として取ります。そのシンボルを未定義のリンカーシンボルのセットに追加し、問題を解決します。モジュールと関連付けられているリンカーシンボルは通常、小文字の文字列に下線が後続するモジュール名です。

たとえば、モジュール `MODULE_1` を含むオブジェクトファイルをアーカイブから取り出すには、リンカーオプション `-u module_1_` を指定します。f95 コマンドを使用してリンクを実行する場合、コマンド行で `-Qoption ld -umodule_1_` を使用してください。

### Linux プラットフォームの gethrtime(3F)

システムの省電力が有効になっている場合、AMD プロセッサのクロックレートを正確に取得するための信頼できる方法はありません。結果として、Linux プラットフォーム上で `gethrtime`(3F) (Fortran コンパイラの、Solaris `gethrtime`(3C) 関数の Linux 版) に基づく時間関数を使用して実際の時間を高精度で取得する場合、精度の高い値が得られるのは、AMD システムの省電力が無効になっている場合だけです。省電力機能を無効にするには、システムをリブートしなければいけない可能性があります。

# ツール

## dbx

### dbx に関する既知の問題と回避策

1. `dbx` がプロセスに接続された場合のデータ収集の問題

   コレクタライブラリ `libcollector.so` を事前にロードしないで `dbx` を実行中のプロセスに接続すると、さまざまなエラーが発生する可能性があります。

   - どのトレーシングデータも収集できません。同期はトレーシング、ヒープトレーシング、または MPI トレーシングを待機します。トレースデータの収集は各種ライブラリに割り込むことで行われますが、`libcollector.so` が事前にロードされていなければ、割り込みを行えません。
   - dbx がプロセスに接続されたあとにシグナルハンドラがインストールされ、そのシグナルハンドラが SIGPROF シグナルおよび SIGEMT シグナルを転送しない場合、プロファイルデータと標本データが失われます。

- プログラムが非同期入出力ライブラリ libaio.so を使用している場合、libaio.so が SIGPROF を使用して非同期の取り消し操作を行うため、時間ベースのプロファイルデータと標本データは失われます。
- プログラムがハードウェアカウンタライブラリ libcpc.so を使用している場合、コレクタとプログラムが両方ともこのライブラリを使用するため、ハードウェアカウンタオーバーフロープロファイリング実験が破壊されます。dbx がプロセスに接続されたあとでハードウェアカウンタライブラリがロードされた場合、libcpc.so 内での検索ではなく一般的な検索によって libcpc ライブラリ関数への参照が解決されるのであれば、ハードウェアカウンタ実験が成功する可能性があります。
- プログラムが setitimer(2) を呼び出す場合、コレクタとプログラムの両方がタイマーを使用しているため、時間ベースのプロファイリング実験に失敗する場合があります。

2. Java コードのデバッグ中に dbx がクラッシュする可能性がある

   dbx シェル内から cd コマンドを発行したり、CLASSPATH 環境変数または CLASSPATHX 環境変数を設定したりすると、その後、dbx がセグメント例外でクラッシュする可能性があります。

   回避策:

   - 上記の実行もしくは設定を行わない。
   - 上記の実行もしくは設定を行う前に、すべてのウォッチポイント (表示) を削除する。

3. Java コードの再デバッグ時に dbx がクラッシュする

   Java コードに対してデバッグコマンドを 2 回続けて発行すると、dbx がクラッシュする可能性があります。

4. アプリケーションを構築時とは異なる J2SE 上でデバッグすると、dbx から例外がスローされる

   アプリケーションの構築時に使用した J2SE テクノロジのバージョンとは異なるリリースの J2SE テクノロジの下でそのアプリケーションをデバッグすると、dbx から例外がスローされます。

5. RTC 前の監視割り当てが原因でRUA エラーが誤ってレポートされていた

   マルチスレッドプログラムを使用する例外的な状況下で、実行時検査 (RTC) がメモリー割り当ての監視を開始する前に割り当てられた内部スレッド関連データへのアクセスを検出したときに、RTC は RUA エラーを誤ってレポートします。このような状況は、通常のスレッド切り替え動作の一部なので、dbx suppress コマンドを使用することにより、このような誤った RUA レポートを安全に無視できます。

## dbx の制限事項と非互換性

Oracle Solaris Studio 12.3 の dbx には次の制限があります。

- 動作中のプロセスに .dbxrc から接続することはできません。このため、.dbxrc ファイルに、コードを実行するコマンドを含めないでください。ただし、別のファイル内にこのようなコマンドを入れておき、dbx source コマンドを使用して、そのファイル内のコマンドを実行することはできます。
- dbx は、-compat=4 オプションでコンパイルされたメンバー関数へのポインタを不適切に復号化します。この問題は、-compat=5 オプションでは発生しません。

---

注 – -compat=4 オプションはこのリリースで削除されました。詳細は、14 ページの「C++ コンパイラ」を参照してください。

---

- SPARC V9 (-m64) システムでは、call コマンドや印刷関数の呼び出しの引数または戻り値として小さな入れ子構造を使用することはできません。
- libC.so.5 または libC.so.4 の古いコピーを使用すると、C++ 例外の領域で dbx の問題が発生することがあります。不正なスタブや未処理の例外に関する警告メッセージが出力されることがあります。

  回避策: 最新の libC.so.5 をすべてのシステムにインストールしてください。
- Fortran ユーザーは、実行時検査をフル活用できるように、-stackvar オプションを使用してコンパイルするようにしてください。
- 一部のプログラムは、-stackvar オプションを使用すると正しく動作しないことがあります。そのような場合には、RTC を使用せずに配列添字検査をオンにする -C コンパイラオプションを試してください。
- マルチスレッドアプリケーションで、fork の追跡が正しくないことがあります。
- マルチスレッドアプリケーションで call コマンドまたは印刷関数呼び出しを使用すると、デッドロック状態が発生する可能性があります。
- あるヘッダーファイルがプリコンパイル済みヘッダー (PCH) コレクションの一部になっていた場合、dbx の修正継続機能を使用してそのファイルを変更しないでください。
- dbx コマンド行インタプリタは、CSI (Code Set Independence) をサポートしない旧バージョンの Korn シェル (ksh) です。マルチバイト文字は、dbx コマンド行に入力すると誤って解釈される場合があります。
- dbx の次の機能は、Linux OS では使用できません。
  - 修正継続
  - パフォーマンスデータの収集
  - 次のイベントのブレークポイント
    - fault
    - lastrites
    - lwp_exit
    - sysin

    - sysout
    - sync
    - throw
  - Java のデバッグ
  - 32 ビットプログラムのデバッグ (-x exec32 オプションを使用して dbx を起動しない場合)。
- Linux プラットフォームでのプログラムのデバッグでは、次の問題が発生する可能性があります。
  - dbx は、exec() の呼び出し時に、Linux プラットフォーム上でフォークされたプロセスを追跡したり新しいプログラムに変更したりできません
  - Linux プラットフォームの場合、Korn シェルの pipe 演算子には制約があります。ターゲットプロセスにアクセスする必要がある dbx コマンドはパイプラインの一部として機能しません。たとえば、次のコマンドではおそらく dbx がハングアップします。

    ```
    where | head -1
    ```

    回避策:

    - Ctrl-C キーを入力して新しい dbx プロンプトを表示します。
    - dbx は多くの情報をキャッシュに格納するので、前述の例の場合、次の一連のコマンドが動作します。

      ```
      where
      where | head –1
      ```
  - プログラムが clone() を使用して独自のスタイルのスレッドを実装している場合、dbx のスレッドサポートではスレッドが正しく識別されません。

    回避策:

    clone() ではなく、libthread.so を使用してください。
  - Linux OS の threads ライブラリは、その内部機構の一部として SIGSTOP シグナルを使っています。通常、dbx はこれらのシグナルをユーザーから隠蔽し、ユーザーがほかのソースからの本物の SIGSTOP シグナルを監視できるようにします。Linux OS はときどき SIGSTOP を予期しない方法で使用しますが、その際に、dbx はシステム生成 SIGSTOP をユーザー生成 SIGSTOP として解釈します。

    回避策:

    ignore コマンドを使用して、SIGSTOP シグナルを捕獲しないよう dbx に指示します。
  - スレッドはときどき終了しますが、Linux OS はその終了のことを dbx に報告しません。

あるスレッドが終了し、その終了が報告されなかった場合、dbx は決して発生しないイベントに対して待機し、新しいプロンプトを表示しません。この状況はおそらく、dbx で cont コマンドを入力したあとで発生しますが、step up コマンド、step コマンド、next コマンド、およびその他のコマンドのあとにも発生する可能性があります。

回避策:

- Ctrl-C キーを入力すると、dbx が待機を停止して新しいプロンプトを表示する場合があります。
- Ctrl-C キーが機能しない場合は、dbx を終了して再起動します。

- dbx は、GNU C および C++ コンパイラで次の機能をサポートしません。
  - VL 配列
  - 例外処理
  - OpenMP
  - RTTI
  - テンプレート定義
  - デフォルト引数
  - using 指令
  - friend
- Oracle Linux 6 上の dbx には、いくつかの問題が存在しています。
  - システムライブラリ内で間接参照シンボルが使用されていると、dbx がブレークポイントを実際の関数にではなくその参照に設定する場合があります。
  - gcc 4.4.4 コンパイラでコンパイルされたコードをデバッグする場合:
    - dbx が可変長配列の出力時にハングアップすることがあります。
    - dbx は、関数のちょうど末尾 (} の行) で停止されると、局所変数を認識できません。
    - dbx は、-D コンパイルオプションを使って定義されたマクロを認識できません。

# パフォーマンスアナライザ

この節では、パフォーマンスアナライザツールの既知の問題について説明します。

- 「呼び出し元-呼び出し先」タブで、切り捨てられたメトリック値が表示される場合があります。
- SPARC T4 プロセッサで icache のストール時間が過大評価される場合があります。

- OMP タスクと OMP 並列領域の OpenMP Wait メトリックを足し合わせても、<Total> の OpenMP Wait メトリックに等しくなりませんが、これは、最初の並列領域エントリの前に到着したプロファイリングパケットはすべて、どのタスクや領域の OMP Wait 時間としてもカウントされませんが、合計にはカウントされるからです。
- 実験の追加や削除は必ずしも正しく処理されません。回避方法は、最初にすべての実験をロードし、削除対象の実験をフィルタリングして除外することです。
- タブの移動が必ずしも正しく動作しません。
- タブ内での項目の複数選択が必ずしも正しく動作しません。
- アナライザがときどき SummaryDisp.updateSummary (SummaryDisp.java:249) でハングアップします
- 「ソース/逆アセンブリ」タブで選択を行ったときに、「概要」タブが更新されません。
- ゼロでないメトリックが存在していても、「ソース」タブのマージンの強調表示が表示されない場合があります。
- 共有オブジェクトの「表示/非表示/API のみ」機能が、フィルタリングと組み合わせた場合に必ずしも正しく動作しません。この問題は、er_print を使用して実験データを表示する場合にも発生します。
- 「フィルタの管理」ダイアログボックスの「一般」タブで「適用」ボタンを押しても、変更が必ずしも適用されません。「適用」を複数回押すとうまくいく可能性があります。より信頼性の高い回避方法は、「タイムライン」、「スレッド」、および「CPU」データタブのコンテキストメニューからフィルタにアクセスすることです。
- 「タイムライン」タブで、スレッドや CPU が数値順に表示されないことがあります。

## collect ユーティリティー

この節では、collect ユーティリティーやデータ収集の既知の問題について説明します。

- 一部の最適化されたコードで、Sandy Bridge マシン上でのスタック巻き戻しが正しくない場合があります。
- JDK 1.7 でのロックアルゴリズムの実装変更のため、Java プログラムの同期トレース時に、すべての Java 同期イベントが二重カウントされます。回避方法は、報告された値を 2 で割ることです。
- Linux システム上のマルチスレッドアプリケーションのクロックプロファイリングで、スレッドに関して不正確なデータが報告されます。プロファイルシグナルは、必ずしもカーネルから各スレッドに指定された間隔で配信されるとは限りません。シグナルが間違ったスレッドに配信される場合もあります。可能であれ

ば、より正確なスレッド単位の結果が得られるように、cycle カウンタによるハードウェアカウンタプロファイリングを使用してください。

- 異なるクロック周波数で動作する複数の CPU を持つシステムのクロックプロファイリングでは、1 つの CPU のクロックレートに基づいてイベントが生成されるため、ほかの CPU 上のイベントが過大または過小にカウントされる可能性があります。

## スレッドアナライザ

この節では、スレッドアナライザの既知の問題について説明します。

次のすべてが真であれば、スレッドアナライザの実行時エラーが発生する可能性があります。

- データ競合を検出するために discover で計測されたバイナリ上で、collect が実行される
- その実行が、UltraSPARC T2 のような、ハードウェア機能フィルタをサポートする SPARC プロセッサを備えたマシン上で行われる
- マシン上で Solaris 10 Update 9 以前のアップデートが実行されている

問題を回避するには、collect を実行する前に、環境変数 LD_NOAUXFLTR=yes を設定してフィルタライブラリのロードをスキップします。フィルタを使用しないことにより、パフォーマンスが若干低下する可能性があります。

## er_kernel ユーティリティー

この節では、er_kernel ユーティリティーの既知の問題について説明します。

- er_kernel はときどき、実行元のマシンをクラッシュさせる可能性があります。この問題は、UltraSPARC T1、T2、および T2+ チップ上で動作する Oracle Solaris 10 でのみ発生するもので、その原因はハイパーバイザのバグです。
- er_kernel プロファイルはときどき、CPU 状態の属性を間違って決定し、ユーザーデータを正しく報告しません。
- 1 つ以上の CPU が er_kernel イベントの生成を 30 秒以上停止することがあります。
- ユーザープロセスの DTrace スタック巻き戻しで 1 つのフレームが省略されることがありますが、それは特に、リーフ関数がその関数プロローグまたはエピローグ内で実行中の場合に発生します。
- 異なるクロック周波数で動作する複数の CPU を持つシステムの er_kernel プロファイリングでは、1 つの CPU のクロックレートに基づいてイベントが生成されるため、ほかの CPU 上のイベントが過大カウントまたは過小カウントされる可能性があります。

# IDE

この節では、IDE の既知の問題について説明します。

- バージョン管理フレームワークがフルリモートモードで動作しません。(195121)

  バージョン管理フレームワークはしばしば `java.io.File` と関係しながら動作するため、リモートファイルオブジェクトを操作できるプラグインを作成することは不可能です。

  回避方法:リモートホスト上のバージョン管理ツールを ssh 経由で直接使用します。

- GDB 7.2 が使用されている一部のプラットフォームで、「ステップオーバー」が「継続」として動作することがあります。 (200196)

  回避方法:以前の GDB バージョンを試すか、プロジェクトプロパティーの「実行」セクションの「コンソールタイプ」を、「内部ターミナル」から別のオプションに変更します。

- メモリーアクセスエラーツールがリモートプロジェクトで動作しません。(7109562)

  リモートホスト上にプロジェクトを作成したあと、そのプロジェクトをメモリー分析用に計測して実行すると、`can't execute: discover: No such file or directory` というエラーメッセージが表示されます。

- 「デバッガコンソール」タブをクリックしても、デバッガコマンドのプロンプトにフォーカスが設定されません。 (7102076)

  回避方法:コマンドをプロンプトに入力できるように、タブ内でもう一度クリックしてフォーカスを設定します。

- バイナリから新しく作成されたフルリモートプロジェクトが「プロジェクト」タブに表示されません。 (7110094)

  IDE のデスクトップ配布を実行し、リモートホスト上の既存のバイナリからプロジェクトを作成した場合、その新しく作成されたプロジェクトが「プロジェクト」タブに表示されません。

  回避方法:「ファイル」 > 「リモート C/C++ プロジェクトを開く」を選択し、開くプロジェクトを選択します。

- SPARC プラットフォーム上のバイナリが認識されないため、バイナリからフルリモートプロジェクトを作成できません。 (7109551)

  IDE のデスクトップ配布を実行し、SPARC プラットフォームであるリモートホスト上の既存のバイナリからプロジェクトを作成し、「ファイル」 > 「リモート C/C++ プロジェクトを作成」を選択したときに、バイナリが認識されません。ファイルチューザのフィルタが「すべてのバイナリファイル」に設定されている場合、バイナリが表示されません。フィルタが「すべてのファイル」に設定されている場合は、バイナリを選択できるものの、「バイナリファイルが見つかりません」というメッセージが表示されます。

# dmake

ここでは、これまでにわかっている dmake ソフトウェアの問題点とその回避策について説明します。

分散モードで dmake を使用した場合に何か問題が発生する場合は、次の点を確認してください。

1. $HOME 環境変数がアクセス可能なディレクトリに設定されているか

   ```
   % ls -la $HOME
   ```

2. ファイル $HOME/.dmakerc が存在するか、このファイルの読み取りが可能か、このファイルの情報が正しいか

   ```
   % cat $HOME/.dmakerc
   ```

3. $HOME/.dmakerc ファイルに示されているすべてのホストが稼働しているか (/usr/sbin/ping コマンドを使用して各ホストをチェック)

   ```
   % /usr/sbin/ping $HOST
   ```

   % /$$HOST には、$HOME/.dmakerc ファイルでホストとして示されているシステムの名前を指定してください。

4. dmake バイナリのパスが正しいか (dmake、rxm、および rxs コマンドを使用)

   ```
   % which dmake
   % which rxm
   % which rxs
   ```

5. 各ホストへのリモートログイン (rsh または ssh) がパスワードなしで動作し、各リモートログインの所要時間が許容範囲 (2 秒未満) である。

   ```
   % time rsh $HOST uname -a
   ```

6. 各ホスト上にファイル /etc/opt/SPROdmake/dmake.conf が存在するか。このファイル内の情報は正しいか。このファイルが存在しない場合は、dmake はこのシステムでジョブを 1 つだけ分散します。

   ```
   % rsh $HOST cat /etc/opt/SPROdmake/dmake.conf
   ```

7. 各ホストの dmake バイナリのパスが正しいか

   ```
   % rsh $HOST ‘which dmake‘
   % rsh $HOST ‘which rxm‘
   % rsh $HOST ‘which rxs‘
   ```

8. 各ホストから構築領域を利用できるか (rwx)

   ```
   % cd $BUILD
   % rm $HOST.check.tmp
   % echo "Build area is available from host $HOST" > $HOST.check.tmp
   % rsh $HOST cat $BUILD/$HOST.check.tmp
   ```

   $BUILD には、構築領域のフルパスを指定してください。

9. その $HOME は各ホストから使用可能か

```
% cd $HOME
% rm $HOST.check.tmp
% echo "HOME is available from host $HOST" > $HOST.check.tmp
% rsh $HOST cat $HOME/$HOST.check.tmp
```

### dmake の制限事項

次の要件を満たしていれば、どのマシンも構築サーバーとして使用できます。

- dmake ホスト (構築プロセスを開始するために使用するマシン) から、構築サーバー上でコマンドをリモート実行するためのパスワードを要求されることなく、rsh または ssh を使用できる必要があります。
- dmake ソフトウェアがインストールされている bin ディレクトリに構築サーバーからアクセスできる必要があります。デフォルトでは、dmake は構築サーバー上の dmake 実行可能ファイルへの論理パスが dmake ホスト上の実行可能ファイルと同じものであると仮定します。この仮定を無効にするには、実行時構成ファイルのホストエントリの属性としてパス名を指定します。
- ホスト上に /etc/opt/SPROdmake/dmake.conf ファイルが存在していて、読み取り可能であり、適切な情報が含まれている。このファイルが存在しない場合は、dmake はこのシステムでジョブを 1 つだけ分散します。

## インストール

- –extract-installation-data オプションを指定して非 GUI インストーラを実行すると、ユーザーの読めるエラーメッセージが一切表示されずに処理が失敗する可能性があります。
- インストールディレクトリ内で register_solstudio ユーティリティーを実行すると、登録ページの生成とブラウザ内でのオープンが行われない場合があります。

  回避策:

  1. register_solstudio ユーティリティーを *installation_directory*/bin から *installation_directory*/bin/condev/bin にコピーします。
  2. *installation_directory*/bin/register_solstudio を、*installation_directory*/bin/condev/bin/register_solstudio へのシンボリックリンクで置き換えます。
  3. register_solstudio ユーティリティーを実行すると、登録ページが生成され、そのページがブラウザで開かれます。

# 索引

## ま



## ら