Oracle® Solaris Studio 12.2 リリースの新機能

Part No: 821–2504
2010 年 9 月

110412@25097

目次

はじめに

このガイドでは、Oracle Solaris Studio 12.2 リリースの新機能および変更された機能について説明します。

Sun Studio のコンパイラとツールから成るソフトウェアの一式を Oracle Solaris Studio と改称しました。このリリースより前のリリースでは、Sun Studio という名前が引き続き使用されます。

サポートされるプラットフォーム

Oracle Solaris Studio のこのリリースは、SPARC または x86 ファミリのプロセッサアーキテクチャー (UltraSPARC、SPARC64、AMD64、Pentium、Xeon EM64T) を使用するシステムをサポートしています。使用している Solaris オペレーティングシステムのバージョンに対するシステムのサポート状況は、ハードウェア互換性リスト (http://www.sun.com/bigadmin/hcl) をご参照ください。ここには、すべてのプラットフォームごとの実装の違いについて説明されています。

このドキュメントでは、x86 関連の用語は次のものを指します。

- 「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、AMD 64 または EM64T システムで、特定の 64 ビット情報を指します。
- 「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。

サポートされるシステムについては、ハードウェアの互換性に関するリストを参照してください。

Solaris Studio マニュアルへのアクセス方法

マニュアルには、次の場所からアクセスできます。

- マニュアルは、次に示すマニュアル索引のページからアクセスできます。http://www.oracle.com/technetwork/server-storage/solarisstudio/documentation。

- IDE のすべてのコンポーネント、パフォーマンスアナライザ、dbxtool、および DLight のオンラインヘルプは、IDE 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。

アクセシブルな製品マニュアル

マニュアルは、技術的な補足をすることで、ご不自由なユーザーの方々にとって読みやすい形式のマニュアルを提供しております。アクセシブルなマニュアルは次の表に示す場所から参照することができます。

マニュアルの種類	アクセシブルな形式と格納場所
マニュアル	HTML 形式。docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択
『Oracle Solaris Studio 12.2 リリースの新機能』(以前のコンポーネントの README ファイル)	HTML 形式。docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択
マニュアルページ	Oracle Solaris 端末で `man` コマンドを使用して表示します。
オンラインヘルプ	HTML 形式。IDE、dbxtool、DLight、およびパフォーマンスアナライザの「ヘルプ」メニュー、「ヘルプ」ボタン、および F1 キーを使用して表示
リリースノート	HTML 形式。docs.sun.com にある Oracle Solaris Studio 12.2 Collection - Japanese から選択

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

次の頻繁に更新されるリソースを検索するには、http://www.oracle.com/technetwork/server-storage/solarisstudio を参照してください。

- リソースは頻繁に更新されます。
- ソフトウェアのマニュアル、およびソフトウェアとともにインストールされる一連のマニュアル
- Oracle Solaris Studio ツールを使用する開発タスク全体を順を追って説明するチュートリアル
- サポートレベルに関する情報
- http://forums.sun.com/category.jspa?categoryID=113 のユーザーフォーラム

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:
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。

表 P–1 表記上の規則 (続き)

字体または記号	意味	例
\	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'`

コード例は次のように表示されます。

- 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]`

- Bourne シェルおよび Korn シェル

  `$ command y|n [filename]`

- Bourne シェルおよび Korn 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。

ドキュメント、サポート、およびトレーニング

追加のリソースについては、次の Web サイトを参照してください。

- ドキュメント (http://docs.sun.com)
- サポート (http://www.oracle.com/us/support/systems/index.html)
- トレーニング (http://education.oracle.com) – 左のナビゲーションバーにある Sun リンクをクリックします。

ご意見の送付先

マニュアルの品質や使いやすさに関するご意見やご提案をお待ちしています。間違いやその他の改善すべき箇所がありましたら、http://docs.sun.com で「Feedback」をクリックしてお知らせください。ドキュメント名とドキュメントの Part No.、および、可能な場合は章、節、ページ番号を記載してください。返答が必要な場合はお知らせください。

Oracle 技術ネットワーク (http://www.oracle.com/technetwork/index.html) では、Oracle ソフトウェアに関するさまざまなリソースを提供しています。

- 技術上の問題やソリューションについては、ディスカッションフォーラム (http://forums.oracle.com) を参照してください。
- 実践的なステップ・バイ・ステップのチュートリアルについては、Oracle By Example (http://www.oracle.com/technology/obe/start/index.html) を参照してください。
- サンプルコードのダウンロードについては、サンプルコード (http://www.oracle.com/technology/sample_code/index.html) を参照してください。

◆ ◆ ◆ 第 1 章

1 Oracle Solaris Studio 12.2 リリースの概要

このリリースの Oracle Solaris Studio では、この新機能ガイドで概説されているとおり、新機能や変更点が数多く盛り込まれています。このガイドは、以前のリリースでは Sun Developer Network ポータルにおいて提供されていたコンポーネントの README ファイルに代わるものです。

もっとも重要な変更点は名前であり、Sun Studio が Oracle Solaris Studio になっています。

Oracle Solaris Studio とは

Oracle Solaris Studio は、Solaris および Linux オペレーティング環境でのアプリケーション開発用の一連のツールで構成されています。

- 共有メモリー並列化のための OpenMP 3.0 API をネイティブに実装する、C、C++、および Fortran 用の高性能最適化コンパイラと実行時ライブラリ (`cc`、`CC`、および `f95`)
- スクリプト化が可能でマルチスレッドに対応する対話型の dbx デバッガおよび dbxtool デバッガ GUI
- メモリーリークおよびコードカバレージを検出するための新しいツールである `discover` と `uncover`
- 高度に最適化されたマルチスレッドの 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のボトルネックと非効率性を検出するパフォーマンスアナライザ、および 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ための 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

- 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、パフォーマンス解析ツール、デバッグツール、IDE、およびその他の関連ツールに関する各章で構成されています。既知の問題、制限事項、および回避策の章では、Oracle Solaris Studio 12.2 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の情報については、Oracle 技術ネットワーク の Solaris Studio ポータルを参照してください。

◆ ◆ ◆ 第 2 章

2 コンパイラ

この章では、Oracle Solaris Studio のこのリリースのコンパイラ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。

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。

- コンパイラは、SPARC-V9 ISA の SPARC VIS3 バージョンをサポートします。`-xarch=sparcvis3` オプションを指定してコンパイルすると、コンパイラは、SPARC-V9 命令セットの命令に加えて、Visual Instruction Set (VIS) バージョン 1.0 を含む UltraSPARC 拡張機能、Visual Instruction Set (VIS) バージョン 2.0 の積和演算 (FMA) 命令を含む UltraSPARC-III 拡張機能、および Visual Instruction Set (VIS) バージョン 3.0 を使用できます。
- x86 ベースのシステムでは、`-xvector` オプションのデフォルトが `-xvector=simd` に変更されました。最適化レベル 3 以上でメリットがある場合、x86 ベースのシステムではストリーミング拡張機能がデフォルトで使用されます。サブオプション `no%simd` を使用することで、それを無効にできます。SPARC ベースのシステムでは、デフォルトは `-xvector=%none` です。
- AMD SSE4a 命令セットのサポートを使用できるようになりました。`-xarch=amdsse4a` オプションによりコンパイルします。
- マニュアルページで、`-xtarget` の値 `ultra3`、`ultra3i`、`ultra3cu`、`ultra4`、および `ultra4plus` が正しく展開されるように更新されました。
- 新しい `-traceback` オプションを使用すると、サーバーエラーが発生した場合に実行可能ファイルはスタックトレースを出力できます。このオプションを指定すると、実行可能ファイルは、一連のシグナルをトラップして、実行の前にスタックトレースとコアダンプを出力します。複数のスレッドがシグナルを生成する場合、最初のスレッドに対するスタックトレースだけが生成されます。トレース

バックを使用するには、f95、cc、または CC とプログラムをリンクするときに、-traceback オプションを追加します。便宜上、このオプションはコンパイル時にも受け付けられますが、無視されます。-traceback オプションと -G オプションを使用して共有ライブラリを作成すると、エラーが発生します。-traceback オプションについての詳細は、コンパイラのマニュアルページを参照してください。

- -mt オプションは、-mt=yes または -mt=no に変更されました。-mt=yes オプションにより、ライブラリが適切な順序でリンクされることが保障されます。詳細は、コンパイラのマニュアルページを参照してください。
- 新しいプラグマが C および C++ に追加されました。詳細は、コンパイラのユーザーガイドを参照してください。
- #warning コンパイラディレクティブ (C および C++) は、ディレクティブ内のテキストを警告として発行し、コンパイルを続行します。
- ヘッダーファイル mbarrier.h (C および C++) が使用できるようになりました。このヘッダーファイルでは、SPARC および x86 プロセッサでのマルチスレッドコード用のさまざまなメモリーバリアー組み込み関数が定義されています。詳細は、コンパイラのユーザーガイドを参照してください。
- -xprofile=tcov[:*prof_dir*] オプションは、省略可能なプロファイルディレクトリのパス名引数を受け付けます。プロファイルディレクトリのパス名を指定した場合は、コンパイル済みのプログラムは、tcov(1) または -xprofile=use:*prof_dir* を指定したフィードバックコンパイルで使用できるデータを生成します。詳細は、コンパイラのユーザーガイドを参照してください。
- このリリースでは、-xMD および -xMMD オプション (C/C++) によって書き込まれる依存関係ファイルは、以前の既存ファイルを上書きします。ファイル名は、-o *filename* (指定されている場合)、入力ソースファイル名に .d 接尾辞を追加したもの、または -xMF オプションで指定されているファイル名が使用されます。-o *filename* または -xMF *filename* が -xMD または -xMMD オプションとともに指定されている場合は、単一のソースファイルのみが受け付けられます。この方法で複数のソースファイルをコンパイルするとエラーが発生します。

C コンパイラ

次に、C コンパイラのバージョン 5.11 のこのリリースでの新機能および変更された機能を一覧します。詳細は、『Oracle Solaris Studio 12.2: 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。

- C コンパイラに対する変更により、64 ビットモードの SPARC で複合型を含む struct が受け渡される方法が修正されました。以前は、このような struct 値は誤ったレジスタで受け渡される場合があり、gcc コンパイラによって作成されたバイナリと互換性のないバイナリが作成されました。この変更によって Solaris Studio の C コンパイラで実装されている既存の ABI の要素が影響を受けるので、アプリケーションのソースファイルが複合型のフィールドを含む struct を使

用している場合は、アプリケーションのソースベース全体を再コンパイルして、正しくない応答が返される可能性を回避する必要があります。32 ビットの SPARC プロセッサ、および 32 ビットまたは 64 ビットの x86 プロセッサに関するコンパイルには、この変更による影響はありません。

C++ コンパイラ

次に、C++ コンパイラのバージョン 5.11 のこのリリースでの新機能および変更された機能を一覧します。詳細は、『Oracle Solaris Studio 12.2: 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。

- `-g` オプションとともに `-O` または `-xO` オプションを指定し、`+` オプションを指定しないと、インライン化が生成されます。次に例を示します。
  - `CC -g foo.cc` は、インライン化のないデバッグ可能な `a.out` を生成します
  - `CC -g -O foo.cc` は、インライン化のあるデバッグ可能な `a.out` を生成します
  - `CC -g0 foo.cc` は、インライン化のあるデバッグ可能な `a.out` を生成します
- C++ のオプション `-xalias_level=compatible` は、プログラムが C++ 標準の要件を満たすことを表明します。
- Oracle Solaris にインストールされている Apache C++ ライブラリのサポートが追加されました。
- `-compat=g` オプションは、特定の `gcc` 互換性を追加します。
- `-features=[no%] rvalueref` オプションは、非 `const` 参照のコンパイラの処理を一時または右辺値に戻します。

Fortran コンパイラ

次に、Fortran コンパイラのバージョン 8.5 のこのリリースでの新機能および変更された機能を一覧します。詳細は、『Oracle Solaris Studio 12.2: 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。

- 新しい **`-xkeepframe`**[**`=`**[**`%all`**,**`%none`**, *name*, **`no%`** *name*] オプションは、名前付き関数のスタック関係の最適化を禁止します。**`%all`** は、すべてのコードに対するスタック関係の最適化を禁止します。**`%none`** は、すべてのコードに対するスタック関係の最適化を許可します。デフォルトは **`-xkeepframe=%none`** です。
- F2003 Fortran のその他の機能が実装されました。
- **`IVDEP`** 指令は、最適化の目的でループ内で検索する配列参照の一部または全部のループによる依存関係を無視するよう、コンパイラに指示します。これによりコンパイラは、この指令がなければ不可能だったさまざまなループ最適化を実行できます。**`-xivdep`** オプションを使用すると、**`IVDEP`** 指令を無効にしたり、指令の解釈方法を指定したりできます。

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る OpenMP 3.0 共有メモリー API の新機能および変更された機能を一覧します。詳細は、『Oracle Solaris Studio 12.2: OpenMP API ユーザーガイド』を参照してください。

- dbx デバッガでの OpenMP デバッグのサポート dbx に対して次の拡張が行われました。
  - OpenMP の領域、タスク、およびスレッドセットについての情報を表示するための新しいコマンド。
  - `print –s`、`thread –info`、`whatis`、および `where` の各コマンドに対する拡張。
  - 新しい OpenMP 同期イベント。
- 自動スコープ宣言がタスク領域に拡張されました。この機能により、プログラマは並列領域およびタスク領域内の変数のスコープを明示的に決定する必要がなくなります。コンパイラが、コードを分析してスマートルールを適用することで、変数のスコープを決定します。
- 新しい `SUNW_MP_WAIT_POLICY` 環境変数は、プログラムでのスレッドの待機動作を改善し、処理を待機 (アイドル) するスレッド、バリアーで待機するスレッド、および `taskwait` で待機するスレッドの動作をプログラマが細かく制御できるようにします。
- `SUNW_MP_WARN` OpenMP 環境変数に新しい機能が追加されました。OpenMP 実行時ライブラリによって発行される警告メッセージの制御に加えて、`SUNW_MP_WARN` を `TRUE` に設定すると、ユーザーによって明示的に設定された環境変数やライブラリによってデフォルトで設定される環境変数など、すべての環境変数の設定が実行時ライブラリによって情報提供用に出力されます。
- Oracle Solaris プラットフォームでの `SUNW_MP_PROCBIND` 環境変数によって制御される動作が変わりました。`SUNW_MP_PROCBIND` を `TRUE` に設定すると、メインスレッドはバインド時に実行しているプロセッサにバインドされます。バインド時とは、並列領域が最初に発生するとき、または `omp_set_num_threads()` など OpenMP の実行時ルーチンを最初に呼び出すときを指します。スレーブスレッドは、メインスレッドがバインドされているプロセッサから始めてラウンドロビン方式でバインドされます。
- OpenMP プログラムでのデータの競合およびデッドロックを検出するには、スレッドアナライザツールを使用します。このリリースでは、スレッドアナライザの機能が、再コンパイルを行わなくてもバイナリ内のデータの競合を検出できるように拡張されました。詳細は、『Oracle Solaris Studio 12.2: スレッドアナライザユーザーズガイド』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` は廃止されました。ライブラリ `libm9x.so.0`、`libmvec.a`、および `libmvec_mt.a` も廃止されて、このリリースでは削除されています。

Sun Performance Library

Sun Performance Library について

このリリースの Sun Performance Library は、Solaris オペレーティングシステムのバージョン 10 で使用できます。一部の Linux オペレーティング環境でも使用できます。

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。Sun Performance Library には次の標準ライブラリが含まれます。

- 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。

Sun Performance Library には、次の追加ルーチンが含まれています。

- 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
- 区間 BLAS ルーチン

互換性

Sun Performance Library 内の LAPACK 3.1.1 ルーチンは、1.x、2.0、3.0 を含む従来のバージョンの LAPACK のユーザールーチン、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 の内部の変更により、内部ルーチンとの互換性を保証できません。

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。

マニュアル類

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
- 区間 BLAS のマニュアルページ (セクション 3pi) - 各区間 BLAS ルーチンに関する説明
- 『Oracle Solaris Studio 12.2: Sun Performance Library User's Guide』では次の項目についての説明とその例が示されています。
  - Sun Performance Library のルーチンの使用方法
  - Fortran インタフェースおよび C インタフェースの使用方法
  - 最適化オプションおよび並列化オプションの使用方法

- SPSOLVE および SuperLU スパースソルバーパッケージの使用方法
- FFT ルーチンの使用方法

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

- このリリースでは、Fortran および C コンパイラに関する Sun Performance Library とのリンクが変更されています。Fortran、C、C++ はすべて、`-xlic_lib=sunperf` オプションの代わりに `-library=sunperf` オプションを使用するようになっています。静的にリンクする場合は、`-library=sunperf` オプションのあとに `-staticlib=sunperf` オプションを追加します。
- `libsuniperf` (IBLAS) は「廃止または互換性がなくなる可能性あり」に分類されており、このリリースでは Oracle Solaris Studio から削除されています。

前のリリースの新機能および変更された機能

Sun Studio 12 Update 1 リリースでは、Sun Performance Library に次の機能が導入されました。

- Sun Performance Library には現在、ScaLAPACK 1.8.0 高性能クラスタライブラリが含まれています。このライブラリは、OpenMPI 1.3 リリースをベースとする Sun HPC ClusterTools 8.1 とともに動作します。リファレンス実装とドキュメントは、`http://www.netlib.org/scalapack/` で説明されています。
- 新しいカスタムライブラリツールには、Sun Performance Library のスケールダウンバージョンを作成するオプションがあります。カスタムライブラリツール `gen_custom` は、アーカイブライブラリからルーチンだけを抽出し、それらを再結合してカスタムライブラリにします。こうすることで、Sun Performance Library のような大型ライブラリの占有サイズを、ユーザの必要なルーチンの分量にまで削減することができます。詳細については、`gen_custom`(3p) マニュアルページを参照ください。
- BLAS、LAPACK、および FFT のルーチンに、多数のパフォーマンス向上が追加されました。
- Intel(R) CoreTM i7 (Nehalem) および AMD のクワッドコア OpteronTM (Shanghai) CPU のサポートが使用できます。これらのライブラリにリンクするには、次のオプションを使用します。

  `-m64 -xlic_lib=sunperf` (C および Fortran)

`-m64 -library=sunperf` (C++)

- Fujitsu SPARC64-VII(R) CPU のサポートが使用できます。このバージョンの Sun Performance Library は、実現可能な最高のパフォーマンスを得るために、浮動小数点の積和演算 (FMA) 命令を使用します。これらのライブラリにリンクするには、次のオプションを使用します。

  `-xtarget=sparc64vii -fma=fused -xlic_lib=sunperf` (C および Fortran)

  `-xtarget=sparc64vii -fma=fused -library=sunperf` (C++)
- SPARC64-VI および SPARC64-VII に対する ZGEMM の向上。
- LAPACK ルーチンは、LAPACK 3.1.1 の最新の仕様に適合するように更新されています。
- Woodcrest CPU のサポートが使用できます。
- SPARC64-VI CPU のサポートが使用できます。

x86 ベースシステム

- SUSE Linux Enterprise Server 9 または Redhat Enterprise Linux 4 オペレーティング環境の 32 ビットおよび 64 ビットシステムでライブラリが使用できる。
- 64 ビット整数パラメータのルーチンが使用できる。つまり、すべてのバージョンの Sun Performance Library に `DAXPY()` および `DAXPY_64()` があります。
- スパースソルバーパッケージ SuperLU のシリアルバージョンが使用でき、C ドライバから、またはライブラリ内の既存の Fortran ベースのスパースソルバーを介して呼び出しすることができる。
- この時点では、4 倍精度ルーチン (`dqdoti`、`dqdota`) は使用できない。
- SSE2 対応システムおよび X86 以上のシステム上の Solaris OS および Linux OS で区間 BLAS ルーチンが使用できる。

SPARC プロセッサ

- UltraSPARC IV+ プロセッサおよび UltraSPARC IV プロセッサ用の BLAS と FFT の改良が組み込まれている。
- SPARC64VI CPU のサポートが使用できる。このバージョンの Sun Performance Library は、SPARC64VI CPU で可能な最高のパフォーマンスを得るために、浮動小数点の積和演算 (FMA) 命令を使用します。このライブラリとリンクするには、`-xtarget=sparcfmaf` フラグを指定してコンパイルおよびリンクします。
- スパースソルバーパッケージ SuperLU のシリアルバージョンが使用でき、C ドライバから、またはライブラリ内の既存の Fortran ベースの SPSOLVE スパースソルバーを介して呼び出しすることができる。

◆ ◆ ◆ 第 4 章

4 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

ここでは、このリリースの Solaris Studio パフォーマンスアナライザおよび関連ツールの新機能および変更された機能について説明します。詳細は、『Oracle Solaris Studio 12.2: パフォーマンスアナライザ』『Oracle Solaris Studio 12.2: パフォーマンスアナライザ』マニュアルを参照してください。

実験形式の変更

実験の形式が拡張されましたが、現在、バージョン番号は変更されていません (10.1)。

ツールは、Oracle Solaris Studio 12.2 の FCS バージョンで作成された実験、および Studio 12 Update 1 と Studio 12 の FCS およびパッチ適用バージョンで作成された実験を読み取ることができます。

Sun Studio 12 より前のバージョンで作成された実験は、Oracle Solaris Studio 12.2 のツールで読み取ることはできません。

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。

新しい「呼び出しツリー」タブ

新しい「呼び出しツリー」タブには、プログラムの動的なコールグラフがツリーとして表示され、ノードとして示される各関数呼び出しを展開したり折りたたんだりできます。関数ノードを展開すると、その関数によって行われたすべての関数呼び出しに加えて、それらの関数呼び出しのパフォーマンス測定基準が表示されます。ノードを選択すると、右側の「サマリー」タブにその関数呼び出しと呼び出し先に関する測定基準が表示されます。属性付き測定基準に対して示されるパーセンテージは、プログラムの全測定基準のパーセンテージです。

もっとも多くの時間を消費するブランチを簡単に見つけるには、いずれかのノードを右クリックして、「もっとも活動的なブランチを展開」を選択します。

「呼び出し側-呼び出し先」タブの拡張

呼び出しスタックに呼び出し側と呼び出し先を追加することで、一度に 1 つの呼び出しの呼び出しスタックフラグメントを中央の「スタックフラグメント」パネルに作成できます。呼び出し側はそのフラグメントを呼び出している関数であり、呼び出し先はそのフラグメントから呼び出されている関数です。次の機能が含まれます。

- スタックフラグメントの関数を追加および削除すると、測定基準がフラグメント全体について計算されて、フラグメントの最後の関数の隣に表示されます。
- 呼び出し側を右クリックするとスタックフラグメントの先頭に関数を追加でき、呼び出し先を右クリックすると末尾に関数を追加できます。また、「スタックフラグメント」パネルの上にあるボタンを使用して、呼び出しスタックフラグメントを操作することもできます。
- 「スタックフラグメント」パネルの上にある「戻る」ボタンと「進む」ボタンを使用して、呼び出しスタックフラグメントに対する変更の履歴内を移動できます。
- コンテキスト (右クリック) メニューを使用して、「呼び出し側-呼び出し先」タブ内のデータにフィルタを適用できます。

新しい実験比較機能

パフォーマンスアナライザでは、同じ実行可能ファイルで収集された実験を比較できるようになりました。この機能は部分的にのみ実装されており、今後のリリースで変更される可能性があります。現在のリリースでは、実験の比較は次のように動作します。

- 複数の実験または実験グループを開くと、デフォルトでデータが集計されます。
- `compare on` を `.er.rc` ファイルに追加し、パフォーマンスアナライザで複数の実験または実験グループを開くと、データが比較モードで表示されます。

- 比較モードでは、実験またはグループのデータが隣り合った列に表示され、追加されるヘッダー行には実験またはグループの名前が表示されます。実験またはグループを区別できるように列には色で影が付けられます。
- 実験の比較をサポートするタブは、「関数」、「呼び出し側-呼び出し先」、「ソース」、「逆アセンブリ」、「行」、および「PC」です。いずれかのタブのコンテキストメニューから比較モードを無効および有効にできます。
- また、アナライザの「データ表示設定」ダイアログの「形式」タブにある「実験の比較」オプションを使用して、比較モードを有効および無効にすることもできます。

その他の拡張機能

- 「ソース」タブの強調表示では、ホット (CPU 使用率がもっとも高い) 行がオレンジ色で表示され、ゼロ以外の測定基準の行が黄色で表示されます。
- 「ソース」タブのコンテキストメニューを使用すると、次または前のホット行または非ゼロ測定基準行に移動できます。
- 「印刷」メニューを使用して、タイムライン、MPI タイムライン、および MPI グラフの JPG ファイルを作成できます。
- HotSpot でコンパイルされたコードのソースと逆アセンブルは、記録されている場合はよりよいマッピングを生成します。

`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。

- 呼び出し側-呼び出し先リストを制御するための新しいコマンドが、呼び出しスタックの作成をサポートするようになりました。`er_print` の新しいサブコマンド `cprepend`、`cappend`、`crmfirst`、および `crmlast` は、作成している呼び出しスタックフラグメントの関数を追加または削除します。各コマンドのあとで、現在のフラグメントの呼び出し側-呼び出し先データが書き込まれます。
- 新しい `calltree` コマンドは、すべての関数の階層的な測定基準を示す、ターゲットの動的なコールグラフを出力します。
- 新しい `describe` コマンドは、実験から記録されたデータを記述して、フィルタに使用できるトークンを出力します。
- HotSpot でコンパイルされたコードのソースと逆アセンブルは、記録されている場合はよりよいマッピングを生成します。
- `er_print` コマンドを使用して、同じ実行可能ファイルで収集された実験を比較できるようになりました。この機能は部分的にのみ実装されており、今後のリリースで変更される可能性があります。現在のリリースでは、実験の比較は次のように動作します。

- 複数の実験または実験グループに対して er_print を呼び出すと、データが集計されます。
- compare on を .er.rc ファイルに追加し、複数の実験または実験グループに対して er_print を実行すると、データが比較モードで表示されます。
- 比較モードでは、実験またはグループのデータが関数リスト、呼び出し側-呼び出し先リスト、およびソースと逆アセンブリリストの隣り合った列に表示されます。列は実験またはグループの読み込みの順序に表示され、追加のヘッダー行に実験またはグループの名前が表示されます。比較モードを有効および無効にするには、compare コマンドを使用します。

新しいデータ収集機能

collect 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。

- 従う子孫に対するデフォルトの設定が、-F on に変更されました。
- Sun HPC ClusterTools (現在は Oracle Message Passing Toolkit と呼ばれています) の任意のリリースでの MPI 実験を、-M OMPT または -M CT で指定できます。
- MPI 実験は、デフォルトで子孫プロセスにも従うようになりました。
- MPI トレース実験に対する後処理が改善されています。
- Oracle Enterprise Linux に対するハードウェアカウンタプロファイリングのサポートが追加されています。
- ハードウェアカウンタ別名が改善され、次のプロセッサでのハードウェアカウンタプロファイリングのサポートが追加されています。
  - SPARC64 VI および VII
  - Intel Core i7: Family 6、Models 30、31、37、44、および 46 (Nehalem EP および EX を含む)
  - AMD Family 10h および 11h
- スクリプトのプロファイリングの実験的サポートが実装されており、今後のリリースで変更される可能性があります。スクリプトをプロファイリングするには、環境変数 SP_COLLECTOR_SKIP_CHECKEXEC を設定し、スクリプト名を collect に渡します。
- Java プロファイリングが、HotSpot でコンパイルされたコードのソース行マッピングにさらに詳細な情報を提供するように拡張されています。Java プロファイリング拡張機能は、JDK 1.6u20 以降の JDK 1.6 更新、および JDK 1.7.0-ea-b85 以降の JDK 1.7 更新についてサポートされています。
- 実験に対するデフォルトのサイズ制限が削除されています。-L オプションを使用してサイズ制限を設定できます。

dbx collector の新機能

dbx デバッガの collector サブコマンドは次のように変更されています。

- ハードウェアカウンタ別名が改善され、次のプロセッサでのハードウェアカウンタプロファイリングのサポートが追加されています。
  - SPARC64 VI および VII
  - Intel Core i7: Family 6、 Models 30、 31、 37、 44、 および 46 (Nehalem EP および EX を含む)
  - AMD Family 10h および 11h
- 実験に対するデフォルトのサイズ制限が削除されています。collector limit コマンドを使用して、サイズ制限を設定できます。

er_kernel に対する変更

Solaris カーネルをプロファイリングするためのコマンドは、シグナル SIGINT、SIGTERM、または SIGQUIT のいずれかがプロセスに送信されたときは er_kernel が次の動作を行うように変更されています。

- SIGINT、SIGTERM、または SIGQUIT をキャッチする
- 実験を終了する
- -A off が指定されていない場合は er_archive を実行する

新しい er_generic コマンド

er_generic コマンドは、プロファイル情報を含むテキストファイルから実験を生成します。その後、シミュレートされた実験を、パフォーマンスアナライザまたは er_print コマンドを使用して調べることができます。詳細は、er_generic(1) のマニュアルページを参照してください。

en_desc に対する変更

デフォルトで、en_desc コマンドはすべての子孫を読み取るようになりました。

スレッドアナライザ

スレッドアナライザは、ソースレベルまたはバイナリレベルで計装されているコードのデータ競合の検出をサポートするようになりました。ソースレベルの計装はこのリリースでは変更されていません。

プログラムのバイナリコードを計装するには、`discover` ツールを使用する必要があります。このツールは、Oracle Solaris Studio に組み込まれており、`discover`(1) のマニュアルページで説明されています。『Oracle Solaris Studio 12.2 Discover および Uncover ユーザーズガイド』も参照してください。

データの競合を検出するためにプログラムのバイナリコードを計装するには、`discover` ツールの入力バイナリを次の条件でコンパイルする必要があります。

- オペレーティングシステムのバージョンは、少なくとも Oracle Solaris 10 5/08 または OpenSolaris バージョン snv_70 である必要があります。
- コンパイラは Sun Solaris Studio 12 Update 1 以降のリリースのものである必要があります。
- いずれかのコンパイラ最適化フラグ (`-xO1`、`-xO2`、`-xO3`、`-xO4`、`-xO5`) を使用する必要があります。

バイナリがコンパイラオプション `-xbinopt=prepare` を指定してコンパイルされている場合、SPARC ベースのシステムで実行している以前のバージョンの Solaris でも `discover` ツールを使用できる場合があります。このコンパイラオプションについては、`cc`(1)、`CC`(1)、または `f95`(1) のマニュアルページを参照してください。

バイナリの名前が `a.out` である場合、次のコマンドで `a.out_i` という名前の計装されたバイナリを作成できます。

```
% discover -i datarace -o a.out_i a.out
```

詳細は、『Oracle Solaris Studio 12.2: スレッドアナライザユーザーズガイド』または `tha`(1) のマニュアルページを参照してください。

◆ ◆ ◆ 第 5 章

5 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 12.2 dbx で追加または変更された機能は次のとおりです。

- 最適化済みコードのデバッグのサポートの向上
  - パラメータおよび局所変数を探すための情報を、x86 プラットフォームで使用できます。
  - インライン関数についての情報を、SPARC プラットフォームで使用できます。
- OpenMP の領域、タスク、およびスレッドセットについての情報を表示するための新しいコマンド
  - `omp_pr` [*parallel_region_id*] [`-ancestors`|`-tree`] [`-v`]

    現在の並列領域または *parallel_region_id* によって指定されている領域を出力します。並列領域 ID、タイプ (暗黙的または明示的)、状態 (アクティブまたは非アクティブ)、チームサイズ (スレッド数)、およびプログラムの位置 (プログラムカウンタアドレス) が含まれます。
  - `omp_tr` [*task_region_id*] [`-ancestors`|`-tree`]

    現在のタスク領域または *task_region_id* で指定されている領域の説明を出力します。タスク領域 ID、タイプ (暗黙的または明示的、結合または結合解除)、状態 (生成済み、実行中、または待機中)、発生スレッド、実行スレッド、プログラムの位置、未終了の子、および親が含まれます。
  - `omp_team` [*parallel_region_id*]

現在のチームのすべてのスレッドを出力します。*parallel_region_id* を指定すると、その領域のチーム内のスレッドが出力されます。

- `omp_loop`

  現在のループの説明を出力します。スケジューリングのタイプ (静的または動的)、待機または非待機、順序付き、上下限、および反復の数が含まれます。このコマンドは、現在ループを実行中のスレッドからのみ発行できます。

- `omp_serialize`

  現在のスレッドで検出される次の並列領域を直列化します。

- OpenMP プログラムの既存コマンドの拡張
  - `print -s` *expression*
  - `thread -info`
  - `what is` *name*
  - `where`
- 新しい OpenMP イベント
  - `omp_barrier` [*type*] [*state*]

    バリアーに入っているスレッドのイベントを追跡します。

  - `omp_taskwait` [*state*]

    taskwait に入っているスレッドのイベントを追跡します。

  - `omp_ordered` [*state*]

    順序付き領域に入っているスレッドのイベントを追跡します。

  - `omp_critical`

    クリティカル領域に入っているスレッドのイベントを追跡します。

  - `omp_atomic` [*state*]

    不可分領域に入っているスレッドのイベントを追跡します。

  - `omp_flush` [*type*]

    フラッシュを実行しているスレッドのイベントを追跡します。

  - `omp_task` [*state*]

    タスクの作成と終了を追跡します。

  - `omp_master`

    マスター領域に入っているマスタースレッドのイベントを追跡します。

  - `omp_single`

    単一領域に入っているスレッドのイベントを追跡します。

ソフトウェアの修正事項

ここでは、Oracle Solaris Studio 12.2 dbx のこのリリースで解決された問題について説明します。

1. tracei ステップを行っているときに dbx の実行を停止できない

   Solaris プラットフォームで `tracei` ステップを実行しているときに、Ctrl-C (^C) キーを押して dbx を停止できませんでした。これは実際には Solaris OS のバグですが、問題を回避するように dbx が変更されました。

2. 計装されている debuglog uttsc バイナリにステップインできない

   dbx は、字句ブロック内の C++ 名前空間の別名を正しく処理していませんでした。これにより、明示的に計装されたバイナリに dbx がステップインできない問題が発生していました。

3. Purify を指定して計装されているマルチスレッドプログラムの dbx でスレッド関連のコマンドを使用できない

   Purify を指定すると、計装されるすべての共有ライブラリの名前に接尾辞が追加されます。たとえば、`libc.so.1` は `libc.so.1_pure_p3_c0_1005282029_510_32` になります。dbx は `libc.so.1` の存在を基にして決定を行っており、読み込まれていませんでした。dbx は `_pure*` 接尾辞を認識するようになりました。

4. 特定の GCC 4.x. sybx を復号化できない dbx が SLES 10.2 のコアファイルからプログラム名を抽出できない

   SuSE Linux Enterprise Server 10.2 システムでは、新しい Linux システムのコアファイルに 2 つの note セクションが含まれていて、2 番目のセクションは空であるため、dbx はコアファイルからプログラムを抽出できませんでした。dbx は 1 番目のセクションから名前を取得するようになります。

5. dbx が gcc コードでコンストラクタのコピーを検索する

   gcc は 1 つのメンバーに対して複数のエントリを生成する場合があり、それが DWARF デバッグの pubnames セクション、プロトタイプ、および抽象インスタンスに含まれていました。dbx は、プロトタイプエントリのインスタンスを検出して処理すると、エントリを削除する必要がありました。

6. dbx が SLES 10.2 のコアファイルからプログラム名を抽出できない

   SuSE Linux Enterprise Server 10.2 システムでは、新しい Linux システムのコアファイルに 2 つの note セクションが含まれていて、2 番目のセクションは空であるため、dbx はコアファイルからプログラムを抽出できませんでした。dbx は 1 番目のセクションから名前を取得するようになります。

7. dbx が変数の出力で SIGSEGV を取得する

   実行可能ファイルが -g オプションを指定してコンパイルされていないオブジェクトファイル (.o) から構築されていて、dbx が変数を評価するためにそのようなオブジェクトファイルの 1 つをインポートする必要がある場合、dbx がこの条件を検査していないためにインポートが失敗する場合がありました。

8. dbx — core segv if が有効ではない

   dbx は長さがゼロのコアファイルの可能性を検査していず、それを正しく処理していませんでした。

9. 「メモリー」ウィンドウおよび「逆アセンブル」ウィンドウによって IDE で dbx がクラッシュする場合がある

   「メモリー」ウィンドウまたは「逆アセンブル」ウィンドウの表示中に IDE のデバッグセッションを終了し、そのあとでセッションを再開した場合、どちらかのウィンドウを前面にすると基になっている dbx がクラッシュしていました。

dbxtool

dbxtool の新機能または変更された機能は次のとおりです。詳細は、`dbxtool`(1) のマニュアルページおよび『Oracle Solaris Studio 12.2 dbxtool チュートリアル』を参照してください。

- 「オプション」ウィンドウの大域デバッグオプションが、「セッション起動」プロパティと「ウィンドウ」プロパティに再編されました。「プログラムが停止したら Dbx コマンドタブを前面に表示する」、「ブレークポイントを保存して復元する」、「開始プロセスへのステップを許可する」、「式の評価を吹き出し表示する」の 4 つのプロパティ設定が削除されました。
- 「局所変数」ウィンドウは「変数」ウィンドウになりました。
- 「新規ウォッチポイント」ボタンおよび「式評価」ボタンがツールバーから削除されました。
- 「再起動」ボタンがツールバーに追加されました。
- 「新規ブレークポイント」ダイアログボックスで、LWP、言語モード、および一時チェックボックスが削除されました。「条件」、「数」、「WhileIn」、および「スレッド」フィールドの配置が変更されました。「詳細/簡易表示」ボタンが削除されました。
- 複数のデバッグセッションがある場合、「セッション」ウィンドウが自動的に開きます。
- 「スタックの呼び出し」ウィンドウには最大で 40 フレームが表示されます。「増やす」をクリックすると、40 より多くのフレームを表示できます。
- 「変数」ウィンドウおよび「ウォッチポイント」ウィンドウは静的なメンバーを表示できます。
- 「変数」ウィンドウには自動のみを表示するためのボタンがあります。
- 「変数」ウィンドウには「新規ウォッチポイント」ボタンがあります。
- 「ディスアセンブラ」ウィンドウは「逆アセンブリ」に名前が変更されました。

Discover および Uncover

メモリーアクセスエラーを検出するための高度な開発ツールである Sun メモリエラー探索ツール (Discover) は、このリリースの新機能です。

アプリケーションのコードカバレージを測定するための簡単で使いやすいコマンド行ツールである Uncover は、このリリースの新機能です。

詳細は、`discover` および `uncover`(1) のマニュアルページおよび『Oracle Solaris Studio 12.2 Discover および Uncover ユーザーズガイド』を参照してください。

DLight

Oracle Solaris Dynamic Tracing (DTrace) テクノロジを利用する C/C++ 開発者用のスタンドアロン対話型グラフィカル可観測性ツールである DLight は、このリリースの新機能です。このツールは、Sun Studio 12 Update 1 に含まれていた DLight ツールと同じものではありません。『Oracle Solaris Studio 12.2 DLight チュートリアル』を参照してください。

◆ ◆ ◆ 第 6 章

Solaris Studio IDE

Oracle Solaris Studio 12.2 IDE (Indegrated Development Environment) には、C や C++、Fortran アプリケーションを作成、編集、構築、デバッグ、パフォーマンス解析するためのモジュールが用意されています。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 についての詳細は、IDE のオンラインヘルプおよび『Oracle Solaris Studio 12.2 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 12.2 IDE で追加または変更された機能は次のとおりです。

- NetBeans IDE 6.9 が基になっています。
- Qt アプリケーション開発フレームワークを使用すると、GUI フォーム、リソース、変換などの Qt ファイルを作成できます。
- Run Monitor には、CPU、メモリー、スレッドの使用状況などのアプリケーション実行時に関する情報が表示されます。Solaris プラットフォームでは、I/O 使用状況に加え、「スレッドの詳細」にてスレッドマイクロステートを追跡することができます。
- コールグラフには、グラフィカル表示に加えて、選択した関数から呼び出されるすべての関数または選択した関数を呼び出すすべての関数のツリー表示が含まれるようになりました。
- ハイパーリンクナビゲーションを使用して、オーバーライドされているメソッドからオーバーライドしているメソッドに、またその逆に、ジャンプできるようになりました。

- ソースコードにコメントを追加して、関数、クラス、およびメソッドのドキュメントを生成できます。IDE は Doxygen 構文を使用するコメントを認識して、ドキュメントを自動的に生成します。
- 「オプション」ウィンドウの大域デバッグオプションが、「セッション起動」プロパティと「ウィンドウ」プロパティに再編されました。「プログラムが停止したら Dbx コマンドタブを前面に表示する」、「ブレークポイントを保存して復元する」、「開始プロセスへのステップを許可する」、「式の評価を吹き出し表示する」の 4 つのプロパティ設定が削除されました。
- 「局所変数」ウィンドウは「変数」ウィンドウになりました。
- 「新規ウォッチポイント」ボタンおよび「式評価」ボタンがデバッグツールバーから削除されました。
- 「呼び出し元を現在に設定」および「呼び出し先を現在に設定」ボタンが、デバッグツールバーから削除されました。
- 「再起動」ボタンがツールバーに追加されました。
- 「新規ブレークポイント」ダイアログボックスで、LWP、言語モード、および一時チェックボックスが削除されました。「条件」、「数」、「WhileIn」、および「スレッド」フィールドの配置が変更されました。「詳細/簡易表示」ボタンが削除されました。
- 複数のデバッグセッションがある場合、「セッション」ウィンドウが自動的に開きます。
- 「スタックの呼び出し」ウィンドウには最大で 40 フレームが表示されます。「増やす」をクリックすると、40 より多くのフレームを表示できます。
- 「変数」ウィンドウおよび「ウォッチポイント」ウィンドウは静的なメンバーを表示できます。
- 「変数」ウィンドウには、現在のソースコード行と前のソースコード行だけの変数を表示するためのボタンがあります。
- 「変数」ウィンドウには「新規ウォッチポイント」ボタンがあります。
- 「ディスアセンブラ」ウィンドウは「逆アセンブリ」に名前が変更されました。

ソフトウェア要件

Oracle Solaris Studio IDE には、Java SE Development Kit (JDK) 6 Update 13 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

IDE のプラグインマネージャーでは、IDE のインストール済みプラグインを動的に更新できます。プラグインマネージャーを使用して、新しいプラグインと機能を IDE に追加することもできます。

プラグインマネージャーを使用して IDE を更新する場合は、登録済みのアップデートセンターが IDE によってチェックされ、新しいプラグインや、すでにインストールされているプラグインの新しいバージョンが使用可能かどうか確認されます。新規または更新されたプラグインが使用可能な場合は、プラグインマネージャーを使用してプラグインの選択、ダウンロード、およびインストールを実行できます。

代わりに、「ヘルプ」>「更新の有無を確認」を選択してプラグインマネージャーを開くこともできます。プラグインインストーラは、インストール済みプラグインに対応する更新を確認します。更新が使用可能な場合は、インストーラの手順どおりに更新をインストールできます。

デフォルトの IDE アップデートセンターに加えて、実験用の新しいプラグインや、通常の配布にすでに含まれていない古いプラグインなど、さまざまなタイプのプラグインを提供するいくつかのアップデートセンターから選択することもできます。

インストール済みのプラグインをアップデートセンターから更新する

1. 「ツール」>「プラグイン」を選択し、プラグインマネージャーを開きます。
2. 「更新」タブをクリックし、インストール済みプラグインに対応する使用可能な更新を表示します。
3. 左側の区画で、更新するプラグインを選択し、「更新」する更新をクリックします。
4. インストーラの各ページで操作を完了し、更新をダウンロードおよびインストールします。

「更新」タブの左側の区画には、使用可能な更新がアップデートセンターに存在する、インストール済みプラグインが表示されます。デフォルトでは、IDE は登録済みのアップデートセンターで、インストール済みのプラグインに対応する使用可能な更新を定期的に確認します。左側の区画でプラグインが表示されていない場合は、IDE がアップデートセンターを前回確認したときに、使用可能な更新がなかったことを意味します。

アップデートセンターから新しいプラグインを追加する

1. 「ツール」>「プラグイン」を選択し、プラグインマネージャーを開きます。
2. 「使用可能なプラグイン」タブをクリックし、使用可能でまだインストールされていないプラグインを表示します。

3. 左側の区画で、追加するプラグインを選択し、「インストール」をクリックします。
4. インストーラの各ページで操作を完了し、プラグインをダウンロードおよびインストールします。

一部のプラグインでは、更新プロセスを完了するために IDE を再起動する必要が生じることがあります。

プラグインマネージャーの「設定」タブで、IDE が更新を確認する頻度を設定できます。「カタログの再読み込み」をクリックし、アップデートセンターを今すぐ確認できます。

構成

NetBeans IDE 6.9 のデフォルトヒープサイズは 128M バイトです。最大 500 個までのソースファイルとヘッダーファイルを持つ小規模なプロジェクトを開発する場合、Oracle Solaris Studio 12.2 IDE は、このデフォルト設定で十分動作します。

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 が実行する Java 仮想マシン (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/netbeans/etc/netbeans.conf ファイルで、netbeans.conf ファイル内にある `-J-Xmx` コマンド行の Java 起動スイッチ (下では太字で表示) を編集し、IDE を再起動します。

```
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潤ォ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 のシステム上での大規模アプリケーション開発 (2000 を超えるソースおよびヘッダーファイル) : 1G バイト

Sun JVM を実行している場合、ガベージコレクタスイッチの -J-XX:+UseConcMarkSweepGC (並行コレクタ) および -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。ただし、これらのオプションはSun以外によるJVMの実装ではサポートされていないことがあります。

NetBeansのパフォーマンスチューニングについての詳細は、「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 7 章

7 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 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、dmake は並行して 2 つのジョブを実行します。つまり、多くのプロジェクトがマルチ CPU システムで、2 倍の速度で構築されます。

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

このリリースでのソフトウェアの修正事項

- 修正されたバグ: dmake は、長い内容の条件付きマクロを処理するときにコアをダンプしました。

- 修正されたバグ: DMAKE_OUTPUT_MODE の値が、実装 (TXT1/TXT2) とドキュメント (TEXT1/TEXT2) で異なっていました。dmake は、値「TEXT1」と「TEXT2」も受け付けるようになりました。
- 修正されたバグ: dmake -v は Linux で正しくないバージョンを出力しました。dmake は正しいバージョンを出力するようになりました。
- 修正されたバグ: Modula が考慮していた Modula コンパイラに対する有害な古いルールが、make.rules ファイルから削除されました。
- 修正されたバグ: KEEP_STATE モードでの dmake のメモリーリーク

以前のリリースで追加された機能

- 現在 dmake は、Solaris Studio IDE に統合されています。つまり、デフォルトでは、すべてのプロジェクトが dmake を使用して、並列モードで構築されます。構築モードの変更または並列ジョブ数の変更は、IDE の内部から行うことが可能です。

  1. メインメニューから、「ツール」->「オプション」を選択し、「オプション」ダイアログを開きます。
  2. 「オプション」で、左区画の C/C++ アイコンを選択し、右区画に C/C++ オプションを表示します。
  3. 右区画の「プロジェクトオプション」タブをクリックしてプロジェクトのオプションを表示し、「make オプション」を選択します。
  4. **-m parallel -j 24** と入力します。
  5. 「了解」ボタンを押します。

  これで、すべてのプロジェクトは、最大 24 個の ジョブまで並行モードで構築されます。

- -x SUN_MAKE_COMPAT_MODE=*compatibility-mode* コマンド行オプション

  Solaris make との互換性のための -x SUN_MAKE_COMPAT_MODE=SUN (デフォルト)

  POSIX make との互換性のための -x SUN_MAKE_COMPAT_MODE=POSIX

  GNU make との互換性のための -x SUN_MAKE_COMPAT_MODE=GNU

- 同様に、SUN_MAKE_COMPAT_MODE 環境変数には、互換モードでの dmake の動作を指定する同じ 3 つのオプションがあります。

  Solaris make との互換性のための SUN_MAKE_COMPAT_MODE=SUN (デフォルト)

  POSIX make との互換性のための SUN_MAKE_COMPAT_MODE=POSIX

  GNU make との互換性のための SUN_MAKE_COMPAT_MODE=GNU

- UNIX 2003 準拠。dmake および Solaris 10 OS 付属の make ユーティリティーが、UNIX 2003 適合検査 (XPG5) に合格。
- dmake が AMD64 アーキテクチャーで Sun Grid Engine をサポート。

- AMD64 アーキテクチャーでシステム過負荷制御をサポート。
- ログファイルに関する 2 つの形式オプションをサポートする DMAKE_OUTPUT_MODE 環境変数。オプションの 1 つは、並列ジョブの出力をシリアライズする形式で、ログファイルが見やすくなります。

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。

- インストールを選択できるコンポーネントのリストが変更されました。選択できるコンポーネントは次のとおりです。
  - コンパイラサポートファイル (C および C++ コンパイラで必要)
  - C および C++ コンパイラ
  - Fortran コンパイラ
  - dbx デバッガ
  - dbxtool
  - dmake
  - DLight 可観測性ツール (Solaris プラットフォームのみ)
  - IDE
  - パフォーマンスおよびスレッド分析ツール (Solaris プラットフォームのみ)
  - パフォーマンスライブラリ (Solaris プラットフォームのみ)
  - ScaLAPACK (Solaris プラットフォームのみ)
- GUI インストーラを起動することで、または –libraries-only オプションを指定して非 GUI インストーラを起動することで、実行時ライブラリのみをインストールできるようになりました。このオプションでは、C++ ライブラリ、Fortran 90 ライブラリ、数学ライブラリ (Solaris プラットフォームのみ) がインストールされます。
- ゾーンのあるシステムにインストールしている場合、GUI インストーラのチェックボックスを使用して、および非 GUI インストーラを起動するときにコマンド行から、現在のゾーンのみへのインストールを指定できます。
- 代替ルートインストールは、非 GUI インストーラにおいてのみサポートされるようになり、GUI インストーラではサポートされなくなりました。
- インストーラではライセンスに同意する必要がありません。
- 以前の Sun Studio リリースがシステムにインストールされている場合にインストーラを実行できるゾーンの制限がなくなりました。

◆ ◆ ◆ 第 8 章

8 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。

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

-xprofile に関する問題

- 同じオブジェクトファイルの異なるバージョンを同じプロセスで読み込むと、libxprof が失敗する

  この現象が発生するのは、1 つのディレクトリ内に同じ名前で作成されたバージョンの異なる 2 つの同一ファイルがそれぞれ別々の共有ライブラリにリンクされ、それらがおそらく別々のタイミングで同じプロセスに読み込まれたときです。

  回避策: -xprofile で共有ライブラリを作成するときは、オブジェクトファイル名が異なる UNIX パス名になるようにします。パス名はベース名がない場合であっても区別できるようにしてください。たとえば、次を見てください。

  ```
  /work/mylib/unshared/x.o
  /work/mylib/shared/x.o
  ```

  これらは異なるものとみなされます。

- OMP: libxprof: 表明が失敗する

  プロファイリング実行時ルーチンの呼び出しの間に malloc() が失敗した場合、メモリーが不足している状況では、表明が失敗する場合があります。

回避策: メモリーまたはスワップ空間を追加します。

- `-xprofile=tcov:`*prof_dir* が相対 *prof_dir* を誤って解決する

  `–xprofile=tcov:`*dir* では、絶対パスではない UNIX のパス名は、オブジェクトファイルが生成されるディレクトリを基準にして解決されます。

  回避策: `–xprofile={ collect,use,tcov}`: *dir* では絶対パス名を使用します。

C++

サイズの大きい 10 進整定数の適切な解釈

C++ 規格では、接尾辞のない 10 進整定数は、値が `int` に収まる場合は `int` として、そうでない場合は `long int` として扱うようになっています。値が `long int` にも収まらない場合の結果は定義されていません。

32 ビットモードの場合、型 `int` と `long` のサイズおよびデータ範囲は同じです。1990 C 標準規則に準拠した C++ コンパイラは、`INT_MAX+1` 潤ォ `LONG_MAX` の範囲にある値を unsigned `long` として処理します。この処理は、一部のプログラムでは予期しない結果をもたらします。

1999 C 規格では、接尾辞のない 10 進整数に関するこの規則が変更され、unsigned 型として扱われなくなりました。型は、`int`、`long`、`long long` のうちの最初に値を表せる型になります。

標準モードでは、C++ コンパイラはこの C99 規則に従いますが、`-compat=4` モードではこれまでどおり C90 の規則に従います。`-compat=4` モードでは、コンパイラは C++ 4.2 コンパイラのように動作します。

サイズの大きい 10 進整数を unsigned として扱う場合の移植可能な解決策は、`u` または `U` 接尾辞を使用することです。その他の型にも、それぞれ接尾辞を使用することができます。静的関数

```
// note: 2147483648 == (INT_MAX+1)
2147483648     // (signed) long long
2147483648LL   // (signed) long long
2147483648U    // same as 2147483648u
```

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。

最初の行は、次のように書くこともできます。

```
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
```

テンプレートの構文エラーの検出

次のテンプレートの構文は不正ですが、Sun C++ 4 および 5.0 では、エラーになりませんでした。5.1 以降のすべてのバージョンの C++ コンパイラでは、標準モード (デフォルトのモード) のコンパイルで、構文エラーとして報告されます。

```
template<class T> class MyClass<T> { ... }; // definition error
template<class T> class MyClass<T>; // declaration error
```

どちらの場合も、MyClass<T> の <T> は無効で、次に示すように削除する必要があります。

```
template<class T> class MyClass { ... }; // definition
template<class T> class MyClass; // declaration
```

-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。

言語間リンクエラー

-xlang=f77 コマンド行オプションを使用すると、コンパイルプロセスでリンカーエラーが発生します。エラーを回避するとともに適切な実行時ライブラリをインクルードするには、代わりに -xlang=f77,f90 を使用してコンパイルします。

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。

  次に例を示します。

  ```
  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。

  次に例を示します。

  ```
  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。静的関数

```
#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。

デバッグツールから、メンバー関数に余分な先行パラメータがあるという誤ったメッセージが返される

互換モード (-compat) では、C++ コンパイラはメンバー関数を指すポインタのリンク名を正しく符号化しません。このエラーのため、復号化プログラムおよび、dbx や c++filt などのデバッグツールから、メンバー関数に余分な先行パラメータ (メンバー関数が属しているクラスタイプを示す) があると報告されます。この問題を解決するには、-Qoption ccfe -abiopt=pmfun1 フラグを追加します。しかし、一般に、このフラグを使用してソースをコンパイルすると、このフラグなしでコンパイルしたソースとの間のバイナリレベルの互換性が失われることがあります。標準モード (デフォルトモード) では、この問題は起きません。

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。

ユーザーは、k をどのように使用するか、それによってどのような効果を得ようとするかに基づき、次に示す代替方法を実施できます。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;
}
```

関数の多重定義の解決

C++ コンパイラの従来のリリースでは、C++ 標準の要件に従って関数の多重定義の解決を行いませんでした。今回のリリースでは、多重定義された関数の呼び出しを解決して、多くのバグを修正しています。特に、コンパイラは、呼び出しが実際にあいまいな場合は関数をピッキングしたり、実際にはそうでない場合にも、呼び出しがあいまいであると表示したりする場合がありました。

あいまいであることを示すメッセージに関する回避策には、不要なものもあります。以前には報告されなかった、あいまいに関する新しいエラーが発生しています。

あいまいな関数呼び出しの主な原因の 1 つは、組み込み型のサブセットにさえも多重定義が発生することです。

```
int f1(short);
int f1(float);
...
f1(1); // ambiguous, "1" is type int
f1(1.0); // ambiguous, "1.0" is type double
```

この問題を修正するには、f1 をまったく多重定義しないか、昇格を経験しない各型、つまり int、unsigned int、long、unsigned long、double を多重定義します (long long、unsigned long long、および long double 型がある場合もあります)。

もう 1 つのあいまいに関する主な原因はクラスにおける型変換関数で、特に多重定義された演算子またはコンストラクタが存在する場合です。

```
class T {
public:
        operator int();
        T(int);
        T operator+(const T&);
};
T t;
1 + t // ambiguous
```

この演算は、次のように解決できるので、あいまいです

```
  T(1) + t     // overloaded operator
 1 + t.operator int()    // built-in int addition
```

多重定義された演算子または型変換関数を使用できますが、両方使用すると、あいまいと判断されます。

実際、型変換関数そのものは、あいまいと判断されたり、意図しなかった場所で変換が発生したりすることがたびたびあります。変換を有効にする必要がある場合は、型変換関数ではなく名前付き関数を使用してください。たとえば、`operator int();` の代わりに `int to_int();` を使用します。

この変更により、演算子 `1 + t` はあいまいではなくなります。`T(1) + t` としか解釈できません。ほかの解釈が必要な場合は、`1 + t.to_int()` のように記述する必要があります。

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。

- このリリースでは廃止された FORTRAN 77 ライブラリが削除されているため、以前の 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は動作しません。
- 拡張された配列構成子を使用して、型が長さ引き継ぎの文字であるパラメータ定数を設定すると、文字要素の値が連結されます。この問題を回避するには、想定長の代わりに、配列構成子で使用されているものと同じ文字長を使用して、パラメータ定数を定義します。
- 名前の箇所を空白にして C バインド手順を指定すると、適切に処理されず、その手順には空白の名前が使用されます。この問題を回避するには、C バインド名を指定するか、または必要ない場合は C バインド名を使用しないようにします。

従来のリリースの f95 コンパイラで生じた互換性の問題は今回のリリースのコンパイラにも継続して存在します。従来の f95 のリリースから更新を行う場合は、それらの互換性の問題に注意してください。互換性の問題とは次のとおりです。

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。

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ると、データ収集で問題が発生する

   コレクタライブラリ libcollector.so を事前に読み込まずに実行プロセスに dbx を接続すると、多数のエラーが発生します。

   - 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。トレーシングデータはさまざまなライブラリへの割り込み処理によって収集されます。libcollector.so が事前に読み込まれていない場合、割り込み処理ができなくなります。
   - 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ハードウェアカウンタライブラリが読み込まれる場合、ハードウェアカウンタ実験は成功しますが、libcpc ライブラリ関数への参照は libcpc.so の検索ではなく一般的検索によって解決されます。
   - 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. dbx で Java コードのデバッグ中に障害が発生する場合がある

   dbx シェルの中で、**cd** コマンドを実行した場合、もしくは CLASSPATH 環境変数または CLASSPATHX 環境変数を設定した場合、dbx でセグメント例外が発生することがあります。

   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. dbx で Java コードの再デバッグ中に障害が発生する

   Java コードに対して 2 つの debug コマンドを実行することによって、dbx で障害が発生する場合があります。

4. dbx で、アプリケーションをその構築に使用したものと異なる J2SE 上でデバッグすると、例外がスローされる

   アプリケーションを、そのアプリケーションの構築に使用したバージョンの J2SE テクノロジと異なるリリースの J2SE テクノロジの下でデバッグすると、dbx が例外をスローします。

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 12.2 dbx には次の制限事項があります。

- 次の dbx の機能は、 x 86 ベースのシステム上にある 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
```

- 次の dbx の機能は、 x 64 ベースのシステム上にある Linux OS では使用できません。
  - 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 を起動した場合を除く)。
- dbx は、Linux プラットフォームでフォークされたプロセスを追跡できません。また、`exec()` が呼び出されたときに新しいプログラムに切り替えられません。
- 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は、dbx をハングアップさせる可能性があります。

  ```
  where | head –1
  ```

  回避策:
  - Ctrl-C キーを押し、新しい dbx プロンプトを表示します。
  - dbx は大量の情報をキャッシュに書き込むため、前述の例の場合は、次のコマンドシーケンスで機能します。

    ```
    where
    where | head –1
    ```
- 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。
  - 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によって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 はそれらのシグナルをユーザーから隠し、ほかのソースからの純粋な SIGSTOP シグナルを監視できるようにします。しかし、まれに Linux が予期しない方法で SIGSTOP を使用することがあり、その場合、dbx はシステム生成の SIGSTOP をユーザー生成の SIGSTOP と解釈します。

    回避策:

    `ignore` コマンドを使用して、SIGSTOP シグナルをキャッチしないよう dbx に指示してください。
  - スレッドは終了するが、Linux から dbx にその終了が報告されないことがあります。この問題は、新しいスレッドライブラリ (NPTL) を利用すると発生することが少なくなります。

スレッドが終了し、その終了が報告されない場合、dbx は決して起こることのないイベントを待ち、新しいプロンプトを表示しません。この状況は、dbx で cont コマンドを発行したあとでもっとも多く発生しますが、step up コマンドや step コマンド、next コマンドのあとでも発生することがあります。

回避策:

- Ctrl-C キーを押すと、dbx が待ち状態を終了し、新しいプロンプトを表示することがあります。
- Ctrl-C キーが機能しない場合は、いったん dbx を終了して、再起動します。

- g++ コンパイラでコンパイルされているプログラムの場合、C++ 式に関する実行時型情報は得られません。
- 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。
- compat=4 のとき、dbx がメンバー関数に対するポインタを不正に復号化します。compat=5 では、この問題は発生しません。

  回避策: 次のコマンドを使って、プログラムを再コンパイルしてください。

  ```
  CC –compat=4 –Qoption ccfe –abiopt=pmfun1
  ```

  このフラグによって ABI が変更されるため、正規の構築には使用しないでください。
- 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。
- 古い `libC.so.5` または `libC.so.4` を使用すると、C++ の例外領域で dbx に問題が発生します。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 の場合、実行時検査機能を最大限に活用するには、`-stackvar` コンパイラオプションを使用してください。
- プログラムによっては、`-stackvar` が正しく機能しないことがあります。そのような場合は、`-C` コンパイラオプションを試してください。このオプションは、添字の検査を有効にします。
- 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。
- call コマンドまたは印刷関数の呼び出しを使用すると、マルチスレッドアプリケーションがデッドロック状態になることがあります。
- ファイルがプリコンパイル済みヘッダー (PCH) によって収集されたものの一部であった場合は、ヘッダーファイルの変更に dbx の修正継続機能を使用しないでください。

- 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。

パフォーマンスアナライザ

Linux で Oracle Message Passing Toolkit 8.2 または 8.2.1 を使用している場合、回避策が必要な場合があります。バージョン 8.1 または 8.2.1c では、または Oracle Solaris Studio コンパイラを使用している場合はすべてのバージョンで、回避策は必要ありません。

Oracle Message Passing Toolkit のバージョンは、`/opt/SUNWhpc/HPC8.2.1` などのインストールパスで示されています。または、`mpirun –V` と入力して表示される次のような出力では、斜体の部分でバージョンが示されています。

```
mpirun (Open MPI) 1.3.4r22104-ct8.2.1-b09d-r70
```

アプリケーションを GNU または Intel コンパイラでコンパイルし、Oracle Message Passing Toolkit 8.2 または 8.2.1 を MPI 用に使用している場合、MPI の状態データを取得するには、Oracle Message Passing Toolkit の `link` コマンドで `-WI` および `--enable-new-dtags` オプションを使用する必要があります。これらのオプションを使用すると実行可能ファイルで `RPATH` に加えて `RUNPATH` が定義され、MPI 状態ライブラリが `LD_LIBRARY_PATH` 環境変数で有効になります。

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. 各ホスト上のリモートログイン (rsh) はパスワードなしで可能か。また、各リモートログインは妥当な時間内 (2 秒未満) に行えるか。

```
% 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 (構築プロセスの開始に使われるマシン) 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 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 オプションを使用して非 GUI インストーラを実行すると、ユーザーが判読できないエラーメッセージで失敗する場合があります。

索引