Oracle® Solaris Studio 12.4 リリースの新機能

Part No: E57204
2014 年 12 月

ORACLE®

目次

このドキュメントの使用法

- **概要** – Oracle Solaris Studio 12.4 のこのリリースにおけるコンパイラとツールの新機能や変更点について説明します。
- **対象読者** – アプリケーション開発者、システム開発者、設計者、サポートエンジニア
- **必要な知識** - プログラミング経験、ソフトウェア開発テスト、ソフトウェア製品を構築およびコンパイルできる能力

製品ドキュメントライブラリ

コンパイラおよびツールの詳細については、関連するマニュアルページまたはヘルプ、および http://docs.oracle.com/cd/E37069_01 にあるドキュメントライブラリを参照してください。

システム要件および既知の問題は、『Oracle Solaris Studio 12.4: リリースノート 』に記載されています。

Oracle サポートへのアクセス

Oracle のお客様は、My Oracle Support を通じて電子的なサポートを利用することができます。詳細は、http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info (聴覚に障害をお持ちの場合は http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs) を参照してください。

フィードバック

このドキュメントに関するフィードバックを http://www.oracle.com/goto/docfeedback からお聞かせください。

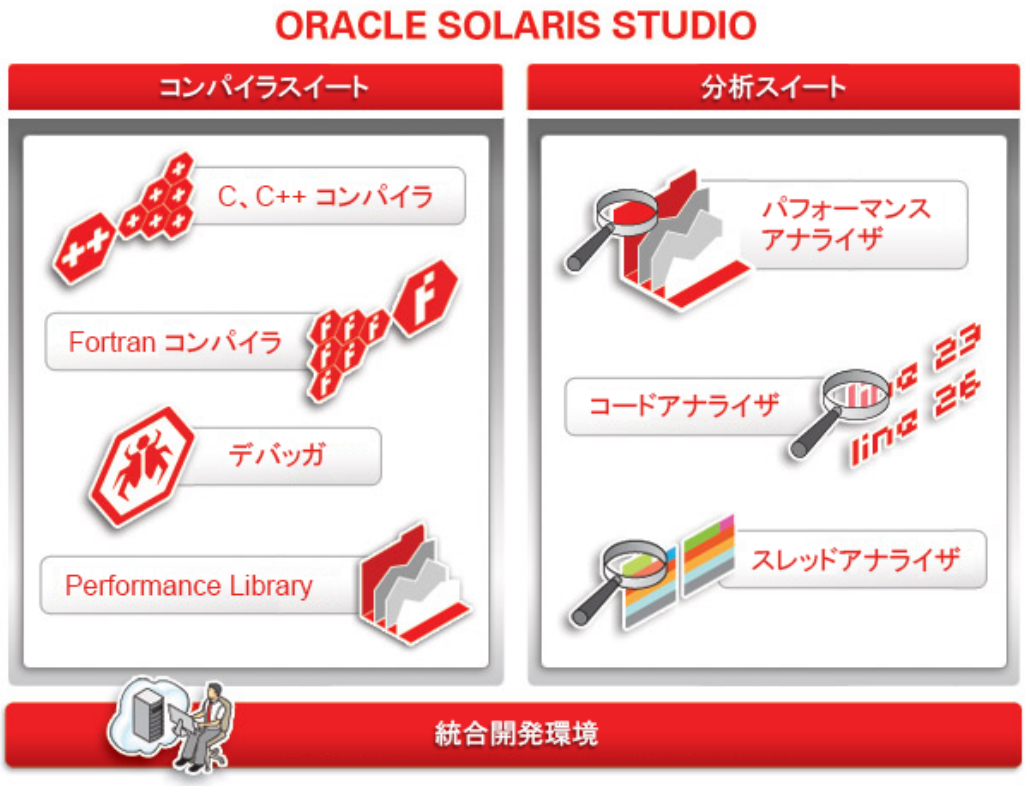
♦ ♦ ♦ 第 1 章

1 Oracle Solaris Studio 12.4 リリースの紹介

この章では、このリリースの重要な更新の概要について説明します。

Oracle Solaris Studio の概要

Oracle Solaris Studio には、コンパイラスイート、分析スイート、および両方のスイートからコンパイラおよびツールとともに使用するように調整されたグラフィカル統合開発環境 (IDE) が組み込まれています。これらを組み合わせることで、Oracle Sun ハードウェア上で最高のパフォーマンスを発揮するアプリケーションの開発用に最適化された開発環境が提供されます。



このリリースでの主な特長

Oracle Solaris Studio 12.4 では、そのすべてのコンパイラおよびツールの機能が拡張されています。主な特長は、次の領域に見られます。

C++ コンパイラ

C++11 標準をサポートします。第2章「C++ コンパイラ」を参照してください。

パフォーマンスアナライザ

パフォーマンスデータをナビゲートおよび把握し、アプリケーションを分析する箇所を指定する操作を容易にするために、再設計されたグラフィカルインタフェースです。

パフォーマンスアナライザは Linux、Windows、または Mac クライアントから実行でき、Oracle Solaris *または Linux* システム上のパフォーマンスデータにリモートからアクセスできます。

	第3章「パフォーマンス解析ツール」を参照してください。
コードアナライザ	静的解析中の誤検出を除去する能力が大幅に向上されました。 実行時データを収集するときのオーバーヘッド時間が削減されました。 第4章「コード分析ツール」を参照してください。
デバッガ	大規模なバイナリを処理するときの `dbx` 起動時間が短縮されました。 デバッグ情報のサイズが削減されました。 第5章「デバッグツール」を参照してください。
IDE	IDE 内のメモリーフットプリントが大幅に削減されたため、大規模プロジェクトのオープン、検索、および変更をさらに速く行うことができます。 プロジェクト設定により、大規模プロジェクトがバージョン管理システムで使いやすくなりました。 第6章「Oracle Solaris Studio IDE」を参照してください。
新規サーバーのための最適化	次に示す Oracle Sun ハードウェアサーバーのアプリケーションパフォーマンス向上のためにコンパイラおよびライブラリが最適化されました。SPARC T5、SPARC M5、SPARC M6、SPARC M10、SPARC M10+、Intel Haswell、および Intel Ivy Bridge。これらの改良の一部は、Oracle Solaris Studio 12.3 プラットフォーム固有の機能拡張リリースですでに導入済みのものです。 77 ページの「新しいハードウェア上でのアプリケーションパフォーマンス」を参照してください。
OpenMP 4.0	OpenMP API 標準言語仕様の主要アップグレードである OpenMP API バージョン 4.0 の新しい機能が実装されています。 第7章「OpenMP API とスレッドアナライザ」を参照してください。

♦ ♦ ♦ 第 2 章

2 C++ コンパイラ

この章では、このリリースの Oracle Solaris Studio の C++ 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。

C++ コンパイラについて

Oracle Solaris Studio C++ コンパイラを使用すると、Oracle の最新の SPARC プロセッサベースシステムおよび Intel x86 プロセッサベースシステム用に、高性能の並列 C++ アプリケーションをビルドすることができます。

C++ コンパイラ (cc) は、指定されたコマンド行オプションに従って、特定のオペレーティングシステム、プロセッサ、アーキテクチャー、メモリーモデル (32 ビットおよび 64 ビット)、浮動小数点演算などを対象とするコードを生成します。コンパイラは、シリアルソースコードに対して自動並列化を実行し、マルチコアシステムで強化されたパフォーマンスを発揮するバイナリを生成するほか、ほかの Oracle Solaris Studio ツールによる強化されたデバッグまたは分析用にバイナリを準備することもできます。また、コンパイラは GNU C/C++ 互換機能もサポートします。

C++11 標準のサポート

新しい C++11 標準は C++ を強化して、コードを無駄のない安全なものにすることができる追加のツールを提供します。コンパイラは、Oracle ハードウェア上で SPARC および x86 の高速化されたパフォーマンスを維持します。auto、スマートポインタ、nullptr、範囲 for、非メンバーの

begin および end、ラムダ関数およびアルゴリズム、右辺値参照、移動コンストラクタ、統一初期化、初期化子リストといった新機能は、C++ ライブラリの設計方法を変化させるものとして期待されます。

これは、C++11 標準のサポートが含まれるはじめての Oracle Solaris Studio リリースです。このリリースでは、以下を除くすべての C++ 11 機能がサポートされます。

次の機能はサポートされません。

- C++ 11 の並行性および不可分操作
- ユーザー定義リテラル

C++11 機能の使用

Oracle Solaris Studio 12.4 で、C++ コンパイラは新しい言語であり ABI (Application Binary Interface) である C++11 をサポートします。

C++ 11 モードの CC コンパイラは g++ ABI、および Oracle Solaris Studio に付属するバージョンの g++ ランタイムライブラリを使用します。このリリースでは、g++ ランタイムライブラリのバージョン 4.8.2 が使用されます。

ABI は、生成されるオブジェクトコードの低レベルの詳細を記述します。異なる ABI を使用するモジュールをプログラムへと正常にリンクすることはできません。つまり、プログラムのすべてのモジュールで C++11 モードを使用するか、すべてのモジュールで使用しない必要があります。

Oracle Solaris Studio 12.4 C++ を Oracle Solaris Studio 12.3 (C++ 5.12) へのアップグレードとして使用し、C++11 機能を使用しない場合はスクリプトまたは Makefile の変更は必要ありません。例外は Rogue Wave Tools.h++ であり、使用できません。サポートされなくなった機能の詳細は、『Oracle Solaris Studio 12.4: リリースノート 』の「今回のリリースで削除された機能」を参照してください

C++11 モードでコンパイルするには、オプション `-std=c++11` を `CC` コマンド行に追加します。コマンド行での位置は重要ではありません。このオプションにより、コンパイラは C++11 での新しい言語機能を認識し、標準ライブラリの C++11 バージョン (付属する g++ ランタイムライブラリ) を使用します。`-std=c++11` と互換性がないとしてマークされているオプションを除き、その他すべてのコマンド行オプションを C++11 とともに使用でき、通常どおりの効果を発揮します。`-std=c++11` オプションは、ライブラリまたは実行可能プログラムをビルドするときに使用される `CC` コマンドごとに、一貫して使用する必要があります。

注記 - C++11 の機能は、デフォルトで使用できません。任意の C++11 機能を使用するには、新しい `-std=c++11` オプションを `CC` コンパイラで使用する必要があります。このオプションは g++ ABI を使用しますが、異なる ABI を選択するオプションはありません。プログラムのすべてのモジュールをコンパイルするときに、このオプションを使用する必要があります。

このリリースにおける C++11 の互換性情報

このリリースのコンパイラには、次のように更新されたバージョンの詳細情報があります。

コンパイラのバージョン: C++ 5.13

コンパイラのバージョンのマクロ: `__SUNPRO_CC = 0x5130`
コンパイラのバージョンのマクロは、以前のどのリリースよりも厳密に大きいため、`__SUNPRO_CC>=0x5100` などのバージョン比較は動作し続けます。

コンパイラのデフォルトモード: C++03、`-compat=5` を指定。
これは、Oracle Solaris Studio 12.3 リリースの C++ 5.12 と同じデフォルトモードです。

Oracle Solaris Studio 12.3 には、次のコンパイラモードオプションがありました。

`-compat=5` このオプションは、C++03 と Sun ABI を選択します。これはデフォルト値です。

`-compat=g` このオプションは、C++03 と g++ ABI を選択し、コンパイラに付属する gcc ヘッダーとライブラリを使用します。Oracle Solaris Studio 12.3 では、gcc ランタイムライブラリが存在する場合、Oracle Solaris の `/usr/sfw/lib` にインストールされます。このリリースでは、代わりにコンパイラに付属する gcc ランタイムライブラリが使用されます。

このリリースでは、オプション `-std=[ c++11 | c++0x | c++03 | sun03]` が追加されました。オプションの値は、次のように定義されます。

`-std=c++11` このオプションは、C++11 と g++ ABI を選択し、Oracle Solaris Studio 12.4 の一部としてインストールされる g++ 4.8.2 ランタイムライブラリを使用します。

`-std=c++0x` このオプションは、`-std=c++11` オプションと同等であり、GCC 互換性のために提供されます。C++11 標準には、当初 C++0x というニックネームが付けられていました。

`-std=c++03` このオプションは、`-compat=g` オプションと同等です。

`-std=sun03` このオプションは、`-compat=5` と同等です

注記 - `-compat` オプションと `-std` オプションを混在させることはできず、両方を使用するとエラーが発生します。複数の `-std` オプションまたは複数の `-compat` オプションが 1 つのコマンド行に出現する場合、最後に指定されたものがそれまでに指定されたものをオーバーライドします。例:

```
-compat=g   -compat=5   // OK, -compat=5 is used
-std=c++11  -std=c++03  // OK, -std=c++03 is used
-std=c++11  -compat=g   // always an error
-compat=g   -std=c++03  // always an error
```

---

16 ビット Unicode と C++11 の非互換性

オプション `-xustr=ascii_utf16_ushort` は、C++11 と互換性がなく、許可されません。

このオプションは、`U"`*ASCII_string*`"` を 16 ビット Unicode として解釈しますが、C++11 ではこの構文で 32 ビット Unicode を必要とします。

このリリースにおける C++11 とのライブラリ非互換性

`-std=`*x* または `-compat=g` の場合、次の `-library=`*v* オプションは許可されません。

- `Cstd`
- `stlport4`
- `stdcxx4`
- `Crun`
- `iostream`

リンクされるライブラリをリストする必要がある場合 (たとえば共有ライブラリを作成するとき)、`-compat=g` または `-std=`*x* オプションの使用時に、次のオプションをこの順序で使用します。

```
-lstdc++ -lgcc_s -lCrunG3
```

`CC` を使用して実行可能プログラムを作成する場合は、これらのライブラリは `CC` ドライバによってリストされるため、自分でリストしないでください。

このリリースで削除されたライブラリについては、『Oracle Solaris Studio 12.4: リリースノート』の「今回のリリースで削除された機能」を参照してください。

C++11 モードの使用例

モジュール `main.cc`、`f1.cc`、および `f2.cc` から実行可能プログラム `myprogram` をビルドする場合、次のコマンドを使用することができます。

```
% CC -std=c++11 -m32 -O -c main.cc
% CC -std=c++11 -m32 -O -c f1.cc f2.cc
% CC -std=c++11 -m32 -O main.o f1.o f2.o -o myprogram
```

Oracle Solaris Studio の以前のバージョンを使用して C++ プログラムをビルドする Makefile がある場合は、`-std=c++11` を各 `CC` コマンド行 (通常は `CCFLAGS` および `LFLAGS` マクロ内) に追加し、互換性のないオプション (`-compat=5`、`-library=stlport4` など) を削除することで、C++11 プログラムをビルドするように変換できます。有効な C++ プログラムは、C++11 モードでコンパイルするときに、通常は変更なしでコンパイルして実行されます。しかし、実際のプログラムの多くは、標準的でないコンパイラの動作や機能拡張に依存しており、意図せずに依存していることもあります。そのようなコードは、C++11 モードでコンパイルできないことがあります。

その他の C++ コンパイラの変更点

次に、C コンパイラに固有のバージョン 5.13 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、`cc` (1) のマニュアルページを参照してください。

C++ コンパイラの変更点には、77 ページの「全コンパイラに共通の新機能および変更点」で説明する変更点も含まれます。

詳細は、『Oracle Solaris Studio 12.4: C++ ユーザーズガイド』と `CC` のマニュアルページを参照してください。

- すべてのプラットフォームでの `-compat=g` のサポート。
- 新しいコンパイラオプション: `-std` を使用すると、g++ 互換性のある C++ 03 または C++ 11 文法を選択できます。`-std=c++11` を使用する場合、次の制限が適用されます。
  - 汎用文字名 (エスケープされた Unicode 文字) は、現在サポートされていません。

  - <iostream.h>、<fstream.h> など、.h で終わる非標準の iostream ヘッダーは使用できません。これらのヘッダーは、旧形式の C++ から C++98 への移行を容易にするためのものでした。
- 新しいコンパイラオプション: -features=[no%]rtti は、実行時型情報 (RTTI) を無効にします。
- 新しいコンパイラオプション: -xprevise は、コードアナライザで表示できるソースコードの静的分析を生成します。
- -x*option* と同等の次のオプションは、非推奨になりました。

  -help

  -inline

  -libmieee

  -libmil

  -nolib

  -nolibmil

  -pg

  -time

  -unroll

  代わりに、-xhelp、-xinline、-xlibmieee などを使用してください。
- x86 での -xregs=float のサポート。
- -errtags の動作は、C コンパイラと同じく、警告メッセージに対してのみタグを発行するようになりました。以前の C++ コンパイラでは、-errtags オプションにより、タグが警告とエラーの両方のメッセージの一部として出力されました。
- -template=extdef から -template=no%extdef に変更されたデフォルトの -template オプション。

  この変更は、その他のコンパイラでは -template=extdef によって想定される definitions separate テンプレートモデルを使用しないためです。-template=extdef オプションは、ほとんどのコードが従わないソースコードの編成方法に関する厳密な要件を課します。Oracle Solaris Studio C++ のみを使用して開発するのでないかぎり、-template=no%extdef オプションが必要になる可能性があります。

  詳細は、『Oracle Solaris Studio 12.4: C++ ユーザーズガイド 』の第 6 章「テンプレートの作成と使用」、および変更されたデフォルト C++ テンプレートコンパイルモデルの効果

についての説明 (http://www.oracle.com/technetwork/articles/servers-storage-dev/changed-default-cpp-template-model-2292727.html)を参照してください。

注記 - `-library=stdcxx4` オプションは、現在 `-template=no%extdef` とともに使用できません。`-library=stdcxx4` オプションを使用する場合、ライブラリのパッチが利用可能になるまでは、コマンド行で C++ コードをコンパイルするときに `-template=extdef` を指定します。

コンパイラによって適用されるより厳密な C++ ルール

C++ コンパイラは、一部の C++ ルールを以前のコンパイラよりも厳密に適用します。より厳密な適用、違反しているコードの例、違反しているコードを修正するための解決方法については、『Oracle Solaris Studio 12.4: C++ ユーザーズガイド 』の「Oracle Solaris Studio 12.4 C++ 5.13 コンパイラの新機能」を参照してください。

♦ ♦ ♦ 第 3 章

3 パフォーマンス解析ツール

パフォーマンス解析ツールを組み合わせて使用することで、ユーザーはアプリケーションの動作を解析し、パフォーマンスに影響を及ぼす問題点を見つけることができます。

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について

パフォーマンスアナライザは、アプリケーションの動作をユーザーが把握できるようにすることで、コード内で問題となっている領域を見つけることができます。パフォーマンスアナライザは、システムリソースをもっとも多く使用している関数、コードセグメント、およびソース行を識別します。パフォーマンスアナライザは、単一スレッド、マルチスレッド、およびマルチプロセスのアプリケーションのプロファイルを作成でき、プロファイリングデータを提供することで、アプリケーションのパフォーマンスを改善できる場所を特定するのに役立てることができます。

パフォーマンスアナライザは一連のコマンドおよびツールで構成されており、プロファイリングデータを収集する `collect` ユーティリティー、解釈されたプロファイリング情報をテキスト形式で表示する `er_print` ユーティリティー、プロファイリング情報をグラフィカルに表示するパフォーマンスアナライザ GUI などがあります。

スレッドアナライザは、マルチスレッドの問題に的を絞ることが可能な関連ツールです。詳しくは、75 ページの「スレッドアナライザ」を参照してください。

パフォーマンスアナライザのドキュメント

このリリースには、パフォーマンスアナライザでのサンプルアプリケーションのプロファイリングと、データ実験の検査についてのステップごとの手順を含む、新しい『Oracle Solaris Studio 12.4: パフォーマンスアナライザチュートリアル 』マニュアルが含まれます。

『Oracle Solaris Studio 12.4: パフォーマンスアナライザ 』リファレンスマニュアルには、パフォーマンスアナライザ、`collect`、`er_print` およびその他のユーティリティーの詳細が含まれています。

パフォーマンスアナライザの新機能

このセクションでは、パフォーマンスアナライザおよび関連ツールのこのリリースでの新機能の概要を示します。詳細は、パフォーマンスアナライザのヘルプおよび各コマンド行ツールのマニュアルページを参照してください。

- パフォーマンスアナライザの UI が、新しいナビゲーション機能と新しい「概要」および「ようこそ」画面で再設計されました。23 ページの「ユーザーインタフェースの再設計」を参照してください。
- タイムライン機能により、データ表示、ナビゲーション、およびフィルタリング機能が改善されました。26 ページの「タイムラインの改善」を参照してください。
- 「ソース」および「逆アセンブリ」データビューに、構文強調表示およびナビゲーション用のハイパーリンクが表示されます。詳しくは、27 ページの「「ソース」と「逆アセンブリ」の改善」を参照してください。
- 「呼び出しツリー」ビューに新しいグラフィックインジケータが表示され、ナビゲーションが改善されました。詳しくは、29 ページの「呼び出しツリーの改善」を参照してください。
- Java アプリケーションのプロファイリング作成が実行しやすくなりました。30 ページの「Java プロファイリングの改善」を参照してください。
- メモリー領域プロファイリングのデータ収集と表示が改善されました。31 ページの「メモリー領域プロファイリングの改善」を参照してください。
- 新しいメトリックはハードウェアカウンタ実験内で計算できます。詳しくは、32 ページの「新しい派生メトリック: CPI および IPC」を参照してください。
- すべてのプラットフォーム上で記録された実験は、ほかのすべてのプラットフォーム上で実行中のパフォーマンスアナライザまたは `er_print` ユーティリティーで読み取ることができます。33 ページの「クロスプラットフォーム解析」を参照してください。
- パフォーマンスアナライザを Windows または Mac マシンからリモートで使用したり、Oracle Solaris Studio をフルインストールしていない Oracle Solaris または Linux マシンから使用したりすることが可能になりました。34 ページの「パフォーマンスアナライザのリモート使用」を参照してください。
- ターゲットプログラムの I/O トレースがツールでサポートされるようになりました。35 ページの「新しい I/O データビュー」を参照してください。

- 新しい「ヒープ」ビューでヒープトレースデータの表示が改善され、メモリーリークの検出に役立ちます。36 ページの「新しいヒープデータビュー」を参照してください。
- 37 ページの「パフォーマンスアナライザへのその他の変更」

詳細は、パフォーマンスアナライザのヘルプを参照してください。

ユーザーインタフェースの再設計

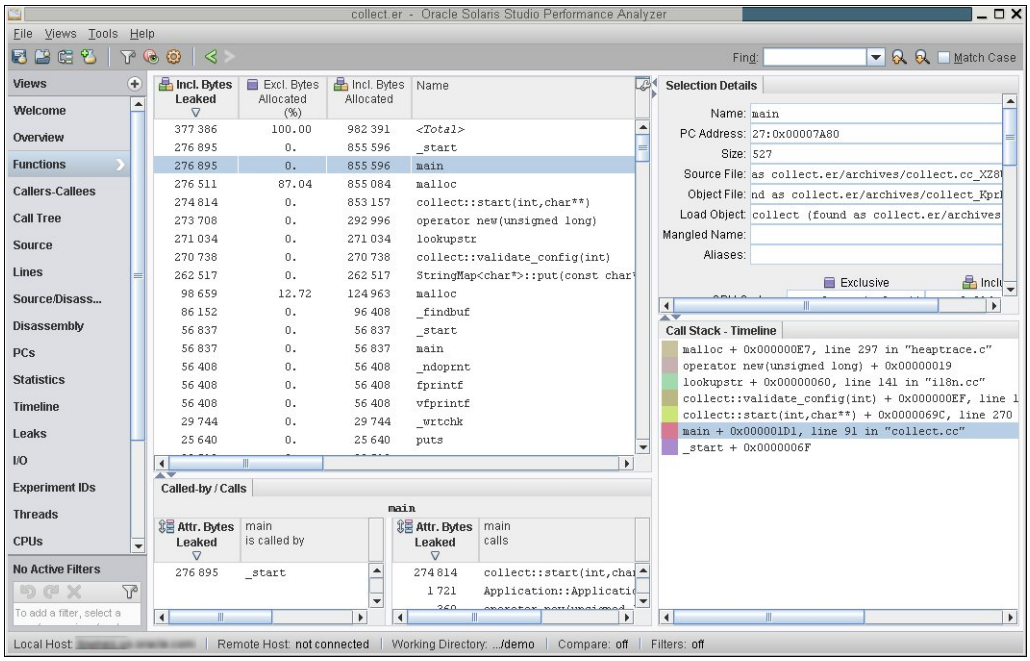
パフォーマンスアナライザユーザーインタフェースでは、データの表示およびナビゲーションが改善されました。UI 変更のハイライトとして、次のものがあります。

- パフォーマンスデータは、左側のナビゲーションバーからアクセスするデータビューに表示されるようになりました。これは、以前使用していた水平のタブナビゲーションに置き換わるものです。詳しくは、23 ページの「パフォーマンスアナライザのナビゲーション」を参照してください。
- 新しい「ようこそ」画面は、実験名を指定せずにパフォーマンスアナライザを開始したときに表示されます。詳しくは、24 ページの「パフォーマンスアナライザの「ようこそ」画面」を参照してください。
- 新しい概要画面は、実験を開いたときの最初のデータビューとして表示されます。詳しくは、25 ページの「パフォーマンスアナライザの「概要」画面」を参照してください。

パフォーマンスアナライザのナビゲーション

パフォーマンスアナライザは、パフォーマンスアナライザウィンドウの左側のナビゲーションバーからアクセスするデータビューを中心にして整理されました。各ビューには、プロファイルされたアプリケーションのパフォーマンスメトリックが異なる観点で表示されます。従来どおり、「関数」ビューが中心的な役割を果たします。関数を選択すると、ほかのデータビューも更新され、その選択した関数に焦点が当てられます。

図 3-1 縦型ナビゲーション領域で「関数」ビューが選択された状態



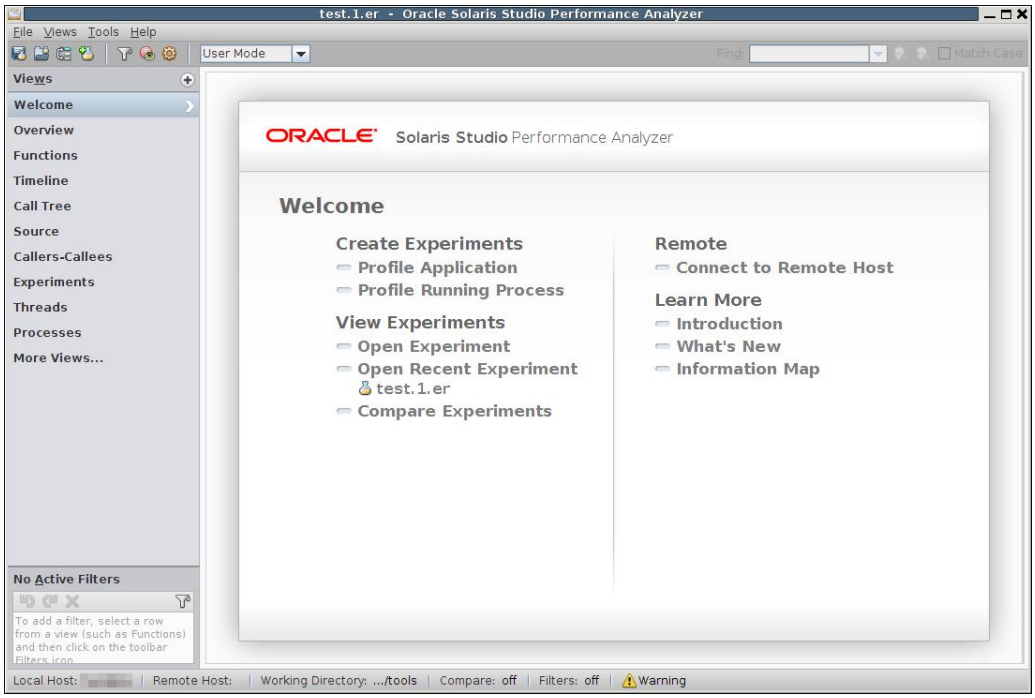
ナビゲーションバーで使用可能なデータビューは、開いた実験で使用可能なデータによって決まります。ナビゲーションバーの上にある「ビュー」メニューまたは「+」ボタンを使用して、データビューを簡単に追加および削除できます。前の図に示すように、データビューで選択した項目の詳細は、右側に表示されます。

ナビゲーションバーの下にある「アクティブなフィルタ」領域には、現在適用されているデータフィルタの名前を示し、フィルタを削除および再適用したり、すべてのフィルタを削除したりすることができます。データビューで項目を右クリックしてポップアップリストからフィルタを選択するか、ツールバーのフィルタアイコンをクリックしてフィルタを適用することができます。

パフォーマンスアナライザの「ようこそ」画面

パフォーマンスアナライザでは、ツールを起動したときに実験名が指定されない場合、新しい「ようこそ」画面が表示されます。「ようこそ」画面には、アプリケーションのプロファイルの作成、最近の実験の表示、実験の参照、およびドキュメントの表示を容易に行うためのリンクが表示されています。

図 3-2 パフォーマンスアナライザの「ようこそ」画面



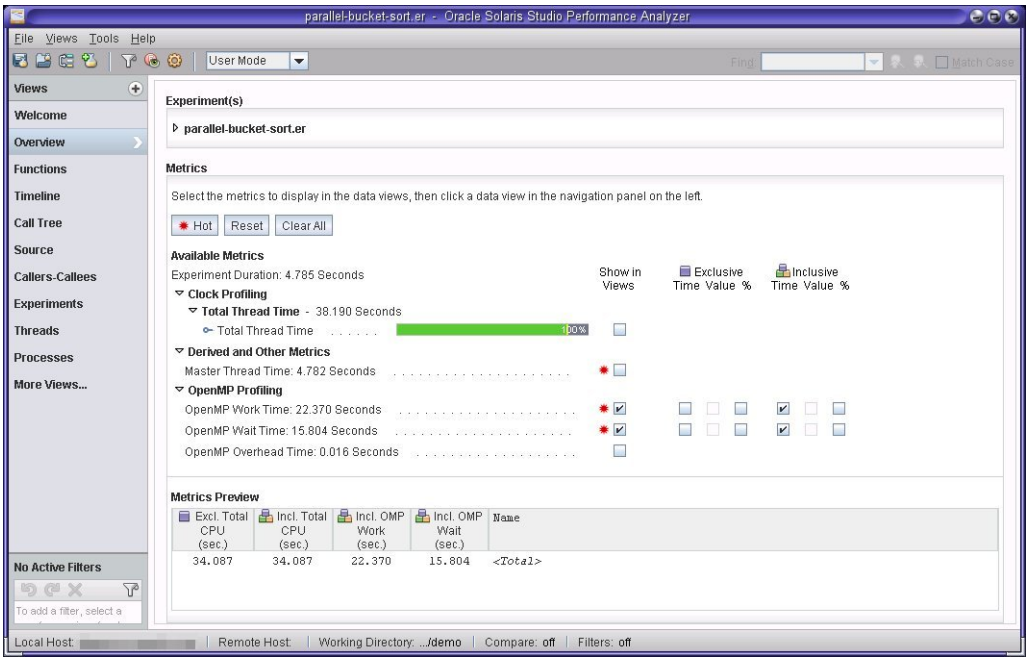
「ようこそ」画面はナビゲーションバーから常に使用可能であるため、パフォーマンスアナライザを使用するときのホームページとして使用することができ、ここからリンクをクリックしてアクティビティーを実行したり、パフォーマンスアナライザのヘルプビューアを開いてドキュメントを読んだりすることができます。

パフォーマンスアナライザの「概要」画面

「概要」には、ロードされた実験のパフォーマンスメトリックが表示されます。「概要」を使用して、ほかのデータビューで調べるメトリックを選択します。

メトリックごとに、ロードされたすべての実験の合計を表す値が表示されます。関連するメトリックの相対値を示す色付きのバーが表示されます。高いアクティビティーメトリックの強調表示マーカーは、CPU 時間に比べて高いアクティビティーを示したメトリックを示します。これらのメトリックは、プログラム内の問題領域を特定するのに役立ちます。

図 3-3 パフォーマンスアナライザの「概要」と新しいナビゲーション



「概要」画面の詳細については、パフォーマンスアナライザのヘルプを参照してください。

タイムラインの改善

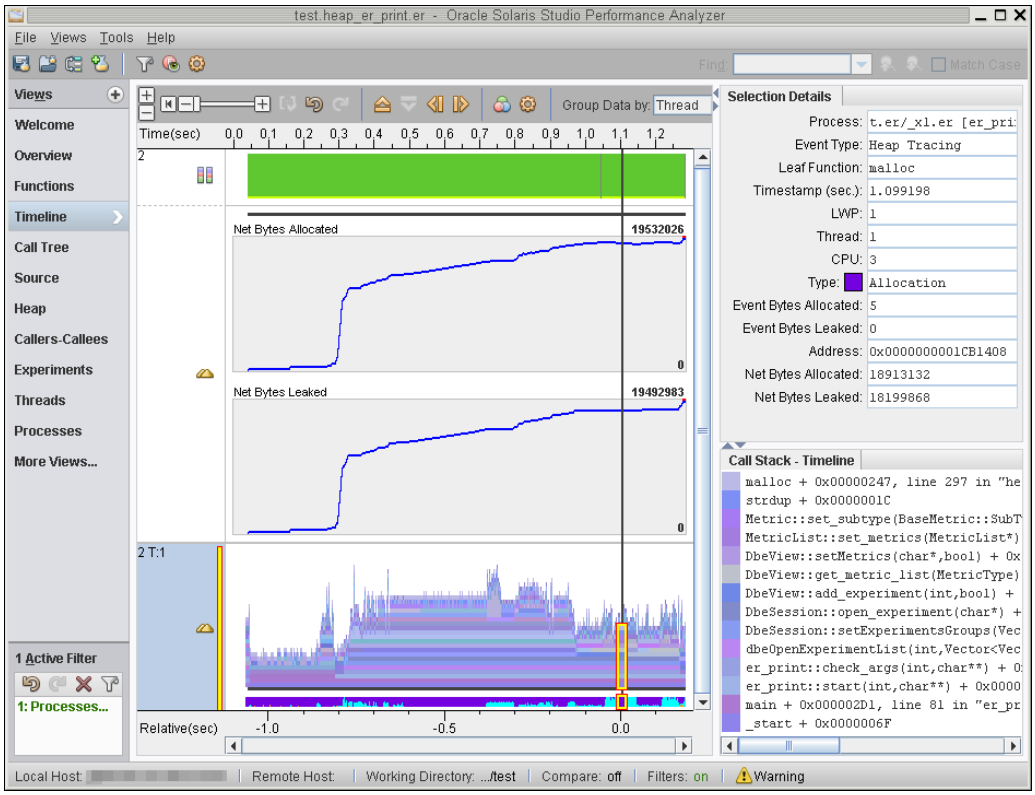
「タイムライン」ビューには、データの調査が簡単になる次の拡張機能があります。

- マウスをクリックしてドラッグすると、ズームやフィルタ処理の**精密な**対象期間を定義するマーカーが設定されます。
- 垂直ズーム用の新しいツールバーコントロールを使用して、ズームしてプロセスとスレッドの最大数を表示します。
- Shift キーを押しながらマウスをドラッグしてズームインします。

タイムライン内のナビゲーションについて詳しくは、ヘルプメニューの**キーボードのショートカットとニーモニック**のタイムラインセクションを参照してください。

次の図は、ヒープトレースデータのタイムラインビューを示します。タイムラインには、ヒープ割り当ておよびリークのグラフと、割り当ておよび解放の呼び出しスタックが表示されます。右側には、現在選択されているメモリー割り当てと呼び出しスタックについての詳細が表示されます。

図 3-4 「タイムライン」ビュー



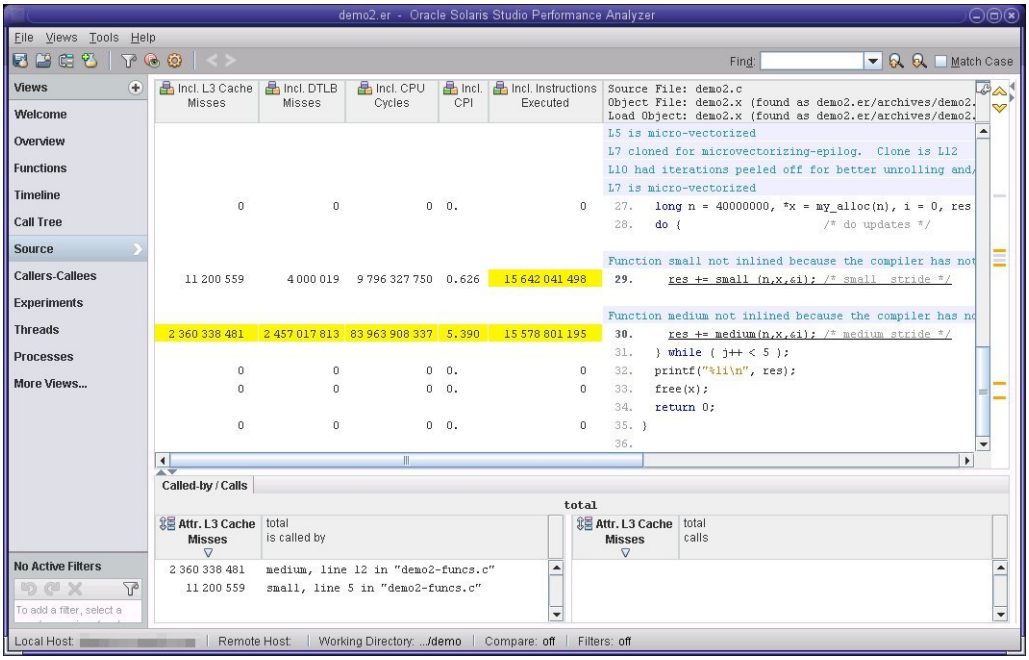
サンプル実験でタイムラインを使用するステップごとの手順については、『Oracle Solaris Studio 12.4: パフォーマンスアナライザチュートリアル 』を参照してください。

「ソース」と「逆アセンブリ」の改善

「ソース」ビューには、ソース言語に基づく構文強調表示が表示されるほか、呼び出し元関数および呼び出し先関数のハイパーリンクなどの多くのナビゲーションの改善が含まれています。

次の図は、ソースコードの 2 つの行に起因するハードウェアカウンタメトリックを示す「ソース」ビューを示します。

図 3-5 ソースデータビューの構文強調表示とナビゲーションの改善



「ソース」ビューと「逆アセンブリ」ビューのその他の変更には、次のものがあります。

- 右マージンのナビゲーションコントロールを使用すると、メトリックが高い行にジャンプすることができます。
- 右クリックメニューおよびハイパーリンクを使用すると、関数の呼び出し元および呼び出し先の「ソース」ビューおよび「逆アセンブリ」ビューの間を簡単にナビゲートできます。
- 「ソース」および「逆アセンブリ」ビューの下部にある「呼び出し元/呼び出し回数」タブを使用すると、呼び出しパスをナビゲートできます。ビュー内の関数を選択してからこれらのタブを使用すると、呼び出し元の関数または呼び出し先の関数にナビゲートできます。「呼び出し元/呼び出し回数」タブで関数をクリックすると、データビュー内でその関数が選択されます。
- ソースファイルが実験より新しい場合、「ソース」ビューに警告が表示されます。

- 「進む」ボタンと「戻る」ボタンを使用すると、「ソース」または「逆アセンブリ」ビューで実行したアクションの履歴をナビゲートできます。

「ソース」ビューおよび「逆アセンブリ」ビューの詳細については、パフォーマンスアナライザのヘルプを参照してください。

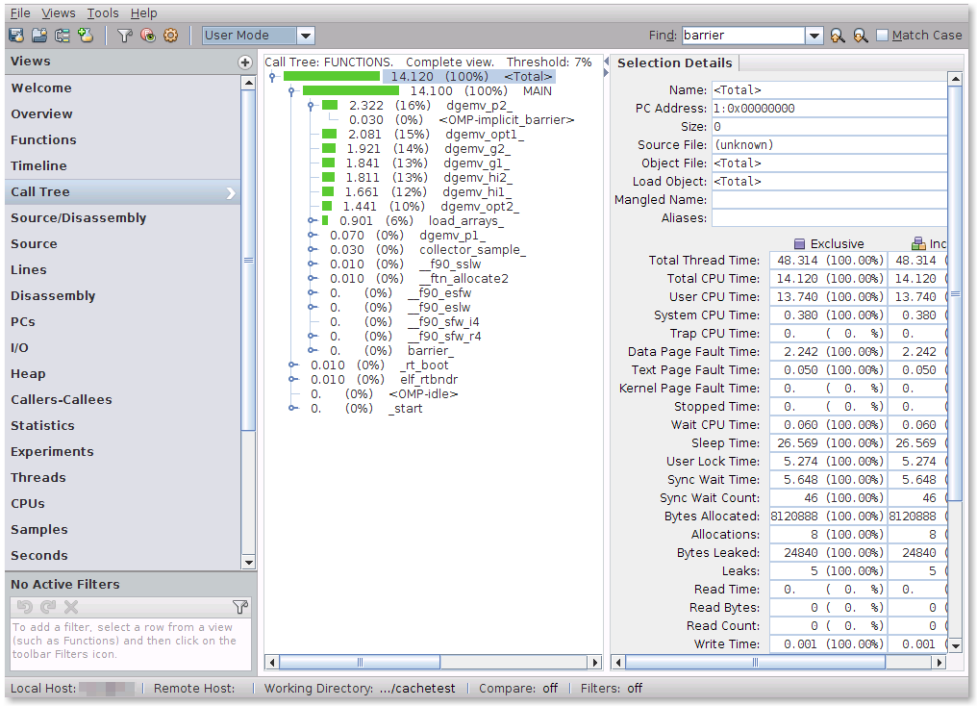
呼び出しツリーの改善

呼び出しツリービューの機能は次のように拡張されています。

- メトリックのパーセントがカラーバーで表示されます。
- 新しいしきい値設定により、メトリックが高いときに分岐を展開する時期を指定できます。
- 「次の参照を表示」および「すべての参照を表示」コンテキストメニューにより、選択された関数を含む分岐を見つけることができます。
- ツールチップおよびカラーバーを表示および非表示にする、右クリックメニューでの新しいアクションが提供されました。
- 別のデータビューで選択した関数は、その関数を含むもっともホットな分岐を展開することによって「呼び出しツリー」ビューに反映され、「呼び出しツリー」ビューで選択した関数もほかのビューで選択されます。
- 名前またはメトリックでソートできます。

「呼び出しツリー」ビューの詳細については、パフォーマンスアナライザのヘルプを参照してください。

図 3-6 「呼び出しツリー」ビュー



Java プロファイリングの改善

Java アプリケーションのプロファイリングが次のように改善されています。

- ターゲットアプリケーションが Java 仮想マシンの場合、Java プロファイリングはデフォルトで必ず有効化されるようになりました。以前は Java アプリケーションのプロファイルを作成するために、`collect` コマンドに `-j on` オプションを指定する必要がありました。
- パフォーマンスアナライザの「実行中プロセスのプロファイル」ダイアログを使用して、Oracle Solaris 上で実行中の Java アプリケーションのプロファイリングを作成できるようになりました。コレクタはプロセスに接続し、JVM および Java プログラムのプロファイリングデータや呼び出しスタックを記録します。
- パフォーマンスアナライザおよび `er_print` は Java スレッド名およびスレッドグループを表示でき、ユーザーはこれらをフィルタ処理に使用できます。

- JDK 8 のプロファイリングがサポートされるようになりました。

Java プロファイリングのチュートリアル『Oracle Solaris Studio 12.4: パフォーマンスアナライザチュートリアル 』の第 3 章「Java プロファイリングの概要」を参照してください。

簡略化したハードウェアカウンタプロファイリング

Oracle ハードウェアの場合、デフォルトのプロファイリングは新しいオプション「-h on」で有効にできるようになりました。このオプションにより、CPI/IPC と待機時間の長いメモリーアクセスを測定するカウンタが選択されます。

ハードウェアカウンタの指定も簡素化されています。Solaris では、[on|high|low] によって、クロックプロファイリングに使用されるレートとほぼ同じプロファイリングレートが選択されます。[on|high|low] オプションにより、適切なオーバーフロー期間を選択するための推測作業の大部分が省かれ、過少または過大な標本収集のリスクを大きく下げることができます。Linux の場合、[on|high|low] オプションは、別名を付けたハードウェアカウンタについてはサポートされますが、raw カウンタでは数値指定が必要です。

ハードウェアカウンタプロファイリングの使用方法の詳細は、『Oracle Solaris Studio 12.4: パフォーマンスアナライザチュートリアル 』の第 5 章「マルチスレッドプログラムでのハードウェアカウンタプロファイリング」を参照してください。

メモリー領域プロファイリングの改善

メモリー領域プロファイリングを使用すると、どのメモリーアドレスでパフォーマンスがもっとも低下しているかを確認できます。メモリー領域プロファイリングは、Oracle Solaris 10 および 11 を実行している SPARC プラットフォームと Oracle Solaris 11.2 を実行している x86 プラットフォームで使用できます。

このタイプのプロファイリングでは、*精密なロードストア* カウンタと呼ばれるハードウェアカウンタが使用されます。

コマンド `collect -h` を使用して、システムで使用可能な精密なロードストアを表示します。これらのカウンタを使用してメモリー領域プロファイリングを実行する方法について詳しくは、`collect`(1) のマニュアルページを参照してください。

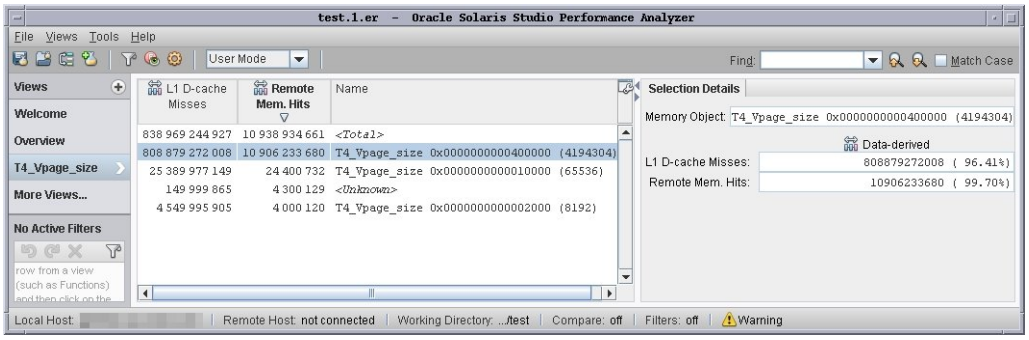
メモリー領域プロファイリングでは次のことが改善されています。

- デフォルトの `-h on` オプションを使用するハードウェアカウンタプロファイリングには、通常、少なくとも 1 つのメモリー領域カウンタが含まれます
- メモリー領域プロファイリングをトリガーするために、精密なカウンタとともに + 記号を使用する必要がなくなりました
- 実験内にデータが存在する場合は、メモリー領域プロファイリングのデータビューを使用できます

パフォーマンスアナライザでメモリー領域プロファイリング実験を開くとき、データビューでカウンタを表示するには、「概要」ページまたは「設定」ダイアログで関連するハードウェアカウンタを有効にする必要があります。「ビュー」メニューからメモリー領域のビューを選択します。

次の図に、メモリーごとのキャッシュミスのページのサンプルデータビューを示します。

図 3-7 メモリー領域プロファイリングメトリックを示すメモリーページビュー



パフォーマンスのコストは、キャッシュ行やメモリーページに起因する可能性があります。このデータをフィルタ処理とともに使用すると、特定の待機時間の長いメモリー参照を行うソースコード行を正確に識別できます。

新しい派生メトリック: CPI および IPC

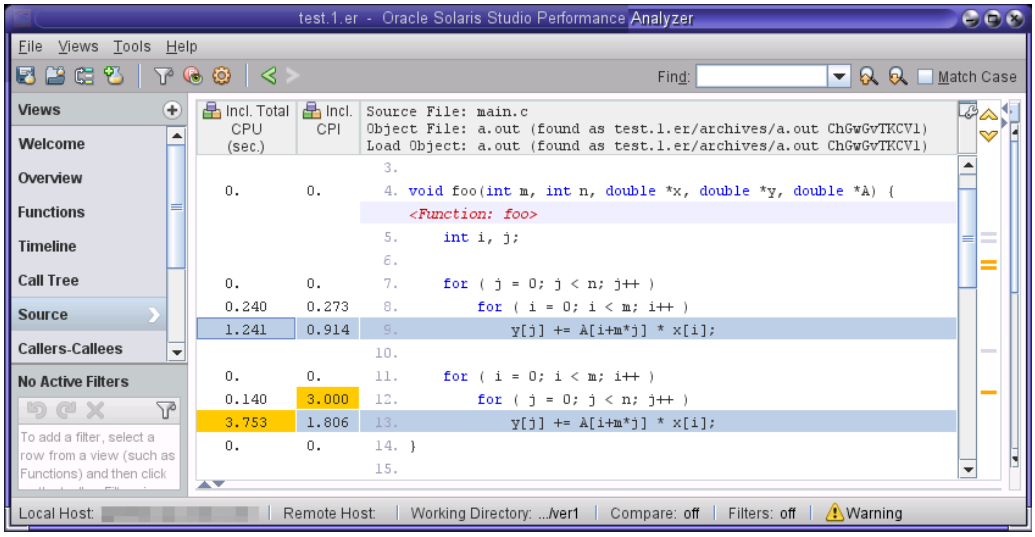
パフォーマンスアナライザでは、アプリケーションの実行が効率的であるか、あるいは非効率であるかを識別するのに役立つ、CPI (命令当たりのサイクル数) および IPC (サイクル当たりの命令数) と呼ばれる新しい派生メトリックが表示されます。CPI および IPC メトリックは、アプリケーション上でハードウェアカウンタプロファイリングを実行して、サイクルと命令のカウン

タを指定した場合に使用できます。たとえば、メトリックを生成するために、コマンド `collect -h cycles,on,insts,on` を使用してプロファイルを作成できます。

非効率な領域では高い CPI または低い IPC が表示されます。プログラムの効率的な領域では低い CPI または高い IPC が表示されます。

次の図は、12 行目の CPI メトリックが高くなっているソースビューを示し、この行の実行が非効率であることを示しています。

図 3-8 ソースビューの CPI メトリック



クロスプラットフォーム解析

パフォーマンスアナライザは、それが稼働する基盤となっているシステムの OS およびアーキテクチャーに関係なく、サポートされるすべての OS およびアーキテクチャー上で記録された実験を読み取ることができます。パフォーマンスアナライザがリモートクライアントシステム上で稼働中の場合、リモートホストに関係なく、サポートされるすべての OS およびアーキテクチャー上で記録された実験を、パフォーマンスアナライザで読み取ることができます。

たとえば、リモートの Windows ラップトップ上でパフォーマンスアナライザを稼働している場合、Oracle Solaris を実行中の x86 ベースのリモートシステムに接続し、Oracle Solaris を実行中の SPARC ベースのシステム上で記録された実験を開くことができます。

パフォーマンスアナライザのリモート使用

パフォーマンスアナライザからリモートホストに接続して、リモートシステムで実験を検査できます。リモート接続を使用する場合、実験が記録された同じ環境で実験を検査できるという利点があります。

クライアントシステムの要件:

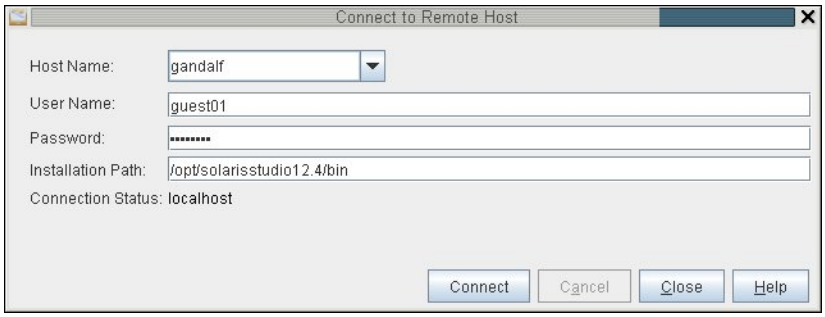
- クライアントオペレーティングシステムは、Mac OS X、Windows、Linux、または Oracle Solaris である必要があります。
- クライアントシステムのパスに Java 1.7 がある必要があります。
- クライアントシステムにインストールされたパフォーマンスアナライザのバージョンは、リモートシステムにインストールされた Oracle Solaris Studio ツールのバージョンと一致する必要があります。

リモートホストの要件:

- リモートホストのオペレーティングシステムは Oracle Solaris *または Linux*である必要があります。
- リモートホストはセキュアシェル (SSH) デーモン sshd を実行している必要があります。
- Oracle Solaris Studio 12.4 ソフトウェアは、リモートホストでアクセスできる必要があり、ツールまでのパスがわかっている必要があります。
- ホストにユーザーアカウントが必要です。

Oracle Solaris Studio がインストールされている Linux および Oracle Solaris システムで、「ファイル」>「リモートホストに接続」を選択して「リモートホストに接続」ダイアログを開くことができます。

図 3-9 「リモートホストに接続」ダイアログ



Oracle Solaris Studio のこのリリースがインストールされていないシステム上で使用する場合、パフォーマンスアナライザのクライアントディストリビューションは、完全な Oracle Solaris Studio のインストールの `lib/analyzer` ディレクトリにある `RemoteAnalyzer.tar` ファイルにあります。パフォーマンスアナライザのクライアントディストリビューションをインストールするには、システムに tar ファイルをコピーして解凍します。`RemoteAnalyzer` ディレクトリには、Windows 用、Mac OS 用、Linux 用、および Solaris 用のスクリプトが格納され、`README.txt` ファイルには、機能および使用法が記載され、`lib` サブディレクトリには、リモート操作に必要なコンポーネントが格納されています。

システムに合ったスクリプトを実行して、クライアントシステム上でパフォーマンスアナライザを起動します。パフォーマンスアナライザが開始したときは、リモートで動作する機能だけが有効になっている開始画面が表示されます。詳細は、`RemoteAnalyzer/README.txt` ファイルおよびパフォーマンスアナライザのヘルプメニューを参照してください。

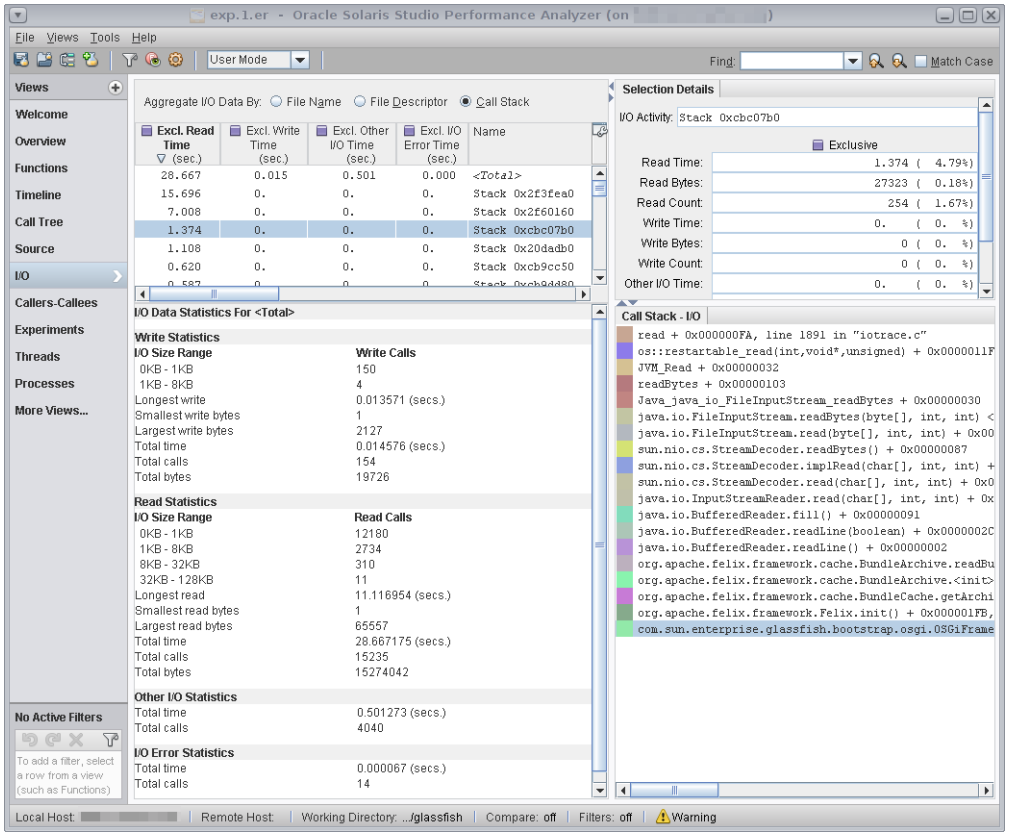
新しい I/O データビュー

新しい「I/O」ビューは、アプリケーション内の I/O パターンを識別し、アプリケーションのパフォーマンスに影響する「I/O」のボトルネックを特定するのに役立ちます。「I/O」ビューは、パフォーマンスアナライザあるいは `collect -i on` コマンドを使用して、アプリケーションの I/O トレースデータをプロファイルした場合に利用できます。

「I/O」ビューには、ファイル名、ファイル記述子、または呼び出しスタックで集計された読み取りおよび書き込みデータが表示されます。また、これを使用して、データから I/O イベントをフィル

タ処理することもできます。新しい「経過時間」ビューを使用して、I/O 操作の期間を解析することができます。新しい「データサイズ」ビューを使用して、I/O 操作の分布をバイトサイズごとに表示することができます。「I/O」ビューの詳細については、パフォーマンスアナライザのヘルプを参照してください。

図 3-10 I/O データビュー

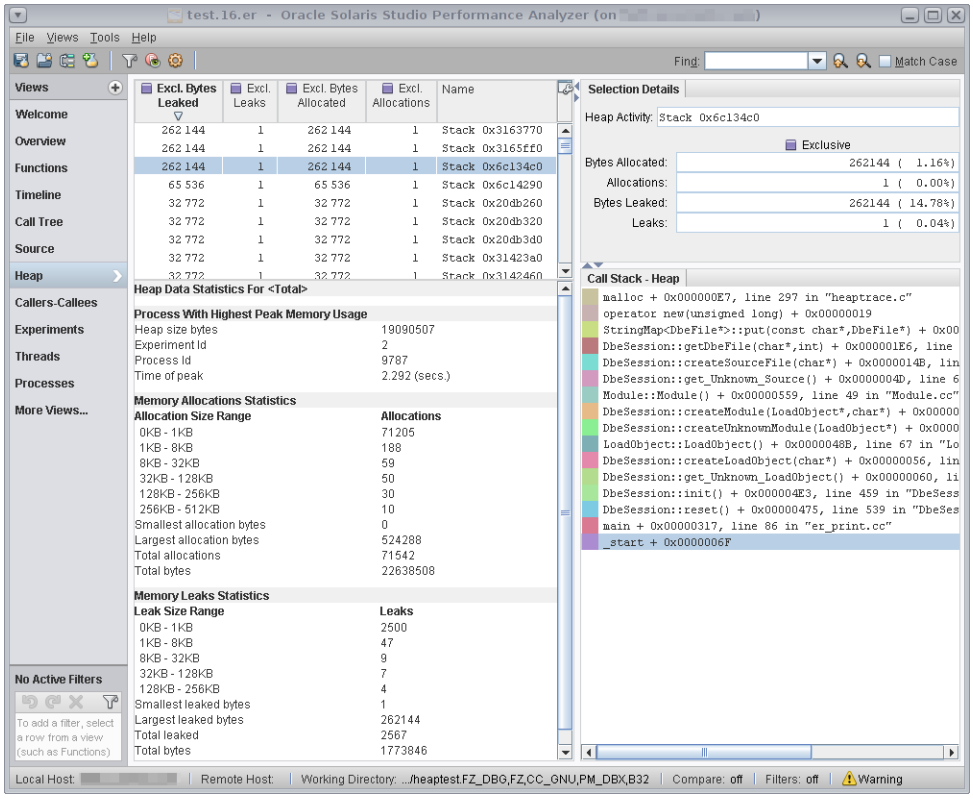


新しいヒープデータビュー

新しい「ヒープ」ビューは、プログラム内で考えられるメモリーリークを示します。このビューは、パフォーマンスアナライザあるいは `collect -H on` コマンドを使用して、アプリケーションのヒープトレースデータをプロファイルした場合に利用できます。

「ヒープ」ビューには、メモリーリークの可能性を示すメモリー割り当てメトリックを伴う呼び出しスタックのリストが表示されます。タイムラインには、割り当てられたヒープのサイズが時間の関数として表示されるようになりました。割り当ておよびリークの分布をバイト数によって表示する、新しい「データサイズ」ビューが利用できます。新しい「経過時間」ビューを使用して、割り当ての期間を解析することができます。「ヒープ」ビューの詳細については、パフォーマンスアナライザのヘルプを参照してください。

図 3-11 新しいヒープデータビュー



パフォーマンスアナライザへのその他の変更

パフォーマンスアナライザツールのその他の機能拡張には、次のものがあります。

- `analyzer` コマンドの `-u` または `--userdir` 引数を使用して、パフォーマンスアナライザのユーザー設定を格納するパスを指定できるようになりました。
- 実験の比較と集計が、さらに実行しやすくなりました。実験を比較するとき、絶対値またはデルタを選択して、メトリック値の変化を表示することができます。
- 共有ライブラリまたは Java クラスによるパフォーマンスメトリックの集計が大幅に改善され、これをサポートするダイアログの名前が「表示/非表示/API のみ」から「ライブラリとクラスの可視性設定」に変更されました
- 出力オプションが「ファイルにエクスポート」に置き換わり、多くの出力がサポートされるようになりました。一部のビューでは、ASCII テキストテーブル、コンマ区切りリスト、または HTML テーブルとしてエクスポートできます。すべてのエクスポートは、パフォーマンスアナライザが稼働中のマシン上のファイルシステムに対して行われます。リモートモードで実行中の場合、データはリモートマシン上のファイルにエクスポートされます。
- クロックプロファイリング、MPI トレース、およびヒープトレースのメトリック名が変更されました。詳細は、`collect` (1) のマニュアルページを参照してください。
- パフォーマンスアナライザに Java スレッド名およびスレッドグループが表示され、これらをフィルタ処理に使用できます。
- パフォーマンスアナライザは、実行中のプロセスへの接続と、このプロセスのプロファイリングをサポートするようになりました。「ファイル」>「実行プロセスのプロファイル」を選択するか、開始画面の「実行プロセスのプロファイル」をクリックします。これまで実行中のプロセスに接続するには、`collect` コマンドまたは `dbx collector` コマンドを使用するしかありませんでした。
- パフォーマンスアナライザから Oracle Solaris カーネルをプロファイルできるようになりました。「ファイル」>「カーネルのプロファイル」を選択するか、開始画面の「カーネルのプロファイル」をクリックします。これまでカーネルをプロファイルするには、`er_kernel` コマンドを使用するしかありませんでした。
- パフォーマンスアナライザが設定を永続的に保存するための新しい方法が導入されました。パフォーマンスアナライザを終了するとき、メトリックやビューなどのほとんどの設定は維持されるため、同じ実験をふたたび開くと、実験は最後に閉じたときと同じように表示されます。選択された設定を構成ファイルに保存しておき、同じ実験または異なる実験を「実験を開く」ダイアログボックスから開いたとき、これらの実験に設定を適用することができます。パフォーマンスアナライザは、`er_print` によって読み取られる `.er.rc` ファイルに設定を保存することもできます。

コマンド行ツールの変更点

このセクションでは、コマンド行のさまざまなパフォーマンス解析ツールに対して行われた変更について説明します。

データ収集ツールに対する変更

データ収集ツールには、collect コマンド、dbx collector コマンド、および er_kernel コマンドがあります。これらの各ツールは、プログラムをプロファイルしてデータを収集し、パフォーマンスアナライザまたは er_print によって読み取ることが可能な実験を作成するために使用されます。

次の変更は、これらのツールに共通するものです。

- ハードウェアカウンタおよびスタックの巻き戻しについて、新しいプロセッサ (SPARC T5、SPARC M5、SPARC M6、SPARC 64 X、SPARC 64 X+、Intel Ivy Bridge、および Haswell) がサポートされます。
- クロックプロファイリングは、ハードウェアカウンタのプロファイリングが指定されていない場合もデフォルトで有効です。
- アーカイブをオンにする設定は、アーカイブのコピーと同じ設定です。つまり、collect および er_kernel の場合、-A on が -A copy と同じになったことを意味します。dbx collector の場合、collector archive on が collector archive copy と同じであることを意味します。
- プロファイリングできる Oracle Solaris のスレッドの最大数は 32768 になりました。

collect ユーティリティーの変更

collect ユーティリティーは、アプリケーションの実行中にアプリケーションをプロファイルしてデータを収集し、パフォーマンスアナライザまたは er_print によって読み取り可能な実験を作成するために使用するツールです。

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。

- I/O トレースは、新しい -i フラグを使用してサポートされます。
- ターゲットが JVM の場合、Java プロファイリングがデフォルトで常に有効化されます。-j on を指定する必要はなくなりました。

- 実行中のプロセスからデータを収集するための -P オプションが、Linux システム上でサポートされるようになりました。これはシングルスレッドアプリケーションでのみ機能することに注意してください。
- カウントデータを収集するための -c オプションが、Linux システム上でサポートされるようになりました。
- ハードウェアカウンタ処理で、複数の -h 引数とデフォルトのカウンタ設定がサポートされるようになりました。環境変数 SP_COLLECTOR_HWC_DEFAULT を設定して、ハードウェアカウンタカウンタプロファイリングをデフォルトで有効にすることができます。
- ハードウェアカウンタベースのメモリー領域プロファイリングが、SPARC および x86 システム上の精密なカウンタについてデフォルトで有効になりました。メモリーアドレスを取得するために、プラス記号 (+) は必要なくなりました。詳しくは、31 ページの「メモリー領域プロファイリングの改善」を参照してください。
- collect -F =expr は、式をプロセス系統に一致させなくなりました。
- -P を使用して Java プログラムに接続するとき、-j on オプションを指定するようにしてください。

dbx collector の変更点

dbx collector は、パフォーマンスデータ収集に使用できる dbx デバッガのサブコマンドです。詳細は、collector(1) のマニュアルページを参照してください。

すべてのデータ収集ツールに共通する変更点に加え、dbx collector コマンドは、このリリースで次のように変更されています。

- detach または実験終了後に別の実験を開始できます。
- 接続は、Linux システム上でシングルスレッドのネイティブアプリケーションについてサポートされています。接続の制限事項の詳細は、『Oracle Solaris Studio 12.4: リリースノート』の「dbx attach を使用したプロファイリング (collect -P)」を参照してください。
- dbx collector は、次の新しいコマンドをサポートします。
  - collector iotrace on - I/O トレースをオンに指定します。
  - collector duration - 実験を実行する時間範囲を指定します。
  - collector java - Java プロファイルデータを収集するかどうかを指定します。デフォルトは off です。JVM に接続するか JVM を起動するかにかかわらず、Java をプロファイリングするときは常にこのコマンドを使用するようにします。

- `collector pausesig` - データ収集の一時停止または再開に使用するシグナルを指定します。
- `collector samplesig` - 標本の記録に使用するシグナルを指定します。
- `collector hwprofile addcounter` - ハードウェアカウンタのオーバーフロープロファイリング用の追加カウンタを指定します。

`er_kernel` ユーティリティーの変更

`er_kernel` コマンドは Oracle Solaris カーネルをプロファイルし、パフォーマンスアナライザまたは `er_print` で調査できる実験を生成します。

すべてのデータ収集ツールに共通する変更点に加え、`er_kernel` ユーティリティーは次のように変更されています。

- ユーザーのサブ実験のクロックプロファイリングメトリックは `collect` 実験と同じように記録されますが、ユーザー CPU 時間とシステム CPU 時間のみが記録され、待ち時間は記録されません。
- カーネル初期実験で報告されるハードウェアカウンタメトリックの名前に `K_` が付けられます。ユーザーのサブ実験のメトリックでは `collect` 実験のメトリックと同じ名前が使用されます。
- カーネルデータ領域プロファイリングは、精密カウンタについて、DTrace バージョン 1.8 以降を持つ Solaris SPARC システムでサポートされます。
- 初期の実験のクロックプロファイリングは `collect` 実験と同じように記録されますが、ただしカーネル CPU 時間メトリックしかありません。ハードウェアカウンタプロファイリングはデフォルトで有効にされません。
- カーネル呼び出しスタックの記録が改善されました。

`er_print` ユーティリティーの変更

`er_print` ユーティリティーは、パフォーマンスアナライザで提供されるデータビューのプレーンテキストバージョンを生成します。その出力は、標準出力に表示されます。

`er_print` ユーティリティーはこのリリースでは次のように変更されています。

- IO トレースデータは、`ioactivity`、`iodetail`、および `iocallstack` の 3 つの新しいレポートで使用できます。`ioactivity` レポートでは、データはファイル名によって集計さ

れます。iodetail レポートでは、データは開くときにファイル記述子によって区切られます。iocallstack では、データは共通呼び出しスタックによって集計されます。

- ヒープトレースデータは新しいレポート heap および heapstat で使用できます。共通呼び出しスタックによって集計されたすべての割り当ておよびリークを出力するには、heap を使用します。プロセスの存続期間全体でのピーク使用量を含むヒープ使用量の統計のサマリーを出力するには、heapstat を使用します。
- アプリケーションをプロファイルして、cycles および insts ハードウェアカウンタを収集する場合、新しい派生メトリックの CPI (命令当たりのサイクル数) および IPC (サイクル当たりの命令数) を使用できます。CPI および IPC はメトリックとして指定できます。
- 新しいレポートの overview には、実験のサマリー情報が表示されます。
- object_list からの表示には、インデックス (他のデータビューでは「PC」として表示される)、オブジェクトへのフルパス、および共有ライブラリ関数の可視性の設定が含まれるようになりました。
- er_print は、サポートされるほかのすべてのアーキテクチャーで稼働しつつ、あらゆるアーキテクチャー上で記録された実験を読み取ることができます。
- 新しい printmode *string* コマンドは、以前のリリースと同じ ASCII 形式 (string = table)、区切り文字で区切られたリスト (string = *X*、X は単一文字の区切り文字)、または HTML 形式のテーブル (string = html) のいずれかをサポートします。printmode コマンドは、er.rc ファイル内に指定できます。
- ユーザーの er.rc 処理中や、入力コマンドとして、またはスクリプト内のコマンドとして、machinemodel が指定された場合、あるいはロードされた実験が machinemodel を記録していた場合、マシンに固有のメモリーオブジェクトが作成されます。
- er_print は、Java スレッド名およびスレッドグループを表示し、これらによるフィルタ処理が可能です。
- 比較モードでは、比較データを絶対値またはデルタで表示することで、実験と実験の間でのメトリック値の変化を表示することをサポートしています。
- setpath ディレクティブは .er_rc ファイル内で許可されなくなりました。以前のリリースで作成した .er_rc ファイルに setpath ディレクティブがある場合、setpath 行を addpath に変更するようにします。

ほかのコマンドに対する変更

er_archive コマンドは、このリリースでは次のように変更されました。

- er_archive (およびその他すべてのツール) は、シンボリック情報がサイドファイルに記録された実行可能ファイルを処理することができます。
- er_archive は共有オブジェクトのみコピーし、アーカイブファイルを生成しなくなりました。-A フラグはサイレントに無視されます。
- er_archive は、ソース、オブジェクト、およびファイルのアーカイブを指定する -s *type* フラグをサポートします。*type* が all の場合、検出可能なすべてのソースファイルがアーカイブされ、*type* が used の場合、実験によって参照されたソースファイルのみがアーカイブされます。
- er_archive は、-s および -m の引数を指定するために使用できる、環境変数 SP_ARCHIVE_ARGS を調査します。実験が記録されたときに SP_ARCHIVE_ARGS 変数が設定された場合、自動的に実行される er_archive がソースアーカイブを実行できます。これは er_archive が手動で実行されたときに処理されます。

実験の変更点

パフォーマンスデータ実験は次のように変更されました。

- 実験の形式が変更され、そのバージョン番号が 12.4 になりました。
- パフォーマンス解析ツールのこのリリースでは、次の実験のバージョンを読み取ることができます。

10.2	プレリリース Oracle Solaris Studio 12.3 で作成
12.3	リリース済みの Oracle Solaris Studio 12.3 で作成
12.4	今回の Oracle Solaris Studio 12.4 で作成

  バージョン 10.2 以前の実験を使おうとすると、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メッセージがツールに表示されます。
- 詳細な情報を表示するためにクロックプロファイリングメトリックが再編成され、一部メトリックは名前が変更され、異なるデフォルトが設定されました。詳細は、collect のマニュアルページを参照してください。メトリックの名前と省略名が一覧で表示されています。

♦ ♦ ♦ 第 4 章

4 コード分析ツール

コード分析ツールスイートは、メモリーリークやアクセス違反といった一般的なコーディングエラーを検出することでアプリケーションの信頼性および安定性を確保し、開発者が優れたコードを少ないエラーで迅速に記述できるようにします。

この章では、Oracle Solaris Studio のこのリリースにおけるコード分析ツールの新機能や変更点について説明します。次のセクションで構成されています。

コード分析ツールについて

コード分析ツールを使用すると、静的分析、動的分析、およびコードカバレージ分析を使用して、メモリーリークやメモリーアクセス違反など一般的なコーディングエラーの多くを検出することにより、アプリケーションの信頼性を高めることができます。Previse ではコンパイル時に静的分析を実行し、Discover ではアプリケーション実行時に動的分析を実行することにより、コード品質の問題を特定します。Uncover ツールは、コードカバレージデータを分析して、テストスイートの対象にならない関数に関する情報と、それらの関数を対象とすることで得られる利点について通知します。

コードアナライザグラフィックツールまたは新しい `codean` コマンド行ユーティリティーを使用すると、3 種類の分析を表示してアプリケーションの脆弱性について包括的な見地を得ることにより、妥当性および信頼性を向上させることができます。

新しいコマンド行コードアナライザツール codean

新しいユーティリティー codean を使用すると、コードアナライザグラフィカルツールではなくコマンド行を使用して、Previse、Discover、および Uncover によって生成されたコード分析データを表示することができます。codean ツールは、コードアナライザと似た機能を提供しますが、グラフィカル環境を利用できないシステムで使用したり、コマンド行を利用したいときに使用したりすることができます。codean ツールは、自動化スクリプトで使用することもできるほか、コードアナライザツールでは利用できない機能も備えています。

codean ツールには、次の独自機能があります。

- 結果を HTML およびテキスト形式で表示します。
- チェックしているバイナリの新しいバージョンと比較するために、ツールレポートを保存します。
- 現在のレポートを保存されたレポートと比較して、新しいエラーのみを表示します。46 ページの「--whatisnew オプションの使用」を参照してください。
- 現在のレポートを保存されたレポートと比較して、修正されたエラーのみを表示します。47 ページの「--whatisfixed オプションの使用」を参照してください。
- 複数の動的エラーチェックの実行からすべてのエラーを表示します。たとえば、テストスイートが Discover 計測済みのバイナリで実行された場合、すべてのデータを 1 つのテキストレポートに集約します。
- ディレクトリ下にあるすべてのレポートについてサマリー HTML ページをコンパイルします。47 ページの「codean を使用したサマリー HTML ページの生成」を参照してください。

codean の詳細については、codean(1) のマニュアルページを参照してください。

--whatisnew オプションの使用

以前に保存されたツールレポートと比較して新しいエラーのみのレポートを生成するには、--whatisnew オプションを使用します。たとえば、コード分析ツールを使用して、ツール採用時のソースベースの状態の凍結済みコピーを作成します。次に --whatisnew を使用して、ソースベースに対する進行中の変更によって新しいセキュリティー脆弱性が生じていないことを確認することができます。

次は、--whatisnew を使用して新しいエラーのみを表示する例です。

```
%codean --whatisnew a.out
STATIC report of a.out showing new issues:
Compare the latest results against a.out.analyze/history/09:58:35May152013...
MEMORY LEAK 1 : 1 block left allocated on heap with a total size of 400 bytes
    sample1()  <sample1.c : 20>
        17:    {
        18:          global = (int *)malloc(100);
        19:          int *p = malloc(100*sizeof(int));
        20:=>          int *q = malloc(100*sizeof(int));
        22:          add_0_1_put_in_2(p);-
PREVISE SUMMARY:
    0 new error(s), 0 new warning(s), 1 new leak(s) in total
```

--whatisfixed オプションの使用

--whatisfixed オプションは、以前にキャッシュされたツールレポートで発生したエラーのみを表示するレポートを生成することで、--whatisnew オプションを補完します。開発チームまたは品質保証チームは、このオプションを使用すると、セキュリティーバグの修正を追跡することができます。

次は、--whatisfixed を使用して修正されたエラーのみを表示する例です。

```
% codean --whatisfixed a.out
STATIC report of a.out showing fixed issues:
Compare the latest results against a.out.analyze/history/10:02:05May152013...
MEMORY LEAK 1 : 1 block left allocated on heap with a total size of 400 bytes
(Warning: Source files have changed. Source code shown below may not be accurate.)
    sample1()  <sample1.c : 20>
        17:    {
        18:          global = (int *)malloc(100);
        19:          int *p = malloc(100*sizeof(int));
        20:=>          int *q = malloc(100*sizeof(int));
        21:          free(q);
PREVISE SUMMARY:
    0 fixed error(s), 0 fixed warning(s), 1 fixed leak(s) in total
```

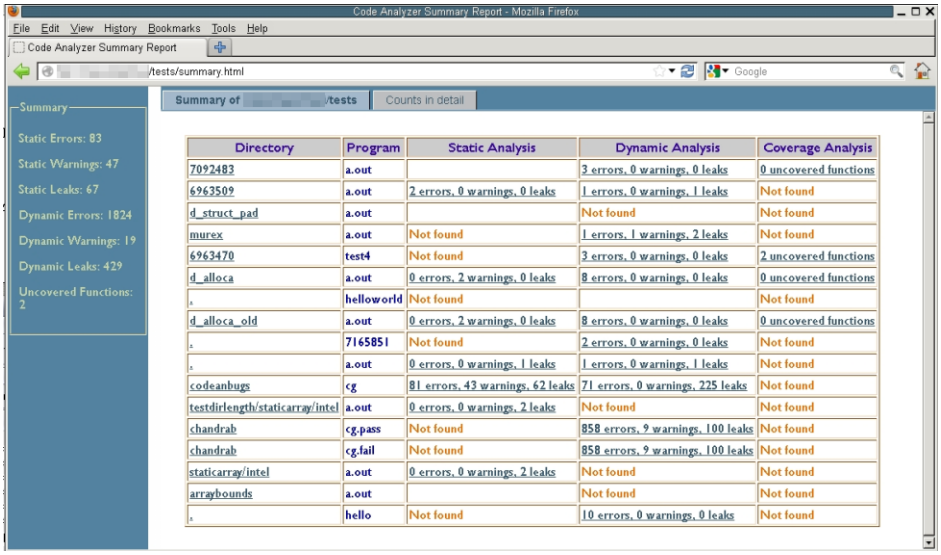
codean を使用したサマリー HTML ページの生成

ディレクトリツリー内の複数のバイナリに対して以前実行したコード分析ツールによって生成されたすべての情報のサマリーレポートを作成することができます。HTML ページでは、複数のバイナリに関する静的レポート、動的レポート、カバレージレポートからのすべてのエラーがわかりやすい表に整理されています。

たとえば、最上位ディレクトリ `./tests` で HTML ページを作成する場合は、`codean ./tests` を使用します。

次の図は、`/tests` ディレクトリ下にあるすべてのレポートに関して生成された `summary.html` ページの例です。

図 4-1 サマリー HTML レポート



コードアナライザの変更点

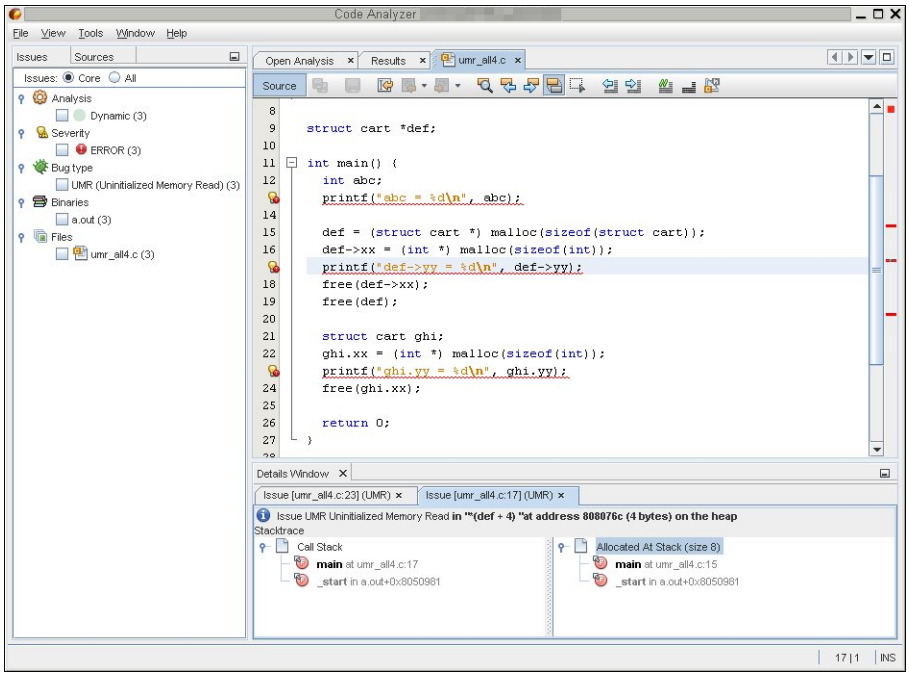
次の共通機能がすべてのコードアナライザコンポーネントツールに追加されました。詳細は、コードアナライザ内のヘルプと『Oracle Solaris Studio 12.4: コードアナライザユーザーズガイド』を参照してください。

- Oracle Enterprise Linux で利用できるようになりました。
- コードアナライザコンポーネントの計測エンジンで 64 ビットテクノロジを使用することにより、エンタープライズアプリケーションサポートが向上しました。
- 計測エンジンランタイムを大幅に改良しました。

- エンジンのメモリーフットプリントを大幅に削減することにより、小規模マシンで大規模アプリケーションを計測できます。
- SPARC T5 および M5 プロセッサのサポートが追加されました。
- Intel Ivy Bridge プロセッサのサポートが追加されました。
- 動的エラーレポートで変数名情報の表示が追加されました。

次の図では、変数名を表示するコード例のコードアナライザ GUI を示しています。

図 4-2 変数名を表示したコードアナライザ GUI のスクリーンショット



コードアナライザ全般についての詳細は、コードアナライザ内のヘルプ、『Oracle Solaris Studio 12.4: コードアナライザユーザーズガイド 』、『Oracle Solaris Studio 12.4: 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。

新しい Previse 静的分析機能

このリリースで次の機能が Previse 静的分析ツールに追加されました。

- 静的コードエラーを収集するために、コードをコンパイルするときに `-xanalyze=code` オプション (EOL です) の代わりに `-xprevise` オプションを使用できるようになりました。
- 静的分析を Oracle Enterprise Linux で利用できるようになりました。
- 誤検出を排除することにより、正確性を大幅に向上。

新しい Discover の機能

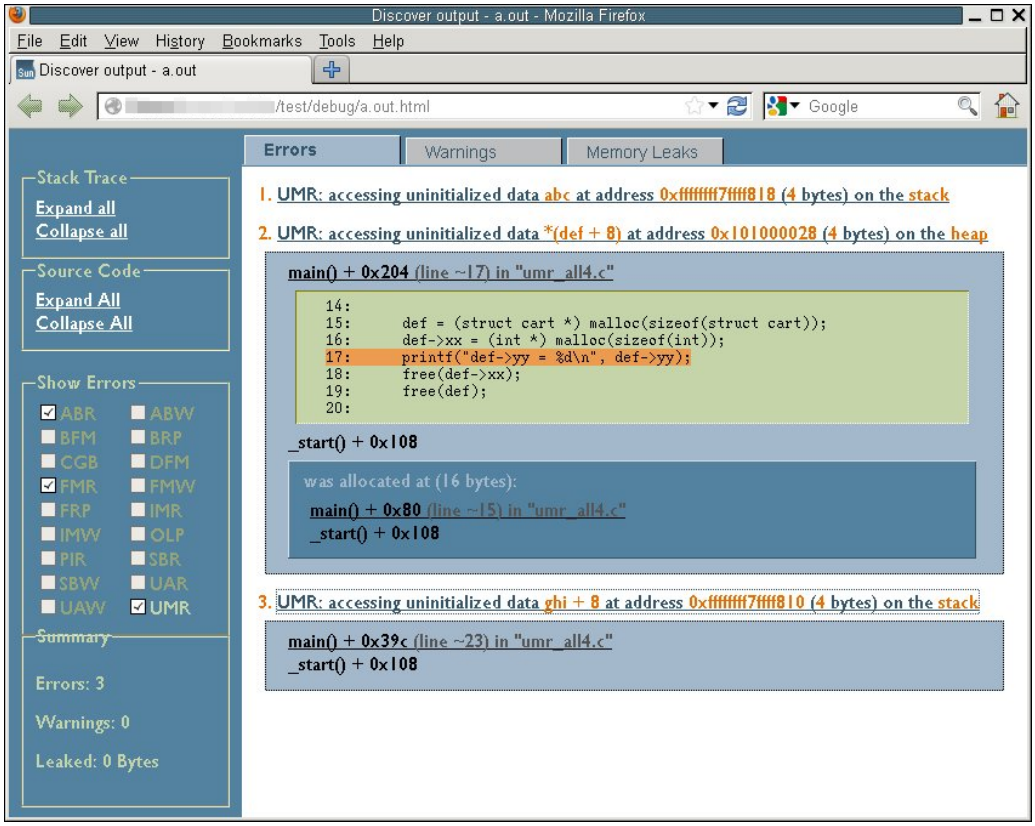
このリリースで Discover メモリー分析ツールに追加された機能を、次に示します。詳細は、`discover` (1) のマニュアルページ、および『Oracle Solaris Studio 12.4: Discover および Uncover ユーザーズガイド 』を参照してください。

- Discover では、`-c[- | lib[:scope...] | file]` オプションを使用して、実行可能ファイルまたはライブラリの一部をチェックすることができます。詳細は、『Oracle Solaris Studio 12.4: Discover および Uncover ユーザーズガイド 』の「ライブラリまたは実行可能ファイルの部分的な検査」を参照してください。
- 新しい Discover API は、要求時にメモリーリークおよびメモリー使用量を表示します。詳細は、51 ページの「新しい Discover API」を参照してください。
- Discover を Oracle Enterprise Linux で利用できるようになりました。
- Discover は、`-F both` オプションを使用すると、メモリーアクセスデータを子と親の両方のプロセスから追尾して収集することができます。これは新しいデフォルトです。
- Discover エラーレポートは、テストスイートの場合と同様に、ターゲットバイナリの複数回実行をサポートします。エラー報告フォーマットは、新しいコマンド行 `codean` ユーティリティーと連携して動作します。
- `mmap(2)` で割り当てられたコードのメモリーエラーをチェックする新機能。
- 変数名、行番号、およびアドレスを強調表示するように改善された HTML レポート。
- メモリー破損が発生するようなメモリーエラー時に、変数の名前が表示されるようになりました。
- Discover は、デフォルトで静的型配列の範囲外エラーをキャッチできるようになりました。詳細は、『Oracle Solaris Studio 12.4: Discover および Uncover ユーザーズガイド 』の「メモリーアクセスエラーと警告」を参照してください。

- `-i datarace` オプションを使用すると、Discover を使用したバイナリ計測で検出された競合のデュアルスタックトレースを報告するようになりました。
- 大規模ファイルのサポートが追加されました。

次の図は、Discover を使用して HTML 強調表示と変数名のレポートを生成する例です。

図 4-3 Discover の HTML 強調表示および変数名の例



新しい Discover API

6 つの新しい Discover API がコード分析ツールに追加されました。アプリケーションが使用しているメモリーが多すぎる場合は、これらの API を使用して、アプリケーションが終了する前に

メモリーリークが発生する場所を特定することができます。また、API を使用して、使用中のメモリーブロックをすべて検索できます。

これらの API は、アプリケーションソースに直接挿入したりデバッグセッション中に呼び出したりして、任意の開始点からの新規および合計のメモリー使用量や新規および合計のメモリーリークをいつでも報告することができます。特に、長時間実行または常時実行のエンタープライズサーバーアプリケーションで役立ちます。

次の API がコード分析ツールに追加されました。

- `discover_report_all_inuse()`
- `discover_mark_all_inuse_as_reported()`
- `discover_report_unreported_inuse()`
- `discover_report_all_leaks()`
- `discover_mark_all_leaks_as_reported()`
- `discover_report_unreported_leaks()`

例 4-1 `discover_report_unreported_leaks()` API の使用

次は、`discover_report_unreported_leaks()` API を使用する例です (Discover でバイナリを実行してから `dbx` を使用してこの API を実行しています)。

```
%discover -w - a.out
% dbx a.out
 (dbx) stop in foo2
 (dbx) run
 (dbx) call discover_report_unreported_leaks()
 ******** discover_report_unreported_leaks() Report ********

  1 allocation at 1 location left on the heap with a total size of 1 byte

      LEAK 1: 1 allocation with total size of 1 byte
          foo1() + 0x5e  <api_example.c:7>
                    4:    #include <discoverAPI.h>
                    5:
                    6:    void foo1() {
                    7:=>    char *x = (char *) malloc(sizeof(char));
                    8:      *x = 'a';
                    9:      printf("x = %c\n", *x);
                   10:      /*free(x);*/
          main() + 0x1a  <api_example.c:21>
                   18:    }
                   19:
                   20:    int main() {
```

```
                21:=>    foo1();
                22:      foo2();
                23:      return 0;
                24:    }
        _start() + 0x71

*******************************************************
```

各 Discover API の詳細と、その使用方法については、Discover のヘッダーファイルおよび『Oracle Solaris Studio 12.4: Discover および Uncover ユーザーズガイド 』の「discover API」を参照してください。

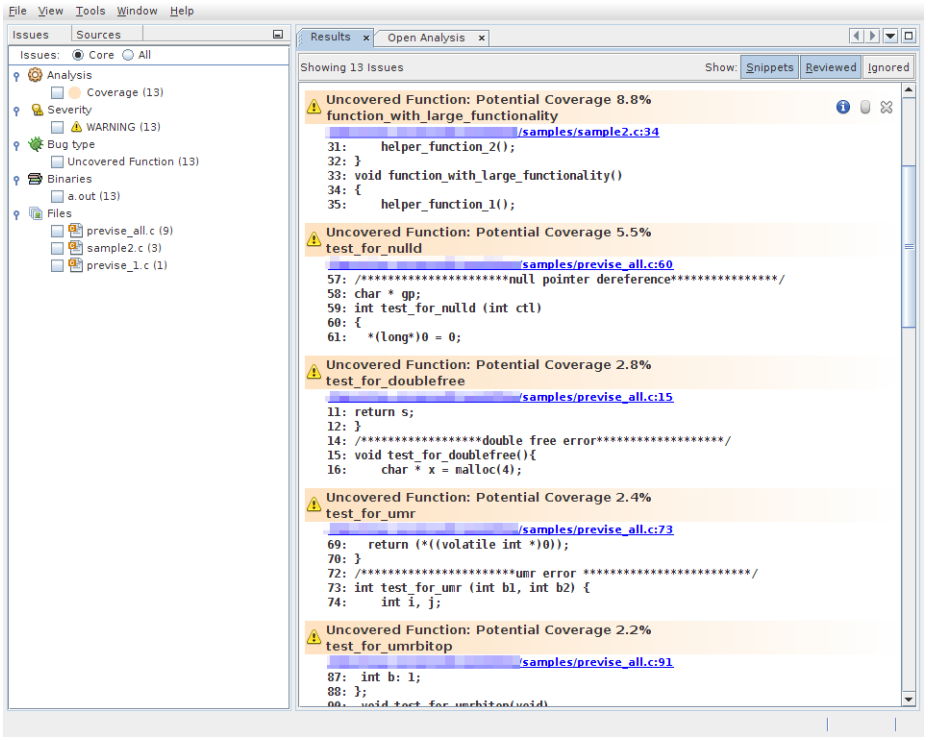
新しい Uncover の機能

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。

- Uncover を Oracle Enterprise Linux で利用できるようになりました。
- C++ アプリケーションのカバレージレポートの妥当性を向上しました。
- 長時間実行するサーバーアプリケーションで通常見られる異常なプロセス終了のサポート。
- Uncover 計測および実行時の時間およびメモリーフットプリントのオーバーヘッドを削減しました。
- Uncover はプログラムの実行中にカバレージおよびプロファイリングのデータを取得できるようになりました。
- 大規模ファイルのサポートが追加されました。

次の図は、コードアナライザ GUI で Uncover を使用する例です。

図 4-4 コードアナライザ GUI における Uncover の結果の例



詳細は、uncover (1) のマニュアルページおよび『Oracle Solaris Studio 12.4: Discover および Uncover ユーザーズガイド 』を参照してください。

♦ ♦ ♦ 第 5 章

5 デバッグツール

Oracle Solaris Studio には、コマンド行の dbx デバッガ、および dbx を使用するための dbxtool グラフィカルツールが用意されています。また、デバッガは IDE に統合されています。IDE を使用したデバッグの詳細は、第6章「Oracle Solaris Studio IDE」を参照してください。

この章には、デバッグツールの新機能に関する次のトピックが含まれます。

dbx デバッガについて

dbx デバッガは、対話型でソースレベルの事後およびリアルタイムのデバッグツールです。コマンド行、dbxtool グラフィカルインタフェース、および Oracle Solaris Studio IDE で使用できます。dbx デバッガは、スクリプティング可能で、マルチスレッドに対応しています。

dbx の新機能と変更された機能

次の機能が dbx で追加または変更されました。詳細は、『Oracle Solaris Studio 12.4: dbx コマンドによるデバッグ 』、dbx (1) のマニュアルページ、および dbx のヘルプファイルを参照してください。

- dbx の起動時間が大幅に短縮され、エンタープライズアプリケーションで最大 7 倍向上しました。

- dbx の新しいコンパイラおよびリンカーオプション。詳細は、56 ページの「デバッグをサポートするための新しいコンパイラおよびリンカーオプション」を参照してください。
- Oracle Solaris での C および C++ 式用にプリティプリンティングフィルタを作成できる新しい組み込み Python インタプリタ。詳細は、58 ページの「Python によるプリティプリンティング」を参照してください。
- 使用するプリティプリンティングメカニズムを決定する新しい dbxenv 変数 output_pretty_print_mode。call に設定されている場合、呼び出し形式のプリティプリンタを使用します。filter に設定されている場合、Python ベースのプリティプリンタを使用します。filter_unless_call に設定されている場合、まず呼び出し形式のプリティプリンタを使用します。
- 新しい dbxenv 変数 filter_max_length。この dbxenv 変数をプリティプリンティングフィルタによって配列に変換されるシーケンスの最大長に設定します。
- C++11 標準のサポートが追加されました。
- C11 標準のサポートが追加されました。
- 次のコンパイラオプションのサポートが追加されました。これらのオプションの詳細は、77 ページの「コンパイラに対する変更」を参照してください。
  - -g1
  - -xdebuginfo
  - -xglobalize
  - -xpatchpadding

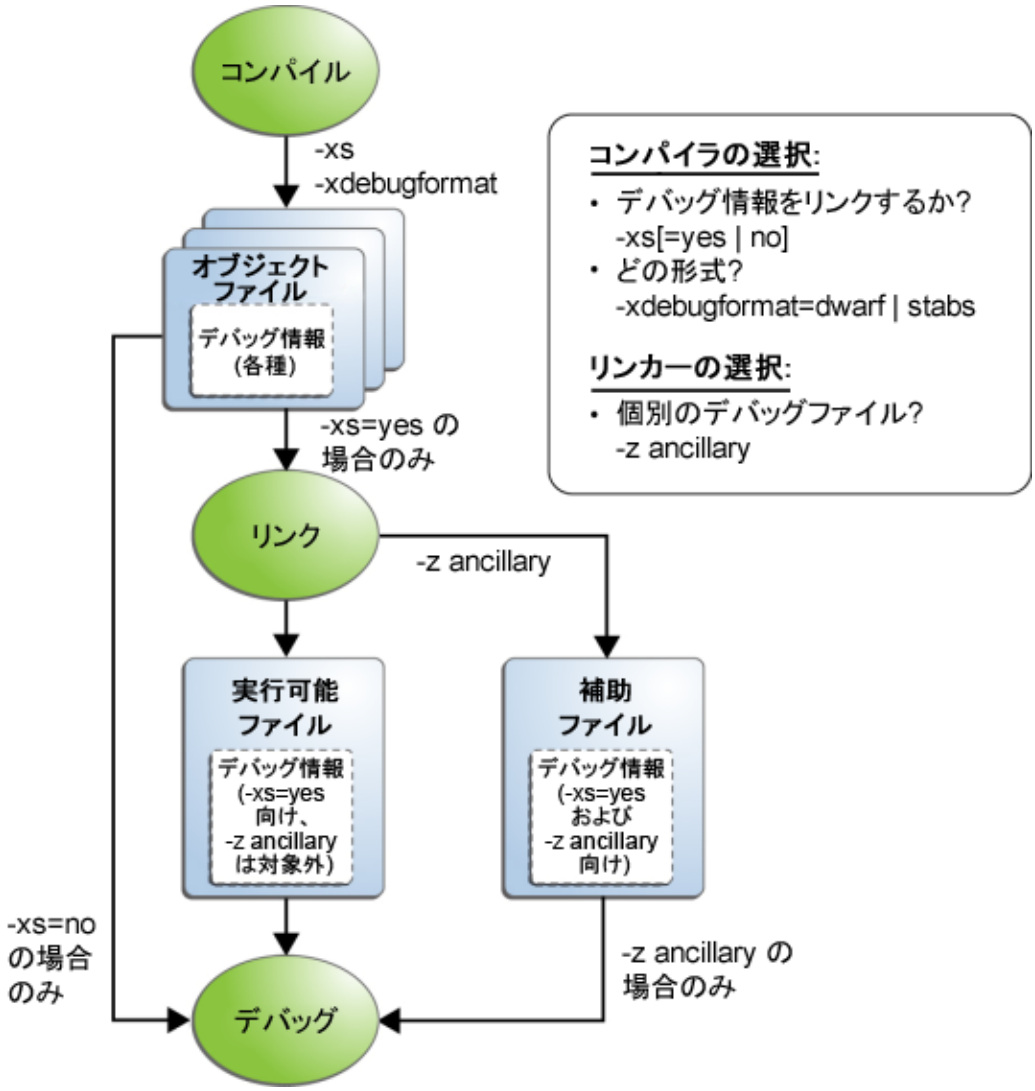
詳細は、dbx で help changes コマンドを発行して、dbx のヘルプファイルにアクセスしてください。

デバッグをサポートするための新しいコンパイラおよびリンカーオプション

新しいコンパイラおよびリンカーオプションにより、ユーザーはデバッグ情報をより自由に生成したり、使用したりできるようになります。コンパイラは、インデックススタブに似た DWARF のインデックスを生成します。このインデックスは常に存在し、これによって dbx の起動が速くなるほか、DWARF でのデバッグではその他の項目も改善されます。

次の図は、デバッグ情報のさまざまな種類と場所を示しています。ここでは、特にデバッグデータがどこに存在するかを強調しています。

図 5-1 デバッグ情報のフロー



インデックス DWARF (-xs[={yes|no}])

DWARF は、デフォルトでは実行可能ファイルにロードされます。新しいインデックスにより、-xs=no オプションで DWARF をオブジェクトファイル内に残すことが可能になります。これ

により、実行可能ファイルのサイズがより小さく、リンクがより高速になります。デバッグするには、これらのオブジェクトファイルを保持しておく必要があります。これは、スタブの動作と同様です。

個別のデバッグファイル (-z ancillary[=*outfile*])

Oracle Solaris 11.1 リンカーは、実行可能ファイルの構築中に、デバッグ情報を個別の補助ファイルに送信できます。個別のデバッグファイルは、すべてのデバッグ情報を移動、インストール、またはアーカイブする必要がある環境で役立ちます。実行可能ファイルは個別に動作できますが、その個別のデバッグファイルのコピーを使用して複数のユーザーがデバッグすることもできます。

dbx は、デバッグ情報を個別のファイルに抽出するための GNU ユーティリティー objcopy の使用を引き続きサポートしていますが、Oracle Solaris リンカーの使用には、objcopy に比べて次の利点があります。

- 個別のデバッグファイルはリンクの副産物として生成される
- 大きすぎて 1 つのファイルとしてリンクできなかったプログラムは 2 つのファイルとしてリンクされる

注記 - インデックス DWARF と別個のデバッグファイルの両方を使用すると、インデックス (リンクされる情報のみをもたらした) だけを別個のデバッグファイルにコピーする効果があります。インデックスで使用されるわずかなスペースが重要である場合を除き、これは通常はあまり役に立ちません。

デバッグ情報の最小化

-g1 コンパイラオプションは、配備されるアプリケーションのデバッグ可能性を最小限に抑えることを目的にしています。このオプションでアプリケーションをコンパイルすると、ファイルと行番号のほか、事後デバッグ中に重要であると見なされた単純なパラメータ情報が生成されます。詳細については、コンパイラのマニュアルページおよびコンパイラのユーザーガイドを参照してください。

Python によるプリティプリンティング

dbx には、Python でプリティプリンティングフィルタを作成できるメカニズムが用意されました。プリティプリンティングフィルタは、値を dbx で読み取り可能な形式に変換します。

dbx コマンド行では、print、display、および watch コマンドの -p オプションを使用するか、または dbxenv output_pretty_print on と入力して、プリティプリンティングを有効にすることができます。IDE および dbxtool では、dbxenv 変数 output_pretty_print を on に設定すると、プリティプリンティングを有効にすることができ、「ウォッチ」および「変数」ウィンドウのコンテキストメニューで「清書印刷」チェックボックスを使用できます。

フィルタは、C++ 標準テンプレートライブラリの 4 つの実装の選択したクラスに組み込まれています。次の表に、ライブラリ名とそのライブラリのコンパイラオプションを示します。

ライブラリのコンパイラオプション	ライブラリ名
-library=Cstd (デフォルト)	libCstd.so.1
-library=stlport4	libstlport.so.1
-library=stdcxx4	libstdcxx4.so.4.**
-library=stdcpp (-std=c++11 オプションを使用した場合のデフォルト)	libstdc++.so.6.*

次の表に、C++ 標準テンプレートライブラリで使用できるクラスフィルタと、インデックスおよび断面を出力できるかどうかを示します。

クラス	インデックスと断面が使用可能
string	N/A
pair	N/A
vector	はい
list	はい
set	はい
deque	はい
bitset	はい
map	はい
stack	はい
priority_queue	はい
multimap	はい
multiset	はい
tuple (C++ のみ)	N/A
unique_ptr (C++ のみ)	N/A

例 5-1 フィルタ付きの pretty-print

次の出力は、dbx の print コマンドを使用したリストの出力の例です。

```
(dbx) print list10
list10 = {
   __buffer_size = 32U
   __buffer_list = {
     __data_ = 0x654a8
   }
   __free_list   = (nil)
   __next_avail  = 0x67334
   __last     = 0x67448
   __node     = 0x48830
   __length   = 10U
   }
```

次は、dbx で出力された同じリストですが、pretty-print フィルタを使用しています。

```
(dbx) print -p list10
list10 = (200, 201, 202, 203, 204, 205, 206, 207, 208, 209)

(dbx) print -p list10[5]
list10[5] = 205

(dbx) print -p list10[1..100:2]
list10[1..100:2] =
[1] = 202
[3] = 204
[5] = 206
[7] = 208
```

プリティプリンティングおよび呼び出し形式のプリティプリンタの全般的な情報については、『Oracle Solaris Studio 12.4: dbx コマンドによるデバッグ 』の「pretty-print の使用」、および dbx のヘルプファイルのトピック prettyprint を参照してください。

dbxtool の変更点

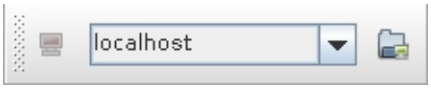
dbxtool は、スタンドアロンのデバッガ GUI です。Oracle Solaris Studio 12.3 で提供された dbxtool の機能はすべて残っていますが、Oracle Solaris Studio IDE に似た新しいルックアンドフィールになりました。

次の機能が dbxtool に追加されました。

- **「最近行なったデバッグ」ボタン** - dbxtool に「最近行なったデバッグ」ボタンとドロップダウンのデバッグ履歴リストが用意され、異なる引数を指定したコードの実行を選択すること

ができます。特定の実行を選択せずに「最近行なったデバッグ」ボタンを押すと、最新のターゲットおよび引数をデバッグします。

- **コード支援** - dbxtool では、コードをデバッグしているときにエディタでコード支援が自動的に有効になります。dbxtool ではコードを直接再コンパイルすることは依然としてできませんが、エディタでコード支援機能を利用してコードを修正することができます。コード支援を無効にするには、「プロジェクト」タブでデバッグターゲットを右クリックし、「コード支援」オプションを選択解除します。また、コード支援をオフにした状態で dbxtool を起動することもできます。コード支援の詳細は、68 ページの「コード支援の改善」を参照してください。
- **`--disable-code-assistance` オプション** - コード支援を使用しない場合、またはメモリー使用に問題があるもののコード支援を必要としない場合は、dbxtool を起動するときに `--disable-code-assistance` オプションを指定できます。
- **リモートホストツールバー** - リモートホストツールバー (デフォルトで有効) を使用して、dbxtool でリモートホストを管理および追加することを選択できるようになりました。



- **「プロジェクト」タブのデバッグターゲット** - dbxtool では、デバッグターゲットのすべてを 1 か所で自動的に表示するようになりました。デバッグセッションを開始すると、デバッグターゲットが作成され、「プロジェクト」タブに表示されます。ターゲット情報は、`userdir` ディレクトリに格納され、dbxtool の実行間で持続します。デバッグターゲットを展開すると、ルートソースディレクトリからのフォルダおよびファイル (ビルドされたバイナリから取得) が表示されます。

♦ ♦ ♦ 第 6 章

6 Oracle Solaris Studio IDE

Oracle Solaris Studio IDE は、グラフィカルプログラミング環境を好むユーザー向けに Oracle Solaris Studio の多くのコンポーネントを統合します。

この章は次の各節から構成されています。

Oracle Solaris Studio IDE について

Oracle Solaris Studio では、NetBeans プラットフォーム上に構築されたグラフィカル統合開発環境 (IDE) を提供します。この IDE は、Oracle Solaris Studio C、C++、および Fortran コンパイラ、`dmake` 分散 make コマンド、および `dbx` デバッガを使用するように構成されます。また、IDE は分析スイートの一部のアナライザツールを統合するため、IDE から離れることなくコードを分析することができます。

IDE を起動するコマンドは、`solstudio` です。このコマンドの詳細は、`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は、IDE のヘルプを参照してください。IDE の基本機能を使用するステップごとの手順については、『Oracle Solaris Studio 12.4: 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。

IDE の新機能および変更された機能

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

- **IDE プロジェクトに含まれない実行可能バイナリのデバッグ (プロジェクト不要デバッグ)。**「デバッグ」 > 「実行可能ファイルをデバッグ」を選択して、実行可能ファイルのパス、および実行可能ファイルの実行に必要な引数または環境変数を指定することで、実行可能ファイルをデバッグすることができます。また、「お気に入り」ウィンドウで実行可能ファイルに移動し、ファイルを右クリックして「デバッグ」を選択することもできます。実行可能ファイルは IDE プロジェクトに含まれる必要はありませんが、デバッガがデバッグ情報を検出できるように、その構築に使用されたソースコードで検出できる必要があります。
- **C++11 のサポート。**コードで C++11 標準を使用し、C++ コンパイラで C++11 標準を実装してある場合、IDE で C++11 のサポートを有効にすることができます。これにより、「auto」指定子などの機能のコード支援を使用することができます。プロジェクトの C++11 サポートを有効にするには、プロジェクトを右クリックして「プロパティー」を選択し、「構築」 > 「C++ コンパイラ」 > 「C++標準」 > 「C++11」を選択します。個別ファイルでのみ C++ サポートを有効にするには、そのファイルを右クリックして「プロパティー」を選択し、「C++ コンパイラ」 > 「C++標準」 > 「C++11」を選択します。
- **メモリー使用率の改善。**大規模プロジェクトのメモリー使用率が 50% 削減されました。
- **高速化された検索。**「使用状況を検索」がさらに高速になり、インタフェースが改善されました。「使用状況を検索」はバックグラウンドで実行されるようになったので、多数のファイルの検索中にほかのタスクを実行できます。結果は、別の「使用状況」パネルにただちに表示され、増分的に見つかった検索結果の表示を続行します。「使用状況」パネルには、進行状況インジケータと検出件数の増分カウントがあります。検索はいつでも停止でき、停止した時点までの検索結果が保存されます。検索一致結果間のナビゲート、論理ビューから物理ビューへの切り替え、異なる設定での「使用状況を検索」の再実行が可能です。また、検索にフィルタを追加したり、コメント内を検索したりすることもできます。
- **軽量な部分再解析。**再解析が改善したため、大量の依存関係が含まれるソースファイルの編集が大幅に高速化しました。再フォーマット、関数本体内の編集、コメントやスペースの追加といった変更によって、プロジェクト全体が再解析されることはなくなりました。特定の変更の場合に、IDE がプロジェクト全体を再解析します。
- **デバッガのブレークポイントグループ。**ファイル別、プロジェクト別、タイプ別、言語別といった、複数の異なるカテゴリを使用するとブレークポイントをグループ化できます。「ウィンドウ」 > 「デバッグ」 > 「ブレークポイント」ウィンドウで、ブレークポイントグループの選択アイコンをクリックし、ブレークポイントグループを選択します。ブレークポイントは、選択に従って配置されます。

- **ウィンドウの管理とグループ化。**個別ウィンドウのほかに、ウィンドウのグループに対してアクションを実行できます。各ウィンドウは 1 つのグループに属し、このグループに対して、最小化と復元、新しい場所への移動、別個のウィンドウでのフロート、または IDE ウィンドウへの再連結を行うことができます。たとえば、グループの右側にある「ウィンドウ・グループの最小化」ボタンをクリックして、左上にある「プロジェクト」、「ファイル」、「クラス」、および「サービス」ウィンドウを最小化することができます。グループのタブ領域内を右クリックしてグループのオプションを選択するか、「ウィンドウ」>「ウィンドウの構成」を選択します。ウィンドウの動作を制御するオプションは、「ツール」>「オプション」>「外観」>「ウィンドウ」にあります。
- **クリップボードへのファイルパスのコピー。**IDE 内で任意のファイルにマウスカーソルを合わせて Alt+Shift+L を押すと、ファイルのパスをクリップボードにコピーできます。
- **起動ツール機能。**66 ページの「IDE の新しい起動ツール機能」を参照してください。
- **コードエディタの改善。**67 ページの「IDE コードエディタの改善」を参照してください。
- **コード支援の改善。**68 ページの「コード支援の改善」を参照してください。
- **ブレッドクラム。**71 ページの「ブレッドクラムナビゲーションの使用」を参照してください。
- **IDE の「アクション項目」ウィンドウが C/C++ プロジェクトで使用できるようになりました。**「アクション項目」ウィンドウには、コメント (`TODO`、`FIXME`、`Pending`、`<<<<<<<<<` など) でマークしたソースファイルの行が表示されます。検出される文字列は、「ツール」>「オプション」>「チーム」>「アクション項目」で指定されます。C/C++ プロジェクトの場合、「アクション項目」ウィンドウには、プロジェクトを最後にビルドしたときに発生したコンパイルエラーおよび警告も表示されます。
- **バージョン管理システムのサポートの改善**:
  - Git リポジトリのサポート。
  - ローカル履歴: 「削除を元に戻す」と新しい「履歴」タブ。
  - 変更をシェルブ: ローカルの変更を破棄し (シェルブ)、別の機能で作業を開始できます。Mercurial および Subversion については、「チーム」>「変更をシェルブ」メニューオプションを参照してください。
  - Mercurial の強化: ブランチ、タグ、およびキューの基本的なサポート。
- **新しい solstudio コマンド行オプション。**`--open-group` および `--close-group` オプションを使用して、IDE を起動するときにプロジェクトグループを開いたり閉じたりすることができます。
- **ブックマークの更新。**「ブックマーク」ビューを「ウィンドウ」>「IDEツール」>「ブックマーク」を使用して開いたり、ファイルで Ctrl+Shift+M を使用してブックマークを作成したりすることができます。

- **Ctrl+Space を押すことにより、検索バーで自動補完。**エディタで検索するときは、エディタに入力するときと同様に、検索語句を自動補完することができます。
- **「メインプロジェクト」という概念がほとんどのタスクで使用されなくなりました。**これまでどおり「実行」 > 「メイン・プロジェクトとして設定」を使用してメインプロジェクトを設定することはできます。
- **オプションを探しやすくなりました。**「ツール」 > 「オプション」ダイアログに検索ボックスが用意され、オプションを簡単に探せるようになりました。
- **ツールバーインタフェースの改善。**メインツールバーでは、一部のツールバー項目が表示されていないことを示すために、表示されていないボタンにアクセスできるドロップダウンリストが表示されます。以前は、有効なツールバーが多すぎる場合に、すべてが表示されませんでした。
- **既存のコードを持つプロジェクトから単一ファイルをコンパイルします。**
- **プロジェクトごとの C/C++ フォーマットスタイル。**プロジェクトのプロパティーでプロジェクト固有のフォーマットスタイルを選択するときは、C フォーマットスタイル、C++ フォーマットスタイル、および C/C++ ヘッダーフォーマットスタイルを指定することができます。

IDE の新しい起動ツール機能

「起動ツール」を作成することで、たとえば簡単にプロジェクトをプロジェクトコンテキストメニューから別の引数を使用して実行したり、プロジェクトをスクリプトから起動したりできます。通常、アプリケーションを IDE から実行すると、プロジェクトのプロパティー内で実行コマンドとして指定されている実行可能ファイルが実行されます。起動ツールを作成すると、実行する複数のコマンドを指定し、コンテキストメニューから選択できます。起動ツールはデバッグ目的でも使用できます。

起動ツールを作成するには、`nbproject/private` フォルダに移動し、起動ツールファイル (`launchers.properties`) をカスタマイズします。

新規起動ツールファイルダイアログで、起動ツールの定義をプロジェクトの `nbproject/private` サブフォルダに格納する場合は、ファイルのプライバシのオプションを選択します。このオプションは、特にバージョン管理システム内でプロジェクトがほかの開発者と共有されている場合に便利です。VCS 内では `nbproject/private` を無視できるため、プロジェクトを共有するときに含まれません。プライベート起動ツールファイルが存在する場合、プライベートファイル内の起動ツールは、パブリック起動ツールファイル内の同名の起動ツールをオーバーライドします。

「終了」をクリックすると、launchers.properties テキストファイルが IDE エディタで開きます。実行するコマンドと、これらのコマンドを実行するために IDE 内で表示する表示名を指定することができます。たとえば、Arguments という名前の IDE の C/C++ サンプルアプリケーションの場合、launchers.properties ファイルに次を追加できます。

```
launcher1.runCommand="${OUTPUT_PATH}" "arg 1" "arg 2" "arg 3" "arg 4"
launcher1.displayName=Four Args

launcher2.runCommand=../dist/Debug/OracleSolarisStudio-Solaris-x86/arguments_1 "arg 1"
launcher2.displayName=One Arg

launcher3.runCommand=/bin/sh runMyProgram.sh
```

ファイル runMyProgram.sh は、たとえば環境変数を設定したり、必要な何らかの処理を行なったりするスクリプトの場合もあります。

スクリプトを実行する起動ツールを使用してアプリケーションをデバッグする場合、その起動ツールに対してオプション symbolFiles を指定することで、スクリプトを実行するために使用されるシェルの代わりにデバッガがアプリケーションをデバッグできるようにする必要があります。上記の launcher3 の例では、このオプションは次のように追加できます。

```
launcher3.runCommand=/bin/sh runMyProgram.sh
launcher3.symbolFiles=${LINKER_OUTPUT}
```

起動ツールオプションの追加を終えたら、launcher.properties ファイルを保存します。

その後、プロジェクトを右クリックしていずれかのコマンドを選択することで、これらのコマンドを実行することができます。上記の例では、右クリックして「実行方法」コマンドを選択し、コマンド名を選択します。たとえば、「実行方法」>「Four Args」または「名前を付けてデバッグ」>「/bin/sh runMyProgram.sh」です。

IDE コードエディタの改善

コードエディタには、次を含む多くの改善点があります。

- 68 ページの「矩形ブロック選択」
- 68 ページの「クリップボード履歴」
- 68 ページの「検索/置換の機能強化」

矩形ブロック選択

エディタでは、Ctrl+Shift+R を押すか、エディタツールバーの「矩形選択の切替え」アイコンをクリックすることで、テキストの矩形ブロックを選択するモードを有効にすることができます。このモードでは、各行の先頭または末尾を含まずにテキストの複数行の一部を選択することができます。たとえば、先頭のアスタリスクを含めずにコメントブロックを選択してコピーすることや、コメントアウトされたコードの一部からアスタリスクを 1 行ずつ削除せずに 1 アクションですべてのアスタリスクを選択して削除することができます。また、たとえばテキスト表の列をコピー、移動、または削除することもできます。矩形選択されたテキスト内に文字を入力すると、その文字が各行でレプリケートされ、選択されたテキストは置換されます。

クリップボード履歴

デスクトップクリップボードにコピーされたテキストのうち、最新 9 件のテキストバッファーを参照して、貼り付ける行を選択できます。テキストを挿入する位置にカーソルを置いた状態で Ctrl+Shift+D を押すと、クリップボードエントリを表示するポップアップが開きます。矢印キーを使用してクリップボードバッファーをナビゲートし、バッファーのリストの下にあるウィンドウで内容をすべて確認します。バッファーの内容を貼り付けるには、バッファーの番号を入力するか、該当するバッファーが選択されている状態で Enter を押します。このバッファーには、IDE だけではなく、デスクトップ上の任意のウィンドウからコピーされた内容が含まれています。

検索/置換の機能強化

エディタの検索/置換機能は、置換用の個別のダイアログボックスの代わりに、エディタウィンドウ下部の「検索」ツールバーで完全に機能するようになりました。「置換」フィールドとボタンは、ツールバーの「検索」フィールドとボタンの下に表示されます。「検索」ツールバーをアクティブにするには Ctrl+F を押し、「置換」機能をアクティブにするには Ctrl+H を押します。

コード支援の改善

IDE には、次を含む多くのコード支援における改善点があります。

- 69 ページの「コード支援キャッシュの共有」
- 70 ページの「コード支援のプロジェクトのプロパティーの新しいオプション」

■ 70 ページの「ファイルシステムから C/C++ ヘッダーファイルの検索」

コード支援キャッシュの共有

C/C++ ソースコードの解析時に、IDE は解析結果をディスク上のコード支援キャッシュに保存します。プロジェクトを開くと、IDE はキャッシュを検証し、キャッシュが最新かどうかを確認します。キャッシュが最新の場合、IDE はプロジェクトを解析せず、コードのナビゲーションに必要なデータをコード支援キャッシュからロードします。

デフォルトで、コード支援キャッシュは `${`*userdir*`}/var/cache` フォルダ (`${`*userdir*`}` は IDE ユーザーディレクトリ) にあります。Oracle Solaris のユーザーディレクトリは、ユーザーの `$HOME/.solstudio/ide-`*<release>* にあります。ユーザーディレクトリ内のキャッシュを別の場所に共有またはコピーすることはできません。

ただし、コード支援キャッシュがプロジェクトの内部に存在する場合は、コピー先のコンピュータが次の要件を満たす場合に別のコンピュータにコピーできます。

- コンピュータのオペレーティングシステムが、コードが解析されたオペレーティングシステムと同一である
- プロジェクトで使用されるツールコレクションがコンピュータ上の同じ場所で利用できる

IDE に、コード支援キャッシュをプロジェクトメタデータの内部に配置するよう指示する手順。

1. 行「`cache.location=nbproject/private/cache`」を次のいずれかに追加します。
   - プロジェクトプロパティーファイル (`nbproject/project.properties`)
   - プライベートプロパティーファイル (`nbproject/private/private.properties`)

   プロジェクトプロパティーファイルとプライベートプロパティーファイルの違いは、パブリックファイル (`nbproject/project.properties`) はデフォルトでバージョン管理システム経由で IDE で共有されるのに対し、プライベートファイル (`nbproject/private/private.properties`) はそうでないという点です。このため、プライベートプロパティーを変更した場合は、プライベートプロパティーファイルを別のマシン上の同一ファイルと同期する必要があります。プロジェクトプロパティーファイルが変更された場合、バージョン管理システムは、そのファイルを別のマシン上のファイルと自動的に同期できます。
2. プロパティーファイルが変更されたあとで、プロジェクトを閉じてふたたび開きます。

   IDE はプロジェクトを解析し、コード支援キャッシュがプロジェクトメタデータ内のプライベートサブディレクトリに配置されます。

3. プロジェクトを閉じ、nbproject/private/cache をアーカイブするか共有場所にコピーします。

   コピーまたは圧縮する前にプロジェクトを閉じない場合、一部のデータがキャッシュにフラッシュされません。

コード支援キャッシュは、IDE がプロジェクトを解析するまで待たずに、ほかのマシン上のほかのプロジェクトにコピーされ、使用できるようになります。キャッシュがコピーされるマシン上に新しいファイルがある場合は、より新しいファイルのみが解析されます。

注記 - 異なるオペレーティングシステムまたは異なるコンパイラを実行しているマシン間でコード支援キャッシュを共有する必要がある場合は、オペレーティングシステムとコンパイラコレクションの組み合わせごとに別々のキャッシュを作成する必要があります。

コード支援のプロジェクトのプロパティーの新しいオプション

既存のソースまたはバイナリから作成されるプロジェクトでは、プロジェクトをバージョン管理システムで容易に使用できるようにするため、IDE には次のプロジェクトのプロパティーがあります。

一時マクロ	揮発性のマクロ (時間、日付、または特定の環境に依存) のリストを提供できます (-D オプション)。これらのマクロ値は、プロジェクトのパブリックメタデータとともに保存されません。
ユーザー環境変数	プロジェクトがシステム固有のパスを渡すときに使用する環境変数のリストを指定できます。これらのマクロ環境変数値は、プロジェクトのパブリックメタデータとともに保存されません。既存のコードまたはバイナリからのプロジェクトでは、プロジェクトメタデータの格納時に使用する環境変数のリストを指定できます。IDE がコンパイラオプションを格納し、オプション値が変数値と一致する場合は、代わりにマクロが記述されます。

ファイルシステムから C/C++ ヘッダーファイルの検索

ソースがまだ構築されておらず、デバッグ情報が含まれていないプロジェクトを既存ソースから作成すると、IDE ではコード支援の構成で問題が発生することがあります。この場合は、「コード支援を構成」ウィザードで、特殊モード (ファイルシステムから C/C++ ヘッダーファイルを検索) を使用するよう指定できます。このモードでは、IDE はファイルシステムからヘッダーを検索することによって、失敗した include ディレクティブを解決しようとします。ウィザードで、ヘッダーを検索するためのパスを入力するよう求められます。デフォルトでは、このパスはプロジェクトソースルートです。

ブレッドクラムナビゲーションの使用

ブレッドクラムを使用すると、IDE 内の自分の位置を追跡することができます。ブレッドクラムナビゲーションバーはソースエディタのウィンドウの下に配置され、カーソル位置を基準にして、ネストされた要素を表示します。各要素には、その型を識別するためのアイコンによるマークが付いています。アイコンの完全なリストとアイコンの意味については、IDE でクラスとナビゲータウィンドウで使用されるアイコンに関するヘルプページを参照してください。

ブレッドクラムを使用するには、次のいずれかを行います。

- ブレッドクラムバーの矢印をクリックして、ネストされた文またはメンバーのリストを表示し、1 つを選択するとソースエディタ内でその場所に移動します。
- ブレッドクラムバーの項目をクリックして、カーソル位置の履歴を前に移動します。
- Alt+ ← キーおよび Alt+ → キーを押して、カーソル位置の履歴を前後に移動します。

♦ ♦ ♦ 第 7 章

7 OpenMP API とスレッドアナライザ

この章では、Oracle Solaris Studio のこのリリースにおける OpenMP API サポートおよびスレッドアナライザの変更点について説明します。

OpenMP

このセクションでは、OpenMP API の新機能および更新について説明します。

OpenMP 4.0 のサポート

このリリースでは、OpenMP API 標準言語仕様の主要アップグレードである OpenMP API バージョン 4.0 に導入された新しい機能がサポートされています。C、C++、および Fortran コンパイラによってサポートされるこのリリースの新しい OpenMP 4.0 機能には、次のものがあります。

- **エラー処理** - OpenMP 4.0 では、実行時エラーが存在するときの OpenMP アプリケーションの回復力および安定性を改善するためのエラー処理機能が定義されています。OpenMP の並列実行は、条件付き取り消しおよびユーザー定義の取り消しポイントを使用して完全に中止できます。
- **スレッドアフィニティー** - OpenMP 4.0 には OpenMP スレッドを実行する場所を定義するメカニズムが提供されており、その結果、局所性の改善、偽共有の減少、メモリー帯域幅の増加をもたらします。
- **タスク拡張** - OpenMP 4.0 は、タスクベースの並列化サポートについて、いくつかの拡張機能を提供しています。タスクをグループ化して、強いタスク同期化をサポートすることができます。タスク間の同期は、タスク依存関係の指定を通じてサポートされます。

- **Fortran 2003 のサポート** - Fortran 2003 規格では、最新のコンピュータ言語機能が多数追加されました。これらの機能を OpenMP 仕様に持たせることで、ユーザーは Fortran 2003 互換のプログラムを並列化することができます。
- **順序一貫性を持つ原子性** - ストレージの特定の場所が原子的にアクセスされたとき、順序一貫性を強制することができる節が追加されました。
- **ユーザー定義の縮約** - 基本言語演算子と組み込み手続きによる縮約に加えて、OpenMP 4.0 はユーザー定義の縮約をサポートします。カスタム縮約は `declare reduction` ディレクティブを使用することによってプログラマから定義でき、これらの縮約は後に `reduction` 節で指定できます。
- **新しい環境変数** `OMP_DISPLAY_ENV` - `OMP_DISPLAY_ENV` 環境変数は、OpenMP 環境変数に関連付けられた内部制御変数 (ICV) の値を表示するために使用できます。

注記 - このリリースでは、OpenMP 4.0 デバイスと SIMD 構文が使用できます。ただし、すべてのコードはホストデバイス上で実行され、SIMD 構文によって SIMD 命令が使用されないこともあります。

詳細は、『Oracle Solaris Studio 12.4: OpenMP API ユーザーズガイド 』を参照してください。

OpenMP 4.0 の機能の詳細は、OpenMP Application Program Interface Version 4.0, July 2013 (http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf) および OpenMP 4.0.1 Examples, February 2014 (http://openmp.org/mp-documents/OpenMP_Examples_4.0.1.pdf) を参照してください。

OpenMP 関連の機能強化

次の項目は、Oracle Solaris Studio の OpenMP API への追加の拡張機能です。

- **新しいデフォルトのスレッド数** - 並列領域の実行に使用されるスレッドのデフォルトの数が、2 つから、マシンで利用できるコアの数に変更され、上限は 32 です。
- **スタックオーバーフローの検出と診断** - 既存の C、C++、および Fortran コンパイラのオプション `-xcheck=stkovf` が拡張され、オプションでランタイムエラー診断を実行できるようになりました。

  構文は次のとおりです。

  ```
  -xcheck=stkovf [:detect | :diagnose]
  ```

`:detect` が指定された場合、そのエラーに通常関連付けられているシグナルハンドラを実行することによって、検出されたスタックオーバーフローエラーが処理されます。

`:diagnose` が指定された場合、関連付けられているシグナルをキャッチし、stack_violation(3C) を呼び出してエラーを診断することによって、検出されたスタックオーバーフローエラーが処理されます。スタックオーバーフローが診断されると、エラーメッセージは `stderr` に出力されます。これは何も指定されていない場合のデフォルトの動作です。

`-xcheck=stkovf` コンパイラオプションの詳細は、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。

スレッドアナライザ

スレッドアナライザは、マルチスレッドプログラムの実行を解析し、データの競合やデッドロックなどのスレッドの一般的なエラーを検出する強力なツールです。スレッドアナライザがあれば、マルチスレッドアプリケーションを簡単にデバッグでき、生産性が高まります。スレッドアナライザは、次に示す標準およびフレームワークの 1 つか、これらを組み合わせて記述されたプログラムと一緒に使用できます。

- POSIX スレッド API
- Oracle Solaris スレッド API
- OpenMP ディレクティブ

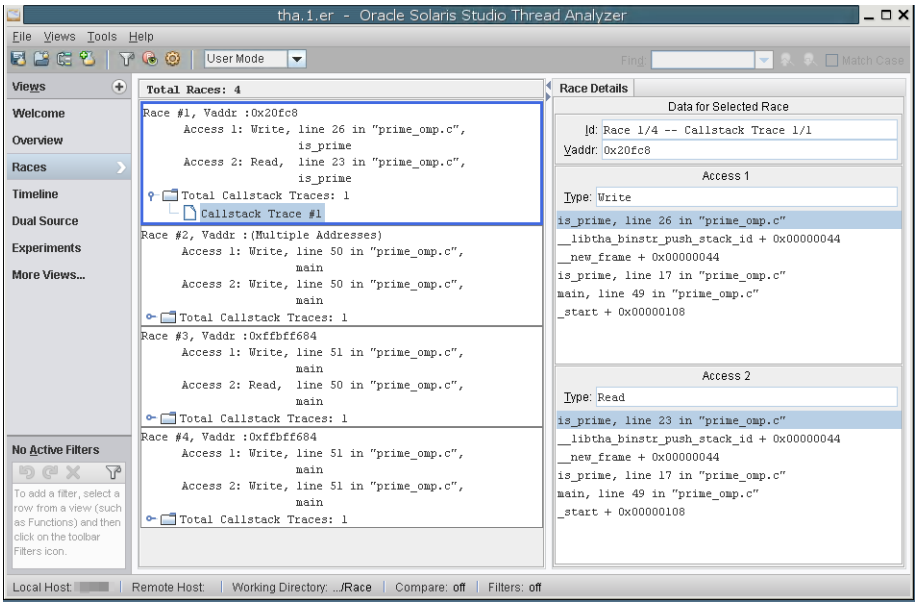
スレッドアナライザの詳細については、`tha(1)` のマニュアルページと『Oracle Solaris Studio 12.4: スレッドアナライザユーザーズガイド 』を参照してください。

Oracle Solaris Studio のこのリリースでは、次の機能が追加されました。

- Discover を使用したデータ競合検出のためのバイナリ計測を実行するとき、データ競合へのアクセスのために、呼び出しスタック全体が表示されます。
- Oracle Solaris 10 で導入された `atomic_ops` API のサポート。
- `collect -r terminate` のサポート。デッドロック検出がオンのときにデッドロックが実際に発生した場合、プロセスの終了を強制します。
- データの表示とナビゲーションを改善するために、スレッドアナライザのユーザーインタフェースがパフォーマンスアナライザ用のユーザーインタフェースとともに再設計されました。詳細は、23 ページの「パフォーマンスアナライザのナビゲーション」を参照してください。

次の図は、discover を使用してバイナリを計測したときに、スレッドアナライザが 2 つの呼び出しスタックを表示する方法を示しています。

図 7-1 スレッドアナライザウィンドウ



♦ ♦ ♦ 第 8 章

8 その他の変更点

この章では、Oracle Solaris Studio ソフトウェアのその他のコンポーネントについて、新機能および変更された機能について説明します。

コンパイラに対する変更

次のセクションではコンパイラに対して行われた変更について説明し、内容は次のとおりです。

全コンパイラに共通の新機能および変更点

C、C++、および Fortran コンパイラに対して前のリリースから次の変更が行われました。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。C++ コンパイラの固有の変更点については、第2章「C++ コンパイラ」に記載されています。

新しいハードウェア上でのアプリケーションパフォーマンス

Oracle Solaris Studio のすべてのリリースには、Oracle Sun ハードウェアサーバー用のパフォーマンス向上が含まれています。このリリースでは、Oracle Solaris Studio 12.3 1/13 Platform-Specific Enhancements リリースで最初に利用可能になった、SPARC

T5、SPARC M5、SPARC M6、SPARC M10、および Intel Ivy Bridge ならびに Haswell のコンパイラおよびライブラリのパフォーマンス向上のための拡張サポートが含まれています。

- x86 上の Ivy Bridge および Haswell プロセッサについての新しい `-xarch`、`-xchip`、および `-xtarget` の値。
- SPARC T5、M5、M6、および M10+ プロセッサについての新しい `-xarch`、`-xchip`、および `-xtarget` の値。
- Ivy Bridge および Haswell アセンブラ命令のサポート。
- Ivy Bridge および Haswell 組み込み関数のサポート。これらは *solstudio-install-dir*`/lib/compilers/include/cc/sys</immintrin.h` にあります。
- x86 および x64 アーキテクチャーで `-xarch=generic` のデフォルト値を `sse2` に設定。詳細は、78 ページの「x86 についてのデフォルトの浮動小数点動作の変更」を参照してください。

x86 についてのデフォルトの浮動小数点動作の変更

x86 上のデフォルトコンパイラ値を使用してコンパイルされたプログラムでの浮動小数点計算の結果が、以前のリリースと比べて Oracle Solaris Studio 12.4 では若干異なる場合があります。同じハードウェアおよびオペレーティングシステム上でも異なる結果が得られる場合があります。これは、すべてのアドレス空間モデルおよびプラットフォームについてのデフォルトの命令セットアーキテクチャーが SSE2 であるためです。

Oracle Solaris Studio 12.4 および以前のリリースでは、32 ビットの Oracle Solaris の場合のデフォルトのアドレス空間モデルは `-m32` です。Linux では、64 ビットハードウェアについてのデフォルトは `-m64` です。すべてのプラットフォームで `-m32` および `-m64` をそれぞれ使用することで、32 ビットアドレス空間モデルまたは 64 ビットアドレス空間モデル用にコンパイルすることができます。

コンパイラは、ハードウェアで実装される命令を決定するために `-xarch` オプションを使用することでコードを最適化するため、コード生成に適しています。以前の Oracle Solaris Studio リリースのデフォルトは、`-m32` の場合は `-xarch=386`、`-m64` の場合は `-xarch=sse2` でした。

32 ビットアドレス指定の x86 について、コード生成オプションが `-xarch`、`-xnative`、または `-fast` で指定あるいは暗黙指定されない場合、コード生成オプションは `-xarch=386` でなく `-xarch=sse2` になります。したがって、浮動小数点演算を使用してデフォルトの `-xarch` でコンパイルされたプログラムでは、浮動小数点演算の結果が異なる可能性があります。

新しい x86 のデフォルトの -m32 -xarch=sse2 では、以前のデフォルトの -m32 -xarch=386 と同じ ABI が実装されます。浮動小数点オペランドおよび結果は x87 浮動小数点レジスタに渡されます。ただし、次の単精度および倍精度浮動小数点演算は、通常 sse2 レジスタで実行されます。

```
+
-
*
/
sqrt
convert
```

x87 レジスタは、長倍精度演算およびハードウェア初等超越関数の評価に引き続き使用されます。

その他のコンパイラの変更

- x86 での -xlinkopt のサポート。最新の Intel プロセッサ用に調整された大規模エンタープライズアプリケーション用に、モジュール間、手続き型コード間の順序付け最適化。大規模アプリケーションについて、完全に最適化されたバイナリよりも最大 5% のパフォーマンス向上が見られます。
- x86 についての新しいコンパイラオプション: -preserve_argvalues は、レジスタベースの関数引数のコピーをスタックに保存します。
- 実行可能なサイズと、デバッグ目的でオブジェクトファイルを保持する必要性とのトレードオフを制御する、拡張された -xs オプション。
- Linux での -xanalyze および -xannotate のサポート。
- -xopenmp=parallel のシノニムとしての -fopenmp のサポート。
- SPARC M10 値 -xchip=sparc64x、-xtarget=sparc64x、-xarch=sparcace のサポート。
- SPARC M10+ 値 -xchip=sparcxplus、-xtarget=sparc64xplus、および -xarch=sparcaceplus のサポート。
- SPARC M10+ プロセッサに共通の SPARC 命令セット拡張のための -xarch=sparc4b および -xarch=sparc4c のサポート。
- x86 プラットフォーム上での Ivy Bridge 値 -xchip=ivybridge、-xtarget=ivybridge、および -xarch=avx_i のサポート。
- x86 プラットフォーム上での Haswell 値 -xchip=haswell、-xtarget=haswell、および -xarch=avx2 のサポート。

- コンパイラの新しいオプション:
  - `-g1` - 事後デバッグの際に重要と思われるファイル、行番号、および簡単なパラメータ情報を生成します。
  - `-xdebuginfo` - デバッグおよび可観測性情報の出力量を制御します。
  - `-xglobalize` - ファイルの静的変数のグローバル化を制御します (関数は制御しません)。
  - `-xinline_param` - コンパイラが関数呼び出しをインライン化するタイミングを判断するために使用するヒューリスティックを変更できます。
  - `-xinline_report` - コンパイラによる関数のインライン化に関する報告を生成し、標準出力に書き込みます。
  - `-xipo_build` - コンパイラを介した最初の受け渡し時には最適化せず、リンク時にのみ最適化することによって、コンパイルの時間が短縮されます。
  - `-xkeep_unref` - 参照されない関数および変数の定義を維持します。
  - `-xpatchpadding` - 各関数の開始前にメモリー領域を予約します。
  - `-xsegment_align` - ドライバがリンク行で特殊なマップファイルをインクルードするようにします。
  - `-xthroughput` - システム上で多数のプロセスが同時に実行されている状況でアプリケーションが実行されることを示します。
  - `-xunboundsym` - 動的に結合されたシンボルへの参照がプログラムに含まれているかどうかを指定します。

C コンパイラ

C コンパイラの変更点には、77 ページの「全コンパイラに共通の新機能および変更点」で説明されている変更点と、次の追加の変更点が含まれます。

- x86 での `-xregs=float` のサポート。
- C コンパイラの新しいオプション:
  - `-ansi` - `-std=c89` と同等です。
  - `-pedantic` - ANSI 以外の構文に対するエラー/警告への厳密な準拠を強制します。
  - `-staticlib` - `-library=sunperf` とともに使用すると、Sun パフォーマンスライブラリと静的にリンクします。
  - `-std` - C 言語標準を指定します。`-std=c11` はデフォルトコンパイラモードです。

- -temp - 一時ファイルのディレクトリを定義します。
- -xlang - -std フラグで指定されたデフォルトの libc 動作をオーバーライドします。
- -xprevise - コードアナライザを使用して表示可能なソースコードの静的解析を実行します。

- C11 機能のサポート:
  - _Static_assert
  - 無名の構造体/共用体
  - _Noreturn 関数表明
  - _Thread_local 記憶指定子
  - _Alignof 演算子
  - _Alignas 整列指定子
  - C11 で指定された UCN の文字セット

詳細は、cc のマニュアルページと『Oracle Solaris Studio 12.4: C ユーザーガイド 』を参照してください。

Fortran コンパイラ

Fortran コンパイラは、Fortran77、Fortran90、および Fortran95 規格のための、過去最高のランタイムパフォーマンスおよび互換性オプションにより、科学技術アプリケーション開発をサポートします。Fortran 2003 の機能と OpenMP 4.0 のサポートの大部分が含まれます。Fortran コンパイラでは C および C++ コンパイラと同様のハイパフォーマンスなコード生成テクノロジが使用され、最新の SPARC および x86 ベースの Oracle システムのための最大パフォーマンスの並列コードがアプリケーションで生成されることを保証します。

Fortran コンパイラへの変更は、77 ページの「全コンパイラに共通の新機能および変更点」に記載されている変更を含みます。

次に、Fortran コンパイラのバージョン 8.7 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、f95 (1) のマニュアルページと『Oracle Solaris Studio 12.4: Fortran ユーザーズガイド 』を参照してください。

- -xM オプションを使用して、メークファイルの依存関係を自動的に生成できます。新しい -keepmod=yes オプションと組み合わせることで、モジュールを使用する Fortran アプリ

ケーションについての最適化された増分ビルドが可能です。新しい -keepmod オプションは、コンパイル時に変更されないモジュールを保持するために使用されます。デフォルトは -xkeepmod=yes で、以前のコンパイルから変更がない場合でも、新しいモジュールファイルが作成されるたびに古い動作を置換します。

- モジュールを使用するアプリケーションのコンパイル時間が大幅に改善され、モジュール処理が原因のメモリーオーバーフローが排除されました。
- #pragma ident は、コンパイルされるオブジェクトのソースバージョンを識別するためにソースファイル内で使用できます。
- 宣言で使用される文字型で LEN 型パラメータとしての遅延型パラメータ (コロン) のサポート。例:

  ```
  character(LEN=:), pointer :: str
  ```

- 手続きポインタのサポート。
- ISO_C_BINDING モジュールについての Fortran 2003 関数 C_F_PROCPOINTER() のサポート。C_FUNLOC() 関数が、手続きポインタを引数として使用できるように拡張されました。
- オブジェクト指向の Fortran を完全にサポート。次の属性を持つ型束縛手続きが使用できるようになりました。
  - GENERIC
  - DEFERRED
  - NON-OVERRIDABLE
  - PASS
  - NOPASS
- 構造型と総称関数が同じ名前を持つことができるようにする Fortran 2003 機能のサポート。
- TARGET オブジェクトを INTENT(IN) ポインタダミーに渡す Fortran 2008 機能のサポート。
- すべての要素組み込み関数 (各引数そのものが初期値式であるもの) を、Fortran 2003 規格で指定されている初期値式に使用できるようにする拡張サポート。これまでは、このコンテキストで使用される要素組み込み関数は、整数型および文字型を返すものだけに制限されていました。
- プログラムが一度に複数のスレッド内で I/O を実行しないことを指定する -fserialio のサポート。

パフォーマンスライブラリの変更点

Oracle Solaris Studio パフォーマンスライブラリは、線形代数や大量の数値計算を伴う問題を解くために最適化された高速な数学サブルーチンのセットです。Oracle Solaris Studio パフォーマンスライブラリは、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に基づいています。Oracle はこれらのパブリックドメインサブルーチンを強化し、Oracle Solaris Studio パフォーマンスライブラリとしてバンドルしました。

このリリースでは次の変更が行われました。

- SPARC T5、M5、および M6 プラットフォームおよび SPARC 64X+ でのパフォーマンス調整。
- Oracle Solaris Studio パフォーマンスライブラリの LAPACK がバージョン 3.4.2 にアップグレードされました。次の機能を含む LAPACK 3.4.2 のすべての新機能が Oracle Solaris Studio パフォーマンスライブラリに実装されています。
  - 超高精度反復絞り込み線形ソルバー (3.2)
  - 新しい高速精密 Jacobi SVD。(3.2)
  - Rectangular Full Packed フォーマット (RFP) 用ルーチン。(3.2)
  - ピボット法によるコレスキー分解。(3.2)
  - 高速単一精密ハードウェアを活用した混合型精密反復絞り込みサブルーチン。(3.2)
  - 完全な CS 分解の計算。(3.3)
  - レベル 3 BLAS 対称無限解法および対称無限反転。(3.3)
  - `xGEQRT`: QR 因数分解 (インタフェースの改善)。(3.4)
  - `xGEQRT3`: 再帰的 QR 因数分解。(3.4)
  - `xTPQRT`: 通信抑制 QE シーケンシャルカーネル (3.4)

  導入された LAPACK バージョンが括弧に記載されています。

索引

U