

K-scope利用手引き

2013年10月31日(木)

理化学研究所 計算科学研究機構
運用技術部門ソフトウェア技術チーム
寺井 優晃

- はじめに
- ツールの起動
- 新規プロジェクトの作成
- インターフェイス
- 静的プログラム構造解析
- 検索機能
- 付加情報機能
- 「京」プロファイラ連携機能
- 分析機能
 - 変数特性一覧機能
 - 宣言・定義・参照
 - 変数有効域
 - トレース機能
 - 変数アクセス先設定
 - Byte/FLOP算出機能
- 補足

はじめに(1/2)

近年、プログラミング開発環境及び実行環境の性能向上を背景に、より精緻なシミュレーションを目的とした大規模で複雑な科学技術計算コードが比較的容易に用いられる傾向があります。

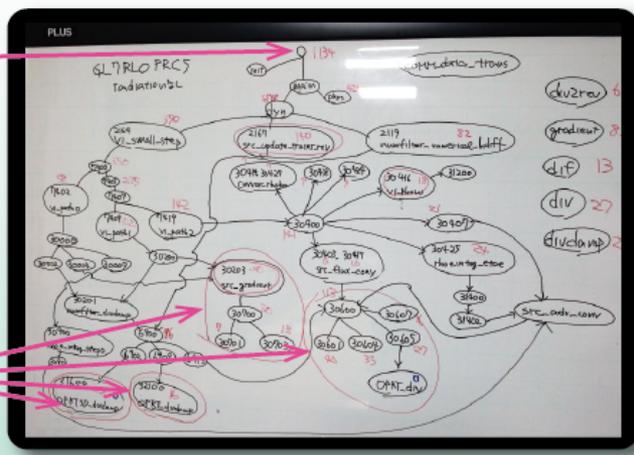
チューニングの最初のステップにおいて、エンジニアは静的および動的な解析結果を元にプログラムの論理構造を把握した上で最適化作業を開始します。とりわけ、コード開発者以外がチューニングを行う場合、その取り掛かりにおけるコード・リーディングで多くの時間と労力を消費します。特に、大規模計算機センター等の業務で複数の科学技術分野のソースコードに対してチューニングを行う場合、支援ツールの有無は作業の効率に重要な影響を与えます。

その一方で、チューニングに特化した支援ツールはそれほど多くはありません。特に Fortran 90 のソースコードを対象とする無償の支援ツールは皆無な状況です。

はじめに(2/2)

コードリーディングとホットスポットの特定 (以下は NICAM^[3] の例)

プログラム論理構造
のルート

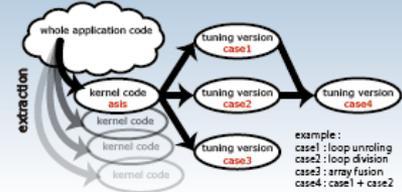


ホットスポット
カーネルの候補群

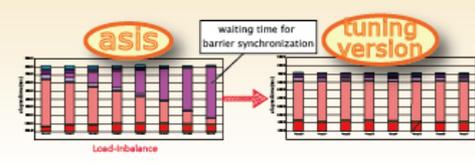
単体性能チューニング



カーネルコードの抽出と 1core 性能
チューニング



1chip (node) 性能チューニング



一般的に、プログラム構造を俯瞰的に理解する場合、ツリー構造に基づくトップダウンなアプローチが多用されます。いわゆるボトルネックとなるカーネルはそのツリー構造の末端に現れることが多く、本当に重要なルーチンかどうかはプロファイラのコスト情報等を用いて決定します。

“K-scope” は、Fortran 90 及び FORTRAN 77 コードのチューニングにおいて、ボトルネックとなり易いループ、分岐、およびプロシージャ呼び出しに代表されるプログラムの論理構造の可視化を軸とし、さらに「京」プロファイルデータに対応した、静的および動的解析に基づくプログラム構造解析支援ツールです。

1. ツールの起動

この章ではK-scopeのダウンロードから起動までを説明します。なお、K-scopeでは、Omni XcalableMPコンパイラのフロントエンドの出力結果を使用します。フロントエンドの使用方法については「[K-scope導入チュートリアル](#)」等を参照下さい。

起動方法

以下URLから実行形式(jar-executable)をダウンロードできます。

<http://www.aics.riken.jp/ungi/soft/kscope/>

ダウンロード後にパッケージを解凍します。Linuxの場合のコマンドによる起動方法は以下となります。

```
$ tar xvzf kscope_bin_${version}.tgz
```

`${version}` の部分を適当に置き換えてください。
(例: version 0.2.4の場合は、kscope_bin_0.2.4.tgz)

kscope.jar を実行することによりK-scopeは起動します。K-scopeは、Javaで構築されており、一般的なPC環境での動作を確認しています。

例) Intel CORE i5, メモリ4GB

なお、起動の際に、ダウンロードパッケージに含まれるプロパティファイル(propertiesディレクトリ配下のproperties.xml)が必要です。存在しない場合は、起動に失敗します。

```
$ java -jar kscope.jar
```

通常は、上記のような起動方法で問題ありませんが、大規模なアプリケーションコードを解析する場合、JVMに対して十分なヒープサイズを割り当ててください。指定には以下のオプションを使用します。

```
$ java -jar -Xms1024m -Xmx1024m kscope.jar
```

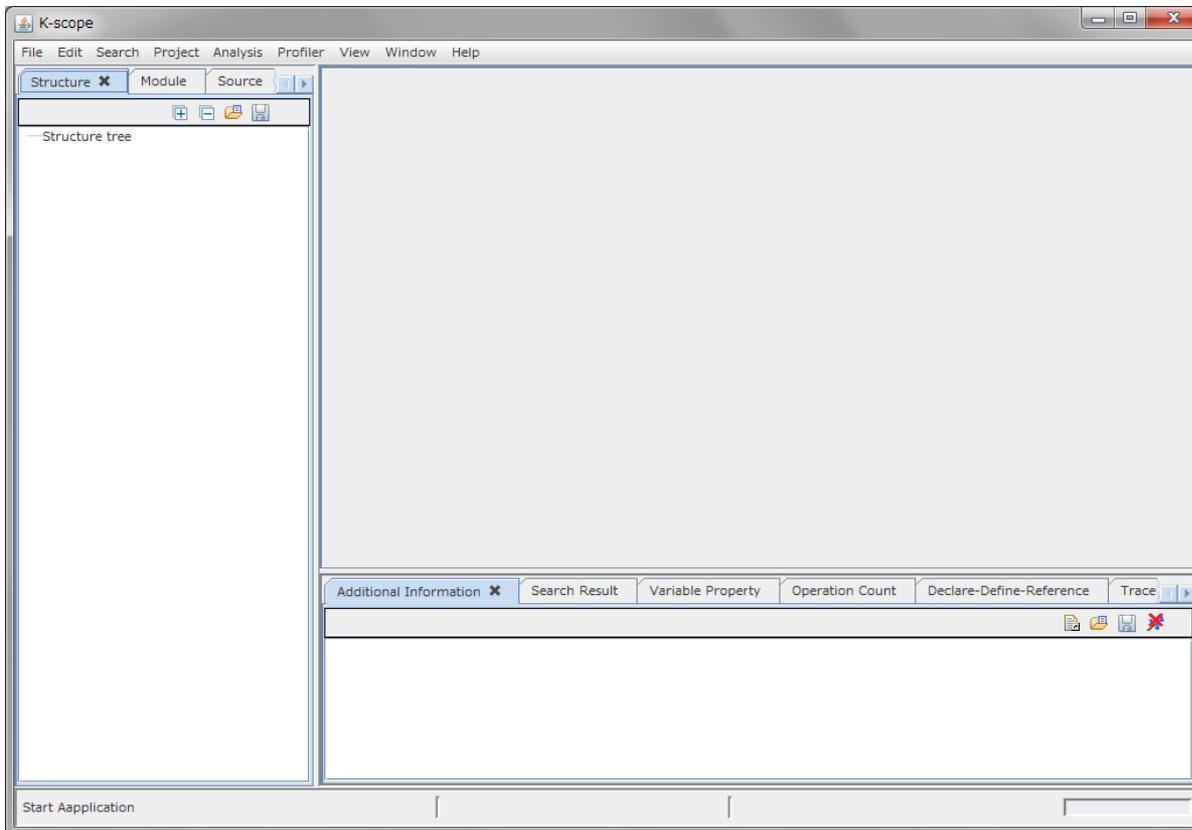
ここで、-Xmsは初期ヒープサイズを指定するオプション、-Xmxは最大ヒープサイズを指定するオプションです。続く数字がMBを表し、例では1024MBを割り当てる事になります。通常、物理メモリ以下の数値を指定してください。

また、Windows環境等では、JARファイルの関連付けを設定することで、kscope.jarをダブルクリックすることによる起動が可能です。

ロケールの指定

起動時にVMオプションでロケールを指定することで、メニューバー等の表記を英語または日本語に指定することができます。以下は英語にする場合です。日本語にする場合は、`-Duser.language=ja` とします。

```
$ java -jar -Duser.language=en kscope.jar
```



2. 新規プロジェクトの作成

K-scopeを用いた解析の第1歩は、プロジェクトを作成することから始まります。

プロジェクト

K-scopeは、中間コードとソースコードを元に、プログラムの構造情報を生成します。これは、Fortran構文と同等なJavaのクラス構造（オブジェクト）に基いて構築されます。この構造情報はメモリに展開されますが、必要に応じてファイルとして保存することもできます。

このようなプログラムの構造情報を軸とした、分析に必要なリソースを一括で管理できるようにしたものを、ここではプロジェクトと呼びます。

ツール再起動後に保存したプロジェクトを読み込むことで中間コードの選択、構造解析を行わなくても、ほぼ前回と同じ状態から作業を開始することができます。

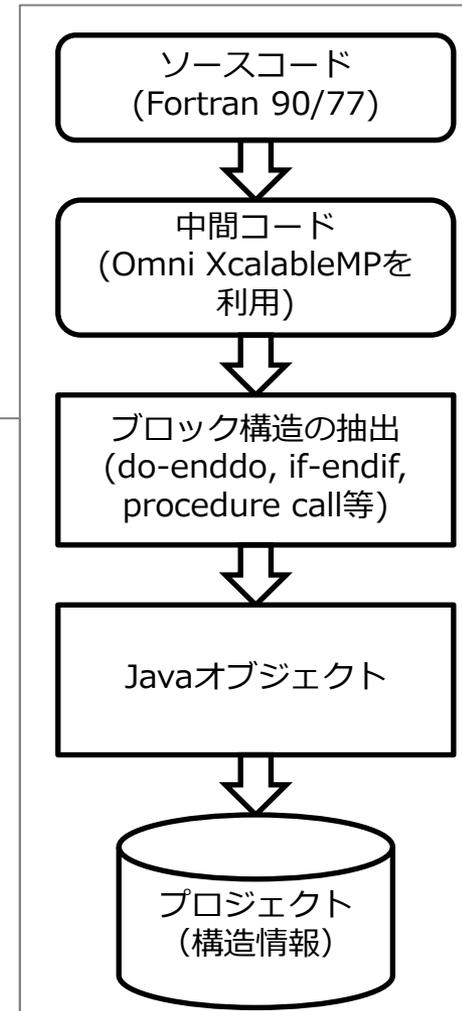
処理の流れとしてK-scopeは、まず始めにプログラム構造を静的に解析し、抽象構文木の構築を行います。この処理はコンパイラの字句解析、構文解析とほぼ同等の機能が要求されます。本ツールでは、筑波大学が開発しているOmni XcalableMPコンパイラのフロントエンドを外部プログラムとして利用することで技術的解決を図っています。

このフロントエンドは、FortranソースコードをXML形式の中間コードに変換します。これは抽象構文木に相当するものですが、チューニングするには詳細過ぎるため、得られた論理構造をフィルタリングすることで情報の間引きを行い、最終的な可視化で用いられるツリー構造を生成します。

支援ツールの内部では、Fortranソースコードと同等の構文情報をクラス定義し、操作は全てJavaオブジェクトに対して行われます。K-scopeでは、これらの関連情報をプロジェクトとして管理します。

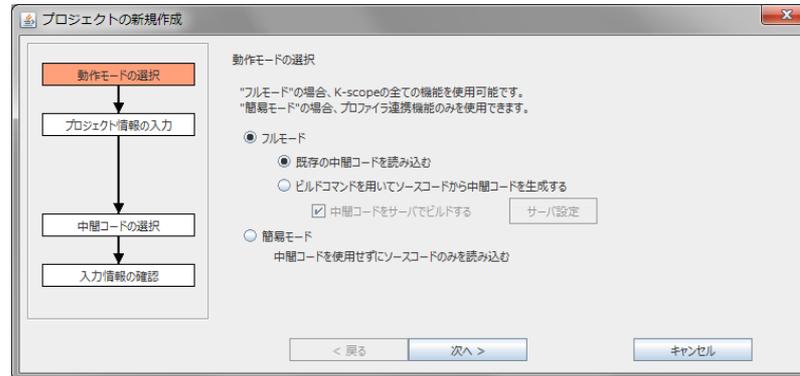
なお、プロジェクトはJavaのシリアライズ機能を使用しています。K-scopeのクラス構造やJVMの仕様が変更された場合は、異なるバージョン、JVM環境下では互換性が保証されない可能性があることをご了承ください。

(ここまで補足)



新規プロジェクトの作成（概要）

メニューバーから、「ファイル」 > 「プロジェクトの新規作成」を選択することでダイアログが表示されます。



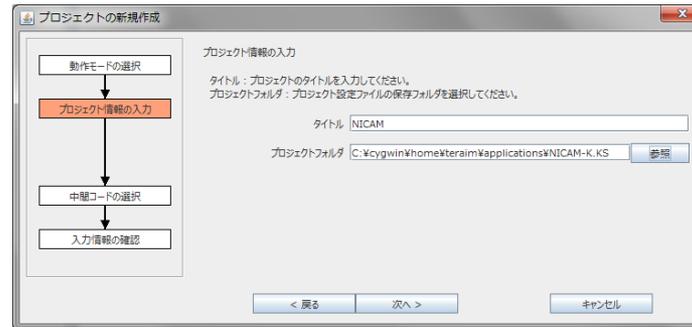
プロジェクトの新規作成には「フルモード」と「簡易モード」があります。さらにフルモードには、1) 既に生成された中間コードを利用する方法と、2) プロジェクト作成時に中間コードの生成を同時に行う方法があります。2)については、ローカルで生成する方法と、リモートで生成する方法があります。どちらにせよ、予め中間コードが生成されるMakefileを用意しておく必要があります。K-scopeはあくまでビルドコマンドをキックするだけとなります。

フルモード	中間コードを読み込み、プログラムの構造解析を行う時に選択します。このモードではK-scopeの全ての機能が使用できます。		
	既存の中間コードを読み込む	中間コードが用意できている場合はこちらを選択します。	
	ビルドコマンドを用いてソースコードから中間コードを生成する	ローカルで処理する	ビルドコマンドを使用してソースコードから中間コードを生成する場合はこちらを選択します。ビルドコマンドは一般的にmakeを想定していますが、シェルスクリプト等でも構いません。なお、makeの場合はMakefileのような、ビルド時に必要となる環境一式は予め用意しておく必要があります。チェックボックスを無効にした場合は、ローカル環境にF_Frontとatoolが用意されている必要があります。
		サーバ（リモート）で処理する	チェックボックスを有効した場合は、上記の処理をサーバ側で行います。サーバにSSHサービスが起動しており、ログインできることが条件となります。また、ログインしたサーバにF_Frontおよびatoolが用意されている必要があります。
簡易モード	ソースファイルのみを読み込みます。このモードではプロファイラとの連携機能は使用できますが、構造解析は行えません。		

選択したモードによって、ダイアログの左側に表示されているフロー図が変化します。

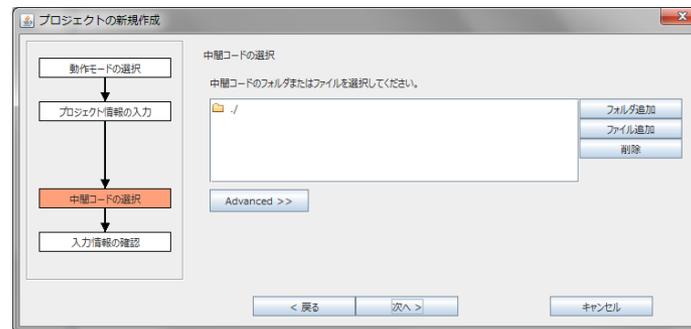
新規プロジェクトの作成(1/3)

フルモード > 1)既存の中間コードを読み込む方法を選択します。
以下のダイアログでプロジェクトのタイトルと、プロジェクトを保存するフォルダを指定します。



その後「次へ」を選択します。この際、指定したプロジェクトフォルダに既存プロジェクトが存在した場合、作業継続を確認する警告がでます。

通常は、以下のダイアログがすぐに表示されます。
ここで中間コードが置かれているフォルダを選択します。



選択後、「次へ」を選択します。

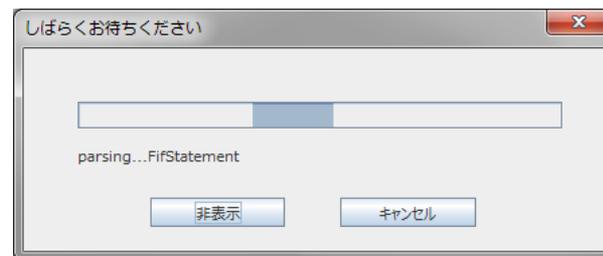
新規プロジェクトの作成(2/3)

以下のように、プロジェクトの最終確認のダイアログが表示されます。



問題なければ「新規作成」を選択します。

実行環境に依存しますが、しばらく経つと以下の様なプロジェクト作成の進捗ダイアログが表示されます。このフェーズでは中間コードのパーズを行います。処理時間は、主にソースコードの量に依存しています。



構築が完了すると、エクスプローラビューにツリーが表示されます。

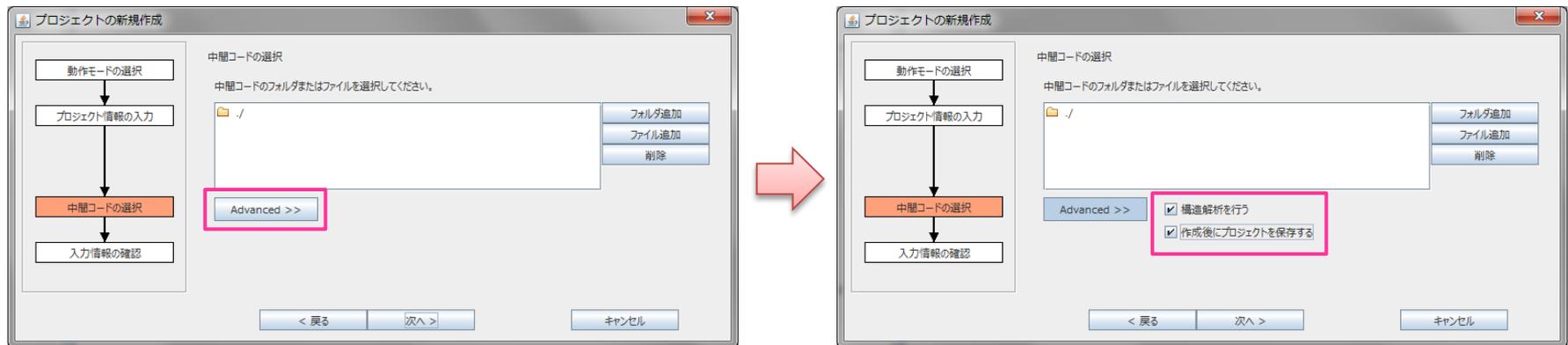
新規プロジェクトの作成(3/3)

現在開いているプロジェクトをプロジェクトフォルダに保存します。

保存される情報は以下となります。

- 構造情報
- 付加情報
- 設定項目 (ソースビュー設定、その他)
- 中間コード

TIPS) 新規プロジェクト作成時に「Advanced」 > 「作成後にプロジェクトを保存する」を選択することで、プロジェクト作成と同時に保存することも可能です。

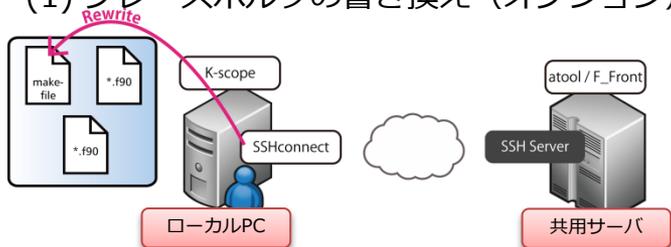


メニューバーから「ファイル」 > 「プロジェクトを閉じる」で、現在開いているプロジェクトを閉じます。構造情報、分析結果、ファイル情報はすべて破棄され、起動時の状態となります。保存されたプロジェクトは「ファイル」 > 「プロジェクトを開く」で再開することができます。

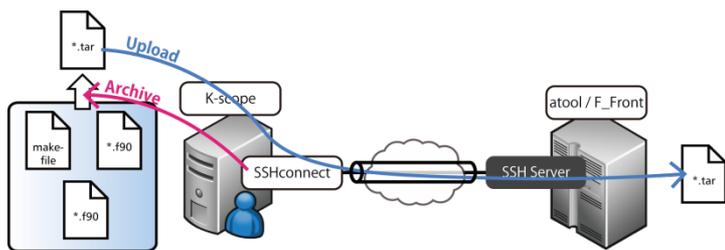
新規プロジェクトの作成・サーバの利用(1/5)

共用サーバ等に atool (F_Front) がインストールされている場合に、このサーバに対してSSHセッションを張り、ソースコード一式を・アップロードし、そこで中間コードを生成し、再度ローカルにダウンロードする機能^{#1}を提供します。大きく以下の5つのプロセスからなります。

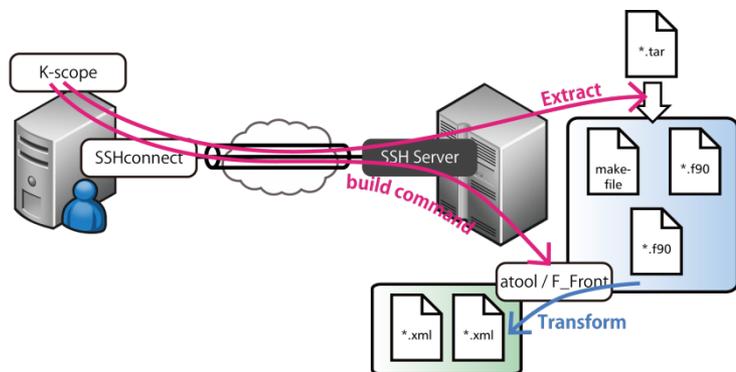
(1) プレースホルダの書き換え (オプション)



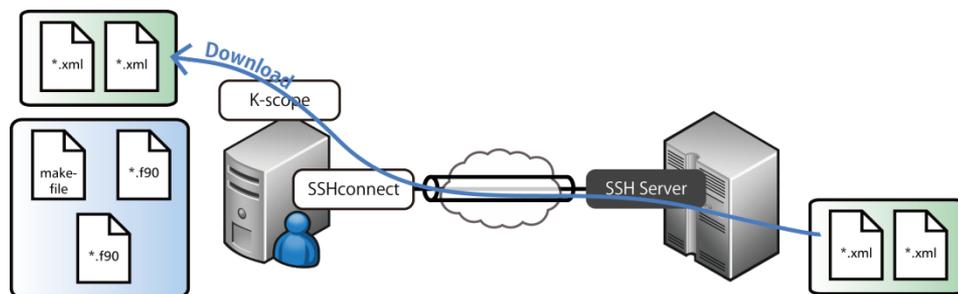
(2) アーカイブ + アップロード



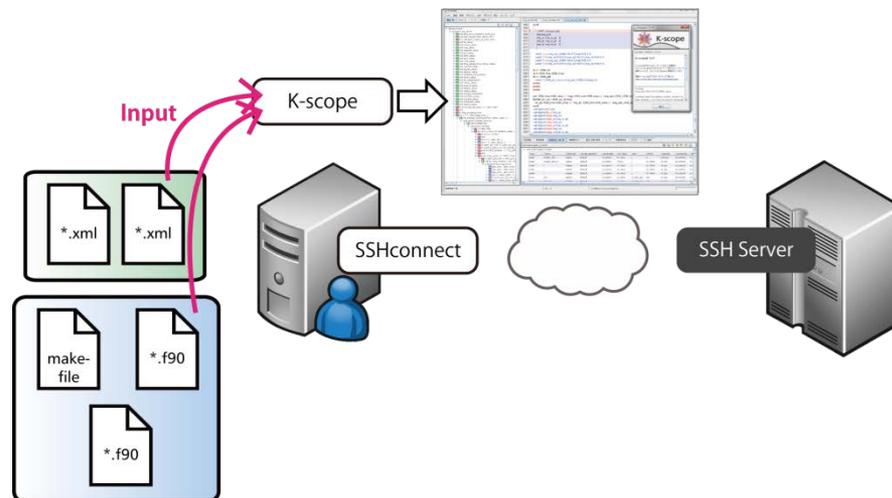
(3) 展開 + 中間コードへの変換



(4) ダウンロード



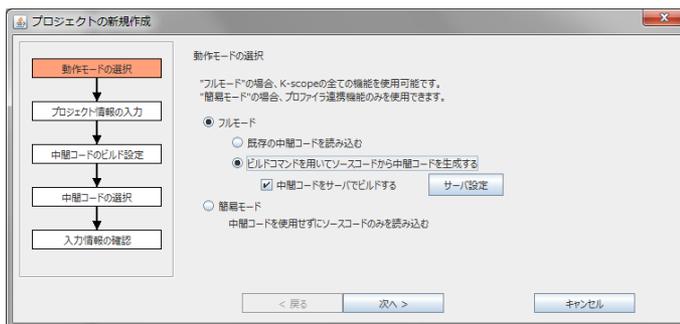
(5) K-scopeによる読み込み (構造解析)



#1) SSHを用いた転送ライブラリには、AICS利用高度化研究チームとの共同プロジェクトにより開発されたSSHconnectを利用しています。

新規プロジェクトの作成・サーバの利用(2/5)

フルモード > 1)ビルドコマンドを用いてソースコードから中間コードを生成する方法を選択します。中間コードをサーバ側でビルドするようにチェックボックスを有効にします。



その後、サーバ設定ボタンを選択すると、以下のようなウィンドウが表示されます。ここでは、SSHサービスが起動しているサーバを設定します。設定を完了したらOKを選択してください。なおSSHの接続にはパスワード認証と公開鍵認証の両方をサポートしています。

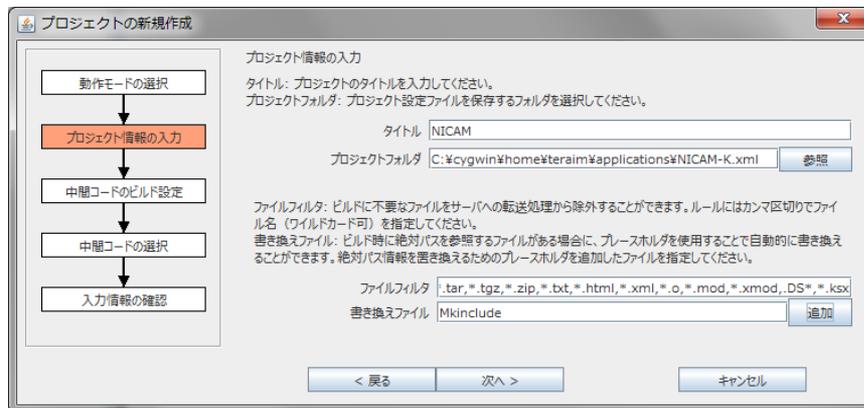


パラメータ名	値	記述
server_address	k.aics.riken.jp	サーバアドレス
port	22	SSHサービスを提供しているTCPポート番号
user		サーバに接続するためのユーザ名
password		サーバに接続するためのパスワード
key		サーバに接続するための秘密鍵
passphrase		サーバに接続するための秘密鍵のパスフレーズ
add_path	/opt/aics/omni/bin	サーバ側でのatool(F_Front)コマンドへのパス
remote_path	/home/teraim/	サーバ側の作業ディレクトリ
local_path		プロジェクトフォルダのパス
preprocess_files		サーバの絶対パスに置き換えるべきプレースホルダのあるファイル
file_filter	*.tar, *.tgz, *.zip, *.bt, *.html, *.x...	ビルドに不要なファイルを転送から除外することができます。ファイル名にはワイルドカードが使用で...

最初のダイアログに戻り、「次へ」を選択します。

新規プロジェクトの作成・サーバの利用(3/5)

以下のダイアログでプロジェクトのタイトルと、プロジェクトを保存するフォルダを指定します。



ファイルフィルタのテキストフィールドには、ビルドに不要なファイルを転送から除外するための指定が可能です。ファイル名には、ワイルドカードが使用できます。（例: *.txt）

例えば、Makefileにビルド時に参照される絶対パスが記述されている場合、その絶対パスの部分を予め、メタ文字としてのプレースホルダ（#[remote_path]）に置き換えることで、転送時にK-scope (SSHconnect) が自動的にサーバ側の絶対パスに置き換える機能を提供します（本機能はオプションです。）。

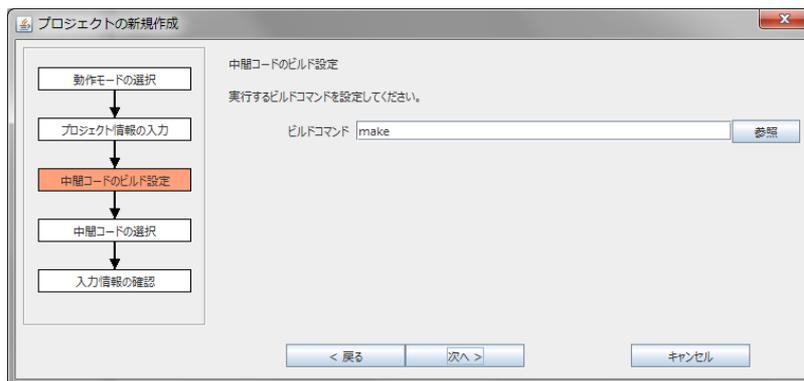
以下はサンプルです。

```
$ vi Mkinclude
#TOPPATH=/home/teraim/Work/NICAM-K
TOPPATH=#[remote_path]
```

書き換えファイルのテキストフィールドには、書き換え対象となるファイルを指定してください。上記サンプルでは Mkinclude となります。

新規プロジェクトの作成・サーバの利用(4/5)

ビルドコマンドをここで指定します。make以外にシェルスクリプト等（例: ./build.sh）の実行も可能です。ここで指定されたコマンド名、オプションはそのままサーバ側で実行されるのでご注意ください。



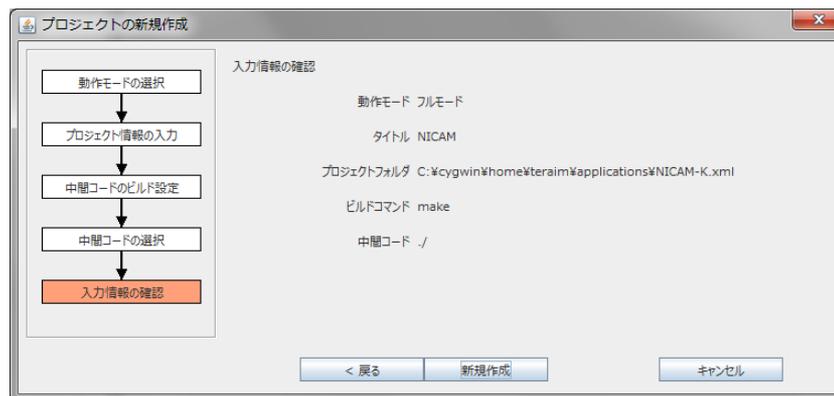
その後「次へ」を選択します。この際、指定したプロジェクトフォルダに既存プロジェクトが存在した場合、作業継続を確認する警告がでます。

通常は、以下のダイアログがすぐに表示されます。カレントよりも上位にあるディレクトリについては、転送をサポートしていないため、このモードでは中間コードのディレクトリを追加することはできません。「次へ」を選択ください。



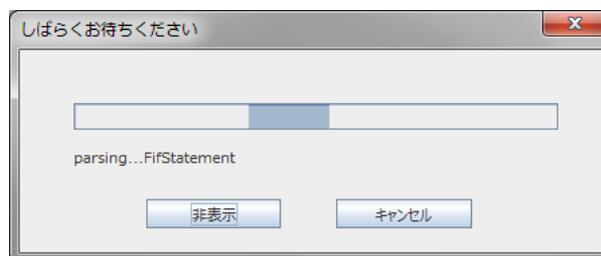
新規プロジェクトの作成・サーバの利用(5/5)

以下のように、プロジェクトの最終確認のダイアログが表示されます。



問題なければ「新規作成」を選択します。

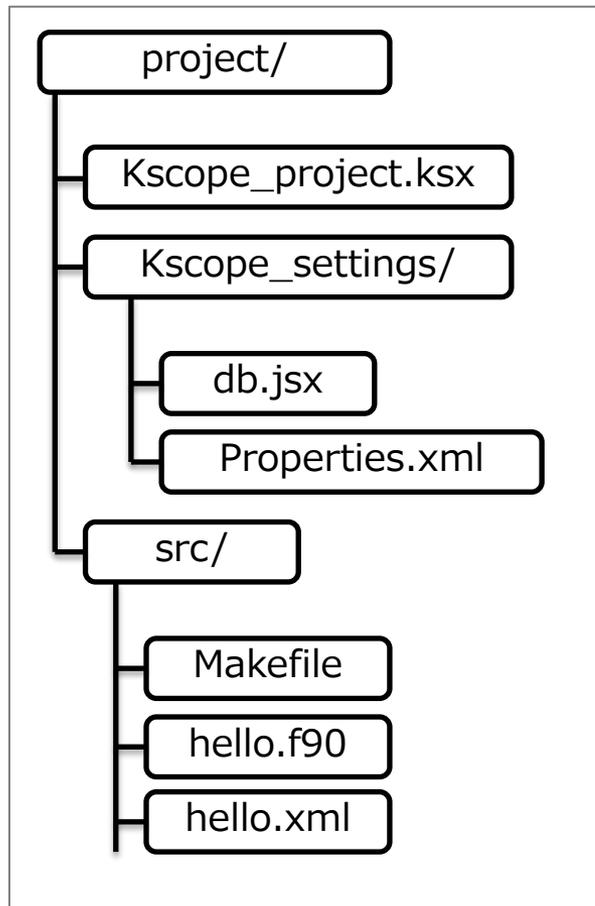
実行環境に依存しますが、しばらく経つと以下の様なプロジェクト作成の進捗ダイアログが表示されます。処理時間は、他のモード同様に主にソースコードの量に依存しています。



構築が完了すると、エクスプローラビューにツリーが表示されます。

プロジェクトの詳細

プロジェクトの構成例は以下となります。



構成フォルダ・ファイル		説明
プロジェクトフォルダ (project)		プロジェクトのルートとなるフォルダです。 プロジェクト名は任意です。
プロジェクト設定フォルダ・ファイル		プロジェクトの保存により自動生成される フォルダ・ファイルです。フォルダ・ファイル は変更、削除しないでください。
	プロジェクト基本情報ファイル (Kscope_project.ksx)	プロジェクトの基本情報を保存します。
	プロジェクト設定フォルダ (Kscope_settings)	プロジェクト毎の設定、構造情報、付加情報 を格納するフォルダです。
	データベースファイル (db.ksx)	構造情報、付加情報を保存するファイルです。
	プロジェクト設定ファイル (properties.xml)	プロジェクトで設定したキーワード、ソース ビュー等の設定情報を保存するファイルです。 補足 I に詳細が記載しています。
ソースコードと中間コード (src)		ソースコードと中間コードを配置するフォル ダです。フォルダ名は任意です。 TIPS) K-scopeにより「プロジェクト設定 フォルダ・ファイル」が生成されます。ソー スコードはプロジェクト配下にコード用の フォルダ（この場合src）を作成し配置する ことを推奨します。

ソースコードから中間コードを生成後、これらの相対位置関係を変更しないで下さい。中間コードには、ソースコードへの相対パスが記述されています。中間コードのみを移動・削除すると、ソースコードが参照できなくなります。

3. インターフェイス

K-scopeでは、グラフィカルユーザインターフェイスを通して、すべての操作と表示が行われます。

インターフェイス

プロジェクト作成後のスクリーンショット例を示します。

K-scopeは、以下のようなインターフェイスを採用しており、「エクスプローラビュー」に静的に解析されたプログラム構造のツリー、「ソースビュー」に選択されたツリー要素に対応するソースコード、「分析ビュー」に変数の特性一覧、浮動小数点演算数のカウント等の分析結果を提供します。

エクスプローラビューを用いた操作は本ツールの基本となる部分なので、はじめにプログラム構造の表示と操作方法について説明します。

メニューバー

エクスプローラビュー
(構造、モジュール、ソース、XML)

ソースビュー

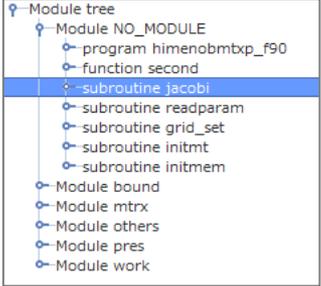
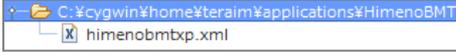
分析ビュー

ステータスバー

type	name	data type	access specifier	parameter	init value	size	intent	optional	pointer/tar...	save
scalar	alpha	real(8)	default	no param	0.0d0	1	no intent	no opt	no pointer	no sa
scalar	g	integer	default	no param	no value	1	no intent	no opt	no pointer	no sa
scalar	k	integer	default	no param	no value	1	no intent	no opt	no pointer	no sa
scalar	l	integer	default	no param	no value	1	no intent	no opt	no pointer	no sa
array	vx	real(8)	default	no param	no value	(1:adm_gal... in	no intent	no opt	no pointer	no sa
array	vy	real(8)	default	no param	no value	(1:adm_gal... in	no intent	no opt	no pointer	no sa
array	vy_pl	real(8)	default	no param	no value	(1:adm_gal... in	no intent	no opt	no pointer	no sa

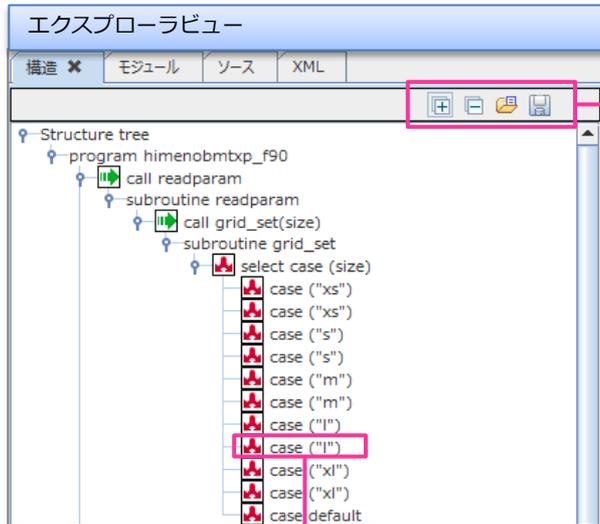
エクスペローラビュー(1/2)

エクスペローラビューでは、ソースコードの構造情報、モジュール、ソースファイル、中間コードファイルをツリー形式による階層表示をします。それぞれのビューはタブにて切り替えを行います。

		エクスペローラビュー
構造	ソースコードの構造情報を表示します。 do文、if文、call文等をツリー階層にて表示します。	
ソース	ソースファイルをディレクトリ階層にて表示します。	
中間コード	中間コードファイルをディレクトリ階層にて表示します。	

エクスプローラビュー(2/2)

エクスプローラビューのタブ右上には、ツリーに対する操作を提供するアイコンがあります。それぞれのアイコンの意味は以下のテーブルの通りです。



ツリー要素をダブルクリックするとソースビューに該当箇所が表示されます。この例では、`case("L")`に該当する部分がハイライトされます。

```
200  mjimax=257
201  mkmax=257
202  case("L")
203  mimax=513
204  mjimax=257
205  mkmax=257
206  case("xl")
207  mimax=1025
208  mjimax=513
209  mkmax=513
210  case("XL")
211  mimax=1025
212  mjimax=513
213  mkmax=513
```

	すべて展開	ツリーをすべて展開して表示します。大きなソースコードの場合は展開に時間が掛ります。
	すべて収納	ツリーをすべて収納して表示します。
	選択箇所を開く	ツリー上の選択ノードをソースビューに表示します。該当行がアクティブとなります。
	エクスポート	ツリー表示をテキストファイルにエクスポートします。

4. 静的プログラム構造解析

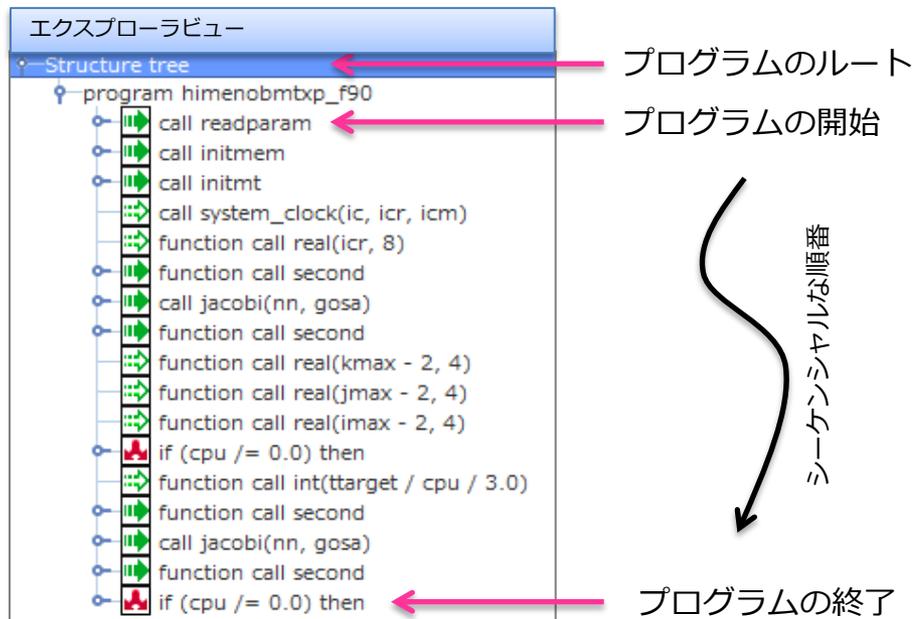
K-scopeは、静的プログラム構造解析を軸とした分析ツールです。静的な解析は本質的には環境依存がなく、また一般的に、動的解析に比べて実行時に大量に出力される情報が解析に必要なことから、比較的小さなコストで解析が行えることが特徴です。

静的プログラム構造解析(1/2)

静的解析ではソースコードを実行せずに、プログラムの論理構造をツリー形式で表示します。利点としては、煩雑になりやすい実行情報が必要なく、小さなコストでソースコードを解析することが可能です。

特徴は、一般的なコールグラフがプロシージャの呼び出し関係のみに注目してグラフを構築するのに対して、K-scopeではループや分岐等についてもノードとして扱うことで、ボトルネック箇所との対応付けが容易になります。また、同じプロシージャがプログラム中で複数回呼び出されている場合、K-scopeではそのプロシージャに該当するノードが複数回出現します。そのため、正確にはグラフではなくツリーになります。ツリーであることの利点は、プログラムのシーケンシャルな流れをトレースすることが容易になります。

ツリー構造はエクスプローラビューの構造タブに表示されます。以下は姫野ベンチマークの場合のツリーとなります。



ノード先頭にあるアイコンは以下のテーブルの通りです。

	分岐	if文、else文、where文、select文、case文に対応します。
	ループ	do文に対応します。
	プロシージャ呼び出し (実体あり)	プロシージャの呼び出し先が中間コードとして提供されている場合に対応します。
	プロシージャ呼び出し (実体なし)	プロシージャ呼び出し先が中間コードとして提供されていない場合に対応します。(mpi関数等)

ツリーは実行単位としてのプログラム全体を表しており、ルートは常に1つになります。ソースコードのFortran構文中で、性能に影響を与える「分岐」、「ループ」、「プロシージャ呼び出し」についてフィルタリングし、可視化しています。

ツリーは、プログラムの開始ステートメントから終了ステートメントに至るシーケンシャルな順番を抽象化し表示しています。

静的プログラム構造解析(2/2)

姫野ベンチマークのカーネルであるjacobiサブルーチンについて、ツリー展開すると以下のようになります。最内からI,J,Kがステンシル計算部分になります。

エクスプローラビュー

```
Structure tree
├── program himenobmtxp_f90
│   ├── call readparam
│   ├── call initmem
│   ├── call initmt
│   ├── call system_clock(ic, icr, icm)
│   ├── function call real(icr, 8)
│   ├── function call second
│   └── call jacobi(nn, gosa)
│       └── subroutine jacobi
│           ├── do loop = 1, nn, 1
│           │   ├── do k = 2, kmax - 1, 1
│           │   │   ├── do j = 2, jmax - 1, 1
│           │   │   │   └── do i = 2, imax - 1, 1
│           │   │   │       └── p(2:imax - 1, 2:jmax - 1, 2:kmax - 1) = wrk2(2:imax - 1, 2:jmax - 1, 2:kmax - 1)
│           │   │   └── function call second
│           │   └── function call real(kmax - 2, 4)
│           └── function call real(jmax - 2, 4)
│               ├── function call real(imax - 2, 4)
│               ├── if (cpu /= 0.0) then
│               │   ├── function call int(ttargt / cpu / 3.0)
│               │   ├── function call second
│               │   ├── call jacobi(nn, gosa)
│               │   ├── function call second
│               └── if (cpu /= 0.0) then
```

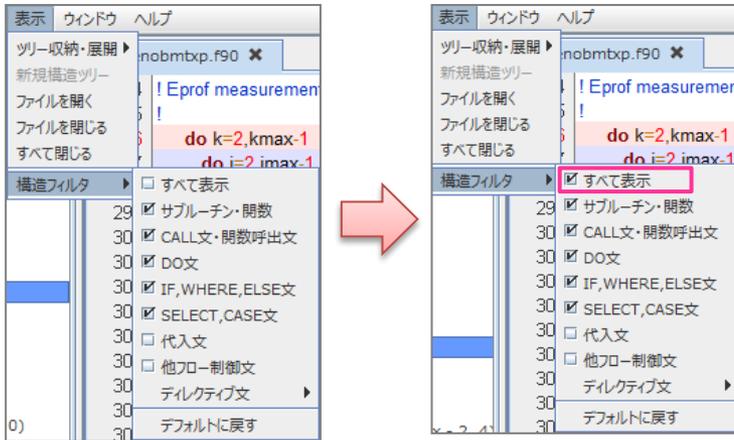
ソースビュー

```
296 do k=2,kmax-1
297 do j=2,jmax-1
298 do i=2,imax-1
299 s0=a(l,j,k,1)*p(l+1,j,k) &
300 +a(l,j,k,2)*p(l,j+1,k) &
301 +a(l,j,k,3)*p(l,j,k+1) &
302 +b(l,j,k,1)*(p(l+1,j+1,k)-p(l+1,j-1,k) &
303 -p(l-1,j+1,k)+p(l-1,j-1,k)) &
304 +b(l,j,k,2)*(p(l,j+1,k+1)-p(l,j-1,k+1) &
305 -p(l,j+1,k-1)+p(l,j-1,k-1)) &
306 +b(l,j,k,3)*(p(l+1,j,k+1)-p(l-1,j,k+1) &
307 -p(l+1,j,k-1)+p(l-1,j,k-1)) &
308 +c(l,j,k,1)*p(l-1,j,k) &
309 +c(l,j,k,2)*p(l,j-1,k) &
310 +c(l,j,k,3)*p(l,j,k-1)+wrk1(l,j,k)
311 ss=(s0*a(l,j,k,4)-p(l,j,k))*bnd(l,j,k)
312 GOSA=GOSA+SS*SS
313 wrk2(l,j,k)=p(l,j,k)+OMEGA *SS
314 enddo
315 enddo
316 enddo
```

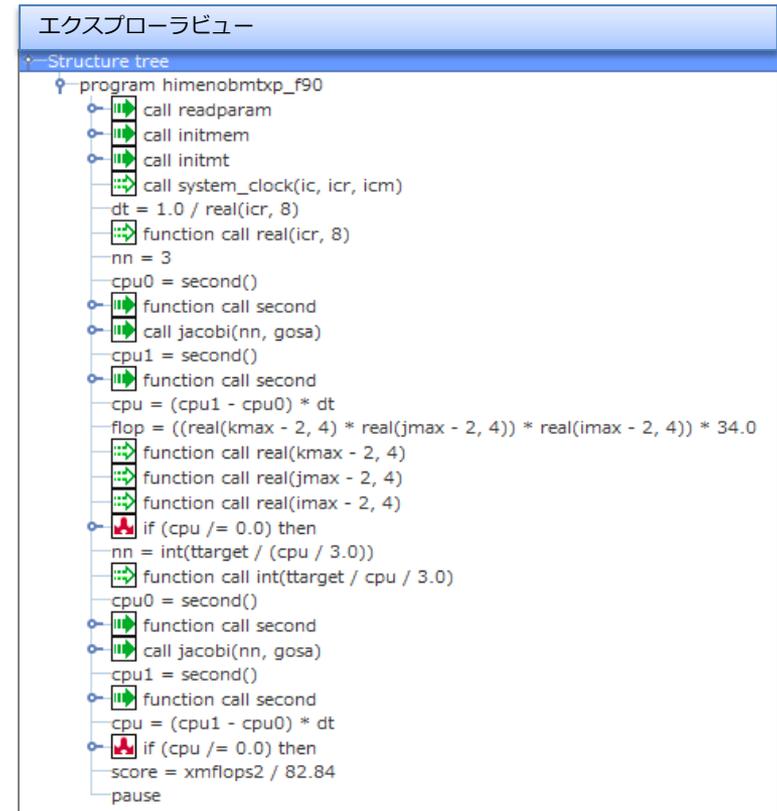
エクスプローラビュー上の該当ノードを選択することで、ソースビューに該当箇所がハイライトされて表示されます。

フィルタリング機能

メニューバーから「表示」>「構造フィルタ」を開き、すべて表示にチェックを入れることで、Javaのオブジェクトに変換されているすべてのFortranステートメントがツリーに表示されます。



反対にチェックを外すことで、必要のない構文情報をフィルタリングすることが可能です。



5. 検索機能

ソースコード、ファイル、ツリーの各レベルから任意の文字列を検索することができます。膨大なソースコードから注目箇所を効率的にスクリーニングするには必須の機能です。

検索機能

エクスプローラビューのツリー表示、およびソースファイルに対してテキスト検索を行うことができます。

K-scopeでは以下の3種類の検索機能を提供します。

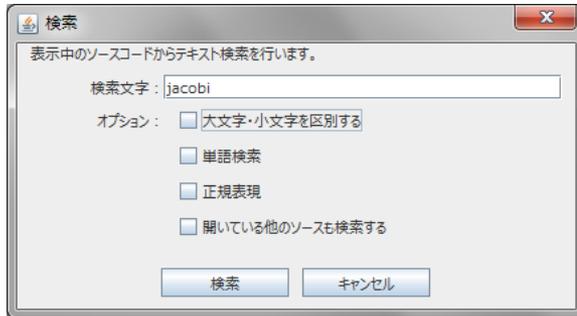
ソース検索	ソースビューにて現在選択されているソースファイルに対してテキスト検索を行います。
ファイル検索	プロジェクトのソースファイルに対してテキスト検索を行います。
ツリー検索	現在開いているエクスプローラビューのツリーノードに対してテキスト検索を行います。エクスプローラビューの構造ツリー、モジュールツリー、ソースツリー、XMLツリーに対して検索を行うことができます。

(注意) ソース検索、ファイル検索、ツリー検索の検索可能件数は1000件程度です。検索結果件数が1000件を超えた場合はエラーメッセージを表示して中断し、途中までの検索結果は表示します。エラーが発生した場合は、検索条件を見直して再検索を行ってください。

ソース検索

ソースビューにて現在選択されているソースファイルに対してテキスト検索を行います。メニューバーの「検索」>「ソース検索」を選択すると、以下のダイアログが表示されます。

ダイアログに表示されているチェックボックス、ボタンの意味は以下のテーブルの通りです。



検索文字	検索対象となる文字列を入力します。
大文字・小文字を区別する	チェック：ON 大文字・小文字の区別をした検索を行います。
単語検索	チェック：ON 検索文字が単語であるものを検索対象とします。
正規表現	チェック：ON 検索文字に正規表現を記述できます。
開いている他のソースも検索する	チェック：ON ソースビューに開いている他のソースファイルも対象とし検索を行います。
検索	検索を実行して、検索ダイアログを閉じます。
キャンセル	検索は行わず、検索ダイアログを閉じます。

検索結果は分析ビューの「検索結果」タブに表示されます。ヒットした文字列はハイライトされます。分析ビューの各項目を選択することで、ソースビューに該当箇所が表示されます。

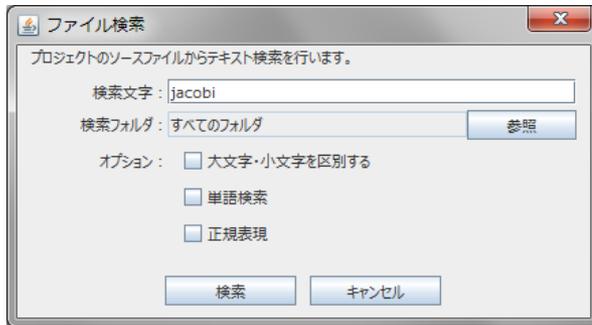
以下は「jacobi」で検索した場合の例です。



ファイル検索

プロジェクトのソースファイルに対してテキスト検索を行います。
メニューバーの「検索」>「ファイル検索」を選択すると以下ダイアログが表示されます。

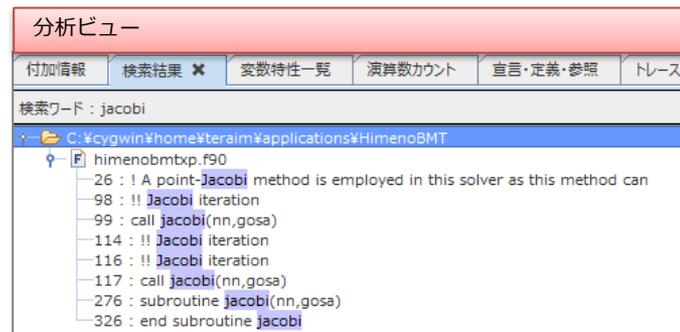
ダイアログに表示されているチェックボックス、ボタンの意味は以下のテーブルの通りです。



検索文字	検索対象となる文字列を入力します。
検索フォルダ	プロジェクトフォルダ配下フォルダから検索対象のフォルダを選択します。
大文字・小文字を区別する	チェック：ON 大文字・小文字の区別をした検索を行います。
単語検索	チェック：ON 検索文字が単語であるものを検索対象とします。
正規表現	チェック：ON 検索文字に正規表現を記述できます。
検索	検索を実行して、検索ダイアログを閉じます。
キャンセル	検索は行わず、検索ダイアログを閉じます。

検索結果は分析ビューの「検索結果」タブに表示されます。
ヒットした文字列はハイライトされます。
分析ビューの各項目を選択することで、ソースビューに該当箇所が表示されます。

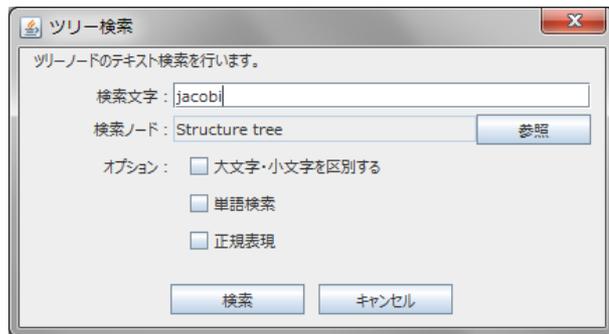
右図は「jacobi」で検索した場合の例です。
姫野ベンチマークの場合は、ソースファイルが1つしかないため、ソース検索と同じ結果になります。



ツリー検索

現在開いているエクスプローラビューのツリーノードに対してテキスト検索を行います。
構造ツリー以外に、モジュールツリー、ソースツリー、XMLツリーに対して検索を行うことができます。

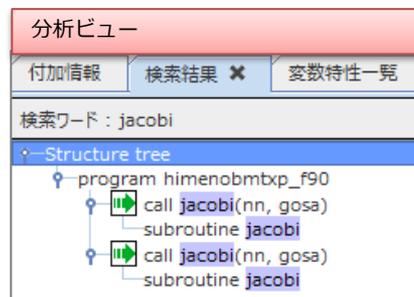
メニューバーの「検索」>「ツリー検索」を選択すると以下ダイアログが表示されます。
ダイアログに表示されているチェックボックス、ボタンの意味は以下のテーブルの通りです。



検索文字	検索対象となる文字列を入力します。
検索ノード	検索を行うツリーから検索対象のノードを選択します
大文字・小文字を区別する	チェック：ON 大文字・小文字の区別をした検索を行います。
単語検索	チェック：ON 検索文字が単語であるものを検索対象とします。
正規表現	チェック：ON 検索文字に正規表現を記述できます。
検索	検索を実行して、検索ダイアログを閉じます。
キャンセル	検索は行わず、検索ダイアログを閉じます。

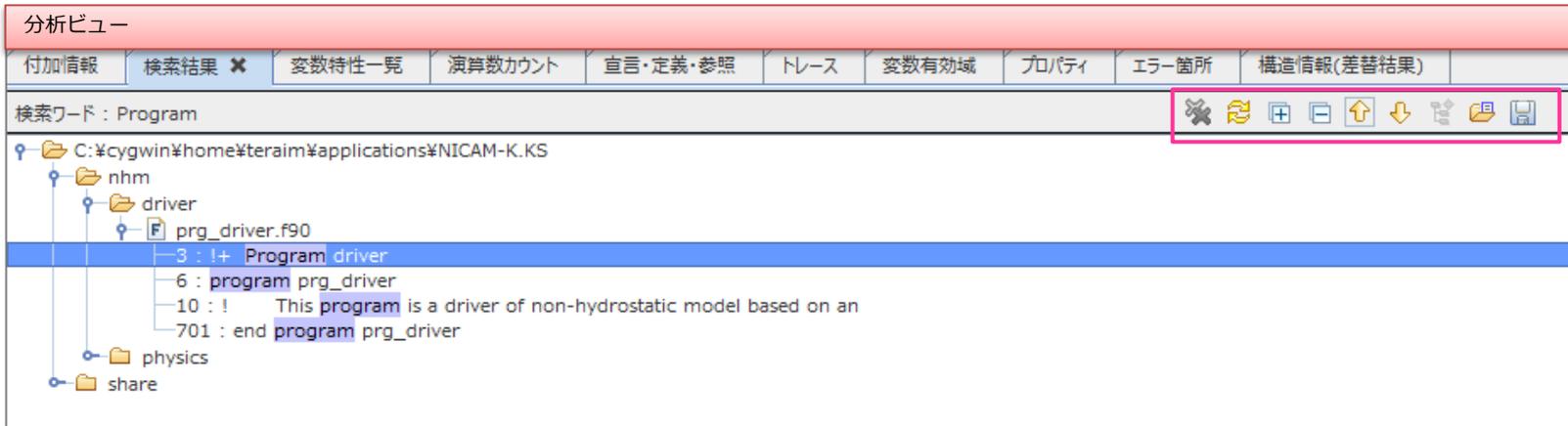
検索結果は分析ビューの「検索結果」タブに表示されます。
ヒットした文字列はハイライトされます。
分析ビューの各項目を選択することで、ソースビューに該当箇所が表示されます。

右は「jacobi」で検索した場合の例です。姫野ベンチマークでは、1回の実行で2度 jacobi サブルーチンが呼ばれているため、それぞれの箇所検索がヒットします。



検索結果タブ（補足）

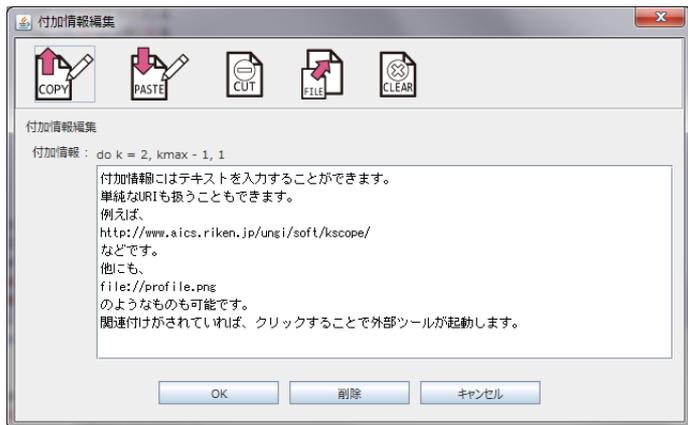
分析ビューの「検索結果」タブの右上には操作アイコンが用意されています。機能は以下のテーブルの通りです。



	クリア	検索結果をクリアします。
	検索結果更新	検索結果の表示を更新します。 ソース検索、ファイル検索の場合は、ソースビューの検索文字がハイライトしますが、トレースを行うとトレース対象の変数がハイライトします。ハイライト文字を検索文字に戻す場合に検索結果更新を選択してください。
	すべて展開	検索結果のツリーをすべて展開します。
	すべて収納	検索結果のツリーをすべて収納します。
	前へ	検索結果の前の該当箇所をアクティブにします。 ソースビューの検索結果行がハイライトします。 ソース検索、ファイル検索の場合は、ソースビューの検索文字がハイライトします。
	次へ	検索結果の次の該当箇所をアクティブにします。 ソースビューの検索結果行がハイライトします。 ソース検索、ファイル検索の場合は、ソースビューの検索文字がハイライトします。
	新規構造ツリー	選択したプロシージャをルートとする構造ツリーを構築し、エクスプローラビューに表示します。
	検索結果箇所を開く	該当するソースコードをソースビューに表示します。
	エクスポート	検索結果をcsv形式で出力します。

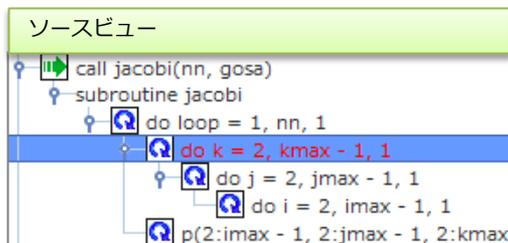
付加情報機能

エクスプロービューの構造タブのノードを選択した後に、メニューバーから「編集」>「付加情報編集」を選択すると、以下の様なダイアログが表示されます。ダイアログ中のテキストフィールドには、任意のテキストを入力することができます。URIをテキストとして登録することで、外部ツールと連動させることが可能です。



	コピー	付加情報から選択範囲をクリップボードにコピーします。
	ペースト	付加情報にクリップボードから貼り付けを行います。
	カット	付加情報から選択範囲をクリップボードに切り取ります。
	ファイルコピー	ファイルダイアログを表示して、選択されたファイル名を付加情報に挿入します。プロジェクトフォルダ配下のファイルの場合は、相対パスで挿入しますが、以外の場合は絶対パスとなります。
	クリア	付加情報をクリアします。

付加情報を追加すると、「構造タブ」のツリーの文字色（デフォルトは赤色）が変更され、分析ビューの「付加情報」タブにその追加された情報を確認することができます。関連付けがされている場合、URIの部分は、クリックすることで外部ツールが起動します。URIがhttp://で始まる場合はブラウザを起動させたり、画像ビューワやAcrobat Reader等を起動させることができます。



クリックすると外部ツールが起動します。（実行環境に依存します）

7. 「京」プロファイラ連携機能

プログラム構造の静的解析と動的解析は、片方があれば十分というものではなく、それぞれ相補的なものです。K-scopeは静的なプログラム構造解析ツールとして開発されてきた経緯がありますが、「京」のプロファイラ結果を読み込むことで、動的プログラム解析機能を強化しています。

動的プログラム解析

プログラムの構造解析には、静的解析と動的解析に大別されます。静的解析はソースコードの構文情報に基づく解析のみを行います。実機等による実行環境が解析に必要なことが最大の利点です。

一方、プログラムの実際の動作は実行時に決定される要素が多々あるため、動的解析と呼ばれる手法が用いられます。動的解析では、実際にプログラムを動作させる必要性から実機またはエミュレータが必要となります。

K-scopeでは、「京」を実機とし、そのプロファイラ結果をソースコードと連動させる機能を提供することで、動的解析機能を強化しています。ここでは、富士通が提供する基本プロファイラ(DProf)と詳細プロファイラ(Eprof)の出力データに対応することで、コスト情報やコールグラフを可視化することができます。

静的解析で得られたツリー構造とソースコードに、プロファイラのコスト情報を組み合わせることで、静的解析だけでは決定性に欠けたボトルネックを特定できることが、本機能の最大の利点です。

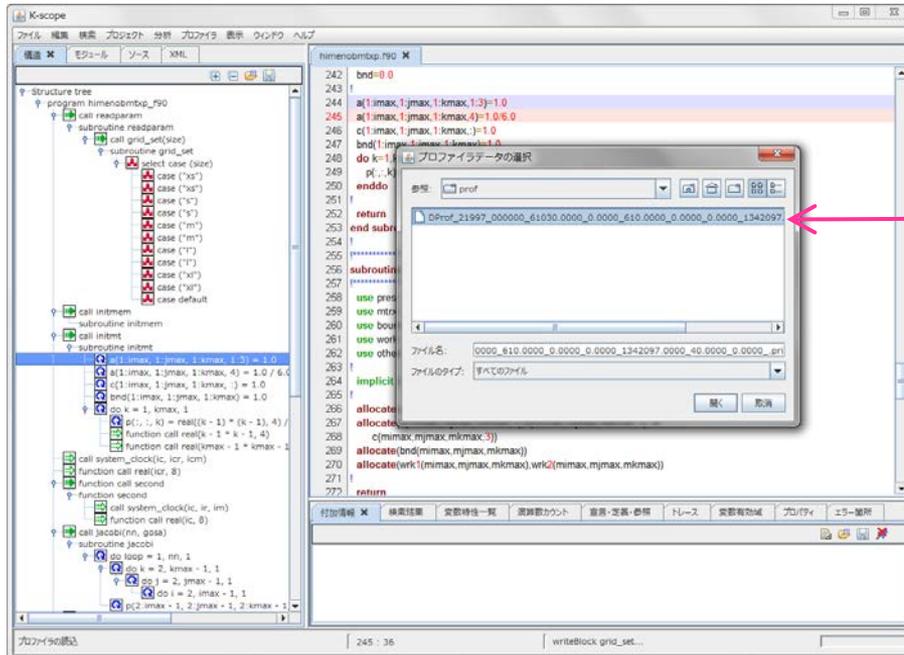
本機能のもう1つの特徴は、中間コードを用いた構造解析が何らかの理由で行えない場合も、ソースコードとプロファイルデータのみで一部の分析機能の利用が可能です。これは、K-scopeのプロジェクト作成時に簡易モードを選択することで行えます。

なお、プロファイラデータの取得方法については富士通提供のマニュアルを参照下さい。

基本プロファイラとの連携(1/3)

はじめにプロジェクトを構築します。(新規プロジェクトの構築方法は、§2を参照下さい。)

その後、メニューバーから「プロファイラ」>「プロファイラの読み込み」を選択し、Dprof で始まる、基本プロファイラの実行結果を読み込みます。



読み込みが完了すると、分析ビューの「コスト情報」タブに以下のような情報が表示されます。

分析ビュー							
トレース	変数有効域	プロパティ	エラー箇所	コスト情報: 手続 *	コスト情報: ループ	コスト情報: ライン	コールグラフ情報
DProf_21997_000000_61030.0000_0.0000_610.0000_0.0000_0.0000_1342097.0000_40.0000_0.0000_...							
サンプリング数	全体に占める割合(%)	手続	ファイル名	行番号			
604	99.67	jacobi_	himenobmbxp.f90	276:329			
1	0.17	__brk					
1	0.17	initmt_	himenobmbxp.f90	224:253			

基本プロファイラとの連携(2/3)

基本プロファイラは、手続き、ループ、ラインに対する粒度でサンプリングによるコスト情報を収集します。例えば、姫野ベンチマークの場合、手続きでは `jacobi_` が高コスト部として検出されています。次に分析ビューの「コスト情報：手続き」を選択することで、ソースビューに該当箇所が表示されます。

分析ビュー				
トレース	変数有効域	プロ/タイ	エラー箇所	コスト情報：手続き ✕
DProf_21997_000000_61030.0000_0.0000_610.0000_0.0000_1342097.0000_40.0000_0.0000_pri				
サンプリング数	全体に占める割合(%)	手続き	ファイル名	行番号
604	99.67	jacobi_	himenobmbxp.f90	276:329
1	0.17	__brk		
1	0.17	initmt_	himenobmbxp.f90	224:253



```
ソースビュー
himenobmbxp.f90 ✕
274 !
275 !*****
276 subroutine jacobi(nn,gosa)
277 !*****
278 use pres
279 use mtrx
280 use bound
281 use work
282 use others
283 !
284 implicit none
285 !
286 integer,intent(in) :: nn
287 real(4),intent(inout) :: gosa
288 integer :: i,j,k,loop
289 real(4) :: s0,ss
```

同様に、ループ、ラインについても分析ビューの項目を選択することで、その該当箇所がソースビューに表示されます。以下はライン単位のコスト情報の場合です。

分析ビュー				
トレース	変数有効域	プロ/タイ	エラー箇所	コスト情報：ループ
DProf_21997_000000_61030.0000_0.0000_610.0000_0.0000_1342097.0000_40.0000_0.0000_pri				
サンプリング数	全体に占める割合(%)	ライン	ファイル名	行番号
432	71.40	jacobi_	himenobmbxp.f90	300:300
76	12.56	jacobi_	himenobmbxp.f90	323:323
48	7.93	jacobi_	himenobmbxp.f90	313:313
26	4.30	jacobi_	himenobmbxp.f90	316:316
8	1.32	jacobi_	himenobmbxp.f90	314:314
6	0.99	jacobi_	himenobmbxp.f90	317:317
4	0.66	jacobi_	himenobmbxp.f90	301:301
4	0.66	jacobi_	himenobmbxp.f90	315:315
1	0.17	initmt_	himenobmbxp.f90	246:246



```
ソースビュー
himenobmbxp.f90 ✕
298 do i=2,imax-1
299 s0=a(l,j,k,1)*p(l+1,j,k) &
300 +a(l,j,k,2)*p(l,j+1,k) &
301 +a(l,j,k,3)*p(l,j,k+1) &
302 +b(l,j,k,1)*(p(l+1,j+1,k)-p(l-1,j-1,k)) &
303 -p(l-1,j+1,k)+p(l-1,j-1,k)) &
304 +b(l,j,k,2)*(p(l,j+1,k+1)-p(l,j-1,k+1)) &
305 -p(l,j+1,k-1)+p(l,j-1,k-1)) &
306 +b(l,j,k,3)*(p(l+1,j,k+1)-p(l-1,j,k+1)) &
307 -p(l+1,j,k-1)+p(l-1,j,k-1)) &
308 +c(l,j,k,1)*p(l-1,j,k) &
309 +c(l,j,k,2)*p(l-1,j,k) &
310 +c(l,j,k,3)*p(l,j,k-1)+wrk1(l,j,k)
311 ss=(s0*a(l,j,k,4)-p(l,j,k))*bnd(l,j,k)
312 GOSA=GOSA+SS*SS
313 wrk2(l,j,k)=p(l,j,k)+OMEGA*SS
314 enddo
315 enddo
316 enddo
```

基本プロファイラとの連携(3/3)

コスト情報をソースコードと連動して表示させます。

メニューバーから「プロファイラ」>「コストバー表示」、または「プロファイラ」>「コストルーラ表示」のチェックボックスを選択することで、コスト情報の表示領域をソースビュー右端に追加します。

The screenshot shows the profiler interface with the following components:

- Source Code View:** Lines 298-317 of a Fortran program. Line 300 is highlighted in red.
- Cost Bar View:** A vertical bar on the right side of the source code, showing cost percentages for each line. Line 300 has a green bar representing 71.40%.
- Cost Rule View:** A vertical bar on the right side of the cost bar, showing cost percentages for each rule. Line 300 has a red bar representing 7.93%.
- Table View:** A table at the bottom showing the cost information for each line.

サンプリング数	全体に占める割合(%)	ライン	ファイル名	行番号
432	71.40	jacobi_	himenobmbxp.f90	300:300
76	12.56	jacobi_	himenobmbxp.f90	323:323
48	7.93	jacobi_	himenobmbxp.f90	313:313
26	4.30	jacobi_	himenobmbxp.f90	316:316
8	1.32	jacobi_	himenobmbxp.f90	314:314
6	0.99	jacobi_	himenobmbxp.f90	317:317
4	0.66	jacobi_	himenobmbxp.f90	301:301
4	0.66	jacobi_	himenobmbxp.f90	315:315
1	0.17	initmt_	himenobmbxp.f90	246:246

コストバー表示
ソースビューと連動して、コスト情報をバーと数値で可視化します。

コストルーラ表示
ソースコード全体のコスト情報を俯瞰的に表示します。ルーラ上で選択した部分はソースビューと連動します。

この例ではソースコードの後半にコストの高い部分が集中していることがわかります。

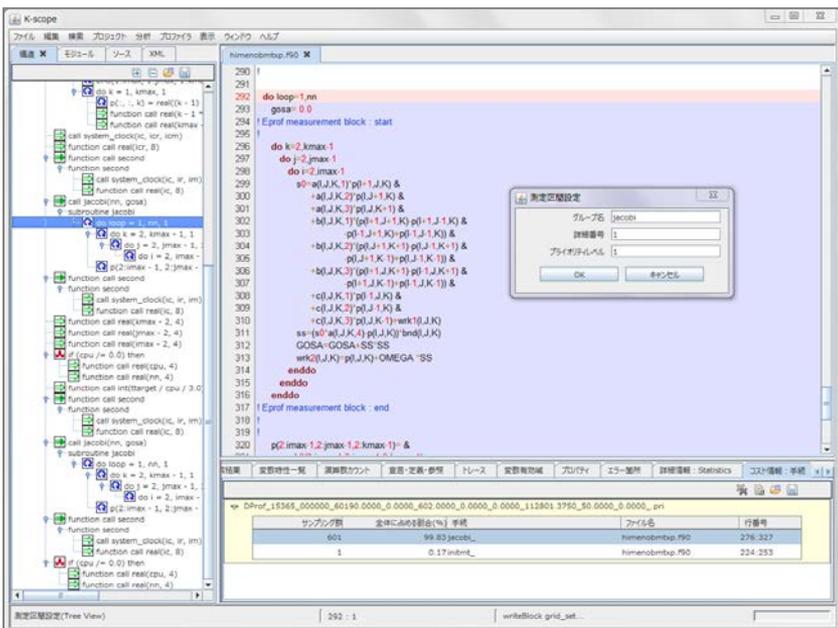
詳細プロファイラ測定区間の設定(1/3)

「京」の場合、基本プロファイラによって明らかになったボトルネック箇所をより詳細に調べる手段として、詳細プロファイラが提供されています。K-scopeでは、構造ビューに表示されているブロックに対して、「京」の詳細プロファイラの測定関数 (fapp_start/fapp_stop) を追加することができます。

測定区間の設定が可能な選択範囲の条件は以下の通りです。

1. 複数のDO文、IF文、計算式に設定できます
2. サブルーチン、関数宣言文には設定できません。
3. 選択ブロックは同一階層であること。

メニューバーから「プロファイラ」>「測定区間設定」で以下のダイアログが表示されます。ダイアログの意味は以下のテーブルの通りです。



グループ名	グループ名を設定します。 入力されたグループ名には自動的にダブルクォート(")を追加して囲みます。
詳細番号	詳細番号を設定します。
プライオリティレベル	プライオリティレベルを設定します。
OK	測定区間を設定します。
キャンセル	測定区間の設定をキャンセルします。

設定が完了すると分析ビューの「測定区間」に以下のような項目が追加されます。

分析ビュー			
数有効域	プロパティ	エラー箇所	詳細情報: Statistics
			コスト情報: 手続
			コスト情報: ループ
			コスト情報: ライン
			コールグラフ情報
			測定区間
測定区間設定	ファイル名	行番号	
jacobi, 1, 1	himenobmtxp.f90	296:316	

詳細プロファイラ測定区間の設定(2/3)

メニューバーから「プロファイラ」>「測定区間:上書き保存」で、測定区間に対して、詳細プロファイラの測定関数が追加されたファイルが出力（上書き）されます。

なお、「測定区間：フォルダ保存」を選択することで別フォルダに出力されます。この場合は、元のファイルは保持されます。

姫野ベンチマークのjacobiサブルーチンのステンシル計算に対して追加した場合は以下のようになります。

```
! Eprof measurement block : start
call fapp_start("jacobi", 1, 1)
!
do k=2,kmax-1
  do j=2,jmax-1
    do i=2,imax-1
      s0=a(I,J,K,1)*p(I+1,J,K) &
        +a(I,J,K,2)*p(I,J+1,K) &
        +a(I,J,K,3)*p(I,J,K+1) &
        +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K) &
          -p(I-1,J+1,K)+p(I-1,J-1,K)) &
        +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1) &
          -p(I,J+1,K-1)+p(I,J-1,K-1)) &
        +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1) &
          -p(I+1,J,K-1)+p(I-1,J,K-1)) &
        +c(I,J,K,1)*p(I-1,J,K) &
        +c(I,J,K,2)*p(I,J-1,K) &
        +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
      ss=(s0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
      GOSA=GOSA+SS*SS
      wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
    enddo
  enddo
enddo
! Eprof measurement block : end
call fapp_stop("jacobi", 1, 1)
```

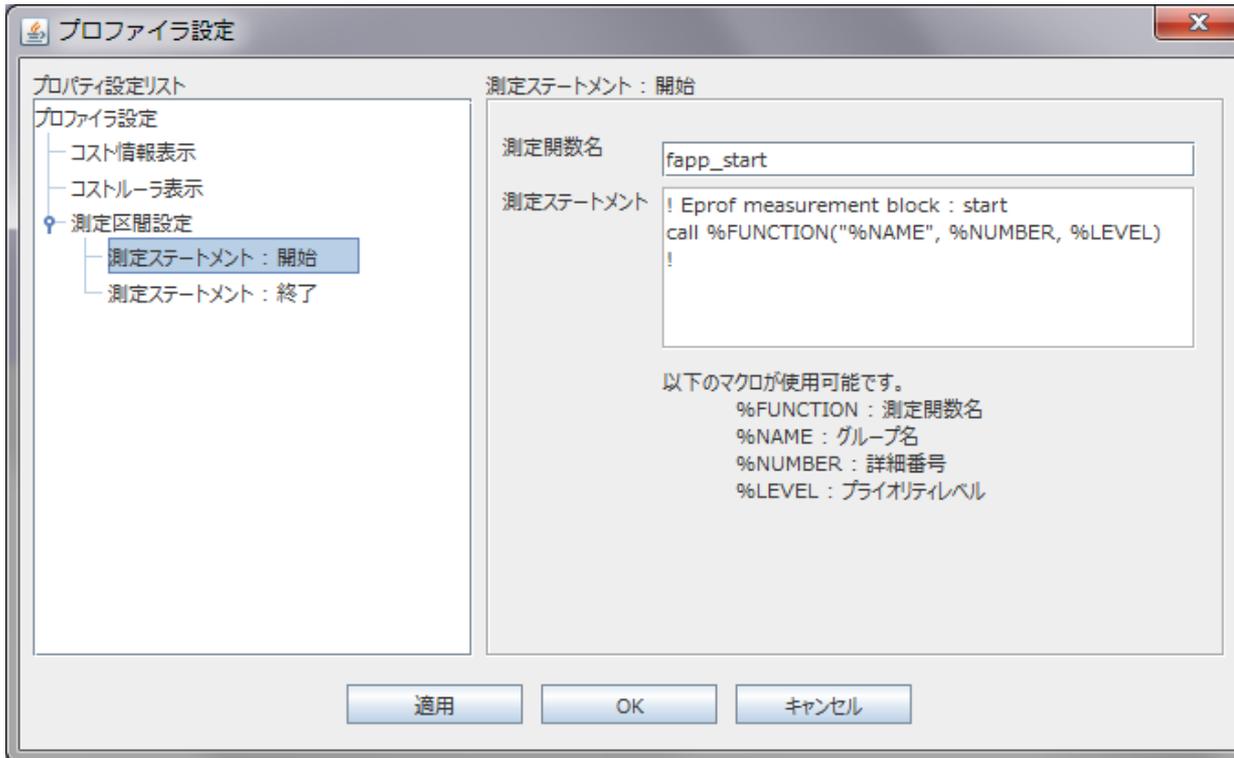
測定関数挿入後は、ソースコードと中間コードの行の対応がずれるため、中間コードの再生成が必要になります。

詳細プロファイラ測定区間の設定(2/3)

追加する測定関数の設定を変更することができます。

「プロジェクト」>「設定」>「プロファイラ」を選択すると以下のようなダイアログが表示されます。

その中の「測定区間設定」で、関数名および引数が変更できます。

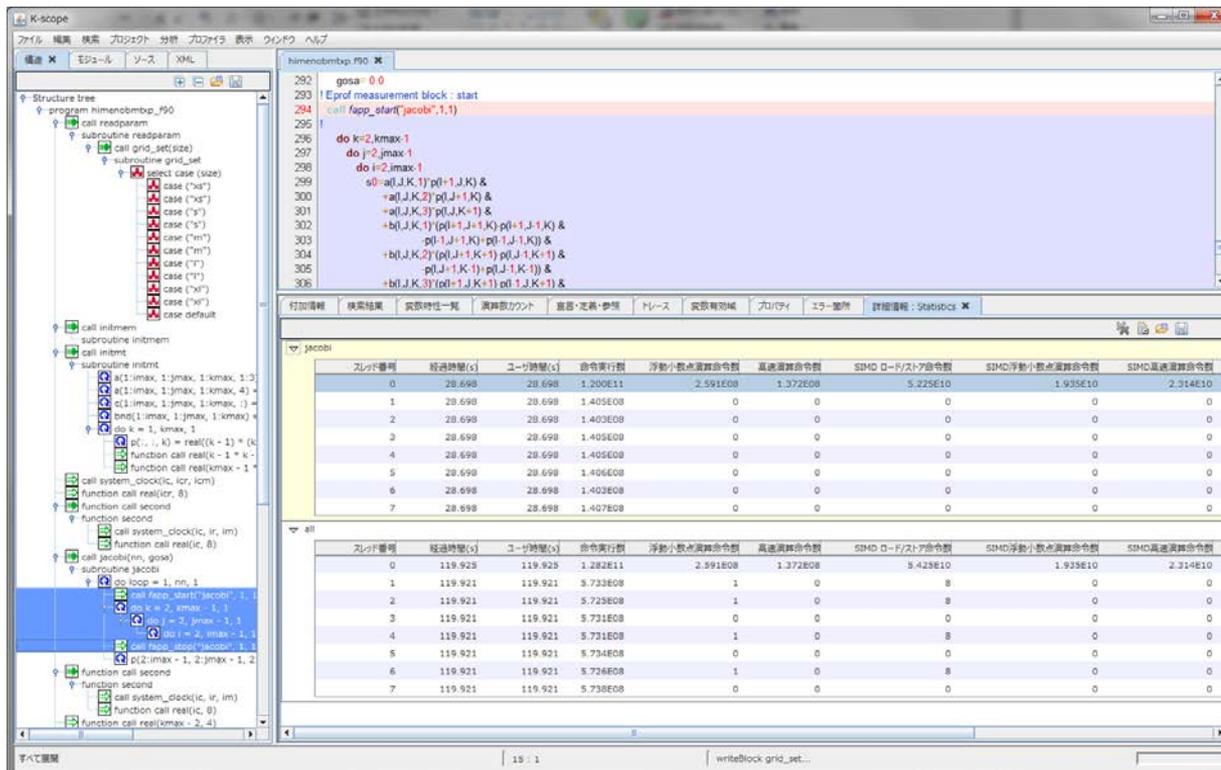


詳細プロファイラとの連携

基本プロファイラとの連携と同様に、メニューバーから「プロファイラ」>「プロファイラの読み込み」を選択し、Eprofで始まる、詳細プロファイラの出力結果を読み込みます。

分析ビューの項目をダブルクリックすることで、エクスプローラビューの「構造」タブとソースビューに、該当箇所が連動して表示されます。

以下は、前スライドで追加した詳細プロファイラの測定結果を読み込んだ例となります。



The screenshot displays the K-scope software interface. On the left, a 'Structure tree' shows a hierarchy of code blocks, with 'jacobi' selected. The top pane shows the source code for the 'jacobi' function, including a loop over 'k' and 'j'. The bottom pane shows two performance tables. The first table is for 'jacobi' and the second is for 'all'. Both tables have columns for 'スレッド番号' (Thread ID), '経過時間(s)' (Elapsed Time), 'ユーザ時間(s)' (User Time), '命令実行数' (Instruction Count), '浮動小数点演算命令数' (Floating Point Instruction Count), '高速演算命令数' (High-Speed Instruction Count), 'SIMDロードストア命令数' (SIMD Load/Store Instruction Count), 'SIMD浮動小数点演算命令数' (SIMD Floating Point Instruction Count), and 'SIMD高速演算命令数' (SIMD High-Speed Instruction Count).

スレッド番号	経過時間(s)	ユーザ時間(s)	命令実行数	浮動小数点演算命令数	高速演算命令数	SIMDロードストア命令数	SIMD浮動小数点演算命令数	SIMD高速演算命令数
0	28.698	28.698	1.200E11	2.591E08	1.372E08	5.225E10	1.935E10	2.314E10
1	28.698	28.698	1.405E08	0	0	0	0	0
2	28.698	28.698	1.403E08	0	0	0	0	0
3	28.698	28.698	1.405E08	0	0	0	0	0
4	28.698	28.698	1.405E08	0	0	0	0	0
5	28.698	28.698	1.406E08	0	0	0	0	0
6	28.698	28.698	1.403E08	0	0	0	0	0
7	28.698	28.698	1.407E08	0	0	0	0	0

スレッド番号	経過時間(s)	ユーザ時間(s)	命令実行数	浮動小数点演算命令数	高速演算命令数	SIMDロードストア命令数	SIMD浮動小数点演算命令数	SIMD高速演算命令数
0	119.925	119.925	1.282E11	2.591E08	1.372E08	5.428E10	1.935E10	2.314E10
1	119.921	119.921	5.733E08	1	0	8	0	0
2	119.921	119.921	5.723E08	1	0	8	0	0
3	119.921	119.921	5.731E08	0	0	0	0	0
4	119.921	119.921	5.731E08	1	0	8	0	0
5	119.921	119.921	5.734E08	0	0	0	0	0
6	119.921	119.921	5.726E08	1	0	8	0	0
7	119.921	119.921	5.738E08	0	0	0	0	0

詳細プロファイラに限らず、プロファイラ連携機能で表示される各項目の意味や制限は、富士通製プロファイラの仕様に従います。詳細はマニュアルを参照ください。

8. 分析機能

ソースコードから変数の特性や有効域、さらには演算数やメモリアクセスを計量し、そのコードの特性を分析します。なお、本機能はプロトタイプとして実装されているものが含まれます。また機械的に定量化しているものがあります。実際の評価に用いる際には十分な検討が必要と考えます。

分析ビュー(1/2)

分析結果は、主に分析ビューを通して表示されます。
なお分析ビューに表示されるタブは以下のようなものがあります。

付加情報	シミュレーション・コードのブロックに関連付けた情報（テキスト、ファイル名等）を表示します。
検索結果	ソースコード、エクスプローラビューの検索結果を表示します。
変数特性一覧	変数の配列、属性等の定義情報を表示します。
演算カウント	サブルーチン、関数、DOループブロック内の演算数、ロード・ストア数を表示します。
要求B/F	要求Byte/FLOPの算出結果を表示します。
宣言・定義・参照	変数の宣言及び変数に対して代入、参照を行っている箇所を表示します。
トレース	変数の代入、参照箇所をサブルーチン・関数内で上下に探索を行います。 また、CALL文実引数、仮引数による別のサブルーチンへの探索移動も可能です。
変数有効域	変数のスコープによる有効モジュール、サブルーチン・関数を表示します。
構造情報（差し替え結果）	構造情報の差替、構造解析再実行における付加情報の差替結果を表示します。
プロパティ	エクスプローラビューに表示されたツリーのノードのプロパティの表示を行います。
エラー箇所	構造解析、分析によって発生したエラーメッセージを表示します。
コンソール (デフォルト：非表示)	ツール内部の詳細メッセージをコンソールに出力します。

分析ビュー(2/2)

プロファイラ情報：コスト情報：手続 (デフォルト：非表示)	DProf基本プロファイラ情報のコスト情報：手続のサンプリング数を表示します。
プロファイラ情報：コスト情報：ループ (デフォルト：非表示)	DProf基本プロファイラ情報のコスト情報：ループのサンプリング数を表示します。
プロファイラ情報：コスト情報：ライン (デフォルト：非表示)	DProf基本プロファイラ情報のコスト情報：ラインのサンプリング数を表示します。
プロファイラ情報：コールグラフ情報 (デフォルト：非表示)	DProf基本プロファイラ情報のコールグラフ情報を表示します。
プロファイラ情報：詳細情報：Cache (デフォルト：非表示)	EProf詳細プロファイラ情報のCache情報を表示します。
プロファイラ情報：詳細情報：Instructions (デフォルト：非表示)	EProf詳細プロファイラ情報のInstructions情報を表示します。
プロファイラ情報：詳細情報：MEM_access (デフォルト：非表示)	EProf詳細プロファイラ情報のMEM_access情報を表示します。
プロファイラ情報：詳細情報：Performance (デフォルト：非表示)	EProf詳細プロファイラ情報のPerformance情報を表示します。
プロファイラ情報：詳細情報：Stactitics (デフォルト：非表示)	EProf詳細プロファイラ情報のStactitics情報を表示します。
プロファイラ情報：測定区間 (デフォルト：非表示)	設定した詳細プロファイラ測定区間の一覧を表示します。

変数特性一覧

以後スライドでは、代表的な分析機能について説明します。

エクスプローラビューの「構造」タブのツリーを選択し、メニューバーから「分析」>「変数特性一覧」を選択することで、該当ブロック中で使用されている変数の特性一覧を表示することができます。

以下は姫野ベンチマークのステンシル計算の部分为例としてピックアップしています。

エクスプローラビュー

```

subroutine jacobi
  do loop = 1, nn, 1
    do k = 2, kmax - 1, 1
      do j = 2, jmax - 1, 1
        do i = 2, imax - 1, 1
          p(2:imax - 1, 2:jmax - 1, 2:kmax - 1) = wrk2(2:imax - 1, 2:jmax - 1, 2:kmax - 1)
        
```

ソースビュー

```

296 do k=2,kmax-1
297 do j=2,jmax-1
298 do i=2,imax-1
299 s0=a(l,j,k,1)*p(l+1,j,k) &
300 +a(l,j,k,2)*p(l,j+1,k) &
301 +a(l,j,k,3)*p(l,j,k+1) &
302 +b(l,j,k,1)*(p(l+1,j+1,k)-p(l+1,j-1,k) &
303 -p(l-1,j+1,k)+p(l-1,j-1,k)) &
304 +b(l,j,k,2)*(p(l,j+1,k+1)-p(l,j-1,k+1) &
305 -p(l,j+1,k-1)+p(l,j-1,k-1)) &
306 +b(l,j,k,3)*(p(l+1,j,k+1)-p(l-1,j,k+1) &
307 -p(l+1,j,k-1)+p(l-1,j,k-1)) &
308 +c(l,j,k,1)*p(l-1,j,k) &
309 +c(l,j,k,2)*p(l,j-1,k) &
310 +c(l,j,k,3)*p(l,j,k-1)+wrk1(l,j,k)
311 ss=(s0*a(l,j,k,4)-p(l,j,k))*bnd(l,j,k)
312 GOSA=GOSA+SS*SS
313 wrk2(l,j,k)=p(l,j,k)+OMEGA *SS
314 enddo
315 enddo
316 enddo

```

分析ビュー

type	name	data type	access specifier	parameter	init value	size	intent	optional	pointer/tar...	save	common	allocatable
scalar	nn	integer	default	no param	no value	1	in	no opt	no pointer	no save	no common	no alloc
scalar	gosa	real(4)	default	no param	no value	1	inout	no opt	no pointer	no save	no common	no alloc
scalar	i	integer	default	no param	no value	1	no intent	no opt	no pointer	no save	no common	no alloc
scalar	j	integer	default	no param	no value	1	no intent	no opt	no pointer	no save	no common	no alloc
scalar	k	integer	default	no param	no value	1	no intent	no opt	no pointer	no save	no common	no alloc
scalar	loop	integer	default	no param	no value	1	no intent	no opt	no pointer	no save	no common	no alloc
scalar	s0	real(4)	default	no param	no value	1	no intent	no opt	no pointer	no save	no common	no alloc
scalar	ss	real(4)	default	no param	no value	1	no intent	no opt	no pointer	no save	no common	no alloc

宣言・定義・参照

ソースビューの変数を選択し、「宣言・定義・参照」を選択することで分析ビューに変数の宣言されている箇所、定義（代入）されている箇所、参照されている箇所が表示されます。分析ビューの項目を選択することで、ソースコードの該当箇所がソースビューに表示されます。

以下は、姫野ベンチマークのステンシル部分の配列変数aについての分析結果を表示したものです。

```
ソースビュー
296 do k=2,kmax-1
297 do j=2,jmax-1
298 do i=2,imax-1
299 s0=a(l,J,K,1)*p(l+1,J,K) &
300 +a(l,J,K,2)*p(l,J+1,K) &
301 +a(l,J,K,3)*p(l,J,K+1) &
302 +b(l,J,K,1)*(p(l+1,J+1,K)-p(l+1,J-1,K) &
303 -p(l-1,J+1,K)+p(l-1,J-1,K)) &
304 +b(l,J,K,2)*(p(l,J+1,K+1)-p(l,J-1,K+1) &
305 -p(l,J+1,K-1)+p(l,J-1,K-1)) &
306 +b(l,J,K,3)*(p(l+1,J,K+1)-p(l-1,J,K+1) &
307 -p(l+1,J,K-1)+p(l-1,J,K-1)) &
308 +c(l,J,K,1)*p(l-1,J,K) &
309 +c(l,J,K,2)*p(l,J-1,K) &
310 +c(l,J,K,3)*p(l,J,K-1)+wrk1(l,J,K)
311 ss=(s0*a(l,J,K,4)-p(l,J,K))*bnd(l,J,K)
312 GOSA=GOSA+SS*SS
313 wrk2(l,J,K)=p(l,J,K)+OMEGA *SS
314 enddo
315 enddo
316 enddo
```

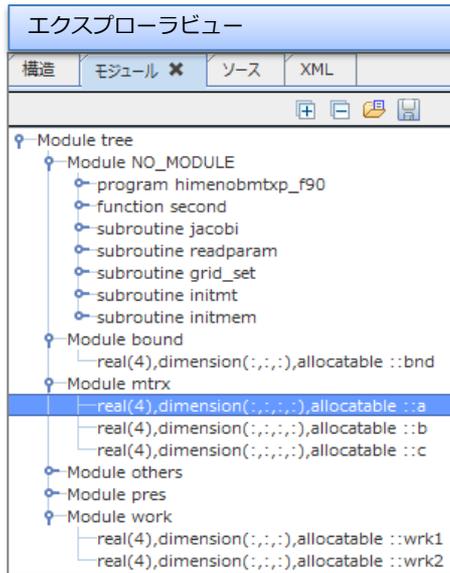


```
分析ビュー
付加情報 検索結果 変数特性一覧 演算数カウント 宣言・定義・参照 * トレース 変数有効域 プロパティ エラー箇所
real(4),dimension(:,:,:,:),allocatable ::a
宣言
  Module mtx
    real(4),dimension(:,:,:,:),allocatable ::a
参照
  subroutine jacobi
    s0 = (((((((((a(i, j, k, 1) * p(i + 1, j, k)) + (a(i, j, k, 2) * p(i, j + 1, k)))) + (a(i, j, k, 3) * p(i, j, k + 1)))) + (b(i, j, k, 1) * (p(l+1,J+1,K)-p(l+1,J-1,K) - p(l-1,J+1,K)+p(l-1,J-1,K)) + b(i, j, k, 2) * (p(l,J+1,K+1)-p(l,J-1,K+1) - p(l,J+1,K-1)+p(l,J-1,K-1)) + b(i, j, k, 3) * (p(l+1,J,K+1)-p(l-1,J,K+1) - p(l+1,J,K-1)+p(l-1,J,K-1)) + c(l, j, k, 1) * p(l-1, j, k) + c(l, j, k, 2) * p(l, j-1, k) + c(l, j, k, 3) * p(l, j, k-1) + wrk1(l, j, k)) - p(l, j, k) * bnd(l, j, k))
定義
  subroutine initmt
    a = 0.0
    a(1:imax, 1:jmax, 1:kmax, 1:3) = 1.0
    a(1:imax, 1:jmax, 1:kmax, 4) = 1.0 / 6.0
```

上記の分析から、配列変数aは、モジュールmtx中で、 allocatable属性が付加された単精度浮動小数点型の4次元配列として宣言されていることが分かります。また、サブルーチンjacobi の中で2つの参照箇所あることが分かります。さらに、代入はサブルーチンinitmt の中で3箇所あることが分かります。

変数有効域

エクスプローラビューの「モジュール」タブで変数を選択し、「変数有効域」を選択することで、変数の有効域（スコープ）を表示します。



配列変数aの有効域は、mtrxモジュール内、および サブルーチン initmem, initmt, jacobi内で有効であることがわかります。なお、親プログラム(Program文、Module文)文を持たないプロシージャは、NO_MODULE で表されます。

トレース機能

ソースビューで選択されている変数に対してトレース機能を提供します。
 以下は姫野ベンチマークのprogram文中で定義されている変数`nn`についてトレースしたものです。

結果からは変数`nn`は、サブルーチン`jacobi`の第一引数として渡されていることがわかります。

```

ソースビュー
himenobmtxp.f90
65 !
66 !
67 program HimenoBMTxp_F90
68 use others
69 !
70 implicit none
71 !
72 integer :: nn
73 integer :: ic,icr,icm
74 real(4) :: flop,xmflops2,score,gosa
75 real(8) :: cpu0,cpu1,cpu,dt
76 ! ttarget specifies the measuring period in sec
77 real(4),parameter :: ttarget=60.0
78 real(8),external :: second
  
```



```

分析ビュー
トレース ( nn:himenobmtxp_f90 )
himenobmtxp_f90->nn
program himenobmtxp_f90
  nn = 3
  call jacobi(nn, gosa)
  if (cpu /= 0.0) then
    xmflops2 = ((flop / real(cpu, 4)) * 1.0e-6) * real(nn, 4)
    function call real(nn, 4)
    nn = int(ttarget / (cpu / 3.0))
    call jacobi(nn, gosa)
  if (cpu /= 0.0) then
    xmflops2 = ((flop * 1.0e-6) / real(cpu, 4)) * real(nn, 4)
    function call real(nn, 4)
  
```

子プロシージャに移動する場合は、タブ右上の「イン」アイコンによりトレースを継続することができます。

機能としては、同じ階層でトレースするアップ・ダウン、プロシージャ呼び出しの親子の階層をトレースするイン・アウトがあります。概要は以下のテーブルの通りです。

	更新	トレース結果の表示の再描画を行います。
	アップ	トレース結果から選択箇所の上のトレース箇所を表示します。
	ダウン	トレース結果から選択箇所の下のトレース箇所を表示します。
	イン	トレース結果から選択箇所のCALL文からサブルーチン内にトレースを移動します。トレース先が複数存在する場合は、選択ダイアログが表示されます。
	アウト	トレース結果から選択箇所のサブルーチンから呼び出し元のCALL文にトレースを移動します。トレース先が複数存在する場合は、以下のトレース先選択ダイアログが表示されます。
	フォワード	
	トレース箇所を開く	選択トレース先の該当行をソースビューにアクティブにします。
	エクスポート	トレース結果をテキスト出力します。

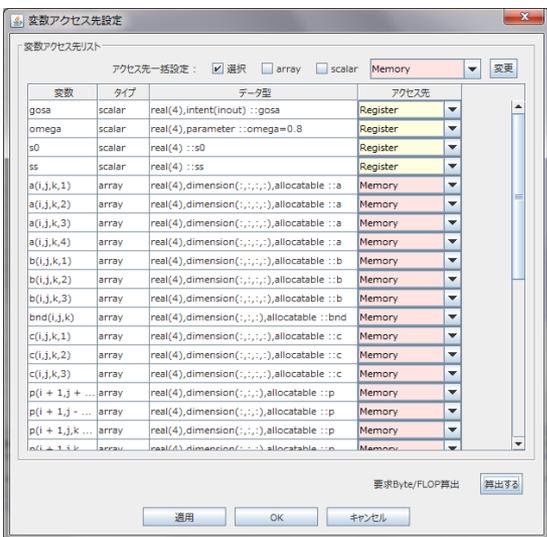
変数アクセス先設定

チューニングを行う際に、プログラムのメモリアクセス特性をソースコード上から算出する場合があります。

一般的な階層型メモリアーキテクチャを採用した環境で、メモリへのアクセスが支配的な場合は、キャッシュ等の影響を全く無視しても、特性の把握という観点ではあまり問題にはなりません。その一方で、キャッシュアクセスが無視できないコードや、より正確な見積もりが必要な場合は、個々の浮動小数点実数型の変数に対して、メモリ階層の候補（メモリ、Level-1データキャッシュ、Level-2キャッシュ、レジスタ、その他ユーザ定義等）から一つのアクセス先を設定し、算出できることが望ましいと考えます。

K-scopeでは、Advancedなモードとして、そのような煩雑になりがちな変数のアクセス先に応じた見積もりを支援する機能を提供します。

エクスプローラビューの構造タブで算出するノードを選択した後に、メニューバーから「分析」>「変数アクセス先設定」を選択すると、以下の様なダイアログが表示されます。ここで、各変数（スカラー変数、配列変数）に対して、どの階層までメモリアクセスが到達するか設定することが可能です。



```

ソースビュー
296 do k=2,kmax-1
297 do j=2,jmax-1
298 do i=2,imax-1
299 s0=a(I,J,K,1)*p(I+1,J,K) &
300 +a(I,J,K,2)*p(I,J+1,K) &
301 +a(I,J,K,3)*p(I,J,K+1) &
302 +b(I,J,K,1)*(p(I+1,J+1,K)-p(I+1,J-1,K)) &
303 -p(I-1,J+1,K)-p(I-1,J-1,K)) &
304 +b(I,J,K,2)*(p(I,J+1,K+1)-p(I,J-1,K+1)) &
305 -p(I,J+1,K-1)-p(I,J-1,K-1)) &
306 +b(I,J,K,3)*(p(I+1,J,K+1)-p(I-1,J,K+1)) &
307 -p(I+1,J,K-1)-p(I-1,J,K-1)) &
308 +c(I,J,K,1)*p(I-1,J,K) &
309 +c(I,J,K,2)*p(I,J-1,K) &
310 +c(I,J,K,3)*p(I,J,K-1)+wrk1(I,J,K)
311 ss=(s0*a(I,J,K,4)+p(I,J,K)) *bnd(I,J,K)
312 GOSA=GOSA+SS*SS
313 wrk2(I,J,K)=p(I,J,K)+OMEGA*SS
314 enddo
315 enddo
316 enddo
    
```

デフォルトでは配列変数はメモリに、スカラー変数はレジスタにアクセスすると仮定しています。ただし、最終的に各変数がどのメモリ階層にアクセスするかはユーザ自身が検討し、設定する必要があります。

変数	選択ブロック、行の変数名を表示します。
タイプ	array, scalar種別を表示します。
データ型	データ型 (real, complex)、配列定義を表示します。
アクセス先	アクセス先を以下から選択、表示します。 <ul style="list-style-type: none"> Memory L1 Cache L2 Cache Register Custom Default Mem,L1,L2,Reg (複数設定済み) 選択メモリの背景色は「要求Byte/FLOP設定ダイアログ」にてメモリ毎に設定した背景色にて表示します。
すべて：選択	リスト中の選択行を同一アクセス先に設定します。
すべて：array	リスト中のarray変数を同一アクセス先に設定します。
すべて：scalar	リスト中のscalar変数を同一アクセス先に設定します。
すべて：コンボボックス	リスト中の設定アクセス先を選択します。
すべて：変更	リスト中のarray, scalar変数を同一アクセス先に変更します。
算出する	要求Byte/FLOPを算出結果を表示する為に要求Byte/FLOP算出ダイアログを表示します。要求Byte/FLOP算出ダイアログからの表示の場合は、非表示となります。
適用	設定アクセス先を変数に設定します。
OK (再計算)	設定アクセス先を変数に設定し、画面を閉じます。要求Byte/FLOP算出ダイアログからの表示の場合は、ボタンテキストは"再計算"となり、要求Byte/FLOP算出ダイアログを再表示します。
キャンセル	設定アクセス先を変数に設定せずに画面を閉じます。

要求Byte/FLOP算出機能(1/2)

K-scopeでは、Advancedなモードとして、メモリアクセス特性に加えて、ソースコードから浮動小数点演算数をカウントし、要求B/Fを算出する機能を提供します。なお、評価値は機械的に算出しますので、その使用については十分な検討を行なって下さい。

エクスプローラビューの構造タブで算出するノードを選択した後に、メニューバーから「分析」>「要求Byte/FLOP算出」を選択すると以下の様なダイアログが表示されます。また、分析ビューにもログとして結果が表示されます。

例えば、姫野ベンチマークのjacobiサブルーチン（下記部分）について、算出すると以下のようなダイアログが表れます。

```
ソースビュー
296 do k=2,kmax-1
297 do j=2,jmax-1
298 do i=2,imax-1
299 s0=a(l,J,K,1)*p(l+1,J,K) &
300 +a(l,J,K,2)*p(l,J+1,K) &
301 +a(l,J,K,3)*p(l,J,K+1) &
302 +b(l,J,K,1)*(p(l+1,J+1,K)-p(l+1,J-1,K) &
303 -p(l-1,J+1,K)+p(l-1,J-1,K)) &
304 +b(l,J,K,2)*(p(l,J+1,K+1)-p(l,J-1,K+1) &
305 -p(l,J+1,K-1)+p(l,J-1,K-1)) &
306 +b(l,J,K,3)*(p(l+1,J,K+1)-p(l-1,J,K+1) &
307 -p(l+1,J,K-1)+p(l-1,J,K-1)) &
308 +c(l,J,K,1)*p(l-1,J,K) &
309 +c(l,J,K,2)*p(l,J-1,K) &
310 +c(l,J,K,3)*p(l,J,K-1)+wrk1(l,J,K)
311 ss=(s0*a(l,J,K,4)-p(l,J,K))*bnd(l,J,K)
312 GOSA=GOSA+SS*SS
313 wrk2(l,J,K)=p(l,J,K)+OMEGA*SS
314 enddo
315 enddo
316 enddo
```



算出範囲 do k = 2, kmax - 1, 1		
メモリ性能算出結果		
Load	144	(Byte)
Store	16	(Byte)
演算数	34	(FLOP)
要求B/F	4.71	(Byte/FLOP)
スループット	179.40	(GB/s)
実効B/F	1.40	(Byte/FLOP)
ピーク性能比	29.78	(%)

最内ループに含まれる、配列変数について添字が異なるものは別ストリームとして機械的にカウントしています。この例では、スカラー変数についてもデフォルトでは算出対象になっています。なお、算出するものについては、「スループット設定」で変更することができます。

上記最内ループ中に、ストアは4箇所あるので、 $4 \times 4 = 16$ [byte]となります。また、ロードは、配列aは4箇所、配列b,cはそれぞれ3箇所、配列pは19箇所、配列wrk1, wrk2, bndがそれぞれ1箇所、スカラー変数ss, s0, GOSA, OMEGAの計4箇所であるので、合計36箇所。 $36 \times 4 = 144$ [byte]となります。浮動小数点演算については34個の四則演算があり、デフォルトでは1としているため、最終的に要求B/Fは、 $(144+16)/34 = 4.7$ [byte/FLOP]と算出されます。

その下に出力されているスループット、実効B/F、ピーク性能比については、ここでは省略します。

要求Byte/FLOP算出機能(2/2)

Advancedなモードとして、四則演算および組込み関数の演算数を設定することができます。

「プロジェクト」>「設定」>「演算数カウント」を選択すると以下のダイアログが表示されます。

この中で四則演算および組込み関数の演算数を設定することができます。なお、これらの値は実行環境に依存していますので、デフォルトでは四則演算はすべて1、組込み関数は0としています。



組込み関数	op(+)	op(*)
abs	0	0
achar	0	0
acos	0	0
adjustl	0	0
aimag	0	0
aint	0	0
all	0	0
allocated	0	0
alog	0	0
alog10	0	0
amax0	0	0
amax1	0	0
amin0	0	0

補足

プロパティファイル

K-scope起動時に設定されるデフォルトの設定は、プロパティファイルに記述されています。デフォルト設定を変更する場合は、propertiesディレクトリ以下のproperties.xml（以後、プロパティファイル）を直接編集してください。

プロパティファイル中には、予約語等のハイライト設定、組込関数の演算カウント等の主に以下の設定が記述されています。

1. 表示設定
2. キーワード設定
3. 外部ツール設定
4. 組込関数演算カウント
5. プロファイラ設定
6. プロジェクト設定
7. 要求Byte/FLOP設定
8. アプリケーション設定

プロパティファイルの構造は以下のテーブルの通りです。

No	要素名	説明
1	properties	ルート要素
2	settings	プロパティ設定
3	property	表示設定
4	keyword	キーワード設定
5	program	外部ツール設定
6	operation_flop	四則演算FLOP数設定
7	operand	組込関数演算カウント
8	profiler	プロファイラ設定
9	project	プロジェクト設定
10	memoryband	要求Byte/FLOP設定
11	application	アプリケーション設定

各設定要素は属性によって設定します。以後、属性の設定項目の詳細説明をします。

プロパティファイル

表示設定：要素 = property

エクスプローラービューのツリーやソースビューの表示フォント、文字、範囲の背景色を設定します。

No	属性	属性値の説明
1	key	設定項目のキー名
2	name	フォント名
3	size	フォントサイズ
4	bold	フォントスタイル：太字
5	italic	フォントスタイル：イタリック
6	color	色の設定値 色名(red, white等)又は#XXXXXX (16進数6桁HTMLカラー表記)
7	value	数値の設定値

ソースビュー設定は、以下の設定項目 (=キー名) を持ちます。

No	設定項目(=key)	設定項目の説明
1	font-source	ソースビューのフォント
2	fontcolor-source	ソースビューの文字色
3	background-source	ソースビューの背景色
4	background-selectedrow	ソースビューの選択行背景色
5	wordwrap	ソースビューの折り返し文字数
6	background-area	ソースビューの有効域の背景色
7	background-block	ソースビューの選択ブロックの背景色
8	fontcolor-search	ソースビューの検索文字色
9	background-search	ソースビューの検索文字背景色
10	background-linenumber	行番号背景色
11	background-selectnode	エクスプローラービュー選択ノード背景色
12	background-view2	分析ビュー選択色
13	fontcolor-informationnode	付加情報ノード文字色
14	fontcolor-brokenlinknode	リンク切れ文字色

プロパティファイル

キーワード設定：要素=keyword

付加情報に記述のURL、ファイル名、及び表示する外部プログラムを設定します。

No	属性	属性値の説明
1	name	キーワード名
2	keyword	ハイライト文字、又は正規表現
3	forecolor	ハイライト文字色
4	bold	フォントスタイル：太字
5	italic	フォントスタイル：イタリック
6	regex	true=keywordを正規表現とする。
7	sensitivecase	true=大文字、小文字を区別する。
8	enabled	true=ハイライト表示を有効とする。

keywordにはハイライトを行う単語又は正規表現を記述できます。
正規表現を設定する場合は、以下の注意点があります。

(注意)

正規表現にグループ（括弧）を使用した場合、グループの最初の一致箇所がハイライト表示されます。正規表現にて一致した全体はハイライトの対象にはなりません。

(例)

```
keyword = "subroutine[ ]+([a-z0-9_-]+)[ ¥n¥("]
```

subroutine後の英数単語検索

ハイライト表示 = subroutine print_cal(a, b, c)

サブルーチン名がハイライト表示されます。

プロパティファイル

外部ツール設定：要素=program

エクスプローラービューのツリーやソースビューの表示フォント、文字、範囲の背景色を設定します。

No	属性	属性値の説明
1	name	外部ツール名
2	pattern	拡張子、又は正規表現
3	regex	true= patternを正規表現とする。
4	relation	true=OS依存の関連付けられたプログラムを使用する。
5	program	外部プログラムパスを設定する。 relation=trueの時は無効

四則演算のFLOP数を設定します。

No	属性	属性値の説明
1	add	加算FLOP数
2	mul	乗算FLOP数
3	sub	減算FLOP数
4	div	除算FLOP数
5	pow	べき乗FLOP数（未使用）

プロパティファイル

組込関数演算カウント：要素=operand

組込関数の演算カウントを設定します。

No	属性	属性値の説明
1	name	組込関数名
2	add	加算演算数
3	mul	乗算演算数
4	sub	減算演算数（未使用）
5	div	除算演算数（未使用）

プロパティファイル

プロファイラ：要素=profiler

プロファイラ情報の表示のパラメータ、色、測定区間を設定します。

No	属性	属性値の説明
1	key	設定項目のキー名
2	value	最大表示行数、測定関数名
3	color	コスト表示の棒グラフ色の設定値 色名(red, white等)又は#XXXXXX (16進数6桁HTMLカラー表記)
4	text()	測定ステートメントを設定します。

以下の設定項目 (=キー名) を持ちます。

No	設定項目(=key)	設定項目の説明
1	costinfo-maxcount	コスト表示テーブルの最大表示行数を設定します。 0の場合は、すべて表示します。
2	costinfo-barcolor-procedure	コスト情報:手続の棒グラフの表示色を設定します。
3	costinfo-barcolor-loop	コスト情報:ループの棒グラフの表示色を設定します。
4	costinfo-barcolor-line	コスト情報:ラインの棒グラフの表示色を設定します。
5	eprof-function-start	測定区間の開始関数名を設定します。
6	eprof-function-end	測定区間の終了関数名を設定します。
7	eprof-statement-start	測定区間の開始ステートメントを設定します。
8	eprof-statement-end	測定区間の終了ステートメントを設定します。

プロパティファイル

プロジェクト：要素=project

プロジェクト全体に関する情報を設定します。

No	属性	属性値の説明
1	key	設定項目のキー名
2	name	表示用の項目名
3	value	設定値
4	type	設定値の型 (text, file, folder, fixed-text)
5	message	説明文

以下の設定項目 (=キー名) を持ちます。

No	設定項目(=key)	設定項目の説明
1	make-command	makeコマンドを設定します。
2	makefile-path	Makefileを設定します。
3	project-title	プロジェクトタイトルはここに設定しても反映されません。

プロパティファイル

要求Byte/FLOP設定：要素=memoryband

要求Byte/FLOPに関する情報を設定します。

No	設定項目(=key)	設定項目の説明
1	operation-performance	演算性能を"value"属性に設定します。
2	throughput-store-mode	要求Byte/FLOP算出時に使用するスループットの選択方法を"value"属性に設定します。 value="auto" : Storeの有り、無しにより自動的にスループット値を判断します。 value="store" : Storeの有りのスループット値を使用します。 value="nonestore" : Storeの無しのスループット値を使用します。
3	unit-type	要求Byte/FLOPの算出単位を"value"属性に設定します。 value="byte_flop" : 算出単位=Byte/FLOP value="flop_byte" : 算出単位=FLOP/Byte
4	default-size	real, integerのデータ型のデフォルトサイズを設定します。 属性:real : データ型realのデフォルトサイズ 属性:integer : データ型integerのデフォルトサイズ
5	memory	アクセス先メモリ:Memoryを設定します。
6	l1_cache	アクセス先メモリ:L1 Cacheを設定します。
7	l2_cache	アクセス先メモリ:L2 Cacheを設定します。
8	register	アクセス先メモリ:Registerを設定します。
9	custom	アクセス先メモリ:Customを設定します。

アクセス先メモリは以下の属性により設定を行います。

No	属性	設定項目の説明
1	name	アクセス先メモリの名称
2	backcolor	ソースビュー表示背景色
3	throughput-store	スループット：ストア有り
4	throughput-nonestore	スループット：ストア無し
5	coef	係数
6	required	要求Byte算出対象："true"=算出対象
7	enabled	有効・無効（未使用）

プロパティファイル

アプリケーション：要素=application

アプリケーションに関する情報を設定します。

No	属性	属性値の説明
1	key	設定項目のキー名
2	value	設定値

以下の設定項目（=キー名）を持ちます。

No	設定項目(=key)	設定項目の説明
1	newproject_save	プロジェクト作成直後に保存を行うか否かのチェックボックスのデフォルト値を設定します。
2	exportsource_exclude	ソースファイルエクスポートダイアログの除外ファイルのデフォルト値を設定します。