XcalableMPの中間コードを利用した Fortranコード分析ツールの紹介

November 1, 2013

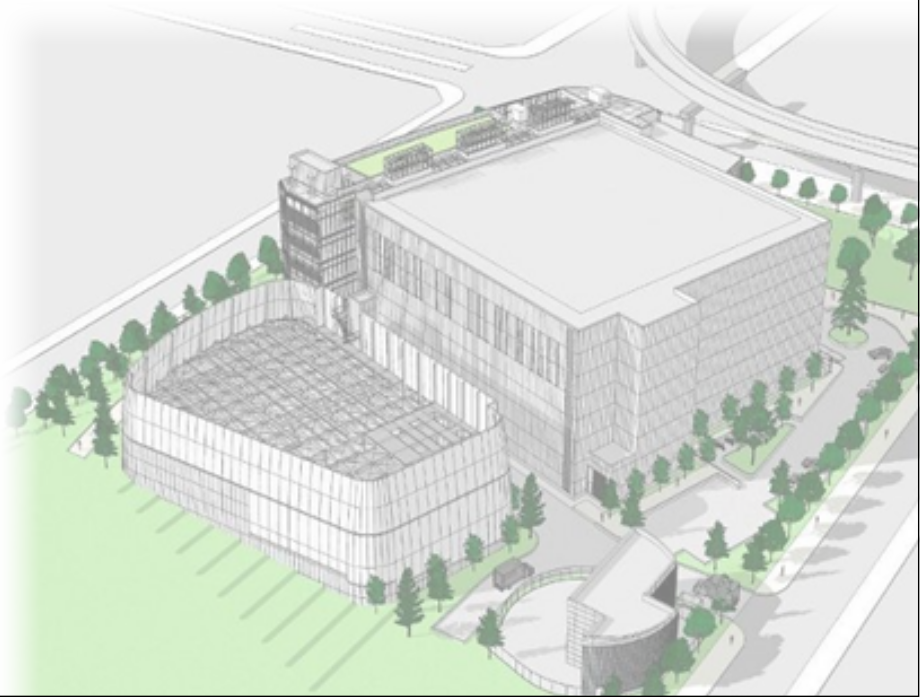
寺井 優晃
理化学研究所 計算科学研究機構

概要

- 背景
- コンセプトとデザイン
- 機能の紹介
  - インターフェイス
  - 静的プログラム解析
  - 動的プログラム解析
- 最近の開発状況

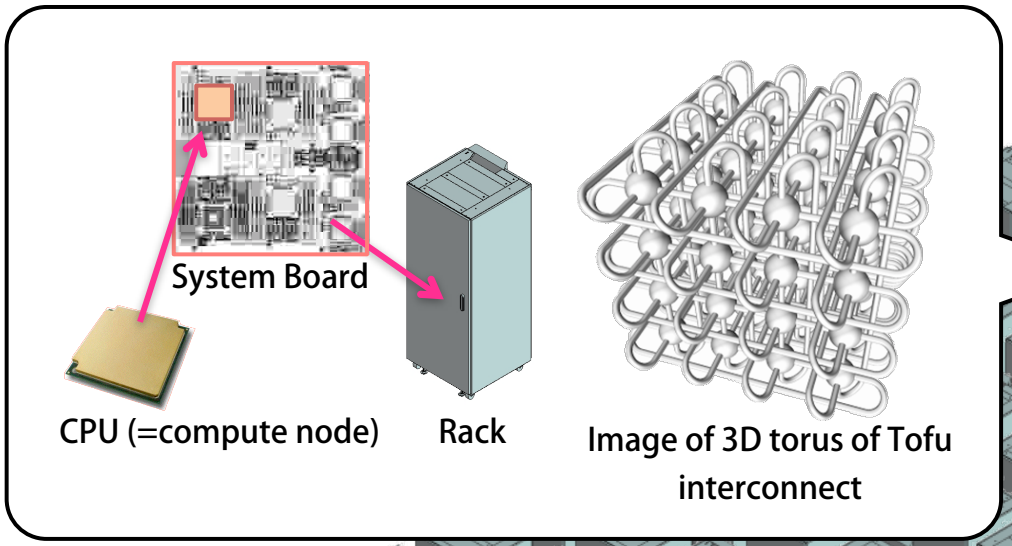
業務・活動内容

- 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
- 内容
  - アプリケーションの性能改善（システムの性能評価も一部含む）
  - 京の高度化に関するフィードバック（コンパイラ、プロファイラ、等）
  - 「京」高度化枠のユーザに対するサポート（運用の一部）
  - プライベートサーバの管理



アプリケーション性能改善

- 性能改善 (= チューニング)
  - 単体性能チューニング × 高並列性能チューニング



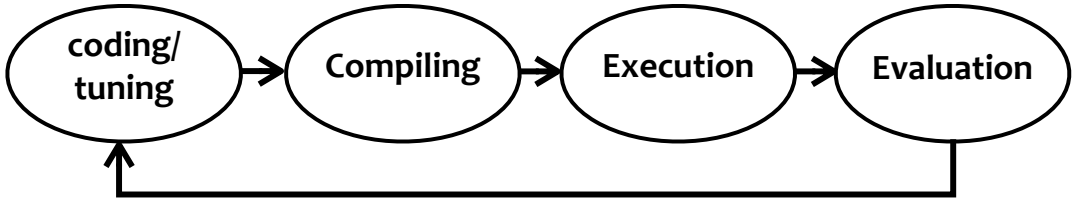
864 racks

(=82,944 compute nodes)

on the K computer

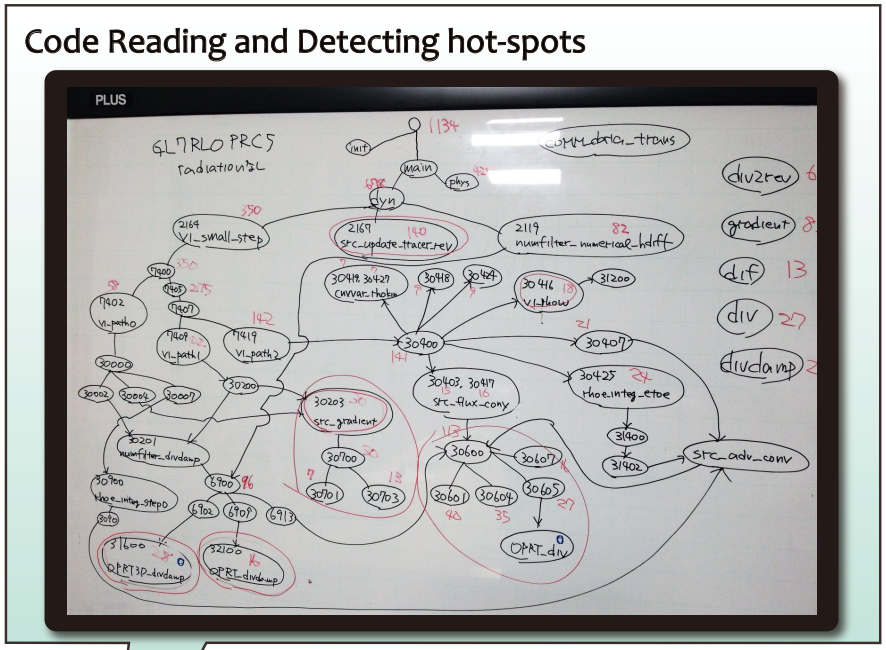
単体性能チューニング

- 本来はコンパイラが行うべき最適化を、期待する効果が得られるように手動によりソースコードを書き換えること。
- 試行錯誤



- 原因と結果に基づくシステマティックなアプローチ
- アプリケーション、コンパイラ、CPUなど、性能に影響を与える因子が測定に介在する。
- 非設計者にとっては、個々のパーツの大部分がブラックボックス（であることが多い ⇒ チューニングの難しさ）
- チューニングはシステマティックとヒューリスティックな作業の中間領域
- ちょっとしたパズル

ワークフロー



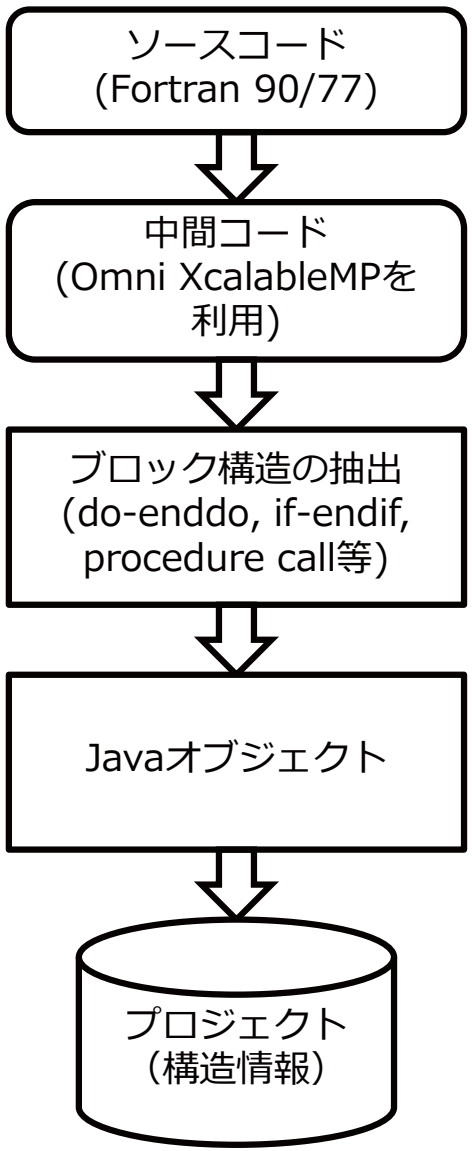
- 南が提案するシステマティックなチューニングのアプローチ（以後、ワークフロー）
- コンパイラ、プロファイラ、デバッガ、IDEなどは、それぞれのワークフローのいずれかで使用される。これらはベンダから提供されている。
- ただし、個々のワークフローにおいて十分なツールが用意されているか、というとそうでもない。
- とくに、Preliminaryでは、問題の取り掛かり方に個々のエンジニアの経験に依存することが多く、ツール化が遅れていた。この数年は、南が中心となりその問題を埋めるためにK-scopeを開発してきた。

ソースコード分析ツール: K-scope

- ソースコードリーディングおよびホットスポットのスクリーニングのためのツール（ワークフローのPreliminaryで用いることを想定）

- コンセプト
  - Fortran プログラムの論理構造（とくにループ、分岐、プロシージャ呼出し）を可視化し、コードリーディングを支援する。
  - 京の基本プロファイラと詳細プロファイラをソースコードに対応づけて可視化を行い、動的なコード構造解析を支援する。
  - F_Front (atool) によって生成された中間コードを用いることで、分析ツールにおける Fortran 構文解析器の実装を省略する。
  - ポータビリティの観点よりJavaで実装
  - OSSライセンス (Apache 2.0 License) で配布

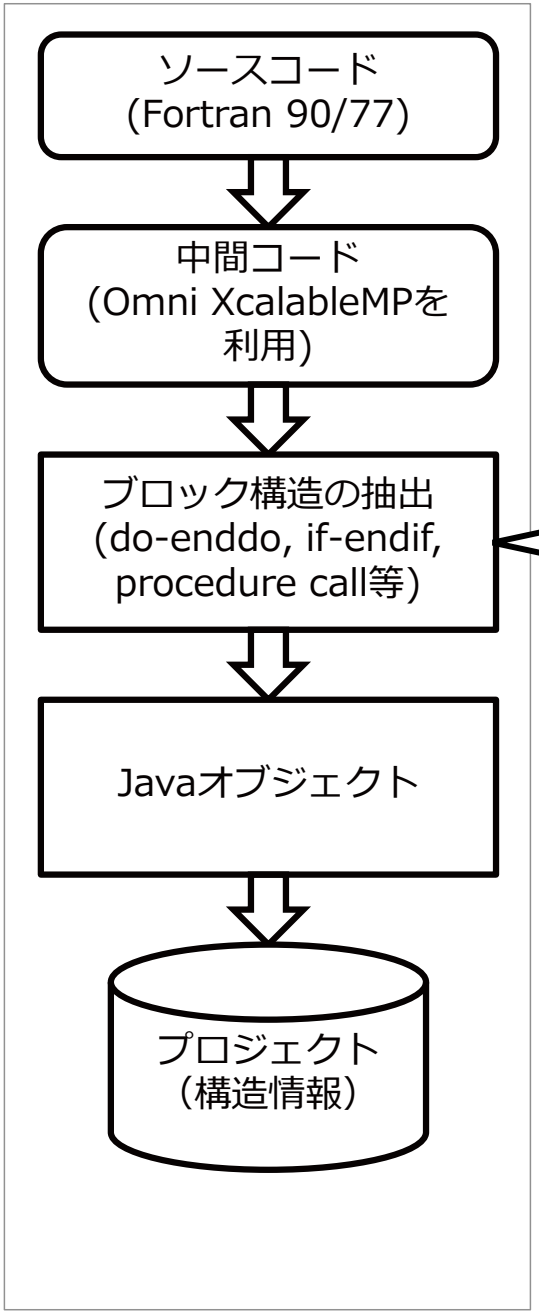
デザインの骨子



- フォーカスすべきプログラムの論理構造をループ、分岐に限定しても、構造的に可視化するためには、構文解析器が必要
- Fortran の構文解析は（控えめに）少し面倒

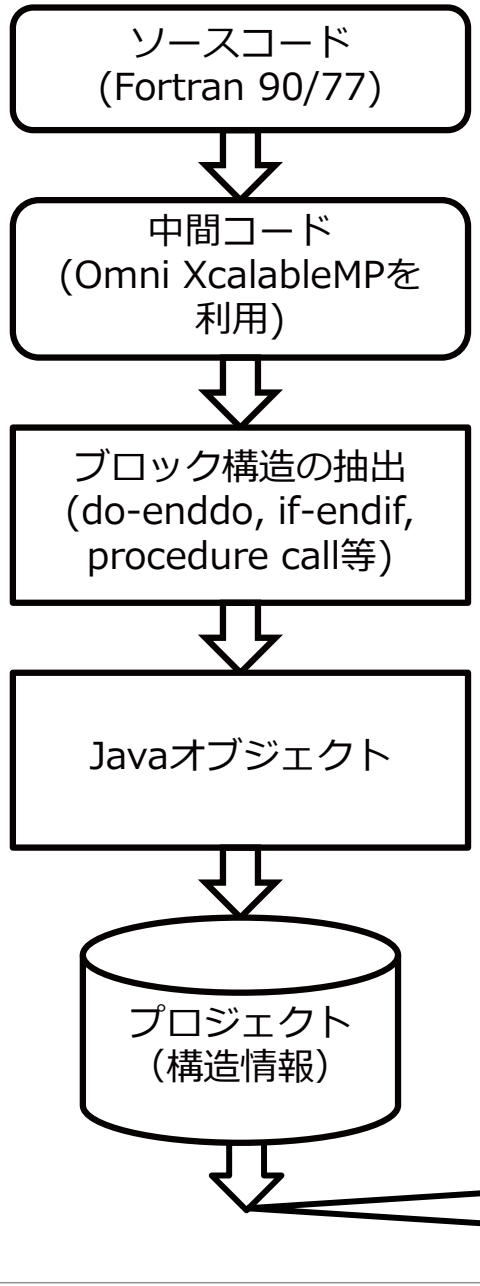
- 解決方法
  - Omni XcalableMPコンパイラのフロントエンドの出力（中間コード）を利用する。
  - F_Frontのロジックは取り込まずに、中間コードを利用することで、疎結合な仕組みを担保
  - 基本は、ユーザが予めソースに対応する中間コードを用意しておく（K-scopeは中間コード作成に介在しないモードがデフォルト）。

JavaクラスによるFortran構文の対応付け

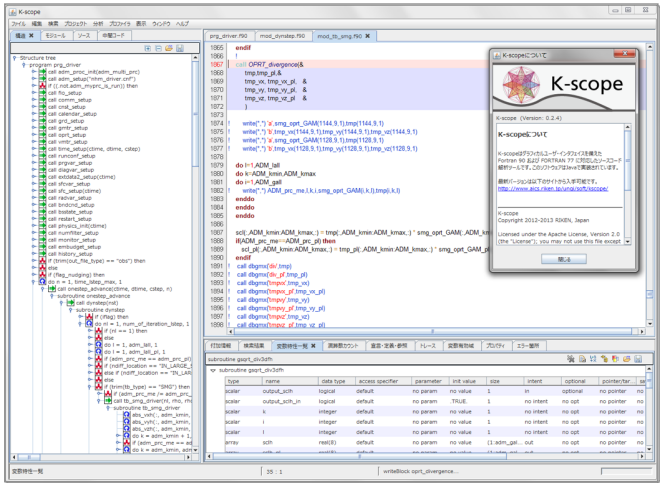


block type	Fortran statement
loopBlock	do, do while, end do, continue
selectionBlock	if, else, end if select, case, end select
substitutionBlock	X=Y, operator, array-expression
useStateBlock	use
procedureCallBlock	call
gotoBlock	goto
pauseBlock	pause
returnBlock	return
terminationBlock	stop
breakBlock	cycle
commonBlock	common
equivalenceBlock	equivalence
dataBlock	data
dynamicAllocationBlock	allocate
dynamicDeallocation	deallocate
dynamicNullification	nullify
directiveBlock	!omp, !ocl

ツール出力例

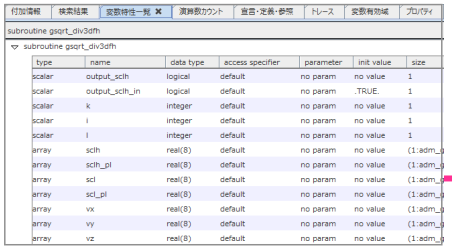


スクリーンショット

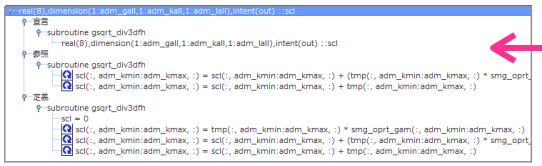


分析ビュー例

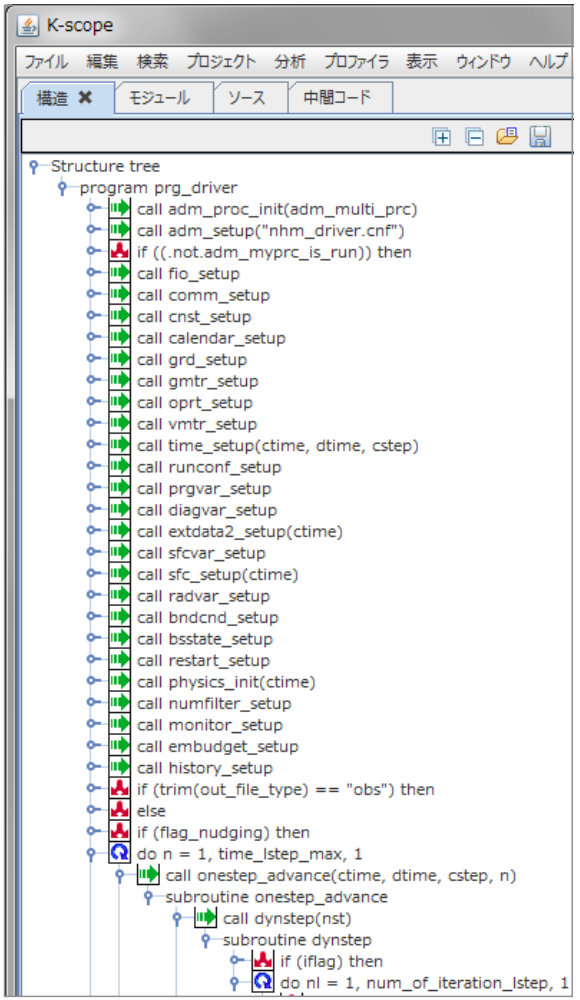
変数の特性一覧



変数の宣言・定義・参照の一覧



ツリー形式によるプログラム構造の可視化例



ツールの起動

- 実行モジュール（jar形式）を以下URLよりダウンロード
  - http://www.aics.riken.jp/ungi/soft/kscope/

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

- インストール作業は必要なく、単純に *kscope.jar* と *properties.xml* のみで起動
- 一般的なPC（例えば、CORE i5, メモリ4GB）で動作確認
- Java 7が必要

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

- もし、メモリ不足する場合は、以下コマンドでJVMのヒープを拡張する必要あり

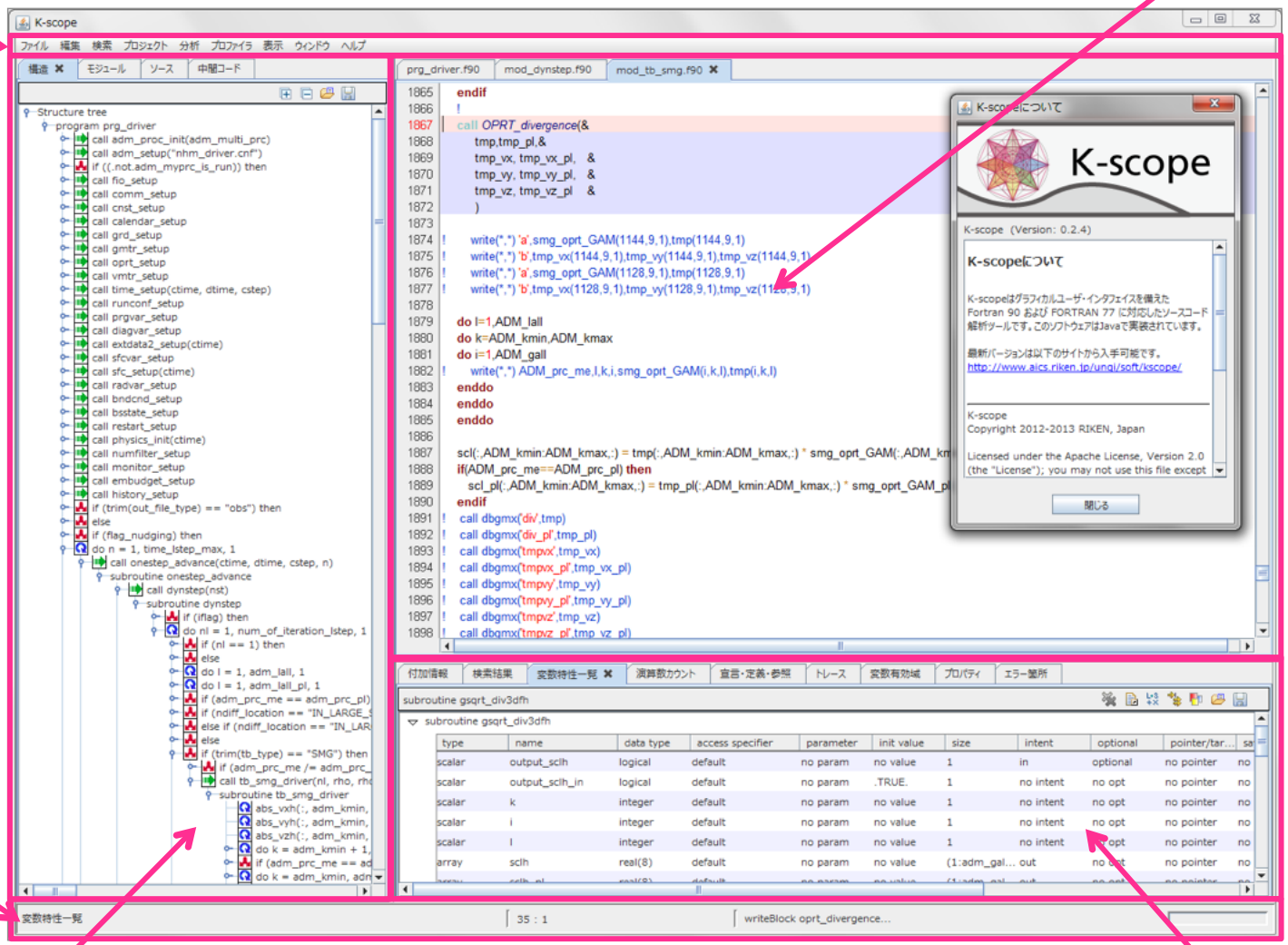
```
$ java -jar -Xmx1024m kscope.jar
```

- VMオプションによりロケールを指定可能（日本語と英語を選択可）

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

インターフェース

- 新規プロジェクト作成後の画面例
- 画面構成としては典型的なものを採用 ⇒ 直感的な操作が可能

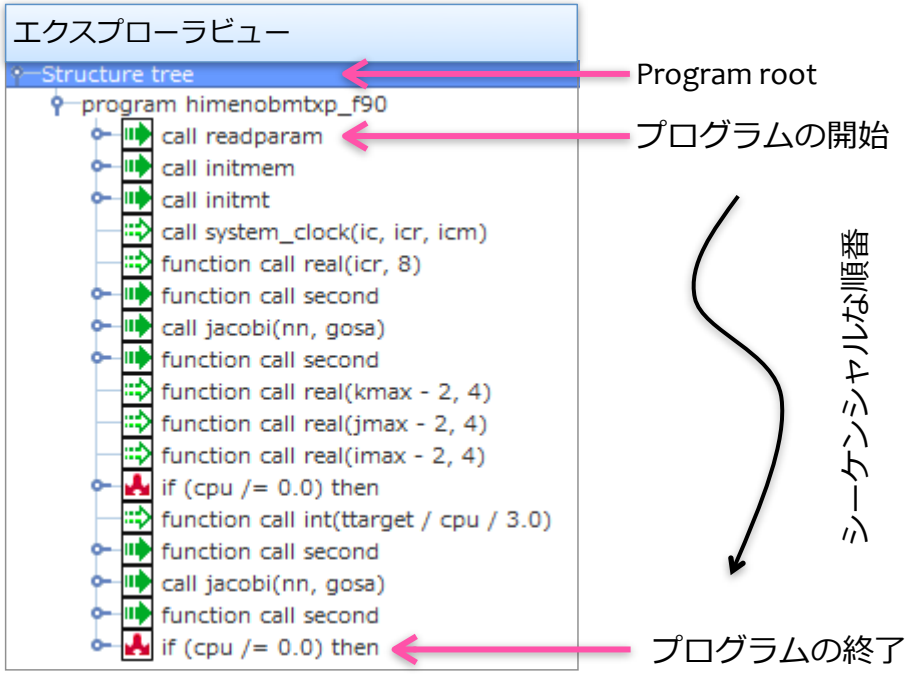


静的プログラム解析 (1/2)

- 実機による計測が必要ないため、小さなコストで解析が行える。

- 一般的なプロファイラでは、手続き呼出しにフォーカスしてコールグラフを作成するのに対して、K-scopeではチューニングに必要なループ、分岐などの構造にフォーカスしてツリー可視化を行う（当然、手続き呼出しもツリー表示が可能）。

- 以下のツリーは姫野ベンチマークによる解析例

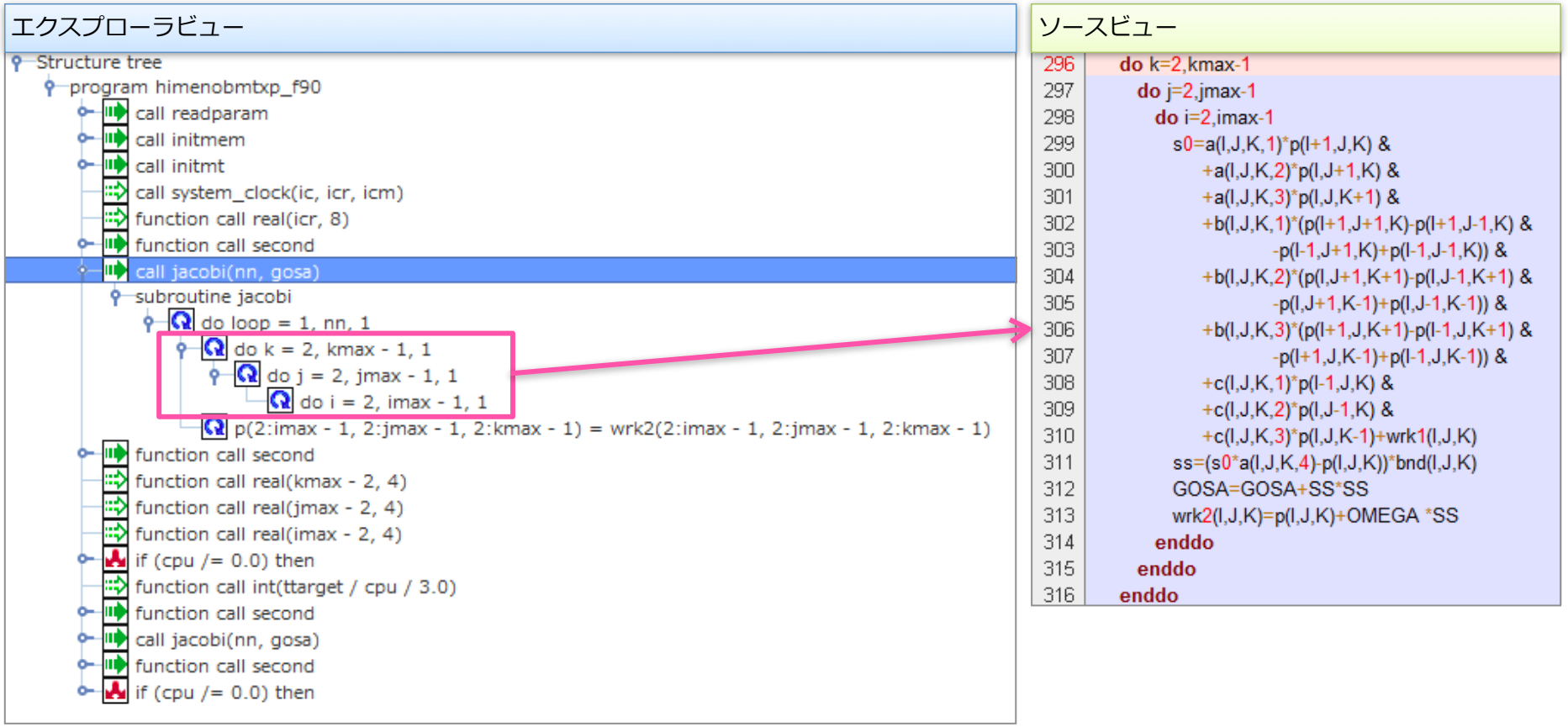


ツリー要素に用いられているアイコンの意味

	分岐	if, else, where, select, case
	ループ	do
	プロシージャ呼出し（中間コードでパースされたもの）	中間コード作成時に呼出し先についてパースされているもの
	プロシージャ呼出し（中間コードでパースされていないもの）	中間コード作成時に呼出し先についてパースされていないもの（MPI関数など）

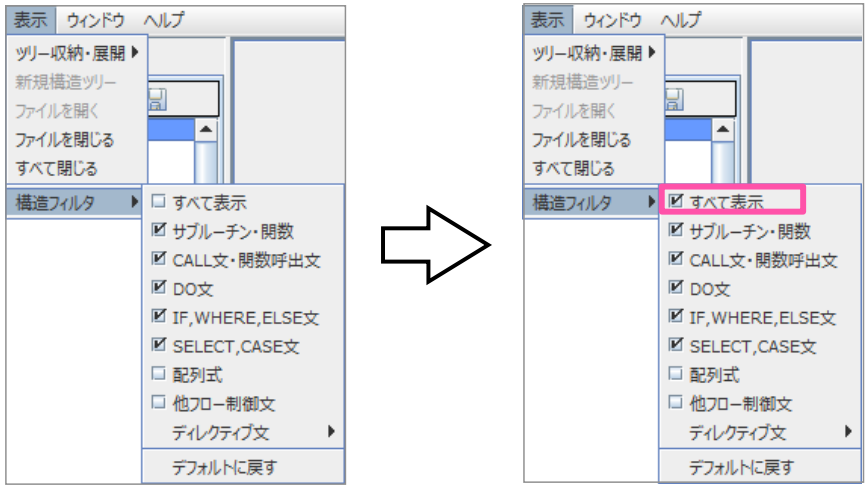
静的プログラム解析 (2/2)

- 姫野ベンチマークの “*jacobi*” サブルーチンのループを展開（左）
- ツリー要素に対応する部分をソースビューに連動して表示させることも可能（右）



ツリーのフィルタリング

- K-scopeの構造情報は、Fortran構文の粒度で保持
- 特定の構文についてのみ表示させるためのフィルタリング機能を提供



チェックボックスをオフにすることで、必要のない構文をツリー上で非表示にすることが可能

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)
    dt = 1.0 / real(icr, 8)
    function call real(icr, 8)
    nn = 3
    cpu0 = second()
    function call second
    call jacobi(nn, gosa)
    cpu1 = second()
    function call second
    cpu = (cpu1 - cpu0) * dt
    flop = ((real(kmax - 2, 4) * real(jmax - 2, 4)) * real(imax - 2, 4)) * 34.0
    function call real(kmax - 2, 4)
    function call real(jmax - 2, 4)
    function call real(imax - 2, 4)
    if (cpu /= 0.0) then
    nn = int(ttarget / (cpu / 3.0))
    function call int(ttarget / cpu / 3.0)
    cpu0 = second()
    function call second
    call jacobi(nn, gosa)
    cpu1 = second()
    function call second
    cpu = (cpu1 - cpu0) * dt
    if (cpu /= 0.0) then
    score = xmflops2 / 82.84
    pause
```

動的プログラム解析

- 実際のプログラムの挙動は実行時に決定される。
- 解析には実機またはエミュレータが必要
- K-scopeでは京にフォーカスした動的プログラム解析を支援

- 富士通製プロファイラ
  - 基本プロファイラ: fipp (出力ファイル DProf*)
  - 詳細プロファイラ: fapp (出力ファイル EProf*)

- 静的プログラム解析によって得られた情報と、プロファイラ情報を並べて表示させることで、より正確なボトルネックが特定できる。

- その他の利点
  - プロファイル結果に行情報が含まれているため、中間コードがなくてもソースコードとの重畳は可能 ⇒ シンプルモード

基本プロファイラ (1/2)

- 基本プロファイラは手続き・ループ・行単位の粒度でのコスト情報を提供する。

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 | 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_15365_000000_60190.0000_0.0000_602.0000_0.0000_0.0000_112801.3750_50.0000_0.0000_.pri

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
601	99.83	jacobi_	himenobmtxp.f90	276:327
1	0.17	initmt_	himenobmtxp.f90	224:253

ソースビュー

himenobmtxp.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
```

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 | 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_15365_000000_60190.0000_0.0000_602.0000_0.0000_0.0000_112801.3750_50.0000_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
428	71.10	jacobi_	himenobmtxp.f90	298:298
73	12.13	jacobi_	himenobmtxp.f90	320:320
62	10.30	jacobi_	himenobmtxp.f90	311:311
17	2.82	jacobi_	himenobmtxp.f90	314:314
9	1.50	jacobi_	himenobmtxp.f90	312:312
8	1.33	jacobi_	himenobmtxp.f90	315:315
4	0.66	jacobi_	himenobmtxp.f90	299:299
1	0.17	initmt_	himenobmtxp.f90	246:246

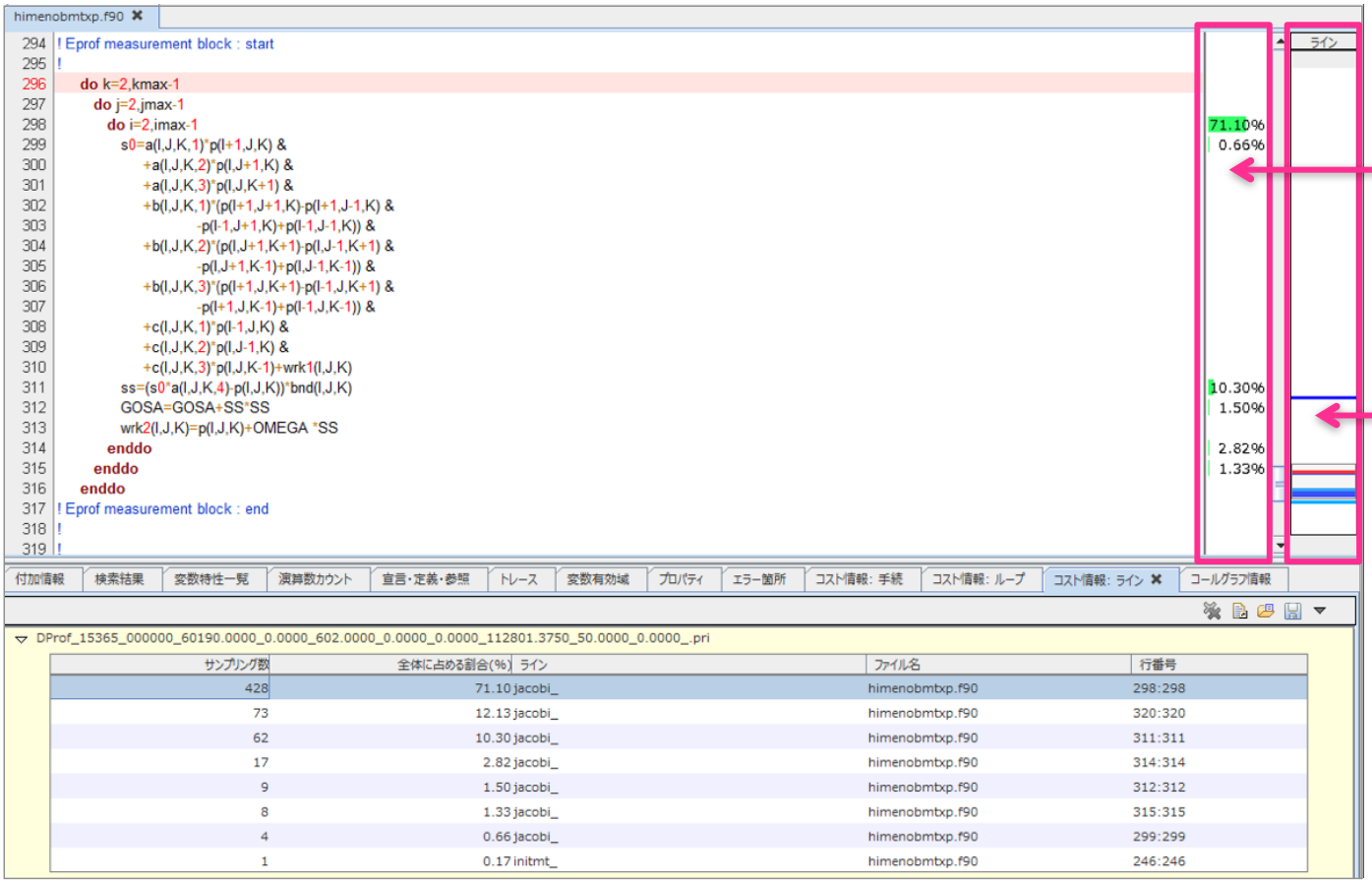
ソースビュー

himenobmtxp.f90 ✖

```
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
```

基本プロファイラ (2/2)

- メニューバーから 「プロファイラ」 > 「プロファイル結果の読込」 を選択
- ソースビューの右側にバーとルーラが表示される。



コストバー
コストの値をソースビューの行に対応づけて表示する。

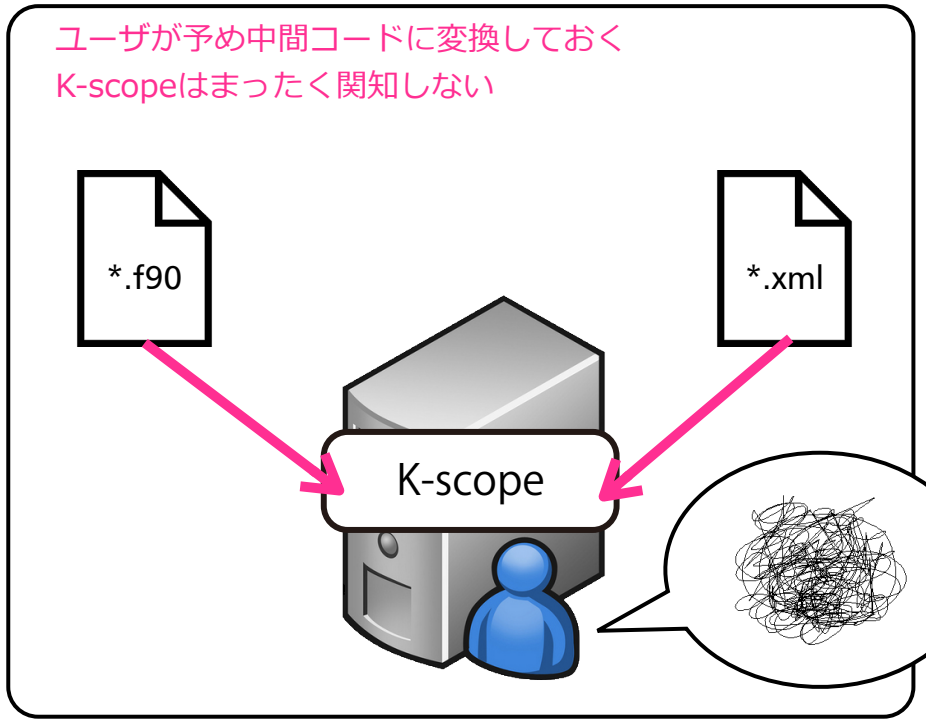
コストルーラ
ルーラはソースコード全体を俯瞰的に表示している。コスト値が存在するところが色付きの範囲により可視化される。マウスで選択することで該当する箇所がソースビューに表示される。

詳細プロファイラの読込にも対応

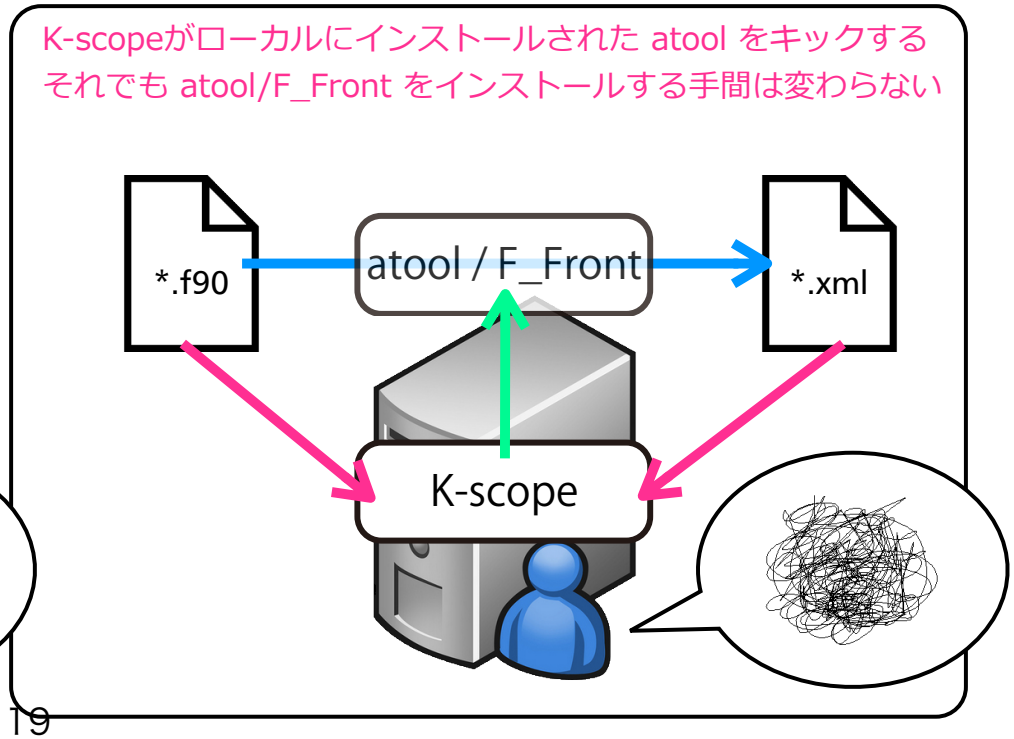
最近の開発状況(1/2)

- 課題
  - K-scopeは新規プロジェクト作成時に、中間コードを用意する必要がある。
  - 中間コード作成のためにF_Front 等の環境を各ユーザが用意する必要がある。
  - Omni XcalableMPのインストールに慣れていないと煩雑な作業
  - ユーザビリティの改善が目下の課題

フルモード１(既存の中間コードを読み込む)



フルモード２（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。）

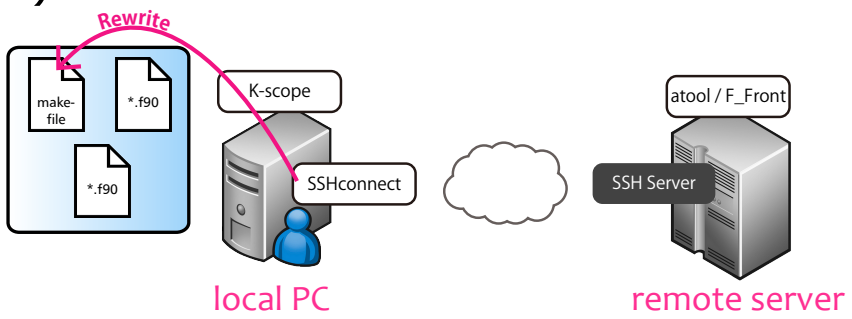


最近の開発状況(2/2)

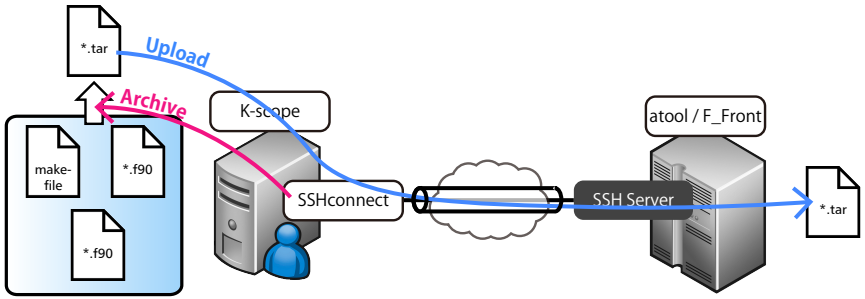
- プロジェクト
  - 2013年5月開始
  - AICS利用高度化研究チームと、チューニング作業支援システムの構築に関する連携
  - K-scopeのユーザビリティの改善を検討
- 骨子
  - まずは K-scopeを利用する立場で、Omni XcalableMP の導入の手間を軽減したい。
  - 現状、京のフロントエンドに F_Front がインストールされている。
  - atool も試験的に導入されている。
  - K-scopeからリモートにインストールされている atool/F_Front を活用することで、手元のPCにインストールする手間を省く（フルモード2の拡張）。
  - SSHを用いたローカル ⇔ リモート間のファイル転送、リモートのコマンド実行機能をユーティリティ化（Peter Bryzgalov研究員が開発）

SSHconnect の機能概要

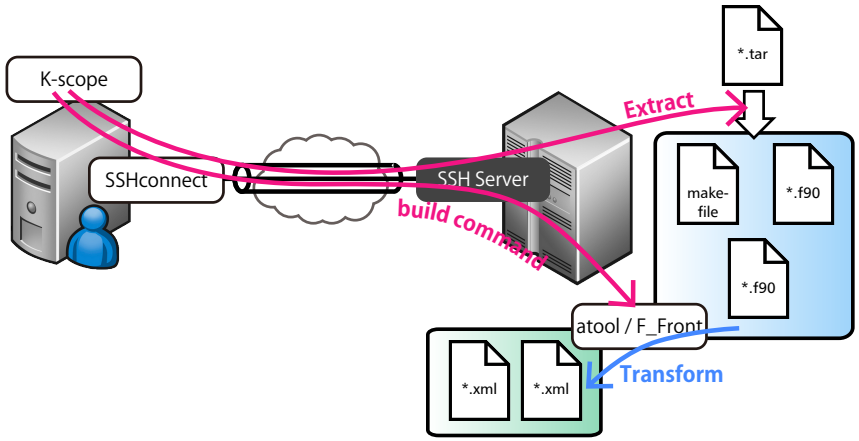
(1) プレースホルダの置き換え



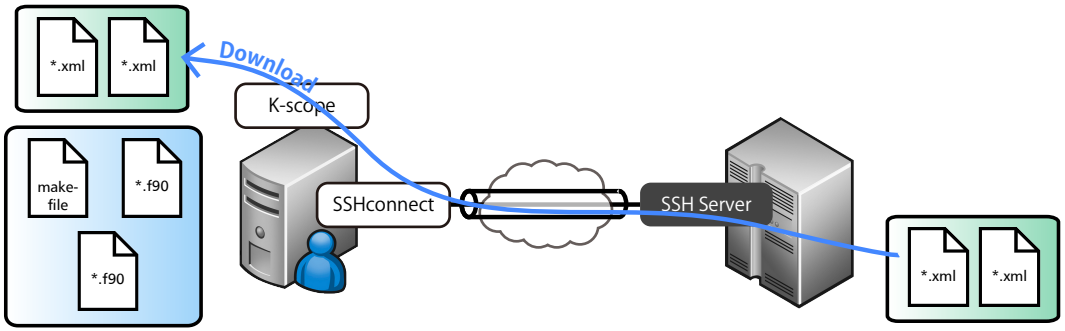
(2) アーカイブ + アップロード



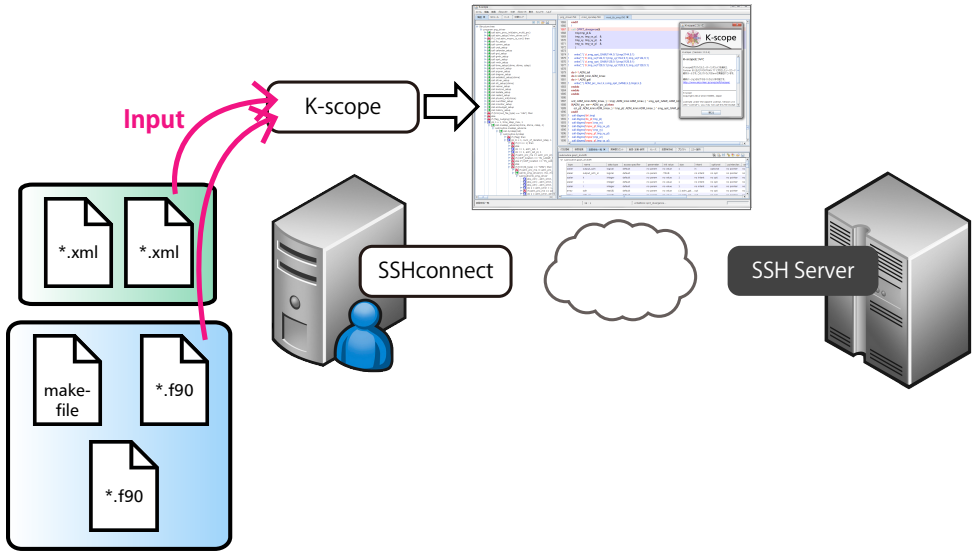
(3) 展開 + 中間コードへの変換



(4) 中間コードのダウンロード



(5) K-scopeによるロード



まとめ

- Omni XcalableMPの中間コードを利用したFortranコード分析ツールを開発した。
- K-scopeは、チューニングのワークフローにおいて、Preliminaryで使用されることを想定したツールである。
- ソースコードリーディングおよびホットスポットのスクリーニングが K-scope の真骨頂
- ユーザビリティの改善として、リモートサーバで中間コードを作成する機能を追加した。
- 京のフロントエンドをターゲットとした利用整備は進行中
- Version 0.2.4 を 2013年10月31日 にリリース

ご清聴有難うございました。

以下のURLからソースコード、実行モジュール、ドキュメントがダウンロードできます。
http://www.aics.riken.jp/ungi/soft/kscope/

質問、アナウンス用のメーリングリストも用意してあります。もしご興味ありましたら、登録する名前、所属、メールアドレスの情報とともに下記のメールアドレスにご連絡ください。

kscope-admin@riken.jp

- full name
- affiliation
- e-mail address