スーパーコンピュータ「京」の性能プロファイルとデバッグ

Performance Profiling and Debugging on the K computer

● 井田圭一　● 大野康行　● 井上俊介　● 南　一生

あらまし

著者らは，スーパーコンピュータ「京」に向けたアプリケーション開発支援ツールを開発した。

本稿では，本開発支援ツールの主な機能であるアプリケーションを高性能化するためのプロファイル機能，およびアプリケーションを検証するためのデバッグ機能について述べる。プロファイル機能およびデバッグ機能を開発するに当たっては，まず利用者におけるアプリケーションの高性能化および検証の作業手順を定義し，次にその個々の作業で必要となる開発支援ツールのあるべき姿を検討し開発した。ここでは，この作業手順に沿って各種開発支援ツールを紹介する。特に，「京」の特徴としてはその大規模性が挙げられるが，大規模アプリケーションのプロファイルおよびデバッグには従来の延長では解決できない特有の課題が存在しており，これらの課題に対する新たな取組みについても述べる。さらに，「京」の大きな特徴である高性能CPU SPARC64 VIIIfxやTofuインターコネクトなどの最先端ハードウェアに特化したプロファイル機能についても言及する。

Abstract

We have developed application-development support tools for the K computer. This paper describes profiling functions for raising the performance of applications and debugging functions for testing applications as main functions of these tools. In developing these tools, we first defined the work procedure that a user would follow for improving the performance of an application and testing it. We then investigated the form that these application-development support tools should take for each task in that procedure. We here introduce these tools in conjunction with those tasks. Additionally, while the large-scale configuration of the K computer is one of its major features, existing profilers and debuggers for large-scale applications still have problems that have yet to be solved, and we here describe new measures for addressing those problems. We also touch upon profiling functions specifically developed for the advanced hardware of the K computer such as the high-performance SPARC64 VIIIfx processor and Tofu interconnect.

まえがき

スーパーコンピュータ「京」(注)は，8万を超えるノードから構成される巨大な分散並列型計算機であり，さらに高性能CPU SPARC64 VIIIfxやTofuインターコネクトなどの最先端ハードウェアを採用している。著者らは，この「京」に向けたアプリケーション開発支援ツールを開発した。

本稿では，本開発支援ツールの主な機能であるアプリケーションの高性能化のためのプロファイル機能，およびアプリケーションの検証のためのデバッグ機能について述べる。プロファイルおよびデバッグ機能を開発するに当たっては，まずは利用者におけるアプリケーションの高性能化および検証の作業手順を定義した。ここでは，開発のねらいを示した後，その作業手順に沿って各種開発支援ツールを紹介する。

開発のねらい

「京」向けのプロファイルおよびデバッグ機能を開発する上で，大きく三つのねらいを設定した。第一には大規模並列処理アプリケーションへの対応と新規ハードウェアへの対応，第二には標準インタフェースの活用，第三にはGUI機能の高度化の実現を目指した。

「京」向けのアプリケーションソフトウェアは，システム同様に大規模であり，また，「京」の最先端ハードウェアを活用することが求められている。そのため，開発支援ツールの第一のねらいは，大規模並列処理アプリケーションに対応し，また各種最先端ハードウェア向け新機能をサポートすることである。詳細は個々に後述するが具体的な機能としては，数万プロセス並列アプリケーションに対応したプロファイラとデバッガ，数万プロセスの性能情報の大規模データ表示，実行時ライブラリ（RTS：Run Time System）およびメッセージパッシングインタフェースライブラリ（以下，MPIライブラリ）の自己チェック機能，プロファイラにおけるSPARC64 VIIIfxのハードウェア性能計測カウンターのサポート，Tofuインターコネクトを意識した通信コスト表示などである。

一方，開発支援ツールを取り巻く環境としては，インタフェースの標準化およびツールコンポーネント自身のオープンソース化が進んでおり，ハードウェアに依存しないソフトウェアプラットフォームが構築されつつある。これらのオープン化の流れを取り入れ，各種機能をできる限り標準インタフェースに載せることも開発のねらいとした。具体的には，コンパイラと各種ツールのインタフェースとなるデバッグ用の情報としては，DWARF 2[(1)]を採用した。これにより，オープンソースの形態で提供されている様々なデバッグツールの活用が可能となる。また，CPUにはハードウェア性能計測カウンターが備えられており，実行命令数やキャッシュミス数などを厳密に計測することができる。このインタフェースとしてはPAPI[(2)]を採用した。さらに，各種イベントに対応したログ出力機能としては，VampirTrace[(3)]がデファクトスタンダードであり，これも標準仕様として採用した。

大規模並列処理アプリケーションのプロファイル情報やデバッグ情報を表示するためには，高性能なGUI機能が必要となる。そこで第三のねらいとして，GUI機能の高度化を掲げた。従来の多くの開発支援ツールではGUIとしてX Window System（X11）を採用しているが，利用者端末の性能向上を享受することが難しく，GUI機能および性能に限界があった。これらの課題を解決するために，Webアプリケーションの分野で急速な進化を遂げているリッチインターネットアプリケーション技術に基づくAdobe社のAIR[(4)]を新たにGUI基盤として採用した。AIRクライアントでは，サーバからは必要最小限のデータを受け取り，視点を変えた場合などの再描画は全て利用者端末側だけで処理するために，利用者端末側のグラフィクス性能を十分に引き出すことができる。

アプリケーションの高性能化ステップ

スーパーコンピュータは，高速に計算することに存在意義があるため，スーパーコンピュータにおいてアプリケーションの性能を把握することは，本質的な作業であると言える。また，もし把握した性能が妥当でない場合は，性能が劣化している原因を分析し，性能を向上させることが必須とな

(注)　理化学研究所が2010年7月に決定したスーパーコンピュータの愛称。

る。これらのアプリケーションの性能把握および高性能化作業は，**図-1**に示すように，性能特性分析と，それに引き続く互いに独立した複数の高性能化のステップから成る。なお，本稿では詳細プロファイラのうち，CPU演算処理とプロセス並列処理の高性能化を掘り下げて紹介する。

● 性能特性の分析

最初のステップでは，まず通常の実運用時のアプリケーション性能を大局的に把握することから始まる。今回，この目的のために，「基本プロファイラ」を開発した。基本プロファイラは，次のような特徴を持つ。

(1) 再翻訳することなく，高度に最適化したオブジェクト・コードのままで性能情報を取得することができる。
(2) 通常のジョブ走行スクリプトに対してfippコマンドを追加するだけで，基本的な性能情報を一度に取得することができる。
(3) CPU演算処理，スレッド並列処理，プロセス並列処理の三つの観点から，それぞれの処理状況を表示することができる。
(4) 性能取得方法が，サンプリングによる一定時間間隔の走行情報の監視をベースとしているため，オーバヘッドは少なくかつ一定である。

基本プロファイラにより高性能化が必要な処理が判明した場合，次のステップに移る。先述したCPU演算処理，スレッド並列処理，プロセス並列処理の三つの観点は，高性能化作業においてそれぞれが独立しており，別々の高性能化アプローチを必要とする。しかし，いずれの観点の高性能化においても，今回開発した「詳細プロファイラ」を主要ツールとして使用する。詳細プロファイラの特徴は次のとおりである。

(1) 特定のプログラム箇所に対して詳細な分析を行うためにアプリケーション中にサービスライブラリを呼び出す方式で測定区間の開始および終了を指定する。
(2) 性能取得方法はサンプリングではなく測定区間におけるCPU処理，スレッド並列処理，プロセス並列処理の実行回数の厳密な測定であるために，区間ごとの詳細かつ正確な情報を取得することができる。
(3) 測定した時間を統計的な平均値で取り扱わずに測定ごとの時間を個々に分析することができるため，ユーザ関数が呼ばれるごとにその性能特性が変わるような場合にも対応できる。

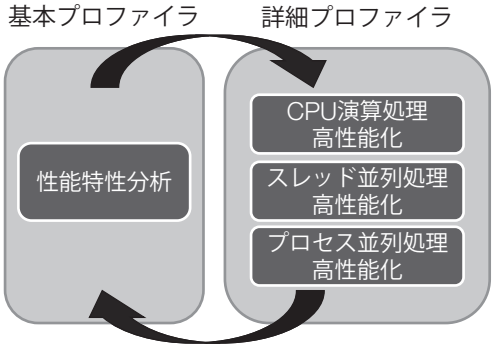

図-1　アプリケーションの高性能化ステップ

● CPU演算処理の高性能化

高性能CPU SPARC64 VIIIfxには，各種ハードウェア性能計測カウンターが実装されているが，特に，実行時間をCPUの動作状態（命令実行中，メモリアクセス待ち，演算待ちなど）で分類し，CPU内のどの部分にボトルネックがあるかを把握し，性能分析や改善を行うことができる。このような手法は，一般にサイクルアカウンティング[5]と呼ばれている。CPU演算処理に性能問題がある場合は，詳細プロファイラのハードウェアモニタ機能を使い，サイクルアカウンティング手法により問題を分析する（**図-2**）。

高性能化手順の例を示すと，サイクルアカウンティング分析によりメモリアクセス待ちの割合が多いことが判明した場合は，キャッシュの有効活用を図る必要がある。具体的には，アプリケーションのデータ構造（空間的）やデータ参照順序（時間的）を変更し，データ（キャッシュライン）の再利用性を高め，キャッシュのヒット率を向上させ，メモリアクセスを削減する。また，SPARC64 VIIIfxは複数の演算を一度に実行するSIMD命令も拡張されており，高性能を得るためにはSIMD命令の割合にも注意を払う必要がある。SIMD命令の割合が低いことが判明した場合には，適切なコンパイルオプションの選択やプログラムソースの書換えによりコンパイラの最適化を促す必要がある。

● プロセス並列処理の高性能化

前述のとおり「京」は，8万を超えるノードから構成される巨大なシステムであり，アプリケーショ

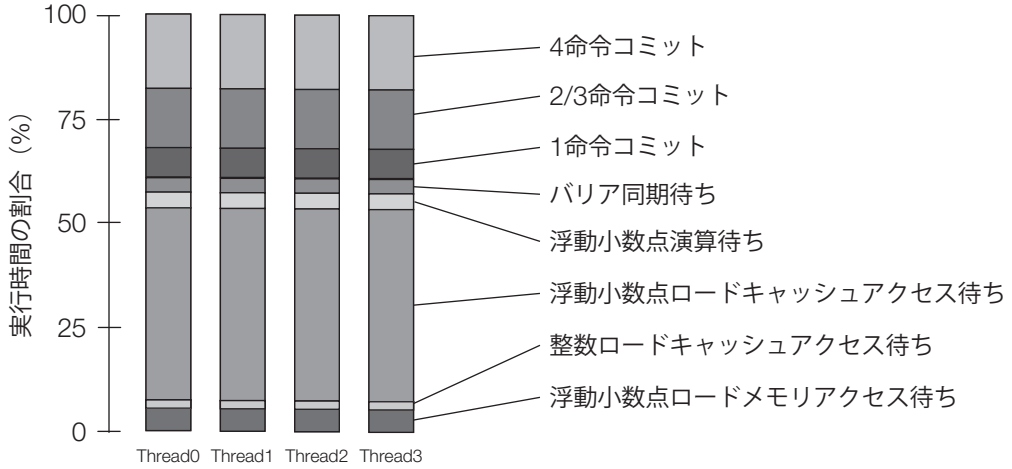

図-2　サイクルアカウンティング手法による分析例

ンの規模としても，ノード数以上のプロセス並列数のジョブが走行可能である。プロセス並列処理における高性能化の観点は，大きく二つ挙げられる。

まず一つ目が，プロセス並列化におけるアプリケーションの負荷バランスである。プロセス並列処理は，一般にはMPIライブラリを使い並列化されるが，プロセス間の負荷が一定でない場合は通信・同期待ち時間が発生し高い性能は得られない。負荷バランスの乱れは，様々な要因で発生する。アプリケーションに内在する，例えば領域分割した各領域の計算量の違いのような本質的な要因もあれば，ノード間で仮想メモリのレイアウトが微妙に異なるためにキャッシュミス率にブレが生じ，最終的にアプリケーション性能が乱れる場合もある。もちろん，プロセス並列処理を行うアプリケーションでは数多くの通信処理が発生するため，通信処理に関連するブレも発生する。一例を挙げると，通信ネットワークにおける衝突に起因したブレである。

このような各種バランスの乱れを是正するためには，まず現状の性能値を大局的に把握することが重要である。基本および詳細プロファイラでは，アプリケーション全体の状況を鳥瞰（ちょうかん）するために，今回新たに各種性能データの大規模な可視化を実現した。基本プロファイラによるコスト分布表示の例を**図-3**および**図-4**に示す。この例では，NAS Parallel Benchmarks[(6)] のCGの性能を分析している。画面の下半分で1024並列のアプリケーションの一つのプログラムループ（conj_grad 1128～1148行）の経過時間に相当するサンプリングコストを表示している。表示の形状は，図-3ではアプリケーショントポロジーである16×4×16の3次元トーラス構造を輪切りにしており，プロセスごとにサンプリングコストの大小を色相で表している。アプリケーショントポロジーに沿って表示することで，当該ループのコストがX軸（図中横軸方向）で大きく変化している様子が分かる。図-4では，同じデータを一次元に展開し棒グラフで表示している。このグラフにより，プロセスごとの負荷バランスの乱れの程度と状況を正確に捉えることができる。さらに画面の最下部では，色相の凡例の上にヒストグラムを表示しサンプリングコストの分布を表している。このようにアプリケーショントポロジーや一次元展開での性能データの表示と，性能データのヒストグラムを表示することで，アプリケーション全体におけるコストバランス状況を鳥瞰することができる。

プロセス並列処理における二つ目の高性能化の観点は，MPIライブラリコストの削減である。通信が頻発する並列化を行った場合には，MPIライブラリの処理が増え，コストが増加し，性能の劣化につながる。対処としては，MPIライブラリの使用方法を見直し，通信操作の削減やメッセージ数の削減（一括化）を行うことが必要となる。詳細プロファイラで区間指定を行うと，その区間で実行されたMPIライブラリの詳細情報（MPI関数ごとの呼出し回数，処理時間，平均メッセージ長など）を得ることができるため，上記の高性能化の

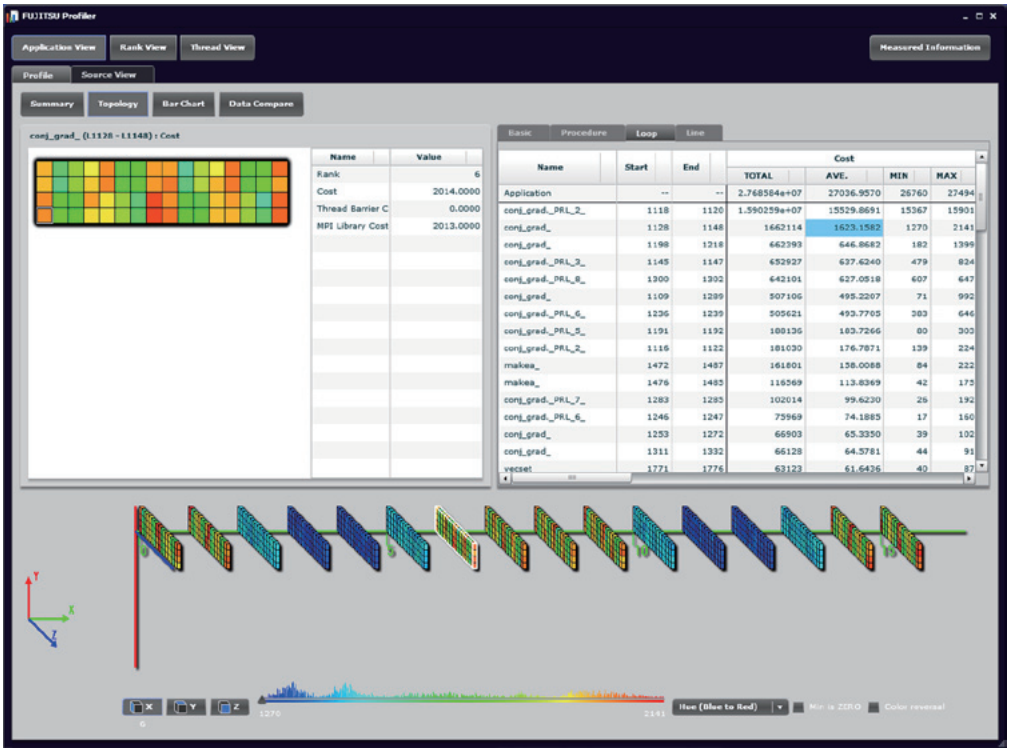

図-3　基本プロファイラによるコスト分析表示(トポロジー表示)
左上：プロセスごとのコスト情報(部分拡大図)
右上：プログラムループごとの性能値の一覧
下：　プロセスごとのコスト情報(全体の3Dトポロジー表示)

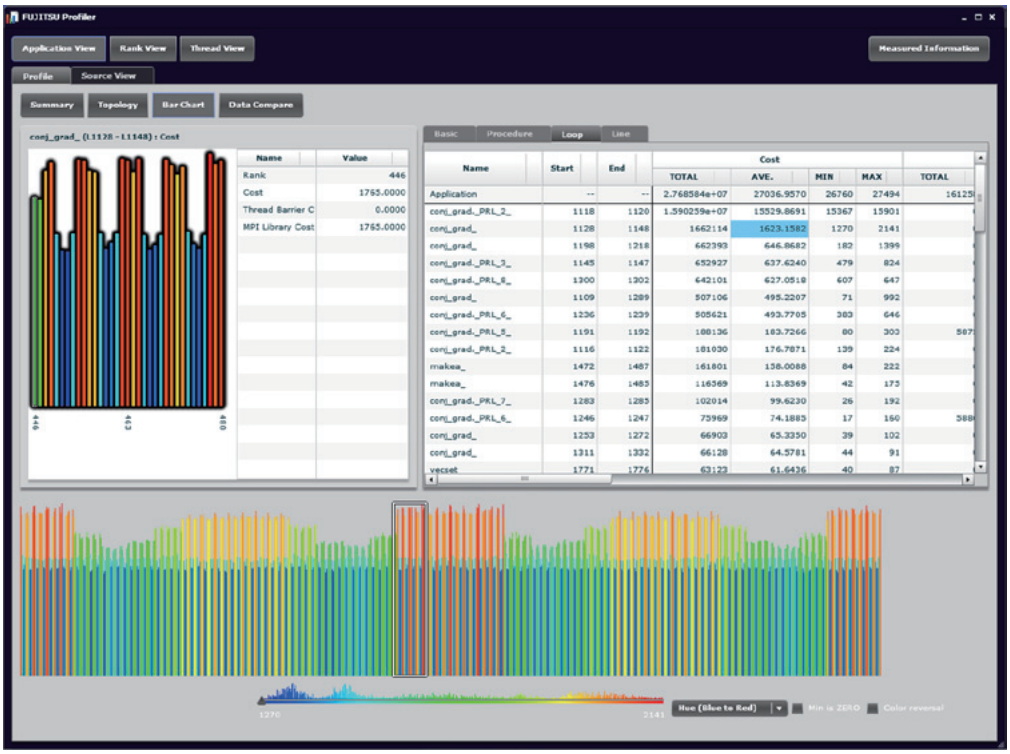

図-4　基本プロファイラによるコスト分析表示(一次元表示)
左上：プロセスごとのコスト情報(部分拡大図)
右上：プログラムループごとの性能値の一覧
下：　プロセスごとのコスト情報(全体の一次元表示)

効果を確認することができる。なお，MPIライブラリコストは，詳細プロファイラに加えて，MPIライブラリの「MPI統計情報」により取得することもできる。これは，MPIライブラリ自身が内部の処理内容に応じてログを取り，アプリケーションの実行終了時に統計情報を出力する機能である。

「京」においては，通信処理の最適化のためには3Dトーラスネットワーク（Tofuインターコネクト）や，高機能バリア通信機能を意識する必要もある。このために，詳細プロファイラやMPI統計情報に新たに通信距離（トーラスにおける通信ホップ数）の計測や，高機能バリア通信機能の計測を追加した。

さらにMPIライブラリの通信処理を極限まで最適化する場合には，Tofuインターコネクトにおける通信パケットの処理状況を把握する必要がある。このために，「Tofu PA（Performance Analysis）情報」を開発した。Tofu PA情報は，Tofuインターコネクトのネットワークルータごとに送受信パケット数，送受信バイト数，送信先バッファ残量などを計測する仕組みであり，詳細プロファイラによって指定した区間ごとに計測することができる。Tofu PA情報の測定結果を分析することで，インターコネクトネットワークにおける通信処理を平準化したり，パケットの衝突を回避したりすることにより，インターコネクトの利用効率を向上させることができ，結果としてアプリケーションの性能向上につなげることが可能である。

アプリケーションの検証ステップ

前述のようにスーパーコンピュータは，高速に計算することに存在意義があるが，同時に計算の正確さも求められる。アプリケーションにおいて，計算の正確さとは設計時の仕様に沿って動作することであり，この確認を検証と言う。「京」のような超大規模並列システムでは，複雑度が増し検証作業が困難であるため，後述する自己チェック機能などの施策が新たに必要となる。

アプリケーションの検証ステップの中でバグの発生を起点としたデバッグ作業は，不具合の分析および切分けと，それに引き続く互いに独立した複数のデバッグのステップから成る（**図-5**）。ここでは，このステップに沿ってデバッグの手法を紹介する。なお，CPU演算処理とプロセス並列処理のデバッグを掘り下げる。

● 不具合の分析および切分け

バグの現象としてはSIGSEGV（SIGnal SEGmentation Violation）などの異常終了，デッドロック，計算結果の間違いなどに分類することができるが，この現象を更に詳細に分析し，バグの発生原因箇所を絞り込むことをバグの切分けと言う。ここでは，まず，CPU演算処理，スレッド並列処理，プロセス並列処理の三者に切り分けることを考える。

切分けの手順は現象の確認から始まり，複数の翻訳時オプションや実行時オプションのバリエーションを使用した実行，さらにデバッガの利用など手順は複雑であるが体系化されている。代表的なものを次に示す。

SIGSEGVなどの異常終了では，異常発生箇所がアプリケーションの呼出し関係を示す「トレースバックマップ」で表示されるため，その呼出し関係から切り分ける。例えばMPIライブラリの中で異常終了していたことがトレースバックマップから確認できた場合，プロセス並列処理固有の不具合と判断することができる。

アプリケーションのデッドロックでは，「会話型デバッガのジョブIDアタッチデバッグ」が有効である。会話型デバッガは，停止したアプリケーションに対して外部からアタッチし，アプリケーションをデバッガの制御下に置き，会話的操作により停止した原因を調査することができる。会話型デバッガのスナップショットを**図-6**に示す。画面左側に各プロセスやスレッドごとの停止位置が表示されており，これらの情報を解析することでデッ

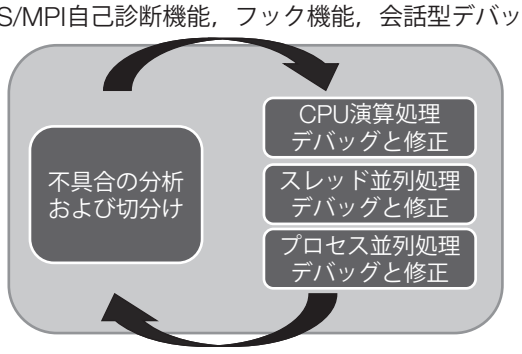

図-5　アプリケーションのデバッグステップ

ドロックの原因を切り分けることができる。

● CPU演算処理のデバッグ

先述したとおり，大規模のアプリケーションにおいては複雑度が増し検証作業がより困難となるため，新たな対策が必要となる。この対策として，「組込みデバッグ機能」の高度化と高性能化に主眼を置いた。組込みデバッグ機能とは，配列添字の間違い，引数の間違い，初期化漏れ，不当メモリ解放などを自動的に検出する機能である。今回，新たに高速な検出モードを開発し，別稿[7]で述べたとおり実アプリケーションにおいて数十倍の性能改善を達成した。そのため，より多くのアプリケーションに対して実運用においても適用できるようになった。

計算結果間違いなど，ソフトウェアロジックに起因するバグは，組込みデバッグ機能では検出されない場合が多い。このようなバグに対処するためには，アプリケーション側でのチェックを強化する必要があり，従来のいわゆる「printfデバッグ」手法（アプリケーションプログラマが実行論理に基づいて計算結果の検証のために中間値をprintf関数を使って表示するデバッグ手法。WRITE文デバッグとも呼ばれる）が有効である。アプリケーションが大規模化し，並列度も1000を超える大きさになった場合，デバッガを会話的に利用したデバッグが有効になるケースは限られており，多くの場合はアプリケーション側に検証コードを埋め込む手法を採用しなければならない。そのため，今回，printfデバッグを容易にするために，「フック機能」を新たに用意した。フック機能は，サブルーチンの入口/出口，スレッド並列処理の開始/終了，MPI関数の呼出しなどのタイミングで，言語処理系（コンパイラやRTS）が自動的に特定のユーザサブルーチンを呼び出す仕組みである。フック箇所でアプリケーションの検証コードを実行することで，より高精度なprintfデバッグを，最小限のプログラムコードの書換えで実現することができる。

● プロセス並列処理のデバッグ

プロセス並列処理固有の不具合と切り分けられた場合は，MPIを使ったプロセス並列化の過程のバグの可能性が高い。このデバッグのために，MPIライブラリに新たに「MPIプログラム実行時の動的デバッグ」機能を盛り込んだ。

この動的デバッグ機能では，デッドロック検出

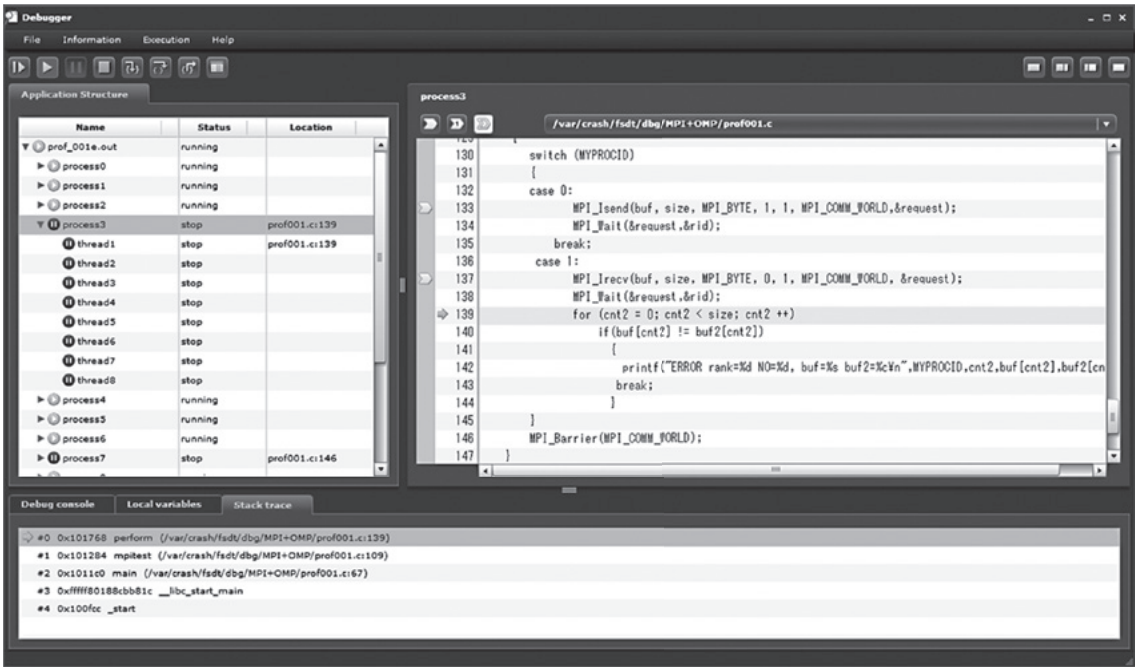

図-6　会話型デバッガによるMPIアプリケーションのデバッグ
左上：並列プロセスおよびスレッドごとの走行状態(停止中のものは停止位置)
右上：ソースコード上でのブレークポイントの位置や現在の停止位置
下：　スタックトレース

（通信待ちの打切り），MPI通信バッファの書込み破壊の監視，引数チェックを行っており，MPIライブラリの使用法に間違いがある場合は，MPIライブラリ側が自動的に検出する。

ほかに，MPI関数の呼出しをトレースするために，前述のprintfデバッグをMPI処理に対して高度化する手段として「MPI関数のフック」を開発した。これは，MPI関数の呼出し時に特定のユーザサブルーチンを呼び出す仕組みであり，ソースプログラムの行番号情報なども取得できるため，高度な検証コードを作成することが可能となる。さらにイベントログ機能である「VampirTrace」も用意されており，MPIライブラリの使用履歴がOTF[8]フォーマットで出力されるため，トレースログの確認による検証が可能である。

む　す　び

本稿では，スーパーコンピュータ「京」に向けたアプリケーション開発支援ツールを紹介した。冒頭でも述べたとおり，本開発支援ツールは，利用者におけるアプリケーションの高性能化および検証の作業手順を定義し，開発支援ツールのあるべき姿を検討することから開始した。結果として本開発支援ツールは，プロファイラや会話型デバッガのGUI表示がもたらす利用者にとっての分かりやすさと，最先端ハードウェアや大規模性に起因した様々な問題にアプローチする術（すべ）の両方を備えることで，当初の目的を達したと考えている。今後，「京」では，数々の世界最先端のグランドチャレンジアプリケーションが走行される予定であるが，それらの更なる高性能化や検証に貢献できるものと自負している。

参考文献

（1） The DWARF Debugging Standard.
http://dwarfstd.org/

（2） Performance Application Programming Interface（PAPI）.
http://icl.cs.utk.edu/papi/

（3） VampirTrace.
http://www.tu-dresden.de/zih/vampirtrace

（4） Adobe AIR.
http://www.adobe.com/jp/products/air.html

（5） SPARC64 VIIIfx Extensions 日本語版（2010年4月26日）.
http://img.jp.fujitsu.com/downloads/jp/jhpc/sparc64viiifx-extensionsj.pdf

（6） NAS Paralel Benchmarks.
http://www.nas.nasa.gov/publications/npb.html

（7） 瀧 康太郎ほか：スーパーコンピュータ「京」の能力を引き出すコンパイラ技術．*FUJITSU*，Vol.63，No.3，p.293-298（2012）.

（8） Open Trace Format（OTF）.
http://www.tu-dresden.de/zih/otf/

著者紹介

井田圭一（いだ　けいいち）
ミドルウェア事業本部アプリケーションマネジメント・ミドルウェア事業部 所属
現在，アプリケーション開発支援ツールの開発に従事。

井上俊介（いのうえ　しゅんすけ）
独立行政法人理化学研究所次世代スーパーコンピュータ開発実施本部 所属
現在，アプリケーションの高速化に従事。

大野康行（おおの　やすゆき）
ミドルウェア事業本部アプリケーションマネジメント・ミドルウェア事業部 所属
現在，アプリケーション開発支援ツールの開発に従事。

南　一生（みなみ　かずお）
独立行政法人理化学研究所次世代スーパーコンピュータ開発実施本部開発グループ 所属
現在，理研においてアプリケーションの高並列化・高性能化に従事。