Vol.2012-HPC-135 No.45
2012/8/3

京速コンピュータ「京」における性能分析ツール Scalascaを用いた性能分析

中村 朋健[1]　佐藤 三久[1,2]

概要：Scalasca は大規模システム上で並列アプリケーションの実行動作を分析するために設計された性能分析ツールセットである．複数のプロセッサから生成される各トレースファイルを分析する Scalasca の機能は，大規模システム上で動作するアプリケーションの分析に有用である．Scalasca を京速コンピュータ「京」に移植し，京速コンピュータ「京」上で NAS Parallel Benchmarks の中の Conjugate Gradient (CG) を実行し Scalasca で性能分析した．計算ノードの 3 次元のジョブ形状，32x32x2 と 16x16x8，で実行したとき，point-to-point 通信の待ち時間が大きく異なることを Scalasca のグラフィックツールで確認した．Scalasca が京速コンピュータ「京」のような大規模システム上で動作するアプリケーションで非効率の通信特性を測定でき，アプリケーションのチューニングに有用なツールであることを示した．

1. はじめに

メッセージパッシングアプリケーションの point-to-point 通信では，受信プロセスの進捗が送信プロセスの進捗に依存するために待ち状態になることがある．また，集団同期も完了するためには，各参加プロセスがある一定のポイントに到達していることが求められる点で，類似している．この通信や同期により，大部分の通信や同期時間が，待ち状態となることがある [3]．特に，通信の多いアプリケーションで大規模なプロセッサでの実行では，このような待ち状態が多く発生し，目標とする性能を引き出すための大きな問題となることがある．

本稿では，京速コンピュータ「京」上の性能分析ツール Scalasca について報告する．Scalasca はテネシー大学と Jülich Supercomputing Centre が開発した性能分析ツールである．京速コンピュータ「京」は，文部科学省が推進する「革新的ハイパフォーマンス・コンピューティング・インフラ（HPCI）の構築」計画のもと，2012 年 6 月の完成を目指して，理研と富士通が共同開発中のスーパーコンピュータである [18]．以下で，京速コンピュータ「京」を京コンピュータと呼ぶこととする．

性能分析ツールセットである Scalasca は，以下のアイデアで特徴的パターンのイベントトレースを自動的に検索（自動トレース分析）する．

- 非効率的な動作パターンの検索
- 実行したときの振る舞いの分類
- トレース情報の重要度合いの定量化

自動トレース分析により，手動よりも高速に分析でき，アプリケーション全体のイベントトレースを対象に分析したことが保証される．Scalasca は，PEARL(Parallel Access to Trace Data)[4] によってすべてのプロセッサまたはスレッドからそれぞれ取得したトレース分析ファイル（ローカルトレースファイル）を操作でき，大規模なトレース情報から効率良く特徴的なイベントパターンを抽出可能である．

Scalasca のトレース分析は，Late Sender パターンのような非効率な point-to-point 通信の特性を正確に測定できるため，アプリケーションのチューニングに役立つ．また，PEARL による複数のプロセッサから生成される各トレースファイルを分析する機能は，大規模システムでの実行時の分析を可能にし，京コンピュータのような大規模なシステム上で動かすアプリケーションの性能を分析するには有用である．

本稿では，京コンピュータ上で Late Sender パターンによる待ち時間をベンチマークプログラム (NAS Parallel Benchmarks の CG) から検出し，実行する計算ノードのジョブ形状によってこの待ち時間が異なることを確かめる．このことにより Scalasca が大規模システムでアプリケーションを分析しチューニングするよに有用であることを示す．

本稿は以下 2 章で Scalasca の機能を説明する．次に，3 章で Scalasca の京コンピュータへの移植方法について説明す

[1] 理化学研究所 計算科学研究機構
Advanced Institute for Computational Science, RIKEN
[2] 筑波大学 計算科学研究センター
Center for Computational Sciences, University of Tsukuba

る．4 章で性能分析実験を示し，6 章で本稿を統括する．なお，2 章の Scalasca の機能説明については文献 [3],[14],[15] を引用した．

2. 性能分析ツール SCALASCA

Scalasca は KOJAK[10] の後継としてテネシー大学と Jülich Supercomputing Centre が開発した性能分析のためのオープンソースのツールセットである．Scalasca は幅広い HPC プラットフォームで C，C++，Fortran で書かれたスケーラブルな HPC アプリケーションでよく使われる MPI, OpenMP，そしてそれらのハイブリッドプログラミングの測定や分析を支援する [13]．Scalasca は IBM Blue Gene や Cray XT などの大規模システム上で動作するように特別設計された性能分析ツールであるが，中小規模の HPC プラットフォームにも適している．

Scalasca には不規則な分散作業負荷の結果として生じる待ち状態を識別する機能がある．特に，大規模システム上で通信を多用するアプリケーションをスケールさせようとすると，待ち状態が多く発生し良い性能を得ることは難しい．

KOJAK に比べて Scalasca は大規模なプロセッサ上で待ち状態を検出可能であるトレース分析手法である。図 1 に Scalasca の基本的な分析手順を示す．性能データを集める前に，ターゲットアプリケーションは計装用のコードを埋め込む必要がある．並列計算機で計装用コードを埋め込んだコードを実行すると，計測フェーズで個々の関数コールパスに関するプロファイルデータが生成される．その後プロファイルやタイムライン可視化を使用できる．要約フェーズの出力（要約レポート）は性能挙動やハードウェアカウンタによるプロセッサ，または，スレッド単位でのローカルな測定の概略を知ることに役立つ．

トレース情報は大規模データになる傾向があるため，要約レポートの値を見ることを推奨する．トレースすることが可能なとき，各プロセスはローカルなイベント記録を含むトレースファイルを生成する．プログラムの終了後，Scalasca はトレースファイルを主記憶に読み込み，並列にトレースファイルを分析する．この分析フェーズでは，Scalasca は待ち状態や性能特性を示す特徴的パターンを探し，検出されたインスタンスをカテゴリ別に分類し，重要度を定量化する．この出力は要約レポートの構造に似たパターン分析レポートであるが，より高度に通信や同期の非効率な部分を分析できる．

要約レポートとパターンのレポートはすべての関数コールパスとシステムリソースに関する性能測定値を含む．関数コールパスとシステムリソースはグラフィカルなレポートブラウザでインタラクティブに調査するときに使える．グラフィカルな表示方法として Scalasca は CUBE GUI を提供する．CUBE は個別にインストールできる．CUBE に似た TAU[8] に含まれる ParaProf などのサードパーティのプロファイル可視化ツールを用いて Scalasca の分析レポートを可視化することもできる．自動分析の代替手段として，まとめたローカルトレース情報をサードパーティのトレースブラウザで可視化させ，調査に利用することが考えられる．Vampir は Scalasca のトレースディレクトリを扱うことができる．

2.1 計装と計測

2.1.1 アプリケーションの実行可能ファイルの準備

計測し分析するための対象アプリケーションの実行可能ファイルは対象アプリケーションのコンパイル時に計装用コードを埋め込む必要がある．すべてのシステムで計装の機構は手動または自動で提供され，ほとんどのシステムで自動的に計測用コードから測定ライブラリに伝えることが可能である．Scalasca はコンパイルコマンドとリンクコマンドにオプションを指定することでソースファイルの計装が可能である．アプリケーションを計測ライブラリに単純にリンクすることで MPI に関連したイベントが適切に PMPI (Profiling MPI) プロファイルインタフェースによってキャプチャされる．OpenMP におけるソースの前処理は，並列実行域などで POMP (Profiling OpenMP)[6] プロファイルインタフェースに基づいて，計装ディレクティブや計装 pragma を自動的に挿入される．ほとんどのコンパイラはすべての関数やルーチンの入口と出口に計装用のフックを加えことができる．プログラマはソースコードのループや関数など重要な領域に手動で計測コードを挿入することができる．この計測コードは pragma やマクロであり，ユーザからの計測指示がない場合は無視される．

2.1.2 計測と分析設定

Scalasca 計測システム [12] は環境変数または設定ファイルを通じて設定することができる．計測の初期化中に，すべての測定データと分析データを格納するための一つの実験結果アーカイブディレクトリが作成される．イベントトレース情報が集まったとき，実験結果アーカイブディレクトリに測定データや分析データを格納する．

測定値は集められ，分析される．ユーザからトレースする指示がでたとき，測定に使われるプロセッサ数と同数の並列トレースアナライザは自動的に設定され実行する．Scalasca で測定するときアプリケーション実行コマンドラインの先頭にコマンドを指定する．

大規模システムでトレースするときの I/O のバンド幅やストレージを考慮して，一般に利用可能なメモリサイズを超えることがないようにアプリケーションプロセスごとのトレースデータの量を制限することが望ましい．特に，アプリケーションが動いている間，非同期にデスクにトレースデータを追い出す処理によって起こることが問題である．これは選択的なトレーシングによって問題解決する．例え

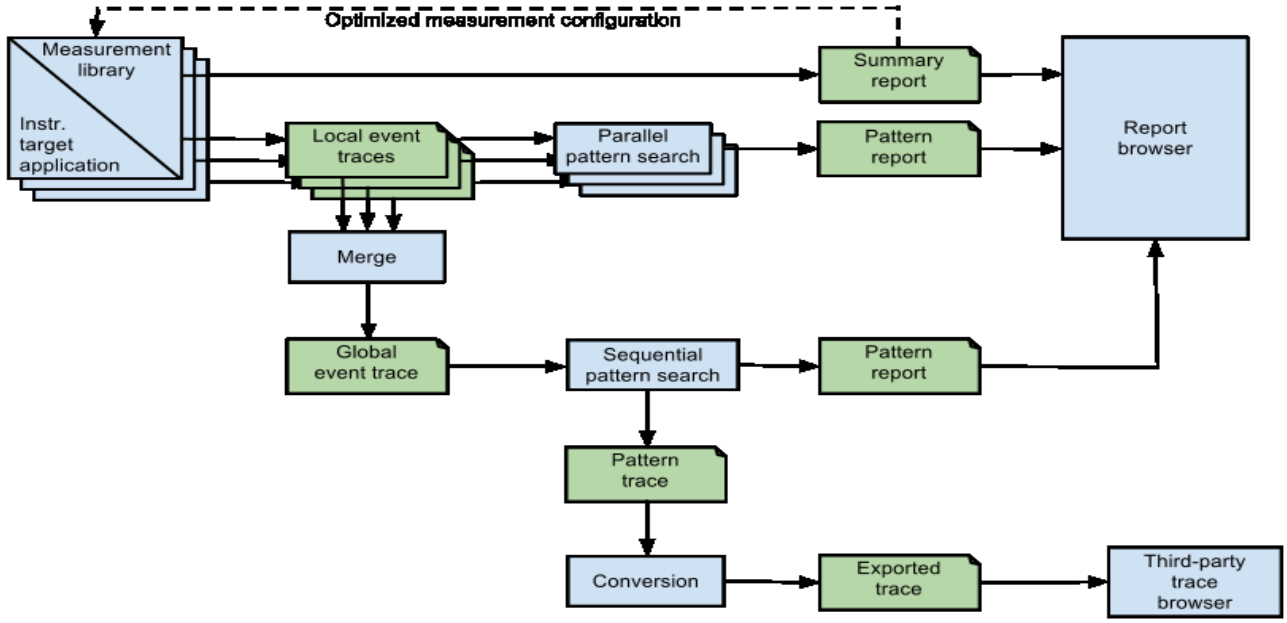

図 1 Scalasa の性能分析ワークフロー

ば，特定の調査したい区間だけイベントを記録したり，測定する間のタイムステップ数を制限するなどが考えられる．

計装する関数を頻繁に実行したとき測定で悪影響がある．そのような関数の測定のオーバヘッドは未計装の関数の実行時間よりも長いことがある．特に記録したイベントの総数に比例するイベントトレースではサイズが重要である．Scalasca は問題のない関数を計測から排除するためにフィルタ機構を提供している．トレース情報の収集前に，一般に計装は前の要約の実行から得られる visit-count 分析に基づいて最適化することが重要である．

2.2 イベント要約と分析

Scalasca の性能分析能力の本質はイベント要約である．ここで，イベント要約とはすべての実行から個々のイベントと関連する測定値を集めるようなイベントストリームから実行動作のシンプルな表現への変換することである．Scalasca はイベントストリーム分析に２つの選択肢がある．

- 即時ランタイム要約
- イベントトレースの事後分析

ランタイム要約の利点はトレースバッファとファイルにイベントを格納する必要がないことである．イベントトレースの事後分析はランタイム要約では検出できない各種の待ち状態を識別するため，複数プロセス間のタイムスタンプの比較が可能である．

2.2.1 ランタイム要約

多くの実行性能測定値は，事前に分析したイベントでの計測データを蓄積するコストを避けるため，計測中に統計情報を蓄積することで効率よく計算している．例えば，ソースコードのある範囲（ルーチンやループなど）の経過時間やハードウェアカウンタをすぐに測定し，差分を蓄積する．一方で，トレース情報の保管要求はイベント数に比例して増加する（測定の継続時間に依存する）．スレッドごとのコールパスプロファイルに関する要約統計情報は少ない一定サイズである（スレッド数や実行したコールパスに依存する）．Scalasca は測定値と各スレッドに関するコールパスを関連付け，測定中に測定値を更新する．

2.2.2 事後のトレース分析

メッセージパッシングアプリケーションで，point-to-point 通信が多く，受信プロセスの進捗が送信プロセスの進捗に依存することになる．集団同期が完了するためには，各参加プロセスがある一定のポイントに到達していることが求められる点で，上記の point-to-point 通信に類似している．これらの通信や同期により，大部分の通信や同期時間が，待ち状態となることがある．特に，通信の多いアプリケーションで大きなプロセス数で実行する場合，このような待ち状態が重大な性能問題となる可能性がある．Scalasca は，特徴的パターンのイベントトレースを自動的に探すことによって，アプリケーションを実行したプロセッサの位置，実行時のアプリケーションの振る舞い分類，およびトレース情報の重要度の定量化をする方法を提供する．Scalasca がサポートするこのようなパターンのリストはオンラインで入手可能である [16]．

特徴的なパターンのイベントトレースを自動的に探すことをスケーラブルに遂行するために，対象のシステム上の分散型メモリ，および，並列処理能力のシステムを利用する．一つのグローバルトレースファイルを連続して分析する代わりに，Scalasca はターゲットアプリケーション自体を実行するために可能な限り多くの CPU 上で通信を実行

することにより，複数のローカルプロセストレースファイルを並列に分析する．この方法は KOJAK も同様である．検索プロセス中は、関連する各プログラムフェーズ，および，システムリソースに対する重要性に応じて，パターンインスタンスが分類そして定量化される．トレース処理能力はアプリケーションプロセス数に比例して増すため，このようなパターン検索は、以前から解決困難であった．

アプリケーションプロセス数が増加してもトレース分析プロセスの有効性を維持するために，トレースデータの並列アクセス手法 PEARL(Parallel Access to Trace Data)[4] を取り入れている．PEARL では各イベントや対応するイベントを識別するための要約にランダムにアクセスする機能があり，またパターン検索をする上で重要な機能である．トレースアクセスライブラリの主な利用モデルは，ターゲットアプリケーションのすべてのプロセスに関して分析プロセスがそのトレースデータを担当してつくることである．分析プロセス間でのデータ交換は MPI 通信を通じて行われる．

2.3 CUBE

CUBE(CUBE Uniform Behavioral Encoding) は MPI や OpenMP を用いた並列プログラムに対して様々な性能データの表示に適した提示ツールである．CUBE は性能データをインタラクティブに調査できる．以下の 2 つの方法でスケーラビリティを確保した．

- 個々の次元の階層的な分解
- 異なる次元の集約

CUBE は CUBE パフォーマンススペースと呼ばれるプログラム挙動のハイレベルデータモデルとして設計された．図 3 に示すような CUBE パフォーマンススペースはメトリック欄，プログラム欄，そしてシステム欄の 3 つの欄から構成される．メトリック欄は通信時間やキャッシュミスなどの測定した項目で構成される．プログラム欄はプログラムのコールツリーから構成される．システム欄は並列実行している項目を含み，並列プログラミングモデルに応じたプロセスやスレッドとして見ることができる．パフォーマンススペースの各欄は実際に測定された数値を対応付けて見ることができる．このマッピングはパフォーマンススペースの severity と呼ばれている．

パフォーマンススペースの各欄は階層的にまとめられている．まず，メトリック欄は下位レベルの測定値はその親の部分集合であり，階層構造で表現される．次に，プログラム欄はコールツリーの階層構造で表現される．最後に，システム欄はマシン，SMP ノード，プロセス，スレッドの複数の下位層構造で表現される．

3. Scalasca の京への移植

Scalasca はプログラムのプロファイルデータを取るために，計測対象プログラムのコードをフックし，コンパイルし計測用バイナリファイルを作成する．コードをフックする箇所は以下の 2 箇所である．

- ユーザ定義関数の出入口
- MPI 関数

ユーザ定義関数の出入口を掴まえるために，__cyg_profile_func_enter と __cyg_profile_func_exit を利用した．これらと GNU Compiler Collection のコンパイルオプション-finstrument-functions を使用することで，ユーザ定義関数が呼び出されたときと関数から復帰するときにトレース用の関数を呼び出す命令が付加される．京コンピュータでユーザ定義関数をフックするため，つまり __cyg_profile_func_enter と __cyg_profile_func_exit を利用するために-Ntl_vtrc オプションを付加し，MPI 関数をフックのために-Ntl_nortrc オプションを付加してビルドした．

トレース情報には実行環境の位置を必要とする．京コンピュータで 3 次元の位置情報を返すために，京コンピュータの拡張インタフェースであるランク問い合わせインタフェースを用いてトポロジの構造を求めた．MPI を用いて OpenMP を用いないとき，以下の 2 つの表示されるようにした．

- 実行時に指定した計算ノードの形状をそのまま表示
- 計算ノードをランクの順に 1 次元目に並べて表示

MPI と OpenMP のハイブリッド並列プログラムのとき，1 次元目に OpenMP によって並列化されたスレッドを並べ，2 次元目に MPI によって並列化された計算ノードを並べて表示した．MPI と OpenMP のハイブリッド並列プログラムのとき，実行時に指定した計算ノードの形状は表示できない．

4. 実験

大規模システム上で Late Sender パターンの point-to-point 通信の待ち時間が性能のボトルネックになるようなアプリケーションで，Scalasca がその通信の待ち時間を測定できることを確かめる．また，Scalasca が京コンピュータ上で動かすアプリケーションのチューニングに有用であることを確かめる．

4.1 実験環境

実験に使用した京コンピュータは，82,944 個の計算ノードを，「Tofu (Torus fusion)」とよばれる 6 次元メッシュ／トーラス構造のネットワークで相互に接続した並列計算機システムである．システム全体のピーク性能は 10.6PFLOPS に達する．各計算ノードは，1 個の CPU (SPARC64^{TM}VIIIfx)，1 個のネットワーク用 LSI(ICC: InterConnect Controller) および 16GB のメモリを持つ．SPARC64^{TM}VIIIfx の諸元を表 1 に示す．通常，Tofu インターコネクトは，論理的な 3 次元トーラス構造として利用される．その場合の帯域は

表 1 SPARC64^{TM}VIIIfx の諸元

演算性能（ピーク）	128 GFLOPS (16GFLOPS x 8 cores)
コア数	8
クロック周波数	2.0 GHz
浮動小数点演算器	乗加算ユニット x 4 (2 SIMD) 除算器 x 2
レジスタ数	浮動小数点レジスタ (64bit): 256 汎用レジスタ (64bit): 188
キャッシュ	L1$: 32 KB (2way) L1D$: 32 KB (2way) L2$: Shared 6 MB (12way)
メモリ帯域	64 GB/s (0.5 B/F)

3 次元の正負各方向にそれぞれ 5GB/s（双方向）である．

京コンピュータで NAS Parallel Benchmark 3.3.1 (NPB) を実行し，京コンピュータ向けに移植した Scalasca で NPB のコードを分析した．NPB は NASA Ames Research Center で開発された並列計算機のためのベンチマークであり，5 つのカーネルコード（EP, MG, CG, FT, IS）と 3 つのアプリケーションコード（BT, SP, LU）から成る [1]．実験では NPB の中で処理時間における通信時間が多いアプリケーションコード Conjugate Gradient (CG) を用いた．CG は正値対象な疎行列の最小固有値を共益勾配法により求めるベンチマークである．CG で処理コストの高いサブルーチン conj_grad におけるデータ転送する一部のコードを図 2 に示す．

```
do i = 1, l2npcols
    if (timeron) call timer_start(t_rcomm)
    call mpi_irecv( rho,
>                   1,
>                   dp_type,
>                   reduce_exch_proc(i),
>                   i,
>                   mpi_comm_world,
>                   request,
>                   ierr)
    call mpi_send( sum,
>                   1,
>                   dp_type,
>                   reduce_exch_proc(i),
>                   i,
>                   mpi_comm_world,
>                   ierr)
    call mpi_wait( request, status, ierr)
    sum = sum + rho
enddo
```

図 2 NAS Parallel Benchmarks のカーネルコード CG におけるサブルーチン conj_grad の一部

図 2 において，mpi_irecv はノンブロッキング通信であるためこれ自身はすぐに終わる．ここでコストが高いのはブロッキング通信の mpi_send や mpi_wait である．

4.2 実験結果

実行時のパラメータは以下の通りである．

計算ノード数: 2,048

計算ノードのジョブ形状: 32x32x2, 16x16x8

CG のクラス: D

ここで，ジョブ形状 32x32x2 を長形状，16x16x8 を立形状と呼ぶこととする．

長形状で実行したときのトレースデータを CUBE で表示した（図 3）．CG でデータ転送の多く，処理時間を長く費やす部分を表示させるために，メトリック欄で処理時間に関する MPI の point-to-point 通信を選択した．Point-to-point 通信に費やした時間は約 84,500 秒であり，全体の 8.46%であることがわかる．ただし，ここでの時間は全ノードで費やした時間の総計である．トレースデータの計測によるオーバヘッドは約 50%であることがわかる．このオーバヘッドを除いた場合の point-to-point 通信は全体の約 17%であった．

プログラム欄ではサブルーチン conj_grad を計測するために，コールツリータブを選択しサブルーチン conj_grad の MPI_Wait を選択した．サブルーチン conj_grad における MPI_Wait は赤色に近く，point-to-point 通信の大部分の費やしていることが視覚的に容易にわかる．ここでの通信時間は約 68,400 秒であり，point-to-point 通信の 81.32%を費やしていた．

システム欄では 3 次元で実行した結果を視覚的にわかりやすくするためトポロジ 0 タブを選択した．性能スペースの下の尺度（低コストが青，高コストが赤）で各計算ノードの負荷率がわかるように最小値を 0 秒，最大値を 50 秒にした．つまり，まったく負荷がかかっていないノードは青色に，20 秒 (40%) 程度費やしたノードは黄色に，50 秒程度費やしたノードは赤色に表示される．

次に，長形状で実行したときのトレースデータを CUBE で表示した（図 4）．実行可能プログラムは長形状での実行でつかったものと同じである．図 3 と同様に，メトリック欄では point-to-point 通信に費やした時間を選択し，プログラム欄ではサブルーチン conj_grad の MPI_Wait を選択し，システム欄ではジョブ形状が 3 次元で視覚的にわかりやすいトポロジ 0 タブを選択した．

立形状で実行したとき，point-to-point 通信に費やす通信時間は約 58,800 秒であり，長形状での通信時間より約 26,000 秒高速であった．図 3 と図 4 のプログラム欄をみるとサブルーチン conj_grad の MPI_Wait だけで 20,000 秒以上立形状が高速であることがわかる．1 ノードあたり約 10 秒の違いがあった．システム欄を比べると，立形状の実行は長形状の実行に比べて全体的にサブルーチン conj_grad の MPI_Wait の通信コストが少ないことが視覚的に容易にわかった．アプルケーションの全体の実行性能は長形状が 95,459[Mops] ，立形状が 133,585[Mops] であり，立形状で

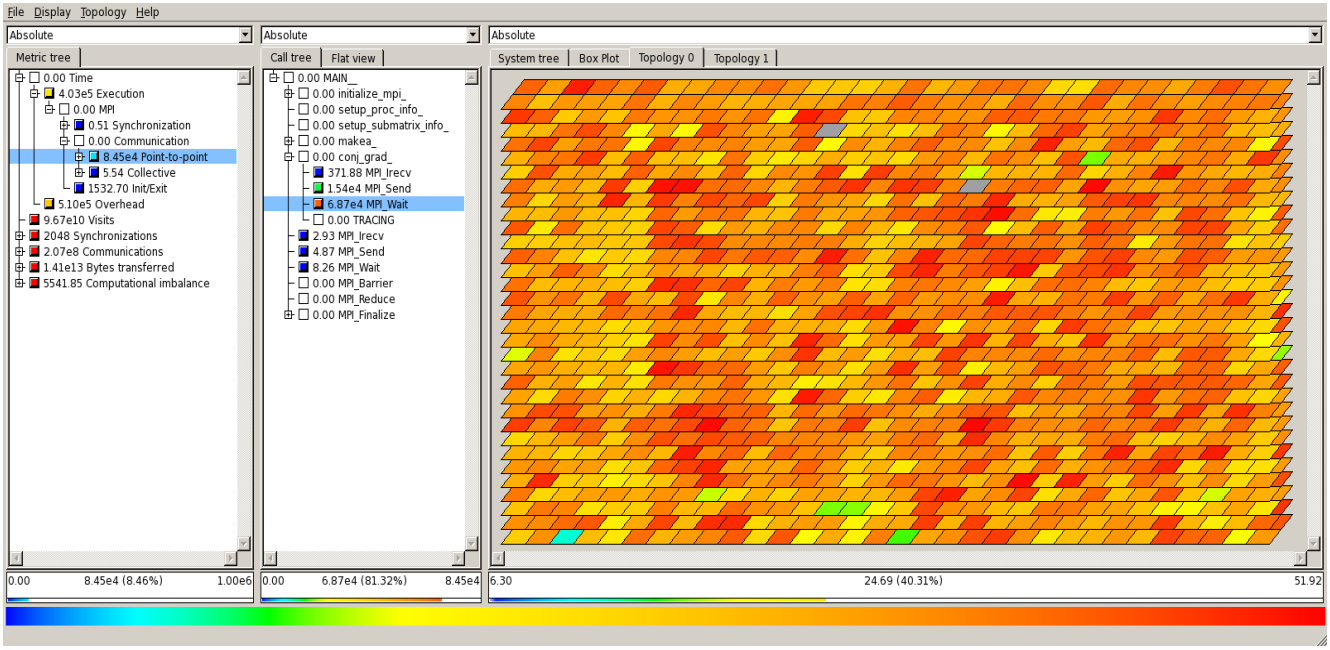

図 3 NPB の CG を 2,048 並列の 32x32x2 の 3 次元で実行したときのプロファイル

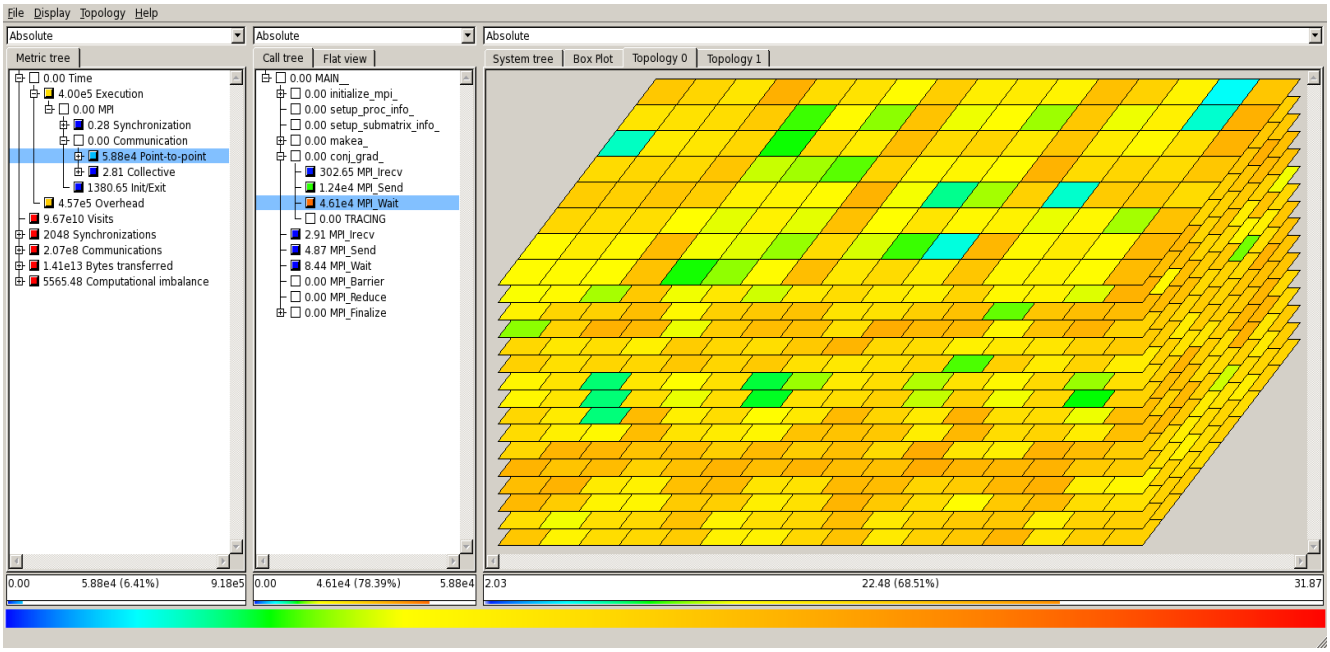

図 4 NPB の CG を 2,048 並列の 16x16x8 の 3 次元で実行したときのプロファイル

の実行が通信性能によって全体の実行性能に大きく影響していることがわかった．

Scalasca の特徴である複数のローカルファイルから待ち状態の時間を測定し，その結果を CUBE を用いてグラフィック表示させた．この表示により，計算ノードのジョブ形状によって通信による待ち時間が大きく異なり，京コンピュータ上で Scalasca がチューニングに役立つことがわかった．

5. 関連研究

並列アプリケーションの開発者は多くの性能分析ツールから使いやすく，かつ，使用したい機能が備わったツールを選んで使用する．性能分析ツールには Scalasca[16], TAU[8], Paraver[5], Vampir[7], Periscope[2] そして京コンピュータのアプリケーション開発支援ツールの基本プロファイラ（富士通プロファイラ）[17] などがある．これらの性能分析ツールは機能が機能が重複していることもあるが，それぞれに特色がある．

TAU は Scalasca に特に類似しており，ランタイムにトレースや要約し，測定後に性能分析する．TAU は性能データに対して，データ管理やデータマイニングする機能的特徴を持つ．Scalasca の性能要約やトレース分析データは

TAU のプロファイル可視化機能で活用可能である．

Paraver や Vampir（VampirServer, VampirTrace）はトレース情報をインタラクティブに分析する可視化機能を持つ．VampirServer は Scalasca のように並列にトレース情報へアクセスする機構があるためスケーラビリティが高い．対照的に Paraver はフィルタを用いてトレース情報を縮小する機構がある．このフィルタは不要な特徴や要約を取り除くものである．

Periscope はスケーラブルな自動性能分析ツールである．分析エージェントによって専門知識をまとめた特性に基づいて性能ボトルネックになる部分を自動的に検出できる．

富士通プロファイラは CPU 演算処理，スレッド並列処理，プロセス並列処理の観点から性能分析可能である．また，トレース情報はサンプリングによる取得であり，オーバヘッドは少なくかつ一定である．

6. おわりに

本稿では，性能分析ツールセット Scalasca を京速コンピュータ「京」(京コンピュータ) に移植し，京コンピュータ上で NAS Parallel Benchmarks の CG を実行し Scalasca で性能分析した．計算ノードの 3 次元のジョブ形状，32x32x2 と 16x16x8，によって point-to-point 通信の待ち時間が大きく異なることを Scalasca のグラフィックツールで確認した．この実験により，京コンピュータで実行した通信の待ち時間が性能のボトルネックになるアプリケーションに関して，Scalasca がその通信の待ち時間を測定でき，かつ，有用であることを確かめた．

今後，さらに大きい規模のプログラムを実行させトレース分析できるか確認する．また，さらに Scalasca が大規模なトレースデータを操作できるように PEARL を改善することや性能ボトルネックになる部分の検出の対話的な支援を考える．

謝辞 本研究の成果（の一部）は，理化学研究所が実施している京速コンピュータ「京」の試験利用によるものです．

参考文献

[1] Bailey, D., Barszcz, E., Dagum, L., and Simon, H.: THE NAS PARALLEL BENCHMARKS, *Technical Report NAS-94-007*, Nasa Ames Research Center (1994).

[2] Fürlinger, K. and Gerndt, M.: Distributed application monitoring for clustered SMP architectures, In. *Proc. European Coference on Parallel Computing (Euro-Par)*, pp. 127–134 (2003).

[3] Geimer, M., Wolf, F., Wylie, B., Ábrahám, E., Becker, D. and Mohr, B.: The SCALASCA performance toolset architecture, *Concurrency and Computation: Practice and Experience, 22(6)*, pp. 702–719 (2010).

[4] Geimer, M., Wolf, F., Knüpfer, A., Mohr, B., and Wylie, B.: A parallel trace-data interface for scalable performance analysis. In. *Proc. Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA, Umea, Sweden)*, pp. 398–408 (2006).

[5] Labarta, J., Gimenez, J., Martinez, E., Gonzalez, P., Servat, H., Llort, G. and Aguilar, X.: Scalability of visualization and tracing tools, In. *Proc. Parallel Computing (ParCo)*, pp. 869–876 (2005).

[6] Mohr, B., Mallony, A., Hoppe, H, Schlimbach, F., Haab, G. and Shah, S.: A performance monitoring interface for OpenMP, In. *Proc. of the fourth European Workshop on OpenMP - EWOMP'02* (2002).

[7] Nagel, W., Weber, M., Hoppe, H. and Solchenbach, K.: VAMPIR:Visualization and analysis of MPI resources, *Supercomputer*, pp. 69–80 (1996).

[8] Shende, S. and Malony, A.: The TAU parallel performance system, *International Journal of High Performance Computing Applications*, pp. 287–331 (2006).

[9] Wolf, F.: EARL — API documentation, *Technical Report ICL-UT-04-03, University of Ten- nessee* (2004)

[10] Wolf, F. and Mohr, B.: Automatic performance analysis of hybrid MPI/OpenMP applications. *Journal of Systems Architecture 49*, pp. 421–439 (2003).

[11] Wolf, F., Mohr, B., Dongarra, J., Moore, S.: Efficient pattern search in large traces through successive refinement. In. *Euro-Par 2004 Parallel Processing. Volume 3149 of LNCS*, pp. 47–54 (2004).

[12] Wylie, B., Wolf, F., Mohr, B. and Geimer M.: Integrated runtime measurement summarization and selective event tracing for scalable parallel execution performance diagnosis, In. *Proc. Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA)*, pp. 460–469 (2006).

[13] Wolf, F., Wylie, B., Ábrahám, E., Becker, D., Frings, W., Fürlinger, K., Geimer, M., Hermanns, M., Mohr, B., Moore, S., Pfeifer, M. and Szebenyi Z.: Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications, In. *Proc. 2nd HLRS Parallel Tools Workshop (Stuttgart, Germany)* (2008).

[14] The Scalasca Development Team: Scalasca 1.4 User Guide, `http://www.scalasca.org/software/documentation`, (2011).

[15] The Scalasca Development Team: CUBE 3.4 User Guide, `http://www.scalasca.org/software/documentation`, (2011).

[16] Jülich Supercomputing Centre. *SCALASCA* performance analysis toolset documentation. `http://www.scalasca.org/software/documentation`.

[17] 井田圭一, 大野康行, 井上俊介, 南一生: 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, *FUJITSU VOL.63, NO.3* , pp. 305–312 (2012).

[18] 高瀬亮，横川三津夫（編）: 情報処理, Vol. 53, No. 8, chapter 特集：京速コンピュータ「京（けい）」, 一般社団法人 情報処理学会 (2012).