情報処理学会論文誌　コンピューティングシステム　Vol.6　No.4　96–104 (Oct. 2013)

ソフトウェア進化のための自動性能追跡システム

平澤 将一[1,2,a)]　滝沢 寛之[1,2]　小林 広明[1]

受付日 2013年4月9日, 採録日 2013年8月8日

概要：本研究では，ソースコード修正にともなう性能変化を解析し，その性能変化を引き起こす要因となったソースコード修正を特定することができる自動性能追跡システムを提案する．自動性能追跡システムは統合開発環境（IDE）と連携して動作し，複数の計算システムにおける性能可搬性を維持しつつ高性能計算アプリケーションの進化を支援する．ソースコードがバージョン管理システムのリポジトリに格納された際，実行対象とするすべての計算システム上で実行性能を評価することにより，自動性能追跡システムは性能可搬性を低下させるソースコード修正をアプリケーション開発者に提示する．さらに，新しい計算システムが対象として追加された場合，自動性能追跡システムはリポジトリからアプリケーションの複数バージョンのコードを抽出し，それらを新しい計算システム上で実行する．この結果，アプリケーション開発者はソースコード変更が複数の計算システムにおける実行性能に与える影響を解析することができる．解析結果に基づいて，アプリケーション開発者はソースコード修正の再考や，アプリケーションの設計の改善を図ることができる．

キーワード：高性能計算，性能解析，ソースコードリポジトリ，総合開発環境

An Automatic Performance Tracking System for Software Evolution

Shoichi Hirasawa[1,2,a)]　Hiroyuki Takizawa[1,2]　Hiroaki Kobayashi[1]

Received: April 9, 2013, Accepted: August 8, 2013

Abstract: In this work, we propose an automatic performance tracking system for analyzing the changes in execution performance and finding the source code modifications that cause the performance changes. The proposed system works together with an Integrated Developing Environment (IDE) in order to interactively support evolving a high-performance computing application while maintaining its performance portability across multiple target computing platforms. By evaluating the performances of an application on every target platform whenever its codes are committed to the repository of a version control system, the proposed system helps application developers find the source code modifications that degrade the performance portability. Moreover, when a new target platform is given, the proposed system retrieves multiple versions of an application from the repository, and automatically executes them on the new platform. As a result, application developers analyze how the source code modifications in the past affect the performance on the new platform. Based on the analysis, the application developers can review the source code changes to improve the software design of the HPC application.

Keywords: high performance computing, performance analysis, source code repository, integrated development environment

1. はじめに

現在，高性能計算（HPC）分野では特定の計算システムを対象としてアプリケーションプログラムを最適化し，実行性能を高めるアプローチがとられている．しかし，特定の計算システム向けに最適化することにより，それ以外の計算システムにおける実行性能が低下する場合がある．すなわち，アプリケーションが様々な計算システムで高い性能を達成できる性質である**性能可搬性**が低下する場合がある．一方，計算システムの多様化にともない既存のアプリ

[1] 東北大学
Tohoku University, Sendai, Miyagi 980–8579, Japan
[2] JST CREST
JST CREST, Kawaguchi, Saitama 332–0012, Japan
[a)] hirasawa@sc.isc.tohoku.ac.jp

ケーションを新しい計算システムへ移植する作業がしばしば求められるため，高い性能可搬性を維持しつつアプリケーションを進化させることが望まれる．性能可搬性を維持するために複数の計算システムで実行性能を意識した開発を行うには，対象となるすべてのシステムで性能低下の要因を特定する必要があり，労力が大きい．

複数の計算システムに対応する性能可搬性を維持しつつアプリケーションを進化させていく開発プロセスのためには，アプリケーションが機能的に正しく動作することを確認したうえで，ソースコード中で性能可搬性を低下させる部分を自動的に特定し，プログラマに提示する機能を実現する必要がある．これまでに，アプリケーションが機能的に正しく動作することを確認するためにバグを自動的に検知する手法 [1], [2] や単体テスト [3] を行う方法が提案されているが，HPC アプリケーション開発において性能可搬性を目的とし自動的な支援を行う性能追跡システムに関する研究は，筆者らが知る限りでは行われていない．

複雑化した現代の計算システムにおいて，アプリケーションを実行することなくその実行性能を予測することは困難である．したがって，様々な計算システム上で実行性能を考慮した開発を進めるためには実行性能を定期的に計測する必要がある．実行性能を定期的に計測することで，性能可搬性を低下させる要因となる，一部の計算システムで実行性能が低下するソースコード変更を検知することができる．そのような機能を実現するためには，プログラマのソースコード変更を追跡して実行性能を記録する必要がある．

本論文では，上記のような HPC アプリケーションの開発支援機能実現のために求められる，自動性能追跡システムを設計・開発する．プログラマが指定したビルド方法と動作確認方法に対してアプリケーションが機能的に正しく同様に動作することを前提にして，本提案システムは性能可搬性低下の原因となるソースコードの変更箇所を特定する．複数の計算システムにまたがって動作可能となるよう開発されたアプリケーションは，新規に登場した計算システムへ移行しやすく，ひいては長期間にわたって使用可能であることが期待できる．

本論文の構成は以下のとおりである．2 章では HPC アプリケーションの性能可搬性維持と問題点を述べる．3 章では，提案する自動性能追跡システムについて議論する．4 章で自動性能追跡システムの評価を行い，5 章で関連研究を述べる．6 章では，本論文の結論と今後の課題について述べる．

2. HPC アプリケーションの性能可搬性維持

本研究では，HPC アプリケーション開発に用いられる計算機を以下のように用途ごとに分類して考える．**図 1** に，本研究で想定する計算機の構成を示す．

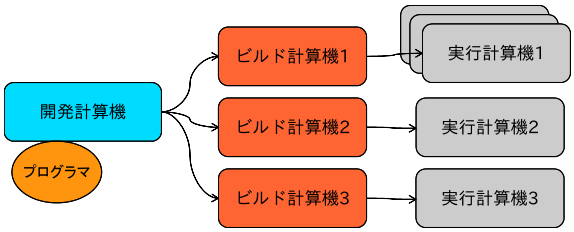

図 1　本研究で想定するアプリケーション開発の計算機構成
Fig. 1　Target computers for application development.

(1) 開発計算機：プログラマがソースコードを編集するために用いられる計算機
(2) ビルド計算機：コンパイルなど，アプリケーションのビルド処理を行う計算機
(3) 実行計算機：アプリケーションを実行する計算機

開発計算機とビルド計算機として同一の計算機を用いる場合，ビルド環境を開発計算機上に構築する必要がある．様々な実行計算機を想定してアプリケーションを開発する場合には，開発計算機上に実行計算機のためのクロスコンパイル環境を準備する必要がある．特に商用コンパイラの使用が必要となる場合にはライセンス数の制限が存在する場合があり，プログラマが操作する開発計算機にビルド環境を構築することは現実的ではない．本研究ではビルド計算機が複数存在する環境を想定し，それらを効果的に連携させることを考える．

2.1　一部の実行計算機での実行性能低下の要因解析

一般的な HPC アプリケーションは，長期間にわたって様々な機能やアルゴリズムが追加されて進化し続けることから，ソースコード量が大規模化する傾向がある．性能可搬性が低下した場合には，膨大なソースコードの中から性能可搬性の低下要因となっている箇所を探し出す必要があり，プログラマに多大な労力を要求する．このため，性能可搬性を維持しつつアプリケーションを進化させるために，一部の実行計算機における実行性能が低下するソースコード変更を特定してプログラマに通知する機能の実現が望まれる．

また，実行計算機もつねに進化していることを考えると，新しい実行計算機がアプリケーション進化の過程で追加されることも想定する必要がある．新規の実行計算機に対する性能可搬性低下要因を検知するためには，最新のソースコードだけではなく，過去に行われたソースコードの変更が新規の実行計算機における実行性能に与える影響も調査する必要がある．

2.2　実行性能計測

複雑化した現代の実行計算機では，アプリケーションの実行性能を静的に予測することは困難である．そのため性能解析に必要な実行時間を取得するためには実際にアプリ

ケーションを実行計算機で実行する必要がある．

また，HPC アプリケーションの実行性能はコンパイラの種類や最適化オプションに強く依存しており，ある実行計算機上で有用な実行性能を得るためには，その実行計算機が想定するビルド計算機でビルドする必要がある．現在の一般的な HPC アプリケーション開発においては，定期的に手動でビルドおよび実行することによって性能が計測されており，複数の実行計算機上での実行性能を得るまでに多数のコマンドを組み合わせた煩雑な作業が求められる．

2.3　ファイル同期

図 1 のように複数のビルド計算機を用いる場合には，ビルド計算機間でファイル同期やバージョン管理が求められる．現在，これらの作業は複数のツールを組み合わせて実現されているため，プログラマの労力が増大するばかりか，ミスの可能性も高くなる懸念がある．また，ソースコード編集作業を中断して確認作業を行っているため開発効率を低下させる要因にもなっている．

3.　自動性能追跡システム

本論文では，プログラマの開発計算機上で動作する開発用の統合開発環境と連携して動作する自動性能追跡システムを提案する．本提案システムでは，プログラマがソースコードのほかにビルド方法（ビルドコマンド）と動作確認方法（実行スクリプト）を指定することを前提にしている．対象とするアプリケーションは，あらかじめプログラマにより指定されたビルドコマンドでビルド可能であり，指定された実行スクリプトと入力データによって正常動作可能であると仮定する．**図 2** に提案する自動性能追跡システムと開発フレームワークを示す．Eclipse [4] に代表される統合開発環境に用意されている機能拡張のためのプラグイン機能を用いることで，本システムは統合開発環境と連携する．複数の実行計算機上での実行性能を自動的に計測，記録することにより，ソースコードの変遷とそれにともなう性能の変化を追跡する．これにより，特定の実行計算機に過度に依存した性能最適化により他の実行計算機での実行性能が低下した場合に，原因となるソースコードの変更個所を特定する．これにより性能可搬性の維持や新規計算プラットフォームへの対応を効果的に支援することができる．自動性能追跡システムは以下の機能によって 2 章で述べた課題に対応し，性能可搬性を維持しつつアプリケーションを進化させる作業を支援する．

(1) ソースコード変更にともなう実行性能の変化を自動追跡する機能
(2) 各実行計算機上でアプリケーションを自動実行する機能
(3) 指定されたすべてのビルド計算機とファイルを同期させる機能

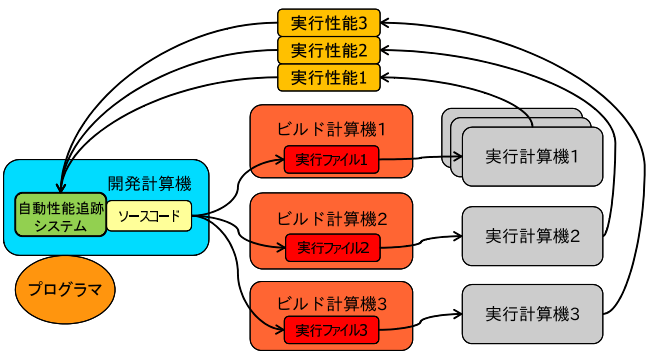

図 2　自動性能追跡システムによる開発フレームワーク
Fig. 2　Development framework of the performance tracking system.

3.1　ソースコード変更にともなう実行性能の変化を自動追跡する機能

開発対象として想定する実行計算機が追加された場合，それまでに開発されてきたアプリケーションの最新のソースコードが新しい実行計算機で高い実行性能を達成できるとは限らない．効果的な性能最適化手法は計算システムごとに異なり，ときとして相反する [5] ことから，最適化のための過去のソースコード変更が新実行計算機では性能を低下させる可能性がある．そこで，自動性能追跡システムでは，CVS [6] や Git [7] などに代表されるバージョン管理システムと連携して，過去に行ったソースコードの編集が新実行計算機上での実行性能に与える影響を調査する．その結果，ソースコードの変更にともなう実行性能の変化を自動追跡することが可能である．

本機能では，プログラマが実行計算機を追加した際に，バージョン管理システムで管理されたリポジトリから複数バージョンのソースコードを自動的に取得し，プログラマにより指定された実行スクリプトと入力データを用いて実行する．隣接する 2 バージョンの新実行計算機における実行性能を比較し，実行性能を低下させる変更点を特定する．変更点を特定することにより，プログラマは新実行計算機を含めた性能可搬性を改善するためのソースコード改変の方針を検討することが可能である．改変を要するソースコードの範囲を限定することによって，アプリケーションの性能可搬性の維持を支援することができる．

バージョン管理システムで管理されたリポジトリにおいて，開発が直線的に進むだけではなく，複数バージョンに枝分かれ（ブランチ）して進んでいく場合がある．ブランチの結果，共通の親バージョンを持つ複数の子バージョンがツリー状に生成される．このため，ブランチが発生した場合には，親バージョンとすべての子バージョンの性能比較によって実行性能の変化を追跡できる．分岐後に合流するマージについても，複数の親バージョンと単一の子バージョンを比較することで実行性能を追跡できる．

3.2　各実行計算機上でアプリケーションを自動実行する機能

自動性能追跡システムは，正常にビルド可能で機能的に正しく動作するバージョンを実行計算機で実行し，その実行性能を取得する一連の作業を自動化する機能を提供する．ソースコード編集作業を中断することなく対話的に実行結果をプログラマに提示するためには，ビルド処理と実行に要する時間が十分に短いことが求められる．この要件を満たすために，本提案システムは必要なファイルを暗黙裡に各ビルド計算機へ転送してビルドコマンドを実行し，成功した場合に限りビルドされたアプリケーションの実行性能を計測する．すべてのビルド計算機上で並行してビルドし，各ビルド計算機と対応する実行計算機で並行して実行することで，プログラマのソースコード編集作業の中断時間の短縮と，問題の起きたビルドや実行が他のビルドや実行に影響しない独立性を実現する．

本システムでは，実行性能を計測するために，アプリケーションの実行スクリプトと入力データはあらかじめ指定されていることを前提とする．指定された実行スクリプトの実行時間を実行性能として計測し，各バージョン間で実行性能を相対的に比較することにより，実行性能の低下を検知する．したがって，前提とするアプリケーションの実行スクリプトと入力データ以外に，アプリケーション自体への変更作業や実行性能取得のための開発作業を必要としない．各種プロファイラが出力する OTF（Open Trace Format）[8] などの性能値やアプリケーションが独自に出力する性能値を用いる場合には，特定のファイルや標準出力，標準エラー出力から性能値を取得する手続きを外部から提供することも可能である．この場合はプログラマによる簡単なスクリプトなどの開発作業を必要とするが，これにより (1) 様々なプロファイラの出力を用いること，(2) アプリケーション中の特定のカーネルの実行性能だけに着目すること，(3) アプリケーション中の複数のカーネルの実行性能に着目すること，などに対応できる．このように本提案システムでは，動作確認方法をプログラマが指定することにより，多様なアプリケーションに対応可能となっている．

以上のように，本システムは正常にビルド可能で，かつ機能的に正常動作可能なバージョンの実行性能計測を自動化する．一部あるいはすべてのビルド計算機上でビルドに失敗する場合や実行計算機上で機能的に正しく動作しない場合については，本システムによる性能取得の対象としては考慮しない．機能的に正しく動作することを保証するためにはバグを自動的に検知する手法 [1], [2] や単体テスト [3] を行う手法が提案されており，これらの手法を Eclipse のビルド機能やバージョン管理システムのコミット時に追加することにより実現が可能である．

3.3　指定されたすべてのビルド計算機とファイルを同期する機能

一般的に，プログラマがビルド計算機および実行計算機に自由にデーモンプログラムをインストールすることは許されていないため，各ビルド計算機に特別なデーモンプログラムの存在を仮定することは現実的ではない．このため自動性能追跡システムは，すべてのビルド計算機で利用可能であることが期待できる SCP のみを用いて，ビルド計算機間のファイルの同期を行う．デーモンプログラムを使って同期するアプローチと比較して，SCP によるアプローチではファイル転送の効率が低いことが懸念されるが，より広い適用可能範囲を期待できる．また，統合開発環境の動作と並行してファイル転送が行われるため，ファイル転送処理がプログラマの作業を妨害することはない．各ビルド計算機で自動的にビルド処理を実行するため，ソースコードファイルに加えてビルド処理を記述したファイルなども SCP 経由で同期する．

4.　自動性能追跡システムによる開発支援機能の評価

本論文では，オープンソースであり広く使用されている Eclipse を統合開発環境として用い，自動性能追跡システムを実装し有効性を示す．自動性能追跡システムは，`Eclipse 4.2.1 Build id:20121004-1855` を対象とし，Eclipse プラグインの開発環境 The Plug-in Development Environment（PDE）を用いて実装されている．また，開発計算機中にインストールされた SCP コマンドをプラグインから起動することにより，遠隔のビルド計算機へのファイル転送・同期機能を実現している．本評価で用いられる SCP プログラムのバージョンは `OpenSSH_6.1p1` である．本実装では，ビルドコマンドとして HPC アプリケーション開発で一般的に使用されている `make` コマンドを用い，ビルドファイルとしては `Makefile` を用いる．ビルドされたアプリケーションは SSH を用いて実行計算機上で実行され，実行に要した実時間を測定して実行性能を取得する．

4.1　性能可搬性低下要因の解析

本評価では，ナノ粒子群形成シミュレーション [9] を評価に用いる．3 台のビルド計算機を開発計算機から利用することを考える．すべてのビルド計算機は，東北大学サイバーサイエンスセンター内に設置されている．また，東北大学青葉山キャンパス内の別の建物に設置されたデスクトップ PC（Intel Core i7-3930K 3.2 GHz, 16 GB Memory, SSD）を開発計算機として用い，プログラマが学内ネットワークを経由して遠隔のビルド計算機を利用する状況を想定して評価を行う．各ビルド計算機のハードウェア構成，開発計算機からのネットワーク条件，ソフトウェア構成を**表 1** に示す．

表 1　ビルド計算機と実行計算機の構成（各計算機はビルド計算機と実行計算機を兼ねる）
Table 1　Build and execution computer specifications.

計算機	Linux	CPU	GPU	開発計算機からの 1 MB SCP 平均時間	CUDA
server1	2.6.18	Core i7 920 2.67 GHz	Tesla C1060	0.90 s	5.0
server2	2.6.32	Core i7 930 2.8 GHz	Tesla C2070	0.59 s	5.0
server3	2.6.18	Core i7 920 2.67 GHz	Tesla K20c	0.90 s	5.0

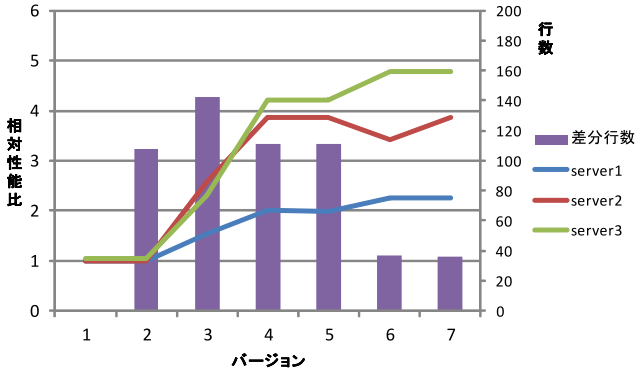

図 3　バージョン推移にともなう性能変化と対応するソースコード変更行数
Fig. 3　Execution performances of application versions and line numbers of corresponding code difference.

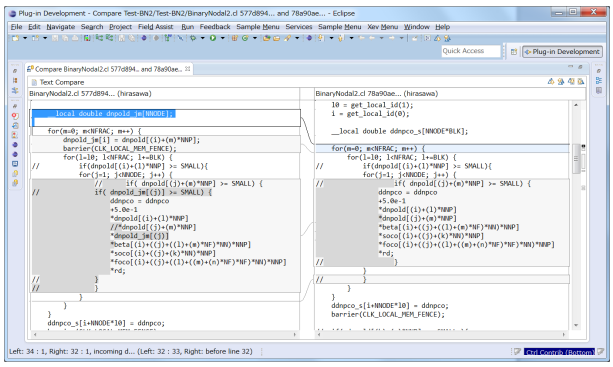

図 4　実行性能を低下させるバージョン更新に対応するソースコード変更の表示
Fig. 4　Presentation of the code difference corresponding to performance degradation.

本提案システムの性能履歴の自動追跡機能を用いて，性能可搬性を低下させる要因となるソースコード変更を特定できることを実証する．server1，server2，および server3 の各計算機はビルド計算機と実行計算機を兼ねる．

評価結果を**図 3** に示す．図 3 の縦軸は，バージョン 1 のコードを server1 で実行したときの実行性能を基準とした場合の速度向上率である．バージョン 1 においては，3 台の実行計算機における実行性能はきわめて近接していた．縦第 2 軸は各バージョンの直前バージョンからの変更行数である．本アプリケーションは表 1 の server1 を対象とし，バージョンが進むに従って最適化されてきた [10]．このため図 3 に示すとおり，バージョンが進むに従って server1 における性能が向上している．一方，server1 に搭載されている NVIDIA 社の GPU である Tesla C1060 の代わりに，より新しい世代の GPU である Tesla C2070 を搭載した server2 では，バージョン 5 から 6 へのコード変更で性能が悪化している．この結果から，バージョン 5 からバージョン 6 への変更は C2070 における実行性能を低下させており，性能可搬性を悪化させる要因となっていることが分かる．さらに新しい世代の GPU である K20 を搭載する server3 では性能低下がみられないことから，当該変更は C2070 を搭載する server2 でのみ性能を低下させる要因となっていることが分かる．本提案システムは，コード変更履歴とそれにともなう性能変化を記録することにより，以上のように性能可搬性を低下させる要因を特定することが可能である．

本アプリケーションの場合，バージョン 5 からバージョン 6 への変更に性能可搬性を低下させる要因が含まれていることは明らかであり，2 個のバージョンの差分から当該変更点が性能可搬性を低下させていることが分かった．バージョン 5 からバージョン 6 への変更部分は 37 行であった．開発したプラグインは性能可搬性を低下させるバージョン変更を発見した場合に，Eclipse に備わるコード差分表示機能を用いて**図 4** のようにコード変更部分を表示し，プログラマに修正を促す．

以上のように，コードの変遷と実行性能とを対応付けて管理することにより，本提案システムは一部の実行計算機での性能を低下させる要因となったソースコードの変更箇所を特定することが可能である．本評価では，世代の異なる GPU を対象とし，世代ごとに異なる記述が求められるソースコードの範囲を 37 行にまで限定した．また，GPU 世代間に限らず，より一般的なシステム間で性能可搬性を低下させる要因となるソースコードの変更候補を同じアプローチで限定できる．このことから，性能可搬性を維持しつつアプリケーションを中長期的に保守・進化させるための支援ツールとして有用であるといえる．

4.2　大規模アプリケーションに対する性能変化要因の特定

大規模なアプリケーションにおいて，提案システムが実際に有用であることを評価する．用いたアプリケーションは DC-RF ハイブリッド熱プラズマの 3 次元流動ダイナミクス計算アプリケーション（以降，熱プラズマアプリケー

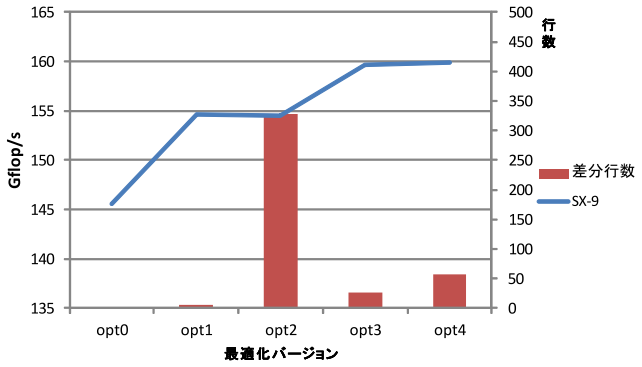

図 5　各最適化による実行性能と対応するソースコード変更行数
Fig. 5　Execution performances and line numbers of corresponding code difference.

ション）[11] である．熱プラズマアプリケーションは東北大学サイバーサイエンスセンターに設置されている SX-9 ベクトル型スーパコンピュータで 2013 年現在において研究開発されている大規模アプリケーション（7,677 行）であり，様々な最適化が施されてきた．

しかしながら，過去行われた最適化が性能に与えている影響を明らかにすることが，SX-9 での性能を維持しつつ他の計算システム上での実行性能も高め，同アプリケーションの性能可搬性を高めるために必要である．そこで本提案システムを用いて，性能が変化した最適化に対応するソースコード部分を限定する．SX-9 における実行性能に大きな影響を与えるソースコード変更を限定することにより，GPU クラスタなどのまったく異なる最適化を要求する計算システムへの移植の際に重要な部分のみ別コードを用意するといった対応策を検討しやすくなる．

評価結果を**図 5** に示す．横軸は各最適化バージョン，縦軸は絶対性能（Gflop/s），縦第 2 軸は直前の最適化バージョンからのソースコード変更行数である．本システムにより，最適化により性能が変化した opt1 と opt3 に特定することにより，性能に変化を与えたソースコード変更を最大でも 27 行に抑えることができた．HPC アプリケーションの最適化技法には性能可搬性を低下させるものが多く存在し，複数の計算システムで実行する際に最適化を困難にさせることが多い [5] が，本システムによりプログラマが考慮するべきソースコード量を限定することができた．したがって，本提案システムを用いることによって大規模アプリケーションの性能への影響が大きいソースコードの範囲を大幅に限定することが可能であることから，性能を意識しつつアプリケーションを進化させる際に有用なツールであることが示された．また，ソースコードの変遷と実行性能との関係を容易に解析することが可能であり，特定の計算システムへの過度な最適化を探し出す際にも有用であることが期待される．

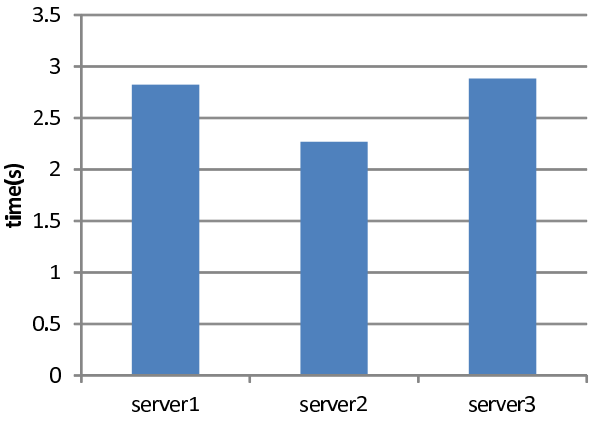

図 6　各ビルド計算機とのファイル同期時間
Fig. 6　Synchronization time to build computers.

4.3　アプリケーションソースコードの遠隔同期

本節では表 1 に示された 3 台のビルド計算機と開発計算機との間でソースコードの同期に要する時間を評価する．提案システムを用いて，4.1 節の評価で用いたナノ粒子群形成アプリケーションのソースコードを開発計算機から各ビルド計算機に転送する評価を行った．本アプリケーションの合計ファイルサイズは，非圧縮の状態で **816** KB であり，ファイル数は 53 である．転送時間の実測値を**図 6** に示す．図 6 から，本アプリケーションの全ソースファイルをビルド計算機に転送しても，転送時間はそれぞれ実測値で 3 秒未満であることが分かる．本提案システムは，ビルド計算機が新たに追加された場合のみすべてのソースコードを新しいビルド計算機に転送し，それ以外の場合には修正されたファイルのみを転送する．このため，通常の開発においてはさらに短い時間内で開発計算機と各ビルド計算機との間でファイルを同期させることができる．また，このファイル転送は統合開発環境の背後で自動的に行われるため，その期間内もプログラマは継続してコードを編集することができる．したがって，プログラマが意識することなく，変更されたファイルが暗黙裡のうちに遠隔のビルド計算機へと転送される．

対象とするビルド計算機の数が多い場合，それぞれのビルド計算機にコマンドを組み合わせて手動でファイルを転送する労力は増加し，作業ミスの確率も増える．一方，本提案システムでは自動的にファイルを同期させるため，図 1 のように開発計算機とビルド計算機が別の計算機になっていることをプログラマが強く意識する必要はない．このため，複数の計算機間で性能可搬性を保ちつつアプリケーションを開発する場合に，本提案システムによる開発作業の効率化が期待できる．

5. 関連研究

5.1 性能追跡の関連研究

CodingTracker [12], [13] は本研究と同じく Eclipse 統合開発環境と連携してプログラマのコード編集にともなうプログラムの性質変化（たとえば，単体テストのカバレッジなど）を追跡する．しかしながら，実際の実行計算機上での実測による実行性能は取得できず，また新たに追加した実行計算機上で過去の履歴を巻き戻して実行することはできない．

5.2 アプリケーションのビルドおよび実行の関連研究

遠隔システム上でプログラムを実行する仕組みとして，Eclipse 統合開発環境から使用可能な Eclipse RSE（Remote System Explorer）[14] が開発されている．しかしながら，RSE では実行したいプログラムがすでに遠隔システム上に存在することを想定しており，本自動性能追跡システムのようにプログラマの編集しているソースコードを自動的に転送する機能は実現されていない．また，遠隔ログインしてビルド，実行することは本質的に遠隔システムのコンソールにログインしてエディタを起動して行う開発作業と同等である．このため，複数の計算システム向けにビルド結果および実行結果を確認しつつ対話的に開発を進めていく作業を支援する目的には不適当である．

アプリケーションを自動的に他の計算システムで実行できるように変換する仕組みとして，ADAPT [15] が提案されている．しかし，ADAPT の場合，変換後のコードをプログラマが保守管理することは困難であり，アプリケーションのさらなる機能追加・変更ができなくなるという問題点がある．HPC アプリケーションには継続的に機能の追加や変更が求められることから，本研究のようにプログラマの保守管理できる形式でコードを維持することがきわめて重要である．

C/C++/Fortran などコンパイルが必要なプログラミング言語に対して，多数のビルド環境でビルドするためのツールとして CMake [16], [17] や SCons [18] が開発されている．これらのツールは CUI コマンドとして実装されており，複数のビルド環境上でビルドを行うためにはプログラマ自身がログインして手作業で行うか，スクリプトファイルなどを記述する必要がある．また本自動性能追跡システムでは，ビルド記述ファイルの転送機能，ビルドコマンドの遠隔実行機能を介してこれらのツールを利用することが可能である．

Slawinska らは HPC アプリケーション向けのポータブルなビルドシステム [19] を提案しているが，統合開発環境との連携を考慮しておらず，プログラマのビルド情報は各ビルドシステム上に分散する．またビルド実行をプログラマの開発計算機から対話的に実行することは考慮されていない．

統合開発環境と密に連携するビルドシステムとして，Apache Maven [20] が存在する．Maven は主に Java 言語を対象とし，プログラマの開発計算機上でビルドを行い，実行計算機へコンパイル済みの実行ファイルを自動的に転送することが可能である．しかしながら，アーキテクチャの異なる実行計算機上で同様に動作することが期待できる Java 言語を前提としている．一方，HPC アプリケーションでは各実行計算機に固有のビルド処理を必要とする場合があり，Apache Maven はそのようなビルド処理には対応していない．

5.3 ソースコードの転送の関連研究

Eclipse 統合開発環境には，HPC 向けなど並列アプリケーションの開発を支援するプラグインとして Parallel Tools Platform（PTP）[21] が存在する．PTP には Git を用いる同期機能である Synchronized Projects が存在するが，ビルド計算機上に Git がインストールされている必要があり，プログラマが自由にコマンドをインストールできないビルド計算機では使用できない．また PTP には特別なサーバプログラムを常駐させることによりソースコードの同期機能やプログラムの実行機能を実現する Remote Development Tools（RDT）も存在するが，サーバプログラムを起動できないビルド計算機においては使用することができない．

6. まとめ

本論文では，アプリケーションのソースコード中で性能可搬性を低下させるソースコード部分を自動的にプログラマに提示するための自動性能追跡システムの設計と実装を行った．本論文で設計した自動性能追跡システムは統合開発環境である Eclipse のプラグインとして実装され，編集したソースコードと自動生成したビルドファイルを複数のビルド計算機に自動的に転送してビルドおよび実行を行う機能を持つ．

自動性能追跡システムにより，プログラマが他のビルド計算機を選択してビルド，実行する労力を低減でき，性能可搬性の低下の要因となっているソースコード部分の限定を支援できる．プログラマが複数の実行計算機において高い実行性能を発揮する状況を維持しつつアプリケーションの開発作業を進めることにより，様々な種類，規模，および世代の実行計算機に対する性能可搬性が期待できる．

本論文における実装と評価の結果，実アプリケーションにおいて性能可搬性を低下させるソースコード変更部分を適切に限定し，プログラマの修正を支援できることが示された．また，複数のビルド計算機に対してソースコードを転送・同期し，複数の実行計算機上でアプリケーションの実行性能を自動的に取得することができる．これらの機能

によりプログラマが手動で行う必要があった実行性能の計測作業を自動化し，性能可搬性を意識したHPCアプリケーション開発の支援に必要な実行性能の自動追跡機能を実現できた．

今後の展望として，実行計算機におけるアプリケーションプログラムの単純な実行に加えて自動的にgprof，nvprofなどのプロファイラと連携して性能プロファイルを取得し，結果をプログラマにフィードバックして性能可搬性を低下させる要因の特定を支援することが考えられる．また，HPCシステムにおける実行管理システムとして一般的に用いられている各種バッチキューシステムに対応した実行を自動的に行う支援が考えられる．また，実装面においてはPTP [21] を拡張する形で提案機能を実装することにより，エディタを通じたプログラマに対する性能情報のフィードバックを行うことが考えられる．

謝辞　ナノ粒子群形成アプリケーションと熱プラズマアプリケーションを提供していただいた大阪大学接合科学研究所の茂田正哉准教授と，ナノ粒子群形成アプリケーションの最適化について様々なご議論をいただいた東北大学大学院情報科学研究科の菅原誠氏に感謝します．熱プラズマアプリケーションの最適化についてご議論いただいた東北大学サイバーサイエンスセンターの江川隆輔准教授と小松一彦助教と山下毅氏に感謝します．査読者の方々には，本論文で不足していた議論を適切にご指摘いただき，また改善の提案をいただいたことに感謝します．本研究の一部は，JST CREST研究課題「進化的アプローチによる超並列複合システム向け開発環境の創出」，および文部科学省科研費若手研究（B）（23700028）の支援によります．

参考文献

[1] Williams, C. and Hollingsworth, J.: Automatic mining of source code repositories to improve bug finding techniques, *IEEE Trans. Software Engineering*, Vol.31, No.6, pp.466–480 (online), DOI: 10.1109/TSE.2005.63 (2005).

[2] Kim, S., Zimmermann, T., Pan, K. and Whitehead, E.: Automatic Identification of Bug-Introducing Changes, *21st IEEE/ACM International Conference on Automated Software Engineering, 2006, ASE '06*, pp.81–90 (online), DOI: 10.1109/ASE.2006.23 (2006).

[3] Zhu, H., Hall, P.A.V. and May, J.H.R.: Software unit test coverage and adequacy, *ACM Comput. Surv.*, Vol.29, No.4, pp.366–427 (online), DOI: 10.1145/267580.267590 (1997).

[4] Watson, G. and DeBardeleben, N.: Developing scientific applications using eclipse, *Computing in Science Engineering*, Vol.8, No.4, pp.50–61 (online), DOI: 10.1109/MCSE.2006.64 (2006).

[5] 小松一彦，江川隆輔，安田一平，撫佐昭裕，松岡浩司，小林広明：HPCアプリケーションの性能可搬性に関する一検討，第136回HPC研究会 (2012).

[6] CVS – Concurrent Versions System, available from ⟨http://cvs.nongnu.org/⟩.

[7] Git – the fast version control system, available from ⟨http://git scm.com/⟩.

[8] Malony, A.D. and Nagel, W.E.: The open trace format (OTF) and open tracing for HPC, *Proc. 2006 ACM/IEEE Conference on Supercomputing*, SC '06, ACM, (online), DOI: 10.1145/1188455.1188480 (2006).

[9] Shigeta, M. and Watanabe, T.: Growth model of binary alloy nanopowders for thermal plasma synthesis, *Journal of Applied Physics*, Vol.108, No.4, pp.043306–043306–15 (online), DOI: 10.1063/1.3464228 (2010).

[10] 菅原　誠，小松一彦，平澤将一，滝沢寛之，小林広明：ナノ粒子群形成アプリケーションのOpenACCによる実装と性能評価，第136回HPC研究会 (2012).

[11] Shigeta, M.: Three-dimensional flow dynamics of an argon RF plasma with dc jet assistance: A numerical study, *Journal of Physics D: Applied Physics*, Vol.46, No.1, p.015401 (online) (2013), available from ⟨http://stacks.iop.org/0022-3727/46/i=1/a=015401⟩.

[12] Negara, S., Vakilian, M., Chen, N., Johnson, R. and Dig, D.: Is It Dangerous to Use Version Control Histories to Study Source Code Evolution?, *ECOOP 2012 – Object-Oriented Programming*, Noble, J. (ed.), Lecture Notes in Computer Science, Vol.7313, pp.79–103, Springer Berlin/Heidelberg (online), DOI: 10.1007/978-3-642-31057-7_5 (2012).

[13] Negara, S., Chen, N., Vakilian, M., Johnson, R.E. and Dig, D.: Using Continuous Code Change Analysis to Understand the Practice of Refactoring, Technical Report (2012), available from ⟨http://hdl.handle.net/2142/33783⟩.

[14] Foundation, T.E.: Eclipse Target Management (RSE) (2012), available from ⟨http://eclipse.org/tm/⟩.

[15] Bourgeois, J., Sunderam, V., Slawinski, J. and Cornea, B.: Extending Executability of Applications on Varied Target Platforms, *2011 IEEE 13th International Conference on High Performance Computing and Communications* (*HPCC*), pp.253–260 (online), DOI: 10.1109/HPCC.2011.41 (2011).

[16] Wojtczyk, M. and Knoll, A.: A Cross Platform Development Workflow for C/C++ Applications, *The 3rd International Conference on Software Engineering Advances, 2008, ICSEA '08*, pp.224–229 (online), DOI: 10.1109/ICSEA.2008.41 (2008).

[17] Hoffman, B., Cole, D. and Vines, J.: Software Process for Rapid Development of HPC Software Using CMake, *DoD High Performance Computing Modernization Program Users Group Conference* (*HPCMP-UGC*), *2009*, pp.378–382 (online), DOI: 10.1109/HPCMP-UGC.2009.62 (2009).

[18] Fomel, S. and Hennenfent, G.: Reproducible Computational Experiments using Scons, *IEEE International Conference on Acoustics, Speech and Signal Processing, 2007, ICASSP 2007*, Vol.4, pp.IV–1257–IV–1260 (online), DOI: 10.1109/ICASSP.2007.367305 (2007).

[19] Slawinska, M., Slawinski, J. and Sunderam, V.: Portable builds of HPC applications on diverse target platforms, *IEEE International Symposium on Parallel Distributed Processing, 2009, IPDPS 2009*, pp.1–8 (online), DOI: 10.1109/IPDPS.2009.5160915 (2009).

[20] McIntosh, S., Adams, B. and Hassan, A.: The evolution of Java build systems, *Empirical Software Engineering*, Vol.17, pp.578–608 (online), DOI: 10.1007/s10664-011-9169-5 (2012).

[21] Watson, G., Rasmussen, C. and Tibbitts, B.: An integrated approach to improving the parallel application development process, *IEEE International Symposium*

on Parallel Distributed Processing, 2009, IPDPS 2009, pp.1–8 (online), DOI: 10.1109/IPDPS.2009.5160941 (2009).

平澤 将一　（正会員）

平成 12 年東京大学理学部情報科学科卒業．平成 14 年東京大学大学院理学系研究科情報科学専攻修了．平成 17 年東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程単位取得退学．平成 18 年電気通信大学大学院情報システム学研究科助手．平成 19 年同助教．平成 23 年同産学官連携研究員．平成 24 年東北大学大学院情報科学研究科産学官連携研究員．高性能計算システム，プログラミング言語処理に関する研究に従事．ACM，IEEE-CS 各会員．

滝沢 寛之　（正会員）

平成 11 年東北大学大学院情報科学研究科博士課程修了．博士（情報科学）．同年新潟大学総合情報処理センター助手，平成 15 年東北大学情報シナジーセンター助手，平成 16 年東北大学大学院情報科学研究科講師を経て，平成 21 年より同研究科准教授．高性能計算システム，コンピュータアーキテクチャとその応用に関する研究に従事．平成 16 年 ISPA04 最優秀論文賞，平成 18 年船井情報科学奨励賞，平成 20 年情報処理学会東北支部野口研究奨励賞，平成 21 年石田記念財団研究奨励賞受賞．電子情報通信学会，IEEE-CS，ACM 各会員．

小林 広明　（正会員）

昭和 63 年東北大学大学院博士課程修了．同年東北大学助手．平成 3 年東北大学講師．平成 5 年東北大学助教授．平成 13 年東北大学教授（平成 20 年 4 月よりサイバーサイエンスセンター長兼任）．平成 18 年より NII 客員教授併任．コンピュータアーキテクチャ，並列処理システムとその応用に関する研究に従事．工学博士．IEEE Senior Member，ACM，電子情報通信学会各会員．