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]

概要：本研究では，コード修正に伴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コード修正を見つけるための自動性能追跡システムを提案する．提案システムは統合開発環境 (IDE) と連携して動作し，複数の計算プラットフォー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を進化させる作業を対話的に支援する．ソースコードがバージョン管理システムのレポジトリに格納された際に，対象とするすべての計算プラットフォーム上で実行性能を評価することにより，提案システムは性能可搬性を低下させるコード修正をアプリケーション開発者が検知することを支援する．さらに，新しい計算プラットフォームが対象として追加された場合，提案システムはレ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，それらを新しいプラットフォーム上で実行する．その結果，アプリケーション開発者は過去のソースコード変更が新しい計算プラットフォーム上での実行性能に与える影響を解析することができる．その解析に基づいて，アプリケーション開発者はソースコード変更の再考や，アプリケーションの設計の改善を図ることができる．

An Automatic Performance Tracking System for Software Evolution

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

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 and improve the software design of the HPC application.

1. はじめに

現在，高性能計算 (HPC) 分野では特定の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採られている．しかし，特定のシステム向けに最適化することにより，その対象システム以外のシステムにおける実行性能が低下する恐れがある．すなわち，アプリケーションが様々な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**が低下し，中長期的なアプリケーションの保守・管理に要する労力が増大してしまう．特に近年，特長の異なるプロセッサを混載する計算システムが増加しており，高い実行性能を達成するためにはそのシステム構成を強く意識した最適化が求められる．そ

[1] 東北大学
[2] JST CREST
[a)] hirasawa@sc.isc.tohoku.ac.jp

の結果，性能可搬性の維持がより一層困難になってきている．以上の背景から，性能可搬性を意識したアプリケーション開発を支援することの重要性が増しており，そのような開発支援フレームワークの実現が求められている．

本研究では，性能可搬性を維持しつつアプリケーションを開発・進化させていくために，ソースコード中で性能可搬性を低下させる恐れのある部分を自動的に特定し，そのコードを編集しているプログラマに対話的に提示する機能の実現を目指している．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．このことから，アプリケーションを様々なプラットフォーム上で暗黙裡に定期的に実行してその性能プロファイルを取得し，性能可搬性を低下させる要因となるコード変更を検出する必要がある．そのような機能を実現するためには，プログラマのコード変更を追跡して実行性能を記録するシステムが必要である．しかし，HPC アプリケーション開発の対話的支援を想定した性能追跡システムを議論した研究はほとんど行われていない．

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の実現のために求められる，自動性能追跡システムを設計・開発し，必要な機能を議論する．複数の計算プラットフォームにまたがって動作可能なように開発されたアプリケーションは，新規に登場したプラットフォームへ移行しやすく，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．

本性能追跡システムは，アプリケーションプログラムを異なる計算プラットフォーム上で並列にビルドして実行し，その性能プロファイルに基づいてチューニング作業を行うことを支援する．特定の計算プラットフォームに過度に依存した性能最適化を行うことにより他の計算プラットフォームでの実行性能が低下した場合に，そのような性能可搬性の低下やその原因となるコードの変更個所の特定を容易に行うことが可能であり，結果として性能可搬性の維持を支援することができる．

本論文の構成は以下の通りである．第 2 章では本研究で想定する計算プラットフォームとそのアプリケーション開発の問題点を述べる．第 3 章では，性能可搬性維持の支援を目的として自動性能追跡システムを設計する．第 4 章で評価を行い，第 5 章では本論文の関連研究を述べる．第 6 章では，本論文の結論と今後の課題について述べる．

2. アプリケーション開発の問題点

一般的にアプリケーションの可搬性が高いとは，ソースコードが複数の計算プラットフォーム上で動作可能であることを示している．また，性能可搬性が高いとは，そのソースコードをビルドすることによって生成された実行プログラムが，複数の計算プラットフォームで高い実行性能を達成可能であることを示している．前者を後者と明確に区別する必要がある場合には，前者を**機能可搬性**と表記する．

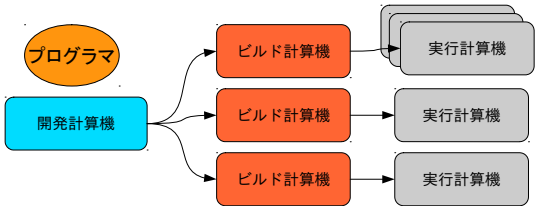

図 1 本研究で想定する計算機構成
Fig. 1 Target computers

HPC アプリケーション開発においては，機能可搬性と同等に性能可搬性が重要視される．このため本研究では，アプリケーションの保守管理，機能拡張のためにソースコードを編集する際に，アプリケーションの性能可搬性の維持を支援することを考える．

本研究で想定する HPC アプリケーション開発において，大きく分けて 3 つの用途で計算機が使用される．

(1) 開発計算機: プログラマがソースコードを編集する計算機
(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，開発計算機上に多く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求められ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のすべてにビルド環境を構築することは現実的ではない．また，実行計算機の処理能力はアプリケーション実行に利用されるべきであり，ビルド処理を実行計算機上で過度に行うべきではない．これらの理由から本研究では開発計算機，ビルド計算機，実行計算機がそれぞれ別々の計算機であることを仮定し，それらを効果的に連携させることを考える．**図 1** に，本研究で想定する計算機の構成を示す．

アプリケーション開発中に高い性能可搬性を維持し続けるためには，ソースコードの変更により性能可搬性が低下する場合に，それをプログラマが即時に認識できることが望まれる．性能可搬性が低下する変更点の検知のためには，最新のソースコードに対する実行性能評価だけでは不十分であり，過去のバージョンのソースコードに対する実行性能評価結果が必要である．また，新しい実行計算機が追加された場合には，過去に行ったソースコードの変更のうち，新実行計算機で性能を低下させる変更を検出する必要がある．そのような検出は，過去に行ったソースコードの変更と他の変更が部分的に重なっていた場合にさらに困難となる [10].

現在の一般的なHPCアプリケーション開発においては，多くの場合，編集中のソースコードを定期的にビルドおよび実行し，機能可搬性や性能可搬性が維持されていることを手動で確認している．また，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，開発者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，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．このため，本論文ではこれらの作業を自動化することによって，性能可搬性を考慮しつつコードを編集する作業を支援することを考える．

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，ソースコードおよびビルド情報を指定する記述ファイルをビルド計算機に自動的に転送し，ビルド作業を行って実行する自動性能追跡システムを提案する．プログラマによるコード改変，ビルド手順の改善，性能チューニングによる，HPCアプリケーションのさらなる進化を支援するため，本システムはプログラマと直接的な接点を持たず，Eclipse[14]に代表される統合開発環境をインタフェースとして利用することを想定している．多くの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，本ビルドシステムも統合開発環境と連携することが可能である．

本システムは

(1) ソースコード，ビルドファイル，その他リソースファイルを指定されたすべてのビルド計算機に転送して同期する機能
(2) 指定されたすべてのビルド計算機上でビルドコマンドを発行する機能
(3) 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
(4) 実行計算機を追加した際に過去のコード変更履歴を追跡して自動的に性能の変遷を取得する機能

を持つ．これらにより，プログラマの開発計算機上で改変されたソースコードを複数のビルド計算機上で自動的にビルド・実行し，各実行計算機上での実行結果をプログラマが操作する開発計算機上に表示することができる．

3.1 ビルド処理の遠隔実行機能

統合開発環境にはプログラマがソースコードを編集するためのエディタが用意されている．そのようなエディタは，さまざまなプログラミング言語に対して編集，補完，キーワードのハイライトなどの編集支援機能を持っているため，今日のHPCアプリケーション開発においては統合開発環境のエディタを利用してコード編集を行うプログラマが増加している[11]．

本提案システムでは，プログラマが統合開発環境のエディタでソースコードを編集することを想定している．プログラマによる編集作業が一区切りした時点でビルド処理が自動的に暗黙裡に実行され，開発中のアプリケーションが各ビルド計算機上でビルド可能であることが確認される．例えば，プログラマがソースコードをファイルに保存したことをきっかけとしてビルド処理を実行することが考えられる．

コード編集作業を中断することなくビルド処理を行い，対話的にビルド結果をプログラマに提示するためには，ビルド処理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，ビルドコマンドを実行する機能を有している．開発計算機とは別の計算機でビルド処理を実行することにより，プログラマは開発計算機上でコード編集を続行することができる．また，本提案システムはビルド処理をすべてのビルド計算機上で並行して実行することで，ビルド時間の短縮と，問題の起きたビルドが他のビルドに影響しない独立性を実現している．

3.2 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．むしろ，過去のコード変更において，新実行計算機における性能を低下させる要因となる変更を行っている可能性が高い．そこで，本提案システムでは，CVS[1]やGit[2]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伴う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レ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，ビルド，実行する．隣接する2つのバージョンの実行性能を比較することにより，過去のソースコード変更のうち，新実行計算機における性能を低下させる要因となる変更を検出する．その変更を特定することにより，プログラマは新実行計算機も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，その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．

自動性能追跡システムによる開発支援機能の評価

本研究では提案システムを試作してその実現可能性を示す．本実装においては，オープンソースであり広く使用されている Eclipse が統合開発環境として用いられている．本提案システムは，`Eclipse 4.2 Build id: 20120614-1722` を対象とし，他のプラグインに依存しない独立したプラグインとして実装されている．また，遠隔のビルド計算機へのファイル転送には，プラグインより開発計算機中にインストールされた SSH コマンドを起動することにより行なっている．用いた SSH 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．

4.1 アプリケーションソースコードの遠隔同期機能の評価

本評価に用いるアプリケーションは，代数的マルチグリッド（AMG）法のライブラリである AMGS[13] の逐次版 Version 0.18 である．本ライブラリは大規模線形方程式を高速に解くことを目的として AMG 法を実装しているライブラリであり，Fortran90，C，および CUDA で記述されたソースコードから構成されている．合計ファイルサイズは非圧縮の状態で **614** KB（ファイル数 48，合計 16795 行）である．

開発計算機として，一般的なラップトップ PC(Intel Core i5-2450M 2.50GHz, 8GB Memory, SSD) を用いる．ビルド計算機として**表 1** の 4 台を用いる．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を示す．gfortran コンパイラ，ifort コンパイラ，CUDA ツールキットの有無およびバージョンを示す．これらは，プログラマ個人が使用する計算機 (pc)，アクセラレータ型クラスタ計算機 (cluster1,cluster2)，スーパコンピュータのログインノード (gen) を想定しており，HPC アプリケーションの開発において使用頻度の高い実行計算機に対応するビルド計算機を想定している．

本実装では，新たなビルド計算機が利用可能となるたびに，すべてのソースコードおよびビルドファイルをその新しいビルド計算機へ転送してファイルを同期する．評価に用いるアプリケーションライブラリのうち，サンプルプログラムを除いたファイル数 45（合計 16639 行，603KB）を用いた場合において，新たにビルド計算機を追加した場合に必要となる初回同期時間の実測値を**図 2** に示す．表 1 に示されている通り，cluster2 の平均 RTT およびファイル転送時間が他のビルド計算機と比較して相対的に長い．このため cluster2 においては他のビルド計算機よりも 2 倍ほどの時間を必要としていることがわかる．このように初回の転送に長時間を要した原因として，各ファイルの転送ごとに独立した SSH コネクションを用いている現状の実装が原因と考えられる．全てのファイルを 1 つの SSH コネクションで転送する実装や，さらに圧縮により転送量を減少させる実装を行うことにより短縮することが可能であると考えられる．

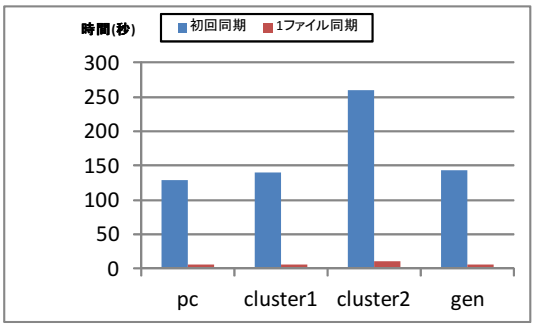

図 2 各ビルド計算機とのファイル同期時間
Fig. 2 Synchronization time to build computers

初回のみ全ファイルを転送する必要があるため，最長で 4 分以上とかなりの同期時間を要している．しかし，2 回目以降はそれぞれ変更のあったファイルのみを転送するため，変更されたファイル数が多くない限り，初回同期時間よりはるかに短時間で同期が終了する．例えば，AMGS ライブラリ中のソースコードで容量が最大の `data_creation_stuben.F90`（4824 行,153KB）を変更した場合の同期に要する時間も図 2 中に示されている．最長となった cluster2 の場合でも 12 秒程度でファイルの同期が完了しており，プログラマがファイルを保存するたびに同期を行ったとしても，許容できる時間で同期を完了できていると言える．

この評価結果から，図 1 のように開発計算機とビルド計算機が別々になっている場合にも，本提案システムによってファイル同期を自動化できるため，プログラマが別々になっていることを強く意識する必要がないことが示された．手動でファイルを同期する必要がないことから，プログラマの労力や作業ミスの可能性を軽減することが可能である．

4.2 性能履歴の自動追跡機能の評価

性能履歴の自動追跡機能の評価に用いるアプリケーションは，Scalable HeterOgeneous Computing (SHOC) Benchmark Suite[4] である．SHOC の Git 管理リポジトリ中には 46 個のバージョン履歴が記録されている．

本評価においては，提案システムを用いてバージョン履歴を追跡して実行性能を取得し，各実行計算機における実行結果を得ることにより性能可搬性を低下させる更新を発見する支援が可能であることを示す．

開発計算機として，ラップトップ PC(Intel Core i5-2450M

表 1 ビルド計算機の構成
Table 1 Build computers

ビルド計算機	Linux	CPU	所在地/開発計算機からの平均 RTT	開発計算機からの 1MB SCP 平均時間
pc	3.2.0	Xeon E31260L 2.4GHz	日本/236ms	11.4s
cluster1	2.6.32	Xeon X5650 2.67GHz	日本/233ms	11.8s
cluster2	2.6.32	Xeon W5590 3.33GHz	米国/426ms	17.8s
gen	2.6.18	Xeon X5570 2.93GHz	日本/216ms	11.1s

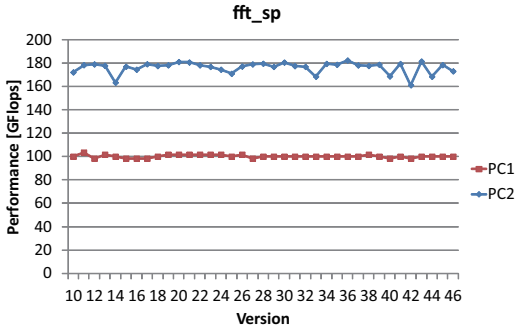

図 3 fft_sp のバージョン推移に伴う性能推移
Fig. 3 Execution time of fft_sp versions

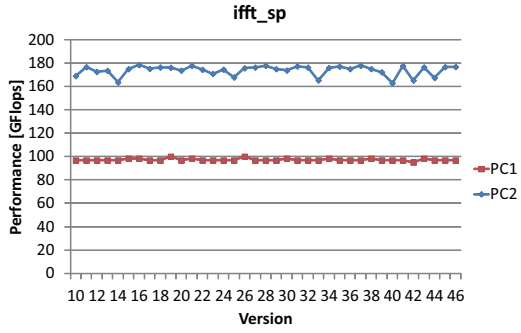

図 4 ifft_sp のバージョン推移に伴う性能推移
Fig. 4 Execution time of ifft_sp versions

2.50GHz, 8GB Memory, SSD) を用いる．実行計算機として**表 2** の 2 台を用いる．

SHOC ベンチマーク中の fft_sp および ifft_sp の実行性能を**図 3** および**図 4** に示す．本提案システムを使うことにより，図 3 および図 4 のようなコードの変遷に伴う性能の変化を自動的に取得することが可能となる．評価結果においてはバージョンの推移に伴う実行性能変化は極めて小さく，対象プログラムがベンチマークプログラムである性質から実行性能が変らないよう維持されつつコードが変遷していることが確認できた．バージョンの推移に伴う実行性能変化が大きいアプリケーションの場合には，実行計算機ごとの実行性能変化に大きな差異が生じた変更点を特定し，アプリケーションの性能可搬性を低下させる要因を解析することが可能である．

これらの実行情報を用いることにより，プログラマは新規世代の GPU を搭載する実行計算機を対象に加えた場合に，過去のコード変更履歴による実行性能変化を検出することができる．ソースコードを適切に編集することにより性能可搬性を維持しつつ開発を継続することができ，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の支援ツールとして有用であることがわかる．

5. 関連研究

CodingTracker[9], [10] 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伴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実行すること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．

Eclipse 統合開発環境には，ネットワークを介して離れた計算機に遠隔ログインすることによりビルド，実行を支援するプラグインとして PTP[15] が存在する．しかしながら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，複数の計算プラットフォー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プラットフォームで実行できるように変換する仕組みとして，ADAPT[3] 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極めて重要である．

遠隔システム上でプログラムを実行する仕組みとして，統合開発環境である Eclipse から使用可能な Eclipse RSE(Remote System Explorer)[6] 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，本ビルド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

表 2　実行計算機の構成
Table 2　Execution computers

実行計算機	Linux	CPU	Memory	GPU	CUDA
PC1	2.6.18	Intel Core i7 920	12GB	Tesla C1060	4.0
PC2	2.6.32	Intel Core i7 930	24GB	Tesla C2070	4.0

動的に転送する機能は実現されていない．

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][7] や SCons[5] 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．また本研究で開発したビルドシステムでは，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 [12] 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[8] 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．

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恐れのある部分を対話的にプログラマに提示することを目指し，そのために必要な機能として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コード変更の特定や，新たな実行計算機への対応を支援できる．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，提案するシステムは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追跡することができた．これらの機能により，これまでプログラマがすべての実行計算機において手動で行う必要があった性能評価作業を自動化することができた．その結果，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達成できた．

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実行に加えて，自動的に gprof，nvprof 等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[15] 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

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] Cvs - concurrent versions system. http://cvs.nongnu.org/.
[2] Git - the fast version control system. http://git-scm.com/.
[3] J. Bourgeois, V. Sunderam, J. Slawinski, and B. Cornea. Extending executability of applications on varied target platforms. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pp. 253 –260, sept. 2011.
[4] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, GPGPU '10, pp. 63–74, New York, NY, USA, 2010. ACM.
[5] S. Fomel and G. Hennenfent. Reproducible computational experiments using scons. In *Acoustics, Speech and*

Signal Processing, 2007. ICASSP 2007. IEEE International Conference on, Vol. 4, pp. IV–1257 –IV–1260, april 2007.

[6] The Eclipse Foundation. Eclipse Target Management (RSE). http://eclipse.org/tm/, 2012.

[7] B. Hoffman, D. Cole, and J. Vines. Software process for rapid development of hpc software using cmake. In *DoD High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2009*, pp. 378 –382, june 2009.

[8] Shane McIntosh, Bram Adams, and Ahmed Hassan. The evolution of java build systems. *Empirical Software Engineering*, Vol. 17, pp. 578–608, 2012. 10.1007/s10664-011-9169-5.

[9] Stas Negara, Nicholas Chen, Mohsen Vakilian, Ralph E. Johnson, and Danny Dig. Using Continuous Code Change Analysis to Understand the Practice of Refactoring. Technical Report http://hdl.handle.net/2142/33783, 2012.

[10] Stas Negara, Mohsen Vakilian, Nicholas Chen, Ralph Johnson, and Danny Dig. Is it dangerous to use version control histories to study source code evolution? James Noble, editor, ECOOP 2012 — Object-Oriented Programming, 第 7313 巻 of *Lecture Notes in Computer Science*, pp. 79–103. Springer Berlin / Heidelberg, 2012. 10.1007/978-3-642-31057-7_5.

[11] Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pp. 12:1–12:10, New York, NY, USA, 2010. ACM.

[12] M. Slawinska, J. Slawinski, and V. Sunderam. Portable builds of hpc applications on diverse target platforms. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1 –8, may 2009.

[13] Kosuke Takahashi, Akihiro Fujii, and Teruo Tanaka. Gpgpu-based algebraic multigrid method. In *Parallel and Distributed Computing and Systems (PDCS 2011)*, December 2011.

[14] G.R. Watson and N.A. DeBardeleben. Developing scientific applications using eclipse. *Computing in Science Engineering*, Vol. 8, No. 4, pp. 50 –61, july-aug 2006.

[15] G.R. Watson, C.E. Rasmussen, and B.R. Tibbitts. An integrated approach to improving the parallel application development process. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1 –8, may 2009.

[16] M. Wojtczyk and A. Knoll. A cross platform development workflow for c/c++ applications. In *Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on*, pp. 224 –229, oct. 2008.