Vol.2012-HPC-136 No.28
2012/10/4

統合開発環境と連携するポータブルなビルドシステム

平澤 将一[1,2,a)]　滝沢 寛之[1,2]　小林 広明[1,2]

概要：本研究では，性能可搬性を保ちつつアプリケーションを開発するためのフレームワーク構築に向けて，ポータブルなビルドシステムを開発する．現在の高性能計算 (High-Performance Computing, HPC) システムの構成は複雑化しており，アプリケーションを実行せずにその実効性能を予測することは困難である．このため本研究では，開発中のアプリケーションを定期的に実行し，その性能プロファイルを暗黙裡に取得して性能可搬性の低い個所を特定し，プログラマに対話的に提示することにより性能可搬性の維持を支援することを想定している．そのようなアプリケーション開発補助ツールを実現するためには，開発中のアプリケーションを暗黙裡に様々なシステム上でビルドし，実行する機能が必要である．本研究では，そのような可搬性を有するビルドシステムを開発し，アプリケーション開発支援環境として必要な機能を議論する．

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，実効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効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．特に近年の HPC システムは，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，高い実効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，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，アプリケーション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効性能を予測することは困難であることから，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，性能可搬性の低い部分を検出することを考える．この場合，開発中のアプリケーションコードを自動的に様々なプラットフォーム向けにビルドし，実行する必要がある．そのような機能を実現するためには，コンパイル作業を行うツールおよびファイル群（例えば `make` コマンドと `Makefile` ファイルなど．以下，**ビルドシステム** と呼ぶ）が様々なプラットフォームに対してポータブル，すなわち可搬性が高い必要がある．しかし，HPC アプリケーション開発の対話的支援を想定してビルドシステムの可搬性を議論した研究はほとんど行われていない．

本研究では，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，ポータブルなビルドシステムを設計・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の移行がしやすく，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．ただし，プログラマのコード編集用端末に様々なクロスコンパイル環境を整備することは非現実的であることから，開発中のコードを定期的にリモートシステム上でビルドするシステムを考える．本ビルドシステムは，プログラマが異なる計算プラットフォーム上で並列にビルドし，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効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個所の特定を容易に行うことも可能であり，その結果として性能可搬性の維持も支援することができる．

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

© 2012 Information Processing Society of Japan

本論文の構成は以下の通りである．まず第 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，ポータブルなビルド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

本研究では，開発されるアプリケーションがソースコードのファイルとそれをビルドするための手順が記述された**ビルドファイル**から構成されるものと仮定する．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効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．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．一方，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もあり，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するためには，前提条件として，そのソースコードが複数の計算プラットフォーム上でビルド可能でなければならない．したがって，性能可搬性を維持するためには，ソースコードの機能可搬性に加えてビルドファイルの機能可搬性も維持する必要がある．しかし，性能可搬性を高めるためには計算プラットフォームごとに別の実装を用意することがしばしば求められるため，ビルド時の複数実装の切り替えに関してもビルドファイル内で考慮する必要がある．このためビルドファイルは複雑化する傾向にあり，その機能可搬性の維持は容易ではない．

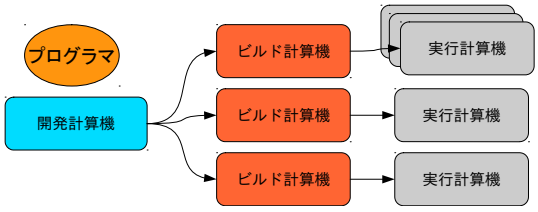

図 1 本研究で想定する計算機構成

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，コード編集作業をいったん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，本研究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. ポータブルで遠隔実行可能なビルドシステム

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，ビルド作業を行うビルド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さらなる HPC アプリケーションの進化を支援するため，本ビルドシステムはプログラマと直接的な接点を持たず，Eclipse[10] 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．

本ビルドシステムは

(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

© 2012 Information Processing Society of Japan

(2) 各ビルド計算機向けにビルドファイルを自動生成する機能
(3) 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

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し，その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開発方法が増加している [6]．

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，そ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 ビルドファイルの自動生成機能

統合開発環境で編集中のソースコードを自動的にビルド計算機でビルドするために，ビルドシステムがポータブルであることも求められる．しかし，性能可搬性を維持するためには計算プラットフォームごとに異なる実装を求められることが多く，環境に依存して異なる動作をするビルド処理が求められる可能性がある．コンパイラのバージョンや種類によって異なるコマンド名（gfortran,g77,ifort など）や，近年 HPC アプリケーションで用いられることが多い GPU 環境 CUDA に対応したソースコードを用意し，CUDA 環境が存在する計算機でのみこれらのソースコードをコンパイルするといった動作が典型的である．環境に依存して異なる動作に対応するために，ビルドファイル中の条件分岐によって異なる動作を記述したり，変数定義をあらかじめ複数記述しておき，1 個の定義以外をコメントアウトすることによって無効とするような記述が行われる．しかしこのように記述されたビルドファイルは，他のビルド環境においてビルド作業を行う際，条件分岐の変更や，コメントアウトする変数定義の変更を手動で行わない限り成功しないビルド構成となり，機能可搬性がない．

```
CUDA = false

ifeq ($(strip $(CUDA)), true)
{ifeq が成立した場合の処理}
endif
```

(a) 元の Makefile

```
CUDA = true

ifeq ($(strip $(CUDA)), true)
{ifeq が成立した場合の処理}
endif
```

(b) 自動生成される Makefile

図 2 条件分岐による Makefile の自動生成

本提案システムにおいては，プログラマの意図による条件分岐や変数定義の記述を残しつつ，手動で行なっていた条件分岐方向の変更や変数定義の置換作業を自動的に行い，複数のビルドファイルを生成する．これら複数のビルドファイルを用いて各ビルド計算機でビルド処理を行うことによって，プログラマは 1 個のビルドファイルを管理するだけで，複数のビルド構成によるビルド処理を行うことができるようになり，ビルドシステムの機能可搬性を向上させることができる．本提案システムでは make コマンドの設定ファイルである Makefile を対象に，条件分岐命令および変数定義によって異なる複数の Makefile を自動生成し，計算プラットフォームに合わせて異なるビルド処理を行う機能を持っている．

図 2(a) のように Makefile 中の条件分岐文に用いられる変数 CUDA が false として代入される記述がされた場合に，この Makefile では ifeq が成立する構成による make を行うことができない．ifeq が成立する構成による make を行うためには，プログラマが手作業によって変数 CUDA を変更する必要がある．そこで変数 CUDA に true を代入するような図 2(b) の Makefile を，元の Makefile に加えて自動生成する．これら 2 個の Makefile に対してそれぞれ make を実行することによって，ifeq が成立する構成および ifeq が成立しない構成によるビルドを行うことができる．

また，変数の定義の記述を元に異なる複数のビルドファイルを生成する．ある変数が 1 度だけ変数定義されてお

© 2012 Information Processing Society of Japan

```
FC = ifort
#FC = gfortran
#FC = gfortran44
```

(a) 元の Makefile

```
#FC = ifort
FC = gfortran
#FC = gfortran44
```

(b) 自動生成される Makefile1

```
#FC = ifort
#FC = gfortran
FC = gfortran44
```

(c) 自動生成される Makefile2

図 3 変数定義による Makefile の自動生成

```
CC=gcc
#CC=icc
CFLAGS=-g -Wall
#CFLAGS=-O2
all:
        $(CC) $(CFLAGS) main.o -o $(TAGRET) -lm
```

(a) 元の Makefile

```
#CC=gcc
CC=icc
#CFLAGS=-g -Wall
CFLAGS=-O2
all:
        $(CC) $(CFLAGS) main.o -o $(TAGRET) -lm
```

(b) 自動生成される Makefile

図 4 生成される Makefile 数の抑制

り，同一の変数名による変数定義がコメントアウトされていた場合に，コメントアウトされた変数定義をコメントから外し，元の Makefile で定義されていた変数をコメントアウトするような Makefile を自動的に生成する．元の Makefile でコメントアウトされた定義が複数存在した場合には，それぞれのコメントアウトされた定義をコメントから外した Makefile を複数自動生成する．例えば Makefile で**図 3**(a) のような記述において，通常では変数 FC は ifort に代入される．同名でコメントアウトされた変数 FC があった場合に，図 3(b)，図 3(c) のようにコメント行を変更した複数の Makefile を自動生成する．

なお，Makefile の自動生成は各条件分岐と変数に対して行うため生成数が多くなる可能性がある．そこで，生成される Makefile 数を削減する．Makefile においては，コンパイラ名を持つ変数 (CC とする) とコンパイラへ与えるオプションを持つ変数 (CFLAGS とする) など，1 対として用いられるような従属した関係を持つ変数が存在する場合がある．CC が値として持つあるコンパイラに対して CFLAGS が持つ別のコンパイラへ与えるべきオプションを与えたコンパイルは意味がなく，エラーが起きる可能性が高い．そこで，従属した関係を持つこのような変数を発見し，これらを組として Makefile を生成することで，自動生成数を削減する．以下の条件を共に満たす変数を従属した関係を持つ変数とみなす．

(1) Makefile 中で，コメントアウトを含めて定義される回数が同じ
(2) Makefile 中で，常に同じ行で使用される

この条件を満たした変数は従属した関係を持つとし，これらを組として Makefile を生成する．この方法による問題点として，1 個の CC に対して複数の CFLAGS を指定した Makefile においては，CC と CFLAGS が従属した変数とはみなされないことが挙げられる．しかし，それぞれの CFLAGS に対応した，同じ値を持つ CC を追加記述することによって Makefile の意味を変更せずに生成する Makefile 数を削減することができる．例えば**図 4** においては，通常では変数 CC は gcc に，CFLAGS は-Wall -g に設定される．しかし，CC と CFLAGS を独立とみなすと元の Makefile を含めて 4 種類生成されることになる．そこで，図 4 では，変数 CC と CFLAGS が常に同じ行で参照されていることに注目し，CC=gcc と CFLAGS=-g -Wall が対応し，CC=icc と CFLAGS=-O2 が対応していることを期待し，その組み合わせのビルドファイルのみを自動生成する．CC=gcc で CFLAGS=-O2 のビルドファイルは生成されない．このルールを適用することによって，自動生成されるビルドファイルの数を制限する．

以上の条件分岐および変数定義の自動生成機能を組み合わせることにより，様々な計算プラットフォームにポータブルなビルドファイルを記述することが可能である．

4. ビルド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とともに，統合開発環境との連携時の対話性を評価する．本評価においては，開発計算機上で編集する実アプリケーションのソースコードを，ネットワーク的に離れた複数のビルド計算機と自動的に同期し，これら複数のビルド計算機上で複数の構成によるビルドが可能であることを示す．さらに，簡便な記述の追加で新たなコンパイラへの対応が容易に可能であることを示す．

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 の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，内部的に SSH が用いられている．本実装では，ビルドコマンドとして，HPC アプリ

© 2012 Information Processing Society of Japan

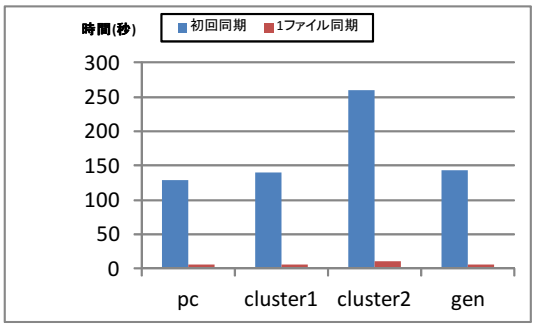

図 5 各ビルド計算機とのファイル同期時間

ケーション開発で最も使われている make コマンドを用い，ビルドファイルとしては Makefile を用いる．

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[8] 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，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)，プログラマの所属組織や共同研究組織の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．

4.1 ファイル同期時間の評価

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 5** に示す．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 cluster2 において，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．

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 5 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．

表 2 単一構成の Makefile によるビルド結果（O:成功，-:実行せず）

ビルド計算機	gfortran	ifort	gfortran + CUDA	ifort + CUDA
pc	O	-	-	-
cluster1	O	-	-	-
cluster2	O	-	-	-
gen	O	-	-	-

表 3 自動生成された Makefile によるビルド結果（O:成功，X:失敗，A:終了せず）

ビルド計算機	gfortran	ifort	gfortran + CUDA	ifort + CUDA
pc	O	X	X	X
cluster1	O	X	O	X
cluster2	O	A	O	A
gen	O	O	X	X

4.2 ビルドファイル自動生成機能の評価

本提案システムのビルドファイル自動生成機能を評価するために，AMGS に含まれている Makefile を用いて各ビルド計算機向けの Makefile を自動生成する評価を行った．

まず，AMGS に含まれる Makefile を用いて gfortran を使用する構成で各ビルド計算機においてビルドを行った場合の結果を**表 2** に示す．すべてのビルド計算機においてビルドが成功しているが，gfortran を使用する構成でビルドを行ったため当然ながら ifort および CUDA を用いたビルド処理は実行されない．これらの構成のビルド処理を実行する場合には，ユーザは手動で Makefile を編集することにより構成を変更しビルド処理を行う必要がある．また，その後においてもビルド構成を変更する度に Makefile を手動で編集する必要がある．

本提案システムでは，条件分岐や変数に対する複数の定義をビルドファイル内に記述し（図 2 および図 3 参照），対応するビルドファイルを自動生成することが可能であ

© 2012 Information Processing Society of Japan

表 1　ビルド計算機の構成

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	gfortran	ifort	CUDA
pc	3.2.0	Xeon E31260L 2.4GHz	日本/236ms	11.4s	4.6.3	-	-
cluster1	2.6.32	Xeon X5650 2.67GHz	日本/233ms	11.8s	4.4.6	-	5.0
cluster2	2.6.32	Xeon W5590 3.33GHz	米国/426ms	17.8s	4.4.4	11.1	4.2
gen	2.6.18	Xeon X5570 2.93GHz	日本/216ms	11.1s	4.1.2	12.1	-

る．AMGS に含まれる Makefile を用いて gfortran を使用する構成では表 2 のように 1 個の構成を用いたビルド処理しか行われなかったが，この自動生成機能を使うことにより，すべてのビルド計算機上でビルドを実行できるようにビルドファイルを記述することができた．ビルド結果を**表 3** に示す．すべてのビルド計算機において gfortran が存在するため，gfortran のみを用いたビルドは表 2 と同様に成功している．pc においては ifort と CUDA 環境は存在しておらず，これらを用いるビルド構成によるビルドは失敗している．cluster1 には CUDA 環境が存在しており，gfortran と CUDA を組み合わせた構成においてもビルドが成功している．cluster2 においてはさらに ifort が存在するが，コンパイルが終了しない問題が存在した．この場合においても gfortran を使用するビルドは並行して暗黙裡に行われており，ifort を用いる構成の問題から影響を受けることなく正常に終了した．gen には CUDA 環境が存在しておらず，CUDA を用いたビルド構成は失敗し，CUDA を用いないビルド構成は成功した．以上から，自動生成した Makefile を用いることにより，各ビルド計算機に存在する複数のコンパイラ環境を用いた複数のビルド構成でビルドを行うことができることがわかった．

なお，cluster1 と cluster2 には CUDA 環境が存在するが，環境変数 LD_LIBRARY_PATH を適切に設定しない限りは必要なライブラリをリンクすることができなかった．また，環境変数 PATH を適切に設定しない限り，CUDA コンパイラである nvcc コマンドを利用可能でなかった．しかしながら，これらの環境設定はプログラマにとってビルド計算機ごとに 1 度行うだけで十分であり，大きな負担とはならない．

4.3　新規ビルド環境への適応

ビルド計算機 gen には，NEC SX シリーズ用の最適化 Fortran コンパイラである sxf90 がインストールされている．しかし，本評価で用いた AMGS に同梱される Makefile には sxf90 コンパイラを使用する記述はないため，sxf90 コンパイラを使ったビルドを行うことはできない．このような場合，本提案システムでは以下の記述を Makefile に追加することで，既存のビルド構成を維持したまま対応する Makefile を自動生成することが可能である．

```
# FC = sxf90
# FFLAGS= -C hopt
```

本記述により，gfortran または ifort が指定されていた変数 FC に sxf90 を代入した場合の Makefile も自動生成されるため，新たに sxf90 コンパイラを用いたビルドを自動的に行うことを確認できた．このように本提案システムでは，新しいビルド環境へ対応する場合に，ビルドファイルの改変ではなく，新たな定義の追記で既存のビルド構成を維持したまま新しい環境に適応することができる．したがって，対応可能なビルド環境を段階的に増やしていくことが可能であり，ビルドファイルの可搬性を安全かつ確実に向上させることができた．

5.　関連研究

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[11] 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[1] 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．

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[12][4] や SCons[2] 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

© 2012 Information Processing Society of Japan

が可能である．

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)[3] 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動的に転送する機能は実現されていない．

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を提案しているが [7]，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[5] 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ポータブルで遠隔実行可能なビルドシステムの設計と実装を行った．本論文で設計したビルド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を行う機能可搬性を持つ．ビルドシステムの機能可搬性により，プログラマが他のビルド計算機を選択してビルドする労力を低減でき，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効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ビルド計算機上で異なるビルド構成によるビルドを実行できた．またビルドを並列に行うことによって，終了しないビルド構成が存在しても他の構成によるビルドが成功することがわかった．評価結果より，通常のコード編集作業においては改変されたファイルのみ同期するため，対話性を大きく損なわない時間でプログラマにビルド結果をフィードバックできることが示された．また，`Makefile` に対する最低限の定義の追加により，既存のビルド構成を維持したまま新規コンパイラを使用したビルドを開始できた．これらの機能により，これまでプログラマがすべてのビルド環境において手動で行う必要があったビルド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要素技術である，ビルドシステムの機能可搬性を高めることができた．

今後の展望として，ビルド計算機におけるビルドに加えて実行計算機で自動的に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，複数のビルド計算機上で行われたビルド結果に基づきプログラマに失敗原因の提示，ならびに修正提案を行うなどビルドファイル（`Makefile` など）の記述に関するさらなる支援が挙げられる．さらに，ビルド結果に加えて，ビルドファイルの静的な解析 [9] を使用してプログラマに対して修正案を提案することが考えられる．これらは，ビルドシステムが統合開発環境と密接に連携することで可能となる機能である．

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」の支援によります．

参考文献

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] M. Slawinska, J. Slawinski, and V. Sunderam. Portable builds of hpc applications on diverse target platforms.

© 2012 Information Processing Society of Japan

In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1 –8, may 2009.

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

[9] A. Tamrawi, Hoan Anh Nguyen, Hung Viet Nguyen, and T.N. Nguyen. Build code analysis with symbolic evaluation. In *Software Engineering (ICSE), 2012 34th International Conference on*, pp. 650 –660, june 2012.

[10] 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.

[11] 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.

[12] 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.

© 2012 Information Processing Society of Japan