pa-39

Charged Particle Tracking Code with Java

Ryoichi Hajima and Shunsuke Kondo

Dept. Quantum Eng. and Systems Sci., University of Tokyo
Hongo 7-3-1, Bukyo-ku, Tokyo 113 Japan

Abstract

A particle tracking code with programming language Java is presented. Using Java, we can treat all the routines for various accelerator components as completely independent modules which make the simulation code simple and plain. General description of the simulation code and an example of the calculation are shown.

Java 言語による荷電粒子追跡コードの開発

1 はじめに

加速器の設計・解析には、様々な計算コードが用いられている。これらのコード(プログラム)の配布は、ソースファイル形式とバイナリファイル形式のいづれかで行なわれている。著者の知る範囲では、ソース配布の例として FEL 解析コードの FRED、複数プラットホームのバイナリ形式で配布されている例として FEL 解析コードの GINGER、単一プラットホームのバイナリ形式で配布されている例として PARMELA、ソース・バイナリ両方の形式で配布されている例として SUPERFISH などがある。

一方、計算を実行する環境(プラットホーム)としては、UNIX と Windows(DOS を含む)が現在のところ広く用いられている。UNIX は、ベンダーや CPU が異なればバイナリファイルの互換性がなくソース配布が原則であるのに対し、Windows では、基本的に Intel CPU の使用が前提になっておりバイナリ配布が可能である。しかしながら、計算能力の高い DEC-Alpha CPU 上で走る Windows-NT では Intel 用のバイナリが動かないなどの問題もある。

このような実行ファイルの非互換性に起因するコード流通の障害を解決する方法として、Java 言語によるコード開発が考えられる。本稿では、著者らが開発中の Java による荷電粒子追跡コード(JPP = Java Particle tracking code Prototype)の概要と計算例について紹介する。

2 Java によるコード開発の利点と欠点

Java による計算コード開発の利点としては、優れた可搬性とモジュール管理・拡張の容易さがあげられる。一方、欠点としては、コンパイルされた実行プログラム(C や Fortran)に比べると実行速度が遅いことがあげられる。

2.1 可搬性

Java は、バイトコードとよばれる命令を記述したファイル(プラットホームに依存しない)を Java 実行系(仮想マシン)が翻訳しながら実行する。このバイトコード(実行ファイル)は、Java 実行系の動作するプラットホームであれば、どこでも動かすことができる。C や Fortran の実行ファイルはプラットホームに依存した機械語で記述されており互換性がないのに対して、Java はすぐれた可搬性を持っているといえる。これは、プログラムを流通させる上では大きな利点である。

2.2 モジュールの管理と拡張性

オブジェクト指向言語である Java を用いることで、加速器の構成機器(加速管、四重極磁石、偏向磁石など)に対応する計算ルーチンのモジュール化が可能になり、それぞれの計算ルーチンの開発と管理が容易となる。また、構成機器の追加や計算モデルの変更といった拡張も簡単に行なえる。

図1 は JPP に含まれるクラス一覧(モジュール)を示したものである。図に示したように、Java ではクラスに階層構造を持たせて継承関係を築くことができる。この階層構造と継承関係を使えば、加速器構成機器に対応する計算ルーチンを整理できる。つまり、これらの構成機器に共通の計算手続き(メソッド)を親クラスにまとめることにより、プログラム中で重複する部分を省くことができる。図2 には、加速器の構成機器を代表する親クラス(AccElement.class)と構成機器の例である偏向電磁石に対応するサブクラス(Bend.class)が図示されている。親クラス(Accelement.class)では、構成機器に共通的な3種類の計算手

続き (メソッド) が定義されている。

Inject メソッドでは、構成機器に粒子が入射した時の計算手続きであり、アパーチャによるスクリーニングなどが含まれる。AdvanceParticle メソッドは、与えられた時間ステップ分だけ粒子を進める処理である。Eject メソッドには、粒子が構成機器を出て次の構成機器へ進む時の処理、具体的には、粒子データの出力バッファーへの書き出しなどが含まれている。

それぞれの機器に対応するサブクラスでは、親クラスで定義されているメソッドを必要に応じて上書き (オーバーライド) すればよい。Bend.class では、磁石エッジによる粒子運動量の変化が Inject、Eject メソッドにそれぞれ追加されている。

また、Java では、変数の保護レベルを設定できることがモジュールどうしの干渉を防ぐのに役立っている。たとえば、加速空洞や粒子の位相計算に使われる変数 MasterClock (現在時刻を表し AccSystem.class 内で定義されている) の値はプログラム中のどこからでも参照できるが、値の変更はできないようになっている。このように、重要な変数は高い保護レベルに置くことでモジュールどうしの干渉による不用意な変数の破壊を防ぐことができる。保護レベルの概念は Java の大きな特徴である、

以上に述べたように、Java を用いることでモジュールの独立性と変数の適切な参照と保護が可能になり、加速器構成機器の追加やモデルの変更といったコードのメンテナンス・拡張が容易に行なえる。

2.3 実行速度

Java は、バイトコードを翻訳しながら実行するためにプラットホームに依存しない可搬性を実現しているが、この動作方法 (インタープリタ動作) は、コンパイルされた機械語プログラムを実行する場合に比べて、計算速度の点では不利になる。

表 1 は、C と Java の計算速度を比較するために、同一の計算量を実行するのに必要な計算時間の計測を行なった結果を示したものである。ここでは、1000 個の粒子数に対して Point-by-Point 空間電荷計算を 100 ステップまで行なった。なお、C プログラムは計算速度で最適化してコンパイルしたものであり、この CPU で得られるほぼ最高のパフォーマンスと考えられる。表に示したように、Java インタープリタ動作は、コンパイルされた C プログラムに比べて約 7 倍の計算時間が必要なことがわかった。なお、ここでは、Java 実行系として標準的な JDK-1.1.2beta を用いた。

ところで、Java プログラムの実行速度を向上させる方法として、「その場コンパイラ (just-in-time compiler: JIT)」の利用がある。JIT は Java プログラムの実行時にバイトコードを機械語に変換し、この機械語プログラムを走らせるものであり、原理的にはコンパイルされた機械語プログラムと同等のパフォーマンスが得られる。フリーの Java 実行系である Kaffe の JIT 機能を使って計算速度の向上を試みたところ、JDK のインタープリタ動作に比べると約 2 倍の速度向上が得られた (表 1)。

今後、各社から高性能の JIT が提供されれば、Java プログラムの実行速度の向上が可能であろう。(例えば最適化した C 実行ファイルの 40% の速度が得られたという報告もある [1])

3 計算例

JPP を使った計算の一例として、エミッタンス増加 (linear emittance growth) の計算結果を示す。linear emittance growth は、バンチビームが直線ドリフトを通過する際にビーム半径が変化しないパラメータ領域で起こるエミッタンス増加であり、理論解が求められている [2]。この理論解と計算結果を比較することで、計算コードの妥当性が検証できる。図 3 に JPP (Point-by-Point 空間電荷計算ルーチンを使った場合)、PARMELA による計算結果を理論解とあわせて示した。JPP では、乱数で生成する粒子の初期分布の違いによる計算結果のばらつきをエラーバーで示している。$z > 60cm$ ではビーム半径の増大が始まるので、linear emittance growth の領域を逸脱し、計算結果は理論解から少しずつ離れているが、$z < 60cm$ の範囲では計算結果は理論解とよく一致していることがわかる。

4 今後の課題

今後の課題として、入出力インターフェースの改良とマルチスレッドの利用による計算速度向上があげられる。現在、入力は PARMELA と同じ形式の入力ファイルを用い、出力はファイルに書き出されたデータを既存のグラフィックツールで表示している。Java Applet ブラウザを用いた入力方法と Java の グラフィックス API を用いたグラフ表示を組み込めば、入出力インターフェースまで含めた可搬性の高い計算コードが完成するであろう。また、Java のマルチスレッド機能を利用すれば、マルチ CPU のマシン上では複数の CPU を並列に使用して計算速度を向上させることが可能である。

5 まとめ

Java による荷電粒子追跡コード JPP の概要を述べた。コンパイルされた C や Fortran プログラムと比較すると、Java による数値計算は計算速度の点で劣るが、すぐれた可搬性はコードの流通を容易し、独立したモジュールを構成することができる点はコードを開発する上で大きな利点である。

参考文献

[1] 「Java 高速実行環境: HGC」、FUJITSU 1997 年 3 月号

[2] M.E.Jones and B.E.Carlsten, in Proc. 1987 Particle Accelerator Conf., p.1319.

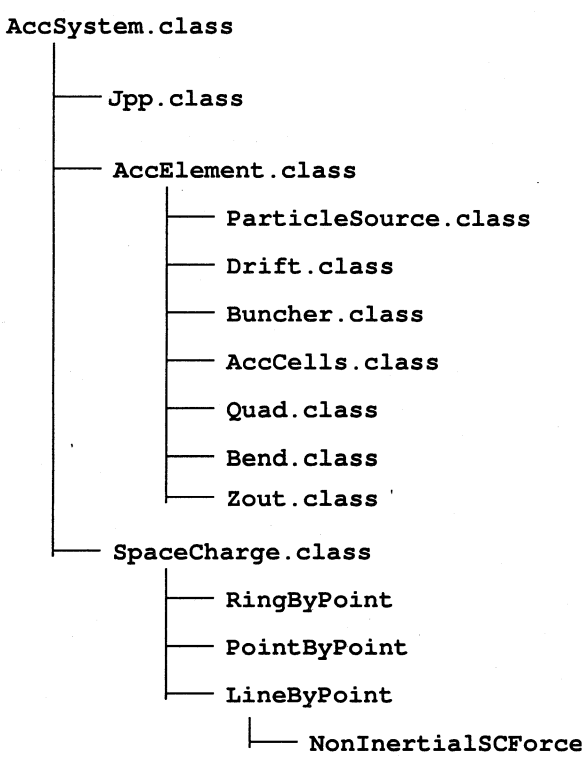

図 1. JPP におけるクラス階層

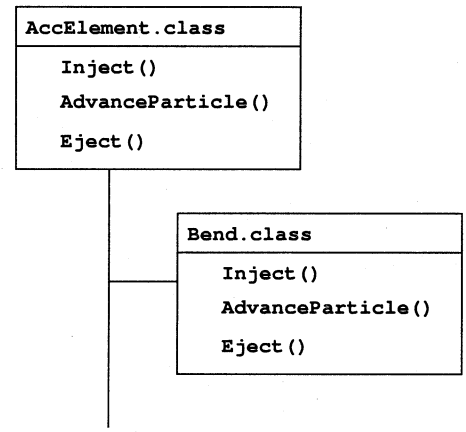

図 2. AccElement.class とサブクラスの継承関係

表 1. Java と C の計算時間の比較 (数字は秒)

ステップ数	C	Java(JDK)	Java(Kaffe)
50	38	274	134
100	76	549	269

実行環境：Solaris 2.5.1 / PentiumPro 200MHz
C: PC3.0.1、JDK: JDK-1.1.2beta、
Kaffe: Kaffe-0.90

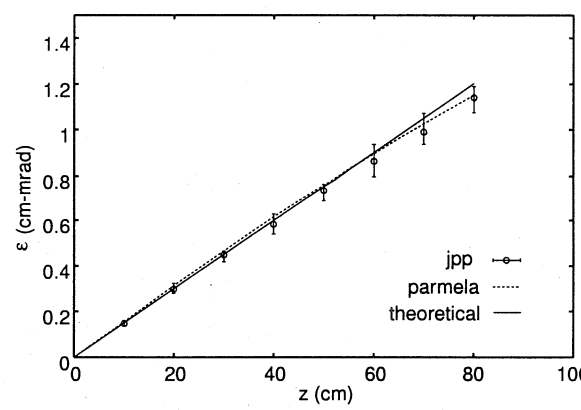

図 3. エミッタンス増加の計算結果
E=1.022(MeV)、I_p=11.8(A)、r=1(cm)