2002年 12月 17日
SPLICE数値演算講演会
@ 宇宙開発事業団

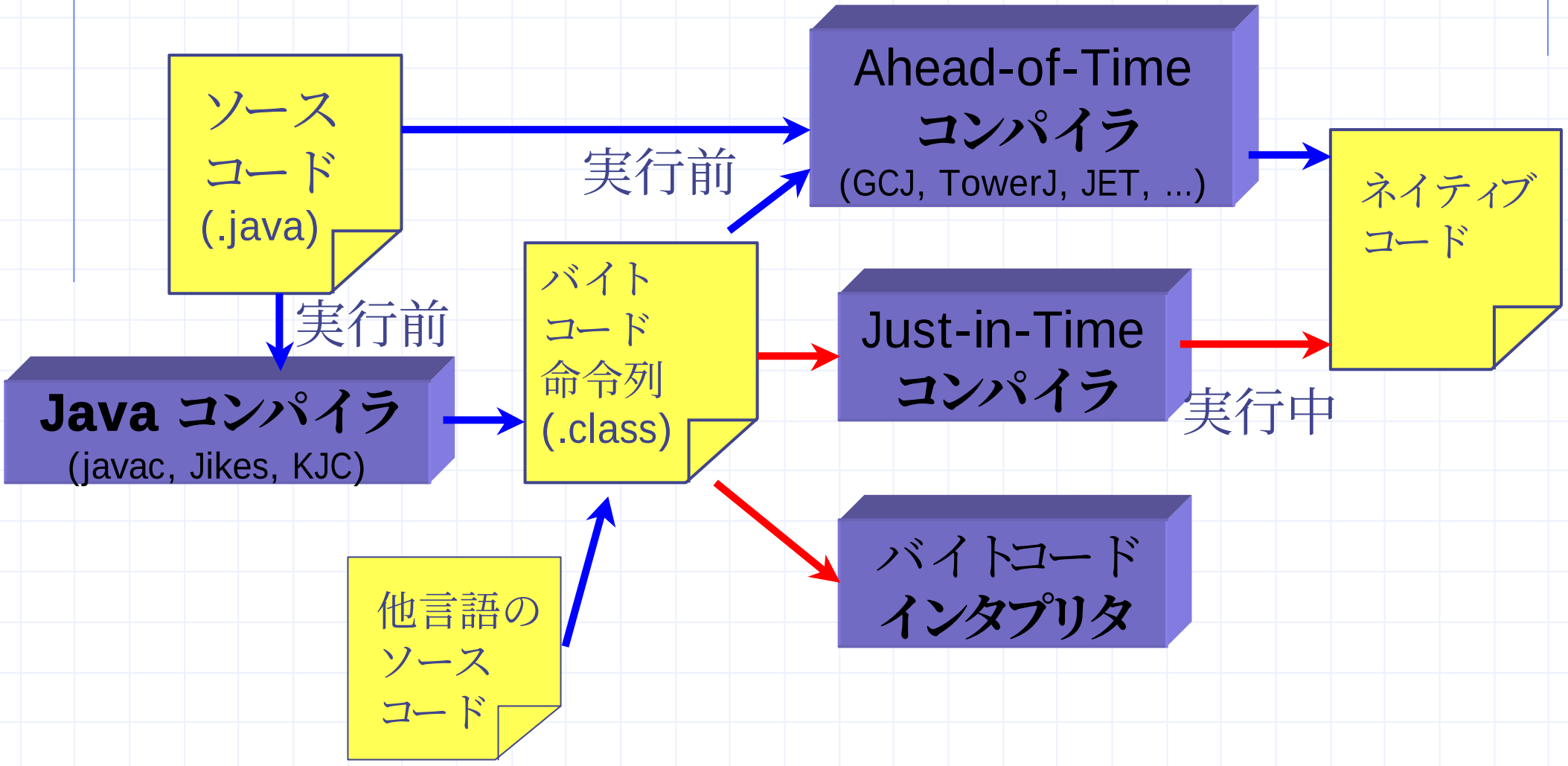
Numerical Computing in Java
ー 演算結果の再現性と性能

首藤 一幸
産業技術総合研究所
グリッド研究センター

自己紹介

- 分散処理, 高性能計算, Java
  - 分散・並列化コンパイラ（1995～1997）
    - ネットワーク接続されたWS/PCを対象に、逐次プログラムを分散。
  - Java仮想マシン間のスレッド移送（1997）
    - ネットワーク越しに、マシン間で実行状態を移送。
  - Java Just-In-Timeコンパイラ（1998～）
  - ネットワーク透過な分散オブジェクトシステム（1998）
    - 遠隔、ローカルの区別なく、オブジェクトを扱える。
  - 浮動小数点演算セマンティクスの効率的な実装（1998～）
  - 組織内、インターネット上の分散計算ミドルウェア（2002～）
- 暗号, 情報セキュリティ
  - 共通鍵暗号方式の、Javaによる高性能実装（1998）
  - 共通鍵暗号方式の強度解析（1995）

Javaプログラムの実行



Javaによる高性能計算の問題

- 実装（JITコンパイラ, インタプリタ）の性能が低かった。
  - ピーク性能はC, Fortranに近づいてきた。
- ハードウェアの性能、機能を搾り尽くす手段がない。
  - PowerPC の fused multiply-add （FMA）を使えない。
    - もし使うと、Java 言語仕様が定める演算のセマンティクスを守れない。
  - 丸めの方向を制御できない。
    - Javaでは、浮動小数点数の表現形式は IEEE 754。
    - IEEE 754は、最近値への丸めだけでなく、+∞, -∞, 0 方向という3通りの丸め方向の提供を義務付けている。
  - 浮動小数点演算の例外でトラップを発生させる方法がない。
    - 例外：invalid operation ((+∞)+(-∞), 0×∞など), divide by zero, overflow, underflow, inexact
    - とはいえ、発生するように設定する方法は、C でも環境依存。場合によっては要アセンブリコード。

Javaによる高性能計算の問題

言語仕様

- 配列アクセス時に境界を越えていないかのチェックが必要。
  - (new int[10])[15] …例外発生
- 2次元配列が、1次元配列の配列である。
  - C言語などとは違い、アドレス計算→アクセス、では済まない。
  - Jose Moreira (IBM) らが改善案を提案中。
    - ライブラリ（例: doubleArray2Dクラス）+ syntax sugar（a[i,j] という表記をライブラリ呼び出しに変換する）
  - C#言語 (Microsoft) には、Javaと同じjagged arrayに加えて、rectangular arrayがある。
    - jagged array: a[i][j], rectangular array: a[i, j]
- 例外をthrowし得る処理を越えて、命令の実行順を入れ替えられない。
  - コンパイラによる最適化が制限される。
- 演算子オーバーロードができない。
  - 複素数の演算も、a.multiply(b) と書かざるを得ない。

Javaによる高性能計算の問題

- 言語仕様 (cont'd)
  - 浮動小数点演算のセマンティクスを厳密に定めている。
    - Java 2 より前の規定：
      「演算の中間結果も、厳密に指数部 11ビット、仮数部 52ビット。」
    - Intel社 x86プロセッサ上で律儀に実装すると、数値計算が数倍以上遅くなってしまう。

- キーワード strictfp が Java 2 から導入された。
  - プログラム中で strictfp 指定がない箇所でのセマンティクスが緩められた。
    - Java 2より前は、常に strictfp の厳しいセマンティクスで演算する必要があった。
    - x86でもオーバーヘッドがなくなった。
      注）ただし単精度数の演算は依然、要注意。きちんと実装すると、SSE, SSE2命令群を使わない限り、オーバーヘッドを避けられない。

strictfp とは

- 文法
  - クラス、メソッド、インタフェースの修飾子。
    - 例) `strictfp void aMethod(…)  { …`
  - static, lexical スコープ。
- 意味
  - 言語仕様が定めるセマンティクス (FP-strict) に厳密に沿わねばならない。
    - SPARC, Alpha, PowerPC, MIPS と同じセマンティクス。
  - 逆に、strictfp ではない文脈では、もう少し、緩くてよい。
- x86 プロセッサ(x87命令)は沿っていない。
  - x86 用の Java 仮想マシンは実装を工夫する必要がある。
  - Sun社, IBM社は Java 2 SE 1.4 でようやく実装した。

演算結果が異なる 実例

- FP-strict か否かによって、結果が異なる。
- SPARC等では常に `strictfp’ と同じ結果。
  - 特に、実装上の工夫は要らない。

```
% java StrictfpTest
   shuJIT  for Sun Classic VM/x86  by Kazuyuki Shudo
1.112808544714844E-308 (0x0080080000000000)
   * 1.0000000000000002 (0x3ff0000000000001)
default : 1.112808544714844E-308 (0x8008000000000000)
strictfp: 1.112808544714847E-308 (0x8008000000000001)

2.2250738585072014E-308 (0x000fffffffffffff)
   / 0.9999999999999999 (0x3feffffffffffffe)
default : 2.2250738585072014E-308 (0x1000000000000000)
strictfp: 2.2250738585072014E-308 (0xffffffffffffffff)
```

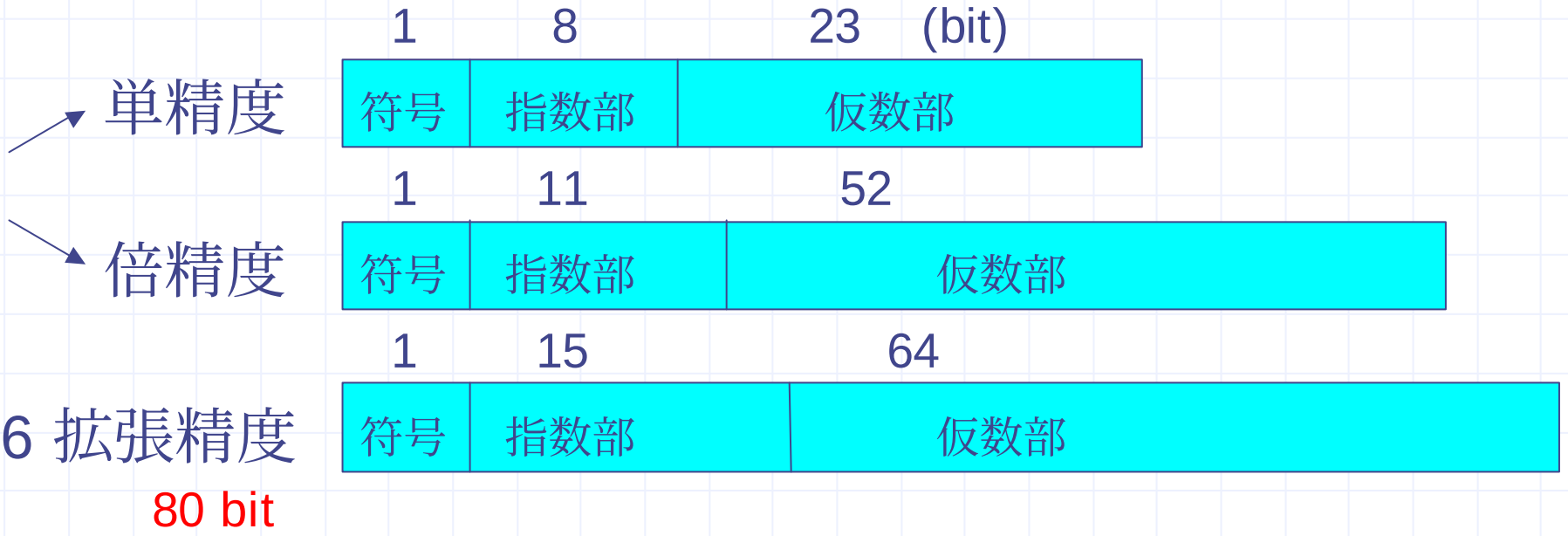
異なる

異なる

x86 の仕様

- 浮動小数点数のレジスタは 80 bit 長。
  - 64 bit でも 32 bit でもない。
- 精度に応じた演算命令を持たない。
  - cf. fmuls と fmuld (SPARC), mul.s と mul.d (MIPS)
  - 仮数部は丸め精度の設定に応じて丸められる。

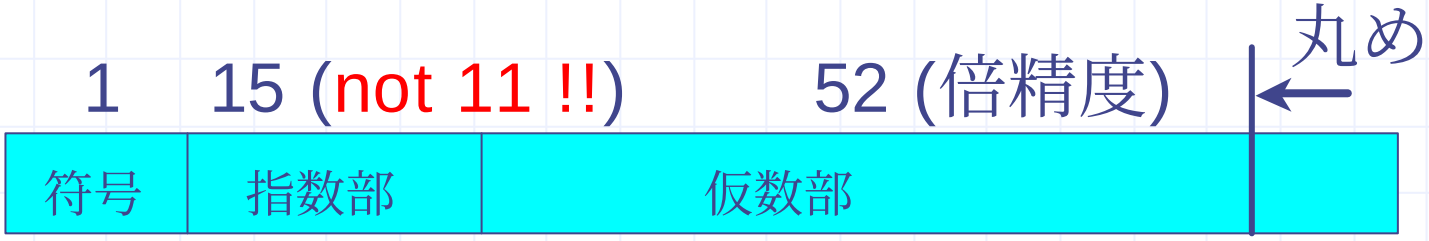
strictfp が要求する形式



x86 の仕様

- 指数部は常に 15 bit (拡張精度) !!
  - cf. 単精度 8 bit、倍精度 11 bit。

x86 にて、丸め精度を倍精度に設定した場合



- IEEE 754 規格はこれを許容している。
  - x86 は IEEE 754準拠。

問題となる場合

- 指数部が常に 15 bit なので
  起こるべき溢れが起きない。

$(-1)^{S} \cdot 2^{E} \cdot 1.\ldots\ldots$ として

	E のビット数	E の範囲
単精度	8	-126 ～ +127
倍精度	11	-1022 ～ +1023
x86拡張精度	15	-16382 ～ +16383

- 次の倍精度演算の結果が …

レジスタ　= 2^{+1023}
レジスタ += 2^{+1023}
レジスタ += 2^{+1023}

- SPARC では　$+\infty$
- x86 では　2^{+1023}

- アンダーフローでは、二度丸めの問題も…

「strictfp を実装する」とはどういうことか？

- Javaコンパイラ
  - Java言語で書かれたソースコードをJavaバイトコード（クラスファイル）に変換する。
    - javacコマンド, Jikes, KJC など
  - キーワード“strictfp” を認識して、クラスファイルにその旨を反映させる。
    - 容易。
  - コンパイル時定数は、FP-strict に演算する必要がある。
- 実行系
  - Javaバイトコードを解釈、実行するか、ネイティブコードにコンパイルする。
    - インタプリタ, Just-In-Timeコンパイラ, Ahead-of-Timeコンパイラ
  - strictfp だという印のついたメソッドについて、FP-strict のセマンティクスで演算を行う。
    - x86 では工夫が要る。性能上のオーバーヘッドがある。
      - もし整数演算でエミュレートしたら、非常に遅くなるだろう。
    - Golliver（Intel）は、効率良い方法を提案し、2～4倍の演算時間、と予想した。
    - 首藤らは、x86での効率良い実装方法を調べた。

strictfp の実装状況

- Sun社, IBM社は、Java 2 SE 1.4 でようやく実装した。
  - strictfp の仕様自体は1.2 (Java 2)で導入されたもの。
  - Javaコンパイラ (javac)はstrictfp対応していたが、Java仮想マシン (JITコンパイラ, インタプリタ)が未対応だった。
- ごく一部の実行系、コンパイラは、以前より実装していた。
  - shuJIT
    - 首藤らによるJITコンパイラ
  - BulletTrain
    - NaturalBridge社によるAhead-of-Timeコンパイラ

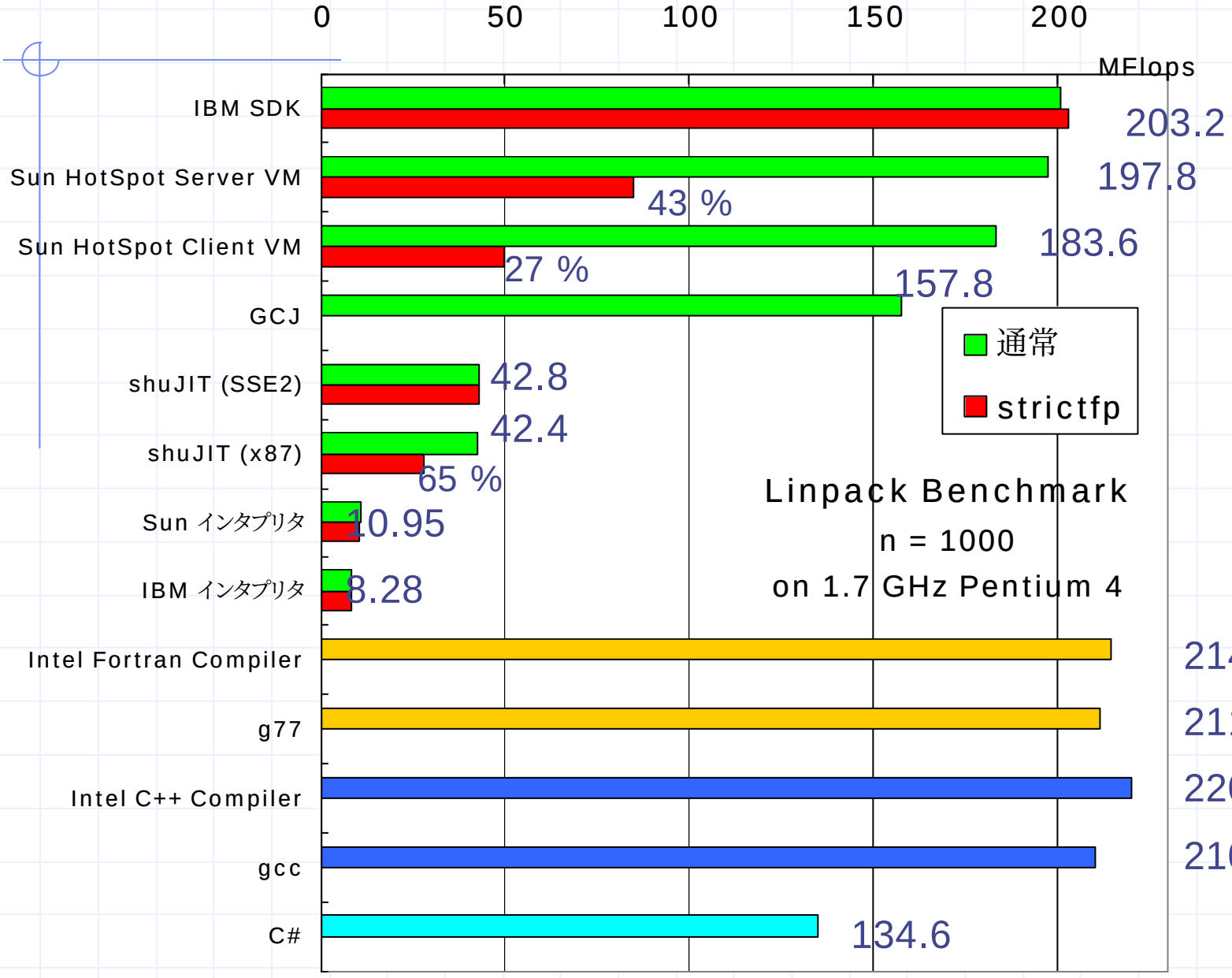
性能評価

- 目的
  - strictfp のオーバーヘッドを知る。
    - FP-strictな演算
  - では、そもそも、JavaはCやFortranとどのくらい性能が違うのか？
- 評価方法
  - Linpack ベンチマーク
    - 連立一次元方程式を解く。密行列。
    - naive なコード。
      - ベンチマークプログラムには変更を加えない。
        変更するのは問題サイズだけ。
      - ブロッキングなど、通常は必須である最適化も行わない。
    - C# 版は、Java 版を移植したもの。
- 環境
  - 1.7 GHz Pentium 4 / Linux
    - C# は、Windows 2000 SP3
  - 800 MHz Pentium III / Linux
    - C# は、Windows .NET Server 2003 RC1

Linpackベンチマーク

1.7 GHz Pentium 4

- キャッシュ: L2 256 KB, L1 8KB
- IBMのJITコンパイラ、インタプリタは、SSE2命令（後述）を使っている？



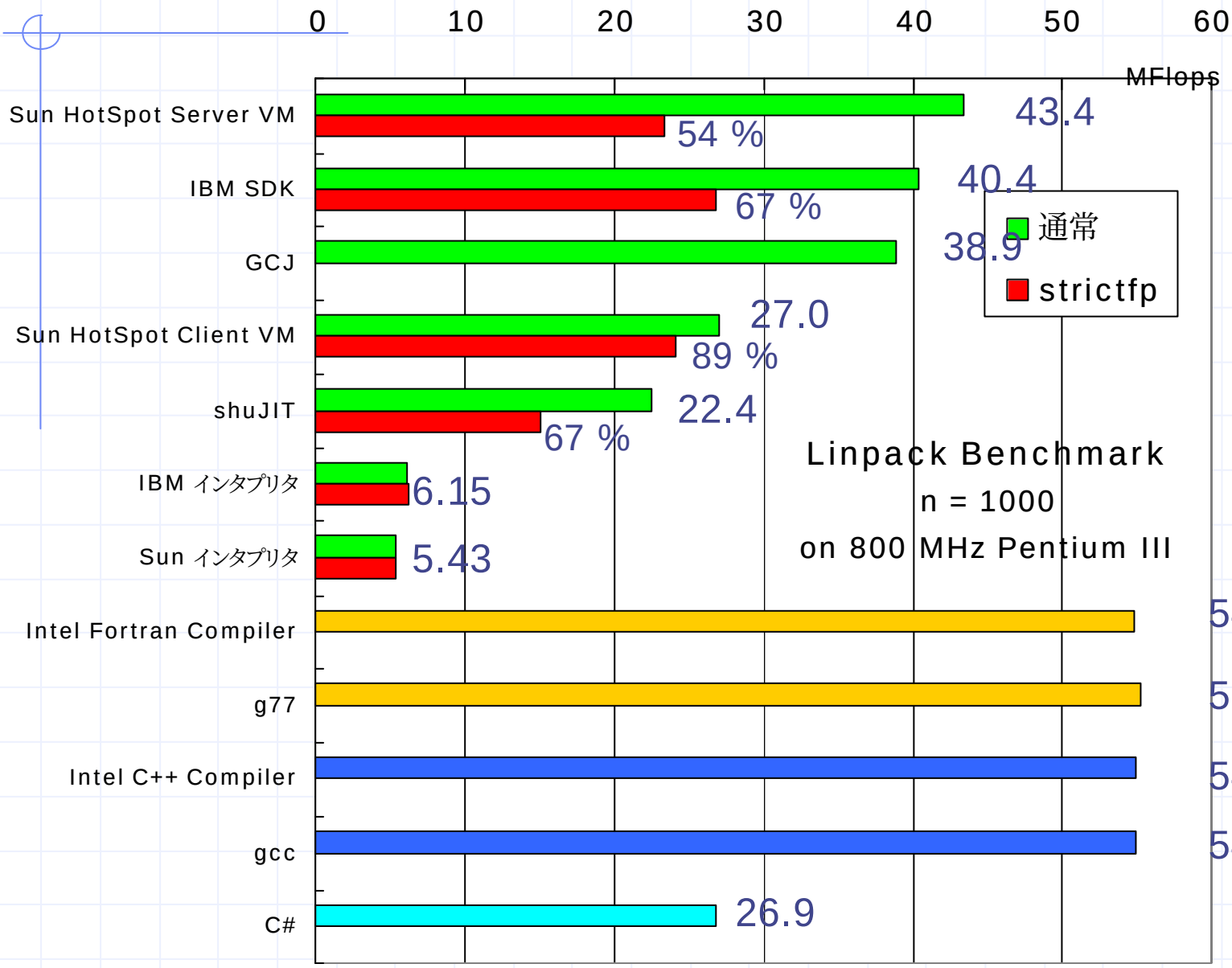
参考

n=100, Intelコンパイラ

FORTRAN (SSE2命令)
813.8 MFlops
FORTRAN (x87命令)
425.9 MFlops
C (SSE2命令)
757.1 MFlops
C (x87命令)
426.2 MFlops

Linpackベンチマーク
800 MHz Pentium III

•キャッシュ: L2 256 KB, L1 16 KB



参考
n=100, Intelコンパイラ

FORTRAN 344.4 MFlops
C 333.7 MFlops

FORTRAN 77

C

C#

SSE2 命令

- 実は… SSE2命令は FP-strict に演算を行う。
  - x87命令のような変態 (80 bit の拡張形式) ではない。
  - 精度ごとの演算命令を持つ。

- SSE2：Streaming SIMD Extensions 2
  - Pentium 4 （2000年末出荷）で追加された命令群。
  - SIMD演算用の命令セットでありながら、
    実は浮動小数点演算全般をサポート。
    - IEEE 754準拠 (丸め制御, 例外, 平方根, …)。
    - x87 とは異なり、超越関数の命令は持たない。

- 何ら工夫は要らない。

まとめ

- Java言語仕様は、C, Fortran などと比べて確かに厳しい。
  - 結果の再現性を重視。
  - 性能の追求や、コンパイラによる最適化を妨げる規定もある。
  - しかし、最近の実行系のピーク性能は、C, Fortran に迫っている。
    - Just-in-Timeコンパイラが、実行時に、頻繁に実行されるメソッドに限って、最適化する。
  - Javaを使うメリット:
    - クラスファイルが機種非依存。
    - 実行結果の再現性が高い。異機種間であっても。
    - 標準クラスライブラリが充実している。
    - 開発、デバッグの労力が比較的低い。
      - ポインタを数値として扱えない、配列の境界チェックがある、…

参考文献

- Validgh Java numerics page
  - http://www.validlab.com/java/
  - “How Java’s Floating-Point Hurts Everyone Everywhere” by W. Kahan
- Java Numerics
  - http://math.nist.gov/javanumerics/
  - Javaを用いた数値計算についての情報源。Java Grande ForumのNumerical WGの打ち合わせ記録など。
- Java Grande Forum
  - http://www.javagrande.org/
  - “Javaと高性能計算” に取り組んでいる集まり。
- Performance comparison of JITs
  - http://www.shudo.net/jit/perf/
  - 各種Java仮想マシン、JITコンパイラの性能評価。
- The Java Performance Report
  - http://www.javalobby.org/members/jpr/
  - 要ユーザ登録 & ログイン。
  - PART III　$ep 2000）ではCとの比較も行っている。
- Languages for the Java VM
  - http://grunge.cs.tu-berlin.de/~tolk/vmlanguages.html