Symbolic Computation in Java: an Appraisement

Laurent Bernardin
ETH Zurich
www.inf.ethz.ch/personal/bernardi

Joint work with: Bruce Char (Drexel University)
Erich Kaltofen (North Carolina State U)
Java advantages

- GUI
- Distributed computing (Schreiner 99, Masdis)
- Visual programming (MathBeans)
- Black-box functions (Sandbox, Melissa)
- Serialization standard (PDG Openmath)
- Standard libraries
- Platform independence
Is Java fit for large-scale computing?

- Large and complex programs
- High performance calculations
Component-wise system design

Plug-and-play components (conglomerates, PSEs)

- Maple
- Mathematica
- Application Program

"plug-and-play" software

"middle-ware" implementation of new algorithms

- NTL
- SAC Lib
- Linpack

"generic" programming
Interfaces between Java/non-Java components

1. Java
   native methods

2. Non-Java
   Java VM / JIT Compiler
   Norman-Fitch 1996

3. Java
   sockets, pipes, CORBA

Non-Java
Generic Programming

- static (compile-time) binding: templates (C++, GJ, NextGen)
  performance

- dynamic (run-time) binding: interfaces (Java)
  type-safety

- C++-style templates: GJ/NextGen
  Are templates really needed?

- STL-style allocators as interface “glue” for divergent storage models

- Algorithmic shortcuts into the basic modules
  partial template specialization
public interface Ring {
    public interface Element {
        public boolean iszero();
        ...
    }
    public Element fromInteger(int n);
    public Element add(Element a, Element b);
    public Element multiply(Element a, Element b);
    ...
}

public class DensePolynomial implements Ring {
    public class Element implements Ring.Element {
        private Ring.Element[] _coeffs;

        public int degree() {
            return _coeffs.length - 1;
        }
        ...
    }// end class Ring.Element

    private Ring _R;

    public Element add(Ring.Element a, Element b) { ... }
}
Ring R = new Zmod(17);
Ring P = new DensePolynomial(R);
DensePolynomial.Element p = P.add( P.monomial(10),
                               P.one() );
DensePolynomial.Element q = P.add( P.monomial(3),
                               P.one() );
p = P.multiply(p,q);
IN 21 DAYS
JAVA BOOK
Teach Yourself to Write a
Java Program without Experience Required

CD-ROM

COMPLETELY REVISED AND UPDATED FOR JAVA 1.7

Kevin McCurley
Performance considerations

- Garbage collection
- Interpreted bytecode / JIT
- Benchmarks
In-Place Polynomial Arithmetic

- Multiply degree $n$ polynomials over $\mathbb{F}_{17}$
- Dense array of hardware integers
- Java: Sun JDK 1.2 (beta 5)
- Maple: modp1 datastructure, R5
- C: Sun Workshop 4.2, flags: -O
- C*: flags: -native -fast -xO4

<table>
<thead>
<tr>
<th>$n$</th>
<th>Java</th>
<th>Maple</th>
<th>C</th>
<th>C*</th>
<th>Java vs. C*</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>7s</td>
<td>9s</td>
<td>9s</td>
<td>3s</td>
<td>2.33</td>
</tr>
<tr>
<td>20000</td>
<td>30s</td>
<td>37s</td>
<td>36s</td>
<td>13s</td>
<td>2.31</td>
</tr>
<tr>
<td>30000</td>
<td>69s</td>
<td>77s</td>
<td>82s</td>
<td>31s</td>
<td>2.23</td>
</tr>
<tr>
<td>40000</td>
<td>124s</td>
<td>137s</td>
<td>146s</td>
<td>56s</td>
<td>2.21</td>
</tr>
<tr>
<td>50000</td>
<td>196s</td>
<td>218s</td>
<td>231s</td>
<td>89s</td>
<td>2.20</td>
</tr>
</tbody>
</table>
Generic Polynomial Arithmetic

- Multiply degree $n$ polynomials over a generic ring
- Particular ring is $\mathbb{F}_{17}$ again
- Aldor: $1.1.10b + \sum^{it}$
- C++: gcc 2.8.1, templates

<table>
<thead>
<tr>
<th>n</th>
<th>Java</th>
<th>Aldor</th>
<th>C++</th>
<th>Java vs. C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2.6s</td>
<td>1.2s</td>
<td>0.9s</td>
<td>2.8</td>
</tr>
<tr>
<td>2000</td>
<td>9.1s</td>
<td>4.3s</td>
<td>3.6s</td>
<td>2.6</td>
</tr>
<tr>
<td>3000</td>
<td>20s</td>
<td>10s</td>
<td>8s</td>
<td>2.5</td>
</tr>
<tr>
<td>4000</td>
<td>36s</td>
<td>19s</td>
<td>14s</td>
<td>2.6</td>
</tr>
<tr>
<td>5000</td>
<td>57s</td>
<td>30s</td>
<td>22s</td>
<td>2.6</td>
</tr>
<tr>
<td>6000</td>
<td>82s</td>
<td>42s</td>
<td>31s</td>
<td>2.6</td>
</tr>
<tr>
<td>7000</td>
<td>111s</td>
<td>56s</td>
<td>42s</td>
<td>2.6</td>
</tr>
<tr>
<td>8000</td>
<td>146s</td>
<td>72s</td>
<td>57s</td>
<td>2.6</td>
</tr>
</tbody>
</table>
Conclusions

- Java as component glue and for GUI code

- Performance is improving (Compiler technology)

- Template facilities are being missed