Using AspectJ to Separate Concerns in Parallel Scientific Java Code

Bruno Harbulot
bruno.harbulot@cs.man.ac.uk

John R. Gurd
jgurd@cs.man.ac.uk

Presentation Outline

I. Performance as an Aspect

II. Code-tangling in scientific software

III. Aspects for the Java Grande benchmarks

IV. Abstraction and OO model for loops

V. Conclusions

Performance as an Aspect (I)

- Blue-sky situation:
  - “Wherever *performance can be improved*, do *improve performance*.” (inspired from Filman & Friedman)
  - Not ready yet...
- Published examples of aspects for performance:
  - rely on languages like Lisp or specifically-created languages, or
  - coarse-grained caching or profiling

Performance as an Aspect (II)

- AspectJ:
  - expects underlying object-oriented design,
  - works mostly on object interfaces (method calls and field accesses),
  - cannot recognise and intervene on loops.
- Few scientific object-oriented designs.

Performance as an Aspect (III)

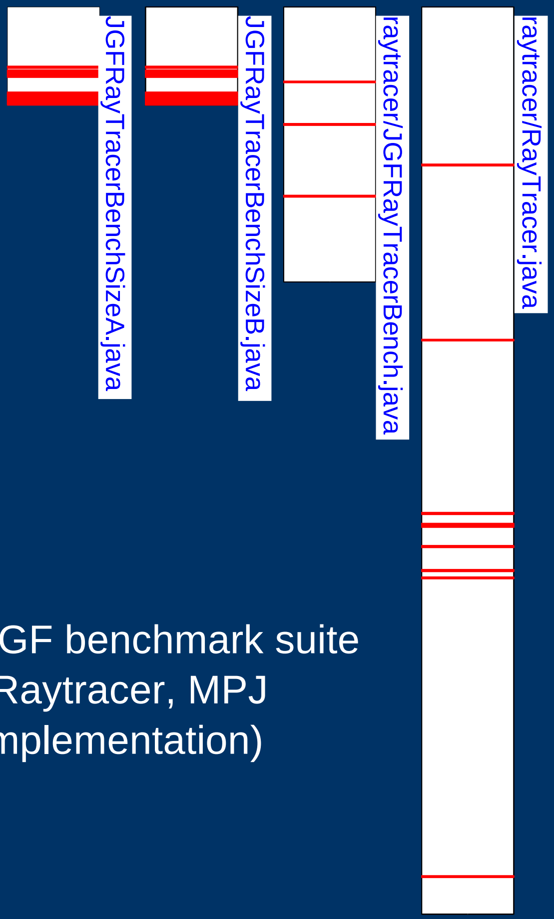
- Which are the points where to intervene? (for example, around loops)
- How to specify and recognise these points? (Abstraction for the aspects)
- How to represent the original algorithm? (Abstraction of the numerical algorithm)

Two representations for matrix multiplication:

```
C=A*B
```

```
for (i=0 ; i < n ; i++)
  for (j=0 ; j < n ; j++) {
    c[i][j] = 0 ;
    for (k=0 ; k < n ; k++)
       c[i][j] = c[i][j] + a[i][k]*b[k][j] ;
  }
```

Code-tangling in Scientific Software



JGF benchmark suite (Raytracer, MPJ implementation)

- Statements for parallelism using MPJ (aka MPI for Java), Java Threads, or OpenMP are tangled within the numerical algorithm.
- The parallelisation concern is spread across several files and cannot be encapsulated in its own entity.
- Problem for readability and reusability.

Java Grande benchmarks

- Numerical applications in three “flavours”:
  - sequential implementation,
  - parallelised using MPJ (MPI for Java),
  - parallelised using Java Threads.
- AspectJ for encapsulating each parallelisation scheme, optionally woven into sequential code.

Java Grande Benchmarks (I) Minor modifications

- Exposing the iteration space in the interface
- ```
  myMethod (...) {
    for (int i=0 ; i<text.length ; i++)
       {...} }
  ```
- ```
  myMethod (..., int iMin, int iMax) {
    for (int i=iMin ; i<iMax ; i++) {...} }
  ```
- Possible to keep default behaviour by overriding
- Aspects can intercept original calls and create sub-calls within several threads

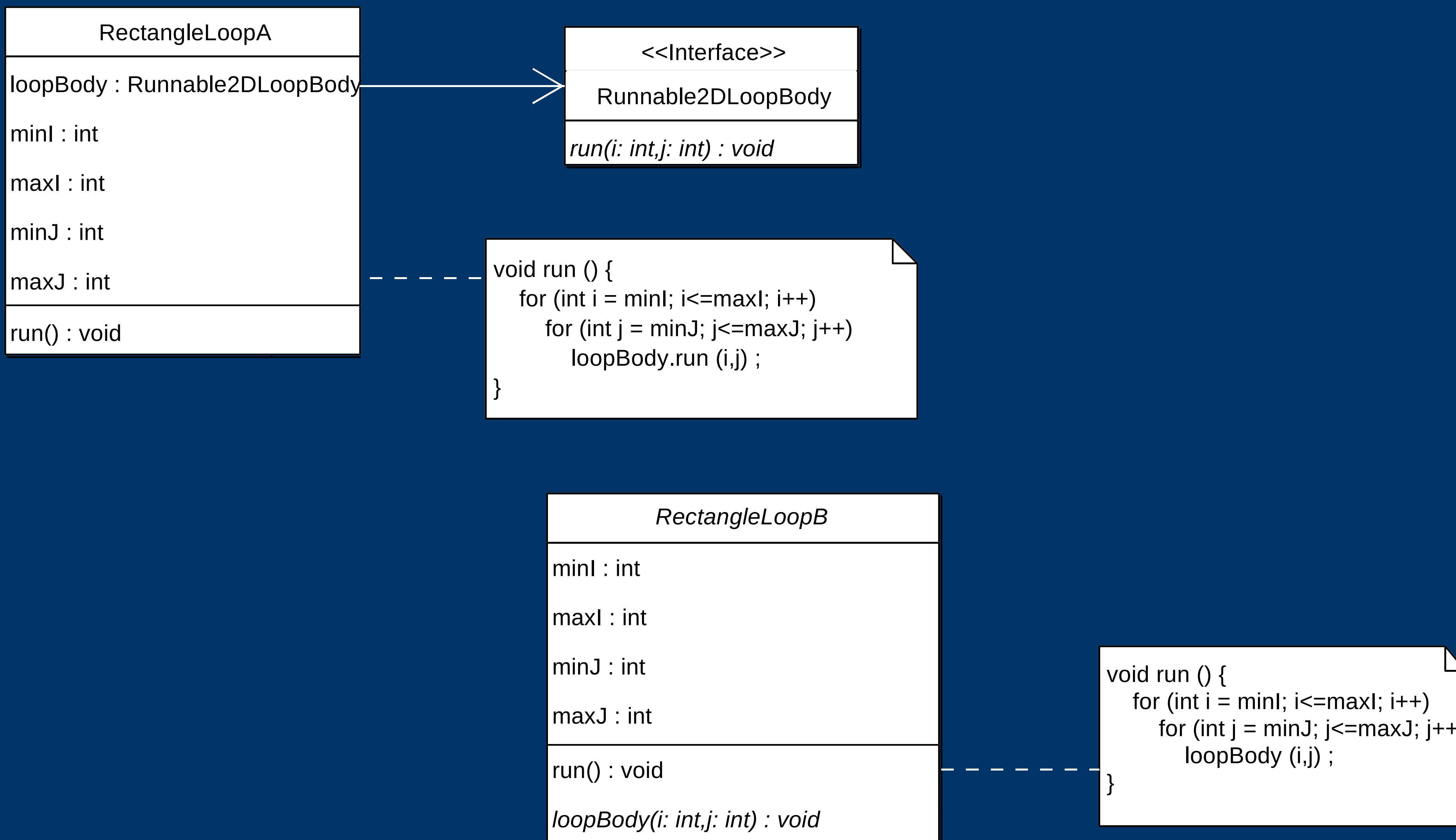
Java Grande Benchmarks (II) Major modifications

- Fortran subrout. -> C functions -> Java methods
- Imperative style of programming
- Using arrays directly, without object information (such as the length)
- Sequence of operations not encapsulated in meaningful and identifiable units
- Not compatible with AspectJ's join-point philosophy

Object-Oriented model for Loops

- Object-Oriented models for “for”-loops.
- AspectJ can handle these models.
- Consists of encapsulating loop information into objects: boundaries and loop-body.
- (at the moment, only embarrassingly-parallelisable loops)

Object-Oriented model for Loops



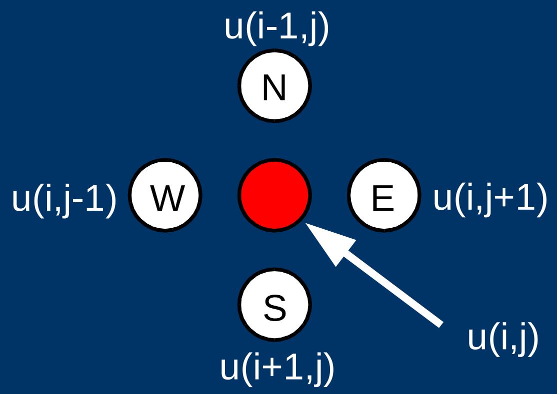
Object-Oriented Loops: example

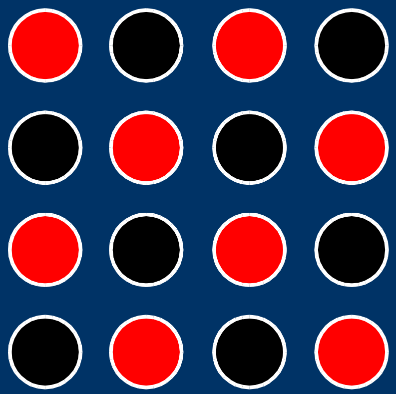
- ```
  for (int i=1; i<=N; i++)
    for (int j=1; j<=(N/2); j++){
       int jtemp = 2*j - (i%2);
       u[i][jtemp] += ... ;
    }
  ```
- ```
  final class RedLoopBody implements Runnable2DLoopBody {
    final private double u[][];
    final private double omega;
    FinalRedLoopBody(double[][] u, double omega) {
       this.u = u;  this.omega = omega;
    }
    public final void run(int i, int j) {
      int jtemp = 2 * j - (i % 2);
      u[i][jtemp] += ... ;
    }
  };
  ```

```
Runnable2DLoopBody redLoopBody = new RedLoopBody(u, omega);
RectangleLoopA redLoop = new RectangleLoopA(redLoopBody,1,N,1,N/2);
redLoop.run();
```

Object-Oriented Loops: Overheads

- Tests on Red/Black SOR algorithm
- Alternative iterations on all red points and on all black points until convergence



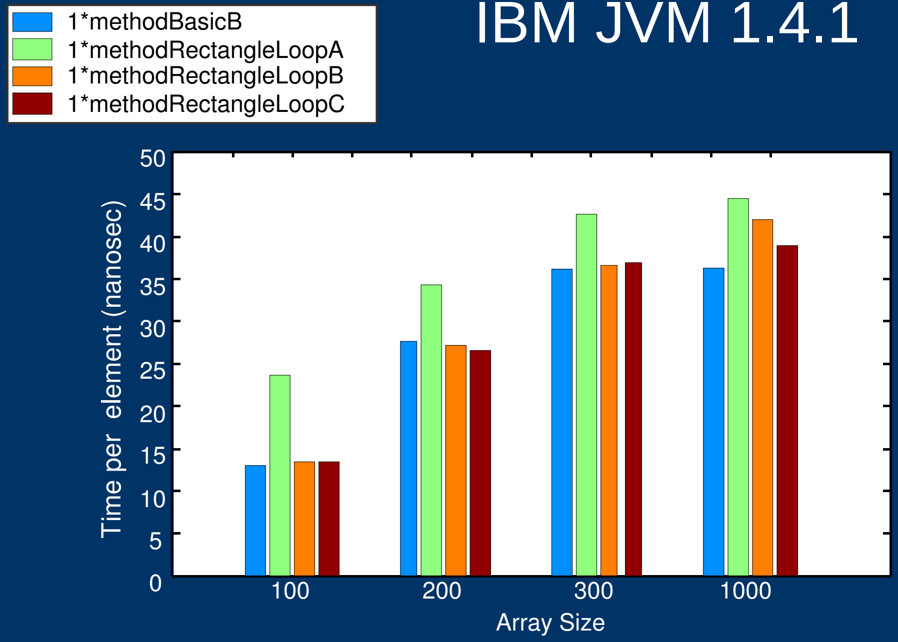


$$u_{(i,j)} = \ldots * (u_{(i-1,j)}+u_{(i+1,j)}+u_{(i,j-1)}+u_{(i,j+1)})$$

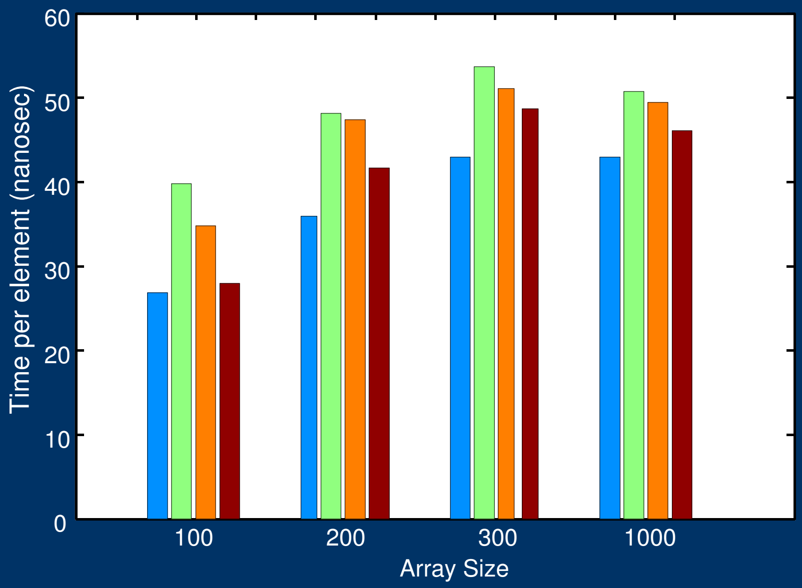
Object-Oriented Loops: Overheads

- Performance results depend on the JVM (IBM/Linux > Sun/Linux > SGI)

IBM JVM 1.4.1



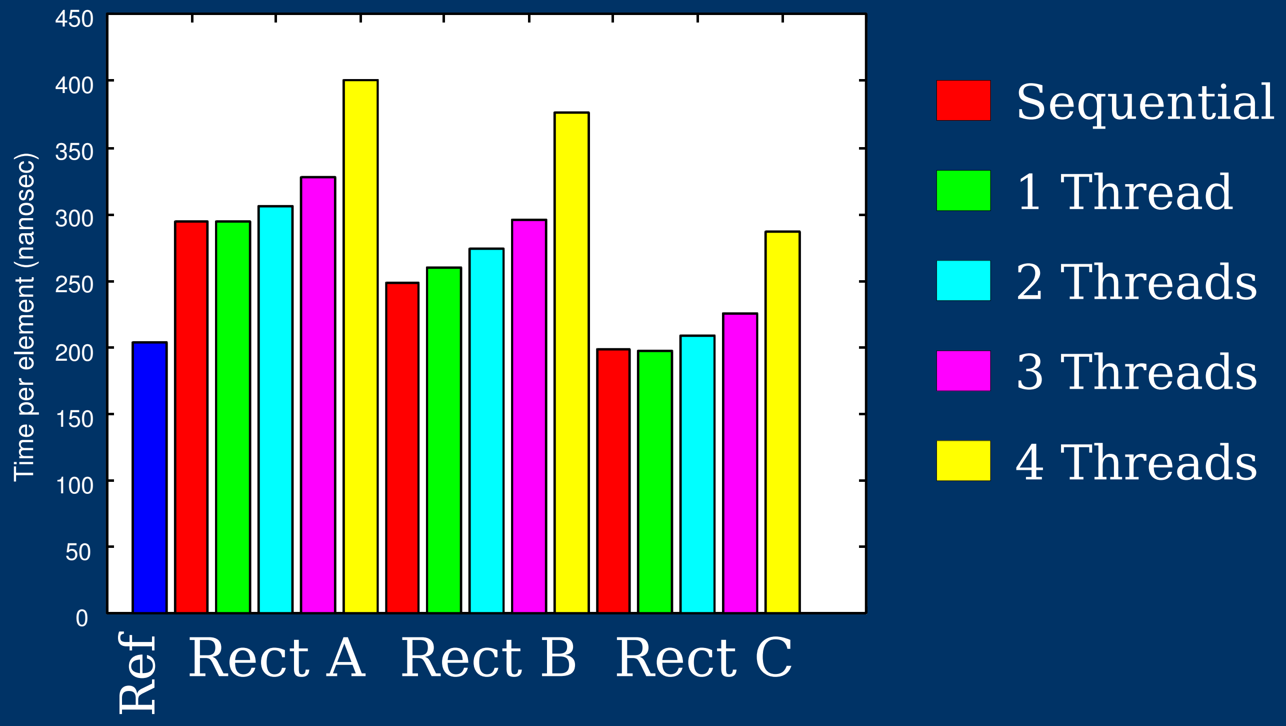
Sun JVM 1.4.2



Object-Oriented Loops: Parallel results using aspects

- Tests on 4-processor SunOS machine.
- Significant overhead when all processors used.
- Competition with GC or JIT.

Sun JVM 1.4.2
(array size: 200)



Conclusions

- Aspect-Oriented Parallel Code possible with:
  - Appropriate abstraction
  - Means to recognise what can be parallelised
- Current lack of object-oriented design in scientific software
- Performance results promising