Compiling NESL into Java

Jonathan Hardwick, Girija Narlikar
Carnegie Mellon University

Jay Sipelstein
Susquehanna Investment Group

Motivation

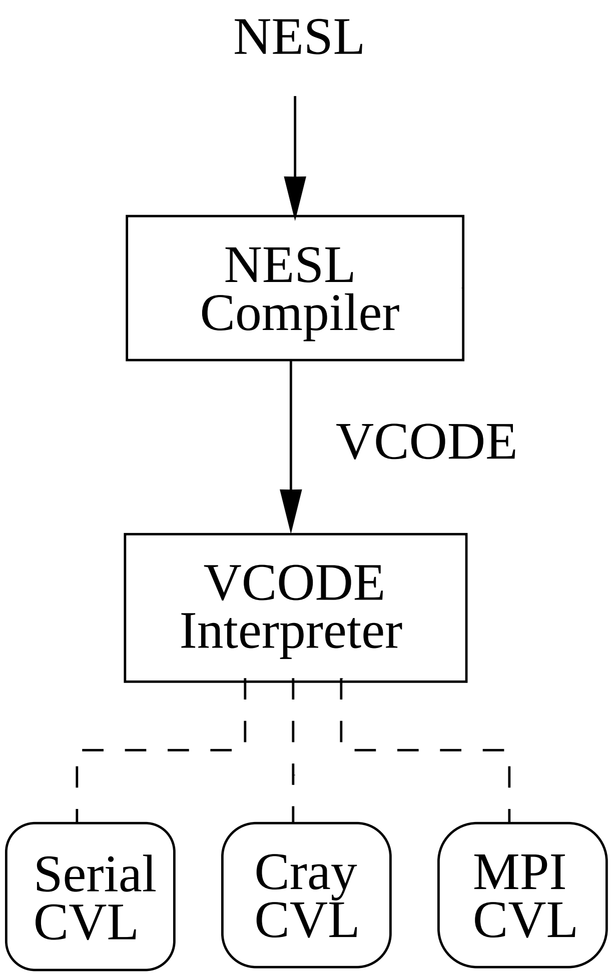
NESL is the first nested data-parallel language

- Good for quickly prototyping parallel algorithms
- Also for animations and visualization via X11

But few people on the web have X (or trust xhost)

So compile into Java, then users can run locally

Structure of NESL system



High-level nested data-parallel language

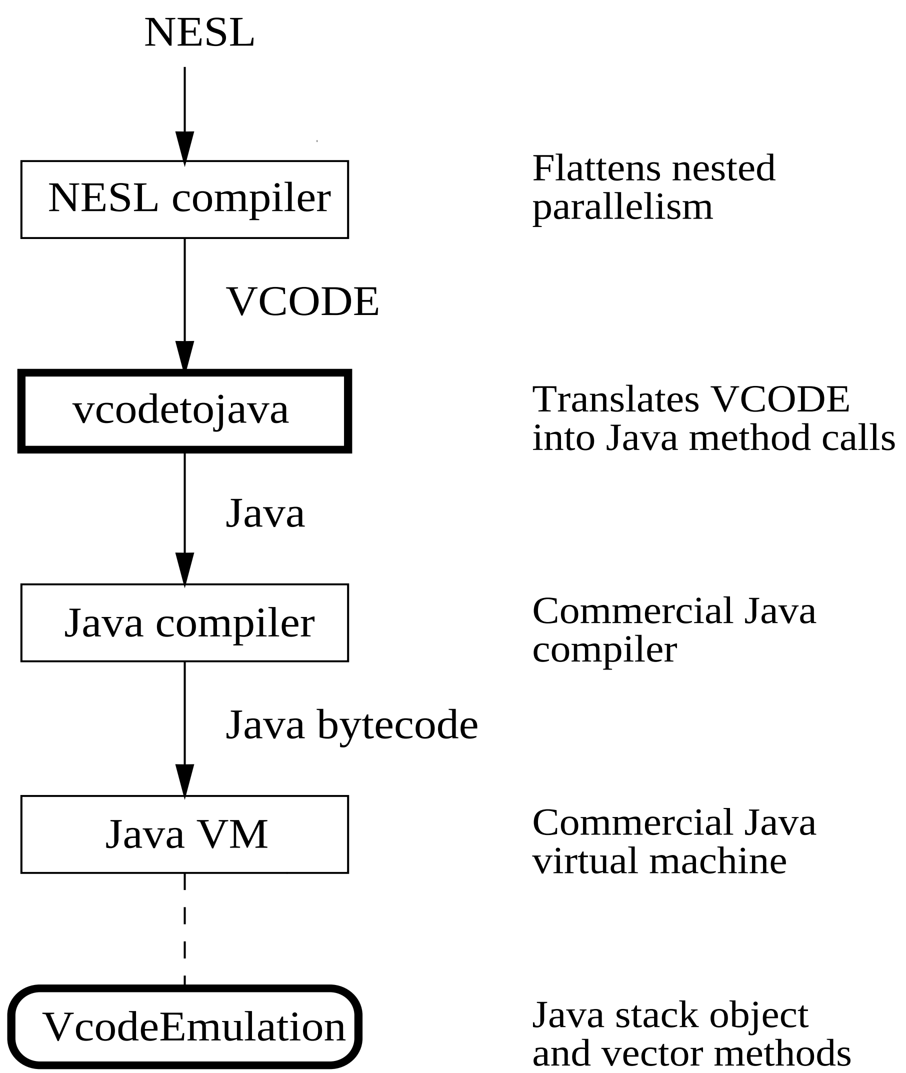
Flattens nested parallelism

Stack-based intermediate vector language

Memory management, runtime error checks

C libraries of vector functions

The NESL-to-Java system



Why compile from VCODE?

By compiling VCODE into Java, we can:

- Replace VCODE interpreter with Java VM
- Minimize changes to rest of system
- Use Java version of vector library
- Replace VCODE interpreter's X11 calls with AWT

Compilation Process

VCODE is translated into:

- Java control operations (calls, conditionals)
- Calls to vector library to do everything else

Vector library:

- Emulates about 130 VCODE operations
- Majority are simple loops (most complex: sort)

Total of 500 lines of Perl (for compiler), and
4000 lines of Java (for the vector library)

Example of compilation

NESL:

```
function dotproduct (X, Y) =
  sum ({x * y}: x in X, y in Y});
```

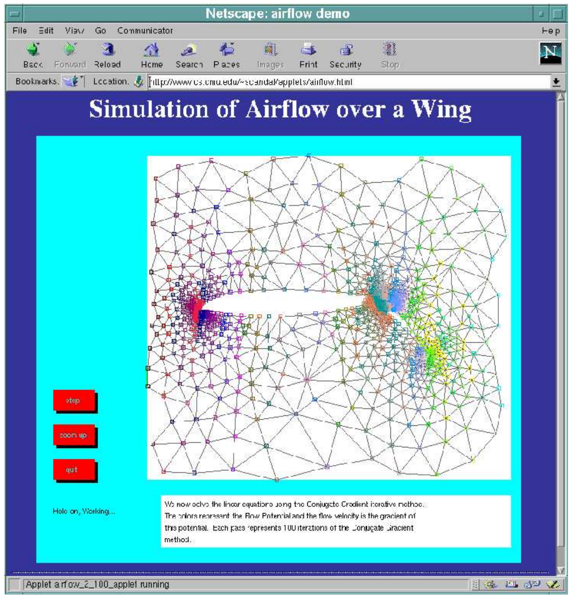
VCODE:

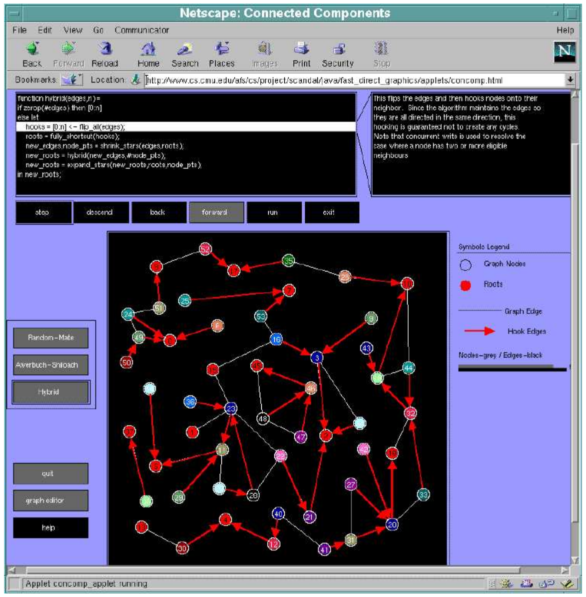
```
FUNC DOTPRODUCT_47
POP 1 1
* FLOAT
COPY 1 1
+_REDUCE FLOAT
POP 1 1
RET
```

JAVA:

```
void DOTPRODUCT_47 () {
     s.Pop (1, 1);
     s.MultF ();
     s.Copy (1, 1);
     s.AddReduceF ();
     s.Pop (1, 1);
}
```

Some compiled applets





Benchmarks

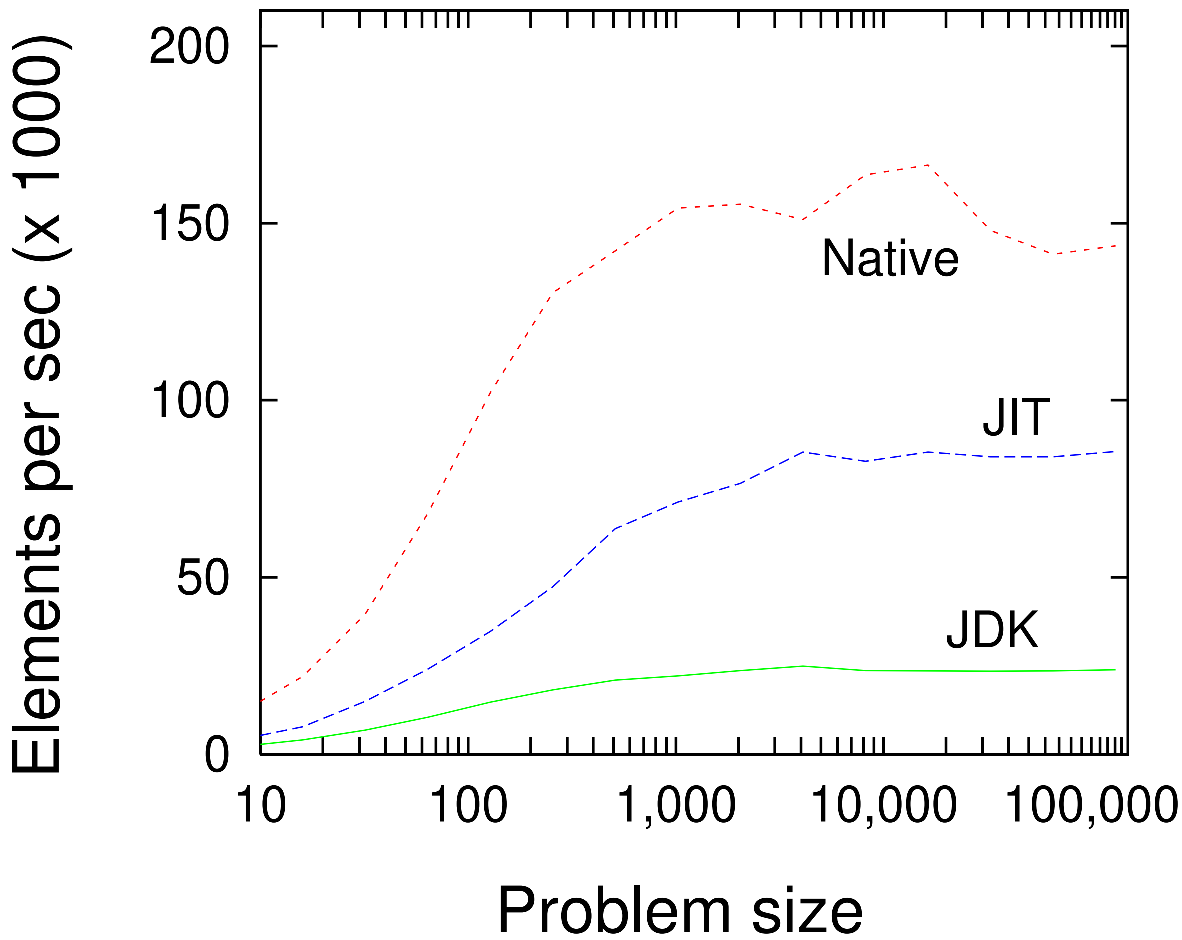
We compared:

- Two platforms, PC and SPARC
- Three computational benchmarks
- Four approaches (interpreted Java, JIT Java, Java with native methods, VCODE interpreter)
- Too much for one talk - see paper for graphs!

Summary:

- JITs are 2x slower than C
- Interpreter overhead is similar for Java,VCODE

Linefit benchmark on a PC



Obvious questions

Why is it still twice as slow as C?

- Bounds checks on array accesses
- Arrays are initialized before use

Why don't you compile straight to bytecode?

- Java and bytecode are roughly equivalent
- Security checks imposed by VM, not compiler
- Easier to use source-based tools with source!

How could this be parallelized?

- Rewrite library to allow one thread per proc.
- Write library on top of data-parallel extension
- Annotate library for parallelizing compiler

Conclusions

We can now write applets in NESL

Java worked nicely as an intermediate language

- Helped by structure of NESL system
- Performance is sufficient for teaching, demos

Lessons learned:

- Quick hacks can take on a life of their own
- Computation is easy: graphics and I/O are hard
- Object creation is really slow in current VMs
- We need smarter VMs to reduce array costs

http://www.cs.cmu.edu/~scandal/applets