Cases

Java as a Language for Scientific Parallel Programming

Bryan Carpenter, Yuh-Jye Chang, Geoffrey Fox, Xiaoming Li

Northeast Parallel Architectures Centre,

Syracuse University,

Syracuse, New York

Why Java?

From the Java whitepaper:

> "Java is designed to meet the challenges of application development in the context of heterogeneous, network-wide distributed environments. Paramount among these challenges is secure delivery of applications that consume the minimum of system resources, can run on any hardware and software platform, and can be extended dynamically."

Goals of Java include

- Portability (cf, P4, MPI, HPF,. . .)
- Support for heterogenous networks (cf, PVM, MPI,. . .)
- Efficiency.

all major concerns in HPC.

Why Java?

- Java is a descendant of C++—used increasingly in scientific programming. Many variations on the theme of C++ for parallel computing (eg, CC++, pC++, A++/P++, . . .).

- Java omits features of C and C++ considered "difficult"—eg, pointers. Suggests Java (like Fortran) may be a suitable source language for highly optimizing compilers.

- Java comes with builtin multithreading. On a shared memory platform threads may be scheduled on different processors. In any case, very convenient in explicit message-passing styles (cf occam, Fortran M, . . .)

Options

Approaches to Parallelism in Java

- Explicit parallelism through Java thread mechanism, scheduled on SMP.
- Automatic or semi-automatic parallelization (eg JAVAR).
- Language extensions (eg SPAR, pizza) or directives?
- Task parallelism. Applets as tasks (or task servers—eg Javelin)? Interacting “applications” as tasks?
-

Options

Approaches to communication in Java

- Message-passing. Internet sockets? Higher-level class libraries—Channels? MPI?

- Communication through shared objects (Nexus, Linda, . . .).

- Distributed shared memory (Java/DSM).

- Data parallelism. High level "array syntax" with implicit communication (cf A++/P++)? Global SPMD style—explicit collective operations on distributed array classes (cf Adlib)?

- Libraries in Java? Libraries in native code?

This talk will focus on traditional message-passing and data-parallel approaches.

Message Passing

Approaches investigated

- Application programming with `java.net` sockets.

- Fortran-M-like dynamic channels, implemented as a class library on top of `java.net`.

- “Native-methods” interface to standard MPI implementation.

MPI Interface

Experimenting with *native methods* interface to MPICH.

Interface modelled on proposed C++ bindings of MPI (but cannot support derived data types . . .).

```
public class Comm {
  public Comm() ;  // Default constructor:
                   //     equivalent to MPI_COMM_WORLD.

  public int size();
  public int rank();

  public void bSendInt(int [] buf, int dest, int tag);
  public Status recvInt(int [] buf, int source, int tag);

      // ...or whichever primitive types are needed
}
```

Work in progress. Simple test cases run, but no full demonstation, yet.

Parallel Arrays in Java

Currently, take a pure class library approach, similar to A++/P++, HPC++ parallel STL, Adlib, etc.

We parametrize an array by a member of the `Array` class, similar to an HPF *template*. Index ranges of array distributed over dimensions of process grid, represented by `Procs` class.

Parallel Java application written as a class extending library class `Node`: maintains some global information and provides collective operations on arrays.

Data-parallel example: initialization

```
Procs p = new Procs(this, 2, 2) ;

Range x = new Range(N, p, 0) ;  // distrib over 1st dim of ‘p’
Range y = new Range(N, p, 1) ;  // distrib over 2nd dim of ‘p’

Array r = new Array(p, x, y) ;

byte [] w = new byte [r.seg()] ;  // main data array

... declare neighbour arrays, ‘cn_’, ‘cp_’, etc, similarly

// initialize the ‘‘life’’ board

for(r.forall() ; r.test() ; r.next())
  w [r.sub()] = fun(x.idx(), y.idx()) ;
```

Data-parallel example: main loop

```
for (int k=0; k<NITER; k++) {

  // Get neighbours

  shift(cn_, w, r, 0,  1, CYCLIC);
  shift(cp_, w, r, 0, -1, CYCLIC);
  ... etc, copy arrays for 8 neighbours

  // Life update rule

  for(int i=0; i<w.length; i++) {
    switch (cn_[i] + cp_[i] + c_n[i] + c_p[i] +
            cnn[i] + cnp[i] + cpn[i] + cpp[i]) {
      case 2 : break;
      case 3 : w[i] = 1; break;
      default: w[i] = 0; break;
    }
  }
}
```

Data-parallel example: comments

Characteristics of the data-parallel style of programming:

- Distribution format of arrays can be changed by altering a few parameters at start of program (eg, change `Range` to `CRange` for cyclic distribution). Main program insensitive to these details.

- Low level message-passing abstracted into high-level collective operations on distributed arrays.

Currently the communications in `Node` are implemented on `java.net` sockets. *Plan to provide a "native-methods" Java interface to the full NPAC PCRC library.*

Array Syntax

Array syntax in Java.

A higher level approach makes `Array` classes into true container classes (for a restricted set of types), and all operations on arrays collective.

```
ArrayFloat a = new ArrayFloat(p, x, y) ;
ArrayFloat b = new ArrayFloat(p, x, y) ;
ArrayInt c = new ArrayInt(p, x, y) ;

a = MATMUL(b, c) ;
```

Communication subsumed into collective array operations. Elements not accessed directly in the Java program.

Can implement on top of previous SPMD Java array library, *or* by making Java program run as a sequential master program controlling parallel back end. Compare with *HPF Interpreter*, A++/P++, etc.

Hampered by lack of user-defined operator-overloading in Java.