Experiments with "HPJava"

Bryan Carpenter, Yuh-Jye Chang
Northeast Parallel Architectures Centre,
Syracuse University,
Syracuse, New York

Cases for Parallel Java

Analysis 1: *Distributed Simulation on the Internet*

- Geographically distributed components.
- Interacting real and simulated modules (eg, vehicles).
- Java as uniform coordination language *and* implementation language for simulation components, including numerically intensive components.

Cases for Parallel Java

Analysis 2: *Java as a general scientific language*

- Java is simple, secure, easy to understand.
- Attractive to a wide range of programmers, professional and otherwise.
- Easy to analyse, therefore good source language for optimizing compilers (?).
- Java takes over from C/C++ as the modern scientific programming language.

Options to Parallelism in Java

- Automatic parallelization (eg, of byte code).
- Language extensions? Directives? Class libraries?
- Fine grain parallelism through extended Java thread mechanism. Interacting objects.
- Task parallelism. Applets as tasks? Interacting "applications" as tasks?
- Tasks started through Web server? Other Java server? `rsh`?

Options to communication in Java

- Message-passing. Communication through Internet sockets? Communication through higher-level class libraries. Channels? MPI?
- Communication through shared objects. Linda-like? (CORBA-like?)
- Data parallelism. High level "array syntax" with implicit communication (cf A++/P++)? Explicit collective operations on distributed arrays (cf Adlib)?
- Libraries in Java? Libraries in native code?

Experiments at NPAC.

Message passing:

- `java.net` Internet socket interface.
- *Fortran-M*-like channels.
- "Native-methods" interface to MPICH.

Data parallel:

- Classes to describe process arrays and distributed data arrays.
- Classes for data traversal (iterators).
- Collective communication via operations on distributed arrays: `shift`, `transpose`, I/O, ...

Mainly interacting Java *applications*, with applet front-end for demos.

Channel communication

Have implemented a Java class library for *channel communication*. Similar to dynamic channels in Fortran M.

Model attractive because

- "Connection oriented", so maps efficiently onto underlying Internet socket operations.
- Directly supports dynamic situations. New remote processes (eg, applets) may come into existence unpredicably.
- Layer of abstraction above Internet addresses, socket port numbers, and resources controlled by Java security model.

Channel communication

Library supports bi-directional communication of data on a channel.

- Pair of locally created channel ends can be connected, creating a local channel.
- A *spawn* function starts another Java instance on a remote processor, and creates an initial channel between parent and child.
- Channel ends can be communicated over channels, on similar footing to data
- Functionality similar to “merge” in Fortran-M available.
- Web applets downloaded from a common server may communicate over a channel, with server transparently through-routing messages (unimplemented).

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 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.