"HPJava": Demo Description

Bryan Carpenter, Yuh-Jye Chang

November 16, 1996

The Web server for the demo runs on a processor from a cluster of high performance workstations located at NPAC. Various other hosts in this cluster run daemons that support remote execution of the Java class files for the demos.

When the demo starts, a menu of available hosts is presented. A subset of these hosts can be selected and added to the set subsequently used for running the examples.

1 Conway's Game of Life

"Life" is a familiar *cellular automaton*, in which cells on a two-dimensional board are updated according to the current state of their eight neighbours. It provides a simple illustration of the the archetypal nearest-neighbour update, which recurs in various numerical simulations, and lends itself well to parallel computation.

The Java implementation uses an experimental version of a class library intended to facilitate *data-parallel* computations on regular arrays. Characteristics of this approach, similar to *High Performance Fortran*, include

- The distribution format of the arrays can be changed just by altering a few parameters at the start of the program—the main program is insensitive to these details
- low level message-passing (in the Java case, through Internet sockets) is abstracted into high-level collective operations on distributed array structures.

Clicking on the "Life" button brings up a menu of parameters which can be changed. These include the size of the grid ("board"), the number of iterations, and the frequency with which the state of the board is displayed. Displaying the board generally takes much longer than updating it: for benchmarking purposes the display should happen infrequently (for demonstration purposes, frequently). The options also allow changing the *distribution format* of the board—the way the board is split between processors. Like HPF, block or cyclic distributions are allowed.

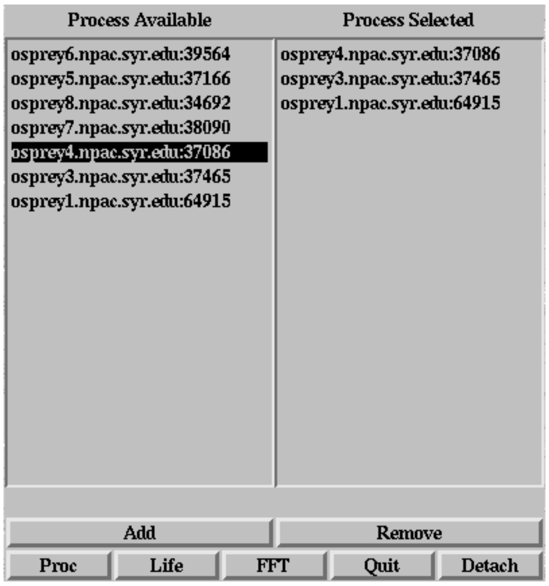

Figure 1: Menu for selecting processors.

Clicking on the "Begin" button causes the simulation to start. The initial state of the board is currently a fixed "cross" of live cells. This will evolve into more interesting patterns as the simulation proceeeds. The processor "owning" a portion of the board is coded through the colour of the cells, providing a clear visualization of the various distribution formats.

The processor set can be changed by clicking the "Proc" button. The parameters or distribution format can be changed by clicking the "Life" button again.

2 2D FFT "Image Compression"

This demo is a Java port of one of the HPFA (*High Performance Fortran Applications*) kernels. As explained in the description of the original code:

> The Fast Fourier Transform (FFT) is the most widely known example of the Spectral method for computational problems. In all such methods, one performs a linear transformation of the stated problem into another physical domain where it is hoped will be more tractable. In Fourier transformations, the mapping is from the time-domain to the frequency-domain. The FFT is widely used in the field of image processing, where one commonly describe an image in terms of intensity values in a two-dimensional matrix. The

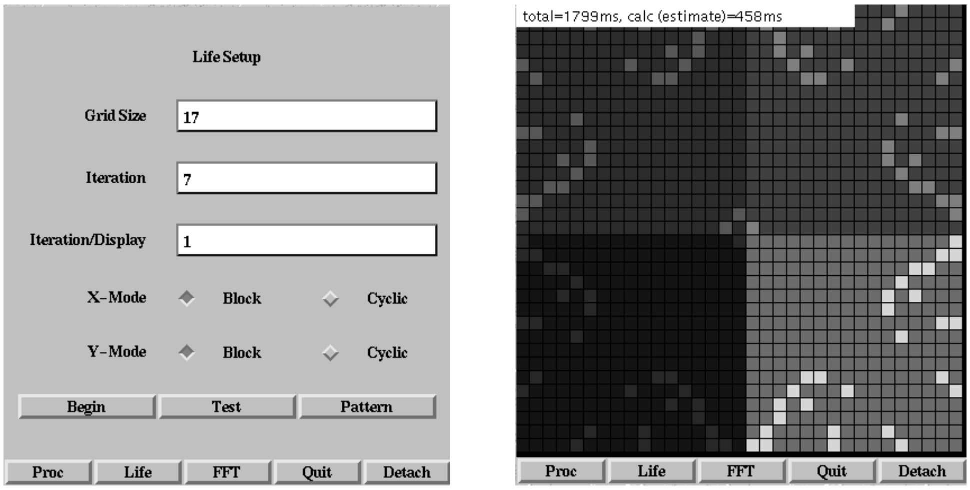

Figure 2: Screens of the "Life" demo.

codes contained here performs a two-dimensional FFT over an example matrix in the following manner:

1. Set up the input 2-D matrix.
2. Perform FFT across the leading dimension of the matrix.
3. Transpose the matrix, using an F90/HPF intrinsic function.
4. Perform FFT again, across the leading dimension.

In our case "transpose" is a new operation in the class library, on a par with the "shift" operation used in nearest neighbour updates. As explained above, sequential FFTs are performed on all columns of the image data in parallel, then on all rows of the image in parallel. The transpose operation sits between these two phases.

After selecting the set of processors click on the "FFT" button. In the demo the image data is a photograph of a wolf. One of several image files (of different resolution) can be selected. In order of increasing size, the options presently are

```
wolf_4.pgm
wolf_2.pgm
wolf.pgm
```

The mask width defines the size of a square centered in the Fourier space representation of the picture. You can choose to delete modes inside or outside this boundary. On clicking the "Begin" button, the original image is displayed then the FFT is performed. The result of the Fourier transform is also displayed as an image, mapping Fourier components into pixels. According to the masking

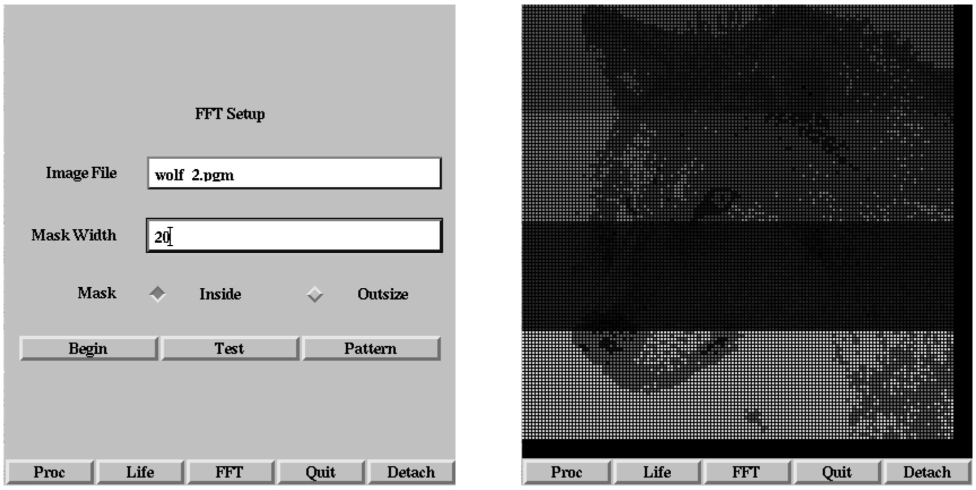

Figure 3: Screens of the "FFT" demo.

selection, some of the components will be deleted. Finally the FFT is reversed, and the modified ("uncompressed") version of the original image is displayed.

Once again the processor "owning" a particular pixel or Fourier component is coded through the colour of the cells.