Goals and Functionality

Charlotte is the first integrated environment for parallel computing on the Web. Now, using a Java-capable browser, any machine on the Web can volunteer to help an ongoing computation, with the Web potentially becoming a gigantic parallel metacomputer.

To use Charlotte, the application programmer writes software for a perfect shared memory virtual machine. Charlotte’s runtime system implements this virtual machine out of the available volunteers, providing load balancing and fault masking.

An application program is a pure Java program with embedded constructs specifying parallel steps. A parallel step consists of concurrent jobs, executable as applets.

A parallel computation is started by the user’s machine. Machines on the Web can volunteer to execute a concurrent job, with the applet and the data downloaded through browsers, as needed.

As Charlotte targets the Web, the computation transparently crosses administrative domains, with the matching of volunteers to computations accomplished through a scalable and fault-tolerant directory service. The overall security is at the level of Java.

Some Implementation Details

In order to implement the predictable, perfect virtual machine on the unpredictable, dynamically changing Web, Charlotte uses two integrated techniques, eager scheduling and two-
A Charlotte program and its execution on the Web

Employing these techniques, an unfinished job can be assigned/reassigned, allowing multiple and partial executions, while maintaining correct once-only semantics. The performance of the volunteer machine is not monitored. Nevertheless, the machines are automatically shifted to increase performance or mask slowdowns and crashes.

The runtime system is optimized to improve performance. A key technique of particular benefit to application software programmers is dynamic granularity management. Even when the programmer has written a fine-grained program, the runtime system will execute in an adaptive coarse-grained manner. Programming and execution parallelisms are decoupled.

An example shows two browsers running a worker applet contributing to the computation of a Mandelbrot set. Notice how the total work is divided among the two browsers.

Bibliography (unless indicated otherwise, available from MILAN web sites at NYU and/or ASU)

A. Baratloo, H. Karl, M. Karaul, Z. Kedem. An infrastructure for networking computing with Java applets.
H. Karl. Bridging the gap between Shared Memory and Message Passing.

MILAN is a joint project of New York University Arizona State University

Zvi M. Kedem
New York University
+1 212 998 3101 (phone)
+1 212 477 3265 (fax)
kedem@cs.nyu.edu
http://www.cs.nyu.edu/milan

Partha Dasgupta
Arizona State University
+1 602 965 5583 (phone)
+1 602 965 2751 (fax)
partha@asu.edu
http://milan.eas.asu.edu