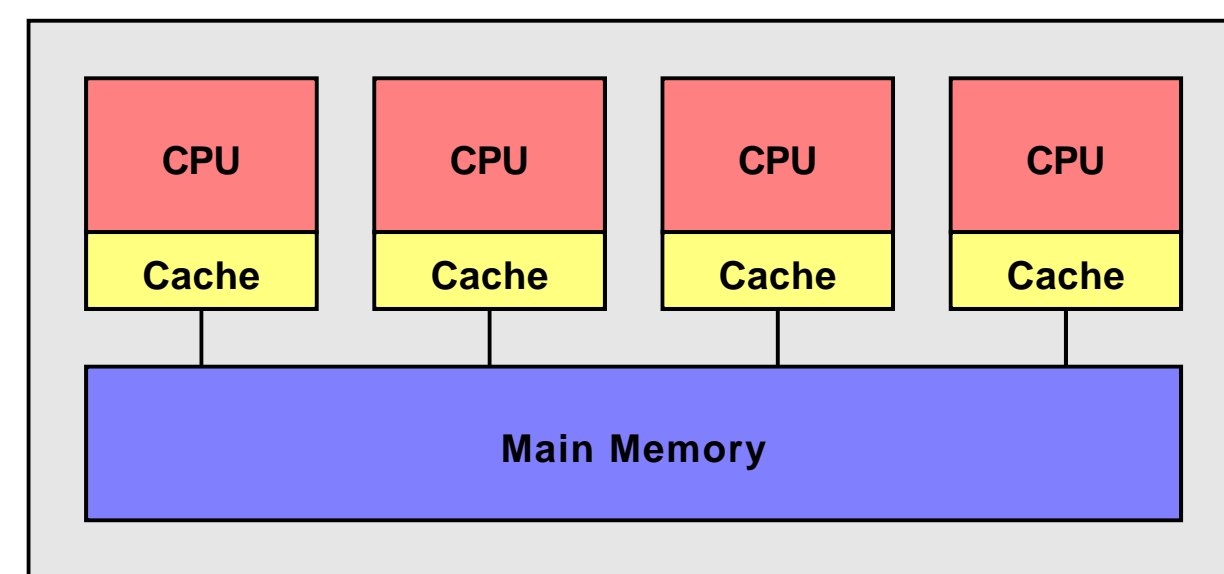


PARALLEL JAVA

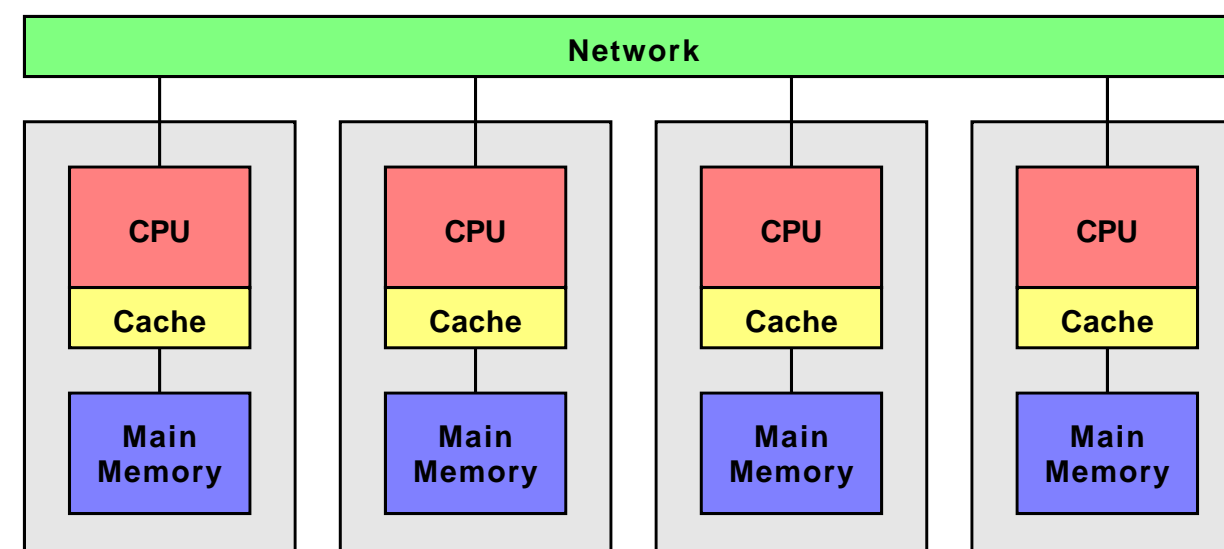
AN API FOR TEACHING AND DEVELOPING PARALLEL PROGRAMS IN 100% JAVA

Alan Kaminsky, Department of Computer Science, Golisano College of Computing and Information Sciences, Rochester Institute of Technology

PARALLEL COMPUTING



- Shared memory multiprocessor (SMP) parallel computer
 - Multiple threads, each running on its own CPU
 - Threads share variables in main memory
 - OpenMP standard for SMP programming



- Cluster parallel computer
 - Multiple processes, each running on its own processor
 - Processes communicate data by message passing through the network
 - MPI standard for cluster programming
- All CS students need to learn parallel computing
 - Computers shifting to multicore chips (SMPs)
 - Multicore chip based computers require parallel programs for full performance
 - OpenMP and MPI are C/Fortran based and are not object oriented
 - Java is becoming the language of choice for teaching programming
 - Middleware is needed for parallel programming in Java: **Parallel Java!**

CODING WITH PARALLEL JAVA

PJ program for computing the Mandelbrot Set on an SMP parallel computer

```
public class MandelbrotSetSmp
{
    public static void main
    (String[] args)
    throws Throwable
    {
        // Parse command line arguments.
        final int width = Integer.parseInt (args[0]);
        final int height = Integer.parseInt (args[1]);
        final double xcenter = Double.parseDouble (args[2]);
        final double ycenter = Double.parseDouble (args[3]);
        final double resolution = Double.parseDouble (args[4]);
        final int maxiter = Integer.parseInt (args[5]);
        final double gamma = Double.parseDouble (args[6]);
        final File filename = new File (args[7]);

        // Initial pixel offsets from center.
        final int xoffset = -(width - 1) / 2;
        final int yoffset = (height - 1) / 2;

        // Create matrix to store results.
        final int[][] matrix = new int[height] [width];

        // Compute all rows in parallel.
        new ParallelTeam().execute (new ParallelRegion()
        {
            public void run() throws Exception
            {
                execute (0, height-1, new IntegerForLoop()
                {
                    // Create image row on top of matrix (thread local).
                    ColorImageRow imagerow = new ColorImageRow (matrix);

                    public void run (int first, int last)
                    {
                        for (int r = first; r <= last; ++ r)
                        {
                            imagerow.row (r);
                            double y = ycenter+(yoffset-r)/resolution;

                            // Compute all columns in row.
                            for (int c = 0; c < width; ++ c)
                            {
                                double x = xcenter+(xoffset+c)/resolution;

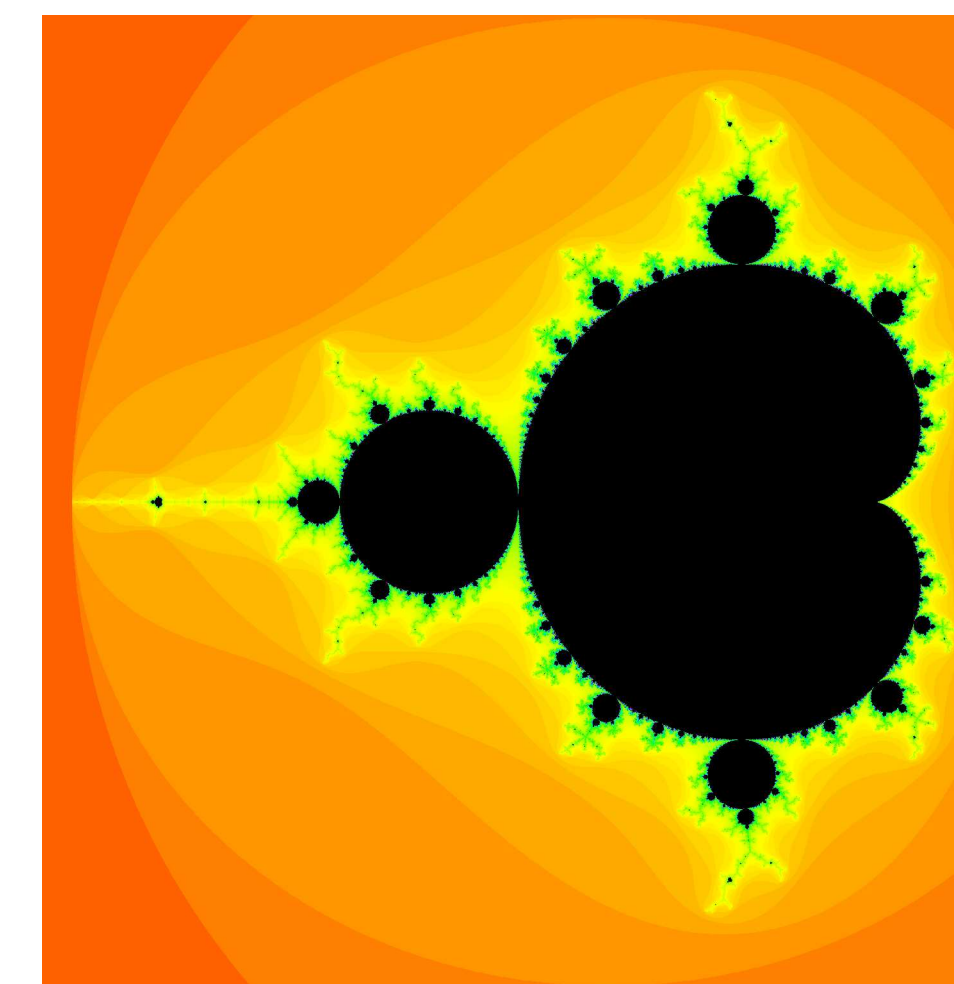
                                // Iterate until convergence.
                                int i = 0;
                                double aold = 0.0;
                                double bold = 0.0;
                                double a = 0.0;
                                double b = 0.0;
                                double zmag2 = 0.0;
                                while (i < maxiter && zmag2 < 4.0)
                                {
                                    ++ i;
                                    a = aold*aold - bold*bold + x;
                                    b = 2.0*aold*bold + y;
                                    zmag2 = a*a + b*b;
                                    aold = a;
                                    bold = b;
                                }

                                // Record number of iterations for pixel.
                                imagerow.setPixelHSB
                                (/*c */ c,
                                 /*hue*/ (float) Math.pow ((double)i)/((double)maxiter), gamma),
                                 /*sat*/ 1.0f,
                                 /*bri*/ i == maxiter ? 0.0f : 1.0f);
                            }
                        }
                    }
                });
            }
        });

        // Write image to file.
        ColorImage image = new ColorImage (matrix);
        ImageIO.write (image, "png", new BufferedOutputStream (new FileOutputStream (filename)));
    }
}
```

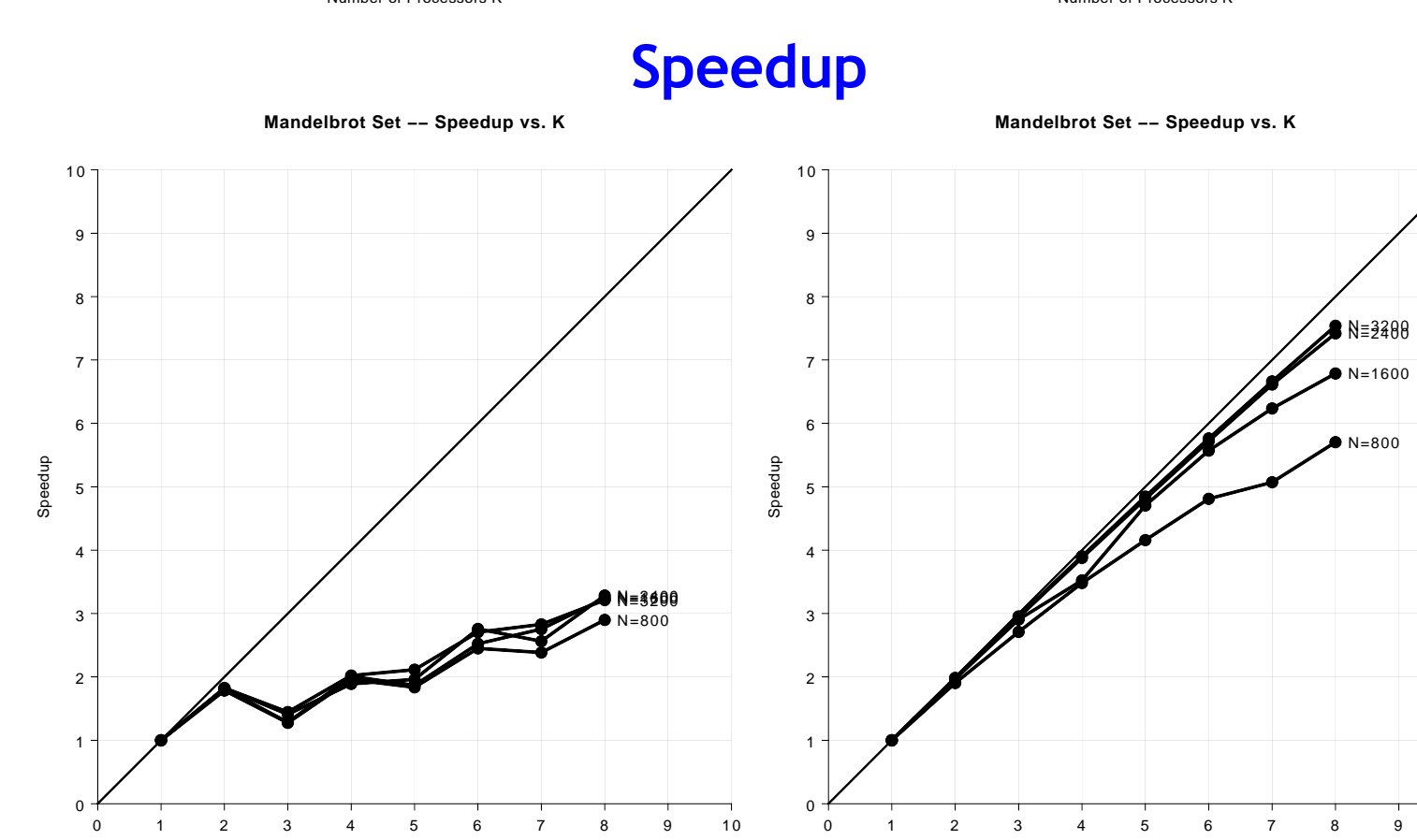
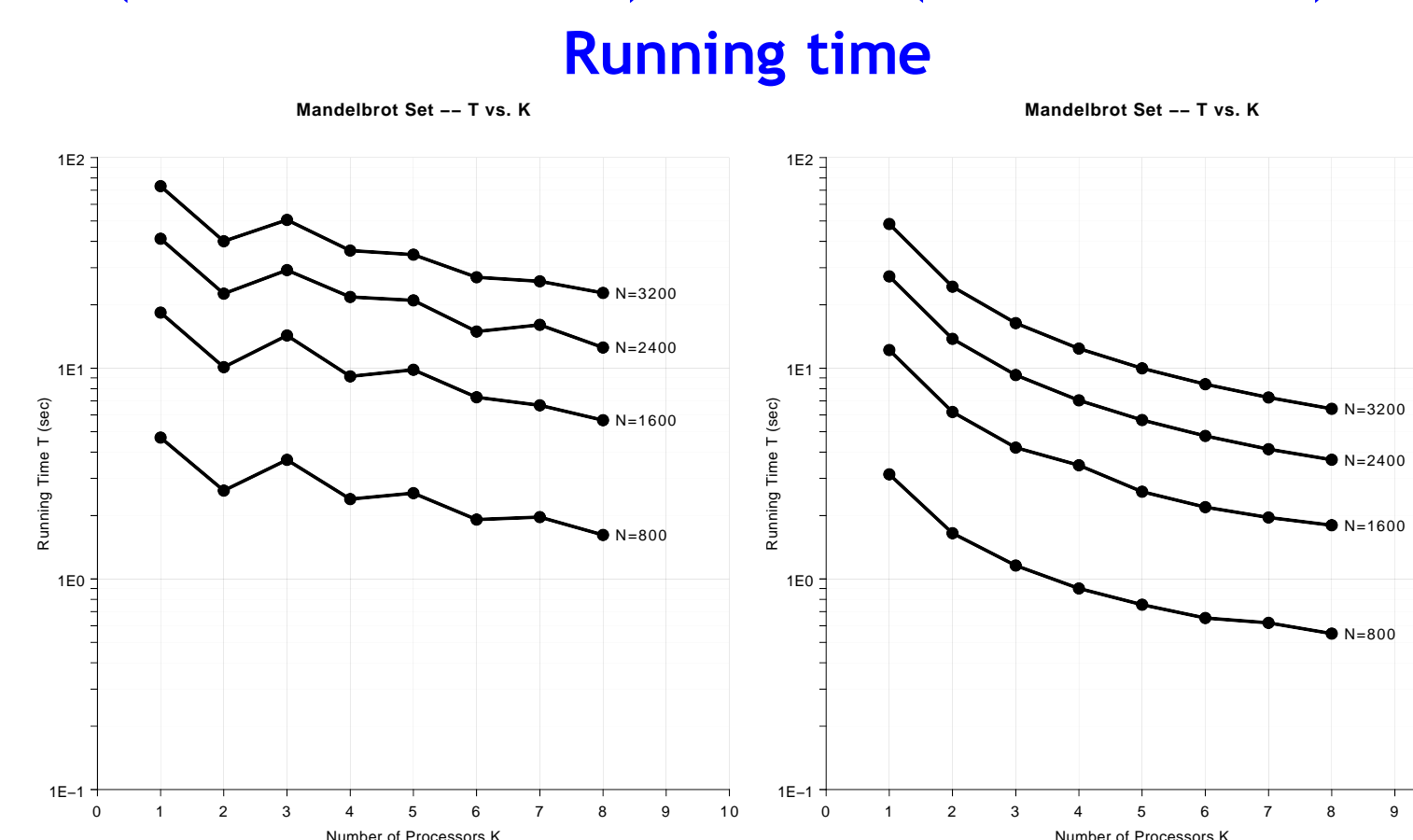
PARALLEL JAVA PERFORMANCE

Output image (N x N pixels)



Fixed schedule (not load balanced)

Dynamic schedule (load balanced)



(computation only)

FEATURES OF PARALLEL JAVA

- PJ itself is written in 100% Java (JDK 1.5)
- PJ's SMP parallel features inspired by OpenMP
 - Parallel thread teams
 - Parallel loops with selectable scheduling
 - Parallel sections and section groups
 - Variable scoping: shared, thread local
 - Reduction variables with arbitrary operations
- PJ's cluster parallel features inspired by MPI
 - PJ middleware automatically runs a program on multiple processors of the cluster
 - Message passing of primitive types and non-primitive types (Java Object Serialization)
 - Message passing of arrays and matrices, or arbitrary portions thereof
 - Message passing operations: send, receive, sendReceive, broadcast, scatter, gather, allGather, reduce (others tba)
 - Reduction with arbitrary operations
- PJ supports hybrid SMP cluster parallel programming
- PJ facilitates teaching parallel programming
 - Students who know Java find it easy to write parallel programs in PJ
 - PJ teaches the concepts of OpenMP and MPI in a 100% Java setting
 - PJ comes with an extensive library of example programs for SMP and cluster computers
- For further information
 - Alan Kaminsky, ark@cs.rit.edu
 - PJ Library (GNU GPL licensed): <http://www.cs.rit.edu/~ark/>
 - Parallel Computing I courseware: <http://www.cs.rit.edu/~ark/531/>

PJ authors: Alan Kaminsky and Luke McOmber
Presented at SIGCSE 2006, Houston, 03-Mar-2006