Embarrassingly Parallel Computations

- Embarrassingly parallel computation
  - Can be divided into completely independent parts, no communication between the parts
  - Data is not shared, but computations may be the same (SPMD model)
- Nearly embarrassingly parallel
  - Results must be distributed and collected and combined in some way
  - Manager & workers, but minimal interaction between workers
  - Workers may be created dynamically or statically
  - If processors are different (e.g., networked workstations) load-balancing techniques may be necessary

Geometrical Transformation of Images

- Processing of 2D images
  - Move image in display space, change its size, rotate it in 2 or 3 dimensions
  - Smoothing, edge detection
- Image is stored as a pixmap, each pixel as a binary number in a 2D array
  - Geometrical transformations affect the coordinates of each pixel to move its position without affecting its value
- Geometrical transformations
  - Shifting — in x or y dimension, or both
  - Scaling — magnification or reduction
  - Rotation — by some angle
  - Clipping — deletes points outside a specified rectangle

Geometrical Transformation of Images (cont.)

- Main concern is division into groups of pixels for each processor (many more pixels than processors!)
  - Usually either by square/rectangular regions, or by columns/rows
  - Doesn't matter here because no communication needed between regions
- Example:
  - Master process and 48 slave processors
  - Image of 480 rows x 640 columns
  - Each slave processes 10 rows x 640 columns
  - Approach (details in figure):
    - Master sends rows to processes, gets back old and new coordinates, and copies values in image from old to new coordinates
    - Slaves add offsets to coordinates

Geometrical Transformation of Images (cont.)

Master:

```
for (i = 0, row = 0 ; i < 48 ; i++, row = row+10)          /* for each process */
        send(row, Pi);                                     /* send row number */

for (i = 0 ; i < 480 ; i++)                                /* initialize temp */
        for (j = 0 ; j < 640 ; j++)
                temp_map[i][j] = 0;

for (i = 0 ; i < (640*480) ; i++) {                        /* for each pixel */
        recv(oldrow,oldcol,newrow,newcol, Pany)            /*accept new coords */
        if !((newrow<0)||(newrow>=480)||(newcol<0)||(newcol>=640))
                temp_map[newrow][newcol]=map[oldrow][oldcol];
}
for (i = 0 ; i < 480 ; i++)                                /* update bitmap */
        for (j = 0 ; j < 640 ; j++)
                map[i][j] = temp_map[i][j];
```

Slave:

```
recv(row, Pmaster);                                        /* receive row num */
for (oldrow = row ; oldrow < (row+10) ; oldrow++)
        for (oldcol = 0 ; oldcol < 640 ; oldcol++) {       /* transform coords */
                newrow = oldrow + delta_x;                 /* shift in x direction */
                newcol = oldcol +delta_y;                  /* shift in y direction */
                send(oldrow,oldcol,newrow,newcol, Pmaster);    /* to master */
        }
```

Geometrical Transformation of Images (cont.)

- Analysis of example:
  - Assume n x n pixels, one computation step per pixel, sequential time is $O(n^2)$
  - Communication
    - $t_{comm} = p(t_{startup} + 2t_{data}) + 4n^2(t_{startup} + t_{data}) = O(p+n^2)$
    - Sending row numbers: p sends, each with a startup cost and 2 data items to send
    - $4n^2$ data items returned to master, each received sequentially
  - Computation
    - $t_{comp} = 2(n^2 / p) = O(n^2 / p)$
    - Image divided into groups of n^2 / p pixels
    - Each pixel requires 2 additions
  - Overall execution time
    - For constant p, $O(n^2)$
    - Constant for communication may be far bigger than that for computation (e.g., $4n^2 + p$ startup times, each 5µs for Ethernet)

Mandelbrot Set

- Displaying the Mandelbrot Set
  - Set of points in the complex plane that are computed by iterating a function until z becomes greater than a specified value or the number of iterations exceeds a specified limit
  - Result is displayed as a 2D image of the complex plane, after the image is scaled to match the coordinate system of the display (very computationally intensive)
  - Regions of the display can be selected and magnified to produce visually pleasing pictures
- Each pixel can be computed without info from neighbors, but amount of computation per pixel can vary
  - Consider both static and dynamic task assignment

Mandelbrot Set (cont.)

- Static task assignment
  - Give each worker 10 rows as before
  - Order in which processed pixels are received by master depends on number of iterations to compute its value
  - Same problems as before in that results are sent back one at a time
- Dynamic task assignment
  - Use load balancing so all processors complete at same time
  - Can not assign different-sized regions to different processors — do not know required number of iterations in advance
  - Use a work pool, which holds a set of tasks to be performed
    - Processing a pixel = task
    - Number of tasks is fixed in advance
    - Idle processor requests task from the pool

Mandelbrot Set (cont.)

- Example:
  - 480 x 640 image as before
  - Processes compute entire rows as a task
  - Approach (details in figure):
    - Each slave is first given one row to process, and then it gets another row when it returns a result until there are no more rows to compute
    - Master sends a termination message when all rows have been taken
    - Different tags for rows sent to slaves, termination message, and results
- Analysis of example:
  - Difficult to analyze since it's impossible to know in advance how many iterations are necessary, although there is a limit of max
  - Sequential time is <= (max)(n), or O(n)

Mandelbrot Set (cont.)

Master:

```
count = 0;                          /* counter for termination */
row = 0;                            /* row being sent */
for (k = 0 ; k < procno ; k++) {    /* assuming procno<disp_height */
    send(&row, Pk, datatag);        /* send initial row to process */
    count++;                        /* count rows sent */
    row++;                          /* next row */
}
do {
    recv(&slave, &r, color, Pany, result_tag);
    count--;                        /* reduce count as rows received */
    if (row < disp_height) {
        send(&row, Pslave, data_tag);           /* send next row */
        row++;                                  /* next row */
        count++;
    } else
        send(&row, Pslave, terminator_tag);     /* terminate */
    rows_recv++;
    display(r, color);                          /* display row */
} while (count >0);
```

Slave:

```
recv(y, Pmaster, ANYTAG, source_tag);    /* receive 1st row to compute */
while (source_tag == data_tag) {
    c.imag = imag_min + ((float) y * scale_img);
    for (x = 0 ; x < disp_width ; x++) {         /* compute new row colors */
        c.real = real_min + ((float) x * scale_real);
        color[x] = cal_pixel(c);
    }
    send(&i, &y, color, Pmaster, result_tag); /* row colors to master */
    recv(y, Pmaster, source_tag);             /* receive next row */
}
```


Mandelbrot Set (cont.)

- Analysis of example (cont.):
  - Communication
    - $t_{comm1} = s(t_{startup} + t_{data})$
    - Row number sent to each slave, one data item to each of s slaves
  - Computation
    - $t_{comp} <= (max \times n)/s$
    - All slaves compute in parallel, assuming the pixels are evenly divided across the processors
  - Communication
    - $t_{comm2} = (n/s)(t_{startup} + t_{data})$
    - Results passed back to master using individual sends
  - Overall execution time
    - $t_p <= (max \times n)/s + (n/s + s)(t_{startup} + t_{data})$
    - Where number of processors p = s+1
    - Speedup approaches p-1 if max is large
    - Parallelizing this example appears to be worthwhile