

COMP/ELEC 526 Homework

Due September 24, 2003, 11:59 PM (under my door)

1 Objective

To design an effective parallel architecture, it is important to have an appreciation of the difficulties of parallel programming. Your first assignment is to implement parallel versions of two algorithms – matrix multiply and mergesort – for a shared-memory architecture.

The parallel programs will run on the Rice Simulator for ILP Multiprocessors (RSIM), a detailed execution-driven parallel architecture simulator.

The purpose of using a simulator (as opposed to a real machine) is to allow you to experiment with a variety of system parameters; for example, cache size, memory access time, and network latency. You should experiment with at least several data points for one such parameter, and justify why your parameter of choice is interesting in the context of parallel architecture.

A disadvantage of using a simulator is that it takes a long time, *so please do not leave this assignment until the last minute*. Additionally, please consider data sizes carefully, starting with small sizes and working your way up to interesting sizes based on approximating runtimes (keeping in mind the order of the algorithms used.).

The two problems to be implemented are described below.

2 Matrix Multiplication

The first problem is matrix multiplication. For our purposes, a matrix is a two dimensional array of integers, and we will only consider square matrices. Your program should take as input the number of rows/columns in the matrices and the data entries of the input matrices. It should produce a third matrix, which is the product of the two input matrices, and should output the resulting matrix. Only the computation phase should be considered for statistics (the initial input and final output should be done serially and will not be considered). An example serial matrix-multiply code that multiplies two $N \times N$ matrices, A and B , appears below:

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    C[i][j] = 0; /* if not initialized to 0 */
    for (k = 0; k < N; k++)
      C[i][j] = C[i][j] + A[i][k]*B[k][j];
```

The above algorithm is $O(n^3)$, implying that doubling a matrix dimension will lead to an eightfold increase in runtime.

3 Merge Sort

You are required to implement a parallel merge-sort to sort an arbitrarily long array of integers. Merge-sort is a recursive sorting algorithm where a array of n elements is first divided into two arrays of length $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$. Each sub-array is sorted and the resulting sorted arrays are then merged. The sorting of each half is done using recursive calls to mergesort until the generated sorted arrays are shorter than some threshold. For example, with a threshold of 2, a 16 element array is first divided into two 8 element arrays. Each 8 element array is further divided into 4

element arrays, which are further divided into 2 element arrays. The 2 element arrays are sorted and then merged into sorted 4 element arrays. The sorted 4 element arrays are then merged into sorted 8 element arrays and so on. The core routine for a simple serial merge that merges sorted array $\{x_1 \leq x_2 \leq \dots \leq x_{\lceil n/2 \rceil}\}$ and sorted array $\{y_1 \leq y_2 \leq \dots \leq y_{\lfloor n/2 \rfloor}\}$ to produce a sorted array $\{z_1 \leq z_2 \leq \dots \leq z_n\}$ is as follows.

```
Merge(..)
{
    int    i, j, k = 1;

    while (i ≤ [n/2] && j ≤ [n/2]) {
        if ( xi < yj ) {
            zk = xi; i = i+1; k = k+1;
        }
        else {
            zk = yj; j= j+1; k = k+1;
        }
    }
}
```

The above algorithm is $O(n \log n)$, implying that doubling the data size will lead to a greater than double increase in runtime.

4 Resources

The RSIM binary is available on owlnet (see accompanying handout). For long simulation runs, you can use the owlnet compute servers. You may also run this on any other Rice machine where the system administrators allow it (e.g., not hoss or titan), if that machine is a SPARC/Solaris machine or other supported host architecture.

5 What is expected?

You may work on the programming and simulation portion of the assignment in groups of 2. However, each person is responsible for writing up a separate report and analysis.

To write the parallel programs, choose your own task partitioning strategy and implement the necessary synchronization to ensure correctness. You are expected to run your parallel programs with several input sizes using the default architectural parameters provided (see accompanying handout). At least some of your input sizes should lead to working sets greater than the available cache. You should run with at least 1, 4, 8, and 16 processors. If the speedups are not ideal, identify the reasons for the deviation. Collect your own statistics to justify your explanations. You should then use at least 3 different values of at least one architectural parameter that you think may affect the performance of your programs, run the programs, and explain the results.

6 Report

For each of the two problems, you should hand in the following.

1. A complete description of your approach to parallelizing the programs, including explanations of how the tasks are partitioned between the various processors, and the synchronization used.
2. A pointer to an owlnet directory for your code. Remember to set read permission for the directory.
3. List of command line and configuration parameters used to invoke RSIM for your various test cases.
4. Sample input file with a corresponding output file (hardcopy or pointer to an owlnet directory). Additionally, confirm that your output was indeed valid.

5. Timing measurements on at least 1, 4, 8, 16 processors for each of the inputs used, with the default cache and timing parameters. Also give the resulting speedup curves.
6. An analysis of the speedups obtained above, using detailed statistics.
7. At least three runs on 1, 4, 8, 16 processors obtained by changing one of the default architectural parameters. List the changed parameters.
8. An explanation of why it is interesting to change the above parameter, what you expected to find, what you found, and why.
9. Describe any code modifications that would likely improve performance more than simple hardware parameter changes.