PHYS405

Advanced Computational Physics
Parallel Computing

Assignment # 3
Due: Friday, October 16, 2009

Purpose: Learn to time c codes - specifically sorting algorithms.

Note: Please identify all your work.

Background: The basic idea of timing a code is to make two time calls - one before and one after the code you want timed, and then take the difference. There are many ways to make such time calls in c. We will use `gettimeofday` because it allows timing to μ-seconds.

The `gettimeofday` call is described by

```
#include <time.h>
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

The return value is 0 for success and -1 for an error. The function places the time and timezone information into memory specified by `*tv` and `*tz` respectively. We don't care about the timezone, so we can place a `NULL` pointer there.

In order to understand how to get at the time we need to understand the `timeval` struct. This is defined by

```
struct timeval{
  time_t         tv_sec;
  suseconds_t    tv_usec;
};
```

The type `time_t` is essentially an integer type used to store the number of seconds elapsed from the beginning of the Unix epoch (12:00 UTC on 1 January 1970). The `suseconds_t` type is similar: it is an integer that holds the the number of μ-seconds that have elapsed

since the last second. So, if we want the current time to the μ-second since the Unix epoch we have

```
struct timeval t1;
gettimeofday(&t1,NULL);
printf("Unix rulz ur world for %d.%06d seconds!\n",t1.tv_sec,t1.tv_usec);
```

Note the formatting for the μ-seconds.

Assignment:

Part I Write a c code (or modify the one in the web pages) that wraps a sorting algorithm in order to time it using the `gettimeofday` call. Print out the total time elapsed. Repeat this for a sequence of lists of exponentially growing size, say 2^n with $1 \leq n \leq N \sim 20$. (hint: make one long unsorted list of length 2^N, say with a random number generator, and just take subsets of this list). Do this for both a bubble and a quicksort. Make a log-log plot of the times to compare the algorithms. What is the order of the algorithms (i.e., visually compare against curves of the form $c_1 n^2$ and $c_2 n \ln n$)? Do you want to bubble sort large lists (i.e. what value of N did you give up at)?

Part II Write a parallel version of the bubble sort code. Use an algorithm that leaves the task of ordering the sub-lists to processes 1 to size-1 (size being the number of processes), using process 0 to merge back the ordered sub-lists into an ordered large one and performing a check (sum of members) on the sort.

Part III Insert timing tools in the parallel code. Time separately: the sorting of the sublists, the merging of the sublists and the total times for both sort and merge (total sorting times). Plot the total sorting times for the serial code AND the parallel code for the values of N used in Part I. Find out the N dependence of the merge times.