Parallel Quicksort

Quick-sort works by selecting a pivot element, moving all elements less than (or equal to) the pivot to one side, and the other elements to the other side of the array. If there are more than one element in one of the two subsequences, it then recursively calls itself passing the subsequence. This is quite easily parallelized by starting a new process, thread or whatever for each recursion.

As the thread creation time[2] is far longer than the time to compare and exchange two elements in most cases, the algorithm cannot start new threads for each partitioning. Instead, the number of elements to be partitioned is compared to a hard-coded limit, and if the number of elements is sufficiently large, it creates a new thread, otherwise it just makes a recursive function call.

The algorithm given in table 3 is the one used for the string sorting measurements, and it is slightly more optimized than the one used for integer sorting. $P_{0...n-1}$ is an array of pointers to the elements to be sorted, EP_i refers to the element pointed to by P_i, and s is the parallel threshold.

Table 3: Parallel Quicksort.

Choose pivot element EP_0

Partition P into two subarrays less than and equal to, and greater than the pivot respectively.

$i \leftarrow$ location of pivot in P

$r \leftarrow n - (i + 1)$

if $i > s$ then

    Start new thread to sort $P_{0...i}$

else if $i > 1$ then

    Sort $P_{0...i}$

if $r > 1$ then

    Sort $P_{i+1...n-1}$

Wait for thread to complete

Parallel Quick Sort