Fine-tuning the partitioning algorithm for Quick Sort and Kth Element led to investigation of alternative ways of moving the median of three elements into position for the algorithm as used by Naps [1]. One is to fully three-sort the low, middle, and high elements; the other, simply to identify the median element and move it into the low position. Unexpectedly, the latter algorithm takes more time when running under Java on a multiprocessor computer.

Introduction

In his second article on QuickSort (after his initial statement of the algorithm in the Algorithms of the ACM [2]), in a section with the title “Partition without Exchange”, Hoare describes a partitioning strategy that depends not on exchanges but on making single data moves. [3] The author first encountered it in Naps’ “Introduction to Data Structures and Algorithm Analysis” [1] but was not surprised to learn that Hoare had already described it 30 years earlier. As Hoare notes, “The expected number of copying operations is obviously twice the corresponding figure for exchanges.” [3] As generally implemented, however, each exchange requires three copying operations for a net reduction in the number of copying operations in the single data movement implementation.

The following is the author’s transformation of Naps’ pseudo-code into Java, and the further transformation into C is trivial.

```java
hold = x[lo];
while (true)
    { while (hold.compareTo(x[hi])<0 &&
        lo < hi)
        hi = hi - 1;
    if (lo == hi) break;
x[lo] = x[hi];
    lo = lo + 1;
    while (x[lo].compareTo(hold)<0 &&
        lo < hi)
        lo = lo + 1;
    if (lo == hi) break;
x[hi] = x[lo];
    hi = hi - 1;
}
x[hi] = hold;
```

The pivot value is taken from the contents of \(x[lo]\), and the search for a value to occupy that position begins with \(x[hi]\). Both of the inner while loops exit when \(lo\) and \(hi\) come together, so there is no need for sentinel values.

Robert Sedgewick, in his article “Implementing Quicksort Programs”, proposes the improvement of using the median of three elements as the pivot value [4], but credits the idea to Singleton. [5] That preface can be added onto the above partition implementation. This paper will examine an unexpected result found on two different implementations of the median of three rearrangement.

Median of Three Rearrangements

One way to accomplish discovery of the median of three (of \(lo\), \(mid\), and \(hi\)) is to three-sort those array element, as is shown in the following code segment.

```java
mid = (lo + hi) / 2;
if ( x[mid].compareTo(x[lo]) < 0 )
    swap ( lo, mid, x );
    // assert:  x[lo] <= x[mid]
if ( x[hi].compareTo(x[mid]) < 0 )
    swap ( mid, hi, x );
    // assert:  x[hi] is the largest value
if ( x[mid].compareTo(x[lo]) < 0 )
    swap ( lo, mid, x );
    // assert:  x[lo] <= x[mid] <= x[hi]
swap ( lo, mid, x );  // into place
```

The intent is to make the three-sort of the elements clear, followed by the final move of the median value into position. Clearly this code segment always performs three comparisons. If all six value permutations are equally likely, the three if statements will result on average in 1 1/2 swaps, followed by the final positioning swap. In the context of some other partitioning schemes, this has the advantage of positioning a sentinel value at \(x[hi]\).

The partition scheme in use here, however, does not require any sentinel values. One can also expressly traverse the decision tree generated by the six possible permutations and only do one swap, the swap that puts the median value into \(x[lo]\). Needless to say, it is not as clear a code segment as the above.
either the optimized QuickSort (using insertion sort for segments less than 10) or the unoptimized version. Specifically, for the results reported in the table, sizes used are between 100 and 1000, stepping by 50, for each size collecting 10000 samples. The quantity of interest is the ratio of the average time required for the full three-sort versus the decision-tree algorithm.

Both in all C runs and in the Java runs on single processor machines (with hyperthreading), the full three-sort takes noticeably more time than the decision-tree algorithm, and the effect is less pronounced when insertion sort is used on segments of size less than 10. What is remarkable, however, is that this behavior is turned exactly on its head when the Java implementation is run on a multiprocessor machine. All but the Intel Xeon were run in the Windows environment (Windows XP except for the Turion processor, which was running Windows 7), while the Xeon computer was an Ubuntu Linux computer. Java version 1.6.0_16 was used, while Microsoft Visual C++ 6.0 and gcc -4.3.3 were used for the C implementation on the Windows and Linux computers respectively.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Optimized C</th>
<th>Optimized Java</th>
<th>Non-Optimized C</th>
<th>Non-Optimized Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium 4</td>
<td>1.013</td>
<td>1.011</td>
<td>1.066</td>
<td>1.042</td>
</tr>
<tr>
<td>Intel Atom</td>
<td>1.033</td>
<td>1.016</td>
<td>1.097</td>
<td>1.061</td>
</tr>
<tr>
<td>AMD Turion 64 X2</td>
<td>1.032</td>
<td>0.966</td>
<td>1.066</td>
<td>0.922</td>
</tr>
<tr>
<td>Intel Xeon 2-proc.</td>
<td>1.006</td>
<td>0.983</td>
<td>1.008</td>
<td>0.921</td>
</tr>
</tbody>
</table>

Ratio of average time for three-sort vs. decision-tree

It would seem that the Java Virtual Machine, when it detects a multiprocessor, is able to generate some parallel execution in apparently sequential code segments. The puzzle is how this is done, and why it is more effective with three separate conditional segments rather than one triply nested conditional segment.

This might be an interesting problem to propose for students in an advanced operating systems class. 

References


TIMOTHY J. ROLFE  
Department of Computer Science, Eastern Washington University  
Cheney, Washington 99004-2493 USA  
Timothy.Rolfe@mail.ewu.edu http://penguin.ewu.edu/~trolfe/  

Categories and Subject Descriptors: F.2.2, Nonnumerical Algorithms and Problems — Sorting and searching.  
General Terms: Algorithms, Performance, Experimentation  
Keywords: Partition, Quick Sort, Benchmark  

DOI: 10.1145/1805724.1805733 © 2010 ACM 2153-2184/10/0600 $10.00