Branch Mispredictions Don’t Affect Mergesort

Amr Elmasry¹, Jyrki Katajainen¹,², and Max Stenmark²

¹ Department of Computer Science, University of Copenhagen,
Universitetsparken 1, 2100 Copenhagen East, Denmark
² Jyrki Katajainen and Company,
Thorsgade 101, 2200 Copenhagen North, Denmark

Abstract. In quicksort, due to branch mispredictions, a skewed pivot-selection strategy can lead to a better performance than the exact-median pivot-selection strategy, even if the exact median is given for free. In this paper we investigate the effect of branch mispredictions on the behaviour of mergesort. By decoupling element comparisons from branches, we can avoid most negative effects caused by branch mispredictions. When sorting a sequence of \(n\) elements, our fastest version of mergesort performs \(n \log_2 n + O(n)\) element comparisons and induces at most \(O(n)\) branch mispredictions. We also describe an in-situ version of mergesort that provides the same bounds, but uses only \(O(\log_2 n)\) words of extra memory. In our test computers, when sorting integer data, mergesort was the fastest sorting method, then came quicksort, and in-situ mergesort was the slowest of the three. We did a similar kind of decoupling for quicksort, but the transformation made it slower.

1 Introduction

Branch mispredictions may have a significant effect on the speed of programs. For example, Kaligosi and Sanders [8] showed that in quicksort [6] it may be more advantageous to select a skewed pivot instead of finding a pivot close to the median. The reason for this is that for a comparison against the median the outcome has a fifty percent chance of being smaller or larger, whereas the outcome of comparisons against a skewed pivot is easier to predict. All in all, a skewed pivot will lead to a better branch prediction and—possibly—a decrease in computation time. In a same vein, Brodal and Moruz [3] showed that skewed binary search trees can perform better than perfectly balanced search trees.

In this paper we tackle the following question posed in [8]: Given a random permutation of the integers \(\{0, 1, \ldots, n - 1\}\), does there exist a faster in-situ sorting algorithm than quicksort with skewed pivots for this particular type of input? We use the word in-situ to indicate that the algorithm is allowed to use \(O(\log_2 n)\) extra words of memory (as any careful implementation of quicksort).

It is often claimed that quicksort is faster than mergesort. To check the correctness of this claim, we performed some simple benchmarks for the quicksort (\texttt{std::sort}) and mergesort (\texttt{std::stable_sort}) programs available at the GNU implementation (\texttt{g++} version 4.6.1) of the C++ standard library; \texttt{std::sort} is
Table 1. The execution time [ns], the number of conditional branches, and the number of mispredictions on two of our computers (Per and Ares), each per $n\log_2 n$, for the quicksort and mergesort programs taken from the C++ standard library.

<table>
<thead>
<tr>
<th>Program</th>
<th>std::sort</th>
<th>std::stable_sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{10}$</td>
<td>Per: 6.5 Branches: 5.3 Mispredicts: 1.47</td>
<td>Per: 6.2 Branches: 5.0 Mispredicts: 2.05</td>
</tr>
<tr>
<td>$2^{15}$</td>
<td>Ares: 6.2 Branches: 5.2 Mispredicts: 1.50</td>
<td>Ares: 5.9 Branches: 4.7 Mispredicts: 2.02</td>
</tr>
<tr>
<td>$2^{20}$</td>
<td>Per: 6.2 Branches: 5.1 Mispredicts: 1.50</td>
<td>Per: 6.1 Branches: 4.6 Mispredicts: 2.01</td>
</tr>
<tr>
<td>$2^{25}$</td>
<td>Ares: 6.1 Branches: 5.1 Mispredicts: 1.51</td>
<td>Ares: 6.1 Branches: 4.6 Mispredicts: 2.01</td>
</tr>
</tbody>
</table>

an implementation of Musser’s introsort \cite{13} and std::stable_sort is an implementation of bottom-up mergesort. In our test environment\cite{1}, for integer data, the two programs had the same speed within the measurement accuracy (see Table\cite{1}). An inspection of the assembly-language code produced by g++ revealed that in the performance-critical inner loop of std::stable_sort all element comparisons were followed by conditional moves. A *conditional move* is written in C as if \((a < b) \times=y\), where \(a\), \(b\), \(x\), and \(y\) are some variables (or constants), and \(<\) is some comparison operator. This instruction, or some of its restricted forms, is supported as a hardware primitive by most computers. By using a branch-prediction profiler (valgrind) we could confirm that the number of branch mispredictions per $n\log_2 n$—referred to as the *branch-misprediction ratio*—was much lower for std::stable_sort than for std::sort. Based on these initial observations, we concluded that mergesort is a noteworthy competitor to quicksort.

Our main results in this paper are:

1. We optimize (reduce the number of branches executed and branch mispredictions induced) the standard-library mergesort so that it becomes faster than quicksort for our task in hand (Section 2).
2. We describe an in-situ version of this optimized mergesort (Section 3). Even though an ideal translation of its inner loop only contains 18 assembly-language instructions, in our experiments it was slower than quicksort.

\footnote{The experiments discussed in the paper were carried out on two computers: \textbf{Per}: Model: Intel® Core™ 2 CPU T5600 @ 1.83GHz; main memory: 1 GB; L2 cache: 8-way associative, 2 MB; cache line: 64 B. \textbf{Ares}: Model: Intel® Core™ i3 CPU M 370 @ 2.4GHz \times 4; main memory: 2.6 GB; L2 cache: 12-way associative, 3 MB; cache line: 64 B. Both computers run under Ubuntu 11.10 (Linux kernel 3.0.0-15-generic) and g++ compiler (gcc version 4.6.1) with optimization level -O3 was used. According to the documentation, at optimization level -O3 this compiler always attempted to transform conditional branches into branch-less equivalents. Micro-benchmarking showed that in Per conditional moves were faster than conditional branches when the result of the branch condition was unpredictable. In Ares the opposite was true. All execution times were measured using gettimeofday in sys/time.h. For a problem of size \(n\), each experiment was repeated $2^{26}/n$ times and the average was reported.}