Lutz Prechelt

Comparing Java vs. C/C++ Efficiency Differences to Interpersonal Differences

The relative efficiency of Java programs is much discussed today, particularly in comparison to well-established implementation languages such as C or C++. Java is often considered very slow and memory-intensive. However, most benchmarks compare only a single implementation of a program in, say, C++, to one implementation in Java—neglecting the possibility that alternative implementations might compare differently. In contrast, this article presents a comparison of 40 different implementations of the same program, written by 38 different programmers (there are two double Java implementations). The data compares, for one particular programming task, the *average* relative performance between languages as well as the performance differences from one programmer to another within a group of programs written in the same language. As noted, these interpersonal program differences are larger than those between the languages.

Origin of the Data

The 40 program implementations investigated were created by graduate students during the course of a controlled experiment on a different question (L. Prechelt and B. Unger, "A Controlled Experiment on the Effects of PSP training: Detailed Description and Evaluation, (Jan. 1999); ftp.ira.uka.de). There are 24 programs written in Java, 11 in C++, and 5 in C. Each program was written by a single person. These programmers had an average of 8 years of programming experience and estimated that they had previously written an average of 100 KLOC each (median: 20 KLOC).

All programs implement the same functionality, namely a conversion of telephone numbers into word strings. The program first loads a dictionary of 73,113 words into memory from a flat text file (one word per line, 93KB overall). Then it reads "telephone numbers" from another file, converts them one by one, and prints the results.

The conversion is defined by a fixed mapping of characters to digits as follows:

```
e j n q r w x d s y f t a m c
i v b k u l o p g h z

0 1 1 1 2 2 2 3 3 3 4 4 5 5 6
6 6 7 7 7 8 8 8 9 9 9
```

The task of the program is to find a sequence of words such that the sequence of characters in these words exactly corresponds to the sequence of digits in the phone number. All possible solutions must be found and printed. The solutions are created word by word and if no word from the dictionary can be inserted at some point during that process, a single digit from the phone number can appear in the result at that position. Many phone numbers have no solution at all. Here is an example of the program output for the phone number "3586-75," when the dictionary contained the words "Dali," "um," "Sao," "da," "Pik," and 73,108 others:

How to Analyze the Data

Figure 1. Total memory consumption for the various programs.

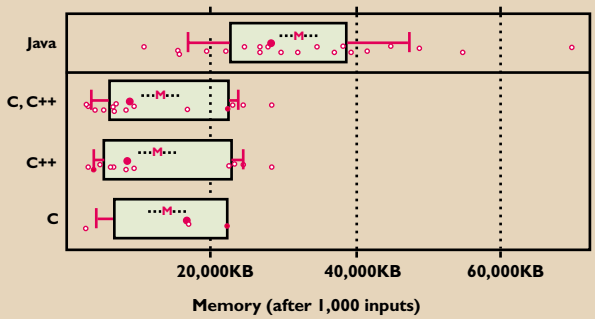


Figure 2. Runtimes for the programs, in minutes.

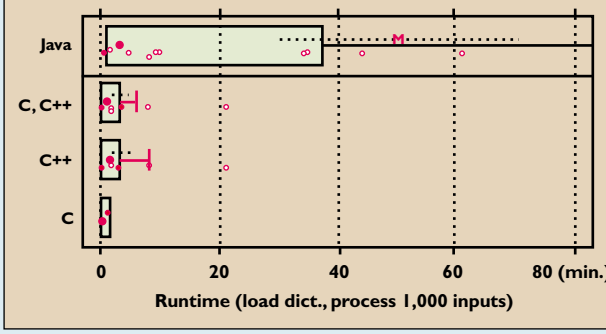
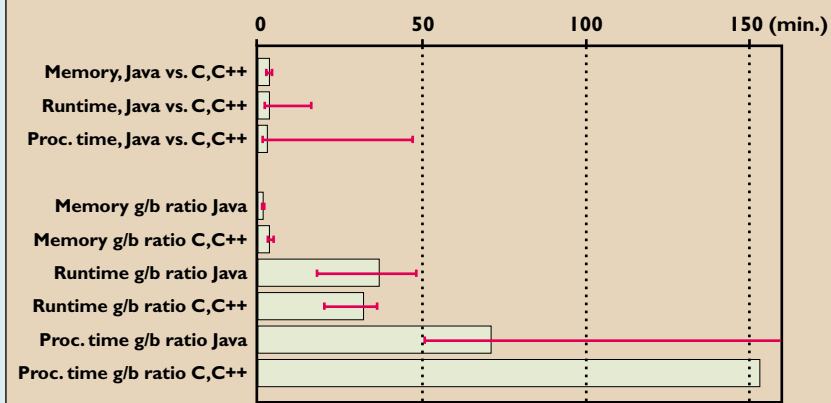


Figure 3. Ratios for each of the groups.



For each metric of interest (such as the runtime of the program) the values could be represented for each language group by their means, but in doing so a lot of interesting information would be thrown away. Therefore, the entire distribution is represented by so-called boxplots. The figures in this sidebar contain four boxplots: one for the values obtained from the Java programs, one for the C++ programs, one for the C programs, and one for the union of the C and C++ programs. The box indicates the location and extent of the "middle half" of the values, for instance, the left edge is positioned so that only 25% of the values are smaller (the 25% quantile), the right edge so that only 25% of the values are larger (75% quantile). The T-shaped whiskers indicate the 10% and 90% quantiles, respectively. The fat dot inside the box (Figures 1 and 2) is the 50% quantile—the median. The letter M in the plot represents the arithmetic mean of the data and the dashed line is plus/minus one standard error of the mean. For comparing the averages of two sets of data, the median is often more appropriate than the mean because it is not influenced by outliers.

An interesting aspect of the data is its variability. Again, the most common measure, the standard deviation, is sensitive to outliers. The width of the box is a robust measure of variability. Note that the 25% quantile (the left box edge) is the median of the lower half of the data and the 75% quantile (the right box edge) is the median of the upper half of the data. Hence, the 25% and 75% quantiles can be called average representatives of the good (efficient) and bad (inefficient) programs, respectively. The quotient of these two values I'll call the "bad/good ratio" (Figure 3). I present the bad/good ratio, because it is easier to interpret. C

```
3586-75: Dali um
3586-75: Sao 6 um
3586-75: da Pik 5
```

A list of partial solutions needs to be maintained by the program while processing each number, and the dictionary must be embedded in a supporting data structure (such as a 10-ary digit tree) for efficient access. Search functions of this kind might be part of a server in a larger client/server software system.

The programmers were asked to write as reliable a program as they could. Efficiency was less important. However, a runtime limit (not quantified to the programmers in advance) was imposed during the acceptance test and many programs failed to satisfy it on the first attempt and had to be optimized before they were accepted. All 40 programmers found that writing the program took between 3 and 63 work hours (median: 10 hours, mean: 14 hours) and the resulting program had between 107 and 614 lines (median: 244 lines, mean: 277 lines, excluding comments).

All measurements presented were taken on a Sun Ultra 1 Unix workstation with 192MB main memory running the SunOS 5.5.1 (Solaris 2.5) operating system. The C/C++ programs were compiled with the GNU gcc/g++ compiler version 2.7.2, the Java programs ran under Sun's Java development kit (JDK) 1.2 reference implementation with a just-in-time compiler (JIT). The execution of the JIT is embedded in the execution of the program, hence all data presented includes the time or memory consumed by the JIT compilation.

Memory Consumption Differences

Let's first investigate the memory requirements of the different programs. Figure 1 shows the amount of memory required by the programs after they have loaded the dictionary and processed 1,000 telephone numbers. The memory size reported includes the size of static and dynamic data structures, the program code and libraries used, and the basic process overhead. For Java programs it also contains the size of the Java Virtual Machine, including the JIT compiler.

The following are some observations:

- The average memory requirement of the Java programs is two to three times that of the C or C++ programs.
- Even the most modest Java programs require a little more memory than the average C/C++ programs and 3 to 4 times as much as the best C/C++ programs.
- The bad/good ratio is 3.7 for the C/C++ programs and 1.7 for the Java programs.

As shown, the Java programs require substantially more memory on average, but the ratio to the C or C++ programs is not larger than the good/bad ratio.

Runtime Differences

The total CPU time required by the programs (runtime) consists of two parts; one for loading the dictionary (load time) and one for actually processing the 1,000 inputs (processing time). Figure 2 shows the total runtime.

These are the main observations:

- The median runtime of the Java programs is over three times that of the C/C++ programs. Due to four huge outliers in the Java group (361, 360, 151, 130 minutes, not shown in the plot), the mean is very different from the median in the Java group so that it is 18 times the mean of the C/C++ group. The ratio of the median load time is 6 to 1 for Java versus C/C++, the ratio of median processing time is about 2 to 1.
- The three fastest Java programs are about twice as fast as the median C/C++ program and 10 times slower than the three fastest C/C++ programs.
- The runtime, bad/good ratio is 32 for C/C++ and 37 for Java. This high variability stems

NC STATE UNIVERSITY

Department of Computer Science

AT&T Solutions Fellowships

MS in Computer Networking

URL: http://www.csc.ncsu.edu/announcements/att.html

from rather huge differences in the actual search routines: The load time bad/good ratio is only 3 for C/C++ and 5 for Java, but the processing time bad/good ratio is as high as 153 for C/C++ and 71 for Java.
- The C programs are substantially faster than the C++ programs. Due to the small number of C programs, the ratio cannot be determined accurately.

The processing time (and hence the total runtime) reflects that part of the programming task that is not straightforward but which requires substantial design considerations by the programmer. The individual differences are huge compared to the differences between languages, even though the latter differences are quite large as well.

The memory, runtime, and processing time data is summarized in Figure 3. From the error bars we can learn another important lesson: Large performance ratios (like many of those presented here) are unstable even if they are estimated from a substantial number of programs. Consequently, performance comparisons based on only a single program pair should be considered highly dubious unless it is guaranteed both programs are equally well designed and appropriate for the language in which they are written.

Other Differences

The results are actually biased against the Java programs: On average, the Java programmers had only half as much programming experience in Java as the C programmers had in C or the C++ programmers had in C++. On the other hand, no clear relationship between the programmer's level of experience and the runtime or memory efficiency of the resulting program could be found in the data. The work time required for writing the program is also quite similar for the Java group versus the C/C++ group, except for three Java outliers who took over 30 hours.

The length of the resulting programs (excluding comments) is similar in all three groups but the Java programmers inserted a significantly larger amount of comments into their programs than the C++ programmers and in particular the C programmers.

I present three important conclusions from this data:

- As of JDK 1.2, Java programs are typically much slower than programs written in C or C++. They also consume much more memory.
- However, even within one language the interpersonal differences between implementations of the same program written by different programmers (bad/good ratio) are much larger than the average difference between Java and C/C++. Performance ratios of a factor of 30 or more are not uncommon between the median programs from the upper half versus the lower half. As a result, an efficient Java program may well be as efficient as (or even more efficient than) the average C or C++ program for the same purpose.
- As a consequence, it is wise to train programmers to write efficient programs and/or to ensure reasonable efficiency by means of design and code inspections.

The programming problem investigated here required a nontrivial algorithm and data structure design. However, the data clearly shows that the importance of an efficient technical infrastructure (such as language/compiler, operating system, or even hardware) is often vastly overestimated compared to the importance of a good program design and an economical programming style. C

Lutz Prechelt (prechelt@ira.uka.de) is a senior research associate at the schoool of Informatics, University of Karlsruhe, Germany.

© 1999 ACM 0002-0782/99/1000 $5.00

continued from page 25

most sighted people take for granted. An example is Home Page Reader, a talking Web browser that allows blind users to participate in the e-commerce explosion in the marketplace.

What are the sources of funding for the work of these dedicated people? Most of these projects have received support from a combination of private and government sources. Because of the success of community technology centers (CTCs) such as Plugged In and the ALA, the U.S. Congress appropriated $10 million in FY 1999 to support CTCs. The administration has proposed a $65 million budget for CTCs in FY 2000. However, tight caps on discretionary spending may put CTC funding in jeopardy. For more information about CTCs: www.ctcnet.org/ impact98. htm and www.ctcnet.org/publics.html. C

Barbara Simons (president@acm.org) is ACM's president.