

B503 Project Draft version (Sept 15th 2009)

Yuan Luo, yuanluo@indiana.edu

Overview

Your project is to write a fast routine that multiplies large non-negative integers. In addition, you must write a report describing the code, its performance, the theory behind its performance, and measurements that support your claims. By large, we mean that the code must be correct up to a million bits, though performance testing will focus more on thousands of bits. Scaffold code that uses the GMP (Gnu Multi-Precision) library for generating large numbers and testing, and that provides timing, will be provided; your job is to fill in a function that does multiplication of large numbers in standard C.

Grading

Correctness of code is essential; you will not get a good grade and may fail the project, if your code sometimes produces incorrect results. You will want to test cases with 0 as one of the operands, operands of different sizes, operands of odd (as in even or odd) bitlength, operands of 'random' length (as in don't fall into a rut of only testing the same small handful of sizes). As a matter of software engineering, you will also want to make sure you handle empty input; specifically, when GMP's random number generator returns 0, the length of the 0 is 0 bits, which is counterintuitive. (These are all problems that tripped up previous students.) (If an operand is empty, assume it has the value 0.)

Speed of code and quality of the written report are equal importance.

This project is to be turned in three times, as explained below. Turn in the paper copy of the report in class on each due-date. Email the AI (yuanluo@indiana.edu) both the report and the code, including things needed to make the code run (usually a Makefile).

Specification

The AI has code to generate problems, call your code, time it, and check the correctness of answers returned. His code will call *one* of the three following routines, declared as C functions:

```
Product32(void *A,void *B,void *C,size_t LA, size_t bA, size_t LB, size_t bB, void *LC, size_t * bC),
```

```
Product16(void *A,void *B,void *C,size_t LA, size_t bA, size_t LB, size_t bB, void *LC, size_t * bC),
```

```
Product4(void *A,void *B,void *C,size_t LA, size_t bA, size_t LB, size_t bB, void *LC, size_t * bC),
```

Here, A is the address of the digits of the first factor, B is the address of the digits of the second factor, and C is the address for the product; LA is the number of words for the first factor, LB is the number of words for the second factor, and LC is the address for storing the number of words of the product. bA, bB, bC are the number of bits in the arguments, from the highest bit: e.g. 100001, 111, 100, and 10 base 2 (33, 7, 4, 2) each take one 8-bit word, but have 6, 3, 3, and 2 bits respectively. This information is provided in case you find it useful.

'void *' and 'size_t' are C pointer and integer types.

All numbers are stored in an array with least significant digit stored at the first location. By the end of the project you will probably want to use the Product32 option since your goal is to multiply numbers as rapidly as possible (and a given number needs less digits in larger bases), but the other possibilities are supported because smaller bases lead to easier programming. Some students may want to do an initial implementation in a small base, and then switch to a large base after they have their ideas debugged.

To obtain good results for moderately large numbers, but to put an upper limit on the amount of effort the project needs, students are to use ideas as sophisticated as those in Algorithm 5.2, but are to avoid algorithms that divide integers into three or more pieces. (E.g. GMP uses a variety of algorithms depending on the size of the numbers; its Karatsuba corresponds to 5.2. The algorithms it uses beyond Karatsuba are beyond the scope of this course.)

Techniques

The logical organization of your program should have a initial routine (for example Product32), a decide routine, and routines to implement various multiplication algorithms. The initial routine will call the decide routine. Those multiplication routines that are recursive in nature will call the decide routine for its sub-problem multiplications. The decide routine will use the size of the inputs to decide which multiplication routine is fastest for doing particular multiplications. I call this the logical organization of your program, because your actual implementation is permitted to actually carry out the decide activity in other routines if you find it more efficient to use such an approach.

In your report, you need to consider the following ideas. (Of course, your final code will use only those that lead to efficient code.) In all cases, the algorithms are described in the book as being $n * n$ (n-bit integer * n-bit integer), but you will need to modify them to be $m * n$ (m-bit * n-bit).

A18basic: Algorithm 1.8, with base of 4, 16, or 32 bits, corresponding to the Product function.

A18large: Algorithm 1.8, modified to use a base size (possibly variable, depending on the size of the operands) chosen to match some multiple of the word size. This will be a recursive routine.

A52: Algorithm 5.2 modified so that each input has its own word size and modified to make the best use of the generalization discussed in the next paragraph. It will in turn need to use Algorithm 1.4 for addition. Also, modifications to do subtraction and modifications to multiply one or both inputs by powers of the base should be considered if they are needed.

A18basic will probably be easiest to implement, and serve as a useful base case for the recursive algorithms to resort to. Coding and timing, or algorithmic analysis before coding, can guide you as to when (or if) you should use A18large or A52 for fastest overall performance.

Consider the calculation:

$$U_2 = \lfloor U2^{-a} \rfloor.$$

$$U_1 = (U \bmod 2^a).$$

$$V_2 = \lfloor V2^{-b} \rfloor.$$

$$V_1 = (V \bmod 2^b).$$

$$T_1 = U_1 + 2^c U_2.$$

$$T_2 = V_1 + 2^d V_2.$$

$$W = 2^e U_1 V_1 + (2^f T_1 T_2 - 2^g U_1 V_1 - 2^h U_2 V_2) + 2^i U_2 V_2.$$

This is a generalization of eq. 5-86 from the text, where U and V are inputs, and where a, \dots, i are parameters. For some values of the parameters this calculation is not useful. For one set of values, this calculation is equivalent to eq. 5-86. Your project should consider which values of the parameters this is a correct multiplication algorithm. Your project should consider which values lead to the fastest possible multiplication algorithms (among those algorithms based on these ideas). It is also possible to change some minus signs to plus signs while changing some plus signs to minus signs and still have a correct algorithm. Such changes should be considered.

It is, of course, important to avoid explicit multiplications when multiplying by powers of the base. Rather, you should just shift the digits to the correct position. For base 10, you were taught this idea when you were taught the algorithm for multiplying multi-digit numbers. Algorithm 1.8 already has these ideas built in, but by Algorithm 5.2 it is assumed that you automatically know about doing this.

Don't forget to use compiler optimizations, especially when the code is turned in. The AI will generally be compiling and running code as you provide it, not making guesses as to what you "should" have meant. If it doesn't work, the AI will notify you, but it is then up to you to fix the problem quickly. If you make the early turn-ins that gives you a chance to get feedback from the AI, so that you can make your final project as good as it can be.

Theory and measurements

Your report should include formulas for predicting the running time of the various subalgorithm and of the overall program. Important results should be explained with theoretical analyses and verified with measurements.

The report writer should give careful consideration to the use of graphs and tables to report these results, as appropriate. Some discussion as to whether the measurements support or contract the theoretical results.

Organization

It is important to organize the report so that the reader can easily find the results that interest him. A well done report is likely to be somewhat long. If so, there is a need to start with an abstract that gives the main results, and there is a need to move minor results into appendices. Also, a table of contents can be quite helpful.

The report should be written at a level suitable for being read by typical students in the course. It is important to include all important details. Don't assume that the reader will know details like which language, compiler, and compiler options you used. Yet these details are probably important to your results. In general, you need to explain all things needed to understand the performance of your program.

It is probably an unavoidable fact of life that reports will end up being written at the last minute, but you should try to proofread your text nonetheless, especially for egregious misspelling or ungrammatical sentences, or else risk your grade suffering.

References

You will probably make use of information from other sources. Please be sure to carefully reference those other sources. If you compare your code with GMP, be sure to reference GMP. If you get help from other students, be sure to reference that help. If you find good ideas on the Internet, be sure to reference them. Having good references shows that you know how to do quality work. Not having appropriate references is an indication of cheating.

Turn-ins

- (1) Oct 7th. You should have a complete report as far as Algorithm 1.8 is concerned. Thus, you should have that code working, formulas for its speed, and measurements to support the formulas. This should all be written up clearly. Preliminary work should be started on the rest of the project.
- (2) Nov 4th. You should have code working for all routines. You should have formulas for running times and for justifying the decisions of your decide routine.
- (3) Dec 2nd. You should have the best code that you are able to produce. You should have the final version of your report.