IMPLEMENTATION OF AN OPTIMIZED AUTHENTICATED ENCRYPTION SCHEME

Alessandro Giaconia, Lorenzo Paleari, Diana Khimey, Filippo Visconti

Department of Computer Science
ETH Zürich
Zürich, Switzerland

ABSTRACT

You should write the abstract last.

1. INTRODUCTION

Motivation. In an era dominated by the exponential growth of data and the increasing prevalence of digital communication, the imperative to secure sensitive information has never been more crucial. As technology advances, so do the challenges in maintaining the confidentiality and integrity of data exchanged over networks. This project embarks on a journey through the implementation of a cutting-edge, optimized, high-performance authenticated encryption scheme. The motivation behind this endeavor stems from the need for robust cryptographic solutions, which still achieve good runtimes. Authenticated encryption schemes are a fundamental building block of modern cryptography, which are used to provide confidentiality, integrity and authenticity of data. Their use is widespread in many applications, such as TLS, SSH, IPsec, and many others. Leveraging the combination of ChaCha20 and Blake3, our project aims to prioritize efficiency and speed.

The key difference between authenticated encryption schemes and encryption schemes is that the former also provide integrity and authenticity of data, while the latter only provides confidentiality.

Related work. Next, you have to give a brief overview of related work. For a report like this, anywhere between 2 and 8 references. Briefly explain what they do. In the end contrast to what you do to make now precisely clear what your contribution is.

2. BACKGROUND: WHATEVER THE BACKGROUND IS

In this section, we give a brief overview of the background necessary to understand the rest of the report. In particular, we introduce the concepts of encryption and authentication, and we explain the algorithms we use.

Encryption.

Authentication. Authentication is the process through which the integrity and origin of data are verified, to ensure that is has not been tampered with, and to confirm that it was sent by the expected sender. This process is critical in various fields, such as secure communication, electronic commerce, and many others.

In order to achieve authentication, a Message Authentication Code is usually used. A MAC is a short piece of information that is used to authenticate a message, in combination with a secret key. The receiver of the message can verify the authenticity of the message by recomputing the MAC and comparing it with the original MAC.

Blake3. We chose Blake3 for the authentication part of the scheme, due to its high performance and security guarantees. By design, Blake3 is also highly parallelizable, which makes it a good candidate for our implementation. For the implementation, we used the C language and two different approaches: one implementation is using a stack, which is more versatile, and the other one uses a more efficient divide-and-conquer approach, which is a bit less flexible, despite performing much better. Blake3 is a cryptographic hash function which also provides a MAC function, the one we're interested in. At a high level, Blake3 works by splitting the input into chunks of size up to 1024 bytes, and then compressing them using a compression function, which performs a series of operations on the input, and returns a 64B output, named the chaining value. The chaining value is then used as input for the next chunk, and the process is repeated until all the chunks have been processed. These chunks form the leaves of a binary tree, which is traversed from the bottom up, until the root is reached, by combining the chaining values of the children nodes, two at a time. After a number of iterations, we're left with a single node, the root node, whose chaining value is the output of the algorithm. By repeating the last call to the compression function with an increased counter, we can obtain an output of arbitrary length.

3. YOUR PROPOSED METHOD

In this section, we dive into the details of our implementation. We describe how we implemented the algorithms, and we explain the optimizations we used to achieve high performance. **Blake3.** In this subsection, we describe our implementations of the Blake3 algorithm, which are based on and closely follow the original Blake3 paper that describes the algorithm in detail. Given their different strenghts and characteristics, we also included a dispatcher that chooses the best implementation based on the input size, to always guarantee a correct output. **Stack version.** TODO

Divide-and-conquer version. The divide-and-conquer version of the algorithm is based on the assumption that all of our input is available at the beginning of the computation. This allows us to evenly distribute the work among the threads, which have no dependencies on each other, since each thread works on a different portion of the input. The threads work in parallel until they reach the base case, which is when the input size is small enough to be processed sequentially. At this point, the threads return the result to the parent thread, which combines them and returns the final result.

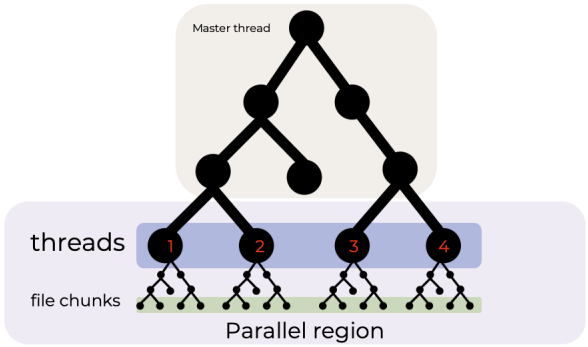

Fig. 1. High-level visualization of the divide and conquer approach

To achieve this, we use the OpenMP library, which allows us to easily parallelize the computation once we have created the parallel region. In addition to that, we vectorized the compression function, which is the most computationally intensive part of the algorithm, using the AVX instruction set and the code available on GitHub from the reference implementation.

The main advantage of this approach is that it's very scalable, since the work is evenly distributed among the threads, and the threads have no dependencies on each other. This allows us to achieve very good speedups, as we will see in the results section. The main disadvantages are that it's not very flexible, since it requires all of the input to be available at the beginning of the computation (although this is not a limitation for us), and that is less flexible, as, to achieve its maximum potential, it requires the input to be full trees. TODO Mention and cite any external resources that you used including libraries or other code.

4. EXPERIMENTAL RESULTS

Here you evaluate your work using experiments. You start again with a very short summary of the section. The typical structure follows.

Experimental setup. Specify the platform (processor, frequency, maybe OS, maybe cache sizes) as well as the compiler, version, and flags used. If your work is about performance, I strongly recommend that you play with optimization flags and consider also icc for additional potential speedup.

Then explain what kind of benchmarks you ran. The idea is to give enough information so the experiments are reproducible by somebody else on his or her code. For sorting you would talk about the input sizes. For a tool that performs NUMA optimization, you would specify the programs you ran.

Results. Next divide the experiments into classes, one paragraph for each. In each class of experiments you typically pursue one questions that then is answered by a suitable plot or plots. For example, first you may want to investigate the performance behavior with changing input size, then how your code compares to external benchmarks.

For some tips on benchmarking including how to create a decent viewgraph see pages 22–27 in [1].

Comments:

- Create very readable, attractive plots (do 1 column, not 2 column plots for this report) with readable font size. However, the font size should also not be too large; typically it is smaller than the text font size. An example is in Fig. 2 (of course you can have a different style).
- Every plot answers a question. You state this question and extract the answer from the plot in its discussion.
- Every plot should be referenced and discussed.

5. CONCLUSIONS

Here you need to summarize what you did and why this is important. *Do not take the abstract* and put it in the past tense. Remember, now the reader has (hopefully) read the report, so it is a very different situation from the abstract. Try to highlight important results and say the things you really want to get across such as high-level statements (e.g., we believe that .... is the right approach to .... Even though

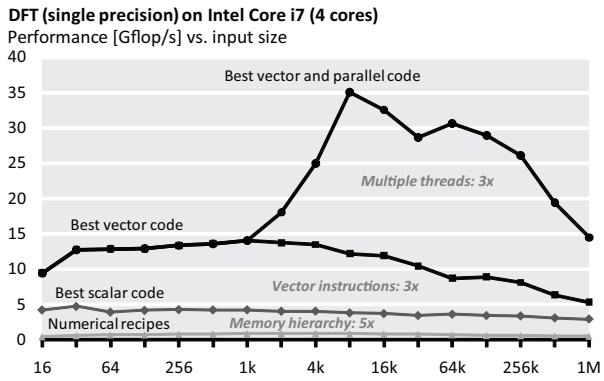

Fig. 2. Performance of four single precision implementations of the discrete Fourier transform. The operations count is roughly the same. The labels in this plot are maybe a little bit too small.

we only considered x, the .... technique should be applicable ....) You can also formulate next steps if you want. Be brief. After the conclusions there are only the references.

6. FURTHER COMMENTS

Here we provide some further tips.

Further general guidelines.

- For short papers, to save space, I use paragraph titles instead of subsections, as shown in the introduction.
- It is generally a good idea to break sections into such smaller units for readability and since it helps you to (visually) structure the story.
- The above section titles should be adapted to more precisely reflect what you do.
- Each section should be started with a very short summary of what the reader can expect in this section. Nothing more awkward as when the story starts and one does not know what the direction is or the goal.
- Make sure you define every acronym you use, no matter how convinced you are the reader knows it.
- Always spell-check before you submit (to us in this case).
- Be picky. When writing a paper you should always strive for very high quality. Many people may read it and the quality makes a big difference. In this class, the quality is part of the grade.
- Books helping you to write better: [2] and [3].
- Conversion to pdf (latex users only):

  dvips -o conference.ps -t letter -Ppdf -G0 conference.dvi

  and then

  ps2pdf conference.ps

Graphics. For plots that are not images *never* generate the bitmap formats jpeg, gif, bmp, tif. Use eps, which means encapsulate postscript. It is scalable since it is a vector graphic description of your graph. E.g., from Matlab, you can export to eps.

The format pdf is also fine for plots (you need pdflatex then), but only if the plot was never before in the format jpeg, gif, bmp, tif.

7. REFERENCES

[1] M. Püschel, "Benchmarking comments," online: http://people.inf.ethz.ch/markusp/teaching/263-2300-ETH-spring11/slides/class05.pdf.

[2] N.J. Higham, *Handbook of Writing for Mathematical Sciences*, SIAM, 1998.

[3] W. Strunk Jr. and E.B. White, *Elements of Style*, Longman, 4th edition, 2000.