High-Performance Implementation and Evaluation of Blowfish Cryptographic Algorithm on Single-Chip Cloud Computer: A Pipelined Approach

Kamak Ebadi          Victor Pena          Chen Liu
Department of Electrical and Computer Engineering
Florida International University
Miami, Florida, USA
E-mail: {kebad001, vpena005}@fiu.edu
Department of Electrical and Computer Engineering
Clarkson University
Potsdam, New York, USA
E-mail: cliu@clarkson.edu

Abstract – In the era of modern data communications, as the need for data security arises, the need to reduce the execution time and computation overhead associated with the execution of cryptographic algorithms increases correspondingly. Parallelizing the computation of cryptographic algorithms on many-core computing platforms can be a promising approach to reduce the execution time and eventually the energy consumption of such algorithms. In this paper, we build a pipelined model to analyze and compare the execution time and energy consumption of the Blowfish cryptographic algorithm on the Single-Chip Cloud Computer (SCC), an experimental processor created by Intel Labs. In this model the Blowfish cryptographic algorithm is divided to smaller chunks and each chunk is run only by one core. Using message passing interface, the input data passes in turn through all the cores involved. Due to the communication overhead and latency associated with this model, we experimented and identified the optimal message size to pass between the cores to avoid saturating the on-chip communication network. Our results illustrate that our parallel approach is 27X faster than the sequential approach and yields close to 16X less energy consumption on the SCC platform.

Keywords: Cryptography, Blowfish, Energy Efficiency, MPI, Pipeline, SCC

I. INTRODUCTION

In the new era of data-intensive computing, global data communications, and inexpensive internet connections, there is a higher demand for data security, energy efficiency, and computational speedup. Cryptography plays an important role for protecting data from destructive forces and the unwanted actions from unauthorized users. In an effort to respond to the ever-increasing need for data security, cryptographic algorithms become mathematically more and more complex with time. However, the increase in the complexity of such algorithms incurs more computation overhead, which leads to a longer execution time and therefore higher energy consumption of the computing system. In recent years, successful studies have been made using hardware acceleration techniques to speed up the execution of cryptographic algorithms. A general hardware-assisted approach was presented by Tang et al. [8, 19-25]. However, their design mainly focused on accelerating the garbage collection function with exploiting prefetch techniques in the middleware layer, which is for a different application domain comparing with ours. Ro et al. [26, 27] presented a decoupled architecture design to remedy the limits of traditional data prefetching methods, improving the overall memory access latency. Liu et al. [1] explored and presented implementation techniques for energy-efficient hardware acceleration of RSA and Blowfish cryptography. They were able to reduce the energy consumption by 9.6% for RSA and 36.0% for the Blowfish cryptographic algorithms, separately. However, their approach is based on co-processor design on an FPGA platform. This could incur extra hardware overhead and power consumption overhead, while in this work we focus on future many-core architecture.

Increasing the number of cores on a single chip can increase the computational speed and improve the energy consumption of the system. Using techniques to increase the energy efficiency in many-core systems will reduce energy consumption and excess heat, lower operational costs and improve system reliability [2]. The emergence of many-core systems provides the opportunity to revisit the realization of high-impact computing problems on more capable hardware [16]. There are major benefits from parallel computing on many-core platforms. The advantage of such a system lies in its ability to handle large and extremely complex computations. However, the big challenge is not only developing powerful many-core hardware architecture, but also developing applications that could effectively run in parallel and take advantage of the capabilities offered by many-core architectures. The idea which serves as our motivation for this paper is to examine if complicated cryptographic algorithms could split up their tasks to run in parallel successfully, so that they achieve faster execution and less energy consumption. In this paper, using the message-passing data type (MPDP) on a many-core platform, we
explore and present a pipelined implementation technique for an energy-efficient Blowfish algorithm.

The rest of the paper is organized as follows. In Section II we briefly review the Blowfish cryptographic algorithm. Section III presents our experimental setup and an overview of the Single-Chip Cloud Computer platform. Section IV presents our methodology. In Section V, we present our results and conclusions are presented in Section VI.

II. BLOWFISH CRYPTOGRAPHIC ALGORITHM

Cryptography plays an important role in the security of any sensitive communication and data transmission. It can be defined as the conversion of data into a scrambled code that can be deciphered and sent across a public or private network. This is typically done with the use of keys [5]. In cryptography, keys are used to encrypt a message into a format, which would appear as unreadable random information to an unauthorized third-party.

Cryptography uses two main forms of encrypting data; symmetrical and asymmetrical. Symmetric algorithms are significantly faster than asymmetric ciphers because they use the same key for encryption as they do for decryption. Blowfish cryptographic algorithm [4], which was designed by Bruce Schneier in 1993, is a symmetric block cipher that divides a message up into fixed-length blocks of 64-bit during encryption and decryption processes, as shown in Figure 1.

![Figure-1 Representation of Blowfish Cryptographic Algorithm [6]](Image 61x68 to 567x510)

The Blowfish algorithm consists of two parts: a key-expansion part and a data-encryption part. Key expansion converts a variable-length key of at most 448 bits into several subkey arrays, totaling 4168 bytes. The algorithm uses Feistel cipher where the input text is split into two halves. The first half is applied round function using a subkey. The output will be XORed with the second half. Then the two halves will be swapped. In total there are 17 rounds and each round consists of a key-dependent permutation and a key- and data-dependent substitution. Hence, this serves as the candidate in our pipelined model.

By the end of the 17th round, the 64-bit cipher text will be produced [17]. The Feistel network of Blowfish algorithm is one that utilizes a structure that makes encryption and decryption very similar through the use of the following elements [3, 18]:

- P-box: Permutation box that performs bit shuffling;
- S-box: Substitution box for non-linear functions;
- XOR: Logic function to achieve linear mixing.

![Figure-2 Representation of F function [6]](Image 47x423 to 292x409)

Figure-2 shows a graphical representation of the $F$ function, which has been shown as the most accessed function of the Blowfish algorithm [1]. It requires a 32-bit input data to be decomposed into four 8-bit blocks. Each block references an S-Box and each entry of the S-Box outputs a 32-bit data. First, the output of S-Box 1 and S-Box 2 are added. Then the result of the addition is XORed with S-Box 3. Finally, S-Box 4 is then added to the output of the XORed operation and provides a 32-bit output.

III. EXPERIMENTAL SETUP

Even though increasing the clock frequency seemed to be a promising approach to decrease the execution time of computationally intensive tasks, however, the excessive power demand and high heat dissipation associated with operating at higher frequency levels has limited the usability of this approach. Utilizing higher core counts instead of increasing the clock frequency can be a promising approach to increase the total execution speed of programs. However, as described by Amdahl’s Law [15], possible speedup gains are limited by the fraction of the software that cannot be parallelized to run on multiple cores simultaneously. This
means the improvement in performance gained by the use of a multi-core processor depends on the software algorithms used and their implementation.

A multi-core processor is a single computing component with two or more independent actual cores that can run multiple instructions at the same time and increase overall speed for programs. Multi-core processors are widely used across many application domains including general-purpose, embedded, network and digital signal processing. The Single-Chip Cloud Computer (SCC) experimental processor [7] is a 48-core ‘concept vehicle’ created by Intel Labs as a platform for many-core software research. It is the second-generation processor design that has been successfully developed by Tera-Scale Computing Research Program [10]. The 48-core SCC processor is a true hardware/software co-design that can support a wide variety of software research projects, which is intended to encourage more research on many-core processor and parallel programming. The SCC platform allows the implementation and study of parallel applications by supporting a message-passing programming model for communication among the cores.

The message passing interface (MPI) is a message passing library interface specification that provides parallel process management and data exchange capabilities [14]. MPI is not an IEEE or ISO standard, but has in fact become the industry standard for writing message passing programs on HPC platforms. For our study, MPI is the key feature to implement our pipelined model. The message-passing library developed for SCC is called RCCE, which is an API library for the message passing programming model provided with the SCC [9] and it uses shared memory to send and receive the messages [14].

SCC also supports dynamic voltage and frequency scaling (DVFS). As Figure 4 represents, every tile can be considered as a frequency domain on the SCC platform, which means it can operate at a different frequency level and every 4 tiles can form a voltage domain that can operate on a different voltage level. For every voltage level, there is a range of supported frequencies. This enables us to minimize the amount of wasted energy on the SCC platform by selecting an appropriate voltage and frequency level for each of the computing cores and the idle cores.

![Figure-4 Frequency, Voltage and Memory domains on SCC Platform](image)

Based on the Blowfish algorithm, in our pipelined model we are not going to utilize all the 48 cores available on the SCC; thus, there are some idle cores that are not participating in the execution and computation of the program. However, when measuring the total energy consumption of the SCC, the power consumed by the idle cores is also taken into account. Due to the limitation that we cannot shut down the idle cores completely on the SCC, therefore, in order to minimize the amount of wasted energy by idle cores, we set the voltage and frequency levels of those idle cores to the lowest stable operational frequency of 100MHZ with the supplying voltage of 0.7V, while active cores are operating at the frequency of 533MHz with the supplying voltage of 0.9 V.

IV. METHODOLOGY

To setup our experiment, firstly, we run non-pipelined Blowfish algorithm on a single core to measure the execution time, power, and energy consumption of the SCC platform. The results serve as our baseline to compare with the results of our proposed model. There are several approaches to achieve parallelism depending on the nature of each cryptography algorithm. Based on the study of the Blowfish cryptographic algorithm, we propose a software-pipelined approach in order to achieve computation parallelism on the SCC platform. Setting up a pipeline serves to reduce the amount of computing that a single processing element needs to perform before switching to the next data chunk. This leads to an increase in the possible throughput of the program by running multiple tasks on multiple cores as well as the reduction in the computation.
requirement on a single processing element. Pipelining is particularly effective when data to be processed is available progressively such as multimedia streaming applications, audio processing applications and cryptographic algorithms.

As explained earlier in Section II, the Blowfish cryptography algorithm consists of 17 rounds, where by the end of the last round the cipher text is generated. In our experiment a plaintext file is enciphered and then deciphered so that the output is the same as the input file. This means there are another 17 rounds needed to decipher the file and therefore in total we can identify 34 rounds for the entire encryption/decryption process based on the Blowfish cryptographic algorithm. Figure 5 illustrates how the Blowfish algorithm is divided to 17 separate chunks of computation.

According to Figure 5, our pipelined model functions as follows:

- Each core receives a single chunk of data from the previous stage,
- Perform the assigned cryptographic operations on received data chunk,
- Send the data to the next stage in the pipeline.

In our pipelined model, sending and receiving steps are synchronous, which means all the cores are almost simultaneously performing one of the three steps described above. However, the first and the last stages of the pipeline are different: the first stage reads a chunk from the input buffer, while the last stage writes to an output buffer. In our design, Core 0 is assigned to receive the text file from input buffer and Core 29 sends the data to the output buffer. Figure 6 represents the flow of our pipelined model.

Please note after Core 25, we send the data to Core 36, 37, 38, 39, and then come back to Core 26, 27, 28, 29. So in total we still utilize 36 cores, corresponding to the 34 rounds for encipher and decipher with 2 cores for input and output. The reason for this topology is to reduce the overall communication overhead. As explained before, there is no extra hop between two cores on the same tile, Therefore, in order to minimize the execution time and delay associated with the communication overhead between the cores, all the communicating cores are located maximum one hop away from each other.

However, the available bandwidth for the message passing is limited; thus, in the case of using large data chunks, communication overhead can cause bandwidth saturation.
and higher latency. This can eventually lead to a longer execution time of the pipelined model, comparing to the sequential model where the program runs on only one core. In our design, the input file is split into different number of chunks and each chunk of data undergoes a series of computations based on the Blowfish algorithm. Each core is responsible for executing only one round of computations associated with the Blowfish algorithm and the data is then sent to the next core. In order to characterize and analyze the latency of the mesh network between the cores, we split the 1MB plaintext file into different chunk sizes. Figure 8 illustrates the change in the behavior of the bandwidth between the cores by shifting from large number of small chunks to small number of large chunks.

![SCC Mesh Bandwidth](image)

Figure-8 Illustrates the behavior of the bandwidth on the SCC based on the different data chunk sizes.

As we can see from the figure, the optimum data chunk size to achieve the highest bandwidth rate on the mesh network of the SCC in this case is 8KB. By using smaller chunk sizes, the bandwidth is under-utilized; while by using chunk sizes larger than 8KB, the latency will increase as the results of the bandwidth congestion.

In order to observe the execution time of the Blowfish cryptographic algorithm, we modified the design to insert a RCCE timer that starts counting only when the root core (Core 0) starts receiving the input text file and stops when the entire cryptographic algorithm (encryption/decryption) is completed and the output is generated by the last core (Core 29).

The power measurement is performed after all the executing cores have finished their computation to avoid the interruption. The power measurement involves the product of the total voltage on the SCC chip and the total current flowed through the SCC chip. At the end of the program, the overall energy consumption was calculated for both pipelined and sequential approach.

V. EXPERIMENTAL RESULTS

The experimental results are separated into two categories; the pipelined and the sequential approach. As explained earlier, in both scenarios, active cores were operating on the frequency of 533MHz with the supplying voltage of 0.9 V.

![Total Execution Time](image)

Figure-9 represents the execution time comparison of pipelined and sequential approach, using the message size of 8KB.

Figure 9 represents the execution time comparison between the sequential and the pipelined approach. As we can see, the pipelined approach runs 27.14 times faster than the sequential approach. While it takes for the sequential approach to run the program in 1.904956 seconds, the proposed pipelined model achieves to finish the program in only 0.0702 seconds.

![Total Power Consumption](image)

Figure-10 represents the power consumption comparison of the pipelined and the sequential approach on the SCC platform, using the message size of 8192 Bytes.

Figure 10 presents the relationship between the number of cores utilized for the execution of the Blowfish algorithm and the average power consumption of the entire execution. When utilizing 36 cores for the pipelined approach, the total power consumption of the SCC is equal to 30.24 Watts, while by running the program on only one core for the sequential approach and setting the idle cores on the lowest frequency level with supplying voltage, the total power consumption is 20.56 Watts. So the pipelined approach does incur a 1.47 times increase in power consumption.

The energy consumption is calculated by multiplying the total execution time with the total power consumption of the
SCC based on the two different approaches. As explained in Figure 10, by using more core counts the total power consumption of the system increases; however, since the program is running in parallel and all the cores are operating for a period of time much shorter than the time needed for a single core to complete the task, thus, it still could lead to a significant energy saving.

![Total Energy Consumption](image)

Figure 11 represents the total energy consumption comparison of the pipelined and the sequential approach on the SCC platform, using the message size of 8192 Bytes.

As Figure 11 presents based on our design, the SCC platform consumes only 2.476 joules when running the Blowfish algorithm in parallel on 36 cores; while the total energy consumption of the SCC increases 15.82 times and it utilizes 39.165 joules when running the same program on one core only.

VI. CONCLUSIONS

To meet the ever increasing need of data security, cryptographic algorithms are getting more mathematically complicated day after day. On the other hand, the need for a reliable, energy-efficient, and fast computation is another necessity of our time. There is a significant improvement in performance gained by the use of multi-core systems. With multi-core and many-core systems becoming mainstream, we can make use of such systems for a faster and more energy-efficient computation of complicated cryptographic algorithms. However, possible performance gains are limited by the fraction of the software that can be run in parallel, since the rest of the program still needs to run sequentially. Also, this gain is only possible if the computation overhead of the program is greater than the on-chip communication overhead. For programs that are not computation intensive, parallelism may not be a promising approach. This is mainly because of the nature of the implementation of the parallelism on many-core systems, which is based on message passing between the cores. In this paper, to achieve parallelism on the SCC platform we proposed a pipelined model for the implementation and execution of the Blowfish cryptographic algorithm. As our results proved, this model yields a significant energy saving as well as a faster execution time of the program on the Single-Chip Cloud Computer platform.

ACKNOWLEDGMENT

We would like to thank Intel Labs for providing us with SCC platform to conduct this research. This work is partly supported by the National Science Foundation under grant number ECCS-1125762. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

18-20, 1967, spring joint computer conference (AFIPS '67 (Spring)).
ACM, New York, NY, USA, 483-485.

execution model and runtime for heterogeneous many core systems. In
Proceedings of the 17th international symposium on High performance
distributed computing (HPDC '08). ACM, New York, NY, USA, 197-200.

September 1995, Dr. Dobb's Journal.

[18] Schneier, Bruce. Applied Cryptography. 2. s.l. : John Wiley & Sons,
1996.

[19] Jie Tang, Pollawat Thanarungroj, Chen Liu, Shaoshan Liu, Zhimin
Gu and Jean-Luc Gaudiot, “Pinned OS/Services: A Case Study of XML
Parsing on Intel SCC”, Journal of Computer Science and Technology,
2012.

[20] Jie Tang, Shaoshan Liu, Chen Liu, Zhimin Gu and Jean-Luc Gaudiot,
“Acceleration of XML Parsing Through Prefetching”, IEEE Transactions

[21] Jie Tang, Shaoshan Liu, Zhimin Gu, Chen Liu and Jean-Luc Gaudiot,
“Memory-Side Acceleration for XML Parsing”, The 8th IFIP International
Conference on Networking and Parallel Computing (NPC 2011),
Changsha, Hunan, China, October 21-23, 2011.

[22] Shaoshan Liu, Jie Tang, Ligang Wang, Xiao-Feng Li and Jean-Luc
Gaudiot, “Packer: Parallel Garbage Collection Based on Virtual
Spaces”, IEEE Transactions on Computers, DOI: 10.1109/TC.2011.193,
October, 2011

[23] Jie Tang, Shaoshan Liu, Zhimin Gu, Chen Liu, and Jean-Luc Gaudiot,
“Prefetching in Embedded Mobile Systems Can Be Energy-Efficient”,
Computer Architecture Letters, February, 2011. DOI:
http://doi.ieeecomputersociety.org/10.1109/L-CA.2011.2.

[24] Jie Tang, Shaoshan Liu, Zhimin Gu, Xiao-Feng Li, and Jean-Luc
Gaudiot, “Hardware-Assisted Middleware: Acceleration of Garbage
Collection Operations”, Proceedings of the 21st IEEE International
Conference on Application-Specific Systems, Architectures and Processors
(ASAP 2010), Rennes, France, July 7-9, 2010.

[25] Shaoshan Liu, Jie Tang, Chengrui Deng, Xiao-Feng Li and Jean-Luc
Gaudiot, “RHE: A JVM Courseware”, IEEE Transactions on Education,
Issue: 99, DOI: 10.1109/TE.2010.2047946, April, 2010

[26] Won W. Ro, Stephen P. Crago, Alvin M. Despain, and Jean-Luc
Gaudiot, “Design and Evaluation of a Hierarchical Decoupled
259, December 2006.

[27] Won W. Ro and Jean-Luc Gaudiot, “SPEAR: A Hybrid Model for
Speculative Pre-Execution”, Proceedings of the 18th International Parallel
and Distributed Processing Symposium (IPDPS 2004), Santa Fe, New