2010 12th IEEE International Conference on High Performance Computing and Communications

A CellBE-based HPC application for the analysis of vulnerabilities in cryptographic hash functions

Alessandro Cilardo, Luigi Esposito,
Antonio Veniero, Antonino Mazzeo
Department of Computer Science,
University of Naples Federico II,
via Claudio 20, 80125 Naples, Italy.
Email: acilardo@unina.it

Vicenç Beltran, Eduard Ayguadé
Barcelona Supercomputing Center
C/Jordi Girona 1-3,
08034 Barcelona, Spain.

***Abstract*—After some recent breaks presented in the technical literature, it has become of paramount importance to gain a deeper understanding of the robustness and weaknesses of cryptographic hash functions. In particular, in the light of the recent attacks to the MD5 hash function, SHA-1 remains currently the only function that can be used in practice, since it is the only alternative to MD5 in many security standards. This work presents a study of vulnerabilities in the SHA family, namely the SHA-0 and SHA-1 hash functions, based on a high-performance computing application run on the MariCel cluster available at the Barcelona Supercomputing Center. The effectiveness of the different optimizations and search strategies that have been used is validated by a comprehensive set of quantitative evaluations, presented in the paper. Most importantly, at the conclusion of our study, we were able to identify an actual collision for a 71-round version of SHA-1, the first ever found so far.**

I. Introduction

Cryptographic hash functions, including for example MD5 and SHA-1, have an essential role in security-critical applications, such as public-key infrastructures. They are expected to be practically impossible to "invert" and there must be virtually a unique correspondence between a given message and its hash. Surprisingly, recent results [18], [7], [14] have shown that some sophisticated and computation-intensive techniques can actually lead to collisions (i.e. two different messages having the same hash value) for the hash functions used in current applications. The studies on the vulnerabilities of the SHA-1 function, in particular, are taking on a decisive role in security research. In fact, due to the numerous breaches discovered in MD5, SHA-1 is currently the only hash function that can be practically used, since it is the only alternative to MD5 in many standards. A deeper understanding of the robustness and weaknesses of SHA-1 is thus of paramount importance, and is also necessary to drive future standardization initiatives and develop next-generation functions overcoming the vulnerabilities identified in today's hash functions. The work presented in this paper addresses the study of vulnerabilities in the SHA family, namely the SHA-0 and SHA-1 hash functions, by identifying new search strategies that can decrease the computational workload required for finding a collision.

As better described in Section III, the whole collision search process is divided in two phases. The first one leads to a so-called *differential characteristic*, having the essential purpose of limiting the search space, while the second phase uses a number of techniques and search strategies in order to efficiently explore the space. During our work, we have built a set of ad-hoc tools to implement the first phase and generate suitable differential characteristics. For the second phase, which has critical performance requirements, we targeted a high-performance computing facility, namely the MariCel cluster, available at the Barcelona Supercomputing Center, based on the Cell Broadband Engine architecture. The effectiveness of the different optimizations and search strategies is validated by a comprehensive set of quantitative evaluations presented in the last part of the paper, and –most importantly– by an actual collision identified for a 71-round version of SHA-1, the first ever found so far. As a result of the techniques developed during our study, moreover, we were able to find such collisions at a relatively small computational cost.

The rest of the paper is structured as follows. Section II summarizes the current state-of-the-art related to the exploitation of vulnerabilities in cryptographic hash functions. Section III gives an overview of the collision search process. Section IV presents the HPC facility used for computation-intensive tasks. Section V provides a thorough description of the highly parallel application developed to drive our study, along with several techniques and optimizations carried out to maximize the performance on the target parallel architecture. Section VI gives a detailed quantitative evaluation showing the impact of the most significant optimizations adopted. It also presents a 71-round collision that has been identified to demonstrate the effectiveness of the proposed techniques. Section VII concludes the paper with some final remarks.

II. Background

In current applications, there are two widely used cryptographic hash functions: MD5 [12] and SHA-1 [11]. Both are iterative hash functions based on the Merkle-Damgård construction along with a compression function. The compression function requires two fixed size inputs, namely a k-bit message block and a n-bit Intermediate Hash Value (internal

978-0-7695-4214-0/10 $26.00 © 2010 IEEE
DOI 10.1109/HPCC.2010.113

state between message blocks, denoted as IHV), and outputs the updated IHV.

MD5 was designed by R. Rivest in 1991. In 1993 some weaknesses in its design were pointed out. In 2004, in particular, X. Wang et al. generated a collision for MD5, presented at EuroCrypt 2005 in [19]. Authors in [15] and [16] then extended this collision construction method, leading to the surprising result presented in [14] at the beginning of 2009. The authors devised a practical attack scenario, where they successfully created a rogue digital identity certificate issued by an unaware real-world Certification Authority (CA) and trusted by all common web browsers. The rogue certificate would allow the attacker to impersonate any website on the Internet, including banking and e-commerce sites secured through the HTTPS protocol.

SHA-1 was issued by the NIST in 1995 as a Federal Information Processing Standard [11] as a new and more reliable function to be used in cryptographic applications. It is based on similar design principles as MD5, although it produces longer digests (160 bits) and overcomes the vulnerabilities then identified for its short-lived predecessor SHA-0 and for MD5. Since its publication, SHA-1 has been adopted by many governments and industry security standards, in particular standards on digital signatures for which a collision-resistant hash function is strictly required.

Although much more robust than MD5, the SHA family is not immune from risks. In 1998 Chabaud and Joux [5] proposed a cryptanalysis technique on the full SHA-0, with a complexity of 2^{61}. It is a differential attack that uses a weakness of the expansion algorithm of SHA-0, and is faster than the generic birthday paradox attack. Biham et al. in [2] and [3] improved these results, with the use of *neutral-bits*. Authors presented a reduced complexity approach to find collisions for SHA-0 and weakened versions of SHA-1. For the first time, in [18], Wang et al. showed a method to find a collision for the standard SHA-1 80-round version with a complexity lower than the theoretical bound of 2^{80}, namely 2^{69}, using the *Non-Linear Connecting Characteristics*. Although this complexity is still out of reach, these results significantly influenced subsequent attacks against SHA-1 [17] and MD5 [4]. Authors in [20] exploited a technique called *message modification* to reduce the complexity of attacks against SHA-1. De Cannière et al. [7] then described a way to automatically find complex Non-Linear characteristics and used it to determine a two-block colliding message pair for a weakened 64-step version of SHA-1. The same researchers then presented a collision for a 70 round version in [6], while an equivalent result was obtained by S. Manuel and T. Peyrin in [10]. An interesting initiative concerning SHA-1 collisions is the "SHA-1 collision search project", hosted by the University of Graz, Austria [13]. It is a distributed computing project based on BOINC, an open source distributed computing framework that allows volunteers on the Internet to join a project and donate CPU-time. The project started in 2007, exploiting the achievements on SHA-1 vulnerabilities available at that time. However it was put on hold in May 2009 due to lack of progress.

III. Collision search process

The purpose of a hash collision search program is to find a pair of messages producing the same hash value. A naïve approach would consist, of course, in searching the whole space of message pairs until a colliding one is found. With an n-bit hash function, this would require, roughly, $2^{(n/2)}$ hash evaluations (the so-called *birthday paradox*), which is of course infeasible with any current hash function. Instead of testing arbitrary pairs, existing approaches try to exploit the internal structure of the hash function to locate a special subset of message pairs which (1) are considerably more likely to collide than random pairs, and (2) can be efficiently enumerated.

Before presenting our collision search strategy and the parallel application proposed, we give an overview of the core structure of SHA family hash functions. While a full description of the algorithm is given in [11], we will refer to a restructured version which is especially suitable for cryptanalysis, as described in [2], [7]. Essentially, the function takes a 512-bit message block, divided into sixteen 32-bit words, and expands them into eighty 32-bit words, where the first sixteen coincide with the initial message, and the remaining words are directly derived from the first message words by means of a "message expansion function". Call W the overall sequence of eighty 32-bit words of the expanded message. SHA-0 and SHA-1 only differ in the way the 80 words of the expanded message are generated. The main loop in both functions is made of 80 rounds. During each round, a state variable -call it A- is updated based on the previous 5 values of A and the corresponding word of W, which are used as input to a "compression function" producing the new value of A. For the first few rounds, where some of the previous values of A do not exist, a set of five standard inizialization constants are used. The last five values of A (rounds 76 through 80) are –after reorganization and chaining operations– the output of the hash function. An illustration of this process is given in Figure 1. The above process can be iterated on more than one 512-bit message block, by using the last five words obtained for one block as the initialization values for the subsequent one. This allows the hash function to be applied to messages of arbitrary length.

The essential idea behind the attack to SHA is to constrain the values of the messages and registers in order to reach a collision with a certain probability. Such set of bit-level constraints on the difference of two messages is called "differential characteristic" (an example is shown in Figure 2). A differential characteristic is organized into two sections. The $\mathcal{W}$ part defines the constraints imposed between the bits of the two messages in each position, while the $\mathcal{A}$ part contains the constraints forced on the internal registers during the hashing process. The constraints are expressed by a set of symbols. '1' and '0' indicate that both bits in the two messages must take on the values 1 and 0, respectively, 'u' and 'n' indicate a signed difference ($1/0$ or $0/1$, respectively), 'x' an unspecified difference, '$-$' two unspecified equal values. Notice that,

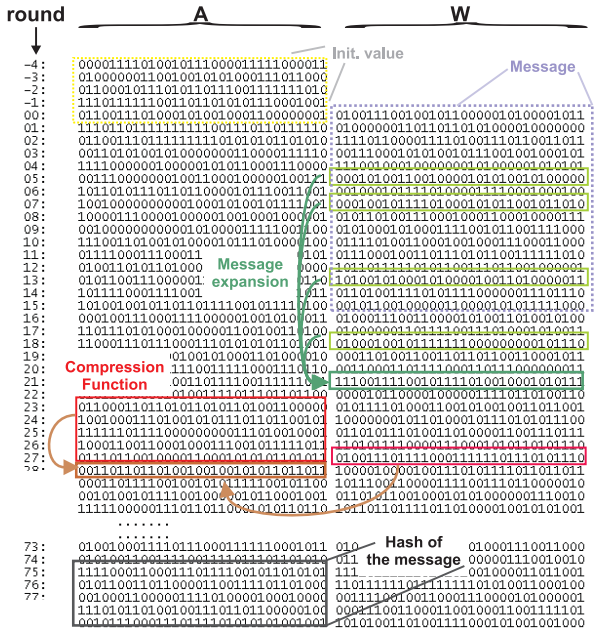

Fig. 1. An example of SHA execution

i	A_i	W_i
-4:	10110001110010000001011110n1111	
-3:	11110100110101011000011000 1un111	
-2:	10110110010111000101000 0unnnn001	
-1:	11000010011110101110111110 u11111	
00:	10100101001100111010100000u1001	u1nn011001101001-0-010001u1n01nu
01:	nn1u000011101010-0-011u11u0nnu0u	10uu1000110-0----1-100-11010n010
02:	nnnu011001100110-u-11u-uu00u0uu1	00010011011-1-01---0--111n1001nn
03:	u00n10001011u11u1n-n0-u101u0uu01	nu0n1100011----------1111nun01n1
04:	00nuu1nn10un1uu010-0-uu0011n11uu	u0nn011101100---------01101n1011
05:	u01001nu1n10000uuu00110000u000u0	nnnu111001000011-------0-un000u1
06:	n011101n0uuu10n1110u10-1111nu100	un001111011110------000--n1010u0
07:	110001unnnnnnn0-0010000u0nn010u0	0111111010-001011111-0--01uu0101
08:	11101000110001 10u1un100u0101uu0n	nun110000010111--00--0-10n0100u0
09:	u100001000100010-10n101u1u101101	10n00000001-------100111-nn00101
10:	01101111000000----11u0n0001n000nu	unu1101 00-1-0------01----n00100u
11:	0011011110000------nu1101n0u0001	10n0101010-----------0101u1n0111
12:	u11101100111--------0nunnuuu11u	un1010110-----------0--1u00101n
13:	01111011000001-------01-001100u0	nnu0101111-0--1---------1nuu01n1
14:	1010010110-11-0---------1011uun0	n1n00001-0---------------01010un
15:	n0101-00111-01----------1un11nu	un1001-1--1-1-------------n00n0
16:	-1001001-------------------10001	0un011011----0---------11u00110u
17:	u00-----------------------0-1011	nu10101-0---------------01nu00n0
18:	--1-----------------------001--	00n10-1-10-0--------------00110u
19:	----------------------------0---	nn101010-------------0-100110n1
20:	----------------------------1-u-	unn100-0---------------0n1101u1
21:	----------------------------u-	1nu1-0-10-0-------------n0111n0
...	...	...
63:	----------------------------	----------------------10-u--x
64:	----------------------------u---	----------------------n0----n-x
65:	----------------------------n--	----------------------u---x--u
66:	----------------------------	----------------------10-u-xx-
67:	----------------------------u----	----------------------n0----n-xx
68:	----------------------------n---	----------------------u---xn-nx
69:	----------------------------n---	----------------------u0-n-xx-u
70:	----------------------------n-----	----------------------u1----nxxx-
71:	----------------------------n----	

Fig. 2. A differential characteristic.

because of the Message Expansion from the first 16 words, all the constraints on the $\mathcal{W}$ part are either certainly verified or are impossible. The definition of a suitable characteristic is of course essential to the success of the attack. A characteristic leads to a collision in one block when it imposes a zero difference on the last five words A_i. However, this implies, in turn, some constraints on the previous rounds that may be incompatible with the expansion and compression functions.

The basic idea in the definition of a differential characteristic is the combination of *local collisions* [5]. These are sets of differences that are strategically placed in the characteristic, in order to introduce discrepancies between the two messages, while affecting as little as possible the state registers. Details are out of the scope of this paper, and can be found in the original article. Local collisions can be linearly combined in a so-called *linear characteristic*. Unfortunately, characteristics obtained in this way have an extremely low probability to lead to collisions, and, in fact, do not allow feasible attacks. A first important improvement can be achieved by considering two-block messages, where the output of the first block is used as input for the second one. In this way, we do not need to find a single characteristic that leads to a collision, but two that cause small opposite differences. In practice, this degree of freedom makes an attack on two blocks (1024 bits) much more likely to succeed, although more complicated to carry out. A further improvement can be achieved by using *non-linear* (NL) characteristics, obtained by imposing, at least on a part of the characteristic, conditions that are totally unconnected with the concept of local collision. Such conditions are especially set on words in the first rounds, the so-called *early rounds*. These in general match the rounds in which the original words of the message are used ($0 - 15$), possibly plus some more. In a typical scenario, thus, we have an overall differential characteristic made of a first NL part followed by a purely linear section. Independent of the type of characteristic, at each round j we can evaluate the probability that the subsequent A_j complies with the characteristic, given that the current word W_j and the previous five A_i do as much. This probability is called $P_u(j)$ [7]. Under the assumption of statistical independence between the conditions at each round, the overall probability of success can be roughly calculated as the product of all the probabilities $P_u(j)$ corresponding to each round. The size of the search space can be calculated by counting the degrees of freedom in the differential characteristic. Then, it is easy to evaluate the probability to actually find colliding message pairs in such space. Once a characteristic with a suitable value of probability is built, we can start the actual collision search. This proceeds by trying pairs of messages whose difference complies with the fixed characteristic. The two messages in each pair are hashed until an incompatibility with the differential characteristic is found, or until a collision is reached at round 80. It is essential to enumerate all available message pairs in the search space, to evaluate them exhaustively, possibly optimizing the search by pruning large subspaces as soon as it is possible to determine that they do not contain collisions.

To summarize, the overall collision search process can be divided into two main phases:

- **Phase 1)** definition of a differential characteristic, ensuring a suitable search space and a certain value of probability for the success of the attack
- **Phase 2)** enumeration of message pairs compliant with the differential characteristic, until a collision is found.

The work presented in this paper aims to identify weaknesses and study new search techniques for the SHA family hash functions, based on the development of a HPC application used for collision search and data analysis. The overall structure of the application is based on the two-phase process just described. Notice that the first phase is semi-automated and is partially interactive in nature. In fact, the support tools we developed for the first phase, based on both existing and original techniques, are manually driven and their input parameters are gradually refined by the user until a reasonably good quality characteristic is found. Compared to the second phase, the first phase is also less critical in performance, meaning that an increase in the computing power does not necessarily yield an improved quality characteristic. The second phase, on the other hand, is highly critical in performance and can be automated by finding out efficient ways to enumerate the message pairs in the subset defined by the characteristic, and by testing them rapidly. The essential idea is thus to run the first phase and find out a reasonably good characteristic "off-line", prior to the parallel search run on the supercomputer.

IV. The Maricel Cluster

To accelerate the exploration of the search space performed in the second phase, we have used the Maricel cluster hosted at the Barcelona Supercomputing Center, which is a heterogeneous multi-core cluster based on 72 QS22 blades. Each QS22 includes 2x Cell/B.E processors at @3.2Ghz and 8 GBytes of RAM. All the QS22 are inter–connected by a 4x DDR Infiniband network. The total number of cores is 1296 for a peak performance of 10 TFlops. In the rest of this Section the Cell/B.E. architecture and the CellMT library are described.

A. Cell/B.E. architecture

The Cell/B.E. is a heterogeneous processor composed of one main general purpose processor (PPU) and eight specialized cores (SPUs) with software-managed local stores. Each SPU only has direct access to its own local storage, so before any data computation may take place the involved data must be explicitly transfered from main memory to a local store. The data is transfered between local storages and main memory with asynchronous DMA operations managed by a specialized memory flow controller (MFC), thus the SPU is able to keep computing while up to 16 DMA operations are in-fly. The ability to issue asynchronous DMA operations between the local stores and the main memory enables an efficient overlapping of computation time and data transfer time, but at the cost of higher software complexity. The most widely used techniques to exploit this processor feature are the double buffering and the multi-threading schemes, that are discussed in Section IV-B.

Figure 3 shows the three basic components of the Cell processor. First, the PowerPC Processor Element (PPE), which is primarily intended to manage global OS resources. Second, the Synergistic Processing Elements (SPEs) that are specialized vector processors with a private local storage and a DMA

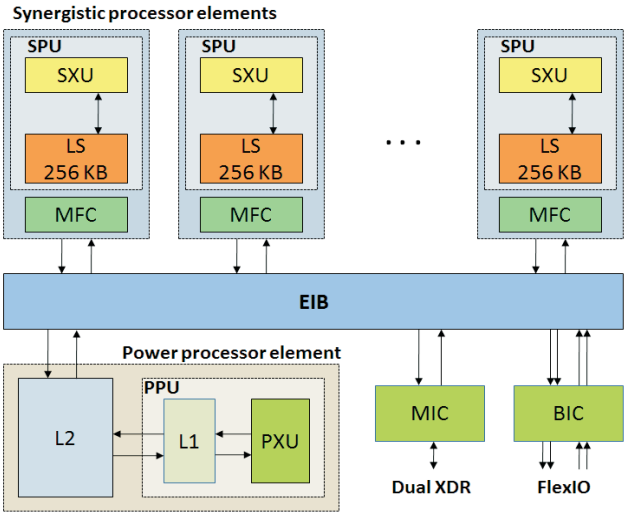

Fig. 3. The Cell Broadband Engine Architecture (CBEA).

engine, which can perform up to 16 asynchronous DMA transfers between the local storage and main memory at the same time. Finally, the communication between the PPE, the SPEs, main memory, and external devices is realized through an Element Interconnect Bus (EIB). The EIB has a theoretical peak data bandwidth of 204.8GB/s, but the DMA operations with main memory are limited to 25.6GB/s.

B. CellMT micro–threading library

The CellMT library provides a micro-threaded environment for the SPUs of the Cell/B.E. processor, which enables the concurrent execution of several threads inside each SPU. The use of multiple threads in the same SPU naturally overlaps the computation time of one thread with the data transfer time of the other threads, without increasing the code complexity. Previous evaluations of the CellMT library with synthetic benchmarks [1] and real applications [8] show that the multi-threaded approach can outperform a hand-coded double buffering scheme, with speedups from 0.96x to 3.2x, while maintaining the complexity of a naïve buffering scheme.

The cooperative multi-threading library is implemented on a core library that provides all the features and flexibility required to run complex multi-threaded applications inside the SPUs. This core library provides a low-level threading API that can be directly called from the SPU application code, although most applications only need to change the blocking DMA wait calls, to the waiting function provided by the library. The *wait_for_dma* function provided by the CellMT library saves the current thread state and relinquishes the control of the processor to the next ready thread following a round–robin scheduling policy. The cost of the context switch is below 200 cycles, despite the number of ready or blocked threads.

V. Parallel Cell/B.E.-based Application

The construction of a differential characteristic suitable for the collision search process, required in Phase 1) (see

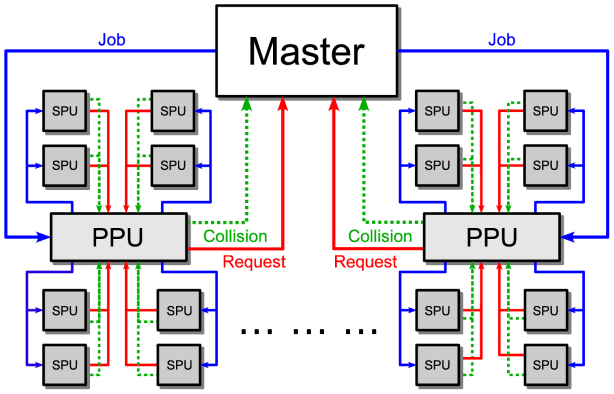

Fig. 4. Representation of the structure of the application.

Section III), is a composite task. It is structured into three main sub-phases, mostly heuristic in nature, conceived and descripted for the first time in [7]. We start with a basic differential characteristic, such as the one Figure 2, but completely unconstrained in its first part (corresponding to the early rounds), and then apply a series of specializations to add conditions and limit the search space. Every sub-phase basically introduces constraints, each with a different criterion. While the first phase operates pseudo-randomly, the following ones put increasingly more efforts towards the construction of a good characteristic, having relatively high probability of being satisfied. Although each of the sub-phases can be automated, so that they can be executed in a sequence, manual adjustments and tests on the quality of the intermediate results have proven to yield better results. The described procedure allowed us to find good quality characteristics, in a relatively small amount of time compared to the effort required by the Phase 2). In addition to the elements mentioned before, this suggests that a parallelization of this phase would be of no practical impact. The search for a message pair satisfying the characteristic's constraints, on the other hand, has an entirely different nature. The problem is well defined and very regular: we need to test for conformance the set of messages allowed by the characteristic as fast as possible, in order to find a near collision. Moreover, there is no qualitative aspect to assess, as in the characteristic construction: either the message pair is conformant, or it is not and must be discarded. This motivated us to design a highly parallelized version of the search program, able to fully exploit the potential of the Cell/B.E. architecture.

A. Structure of the parallel application

The application we developed for Phase 2) of the collision search is based on MPI, and is organized in a hierarchy made of three levels. At the top, a master node analyzes the first few rounds (depending on the particular characteristic, in general less than five), in order to produce more constrained (or specified) characteristics, which are then sent to the slave nodes. Both the master and the slaves are Cell/B.E. processors. On each of the slaves, the PPU receives the assigned job, consisting of a particular characteristic that includes a specific subset of the message pairs. The slave PPU, in turn, performs a similar decomposition, by specifying some of the indeterminate bits and thus creating a series of sub-jobs, that are different partial specifications of the input characteristic. Each of these sub-jobs is then delivered to one of its SPUs. Here, communication and processing are managed, respectively, by two threads of a multithreaded application based on the CellMT library introduced in Section IV-B. The SPU thread receives a mostly constrained characteristic, containing just a few freedom degrees (denoted as *free bits*), concentrated in the last of the early rounds. This keeps the thread busy in its processing for a time long enough to make sure that the communication overhead is negligible. In general, however, we do not desire to assign to the SPUs tasks that are too large. In fact, this would mean to leave more free bits in the upper portion of the characteristic, resulting in a more involved search process for the SPU and fewer opportunities for control simplifications and branch hints. In addition to the actual message pair, it is also convenient to deliver to the SPU the state of the registers for each of the two messages at the end of the last round calculated by the PPU. By doing so, the SPU does not need to recalculate the state again, while the additional information transmitted is relatively small and has little impact on the overall communication cost. Once the job is received, the SPU thread analyzes the message pairs allowed by the characteristic received. In case it finds out a collision, this information is notified to the PPU, and then to the master, in order to stop the multi-node application. Otherwise, the thread will simply ask for a new job.

The tasks accomplished by the SPUs basically consist in searching the space of message pairs conformant to $\mathcal{W}$, and testing whether a given message pair is also compliant to $\mathcal{A}$ (the register part), throughout all the rounds of the compression function. The search proceeds with a depth first tree approach. The original characteristic represents the root of the tree, and every node has two children, each associated to the same characteristic as the father, except for an indeterminate bit, which is specified differently in each of them. Unfortunately, the set of messages theoretically determined in this way is unmanageable, containing up to 2^{200} message pairs. Therefore, in the development of the application, a considerable effort has been spent for optimizing several aspects related to both the search algorithm and the underlying hardware architecture. In the following sections, we discuss the most important ones.

B. Algorithm Optimizations

By using a proper bit specification order, some effective pruning techniques can be adopted, shrinking the search space by several orders of magnitude. It has been observed experimentally that, once a message pair has diverged from the path, it is extremely unlikely[1] that, at a later round, it comes back to following it. Thus, if an inconsistency with the characteristic is detected at a certain round, it is practically unworthy to

[1]Except for some special cases, illustrated in Section V-E.

keep on calculating the registers values. If the specification of the bits is performed with ascending round number, there will be some levels in the tree for which an entire round will be completely determined, not containing any indeterminate bit. As soon as one of these levels (say the one completing round R) is reached, register A_{R+1} can be calculated; at this point we are able to decide if the portion of the message which has been completed is conformant to the characteristic or not. In the second case, we discard all the message pairs having those values in the first R rounds. This operation, in terms of the tree search, corresponds to the *pruning* of the entire subtree rooted in the node associated to the characteristic for which the inconsistency has been detected. The Master performs this task on the few first rounds, and each PPU does the same on almost all of the remaining early rounds. If it reaches a certain round (usually the 14th) without incompatibilities, it prepares the specific configuration of the characteristic and sends it to an SPU. Some degrees of freedom are left for a matter of load balancing between the components of the Cell/B.E.

An analogous technique is used in the search performed by the SPUs; however, note that when an inconsistency is detected at a late round (i.e. after the 16th) it is impossible to directly modify the message word involved, which are part of the expansion. Instead, the search needs to go back and modify the message where it is allowed, and recalculate the register values from there on.

Another important improvement to the algorithm is provided by the use of *Auxiliary Paths* (APs, see [9]). An AP is a set of bits of the message which, if flipped, produce another message pair having, after a transitory phase, the same values in the register until a certain round r_{AP}. Having several (say N) such groups is a great advantage, because once a message pair is tested to be conformant to the characteristic until round r_{AP}, it is possible to generate $2^N - 1$ new message pairs that will certainly be conformant, having the same registers state at round r_{AP}. Afterwards, they will behave as independent message pairs, having different message expansions. A smart way to apply all the AP configurations consists in using the *Gray code*. This allows flipping just one AP at each iteration, as well as avoiding the need to backup and restore the original message pair.

C. MicroThreading

The application makes use of the CellMT library, presented in Section IV-B. On each SPU, two threads are run and operate concurrently. Their purpose is to reproduce the effects of the double buffering, without having to actually manage the twofold increase of data structures. The job performed by a thread consists in a sequence of steps: the request for an authorization to read from the buffer by the PPU, the actual read, the elaboration, and the ACK to the PPU. The library enables I/O operations to be completely masked, as after a thread has started them it immediately yields the control to the other one. This improves substantially the efficiency of an SPU: as shown in Figure 7, around 97% of the time is spent for data processing.

D. SIMDization of the SPU code

The 128-bit datapath of the SPU's SIMD architecture, compared to the 32-bit parallelism inherent in the SHA-1 algorithm, offers an inviting opportunity to perform a vectorization of the search process, providing in principle four slots for independent SIMD processing. Unfortunately, the theoretical speed-up factor of four could be hardly reached in practice. This optimum limit could be reached only if one of the following conditions were true:

- if we could always pick four characteristics to be analyzed on the SPU that reach an inconsistency at the same round, to avoid idle slots. Unfortunately, knowing where the message pair will fail is exactly the objective of the search itself.
- if we could somehow build a data structure allowing us to discard the inconsistent message pairs and replace them dynamically in the SPU slots with others, independent of the round reached by each of them. Although conceptually valid, this solution does not pay off because it causes a very large overhead due to the management of misalignments between the different slots.

Therefore, the best solution is to program the SPU so that it performs the operations for four different message pairs, until a collision is found or *all of them* are discovered to be inconsistent, tolerating inactive slots during some of the three search phases (early, middle, late rounds). This is an acceptable trade-off for the problem: in exchange for a low overhead, we accept the fact that sometimes the effective parallelism is reduced. Nevertheless, experimental results have shown that, due to the behavior of the number of surviving message pairs (exponentially decreasing), the actual speed-up is in fact reasonably close to the theoretical value (see Section VI-A4).

E. New vulnerabilities in SHA-1

The analysis of the runs of the application enabled us to discover two new promising techniques for decreasing the complexity of an attack to the SHA family hash functions. Firstly, we removed one of the basic limitations inherent in the standard collision search approach presented in Section III, i.e. the bit-wise nature of the constraints imposed by the differential characteristic. For a few parts of the characteristic, in fact, we implemented some additional *inter-bit* constraints, taking into account the effects of the expansion function. Secondly, we found out that it is practically possible to relax some of the prescribed constraints, especially in the last, critical, rounds, effectively enlarging the search space and the density of solutions. By doing so, we were able to improve the probability $2^{3.47} \approx 11.08$ times, reducing by the same factor the expected time to find a conformant message pair[2]. Due to the lack of space, we are not able to discuss these new techniques thoroughly here. Further details will be provided in a future publication.

[2]These numbers refer to the characteristic used for the collision in Section VI-B.

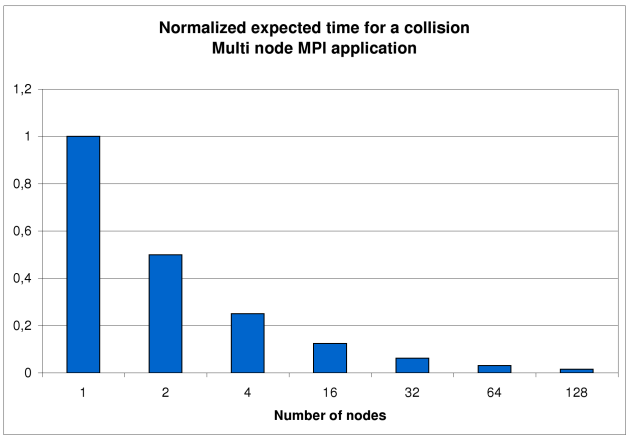

Fig. 5. Normalized expected time for a collision - multiple nodes

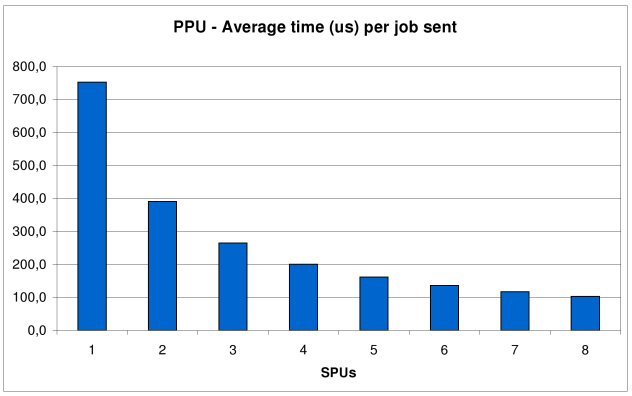

Fig. 6. Average time per job sent as seen from the PPU

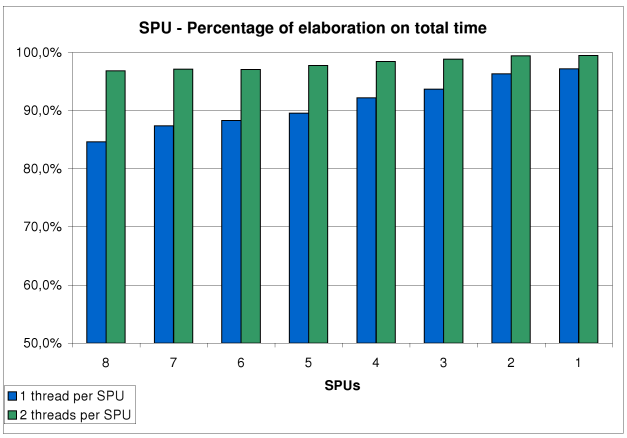

Fig. 7. Fraction of processing time over the total time

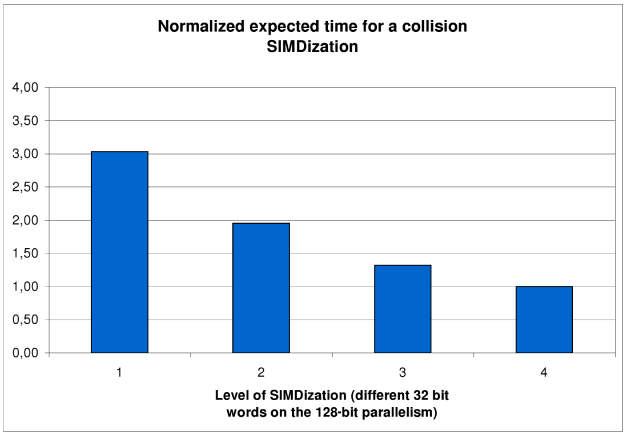

Fig. 8. Normalized expected time for a collision - SIMDization

VI. Quantitative evaluations and results

The optimizations discussed in the previous section have been accurately tested and shown to actually provide considerable improvements. In the following sections we analyze these effects in detail.

A. Performance evaluations

1) Algorithmic techniques: The pruning technique described in the previous section has a substantial impact on the performance, corresponding to several orders of magnitude. The Auxiliary Paths also have a significant role. For example, the two APs used for our evaluation yielded a 2.26 speed-up in the overall search time.

2) Scalability on multiple nodes: In Figure 5 we show how the average time to get a collision varies according to the number of nodes involved. We have verified that the speed-up is nearly proportional to the number of nodes. In fact, once the master has sent a job to all the slaves, the interactions among the elements are extremely rare –or even inexistent, if we send a single very large job at the beginning– except for the "collision found" message. This last message is propagated up in the hierarchy, whereas the subsequent command to stop the execution is sent in the opposite direction.

3) Scalability on a single Cell/B.E. processor: Figure 6 displays how the actual average time to complete a job[3] varies according to the number of SPUs involved in the process. The speed-up is almost linear. The slight overhead is caused by the size of the jobs, which is not completely optimized for communication, while simplifying (and accelerating) noticeably the elaboration on the SPUs.

Figure 7 shows the benefits on the SPU side deriving from the use of the CellMT library (Section IV-B). The presence of two threads allows the overlapping of the I/O phases to the elaboration phases, enabling SPUs to be processing data for more than 97% of the execution time.

4) SIMD Speed-up: Figure 8 shows how the average expected time to find a collision changes according to the level of SIMDization implemented. A 3.03 speed-up factor is achieved, considering a 4-way SIMD parallelization, which is overall an acceptable result. In fact, as discussed in Section V-D, the algorithm does not have a completely regular structure and has a significant portion involving control operations.

[3]Example jobs refer to the run that led to the collision presented in section VI-B.

B. A 71-round SHA-1 collision

Finally, based on the techniques implemented, including those introduced in Section V-E, we were able to show for the first time the feasibility of an attack against a reduced version of SHA-1 as large as 71 rounds (very close to the standard 80-round SHA-1). The colliding messages and the resulting hash value are displayed in Figure 9. Interestingly, by incorporating all the techniques presented in this paper, including the new approaches that have been discovered during this work, we were able to complete the search process with a relatively small computational workload, equal to approximately 500 Cell/B.E. machine-hours per block, further confirming the effectiveness of the proposed techniques.

Message 1			Message 2		
Word	Block1 (Hex)	Block2 (Hex)	Word	Block1 (Hex)	Block2 (Hex)
0	A031284A	C66928E5	0	10312819	766928B6
1	8E0B07E7	B8D273A2	1	BE0B07EF	88D273AA
2	259E60AA	136947A4	2	259E60E9	136947E7
3	26865A7F	4C7277A5	3	F6865A0D	9C7277D7
4	3F7A9945	87640DAB	4	8F7A9955	37640DBB
5	1F3B9AF1	1E439843	5	EF3B9A93	EE439821
6	B79EC755	8F78C12A	6	779EC717	4F78C168
7	41CB1152	7EA5FA75	7	41CB1162	7EA5FA45
8	311807D8	582F0B12	8	D118079A	B82F0B50
9	EA18241F	80286705	9	CA18247F	A0286765
10	C126D406	BA240B89	10	2126D447	5A240BC8
11	AAF12C3D	8A955AE7	11	8AF12C6D	AA955AB7
12	7F82700C	AB5CB1CA	12	BF82704D	6B5CB18B
13	464034BF	2BC7E7B5	13	A64034CD	CBC7E7C7
14	EF55783A	4132CCAA	14	4F557839	E132CCA9
15	B805685B	A57DD240	15	78056849	657DD252
Hash	89EE5B219C39AAB795FEED4483361F39D9B52E69				

Fig. 9. The two messages producing a 71-round collision.

VII. Conclusions

This work presented a study of the vulnerabilities in the SHA family, a hot topic today in security research, due to the recent breaks of cryptographic hash functions MD5 and SHA-1. For the performance-critical phase of the analysis, we relied on a high-performance computing facility, namely the MariCel cluster available at the Barcelona Supercomputing Center, based on the Cell Broadband Engine architecture. The effectiveness of the different optimizations and search strategies was validated by a comprehensive set of quantitative evaluations presented in the last part of the paper, and –most importantly– by an actual collision identified for a 71-round version of SHA-1, the first ever found so far. Moreover, as a result of the techniques developed, we were able to find this collision at a relatively small computational cost, roughly 500 Cell/B.E. machine-hours per block. These results provide new insights into the security robustness of the SHA-1 function and may influence the definition of future standardization initiatives. The findings presented in this work also make a case for the essential role that high-performance computing may have in the study of important open-issues in cryptography research.

Acknowledgment

This work was performed under the HPC-EUROPA2 project (project number: 228398) with the support of the European Commission - Capacities Area - Research Infrastructures. The work was also supported by the GARR consortium, under its "Orio Carlini" grant programme, and by the University of Naples Federico II, under its internal staff mobility programme.

References

[1] V. Beltran, D. Carrera, J. Torres, and E. Ayguadé, "CellMT: A Cooperative Multithreading Library for the Cell/B.E." in *HiPC '09: Proceedings of the 16th annual IEEE International Conference on High Performance Computing, Cochin, India*, 2009.

[2] E. Biham and R. Chen, "Near-Collisions of SHA-0", *Advances in Cryptology*, proceedings of CRYPTO 2004, LNCS 3152, pp. 290305, Springer Verlag, 2004

[3] E. Biham et al., "Collisions of SHA-0 and Reduced SHA-1", R. Cramer (Ed.), proceedings of *EUROCRYPT 2005*, Lecture Notes in Computer Science 3494, pp. 3657, Springer-Verlag, 2005

[4] J. Black, M. Cochran, and T. Highland, "A Study of the MD5 Attacks: Insights and Improvements", in M. Robshaw, editor, Proceedings of *Fast Software Encryption 2006*, Graz, Austria, March 15-17, 2006, volume 4047 of LNCS, 2006

[5] F. Chabaud, A. Joux, "Differential Collisions in SHA-0", proceedings of *Advances in Cryptology, Proceedings of CRYPTO '98*, vol. 1462 of Lecture Notes in Computer Science, pp. 5671, Springer Verlag, 1999

[6] C. De Cannière, F. Mendel, and C. Rechberger, "Collisions for 70-step SHA-1: On the Full Cost of Collision Search", C. Adams, A. Miri, and M. Wiener (Eds.), *SAC 2007*, Springer-Verlag, LNCS 4876, pp. 5673, 2007

[7] C. De Cannière, C. Rechberger, "Finding SHA-1 Characteristics: General Results and Applications", *Advances in Cryptology - Asiacrypt 2006*

[8] R. Ferrer, M. González, F. Silla, X. Martorell, and E. Ayguadé, "Evaluation of memory performance on the cell be with the sarc programming model," in *MEDEA '08: Proceedings of the 9th workshop on MEmory performance*. New York, NY, USA: ACM, 2008, pp. 77–84.

[9] A. Joux and T. Peyrin, "Hash Functions and the (Amplified) Boomerang Attack" in *Advances in Cryptology - CRYPTO 2007*

[10] S. Manuel, T. Peyrin, "Collisions on SHA-0 in One Hour", *Fast Software Encryption*, 2008

[11] National Institute of Standards and Technologies, Secure Hash Standard, *Federal Information Processing Standards*, Publication FIPS-180-1, April 1995

[12] R. Rivest, "The MD5 Message-Digest Algorithm", *Network Working Group, Request for Comments: 1321*, April 1992

[13] SHA-1 collision search project, University of Graz, Austria

[14] A. Sotirov, M. Stevens, J. Appelbaum, A. Lenstra, D. Molnar, D.A. Osvik, B. de Weger, "MD5 considered harmful today: Creating a rogue CA certificate", 2009

[15] M. Stevens, "On collisions for MD5", MSc Thesis, Eindhoven University of Technology, June 2007, available from http://www.win.tue.nl/hashclash/

[16] M. Stevens, A. Lenstra and B. de Weger, "Chosen-prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities", in Moni Naor (eds), proceedings of *Advances in Cryptology - EUROCRYPT 2007*, vol. 4515 of Lecture Notes in Computer Science, pages 1-22, Springer Verlag, Berlin, 2007.

[17] M. Sugita, M. Kawazoe, and H. Imai, "Grobner Basis Based Cryptanalysis of SHA-1", *Cryptology ePrint Archive*, Report 2006/098, 2006.

[18] X. Wang, Y. L. Yin, and H, "Yu. Finding Collisions in the Full SHA-1", in V. Shoup, editor, in proceedings of *Advances in Cryptology - CRYPTO 2005*, vol. 3621 of Lecture Notes in Computer Science, pages 1736, Springer, 2005

[19] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions", in: Ronald Cramer (ed.), *Advances in Cryptology - EUROCRYPT 2005*, vol. 3494 of Lecture Notes in Computer Science, pages 19-35, Springer, 2005.

[20] X. Wang, A. Yao, and F. Yao, "Cryptanalysis of SHA-1", presented at the *Cryptographic Hash Workshop* hosted by NIST, October 2005