A sensible title

Alessandro Barenghi
Politecnico di Milano
Milano, Italy
barenghi@elet.polimi.it

Gerardo Pelosi*
Università degli Studi di Bergamo
Dalmine, Italy
gerardo.pelosi@unibg.it

Abstract

Keywords: Breaking fast

1 Introduction

The Data Encryption Standard (DES) [4] is one of the most popular encryption algorithms, standardized by NIST in 1977 and subsequently maintained as a FIPS security primitive up to 2005 [5], when it was retired, since it had been proved that the cypher could be broken via a brute force attack [3]. Even though DES was not anymore considered safe for government applications, comments had to be addressed holding that "the DES should be retained because it is widely used in the market" and "FIPS 46-3 and associated standards are used in the commercial world and serve important functions, including use by the entertainment industry for real-time broadcast security, to prevent unrestricted copying of files, and for the security of digital television signals" [5]. So, while not anymore relevant for high-security applications, DES continues to be used in many commercial applications by private organizations. The DES encryption primitive is still supported by most encryption suites, including OpenSSL [8].

DES was designed specifically for highly-optimizaed hardware implementations, leaving the software implementations lacking in speed. Thus, both known brute force attacks, Deep Crack [3] and COPACOBANA [2], rely on dedicated hardware designs, the first in the form of specialized chips, the second in the form of FPGA-based hardware. Specialized hardware is, however, costly, so that DES may still remain a viable solution for short-term secrets, when the potential attacker has only access to consumer hardware and non-specialized knowledge.

The goal of this paper is to explore the viability of brute force attacks to the DES cipher with consumer hardware. Given the amount of computation needed to mount such an attack, GPGPU boards appear as the most promising target hardware: not only these boards provide a very low cost/MIPS ratio (and one bound to drop further, given the nature of the GPU market), but they are readily available and easily programmed by anyone with general purpose programming skills. If a brute force attack can be mounted against DES with today's GPUs, even if the performances of modern dedicated hardware solutions such as COPACOBANA are not reached, the fast evolution of the GPU market will provide more and more computational power in the near future, making in the end the software solution more cost efficient than comparable hardware solutions.

2 Des Cipher

Digital Encryption Standard (DES) is a symmetric block cipher with 64-bit block size that uses a 56-bit key. As previously mentioned, it was chosen as U.S.A. federal standard by NIST in 1977, when a key space of 2^{56} items was considered to be a good choice to make unfeasible any brute-force attack. The 56-bit key of DES is used with an additional parity byte to bring its size up to 64 bits. DES design consists of two parts, the encryption/decryption algorithm and the key-scheduling algorithm. It is an iterated block cipher consisting of 16 rounds, each designed with a Feistel structure and composed by bit-shuffling (P-boxes), non-linear functions (S-boxes) and modular algebra linear transformations through exclusive-OR operations. The Feistel schema has the advantage that encryption and decryption operations are identical, thus requiring only a reversal of the key schedule.

The key schedule algorithm, after an initial permutation of the key bits (Permuted Choice 1, PC-1), discards the eight parity bits and divides the key into two 28-bit halves; each half is thereafter treated separately. In successive rounds, both halves are rotated left by one or two bits (depending on a predetermined table that specifies the rotations for each round), and then 48 *subkey* bits are selected through a second fixed permutation (Permuted Choice 2, PC-2) – 24 bits from the left half, and 24 from the right. A different set of key bits is used in each subkey (one for each round of the encryption/description algorithm) in such a way that each bit

*This work was carried out under partial financial support of the Italian MiUR (Project PRIN 2006 ID-2006099978) and in part by project FSE ID-413174.

is used in 14 out of the 16 subkeys: sbk_i, $i \in \{1, \ldots, 16\}$.

For encryption, after the 64-bit plaintext is passed through an initial predetermined permutation (IP), the output is divided into two 32-bit blocks (L_0, R_0, respectively) in order to serve as input of the first round. In the first round, both the block R_0 and the subkey sbk_1, are jointly evaluated by the (Feistel) F function that includes a block expansion operation to align its size to 48 bits, followed by a XOR operation between the subkey and the expanded block and eight substitutions through as mush S-boxes (each with 6 input and 4 outputs). Finally, to the resulting 32-bit value is applied a fixed permutation (P-box). The output from F function is XORed with L_0 to produce R_1, whilst R_0 is directly fed to the other input of the first round, L_1. These operations are iterated for 16 rounds, but after the last round the left and right halves are not swapped and the result is subject to a final permutation (PI) to generate the 64-bit ciphertext. As for decryption, the only difference from encryption lies in the reverse order of the subkeys computed through the key-scheduling algorithm.

The DES cipher was designed to intentionally slow down software implementations. Indeed, permutations of individual bits, or application of an arbitrary function (S-box) to six bits of one word in order to insert a four-bit result into another word, are inefficiently executed on a word-based general-purpose CPU. Eli Biham in [1], was the first to describe a software implementation of DES that exploits the intrinsic bit-level parallelism of the cipher. The basic idea lies in the application of the SIMD (Single Instruction Multiple Data) execution model at level of operations among the n-bit integers of a general purpose CPU. Operations among n-bit integers may be thought as executed by n virtual processors, each executing the same instruction in parallel but operating on different single bits of data. The implementation reported in [1] encodes the DES block values in a non-standard way in order to mimic the fast hardware implementation with a minimal gate counting and computes each gate by a single instruction. It operates on 64-bit CPU as a SIMD machine with 64 one-bit processors. The execution of permutation and expansion operations do not involve any instruction but only registry renaming. Instead, the substitution functions are translated in a sequence of logical operations that trace out the functionality of gate networks used in the hardware implementation of S-boxes. Although, the S-boxes are implemented in more instructions than the ones needed for the usual look-up implementation, the parallelism of this solution achieves a considerable speed up (about $\times 5$), even considering the initial and final translation of the DES blocks in the non-standard representation used by this method.

3 Cuda architecture

copiato secco da quello di PDCAT

In recent times, Graphics Processing Units (GPUs) have been considered a potential source of computational power for non-graphical applications, due to the ongoing evolution of their programming interfaces and their appealing cost-performance figures of merit. Pioneering works attempted to adapt "general purpose" applications using graphic rendering APIs (OpenGL and DirectX) since they were the only way to tap into the GPU computational resources [9].

3.1 The NVIDIA GT200 Architectures

Modern GPUs now include hundreds of processing elements grouped in a hierarchical structure. In our case, the NVIDIA GT200 GPU series provides a set of independent multithreaded streaming multiprocessors.

Figure 1 shows an overview of the NVIDIA GT200 streaming processors array which is the part of the GPU architecture responsible for the general purpose computation. Each streaming multiprocessor is composed by a set of 8 streaming processors, two special functional units and a multithreaded instruction issue unit (respectively indicated as SP, SFU and MT-Issue in Figure 1). A SP is a fully pipelined single-issue core with two ALUs and a single floating point unit (FPU). SFUs are dedicated to the computation of transcendental functions and pixel/vertex manipulations. The MT-Issue unit is in charge of mapping active threads on the available SPs.

A multiprocessor is able to concurrently execute groups of 32 threads called *warps*. Since each thread in a warp has its own control flow, their execution paths may diverge due to the independent evaluation of conditional statements; when this happens, the warp serially executes each path. When the warp is executing a given path, all threads that have not taken that path are disabled. If the control flows converge again, the warp is able to return to a single, parallel execution of all threads. Each multiprocessor executes warps much like the *Single Instruction Multiple Data* (SIMD) paradigm, as every thread is assigned to a different SP and every active thread executes the same instruction on different data. The MT-Issue unit weaves threads into warps and schedules an active warp for execution, using a round-robin policy with aging.

Streaming multiprocessors are in turn grouped in Texture Processor Clusters (TPC). Each TPC includes three streaming multiprocessors in the GT200 architecture.

Finally, the NVIDIA GPU on-board memory hierarchy includes registers (private to each SP), on-chip memory and off-chip memory. The on-chip memory is private to each multiprocessor, and is split into a very small instruction cache, a read-only data cache, and 16 KB of addressable

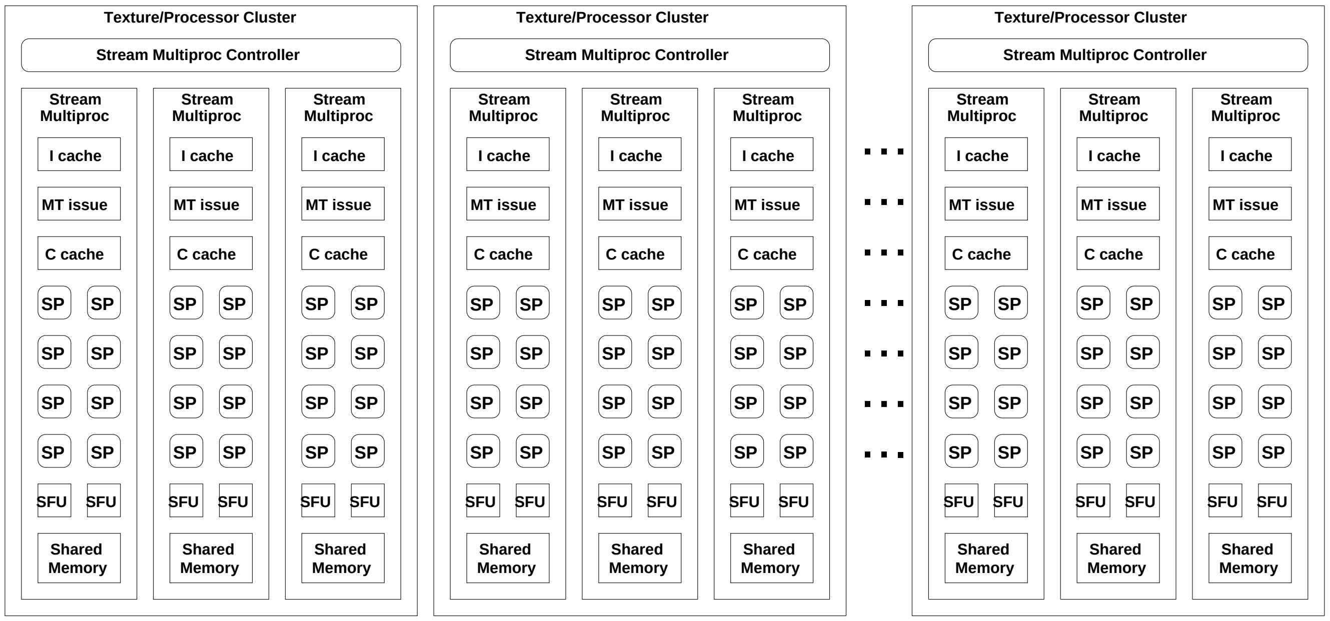

Figure 1. Sketch of the NVIDIA GT200 streaming processors array architecture: each Texture/Processor Cluster contains three stream multiprocessors. In turn, each stream multiprocessor is composed of eight streaming processor cores (SP), plus two special function units (SFU). Shared memory is local to each stream multiprocessor.

shared data, respectively indicated as I-cache, C-cache and Shared Memory in Figure 1. This shared memory is organized in 16 banks that can be concurrently accessed, each bank having a single read/write port.

3.2 CUDA Programming Model

The Compute Unified Device Architecture (CUDA) [6, 7], proposed by NVIDIA for its G80, G92 and GT200 graphics processors, exposes a programming model that integrates host and GPU code in the same C++ source files. The main programming structure supporting parallelism is an explicitly parallel function invocation (*kernel*) which is executed by a user-specified number of threads. Every CUDA kernel is explicitly invoked by host code and executed by the device, while the host-side code continues the execution asynchronously after instantiating the kernel. The programmer is provided with a specific synchronizing function call to wait for the completion of the active asynchronous kernel computation.

The CUDA programming model abstracts the actual parallelism implemented by the hardware architecture, providing the concepts of *block* and *thread* to express concurrency in algorithms. A block captures the notion of a group of concurrent threads. Blocks are required to execute independently, so that it has to be possible to execute them in any order (in parallel or in sequence). Therefore, the synchronization primitives semantically act only among threads belonging to the same block. Intra-block communications among threads use the *logical shared memory* associated with that block.

Since the architecture does not provide support for message-passing, threads belonging to different blocks must communicate through *global memory*. The global memory is entirely mapped to the off-chip memory. The concurrent accesses to logical shared memory by threads executing within the same block are supported through an explicit barrier synchronization primitive.

A kernel call-site must specify the number of blocks as well as the number of threads within each block when executing the kernel code. The current CUDA programming model imposes a capping of 512 threads per block.

The mapping of threads to processors and of blocks to multiprocessors is mainly handled by hardware controller components. Two or more blocks may share the same multiprocessor through mechanisms that allow fast context switching depending on the computational resources used by threads and on the constraints of the hardware architecture. The number of concurrent blocks managed by a single multiprocessor is currently limited to 8.

In addition to the logical shared memory and the global memory, in the CUDA programming model each thread may access a *constant* memory. An access to this read-only memory space is faster than one to global memory, provided that there is sufficient access locality since constant memory is implemented as a region of global memory fit with an on-chip cache. Finally, another portion of the off-chip memory may be allocated as a *local memory* that is used as thread private resource. Since the local memory access is slow, the shared memory also serves as an explicitly managed cache – though it is up to the programmer to warrant that the local data being saved in shared memory are not accessed by other threads. Shared memory comes in limited amounts (threads within each block typically share 16 KB of memory) hence, it is crucial for performance for each thread to handle only small chunks of data.

4 Implementation

Spiegazione di come stato implementato: se Fabrizio mi manda i dati, divisa in

4.1 classical

implementazione straightforward: pro usa un basso numero di registri, possibile tenere tutto nei registri della scheda, contro : la scheda non molto adatta a lavorare con operazioni bitwise

4.2 bitslice

pro : la scheda lavora molto bene con operazioni orientate al byte, la scheda ha registri nativi da 64 bit che aiutano ad ottenere un alta efficienza in un implementazione bitslice. contro : si usano maree di registri, il context switching mi aspetto soffra un po' e il tradeoff finir per essere con meno thread

5 Experimental results and cost

numeri e tabelle : suddividerei la sezione in :

5.1 breaking speeds on a single board

qui si buttano gi i numerilli pratici , cavati a massimo numero di blocchi lanciati (scenario migliore perch minimizza la latenza di chiamare molti kernel) e facendo vedere l' evenutale andamento del throughput al variare del numero di thread per blocco. Per quello liscio mi aspetto i soliti tradeoff points verso 128-256,(dovrebbero esserci addirittura ancora i numeri di Maurizio ,ma ho tirato ancora l' implementazione). Per il bitsliced, who knows , credo ci saranno tradeoff con meno thread per block.

5.2 Cost evaluation of a practical home-made breaker

Qui potremmo proporre l' analisi che mi ero fatto io per costruire un cluster in casa, facendo vedere il tradeoff migliore in termini di potenza di fuoco per ?. Fatto questo si pu tirar fuori un bel grafico di tempi proiettati al crescere dei soldini disponibili e far vedere quanto ci si mette a fronte di che spesa. Probabilmente interessante vedere anche quante macchine in pratica servono per sostenere il discorso di un mini sforzo distribuito tra un gruppo di gamer con interessi.

6 Conclusions

il solito summary di tutto stressando il fatto che ormai il DES si riesce a rompere (si spera) semplicemente mettendosi d' accordo con gli amici al bar.

References

[1] E. Biham. A fast new des implementation in software. In *FSE '97: Proceedings of the 4th International Workshop on Fast Software Encryption*, pages 260–272, London, UK, 1997. Springer-Verlag.

[2] T. Güneysu, T. Kasper, M. Novotný, C. Paar, and A. Rupp. Cryptanalysis with COPACOBANA. *IEEE Trans. Comput.*, 57(11):1498–1513, 2008.

[3] M. Loukides and J. Gilmore, editors. *Cracking DES: Secrets of Encryption Research, Wiretap Politics and Chip Design*. Electronic Frontier Foundation – O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1998.

[4] National Institute of Standards and Technology (NIST). FIPS-46-3: Data Encryption Standard (DES). http://www.itl.nist.gov/fipspubs/, May 1999.

[5] National Institute of Standards and Technology (NIST). Announcing Approval of the Withdrawal of Federal Information Processing Standard (FIPS) 46-3. *Federal Register*, 70(96):28907–28908, May 2005.

[6] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *ACM Queue*, 6(2):40–53, Mar. 2008.

[7] NVIDIA Corporation. CUDA Technology. http://www.nvidia.com/CUDA, Sept. 2008.

[8] OpenSSL Project. OpenSSL. http://www.openssl.org.

[9] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, 2007.