CZNIC Labs Technical Report number 1/2009

Signing of DNS zone using CUDA GPU

Matej Dioszegi
26th June 2009

Abstract

This paper describes the possibility of signing a DNS zone using CUDA (Common Unified Device Access) capable GPU. CUDA capable GPUs are available for more than two years. Their advantage is their relatively low price and high computing power based on parallelism.

As the process of signing a zone can be split up into independent pieces, it seems that parallelism provided by CUDA capable GPUs could potentially speed up whole process. On the other side, processors on the GPU are not as fast the CPU, memory access is slower and the GPUs are optimized to use floating point numbers, not integers, which are used in RSA and DSA.

Keywords: DNSSEC, zone signing, CUDA

Introduction

CUDA is technology introduced by nVidia company. They start to produce CUDA capable GPUs in 2007. Nowadays, over 100 million of these device have been sold worldwide. The newest product with name Tesla does not have any graphical output, it is used as a dedicated device for computing of complex problems providing 30 multiprocessors with 8 cores each. That means 240 parallel computations in parallel.

CUDA allows programmer to use GPU (called device. Computer which the device is present at, is called a host) as a standard computer. The programmer does not have to convert the problem into domain of computer graphics as it is common when using GPUs without CUDA capability. Device provides CUDA instruc-

tion set architecture, CUDA toolkit provides extension to C language. Programmer writes standard C functions that are compiled by special compiler and then executed on the device.

Main advantage of GPUs is high number of processors. Tesla contains 30 multiprocessor with 8 cores each. Our device nVidia GeForce 9800 GTX+ contains 16 multiprocessors with 8 cores each.

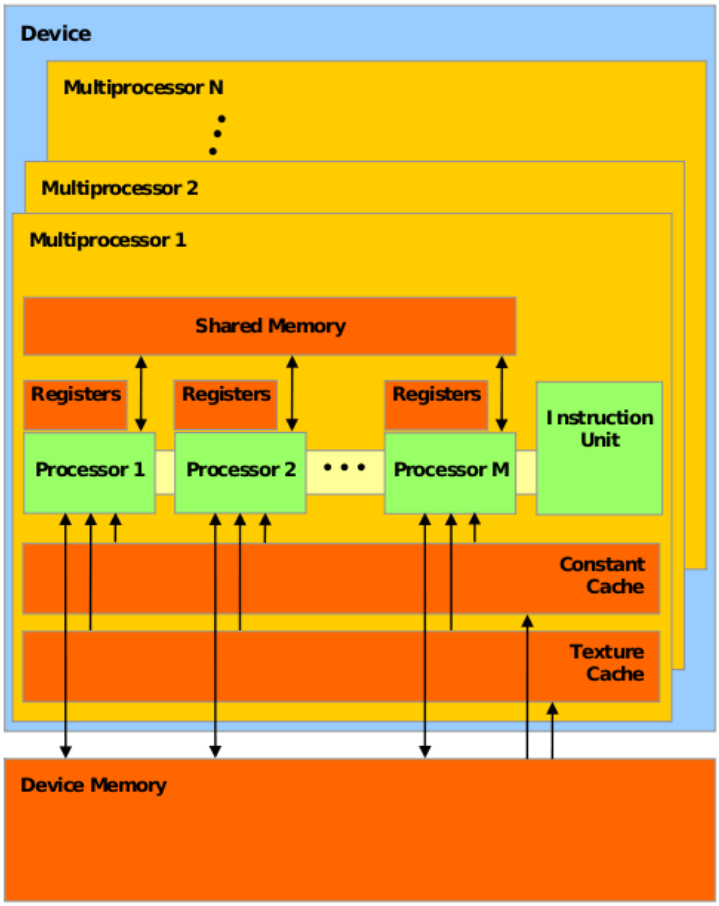

Illustration 1: Hardware model

Function, that is called from host and runs on the GPU device is called **kernel**. When a kernel is launched, programmer specifies its dimensions - in how many blocks (dimension of a grid) and in how many threads per block (dimension of block) the code is going to run. The blocks are then physically mapped on the multiprocessors and the threads in block are mapped to the cores. Threads on device cores are

extremely lightweight, their switching is faster than switching of CPU thread. Each thread has an unique identifier, which is computed from identifier of the block, which contains the thread and identifier of thread within the block. Each thread executes code of the kernel.

If the number of blocks that are specified by programmer is bigger than number of multiprocessors, the blocks “wait” until other finishes and then are run on the free multiprocessor. It is recommended to adjust dimensions of kernel to the parameters of used device.

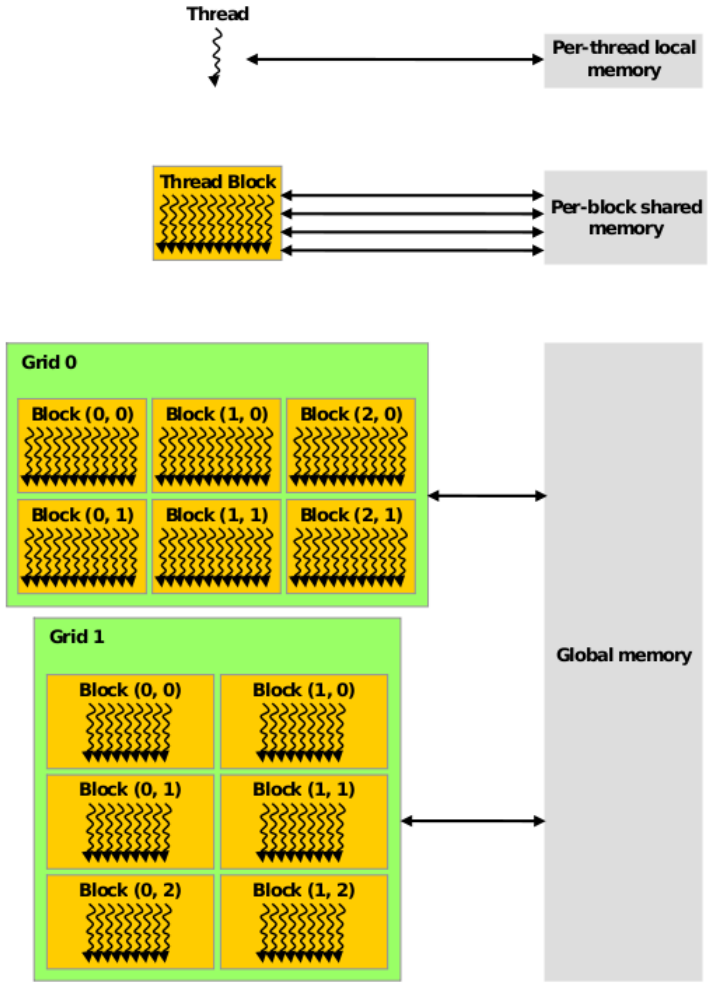

Illustration 2: Memory hierarchy

Device has global memory, which can be accessed by each core. There is also shared memory available for each multiprocessor. This shared memory acts as a software managed cache. Size of shared memory nowadays is 16K. Cores on one multiprocessor could cooperate through this shared memory. CUDA provides barrier synchronization primitive to support this cooperation. For each core there are private registers, where the core stores local data.

Access to the global device memory is not recommended as this access is much more slower than access to the shared memory on multiprocessor.

Motivation

Process of zone signing consist of more steps. RRsets have to be sorted into canonical order and each RRset has to be processed by some algorithm of asymmetric cryptography (RSA or DSA) – output of this algorithm is the digital signature of the RRset.

RRsets are independent on each other. They do not have to be signed in serial order. Here comes the idea of signing RRsets on the GPU, using CUDA. Each thread running on one of many GPU's cores would sign one RRset. As the number of cores on GPU is significantly higher than number of cores on CPU, whole process will finish faster.

Another advantage of using CUDA capable GPU is the price of the device. There exist devices capable of fast RSA signing, but their price is higher than price of a CUDA capable GPU. CUDA capable cards are also available world-wide. More than 100 million devices were sold during last two years.

It is easier for zone administrator to obtain a CUDA GPU and install it, than obtaining and installing RSA/DSA dedicated device.

Device requirements

Signing a DNS zone on the device brings some issues that have to be discussed.

A programmer cannot call methods from libraries present at the host in the device code. That implies programming of signature algorithms (RSA and DSA) by programmer alone.

Minimal requirements for those algorithms are:

- suitable representation of big numbers (multi-precision integers) in the device environment
- algorithm for fast adding of big numbers
- algorithm for fast multiplication and exponentiation of big numbers

There exist more algorithms that solve those mathematical problems (e.g. Montgomery multiplication and exponentiation).

RRset is not signed directly, however it's SHA1 hash is calculated. Calculation of this value could be executed on the device. That would imply implementation of SHA1 algorithm (or other SHA algorithms when the DNSSEC standards change) on the device.

Research and results

As a feasibility study, core of RSA signature algorithm was implemented on the device and compared with openSSL implementation.

Core of the RSA is exponentiation of multi-precision integers (called big numbers) modulo m. Computation of signature is:

$$signature = data^{pe} \ modulo \ m$$

where

- *data* is the message signed (hash of RRset in our case)
- *pe* is private exponent of the signing key.
- *m* is the public modulus of signing key

Modulus and exponent are big numbers. Their length is usually 1024, 2048 or 4096 bits. In this report, 2048 bit keys were used at implementation.

This exponentiation cannot be computed in a “standard” way because of the length of the numbers. This approach would consume a lot of time and also memory space.

For the exponentiation the Montgomery exponentiation algorithm described in Handbook of Applied Cryptography was used. Approximate time complexity of this algorithm is $O(n^3)$, where n is the length of signing key, but it does not need more than (n+1) bytes of memory for one computed number.

Code executing on the device has also some limitations.

- cores are less powerful than CPU cores and are optimized for floating point numbers calculation
- device is intended to use in application with high arithmetic frequency with minimum memory access
- memory access to the device global memory and shared memory on the multiprocessor differs significantly
- code executed on device should not contain many jumps as the cores does not have dynamic branch prediction, which results from the CUDA execution rules
- shared memory access should use specific rules stated in CUDA Programming reference to gain optimal performance

Unfortunately, big number arithmetic does not meet some of these requirement on performance. At first, it needs many memory accesses. Big numbers are stored in arrays and each operation on them needs to run through the whole array. This makes for 2048 bit keys 256 shared memory reads per one adding operation.

As it was mentioned, access to device global memory is slow in comparison with shared memory. Therefore, all the computations have to be performed in shared memory. Shared memory on a CUDA multiprocessor has nowadays only 16K. When calculating with 2048 bit keys, one number takes 256 bytes (plus 1 byte required for Montgomery multiplication) of memory for its representation. It means that shared memory can hold only 63 big numbers.

It must be considered, that for the Montgomery exponentiation we need place for 7 big numbers – data that are signed, exponent, modulus, place for values $r2 = R^2 \bmod m$, x' and big number representing 1 required by Montgomery exponentiation algorithm and one temporal working space.

This is the factor that mostly influences number of parallel RSA computations on one multiprocessor. There is place for 63 numbers (in 2048bit key case) and each computation (thread) needs 7 numbers. Val-

ues of *exponent*, *modulus*, "*big 1*", *r2* can be shared among the threads, but the place for *data*, *x'* and *temporal* must be allocated exclusively for each thread. Therefore number of parallel RSA computations on one multiprocessor is (63-4) div 3 = 19 computations for 2048 bit key.

Latest devices (Tesla) provides 30 multiprocessors, allowing the output of nearly 600 RSA signatures in parallel in a time needed by one device thread to compute RSA signature.

Crucial information now becomes the time for one RSA signature calculation. Here come in play other limitations of the device – speed of cores and shared memory access rules.

Experimental implementation needs about 94 seconds to compute one 2048 bit key signature, which is about 32x slower that the same code executed on the CPU. OpenSSL provides even better results, test on the same CPU shows that it can compute about 240 signatures per second.

Taking in consideration that Tesla has 30 multiprocessors, it can calculate 19x30 = 570 RSA (2048) signatures in 94 seconds, which is about 6 signatures per second.

DSA algorithm was not implemented, however it is unexpected that it will perform better in the device environment.

Conclusion

Signing of DNS zone parallel using CUDA GPU is not a bad idea. However, it has some limitations which result from the device architecture.

Biggest limitation is probably the background of the signing algorithm (RSA or DSA). Algorithms performed on the device should have high frequency of non-expensive arithmetic with minimal memory access, which is not achievable when using multi-precision integers needed by RSA and DSA.

Signing a DNS zone using current generation of CUDA GPUs will not bring any acceleration to the process of zone signing in comparison with openSSL library.

Signing of DNS zone using CUDA GPU