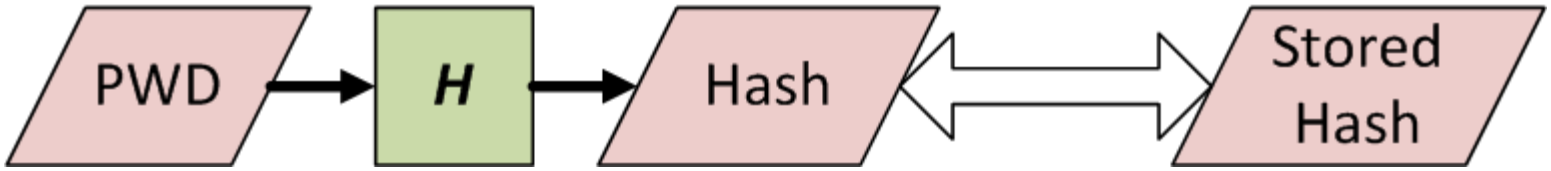
ELCOMSOFT
PROACTIVE SOFTWARE

Faster Password Recovery with modern GPUs

Andrey Belenko (a.belenko@elcomsoft.com)
ElcomSoft Co. Ltd.

Password Cracking: Basics

- System stores ***hash*** of user's password
- Authentication works by hashing user input and comparing result with stored hash



- Password Cracking works exactly the same
  - Try different passwords and eventually you'll find the correct one

Password Cracking: Trends

Changes in past five years:

- «Salting» used to avoid pre-computations
- (Much) Stronger cryptography is used
- Password cracking became much slower

What can be done?

- **Continue research**
- **Increase password recovery speed**

Password Recovery: How to do it **Faster**?

Software Optimization	• Free for End-Users • Limited speedup (10-20%) • Need to re-optimize for every new CPU
Special Hardware	• Expensive devices • Won't work with software from other vendors • Not very cost-effective
Common Hardware	• Hardware already installed in many computers • Cost-effective • Compatible with software from different vendors • Can be used for other applications

GPU: Two Worlds

	NVIDIA	ATI
Name	CUDA	CAL
First public beta	Feb 2007	Mar 2008
Minimum hardware	GeForce 8	HD 2X00
High-level language	C/C++ (NVCC)	C/C++ (Brook+)
Intermediate language	PTX	IL
Low-level language	No	For R600 only

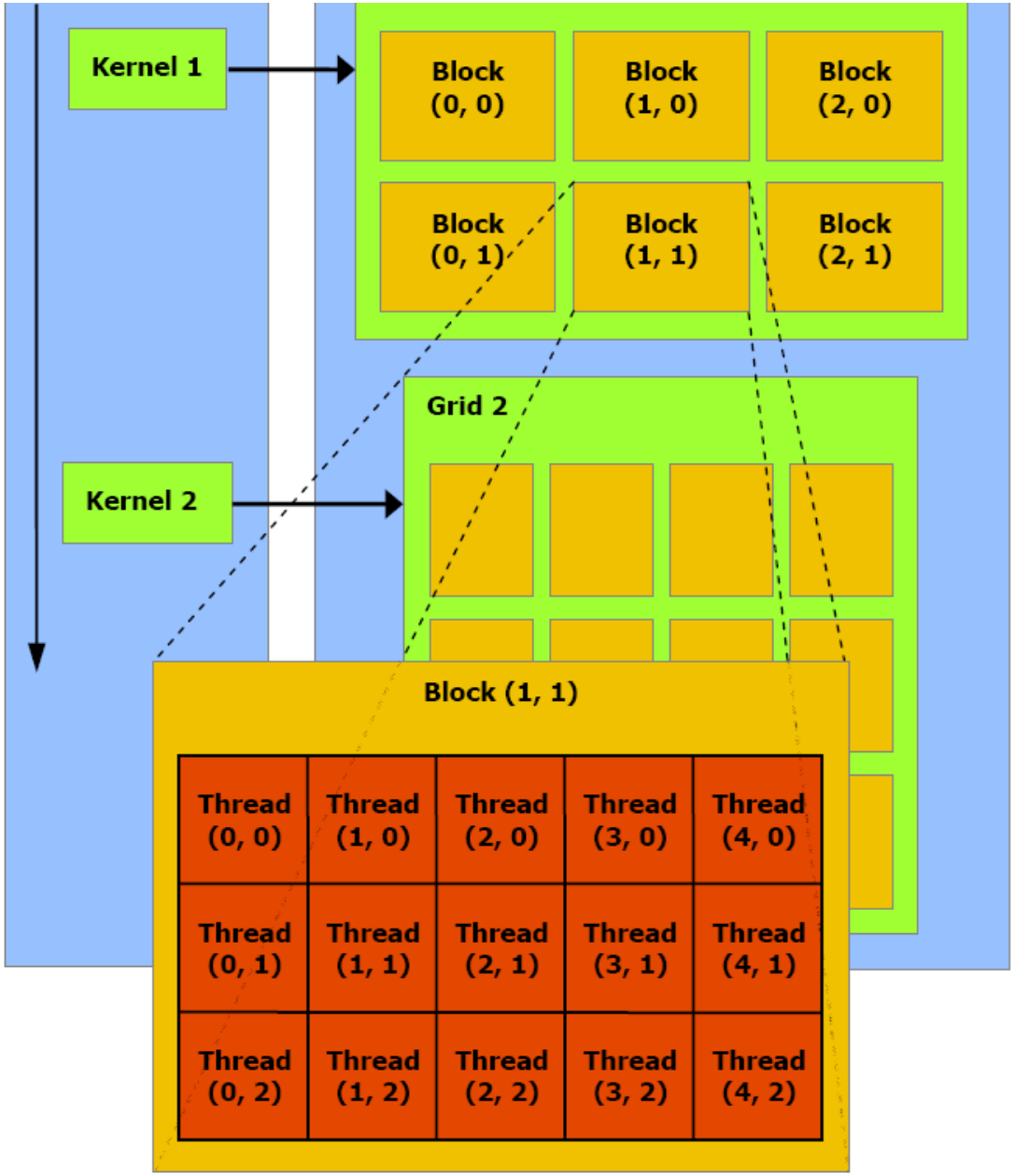
CUDA Basics

- GPU is a highly multithreaded data-parallel coprocessor
  - Up to 128 processors (16 multiprocessors)
- Fast on-board RAM
  - Up to 70 GiB/sec throughput
- Completely different programming model
  - Write your program from scratch rather than porting serial implementation

Task Partitioning

- Function compiled for GPU is called ***kernel***
- Kernel runs as a grid of thread blocks
  - Block may be **1D**, **2D**, or **3D**
  - Grid may be **1D** or **2D**
- Threads can communicate within block
- No grid-level communication or synchronization

- Up to 512 threads in one block
- Grid sizes up to 65535x65535
- Hardware allows up to 2^{41} threads

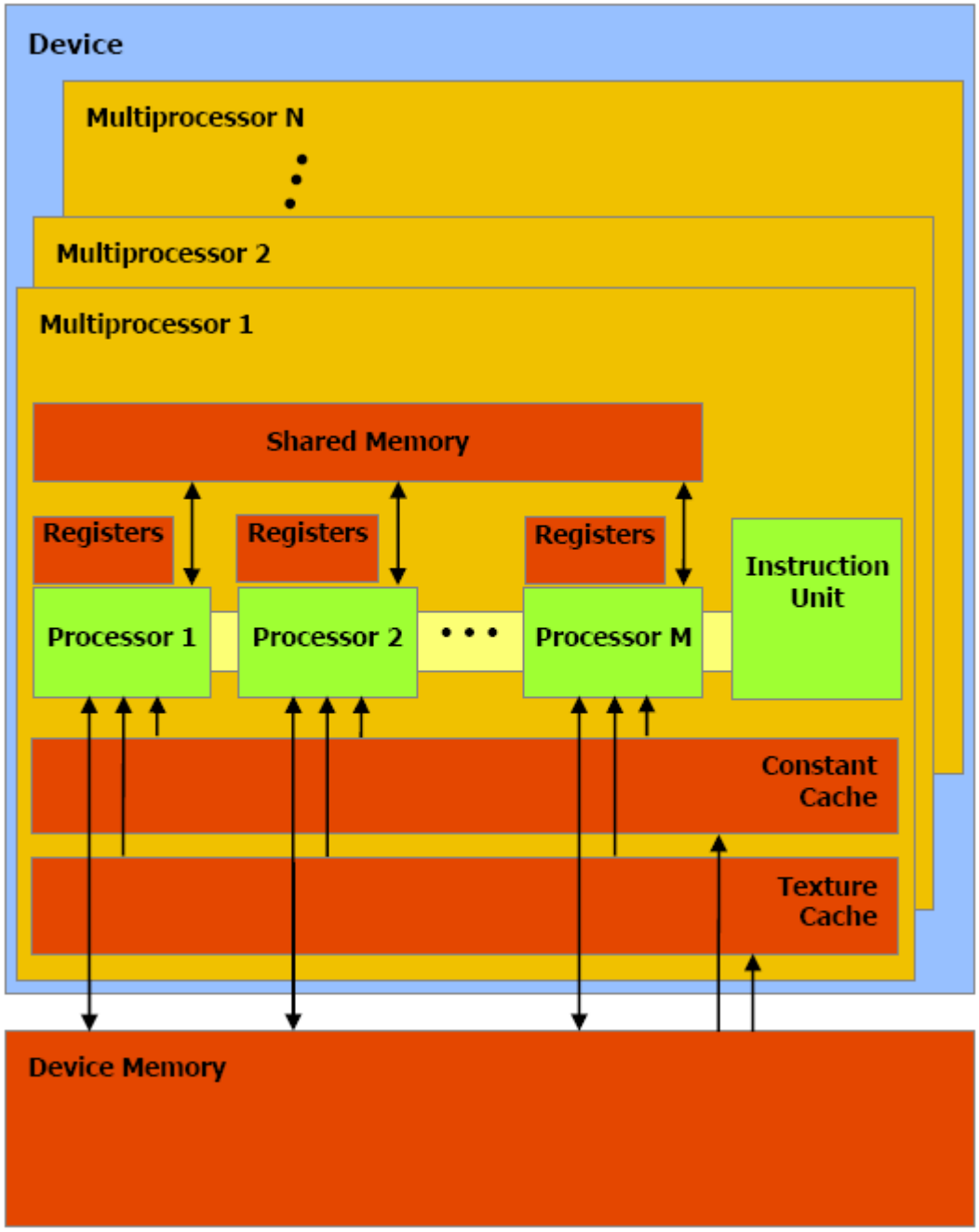


Memory Model

Multiprocessor-level			
Type	**Scope**	**Access**	**Speed**
Registers	Thread	R/W	Very Fast
Shared memory	Block	R/W	Very Fast

Device-level			
Type	**Scope**	**Access**	**Speed**
Global memory	Grid	R/W	Slow
Local memory	Thread	R/W	Slow
Constant memory	Grid	RO	Fast, cached
Texture memory	Grid	RO	Fast, cached

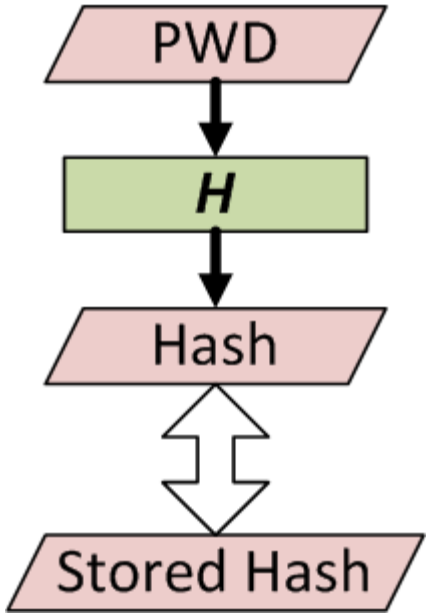
- 8192 32-bit registers per MP
- 16Kb shared memory per MP
- 64Kb constant memory per grid
- 8Kb constant cache per MP
- 8Kb texture cache per MP



Password Cracking on CPU

Very basic password cracker is simple:

```
while( 1 ) {
        password = get_next_password();
        hash = calculate_hash(password);
        if( is_correct(hash) )
        {
                print "Password found:" + password;
                break;
        }
}
```



Easy to parallelize!

Password Cracking on CPUs

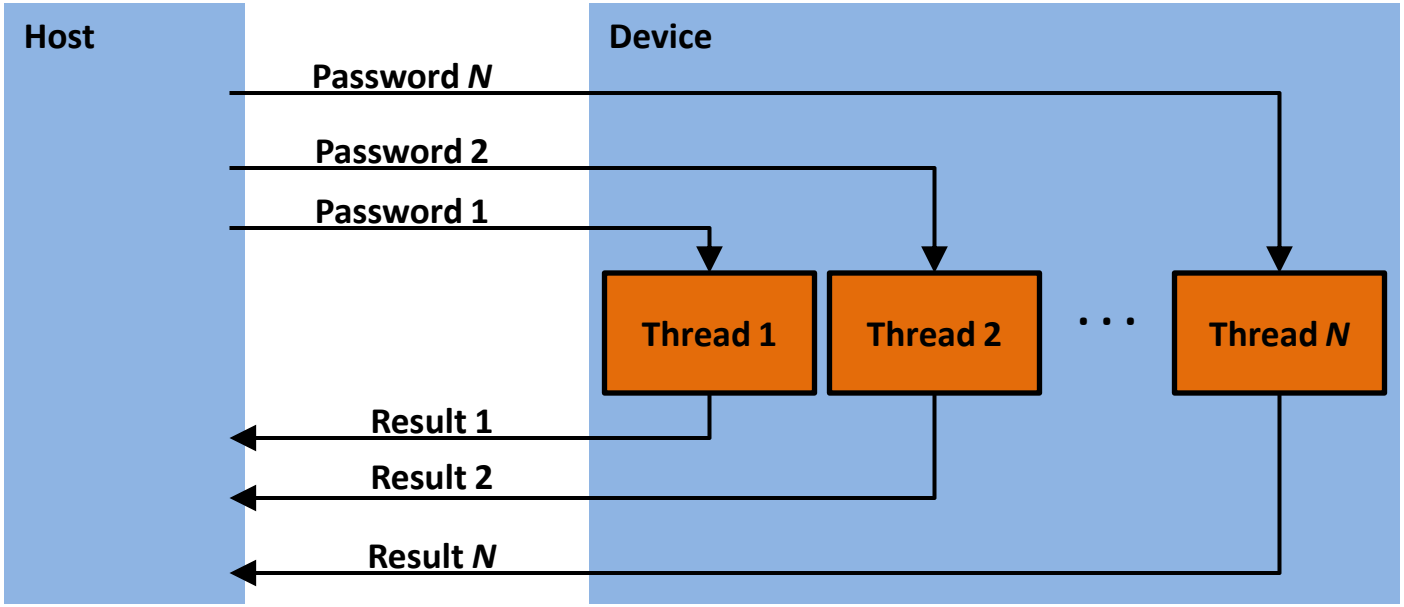
Very basic ***parallel*** password cracking thread:

```
while( not_found ) {
        mutex_lock();
        password = get_next_password();
        mutex_unlock();
        hash = calculate_hash(password);
        if( is_correct(hash) )
        {
                print "Password found:" + password;
                not_found = true;
                break;
        }
}
```

Spawn as many threads as CPUs/cores you have

Password Cracking on GPU

- Fits well to CUDA programming model
- ***N*** threads check ***N*** password in parallel
- No inter-thread communications needed



Tutorial: MD5 cracker on GPU

What *i*-th password cracking thread must do?

1. Generate (start+i)-th password
2. MD5(password)
3. If hash is correct, return *i* to the host

Notes:

1. Host doesn't need to do MD5 at all!
2. Dictionary requires more host to GPU transfers and thus less attractive. We'll do bruteforce.

Tutorial: MD5 cracker on GPU

GPU kernel environment:

```
__constant__ unsigned int nPassword[32];
__constant__ unsigned int nPasswordLen;
__constant__ unsigned int cCharset[256];
__constant__ unsigned int nCharsetLen;
__constant__ unsigned long bHash[4];

__device__ void MD5Transform(     unsigned long *s,
                                  unsigned long *d );

__global__ void MD5_Brute_GPU( unsigned long *pdwResult );
```

Tutorial: MD5 cracker on GPU

Getting password to process:

```
unsigned char Block[64] = { 0 };
unsigned int tid = blockIdx.x * 256 + threadIdx.x;
unsigned int t, r, i, q = tid;
for( i = 0; i < nPasswordLen; i++ ) {
    t = q + nPassword[i];
    q = t/nCharsetLen;
    r = t - q*nCharsetLen;
    Block[i] = cCharset[r];
}
// MD5 padding & length
Block[ i] = 0x80;
Block[56] = nPasswordLen * 8;
. . .
```

Tutorial: MD5 cracker on GPU

MD5 core (taken from RFC1321):

```
__device__ void MD5Transform (    unsigned long *state,
                                  unsigned long *x ) {
   unsigned long a = state[0], b = state[1];
   unsigned long c = state[2], d = state[3];
   /* Round 1 */
   FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
    . . .

   state[0] += a;
   state[1] += b;
   state[2] += c;
   state[3] += d;
}
```

Tutorial: MD5 cracker on GPU

MD5 transform in kernel:

```
. . .
MD5Transform( State, (unsigned long*) Block );
. . .
```

Hash compare:

```
        . . .
        if( State[0] == bHash[0] )
                if( State[1] == bHash[1] )
                        if( State[2] == bHash[2] )
                                if( State[3] == bHash[3] )
                                        *pdwResult = tid;
}
```

Tutorial: MD5 cracker on GPU

Calling kernel from host:

```
extern "C" void RunKernel_MD5(    int grid,
                                  unsigned long *pdwResult ) {
   MD5_Brute_GPU<<< grid, 256 >>>( pdwResult ); }
```

CUDA initialization:

```
int deviceCount = 0;
cudaError rc;
rc = cudaGetDeviceCount( &deviceCount );
if( rc != cudaSuccess ) {
       printf( " ! cudaGetDeviceCount() failed: %s\n",
                     cudaGetErrorString( rc ) );
       return 0;
}
```

Tutorial: MD5 cracker on GPU

Allocate GPU memory:

```
rc = cudaMalloc( &pdResult, 4 );
if( rc != cudaSuccess ) {
    printf( " ! cudaMalloc() failed: %s\n",
                cudaGetErrorString( rc ) );
    return 0;
}
```

Copy data to GPU memory:

```
rc = cudaMemcpy( pdResult, &nResult, 4,
                 cudaMemcpyHostToDevice );
if( rc != cudaSuccess ) {
    printf( " ! cudaMemcpy() failed: %s\n",
                cudaGetErrorString( rc ) );
    return 0;
}
```

Tutorial: MD5 cracker on GPU

Copy data to GPU constant memory:

```
rc = cudaMemcpyToSymbol( "nPassword", nPassword, 32*4 );
if( rc != cudaSuccess )
{
    printf( " ! cudaMemcpyToSymbol() failed: %s\n",
                cudaGetErrorString( rc ) );
    return 0;
}
```

Tutorial: MD5 cracker on GPU

Main loop:

```
while(1)
{
   RunKernel_MD5( 8192, (unsigned long*)pdResult );
   rc = cudaThreadSynchronize();
   if( rc != cudaSuccess ) {
       printf( " ! cudaThreadSynchronize() failed: %s\n",
                     cudaGetErrorString( rc ) );
       break;
   }

   cudaMemcpy( &nResult, pdResult, 4,cudaMemcpyDeviceToHost );
   // Increment password by 8192*256
   . . .
}
```

Tutorial: MD5 cracker on GPU

This unoptimized version runs at

115M p/s

on 8800 GTX

10x faster than dual-core Core2 @ 1.86 GHz

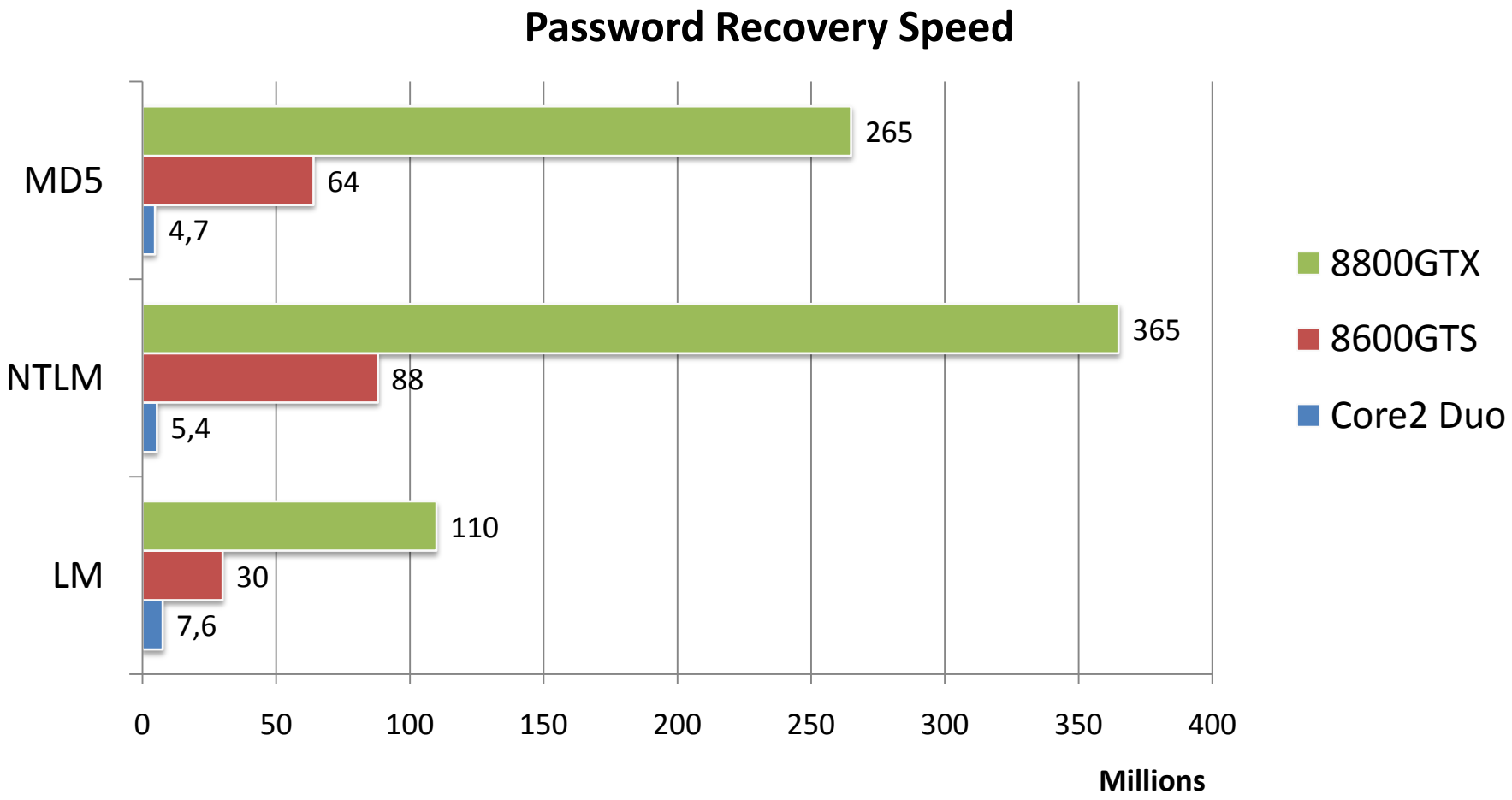
Tutorial: MD5 cracker on GPU

Possible optimizations:

- Don’t use integer division
  - Use multiplication instead
- Unroll loop
  - This will move `Block[]` from local memory to (much) faster registers

We’ve released optimized & free GPU MD5 cracker!

GPU: Performance



ELCOMSOFT
PROACTIVE SOFTWARE

Faster Password Recovery with modern GPUs

Andrey Belenko (a.belenko@elcomsoft.com)
ElcomSoft Co. Ltd.