BANK FOR INTERNATIONAL SETTLEMENTS

GPU accelerated computation of credit risk economic capital

Michal Beres, Technical University of Ostrava

Marek Hlavacek, Risk Analyst, Bank for International Settlements

Modern Tools for Financial Analysis and Modeling, Bratislava, June 2015

Presented results are outcome of a study performed in a collaboration with the Faculty of Electrical Engineering and Computer Science of the VSB - Technical University of Ostrava, and the National Supercomputing Center IT4Innovations.

The views expressed in this presentation are those of their authors and not necessarily the views of the BIS, the Technical University of Ostrava, or IT4Innovations.

Outline

- **Motivation (it is not only “need for speed”)**
- GPU accelerated computing
- Credit risk economic capital model
- GPU accelerated prototype
- Performance and accuracy gains
- How to start

Motivation – background

- The BIS is using a statistical model for the calculation of credit risk economic capital (CREC).
- In line with the high credibility of the BIS, CREC is defined as VaR at 99.995% confidence level of a hypothetical loss distribution.
- Given the nature of the BIS portfolio and the high confidence level, it is not possible to use an approximation in a form of a close-form formulae.
- CREC is thus calculated as the corresponding quantile of a simulated loss distribution (Monte Carlo).
- A large number of trials has to be evaluated to ensure acceptable convergence for such a high quantile (10 millions).

Motivation – little bit more background

- For the whole portfolio, one simulation with **10mn trials** executed on a 16-core server runs for **more than 3 hours**.
- This is not a limiting factor for the daily calculation of CREC. But it is a significant complication for any a bit more complex what-if or sensitivity analysis which requires evaluation of CREC using different parameterizations or portfolios.
- Even a simple what-if analysis (which does not require an execution of the simulation for the whole portfolio) takes several minutes.
- Marginal risk measures estimated from a simulation with 10mn trials appear to be rather unstable.

Motivation – adding a business value

- Performance
  - “Instant” response for a simple what-if analysis (exposure adjustment, rating change)
  - Very fast response for more complex scenarios
  - Possibility to evaluate multiple complex scenarios in reasonable time frame (a prerequisite for detailed sensitivity analysis)
- Accuracy
  - Significantly higher accuracy for the daily calculation of the CREC utilization (potential reduction of an economic capital buffer for model risk)

Outline

- Motivation (it is not only “need for speed”)
- **GPU accelerated computing**
- Credit risk economic capital model
- GPU accelerated prototype
- Performance and accuracy gains
- How to start

GPU accelerated computing – What is it?

"GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications. "

Source: Nvidia, http://www.nvidia.com/object/what-is-gpu-computing.html

GPU accelerated computing – the beginning

- Originally, graphic cards were designed to perform large number of very specific operations in parallel (vector rotations, transitions, …) on data with a predefined structure (vectors, textures…).
- Due to constantly increasing demand for a high-resolution and fast computer graphics, the computation power of GPUs evolved rapidly.
- Soon (late 1990s), pioneers of the **general-purpose computing on graphics processing units** discovered that it worth to convert their data into vectors and textures to speed up their optimizers, solvers etc.

GPU accelerated computing – modern age

- Modern graphics processing units (GPUs) come with flexible and very generic interface allowing to easily access the computation power they offer:
  - OpenCL (Generic)
  - CUDA (Nvidia)
- Benefits for developers and analysts:
  - GPU accelerated mathematical and statistical libraries for various programming languages (C/C++, Fortran, C#, …).
  - GPU accelerated functions available in computing environments (e.g. Matlab)
- Benefits for end users:
  - Commercial software with build-in GPU acceleration (not only computer games and 3D video streaming)

Outline

- Motivation (it is not only “need for speed”)
- GPU accelerated computing
- **Credit risk economic capital model**
- GPU accelerated prototype
- Performance and accuracy gains
- How to start

Credit risk economic capital model in a nutshell

1) Draw random shocks:
   - Multivariate standard normal distribution
   - Beta distribution (Gaussian copula)
2) Calculate loss at exposure level:
   - Loss due to default:

     L = [exposure at default] * [random LGD]
   - Loss due to rating migration:

     L = [value(original rating) – value(new rating)]
3) Aggregate losses at sub-portfolio level:
   - Can be implemented as a matrix multiplication
4) Maintain tails of loss distributions:
   - Sorting of tails corresponding to individual sub-portfolios

Outline

- Motivation (it is not only “need for speed”)
- GPU accelerated computing
- Credit risk economic capital model
- **GPU accelerated prototype**
- Performance and accuracy gains
- How to start

GPU accelerated prototype – “architecture”

- The GPU accelerated implementation is developed in Matlab and CUDA:
  - **Matlab code executed on CPU:**
    - data manipulation
    - data preprocessing
    - calculation of z-scores etc.
  - **Matlab code executed on GPU:**
    - aggregation of loss data (matrix multiplication),
    - sorting of tails
  - **CUDA kernel:**
    - sampling of random shocks
    - calculation of losses
- A benchmark implementation was developed in Matlab – runs in parallel on multiple CPU (parfor)

GPU accelerated prototype – few highlights

- Parallelization using **“one trial per thread” strategy** works for most Monte Carlo simulations given that:
  - execution time for individual trials does not significantly differ
  - data required to process a trial fit into the memory
- It is the large number of cores which gives the computation power - **latency needs to be managed:**
  - In a single kernel run, execute more threads than physical cores (performance increased by 50% when number of threads was doubled)
  - Smooth thread execution times by processing several trials in a single thread (performance increased by 30% when 10 trials were processed by a single thread)

GPU accelerated prototype – few highlights

- **Avoid conditional execution (if, while etc.):** It can be even faster to evaluate an expression instead of checking whether it is needed
- **Choose memory carefully** (Tesla K20):
  - *global* (shared, large but slow)
  - *constant* (shared, very fast but extremely small)
  - *local* (dedicated to a thread, small, can be faster than global memory, fast for data stored in registers)
- **Do not use *doubles* blindly:** In time constrained calculation floats may deliver better result than doubles (6.075728904530946 3 +/- 5)
- **Organize your data:** data from memory is loaded to thread's registers by strides. For this reason, all vectors and matrices should be organized in such a way that threads read consecutive elements

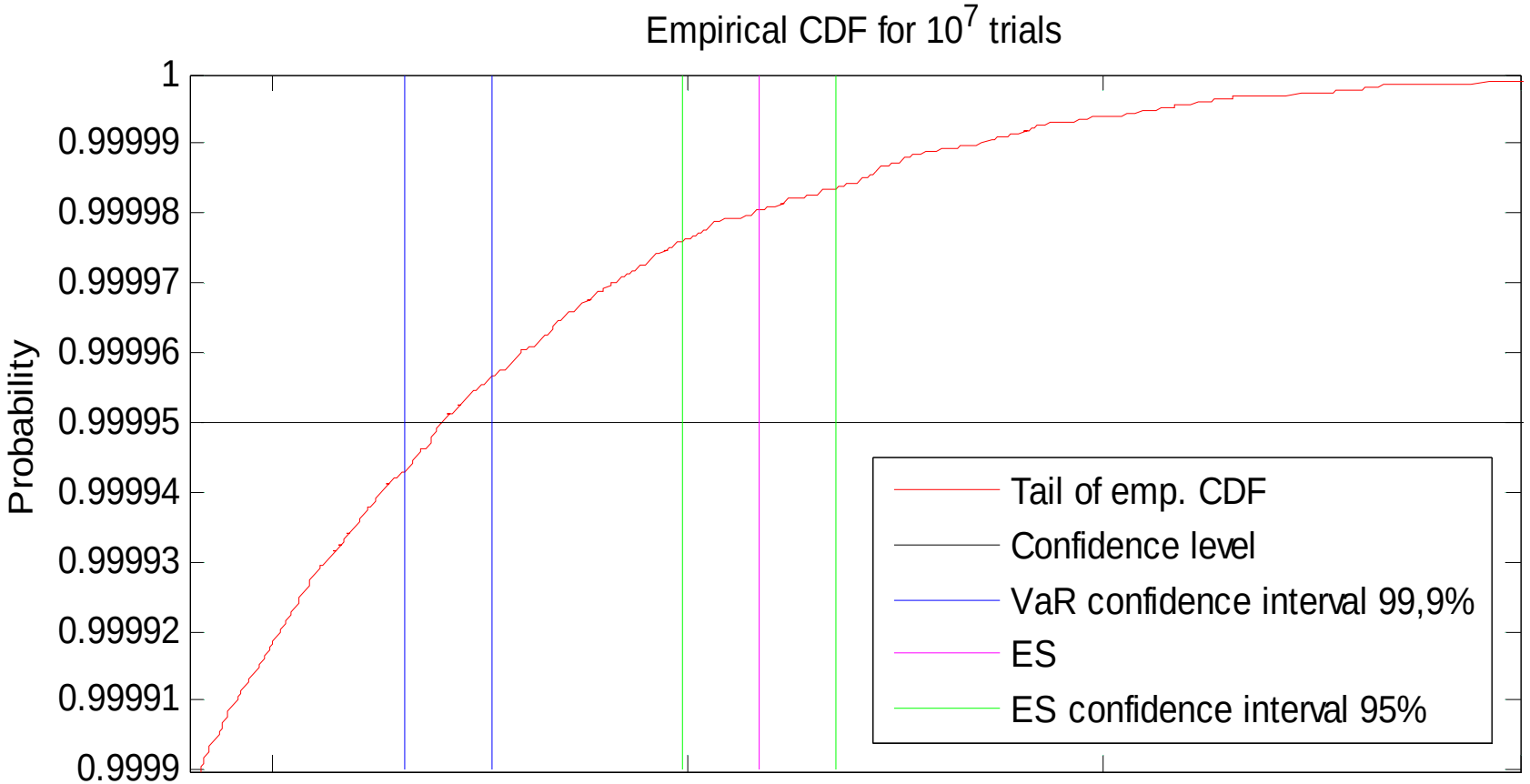
Outline

- Motivation (it is not only “need for speed”)
- GPU accelerated computing
- Credit risk economic capital model
- GPU accelerated prototype
- **Performance and accuracy gains**
- How to start

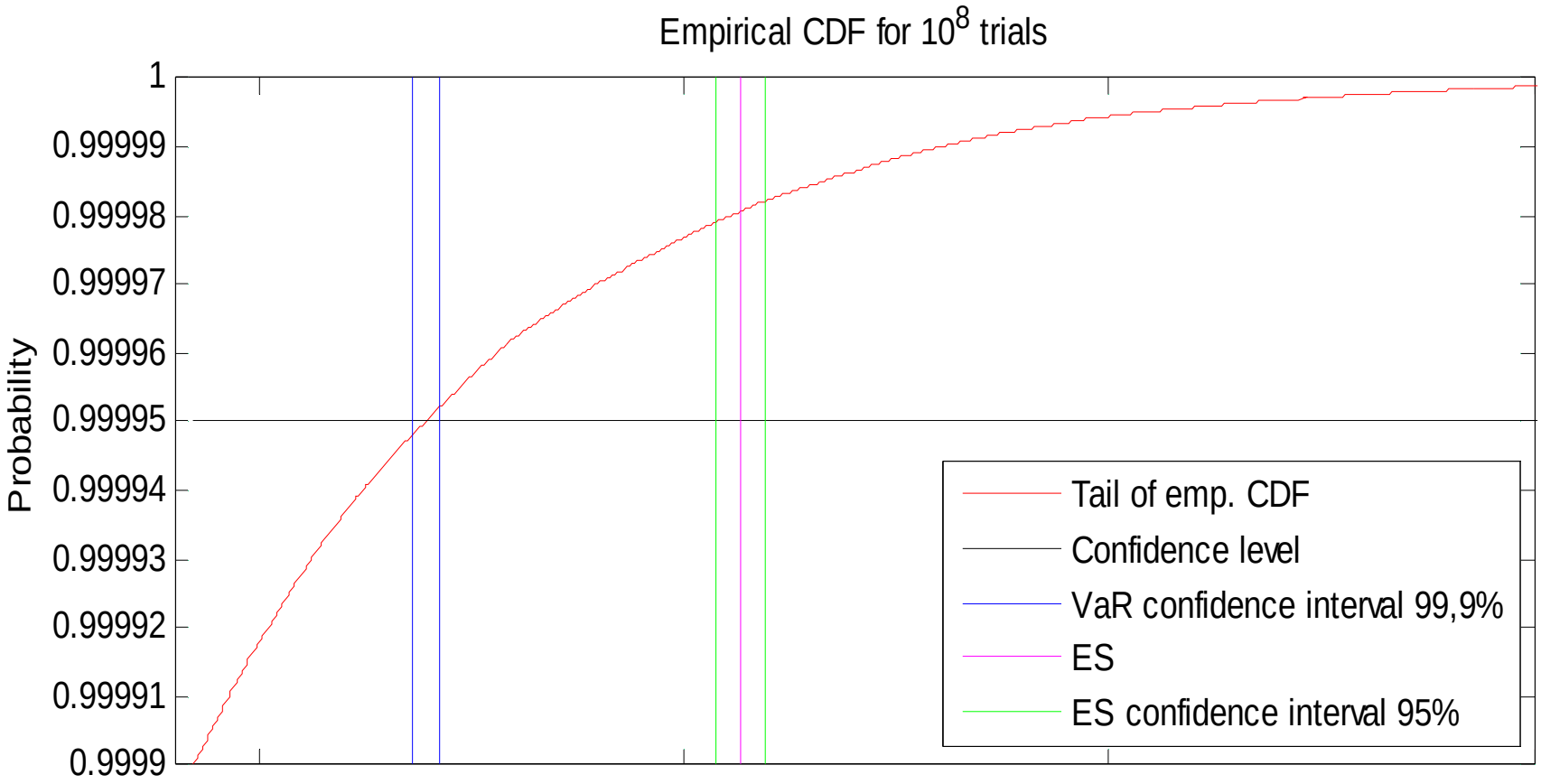
Performance and Accuracy gains - performance

Estimator	Platform	Computation time (10mn trials)	Computation time (500mn trials)
Third party application	Java	**3 hours, 20 minutes**	1 week
Optimized CPU version	Matlab	1 hour	2 days
GPU accelerated version	CUDA + Matlab	4 minutes	**3 hours, 30 minutes**

Performance and Accuracy gains - accuracy



Performance and Accuracy gains - accuracy



Outline

- Motivation (it is not only “need for speed”)
- GPU accelerated computing
- Credit risk economic capital model
- GPU accelerated prototype
- Performance and accuracy gains
- **How to start**

How to start – prerequisites

- **Hardware:**
  - modest gaming card, or
  - specialized computing accelerator

  Do not use (or be careful) when using your primary GPU for computing (timeout)!
- **Software**
  - OpenCL
  - CUDA
  - Matlab + Parallel computing toolbox
  - …

How to start – learn the basic concepts

- Matlab:
  - http://ch.mathworks.com/discovery/matlab-gpu.html
- Nvidia CUDA zone (CUDA Toolkit):
  - https://developer.nvidia.com/cuda-zone
  - https://developer.nvidia.com/cuda-toolkit
- OpenCL:
  - https://www.khronos.org/opencl

How to start – give it a try, it is not that complicated

- Function and data:
- Transferring data to GPU:
- Functions executable on GPU:
- Generic elementwise operations:

```
f = @(x,y)exp(sin(x)+cos(y));
A=rand(10^7,1); B=rand(10^7,1);

Agpu = gpuArray(A); Bgpu = gpuArray(B);
%Elapsed time is 0.028218 seconds.

Cgpu = f(Agpu,Bgpu); %executed on GPU
C=gather(Cgpu);
%Elapsed time is 0.027505 seconds.
C = f(A,B); %executed on CPU
%Elapsed time is 0.069975 seconds.

C=arrayfun(f,Agpu,B); %executed on GPU
%Elapsed time is 0.019498 seconds.
C = arrayfun(f,A,B); %executed on CPU
%Elapsed time is 35.007042 seconds.
```

How to start – give it a try, it is not that complicated

- CUDA kernel:

```
__global__ void fce( float* A, float* B, float* out, int size)
{
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    if(idx<size){
            out[idx]=expf(sinf(A[idx])+cosf(B[idx]));
    }
}
```

- Call in Matlab:

```
CUDA_kernel=parallel.gpu.CUDAKernel('fce.ptx','fce.cu','fce');
CUDA_kernel.ThreadBlockSize=[192*2 1 1];
CUDA_kernel.GridSize= [6 1];

Agpu=gpuArray.rand(N,1,'single');
Bgpu=gpuArray.rand(N,1,'single');
Cgpu=gpuArray.zeros(N,1,'single');

[A,B,Cgpu] = feval( CUDA_kernel,A,B,out,N);
C=gather(Cgpu);
```

This is code was not tested and may not compile.

Thank you for your attention!

michal.beres@vsb.cz, marek.hlavacek@bis.org