«Forest» – GPU-accelerated method for solving global optimization problems

Anton Anikin
ISDCT SB RAS, Irkutsk

Optimization without borders, 2021

The problem

$$f(x) \to \min_{x \in X \subset \mathbb{R}^n}$$

- $f(x)$ – is non-convex function

The problem

$$f(x) \to \min_{x \in X \subset \mathbb{R}^n}$$

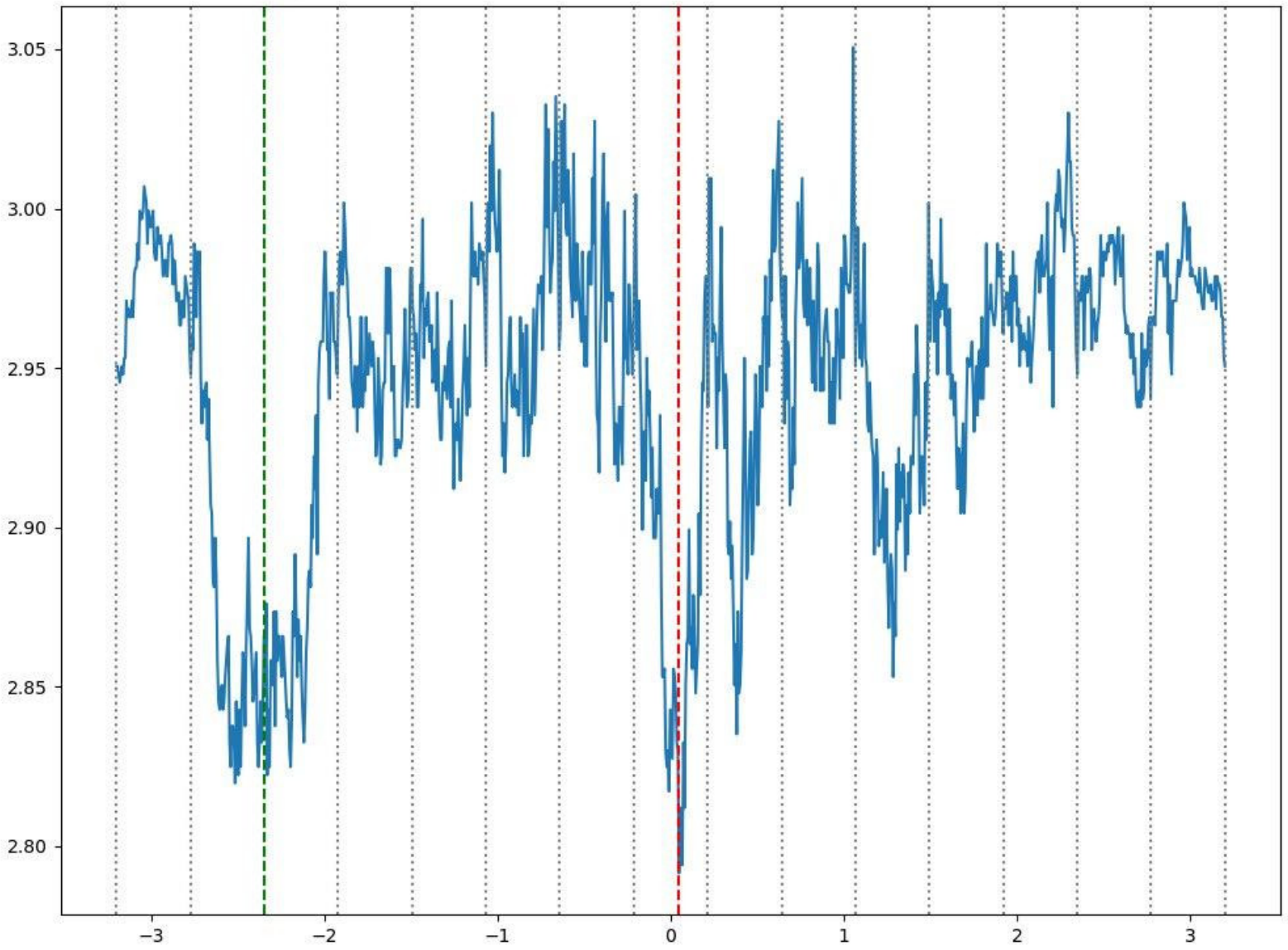
- $f(x)$ – is non-convex function
- $f(x)$ – is non-unimodal function

The problem

$$f(x) \to \min_{x \in X \subset \mathbb{R}^n}$$

- $f(x)$ – is non-convex function
- $f(x)$ – is non-unimodal function
- $f(x)$ – is **multiextreme** function

Real-wold example



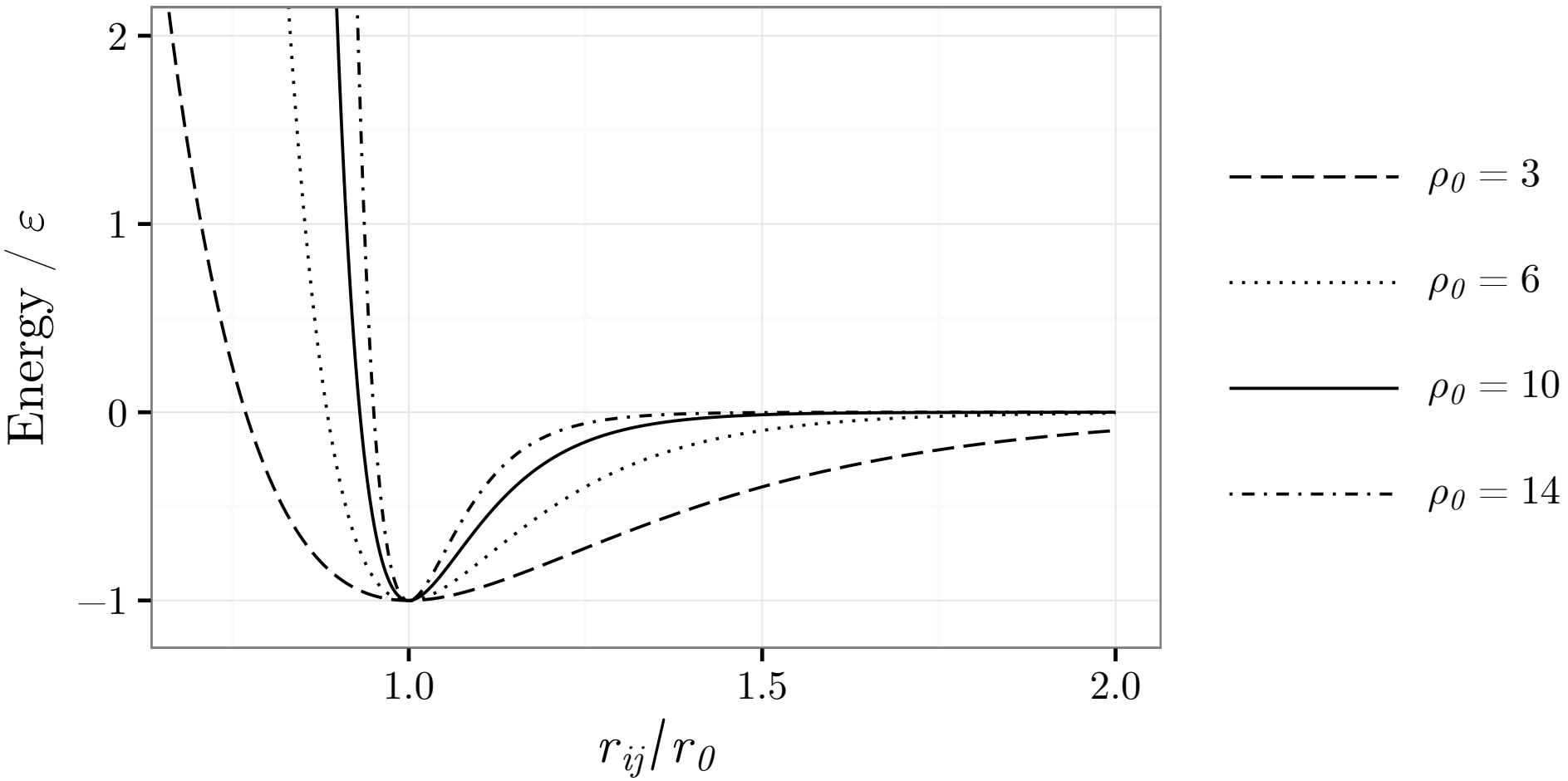
Morse potential

$$V(R) = \varepsilon \sum_{i<j} e^{\rho_0\left(1-\frac{r_{ij}}{r_0}\right)} \left(e^{\rho_0\left(1-\frac{r_{ij}}{r_0}\right)} - 2 \right),$$

$$r_{ij} = \|x_i - x_j\|_2.$$

$$x_i \in \mathbb{R}^3$$

Morse potential



Challenges

- Global optimization problem
- Huge number of extremes
- Modern state of the problem – “large clusters” with more than 200 atoms (over 600 optimizing variables)

The Cambridge Cluster Database

D. J. Wales, J. P. K. Doye, A. Dullweber, M. P. Hodges, F. Y. Naumkin
F. Calvo, J. Hernández-Rojas and T. F. Middleton

www-wales.ch.cam.ac.uk/CCD.html

Hefei National Laboratory for Physical Sciences at the Microscale and School of Life Sciences, University of Science and Technology of China

staff.ustc.edu.cn/~clj

Jorge Marques

Department of Chemistry Research in Computational Chemistry and Molecular Modeling University of Coimbra, Portugal

apps.uc.pt/mypage/faculty/qtmarque/en/clusters

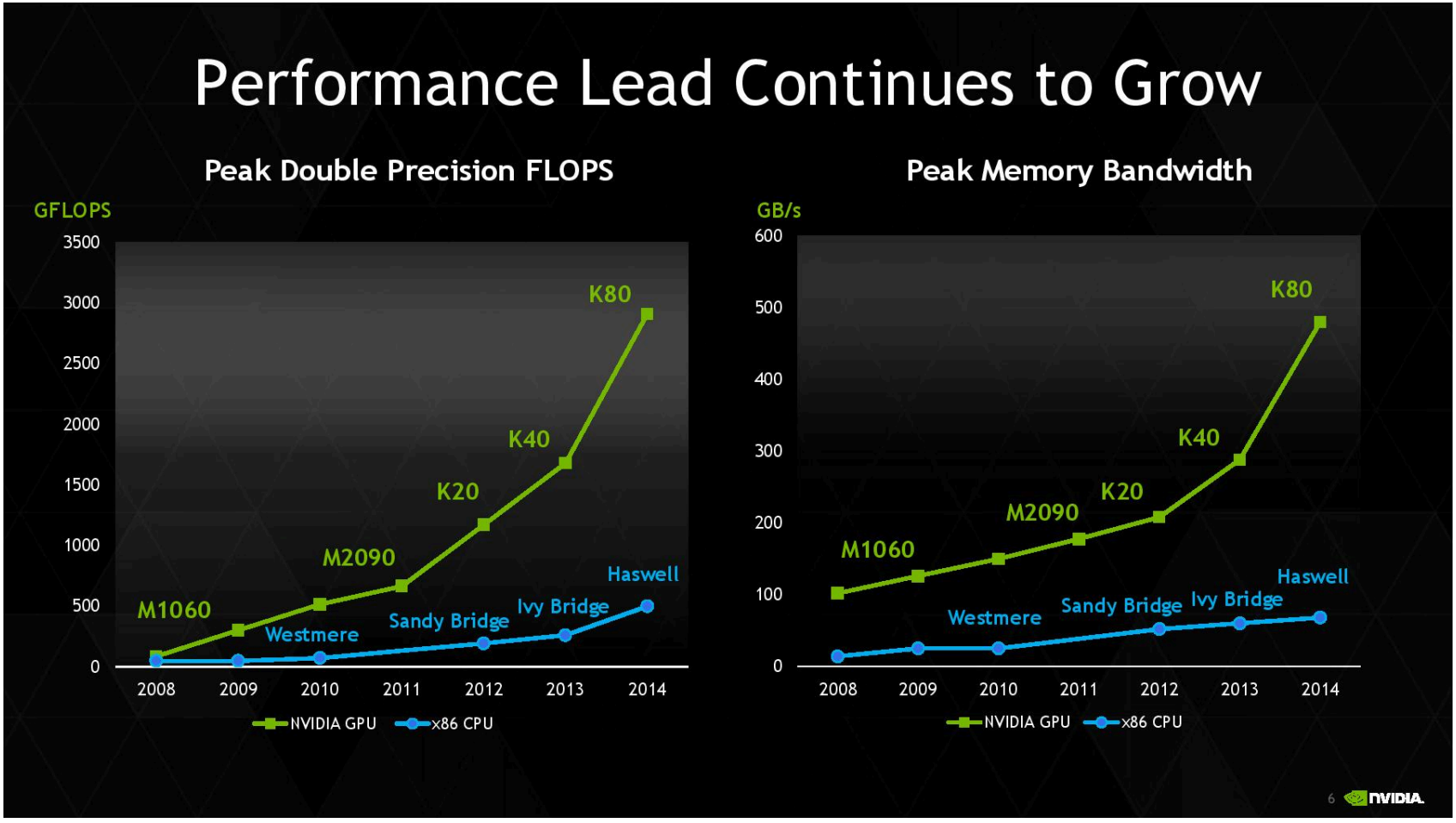
Known global methods, applied to our problem:

- Multi-start method (classic approach)
- MSBH method – Monotonic Sequential Basin-Hopping
- “Big-Bang” method

Finding global extremum (“best-of-known”) for large cluster requires performing of thousands of local descents. So we should use modern parallel computing systems to speedup our algrorithms. All “classic” algorithms uses CPUs (Central Processor Unit) and works on SMP (multi-processor) or cluster systems.

We want to try to use modern GPUs (Graphical Processor Unit) to accelerate global extremum finding.

Why GPU


Performance Lead Continues to Grow
Peak Double Precision FLOPS
GFLOPS
3500
3000
2500
2000
1500
1000
500
0
M1060
M2090
K20
K40
K80
Westmere
Sandy Bridge
Ivy Bridge
Haswell
2008
2009
2010
2011
2012
2013
2014
NVIDIA GPU
x86 CPU
Peak Memory Bandwidth
GB/s
600
500
400
300
200
100
0
M1060
M2090
K20
K40
K80
Westmere
Sandy Bridge
Ivy Bridge
Haswell
2008
2009
2010
2011
2012
2013
2014
NVIDIA GPU
x86 CPU
6
NVIDIA

Why GPU

Pros:

- GPUs have excellent performance
- GPUs are cheap – less than 1000$ for non-professional cards
- As a result, GPUs are accessible to everyone

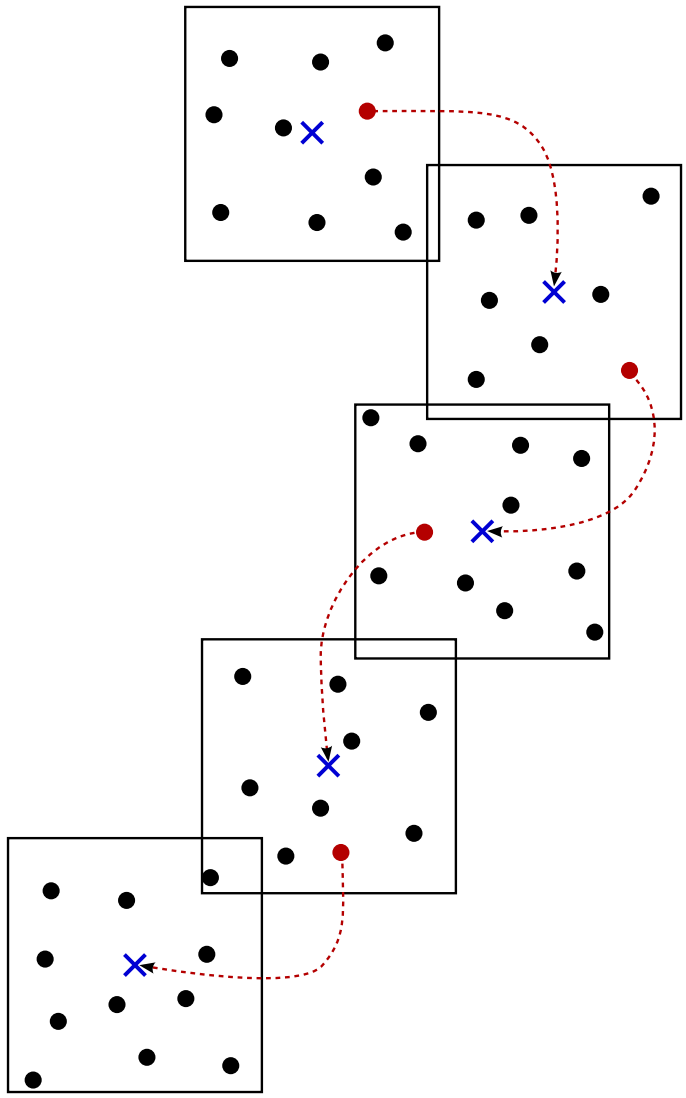
Cons:

- GPUs have a different architecture from the CPU
- GPUs programming is more complex
- Not all problems can be programmed effectively

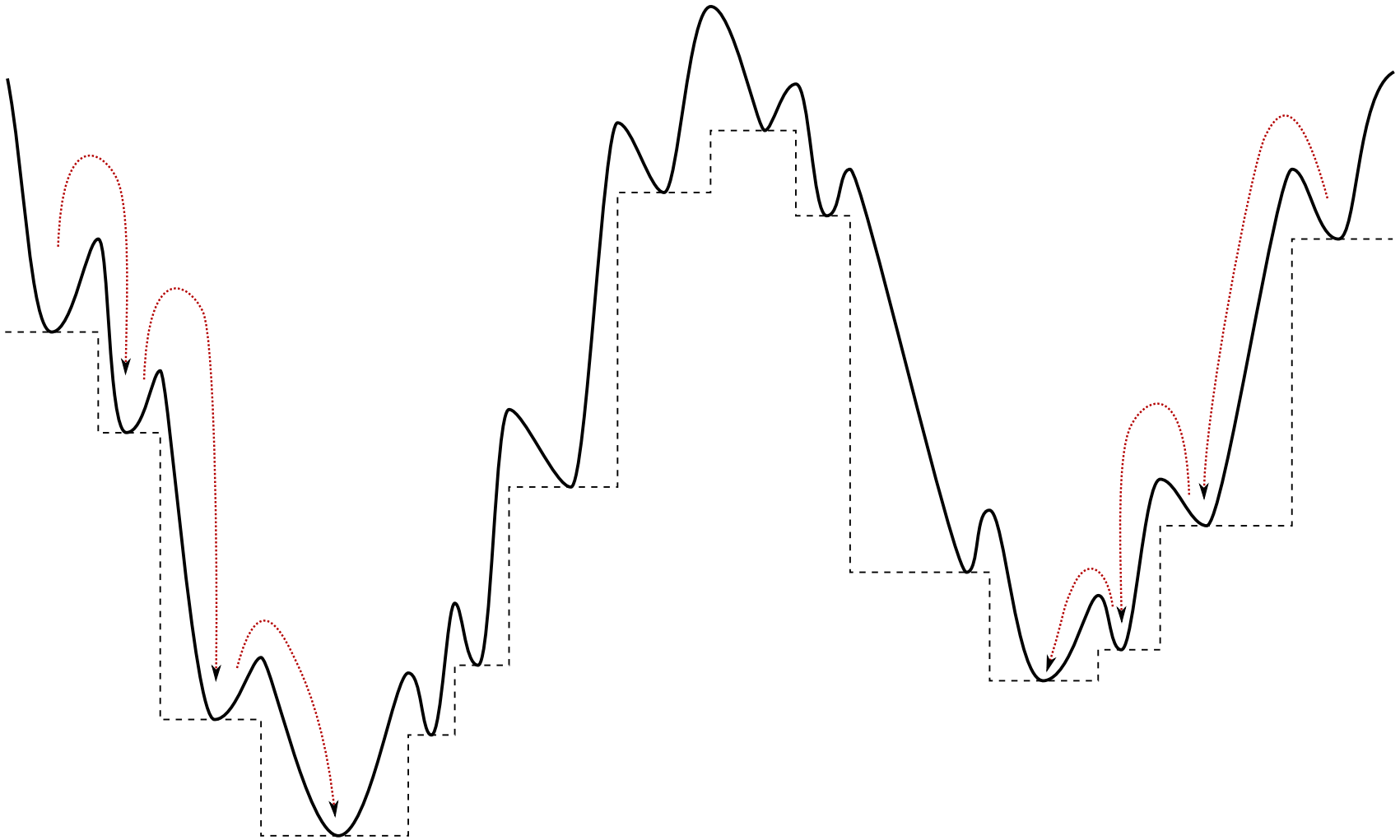
Since we have global optimization problem we should accelerate "upper level" (global method) instead accelerating "lower level" (function/gradient calculation).

Let's take a closer look at the MSBH method as one of the most popular methods for our problem class.

MSBH method

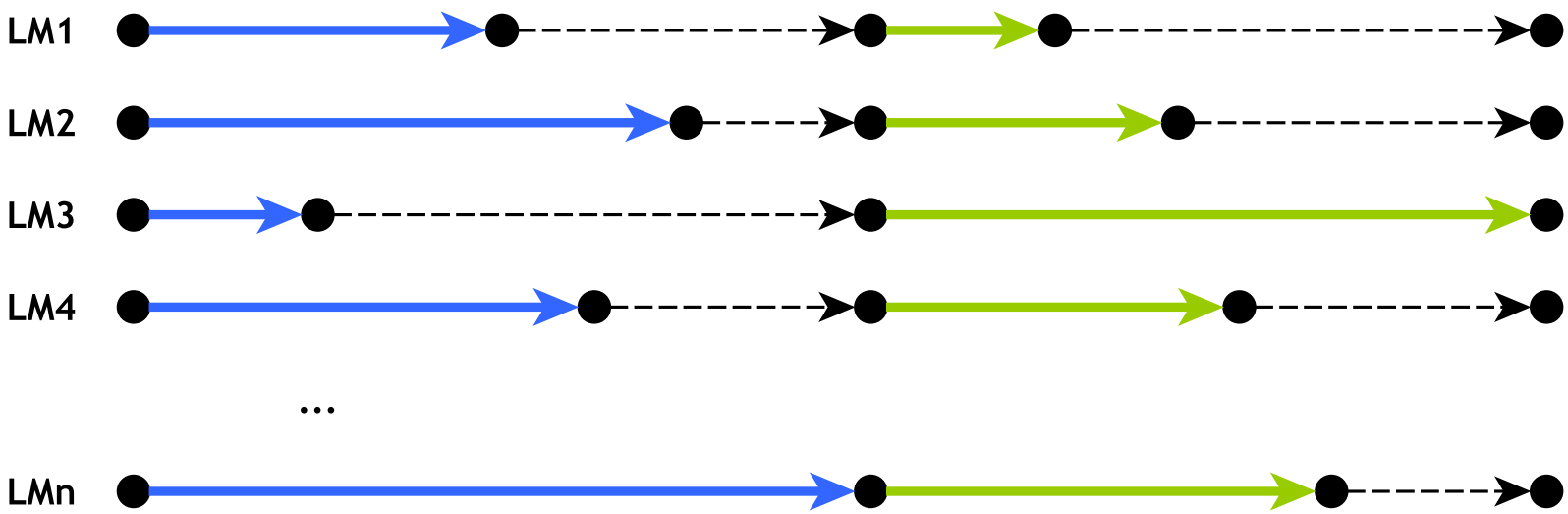


MSBH method



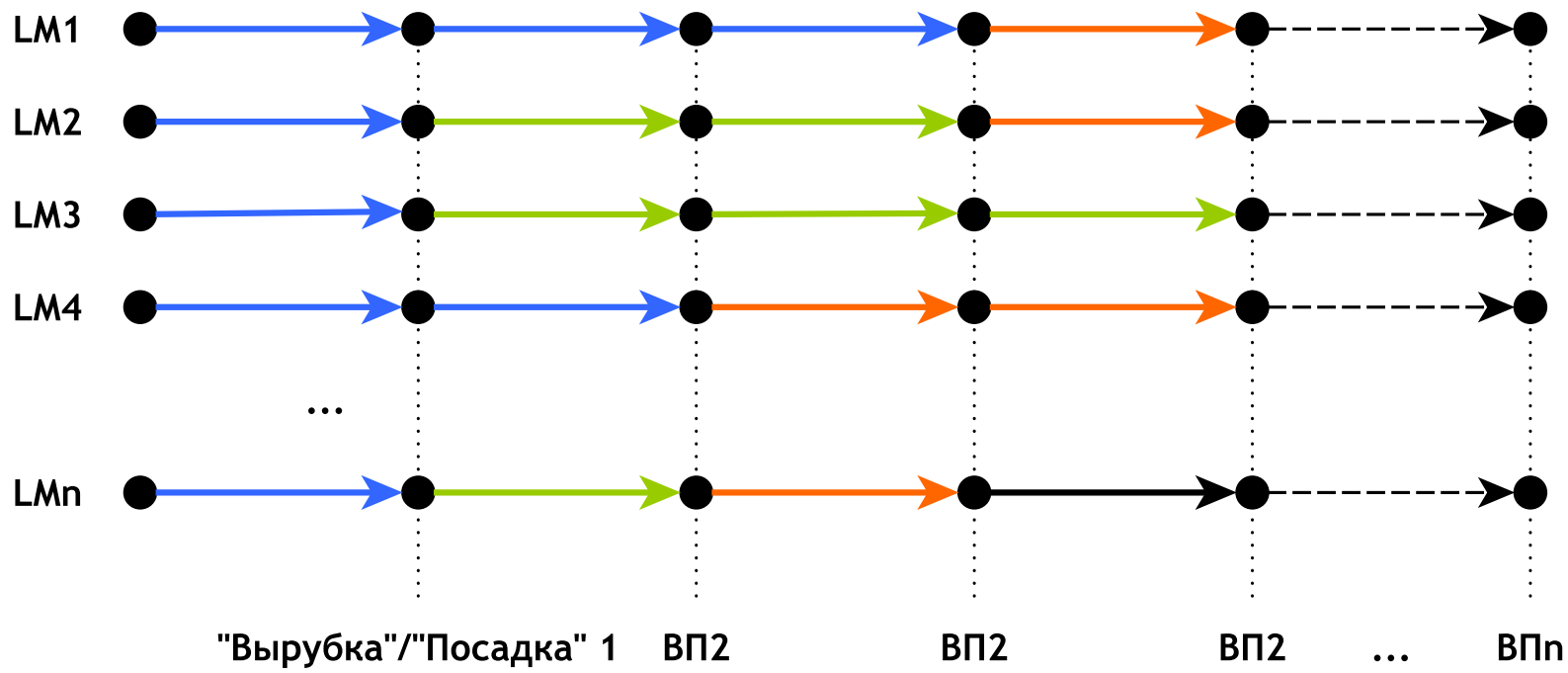
As you can see, we should try to speed up our method by running many local descents in parallel instead of sequential executing.

But here we have problem with different descent execution time. Such operation mode is bad for modern GPUs since they have SIMT (Single Instruction, Multiple Threads) architecture. If some GPU thread finish own local descent it will wait for other threads so one “long” descent slowdown all others.


LM1
LM2
LM3
LM4
...
LMn

To fix this wrong behavior we suggest “Forest” method, which splits local descents into smaller pieces to to minimize GPU threads downtime.

Forest method



SIMT architecture features also require a review of the local descent procedures. Ideally each local method iteration should have permanent computational complexity with unified algorithm.

Such requirements require rethinking of many algorithms, the main of which is the one-dimensional minimization procedure. Typically, one-dimensional search is configured to obtain a sufficiently accurate solution and has a fairly complex algorithm with numerous conditional operators.

In order to maximize the efficiency of work within the GPU architecture, we offer special versions of one-dimensional search, taking into account the features of the problem:

- SA – StepAdation scheme. At each iteration we try to increase step size on success and decrease it when fails (no reaxation).
- `Parabolic(g+)` – quadratic approximation of the minimized function using the gradient value at the current point
- `Parabolic(g-)` – quadratic approximation of the minimized function without using the gradient value at the current point

For all versions we use step history to speedup line-search procedure.

As the basic methods of local optimization, we use our "GPU-friendly" modifications of the **Conjugate Gradient** and quasi-newtonian method **LBFGS**.

For better performance of line-search algorithms we also normalize current descent direction: $d^k = d^k / \|d^k\|_2$. This allows us to unify line-search algorithms for all cases and do not depend on the current absolute values of the gradient.

Numerical experiments

- 2 x Intel E5-2680 v2 (2.8 GHz); total 20 cores, 40 threads
- 128 Gb DDR3 1866 MHz
- GeForce GTX 1060 6GB (1280 CUDA cores)
- Ubuntu server 20.04
- gcc-9.3.0
- CUDA toolkit 11.4

All GPU algorithms works in float-mode (single precision), which is much faster for the current hardware.

CUDA code tricks

- Simplify the CUDA kernels code
- Avoid "`if .. then ... else ...`" with complex branches bodies
- Use `float` instead of `double` if possible
- Provide concurrent CPU and GPU code execution
- Fix memory access patterns
- Fix CPU/GPU memory allocations (pinned memory, etc)
- Reduce data exchange between CPU and GPU
- ...

Morse potential, $n \leq 147$

n	UK (CCD)	ISDCT
20	-97.417393	-97.41739307417
80	-690.577890	-690.577890200415 5952
147	-1531.498857	-1531.498857189995761

Morse potential, $150 \leq n \leq 240$

n	CN	ISDCT
150	-1570.956895	-1570.956894507743300
155	-1639.571558	-1639.571558368015758
160	-1705.794373	-1705.794372516992553
165	-1774.727689	-1774.727688598778741
170	-1842.786500	-1842.786499541551848
175	-1911.754684	-1911.754684452901074
180	-1979.907966	-1979.907965818779076
185	-2048.617785	-2048.617785496087890
190	-2119.104888	-2119.104888297832076
195	-2189.107474	-2189.107474368099702
200	-2260.148943	-2260.148943425931975
205	-2329.258501	-2329.258501195624831
210	-2400.884161	-2400.884161410538582
215	-2473.351504	-2473.226631779617037
220	-2544.094288	-2543.330357862101664
225	-2616.672973	-2616.672972732320432
230	-2691.174648	-2691.174648208746930
235	-2767.215086	-2767.215085893439664
240	-2839.054430	-2839.099924748702961

Morse potential, $n > 240$

n	ISDCT
241	-2852.824892760237617
242	-2866.778881119791095
243	-2882.570361710750603
244	-2897.072046039199449
245	-2910.707949590559110
246	-2924.392845202200078
247	-2940.026081163837716
248	-2955.679013676632167
249	-2971.203337277484479
250	-2985.771711418945870

Numerical experiments

Unfortunately, direct comparison between CPU and GPU methods is impossible:

- Methods are stochastic, have different working threads count and as a result have different pseudo-random sequences (local descent starting points)
- Methods use different line-search algorithms
- Methods use different precision settings (float vs double)

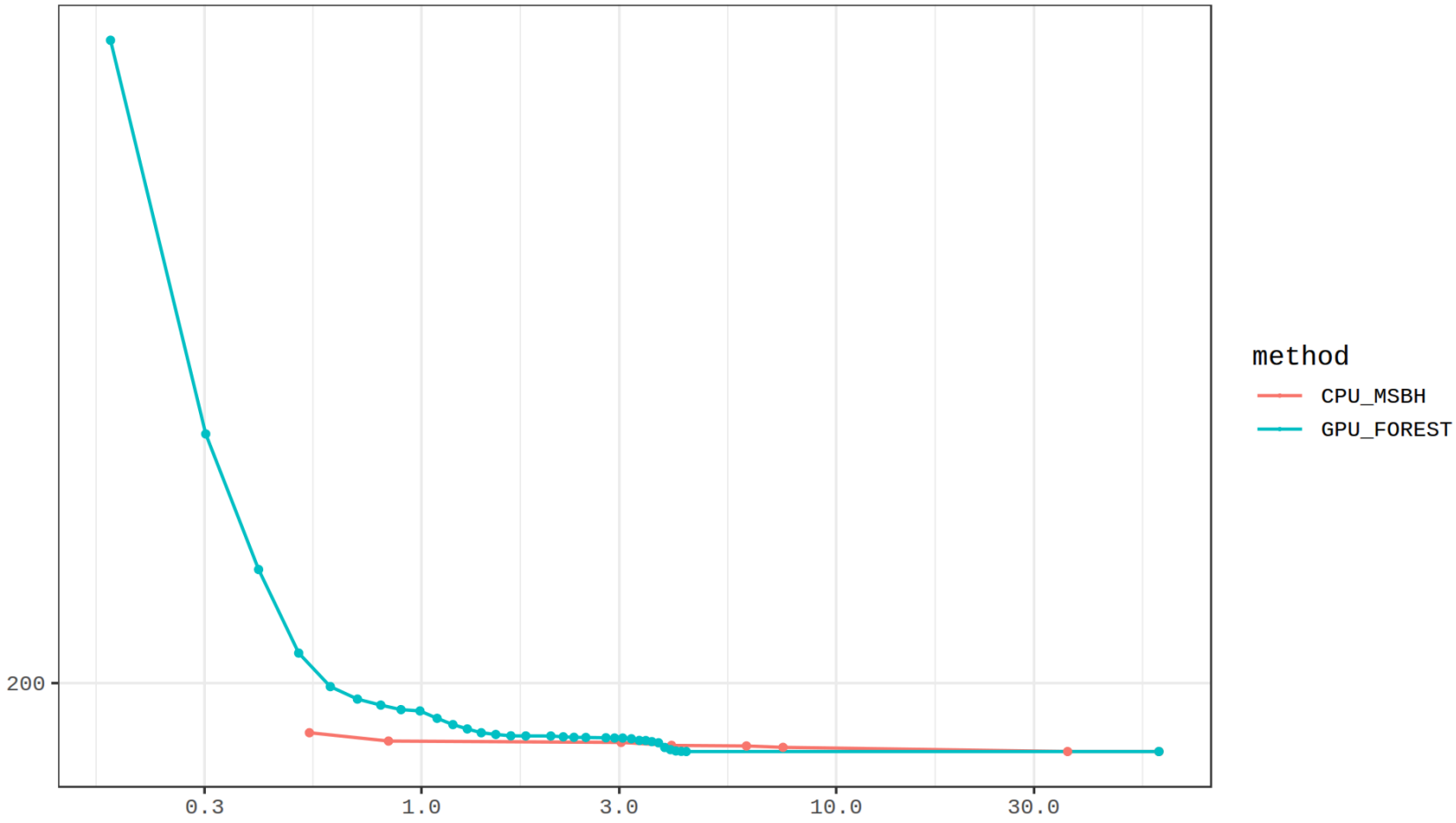
So we will use optimized function value as comparison criteria.

Numerical experiments

Test case:

- Morse cluster with 90 atoms (270 optimized variables)
- Same start point for both methods (MSBH and Forest)
- Time limit – 60 seconds
- CPU algorithm – MSBH + LBFGS with history parameter $m = 3$. 40 CPU cores are used.
- GPU algorithm – Forest + LBFGS with history parameter $m = 3$. 1280 CUDA cores are used.

Example



Function values are increased by 1000 for logarithmic scale, first iteration is removed for more visual display.

Example

- CPU MSBH $f_* = -807.442$, value found in 37 seconds
- GPU Forest $f_* = -807.442$, value found in 4.3 seconds

Both methods found point with same function value (same potential energy) but GPU version done this much faster.

«Forest» – GPU-accelerated method for solving global optimization problems

Anton Anikin
ISDCT SB RAS, Irkutsk

Optimization without borders, 2021