XXIII International Conference
Mathematical Optimization Theory and Operations Research
MOTOR-2024

Application of GPU computing to solving discrete optimization problems

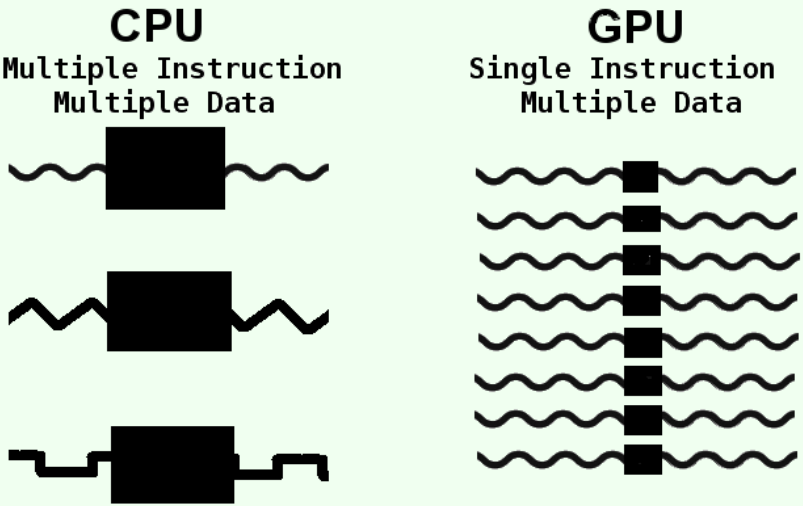
Pavel Borisovsky

Sobolev Institute of Mathematics SB RAS
(Omsk department), Omsk, Russia

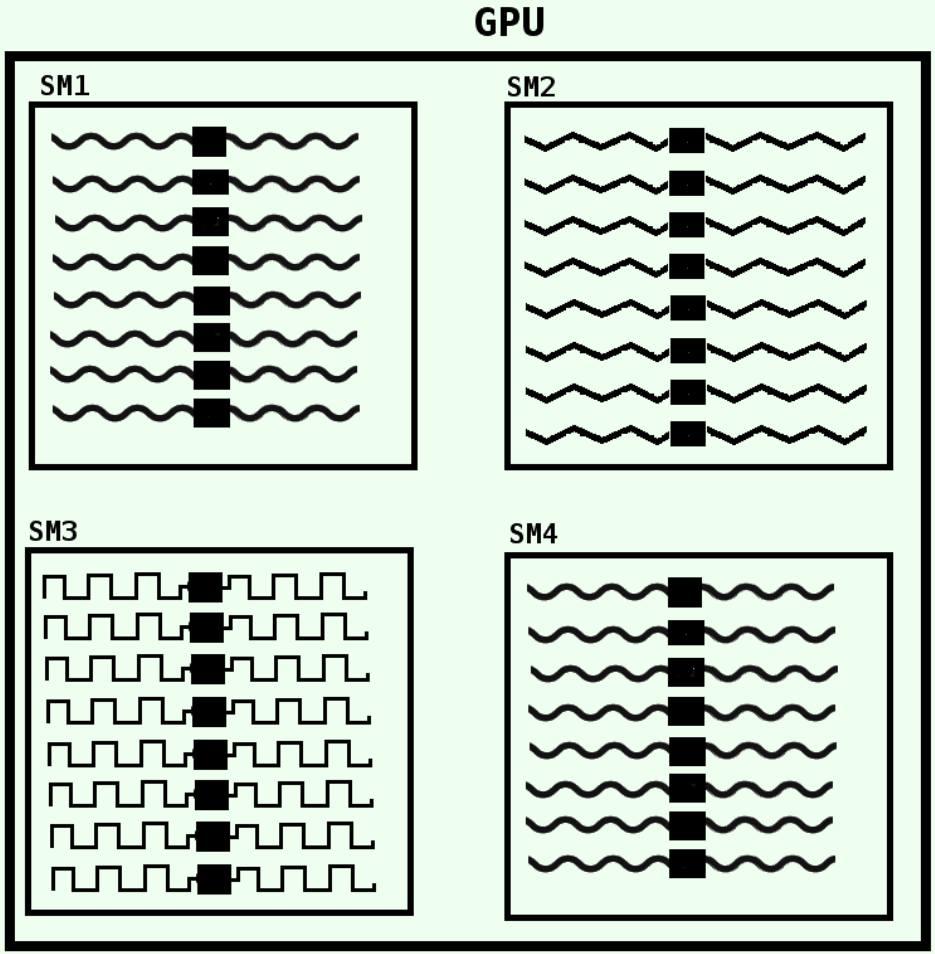
This research is supported by Russian Science Foundation grant 22–71–10015

Graphics processing units

- Large number of cores (several thousands).
- SIMD (single instruction multiple data) architecture.
- Besides video gaming are extensively used in machine learninig, mathematical modeling, cryptocurrency, etc.



Graphics processing units



A very short introduction to CUDA

Nvidia CUDA SDK (Compute Unified Device Architecture) is a tool for GPU programming in C language.

Classic textbook example. Compute the sum of two vectors $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$

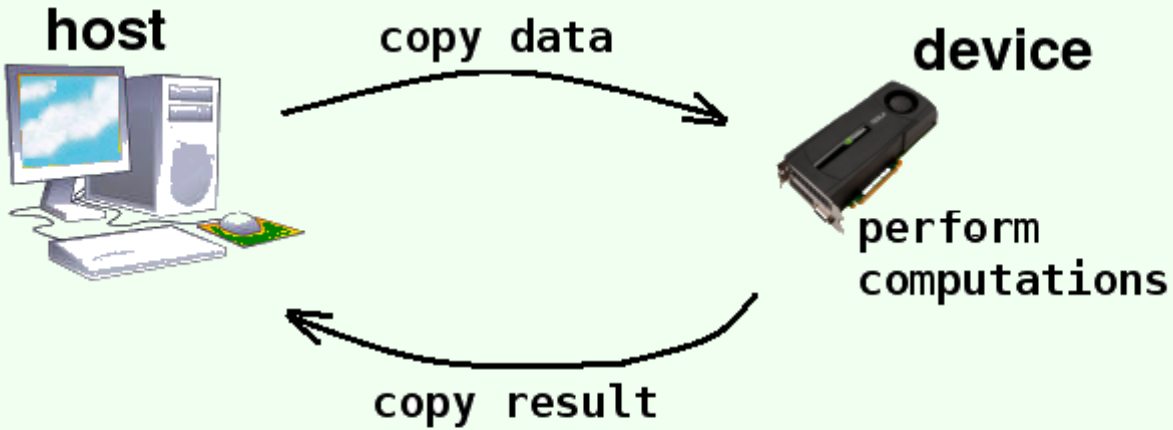
CPU

Sequentially for $\boldsymbol{i} := \mathbf{1}$ to $\boldsymbol{n}$
c[i] = a[i] + b[i]

GPU

Create $\boldsymbol{n}$ threads.
For each thread $\boldsymbol{i}$:
c[i] = a[i] + b[i]

CPU – GPU communications



CUDA basics

HOST

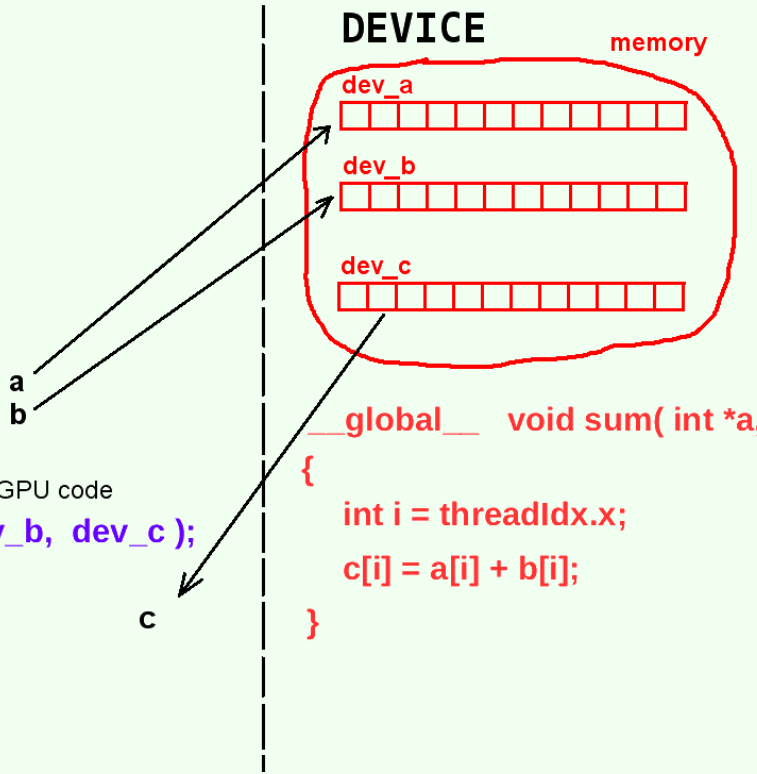
```
void main( void )    {
    int  *a,  *b,  *c;
   // Allocate host memory for a and b
      and fill them with values
   int  *dev_a,  *dev_b,  *dev_c;
   // Allocate GPU memory

   // Copy arrays from host to device

   // Create n threads on GPU and run a GPU code
sum <<< 1, n >>> ( dev_a,  dev_b,  dev_c );

   // Copy result from device to host

   // print the result
}
```



Blocks and threads

Actually, the thing are a bit more complex.

```
sum <<< 1, n >>> ( dev_a, dev_b, dev_c )
```

The number of *threads* $\boldsymbol{n}$ must be $\leq \mathbf{1024}$.

In addition, we may set the number of *blocks* $\boldsymbol{m}$.

```
sum <<< m, n >>> ( dev_a, dev_b, dev_c );
```

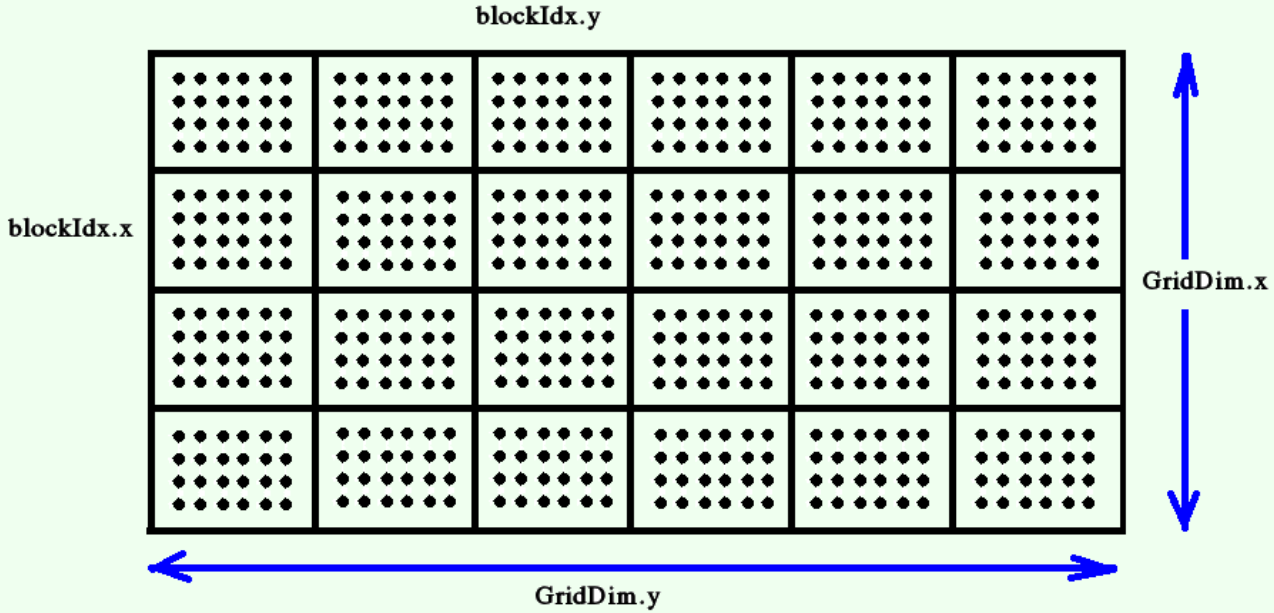
which means $\boldsymbol{m}$ blocks, each one contains $\boldsymbol{n}$ threads.

The balance of threads and blocks significantly affects the performance.
Hint. If you need $\boldsymbol{N}$ threads in total, a reasonable choice for the first attempt could be: $\boldsymbol{m} = \lceil \boldsymbol{N}/\mathbf{32} \rceil$ blocks and $\boldsymbol{n} = \mathbf{32}$ threads in one block. But there could be better settings.

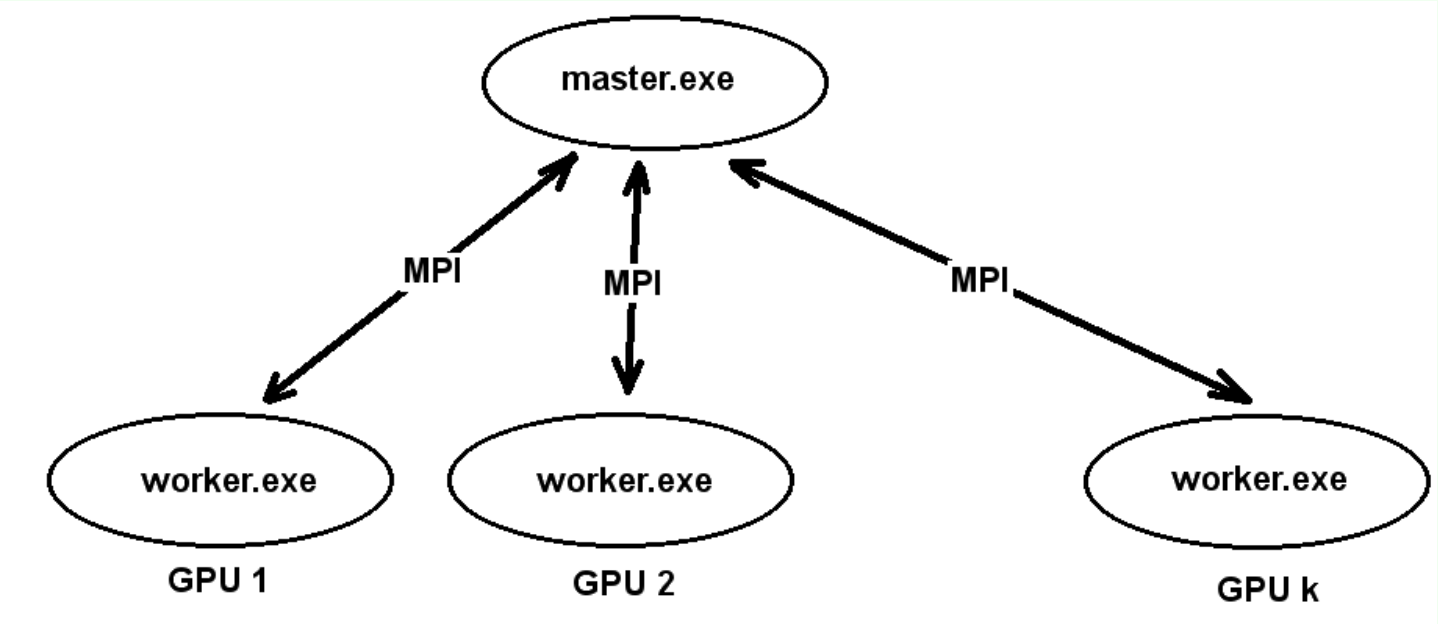
Dimensions of blocks and threads

Blocks and threads can have up to three dimensions.

Example.
`<<< dim3(m1, m0), dim3(n1, n0) >>>`.



Many GPUs


master.exe
MPI
MPI
MPI
worker.exe
worker.exe
worker.exe
GPU 1
GPU 2
GPU k

Example: vertex independent set problem

Find the largest set of vertices in a graph such that there no two vertices in this set connected by an edge. Consider the complete enumeration algorithm. Each set is encoded by an integer number.

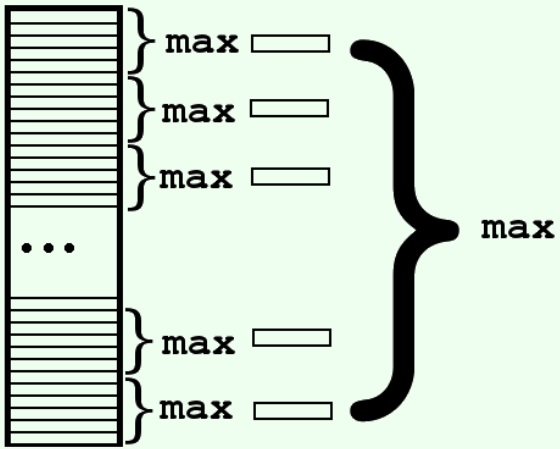
Decimal	Binary	Set
1	00001	{1}
2	00010	{2}
3	00011	{1,2}
...	...	...
30	11110	{2,3,4,5}
31	11111	{1,2,3,4,5}

Let $\boldsymbol{f(x)}$ be the number of vertices if $\boldsymbol{x}$ represents an independent set and $\boldsymbol{f(x) = 0}$ otherwise.

Example: vertex independent set problem

• At the first stage (call it "map"), evaluate $\boldsymbol{f(x)}$ for all $\boldsymbol{x}$ in parallel and store them in memory:
$\boldsymbol{1 \to f(1)}$
$\boldsymbol{2 \to f(2)}$
and so on.

• At the second stage ("reduce"), find $\boldsymbol{x}$ with the maximal $\boldsymbol{f(x)}$. This can be also done in parallel.



https://github.com/pborisovsky/VISP_GPU

Travelling Salesman Problem

Consider classic dynamic programming algorithm (Held, Karp, 1962).
For a set of cities $\boldsymbol{P}$ and city $\boldsymbol{i \notin P}$ let $\boldsymbol{f(i, P)}$ be the optimal path starting from $\boldsymbol{i}$ and visiting all the cities of $\boldsymbol{P}$.

Bellman equations.

$$\boldsymbol{f(i, \{j\}) = a_{ij}}$$

$$\boldsymbol{\dots}$$

$$\boldsymbol{f(i, \{P\}) = \min_{j \in P} \{a_{ij} + f(j, P \setminus \{j\})\}}$$

In the algorithm, for each $\boldsymbol{k}$ the values $\boldsymbol{f(i, P)}$ are evaluated for all the subsets $\boldsymbol{P}$ such that $\boldsymbol{Card(P) = k}$.

https://github.com/pborisovsky/tsp-dp-gpu

Experimental results

n	CPU AMD Phenom II (2008)	CPU AMD EPYC 7502 (2019)	GPU GTS 450 (2010)	GPU Tesla V100 (2017)
24	24 s.	13.5 s.	0.75 s.	14 ms.
25	53 s.	30 s.	1.5 s.	29 ms.
26	120 s.	71 s.	-	65 ms.
27	267 s.	153 s.	-	140 ms.

Can be used in other algorithms for optimization of small parts of a sequence.

Borisovsky P. Exact Solution of One Production Scheduling Problem // In: Optimization Problems and Their Applications. OPTA 2018. Communications in Computer and Information Science, Springer, Cham. Vol. 871. 2018. pp. 56–67.

Borisovsky P. A., Eremeev A.V., Kallrath J. Multi-product continuous plant scheduling: combination of decomposition, genetic algorithm, and constructive heuristic // International Journal of Production Research. V. 58. N.9. – 2020.- P. 2677–2695.

A short literature review: Exact algorithms

Borisenko, A., Haidl, M. & Gorlatch, S. A GPU parallelization of branch-and-bound for multiproduct batch plants optimization // Journal of Supercomputing. 73. - 2017. - P. 639–651.

Картак В.М., Рипатти А.В. Параллельный подход к решению задачи одномерной продолженной упаковки (1cbpp) с использованием технологии CUDA // Вестник Башкирского университета. Т.18.N.1. - 2013. - С. 11–14.

Попов М.В., Посыпкин М.А. Эффективная реализация точных алгоритмов решения задач дискретной оптимизации на графических ускорителях // Современные информационные технологии и ИТ-образование. Т.14. N.2. - 2018. - С. 408–418.

A short literature review: Metaheuristics

Luong T.V., Melab N., and Talbi E. GPU Computing for Parallel Local Search Metaheuristic Algorithms // in IEEE Transactions on Computers, V.62, N.1., 2013. pp. 173-185.

Schulz C., Hasle G., Brodtkorb A.R., Hagen T. R. GPU computing in discrete optimization. Part II: Survey focused on routing problems // EURO Journal of Transportation and Logistics. V.2. - 2013.- P. 159–186.

Pedemonte M., Nesmachnow S., Cancela H. A survey on parallel ant colony optimization // Applied Soft Computing, 11 (2011) 5181–5197.

Tan Y., Ding K. A survey on GPU-based implementation of swarm intelligence algorithms // IEEE Transactions on Cybernetics V.46, N.9. 2016. 2028 — 2041.

Does Gurobi support GPUs?

The Gurobi development team is watching GPUs (Graphics Processing Units) closely, but up to this point, all of the evidence indicates that they aren't well suited to the needs of an LP/MIP/QP solver. Specifically: GPUs don't work well for sparse linear algebra.... GPUs are built around SIMD computations, ... is not well suited to the needs of parallel MIP.

The Gurobi Optimizer is designed to effectively exploit multiple cores in a CPU, so you'll definitely see a benefit from more parallelism in the future.

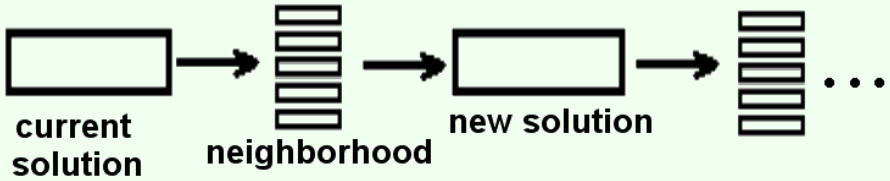
GPUs do not help: Gurobi has invested significant time around this question. Our current assessment is GPUs are not able to provide a performance benefit for solving optimization problems.

https://support.gurobi.com/

Difficulties in practical application of GPU computing

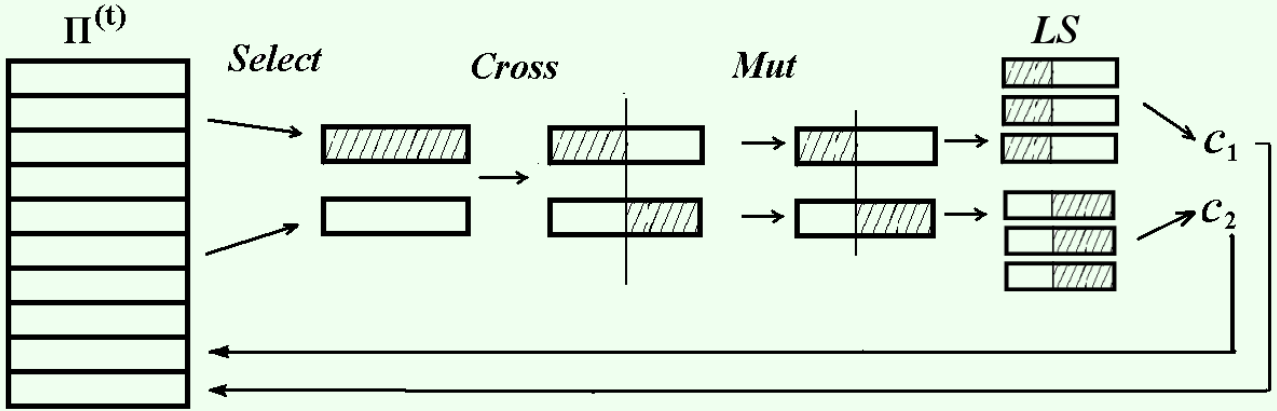
- Complex models (large number of variables and constraints, changing requirements).
- Complex technologies (blocks and threads, memory hierarchy, coalescing of memory requests, complex debugging).
- Complex algorithms (MIP methods, Branch and Bounds, etc.)

Local search



- Deterministic: enumerate the whole neighborhood, or some part, and choose the best neighbor.
- Randomized: generate several random neighbors and choose the best one. This is equivalent to the $(1+\boldsymbol{\lambda})$-Evolutionary algorithm.
- Iterated Local Search (ILS): when a local optimum is reached, perform "shaking" procedure that makes relatively large changes to the current solution and repeat the local search.

Genetic Algorithm with Local Search (Memetic Algorithm)

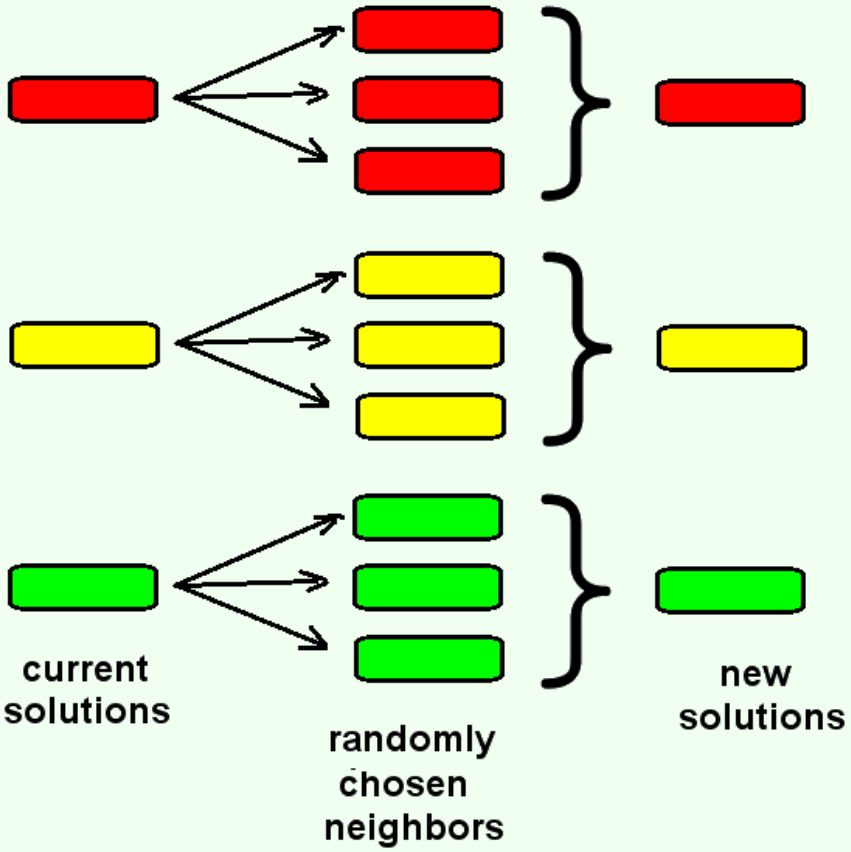


The local search part can be implemented in parallel on GPU.

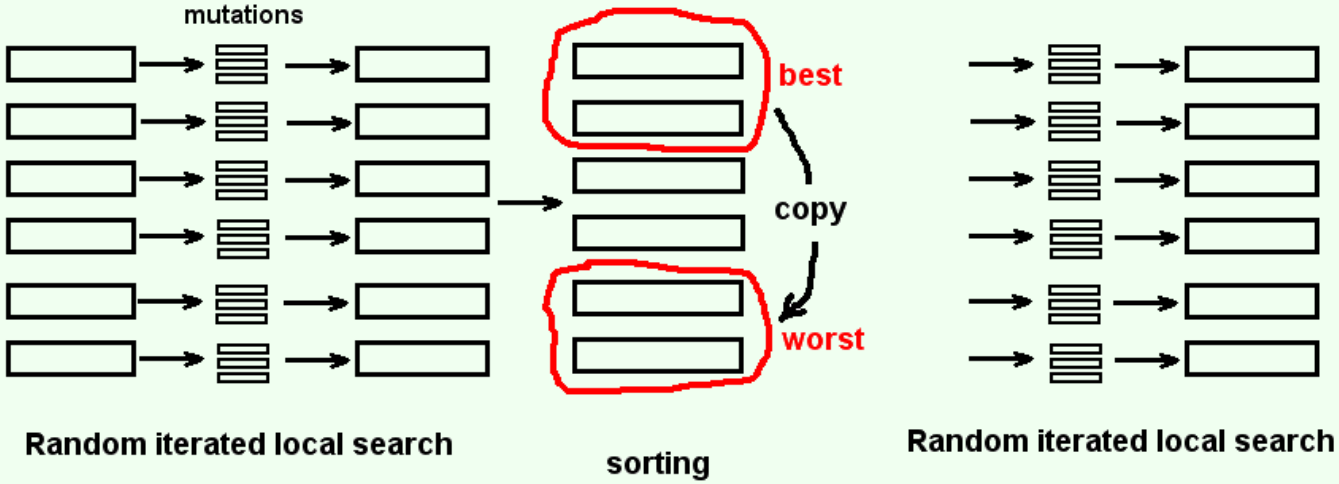
Borisovsky, P., Kovalenko, Y. A Memetic Algorithm with Parallel Local Search for Flowshop Scheduling Problems // In: Bioinspired Optimization Methods and Their Applications (BIOMA 2020). Lecture Notes in Computer Science, 12438 LNCS. pp. 201–213 (2020)

Borisovsky P. A parallel greedy approach enhanced by genetic algorithm for the stochastic rig routing problem // Optimization Letters. V. 18, P. 235–255 (2023)

Parallel randomized local search



Parallel "Go with the Winners (GWW)" algorithm with Iterative Local Search

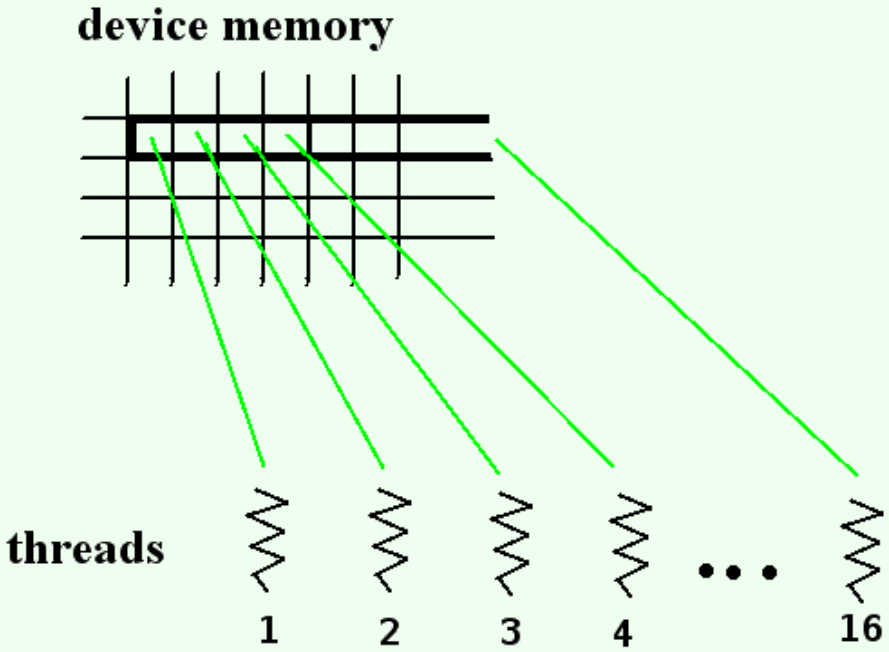


Aldous D., Vazirani U.: "Go with the winners" Algorithms. Proceedings of 35th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 492–501 (1994)

Borisovsky P.A. A Parallel "Go with the Winners" Algorithm for Some Scheduling Problems, Journal of Applied and Industrial Mathematics, V.17, N.4, P. 687–697 (2023)

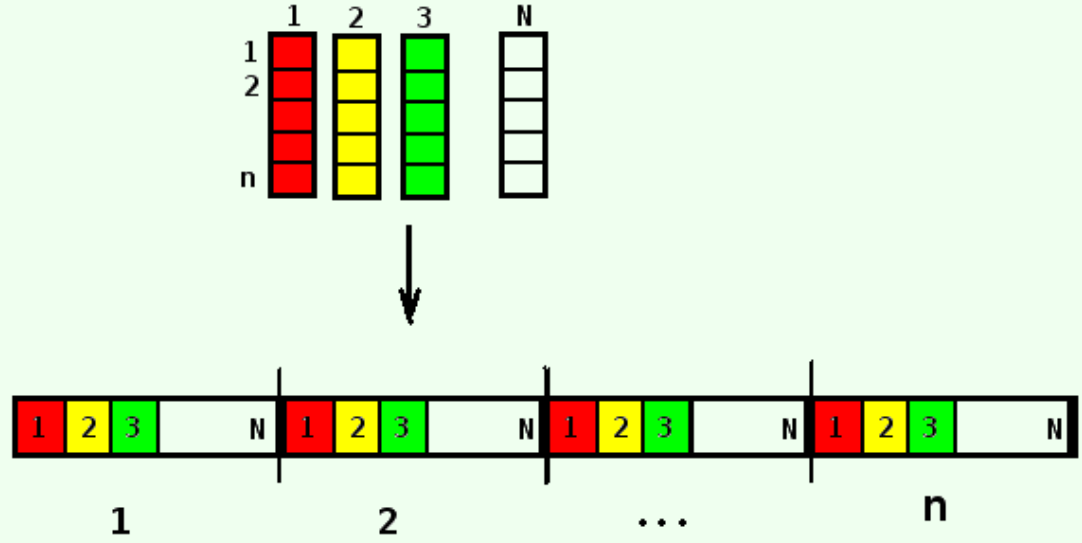
Code optimization: Memory coalescing

If a sequence of threads accesses neighboring memory cells in order (e.g., some thread $\boldsymbol{i}$ reads cell $\boldsymbol{k}$, thread $\boldsymbol{i+1}$ reads cell k+1, etc.) then requests can be combined into one.
(*https://docs.nvidia.com/cuda/cuda-c-best-practices-guide*)



Usage of memory coalescing

In the populational heuristics (Genetic algorithm, Go with the winners, etc.) it is recommended to store population by columns.



OpenCL: an alternative to CUDA

Basically the same concepts but different terminology.

CUDA	OpenCL
Thread	Work item
Block	Work group

More "boilerplate" code in OpenCL.

CUDA

```
my_func<<<m, n>>>(...);
```

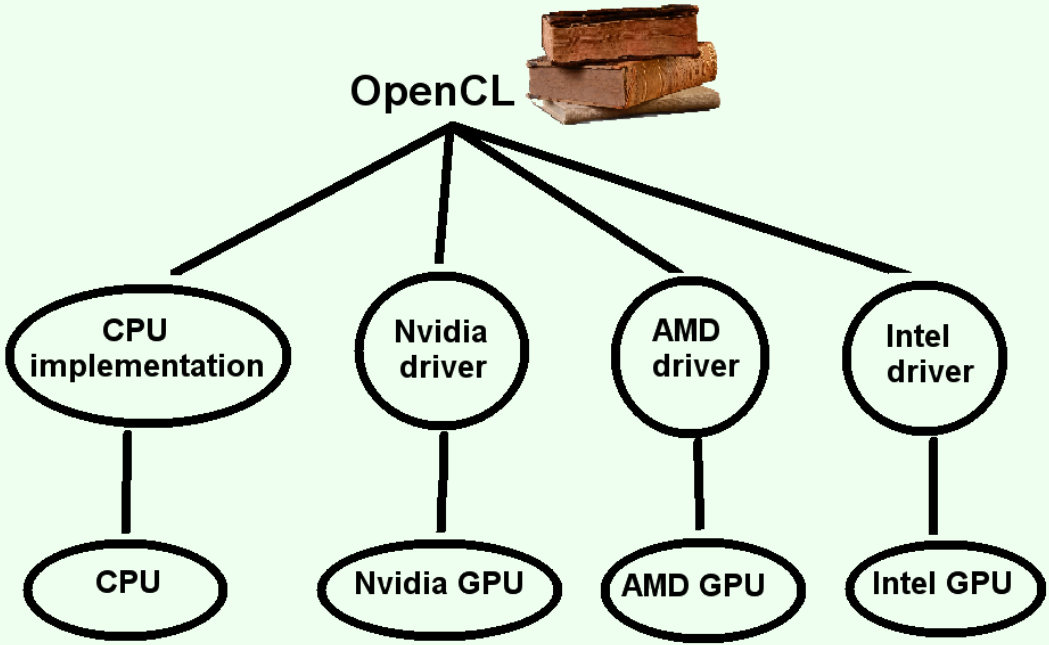
OpenCL

```
kernel = clCreateKernel(...);
queue = clCreateCommandQueue(...);
clSetKernelArg(...);
clSetKernelArg(...);
clEnqueueNDRangeKernel(...)
```

OpenCL: an alternative to CUDA

CUDA is an extension to C programming language (and a compiler).

OpenCL is a specification.



GPU support in high-level languages

Usually, it consists of writing the GPU part of code in C and call it from the high-level language.

CUDA	**OpenCL**
PyCuda	PyOpenCL
jcuda	jocl Aparapi
Julia programming language (high-level model)	

Frequently asked question

"Can I run MFA (My Favorite Algorithm) on GPU?"

- If it is a matheuristic that relies on CPLEX, Gurobi, etc., then the answer is "no".

- If it cannot be parallelized then it will be rather useless.

- If it can be parallelized then it is worth to try.

In fact, the question should be

"What speed-up can be acheved with GPU?".

The answer depends on many things: the algorithm itself, the implementation, and the hardware.

Thank you for your attention!