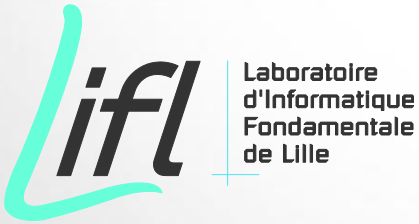
Ph.D. Defense - Thé Van **LUONG**

December 1st 2011

Parallel **Metaheuristics** on **GPU**

Advisors: **Nouredine MELAB** and **El-Ghazali TALBI**



Outline

Optimization problems

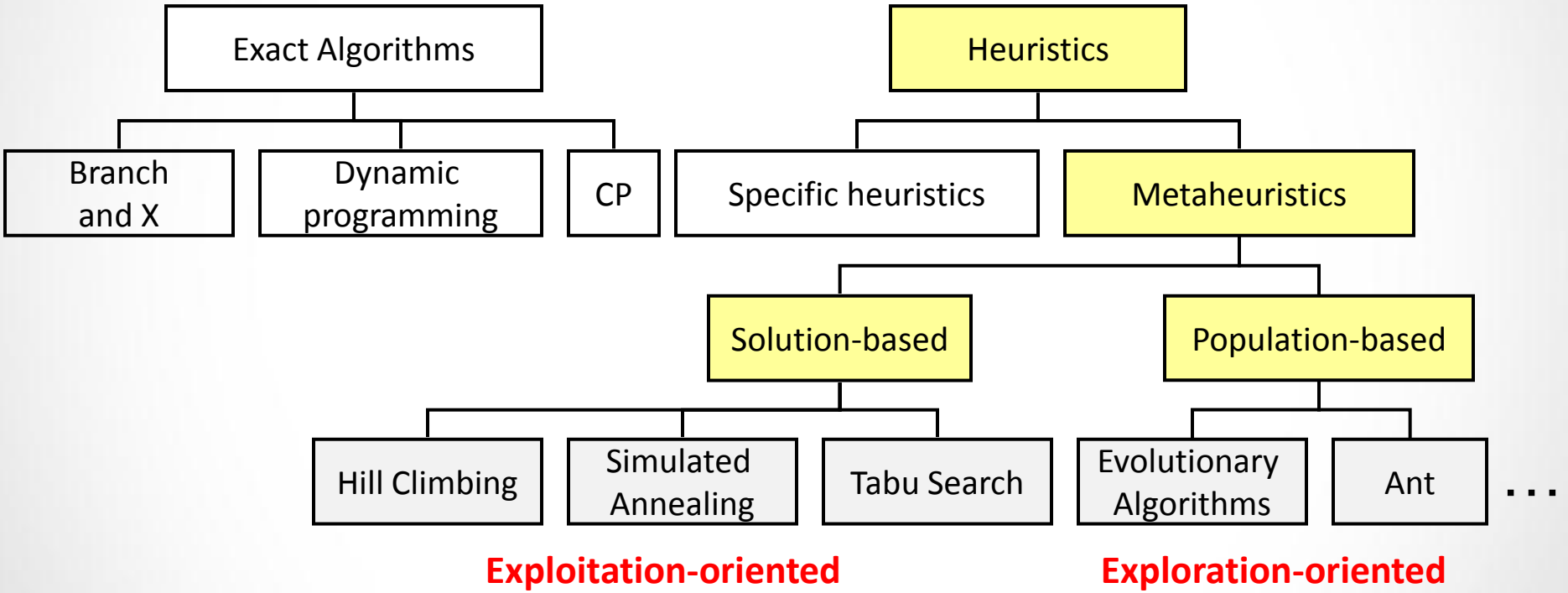
(Mono-Objective) $\quad Min\ f(x) \qquad x \in S$

(Multi-Objective)

$$\begin{cases} Min\ f(x) = (f_1(x),\ f_2(x), \ldots,\ f_n(x)) \qquad n \geq 2 \\ \text{Const.}\quad x \in S \end{cases}$$

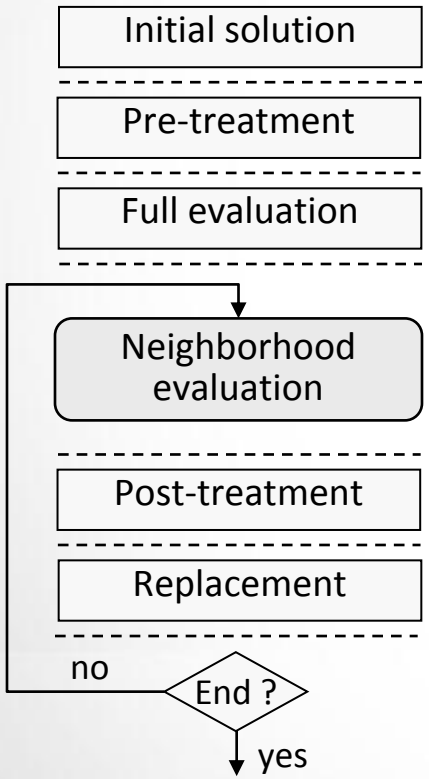
- High-dimensional and complex optimization problems in many areas of industrial concern
  - Telecommunications, Transport, Biology, …

A taxonomy of optimization methods



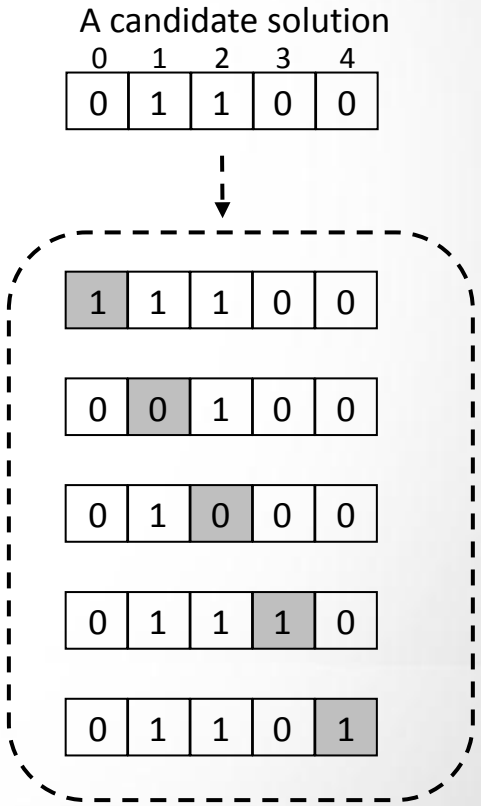
- Exact methods: optimality but exploitation on small size problem instances
- Metaheuristics: near-optimality on larger problem instances, but …
  - … Need of **massively parallel computing** on very **large** instances

Single solution-based metaheuristics

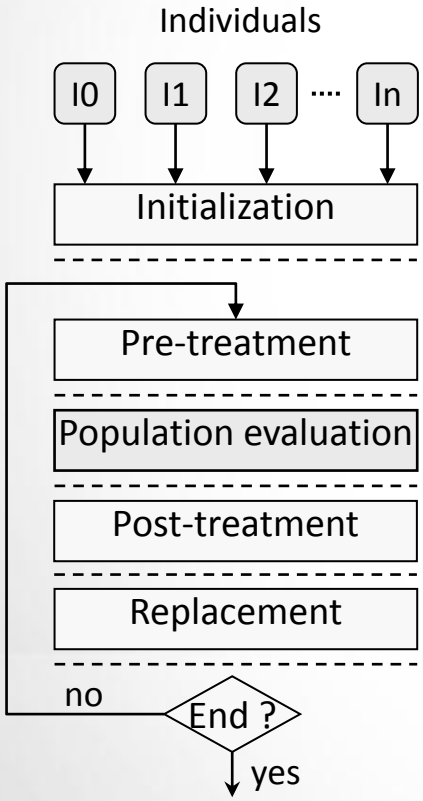


Knapsack problem

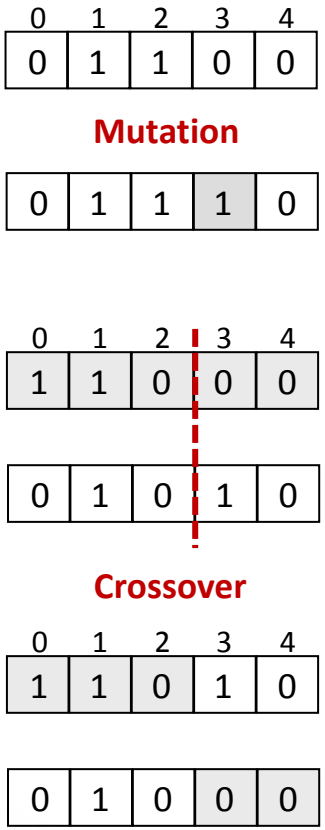
- Solution: binary encoding
- Neighborhood example: Hamming distance of one



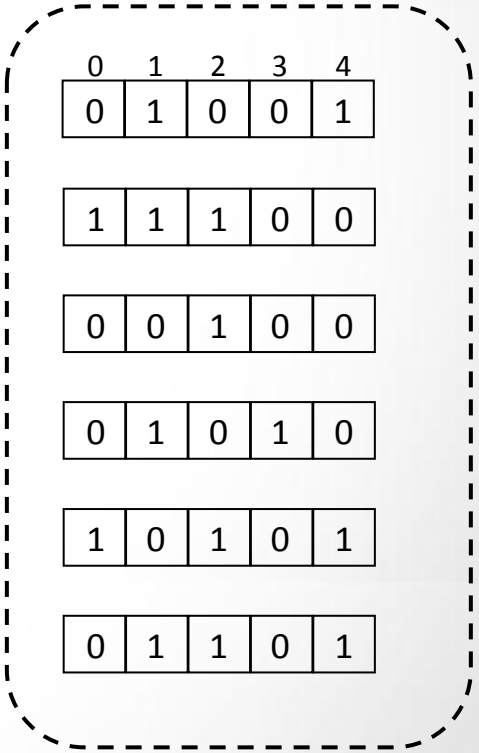
Population-based metaheuristics



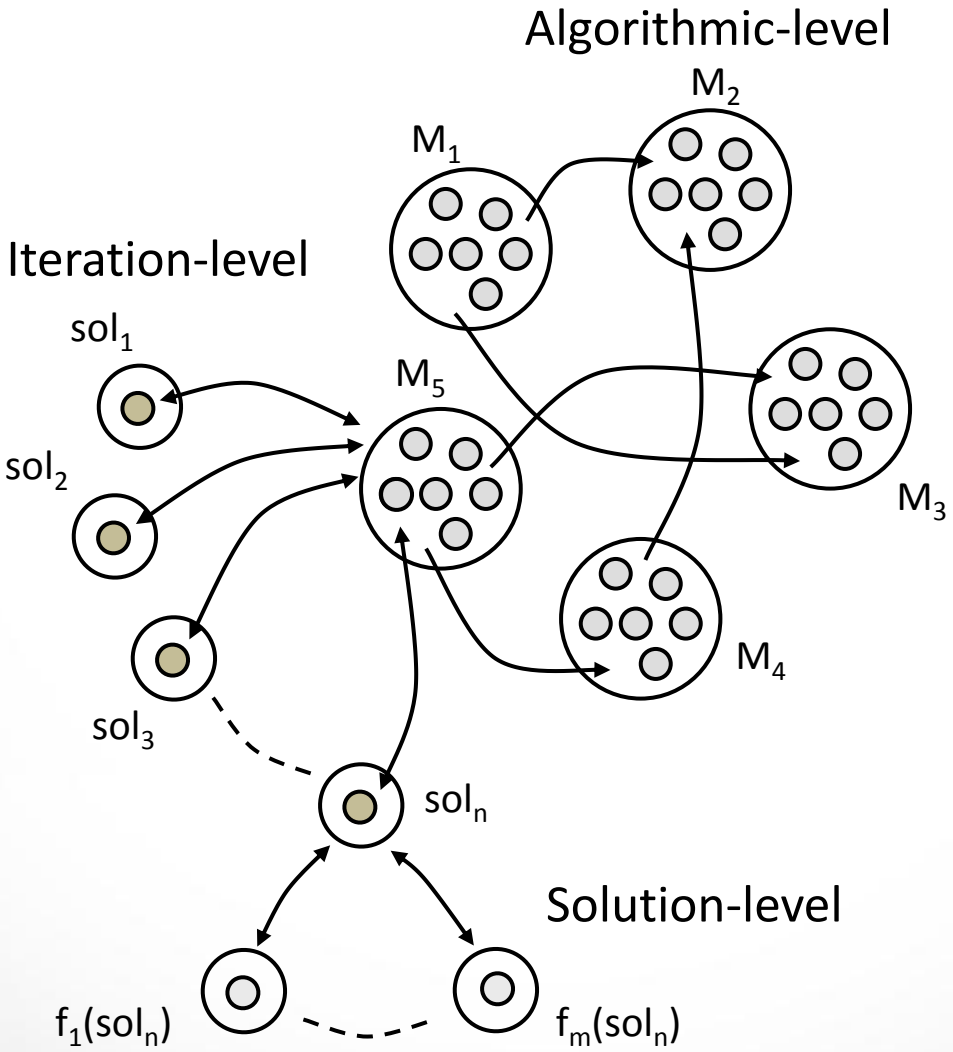
Evolutionary algorithms



Population of solutions



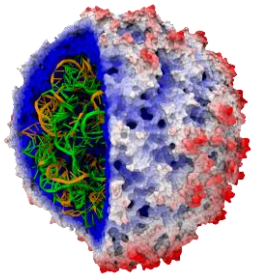
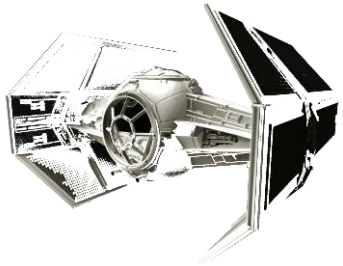
Parallel models of metaheuristics



Previous parallel approaches

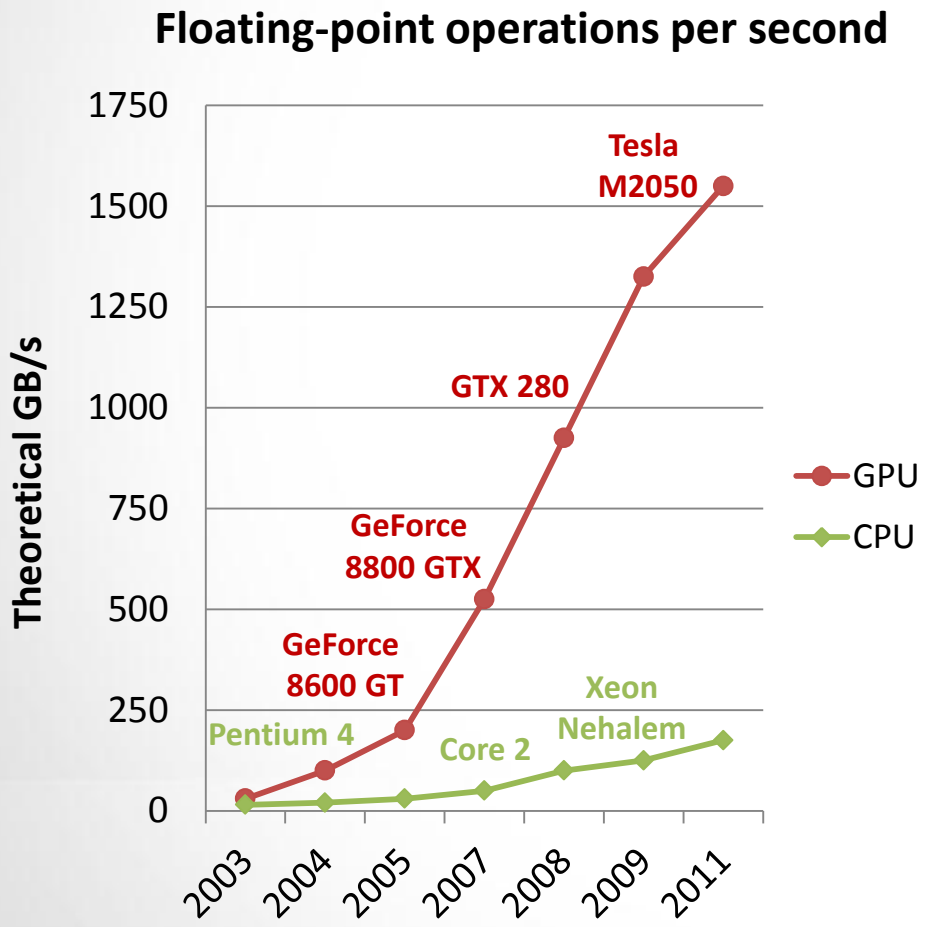
- Parallelization concepts of metaheuristics
  - S-metaheuristics: simulated annealing [Chandy *et al.* 1996], tabu search [Crainic *et al.* 2002], GRASP [Aiex *et al.* 2003]
  - P-metaheuristics: genetic programming [André *et al.* 1996], ant colonies [Gambardella *et al.* 1999], evolutionary algorithms [Alba *et al.* 2002]
  - Unified view of parallel metaheuristics [Talbi *et al.* 2009]
- Implementations on parallel and distributed architectures
  - Massively Parallel Processors [Chakrapani *et al.* 1993]
  - Clusters and networks of workstations [*Garcia et al.* 1994, Crainic *et al.* 1995, Braun *et al.* 2001]
  - Shared memory or SMP machines [Bevilacqua *et al.* 2002]
  - Large-scale computational grids [Tantar *et al.* 2007]

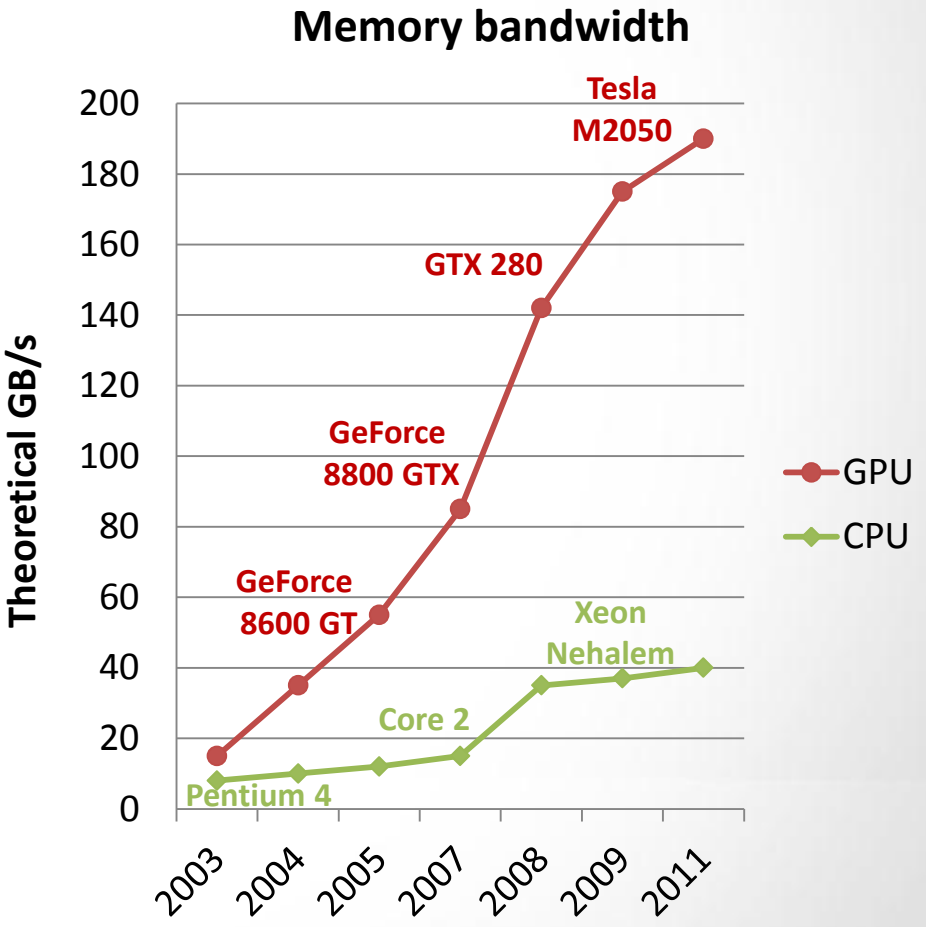
Graphics Processing Units (GPU)


- Used in the past for graphics and video applications …
- … but now popular for many other applications such as scientific computing [Owens *et al*. 2008]
- Popularity due to the publication of the CUDA development toolkit allowing …
  - … GPU programming in a C-like language [Garland *et al*. 2008]

GPU trends

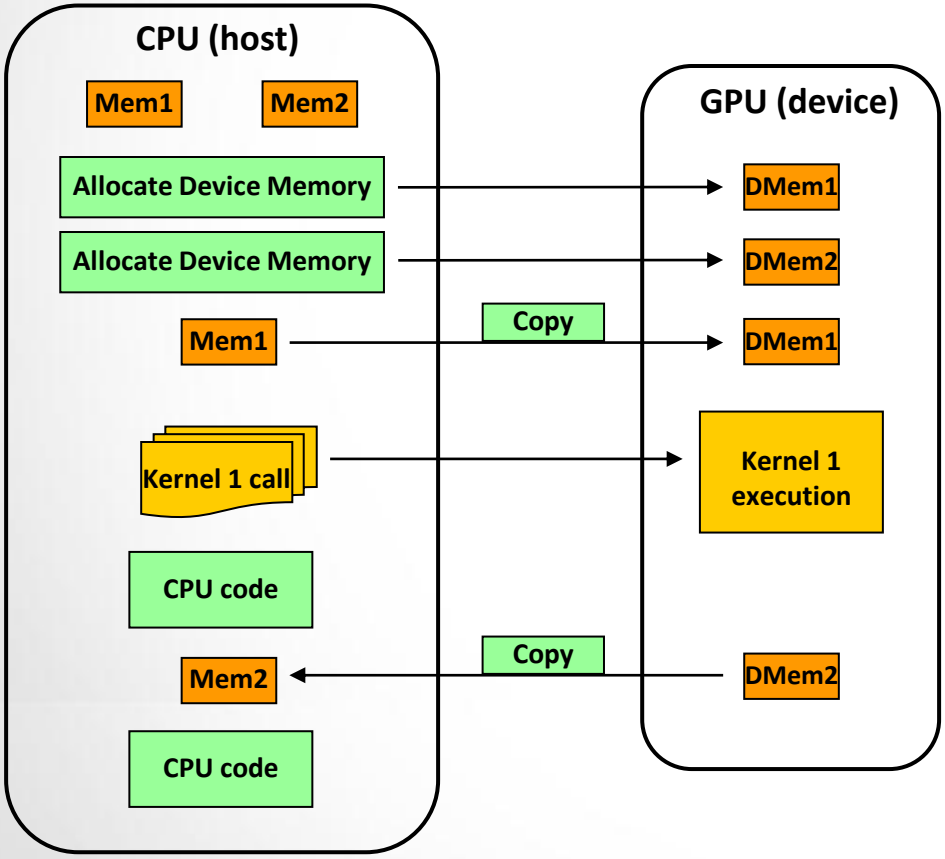

Floating-point operations per second
Theoretical GB/s
1750
1500
1250
1000
750
500
250
0
2003
2004
2005
2007
2008
2009
2011
Tesla M2050
GTX 280
GeForce 8800 GTX
GeForce 8600 GT
Pentium 4
Core 2
Xeon Nehalem
GPU
CPU


Memory bandwidth
Theoretical GB/s
200
180
160
140
120
100
80
60
40
20
0
2003
2004
2005
2007
2008
2009
2011
Tesla M2050
GTX 280
GeForce 8800 GTX
GeForce 8600 GT
Pentium 4
Core 2
Xeon Nehalem
GPU
CPU

Hardware repartition

- CPU: complex instructions, flow control
- GPU: compute intensive, highly parallel computation

General GPU model



- The CPU is considered as a host and the GPU is used as a device coprocessor
- Data must be transferred between the different memory spaces via the PCI bus express …

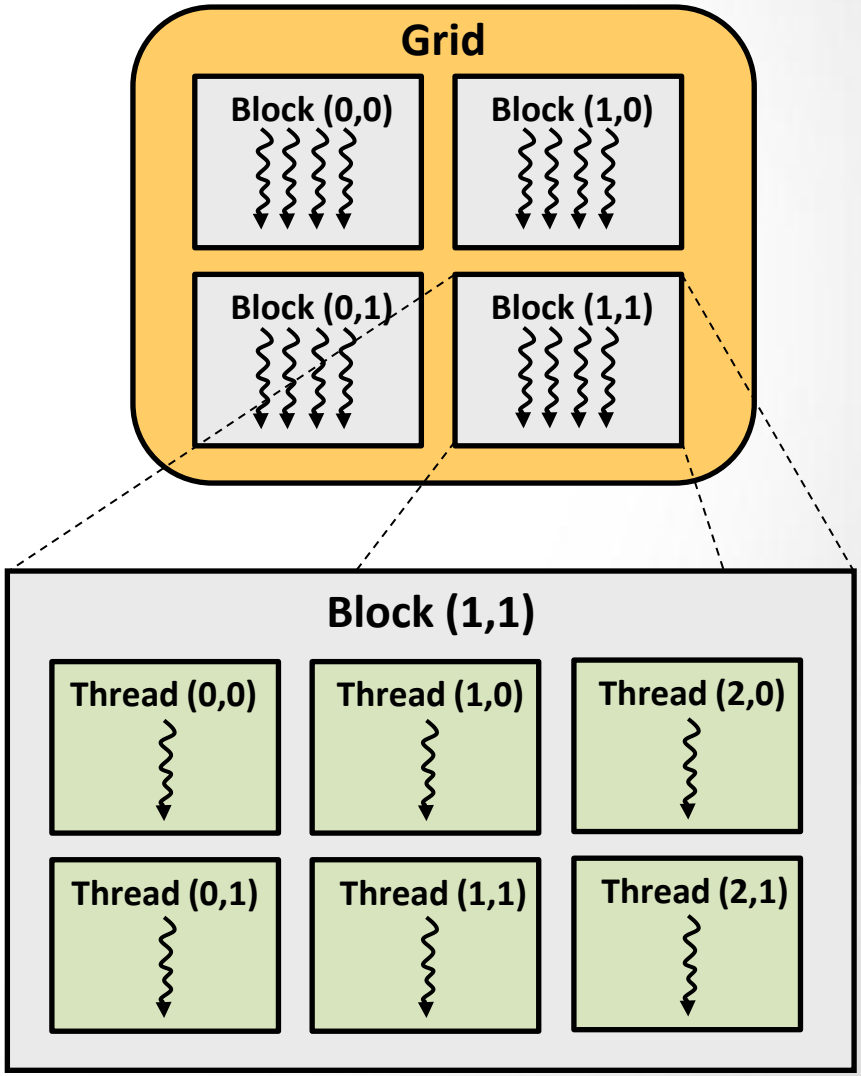
Configuration	CPU -> GPU	
Core 2 Duo T5800 GeForce 8600M GT	1.76×10^{-2}s	1700 MB/s
Core 2 Quad Q6600 GeForce 8800 GTX	1.66×10^{-2}s	1800 MB/s
Xeon E5450 GeForce GTX 280	1.25×10^{-2}s	2400 MB/s
Xeon E5620 Tesla M2050	0.81×10^{-2}s	3700 MB/s

One transfer of 30 MB

- … many data transfers might become a bottleneck in the performance of GPU applications

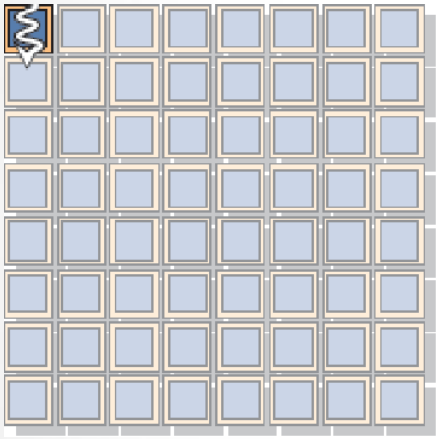
Programming model: SPMD model

- Kernel execution is invoked by CPU over a **compute grid**
  - Subdivided in a set of thread blocks
  - Containing a set of threads with access to shared memory
- All threads in the grid run the **same program**
  - Individual data and individual code flow (Single Program Multiple Data)

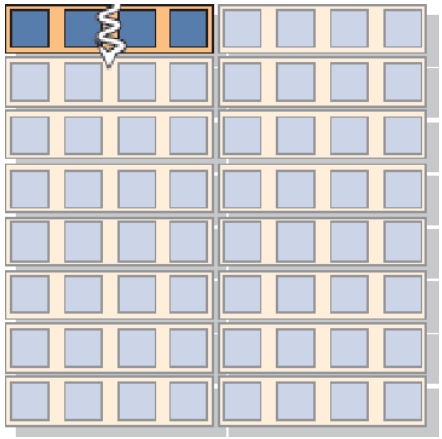


Execution model: SIMD model

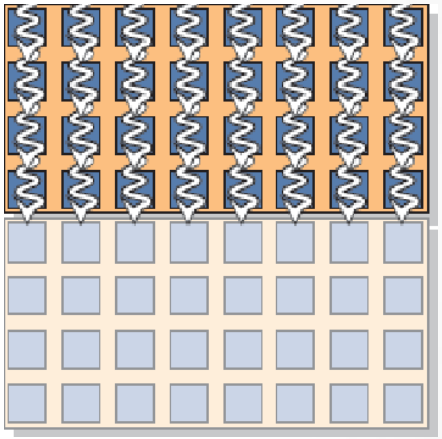
CPU scalar op:



CPU SSE op:



GPU Multiprocessor:

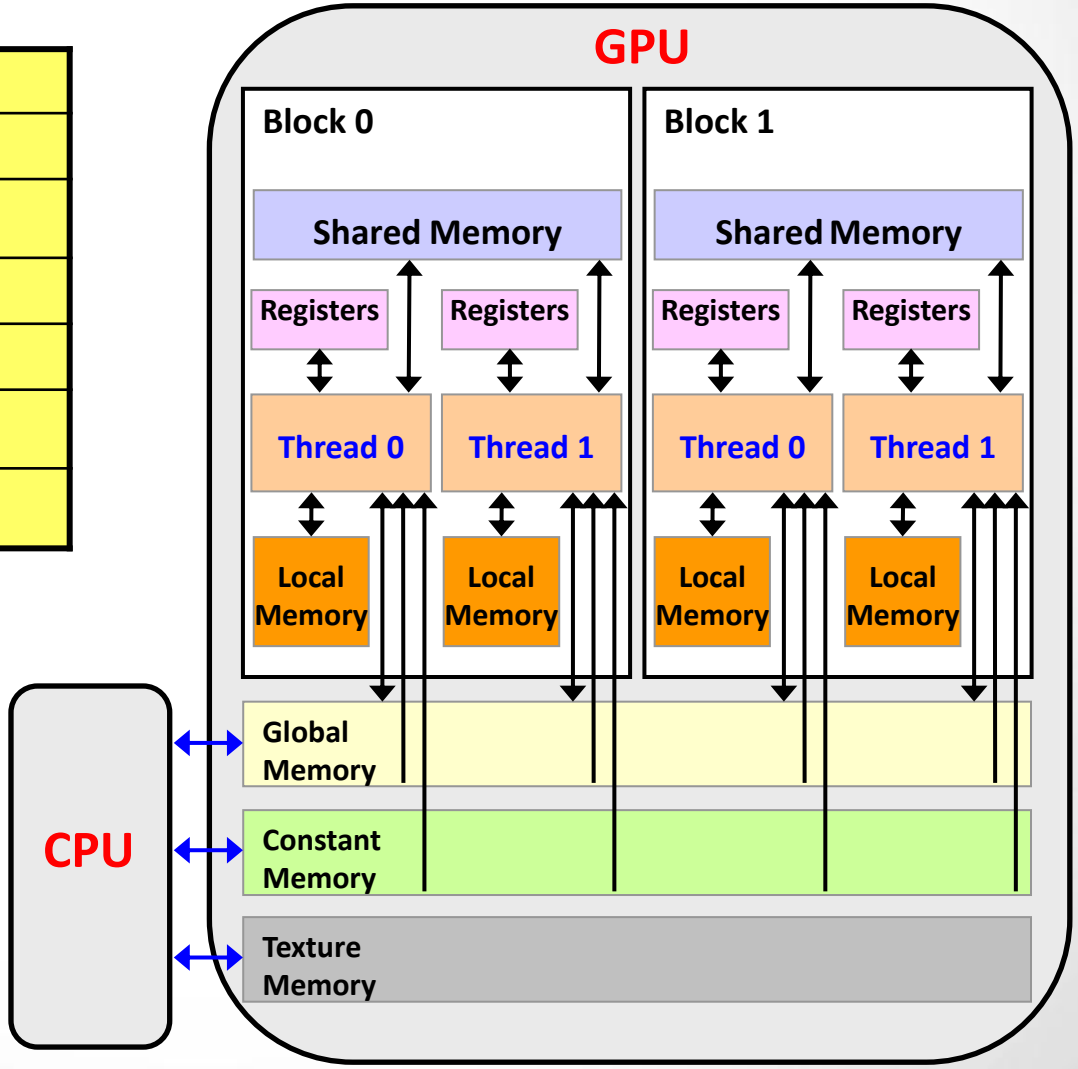


- GPU architectures are based on hyper-threading
- Single instruction executed on multiple threads (SIMD). Instructions are issued per warp (32 threads).
- A large number of threads are required to cover the memory access latency ...
  - ... an issue is to control the generation of threads
- Context switching ...
  - ... between warps when stalled (e.g. an operand is not ready)
  - ... enables to minimize stalls with little overhead

Hierarchy of memories

Memory type	Access latency	Size
Global	Medium	Big
Registers	Very fast	Very small
Local	Medium	Medium
Shared	Fast	Small
Constant	Fast (cached)	Medium
Texture	Fast (cached)	Medium

- Highly parallel multi-threaded many core
- High memory bandwidth compared to CPU
- Different levels of memory (different latencies)



Objective and challenging issues

- Re-think the parallel models of metaheuristics to take into account the characteristics of GPU
  - The focus on: iteration-level (MW) and algorithmic-level (PC)
  - Three major challenges …
- Challenge1: efficient CPU-GPU cooperation
  - Work partitioning between CPU and GPU, data transfer optimization
- Challenge2: efficient parallelism control
  - Threads generation control (memory constraints)
  - Efficient mapping between work units and threads Ids
- Challenge3: efficient memory management
  - Which data on which memory (latency and capacity constraints) ?

Outline

I. Scientific Context
   1. Parallel Metaheuristics
   2. GPU Computing

II. Contributions
   1. Efficient CPU-GPU **Cooperation**
   2. Efficient Parallelism **Control**
   3. Efficient Memory **Management**
   4. Extension of **ParadisEO** for GPU-based metaheuristics

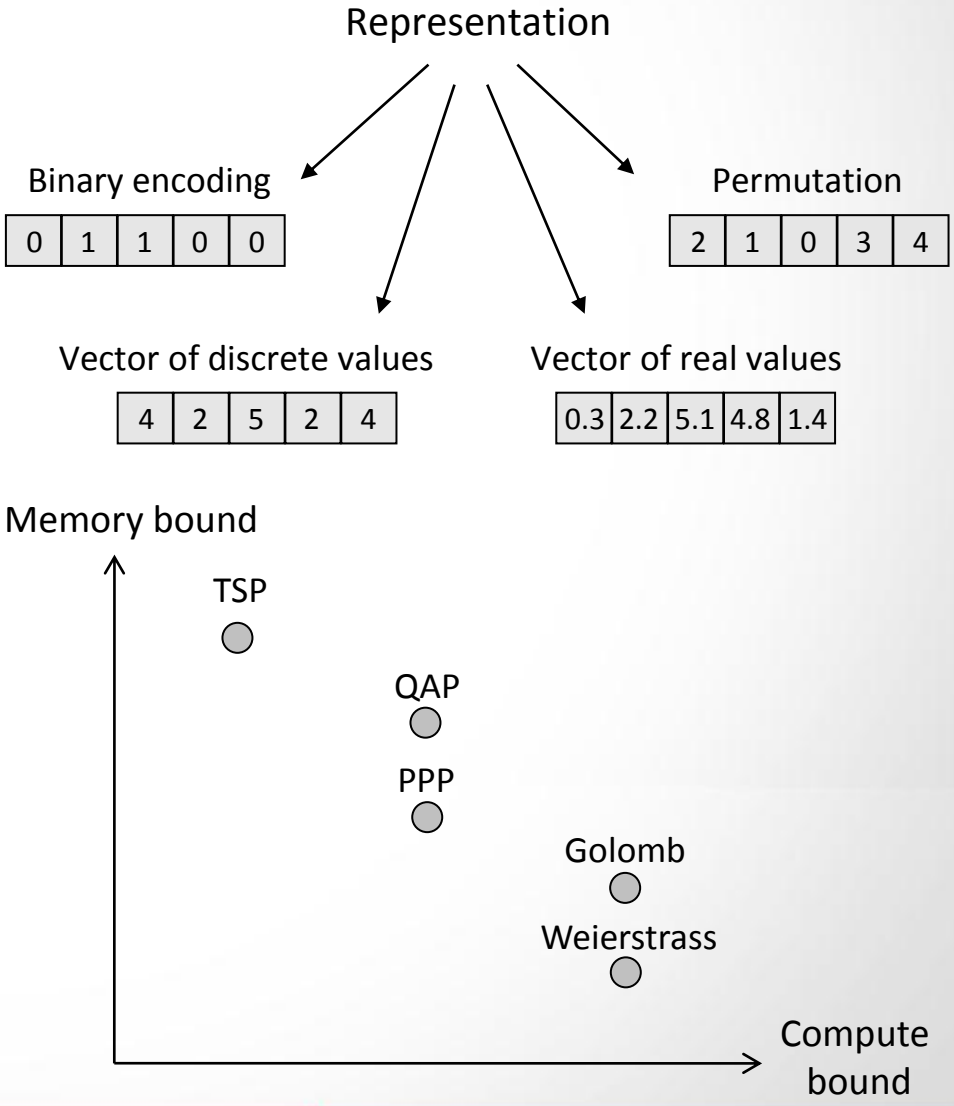
III. Conclusion and Future Works

Taxonomy of major works

Algorithms	CPU-GPU cooperation	Parallelism control	Memory management	38 works
Panmictic	Evaluation on GPU	One thread per individual	Global memory	8
2D toroidal Grid	Evaluation, full parallelization on GPU	One thread per individual	Global, shared and texture memory	10
Island model	Evaluation, full parallelization on GPU	One block per population	Global, shared and texture memory	4
Multi-start	Full parallelization on GPU	One thread per algorithm	Global and texture memory	3
Single solution-based	Generation and evaluation on GPU	One thread per neighbor	Global and texture memory	5
Hybrid	Generation and evaluation, full parallelization on GPU	One thread per individual / neighbor	Global and texture memory	6
Multiobjective optimization	Generation and evaluation on GPU	One thread per individual / neighbor	Global and texture memory	2

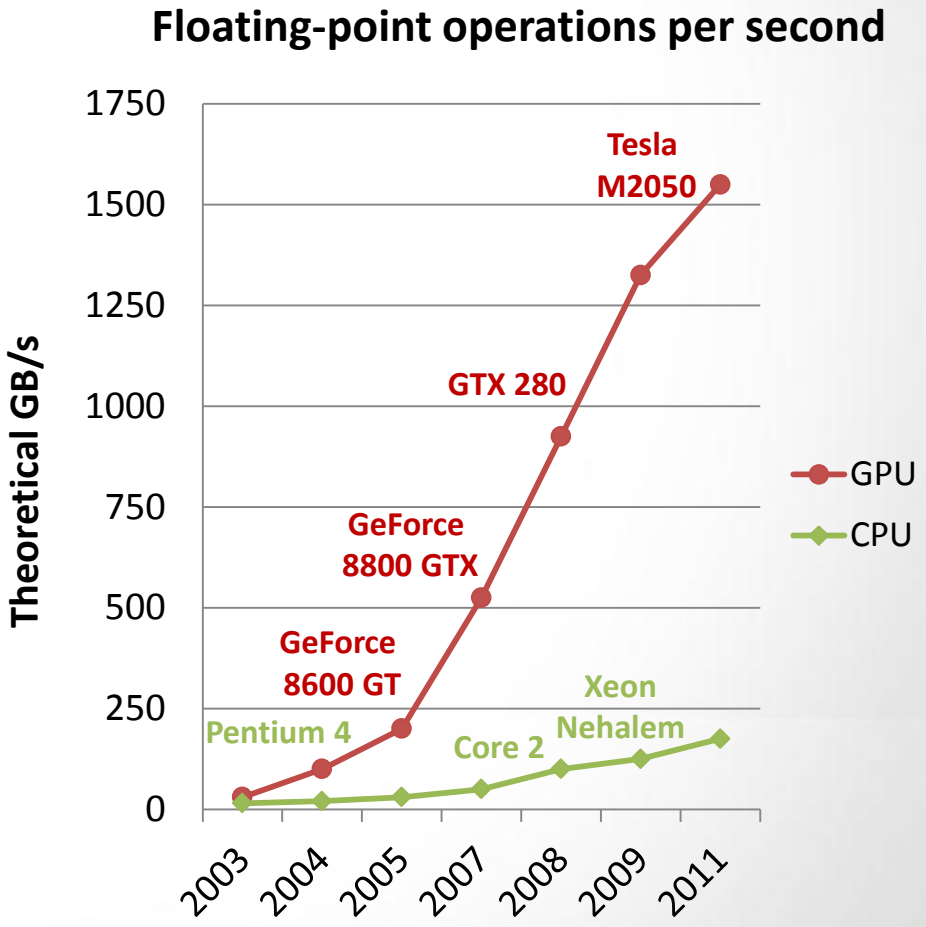
Optimization problems

- Permuted perceptron problem (PPP)
  - Cryptographic identification scheme
  - Binary encoding
- Quadratic assignment problem (QAP)
  - Facility location or data analysis
  - Permutation
- The Weierstrass continuous function
  - Simulation of fractal surfaces
  - Vector of real values
- Traveling salesman problem (TSP)
  - Planning and logistics
  - Permutation (large instances)
- The Golomb rulers
  - Interferometer for radio astronomy
  - Vector of discrete values



Hardware configurations

- Configuration 1: laptop
  Core 2 Duo 2 Ghz + 8600M GT
  (4 multiprocessors - 32 cores)
- Configuration 2: desktop
  Core 2 Quad 2.4 Ghz + 8800 GTX
  (16 multiprocessors - 128 cores)
- Configuration 3: workstation
  Intel Xeon 3 Ghz + GTX 280
  (30 multiprocessors - 240 cores)
- Configuration 4: workstation
  Intel Xeon 3.2 Ghz + Tesla M2050
  (14 multiprocessors - 448 cores)



1. Efficient CPU-GPU Cooperation

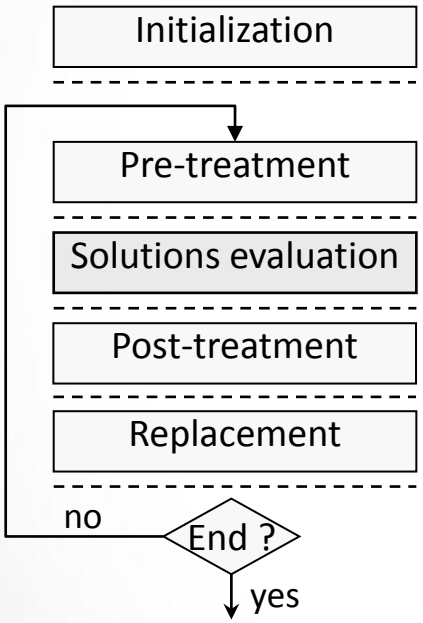
- Thé Van Luong, Nouredine Melab, El-Ghazali Talbi. **GPU Computing for Parallel Local Search Metaheuristic Algorithms.** IEEE Transactions on Computers, in press, 2011

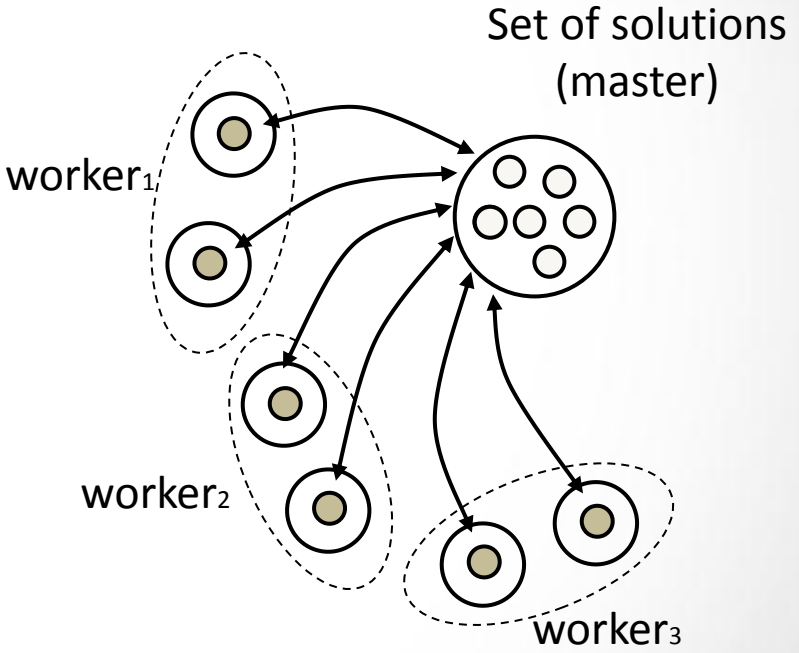
- Thé Van Luong, Nouredine Melab, El-Ghazali Talbi. **GPU-based Approaches for Multiobjective Local Search Algorithms. A Case Study: the Flowshop Scheduling Problem**. 11th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP), Torino, Italy, 2011

Objective and challenging issues

- Re-think the parallel models of metaheuristics to take into account the characteristics of GPU
  - The focus on: iteration-level (MW) and algorithmic-level(PC)
  - Three major challenges …
- **Challenge1: efficient CPU-GPU cooperation**
  - Work partitioning between CPU and GPU, data transfer optimization
- Challenge2: efficient parallelism control
  - Threads generation control (memory constraints)
  - Efficient mapping between work units and threads Ids
- Challenge3: efficient memory management
  - Which data on which memory (latency and capacity constraints) ?

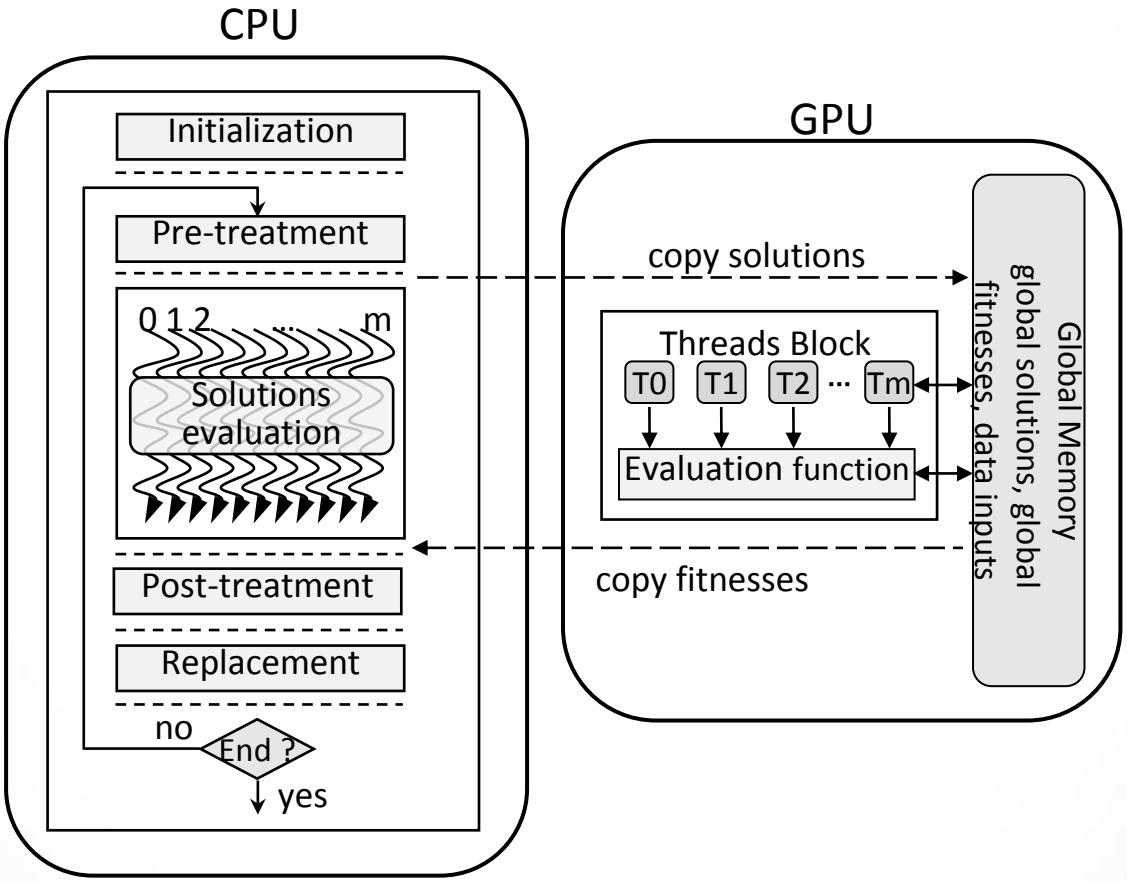
Iteration-level parallel model: MW





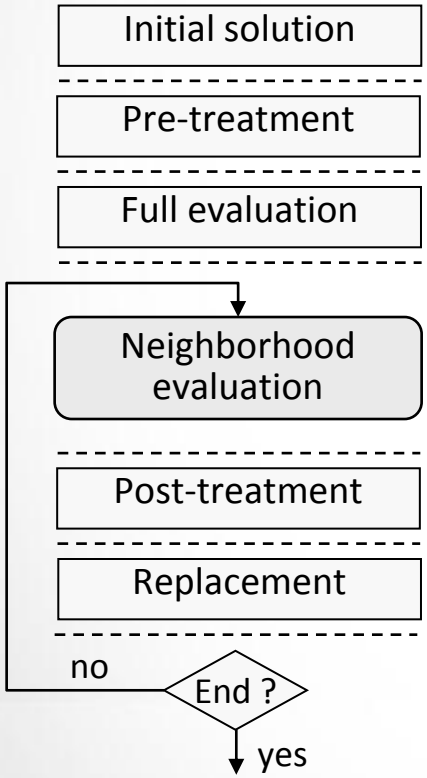
- … Need of **massively parallel computing** on very **large** solutions set

Work partitioning



- CPU (host) controls the whole sequential part of the metaheuristic
- GPU evaluates the solutions set in parallel

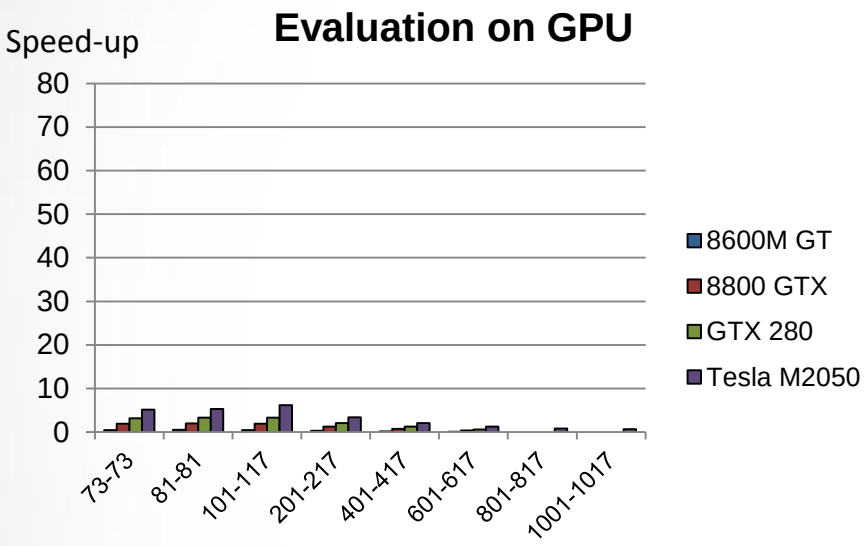
Optimize CPU–>GPU data transfer

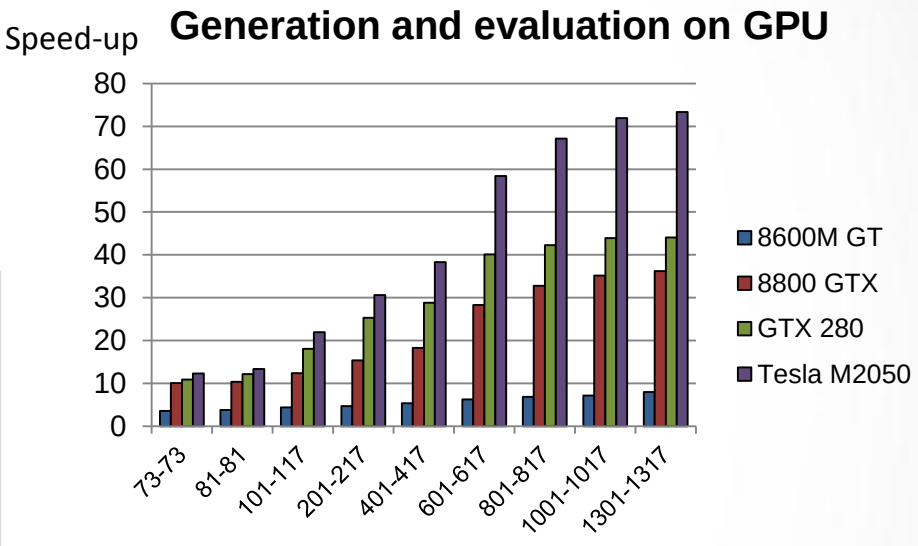


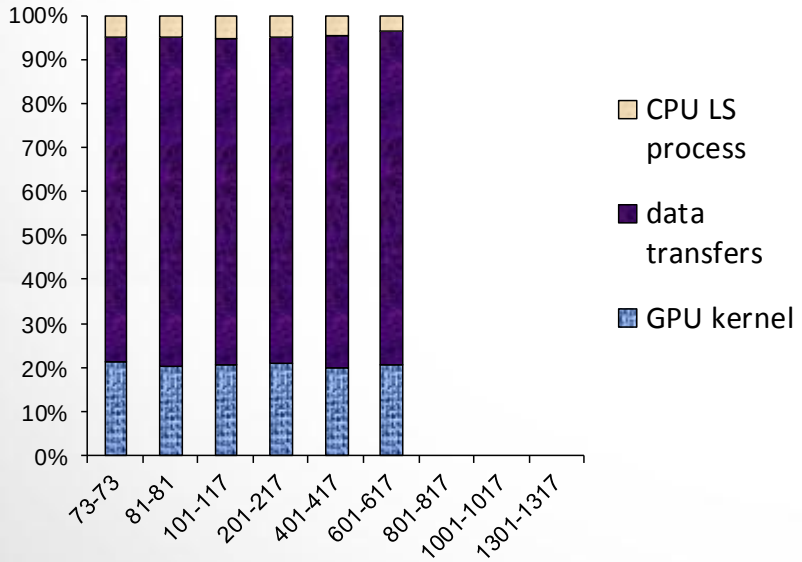
- Issue for S-metaheuristics
  - Where the neighborhood is generated ?
  - Two approaches:
    - **Approach1**: generation on CPU and evaluation on GPU
    - **Approach2**: generation and evaluation on GPU (parallelism control)

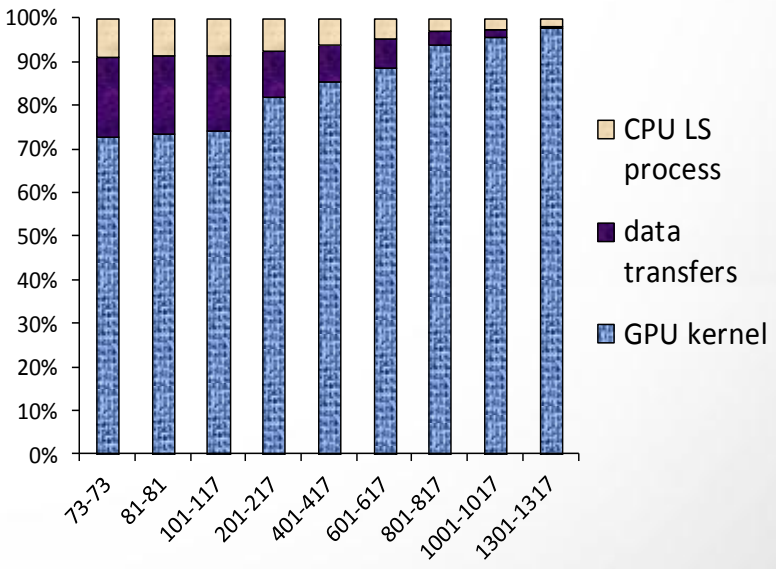
- Tabu search - Hamming distance of two
  - Where the neighborhood is generated ?
  - Two approaches: n(n-1)/2 neighbors
    - **Approach1**: additional $O(n^3)$ transfers
    - **Approach2**: additional O(n) transfers

Application to the PPP

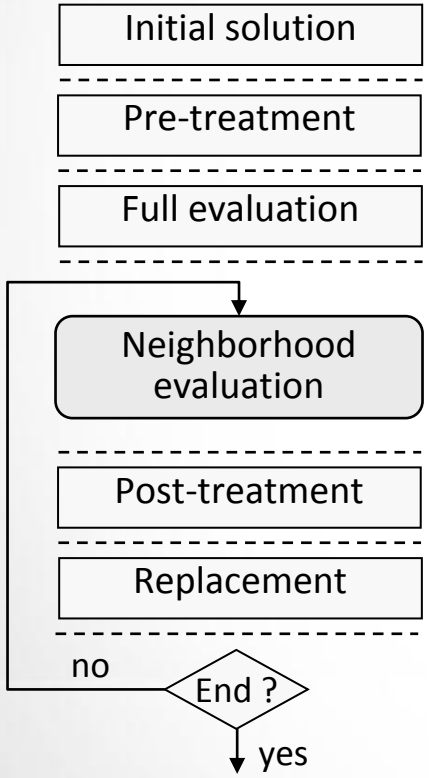








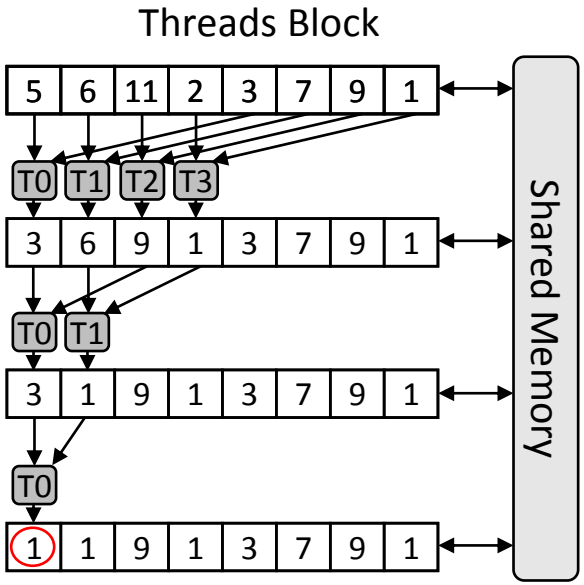
Optimize GPU–>CPU data transfer



- Issue for S-metaheuristics
  - Where the selection of the best neighbor is done?
  - Two approaches:
    - **Approach1**: on CPU i.e. transfer of the data structure storing the solution results
    - **Approach2**: on GPU i.e. use of the reduction operation to select the best solution

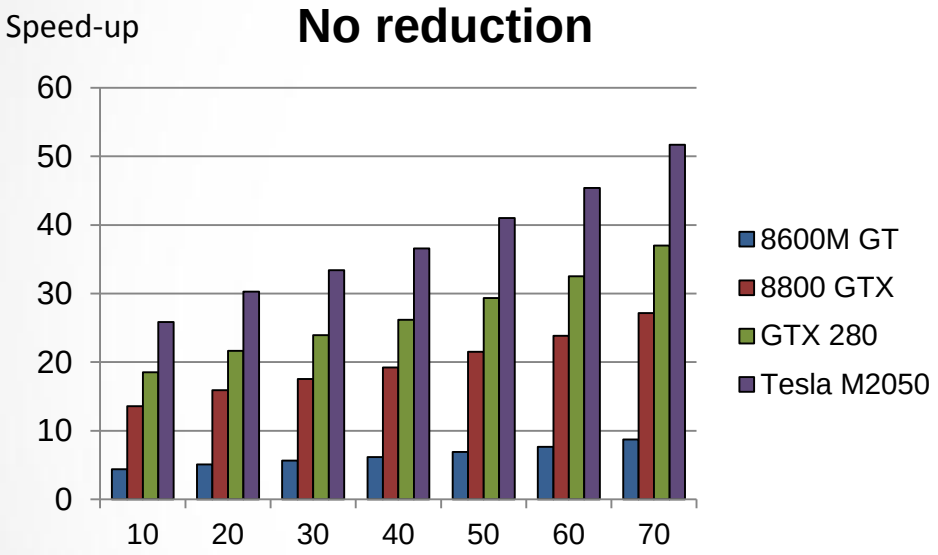
- Hill climbing - Hamming distance of two
  - Two approaches: n(n-1)/2 neighbors
    - **Approach1**: additional $O(n^2)$ transfers
    - **Approach2**: additional O(1) transfer

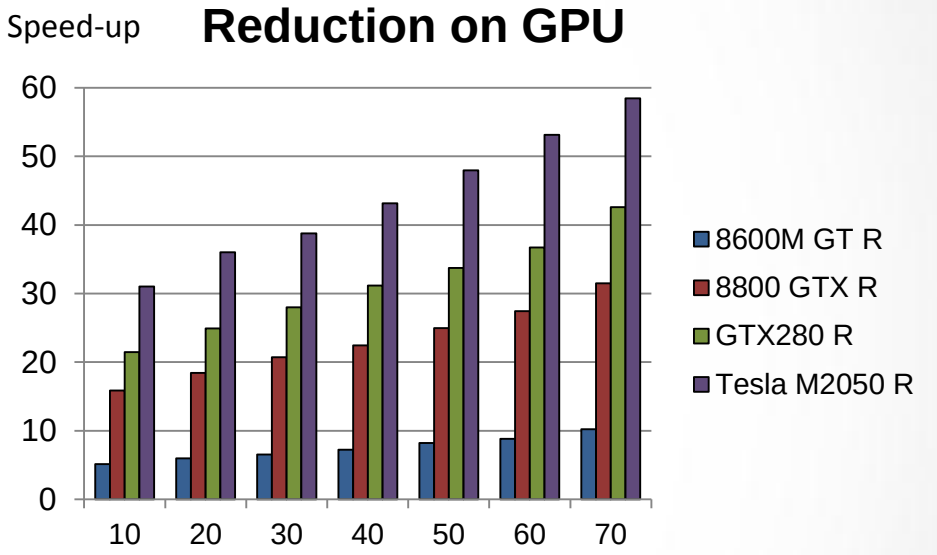
GPU reduction to select the best solution

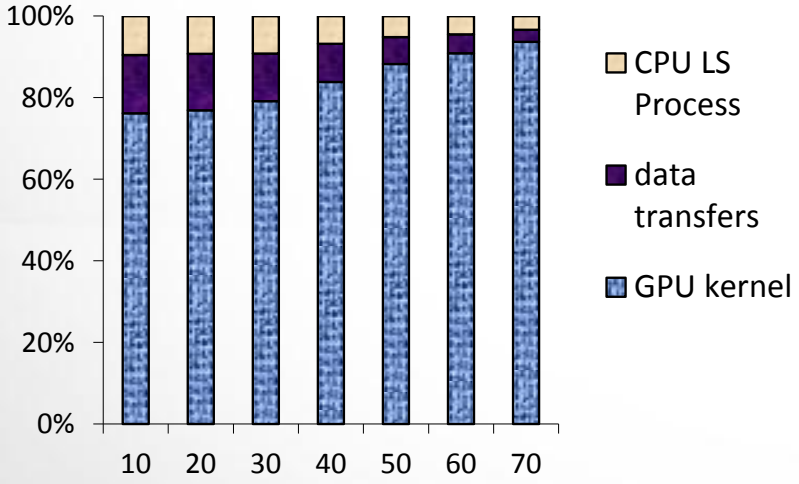


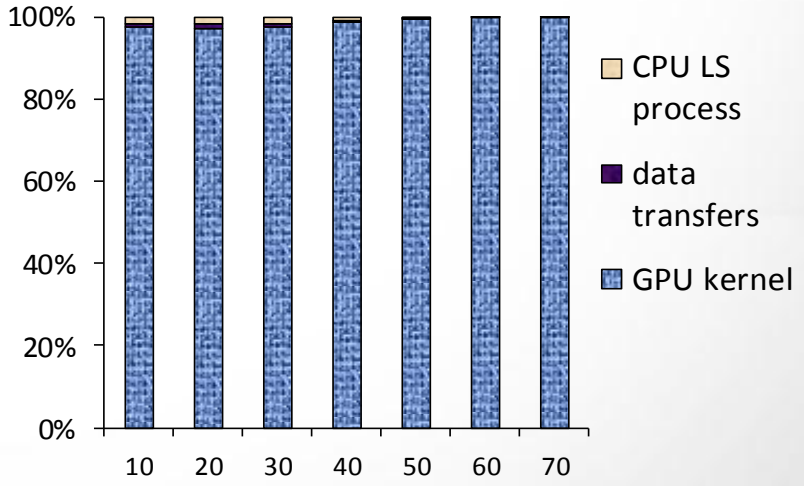
- **Binary tree-based reduction mechanism** to find the minimum of each block of threads
- Cooperation of threads of a same block through the shared memory (latency: ~10 cycles)
- Performing iterations on reduction kernels allows to find the minimum of all neighbors
- Complexity: $O(\log_2(n))$, n: size of the neighborhood

Application to the Golomb ruler







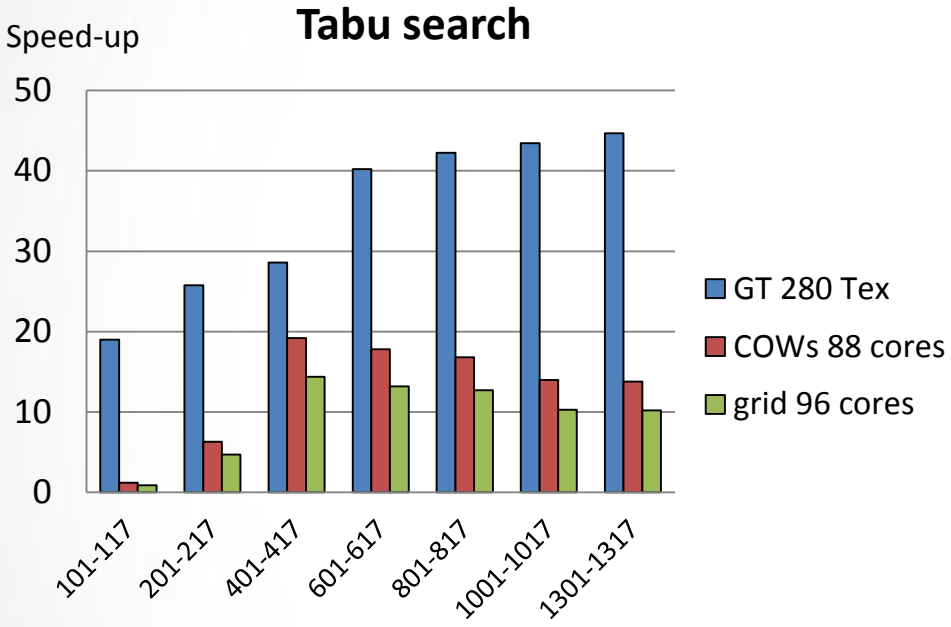


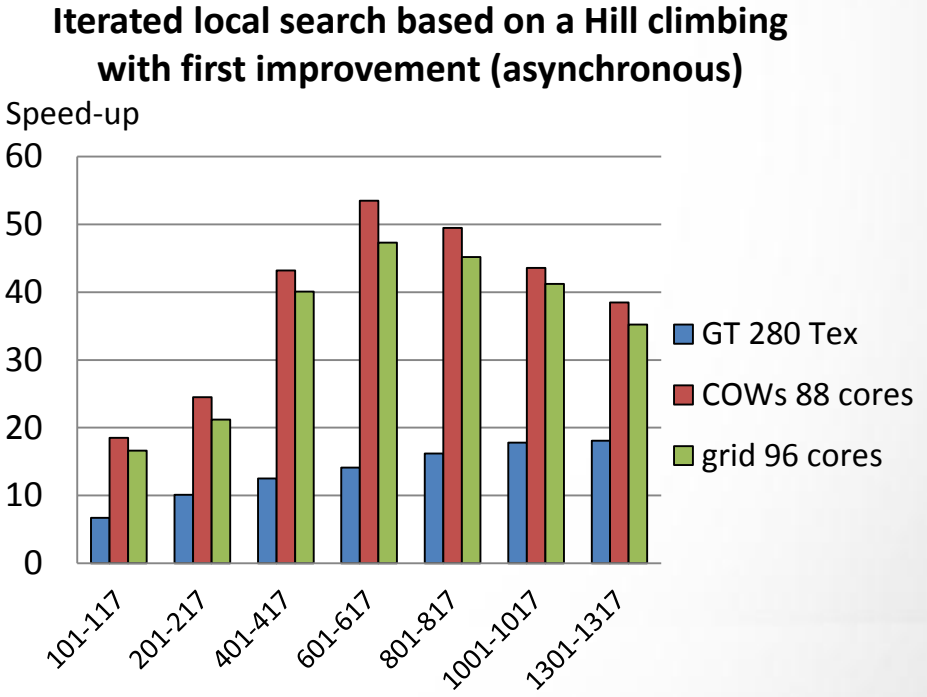
Comparison with other parallel architectures

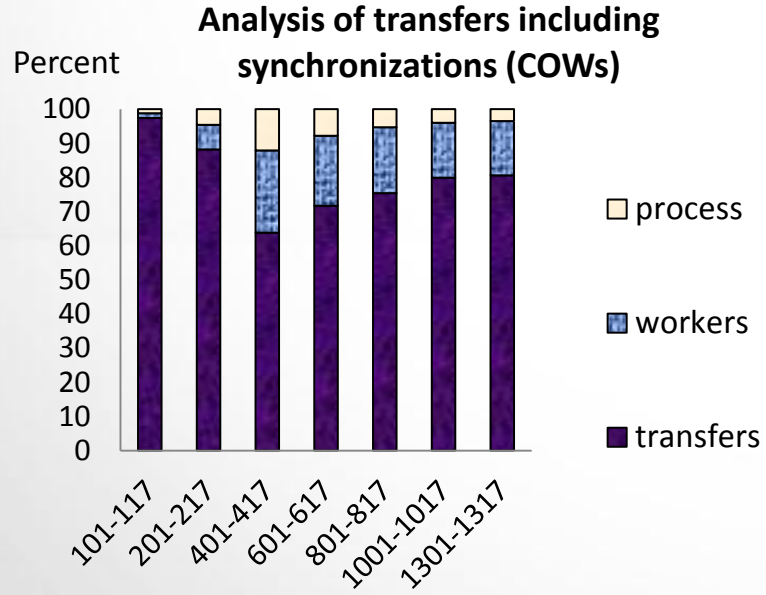
- Emergence of heterogeneous COWs and computational grids as standard platforms for high-performance computing.
- Application to the permuted perceptron problem
- Hybrid OpenMP/MPI implementation

Architecture	Configuration 3	
	Machines	gflops
GPU	Intel Xeon E5450 GeForce GTX 280	981.12
COWs	11 Intel Xeon E5440 88 CPU cores	995.236
Grid	2 Intel Xeon E5520 2 AMD Opteron 2218 2 Intel Xeon E5520 4 Intel Xeon E5520 Intel Xeon X5570 Intel Xeon E5520 96 CPU cores	979.104

Application to the PPP







2. Efficient Parallelism Control

- Thé Van Luong, Nouredine Melab, El-Ghazali Talbi. **Large Neighborhood Local Search Optimization on Graphics Processing Units.** 23rd IEEE International Parallel & Distributed Processing Symposium (IPDPS), Workshop on Large-Scale Parallel Processing (LSPP), Atlanta, US, 2010

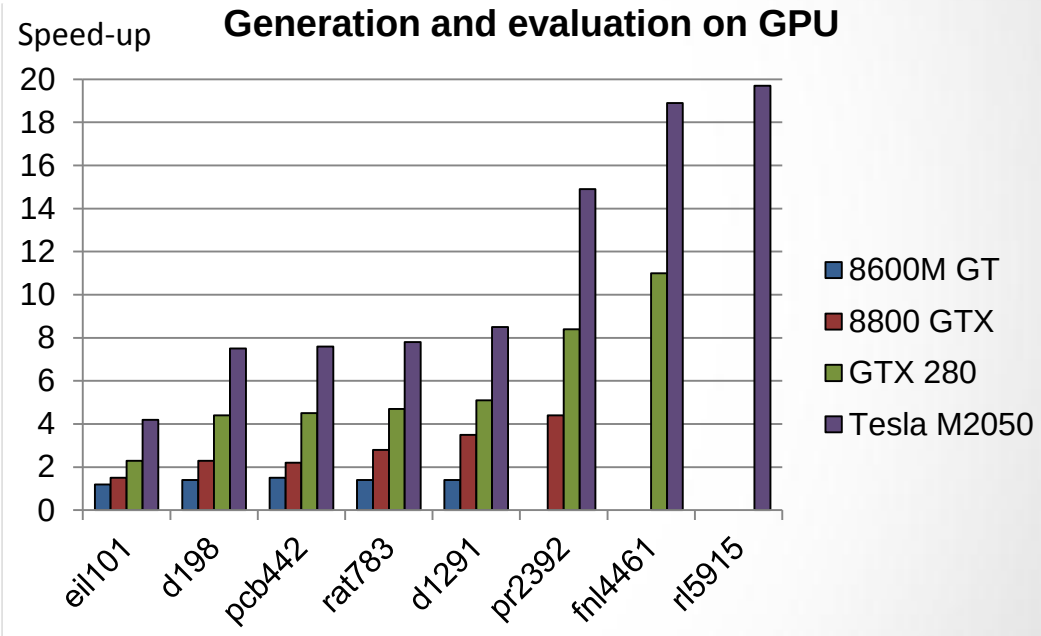
- Thé Van Luong, Nouredine Melab, El-Ghazali Talbi. **Local Search Algorithms on Graphics Processing Units. A Case Study: the Permutation Perceptron Problem**. 10th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP), Istanbul, Turkey, 2010

Objective and challenging issues

- Re-think the parallel models of metaheuristics to take into account the characteristics of GPU
  - The focus on: iteration-level (MW) and algorithmic-level (PC)
  - Three major challenges …
- Challenge1: efficient CPU-GPU cooperation
  - Work partitioning between CPU and GPU, data transfer optimization
- **Challenge2: efficient parallelism control**
  - Threads generation control (memory constraints)
  - Efficient mapping between work units and threads Ids
- Challenge3: efficient memory management
  - Which data on which memory (latency and capacity constraints) ?

Thread control (1)

- Traveling salesman problem
  - Large instances failed at execution time due to memory overflow (e.g. hardware register limitation or max number of threads exceeded)
  - Such errors are hard to predict at compilation time since they are specific to a configuration



- Need a thread control for the generation of threads to meet the memory constraints at execution time …

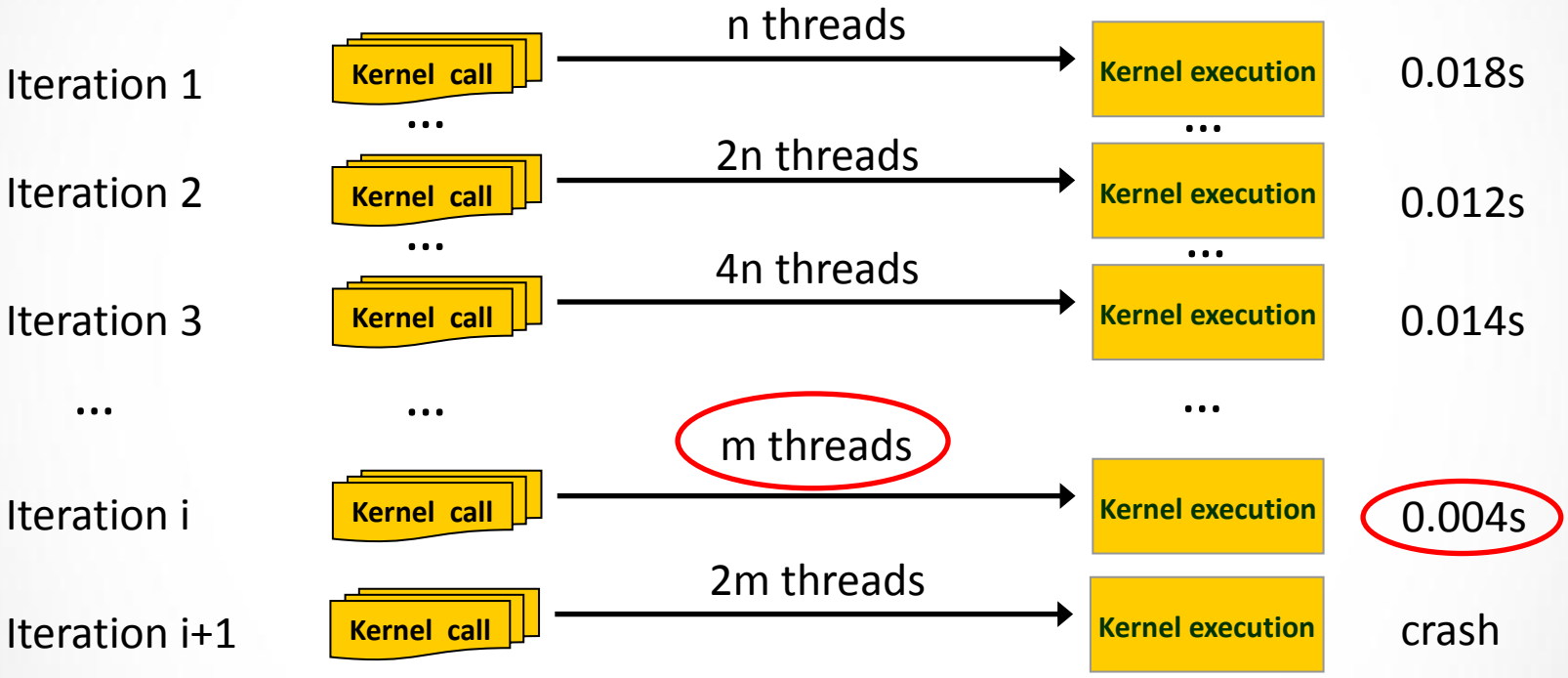
Thread control (2)

Solution set

1	2	3	4	5	6	7	8	9	10	11	12
17	4	3	10	6	11	12	10	7	13	8	21
13	14	15	16	17	18	19	20	21	22	23	24
11	5	8	19	12	19	13	11	13	14	19	29
25	26	27	28	29	30	31	32	33	34	35	36
14	13	16	13	4	15	11	19	6	18	5	24

- Increase the threads granularity …
  - … with associating each thread to MANY solutions
  - … to avoid memory overflow (e.g. hardware register limitation)

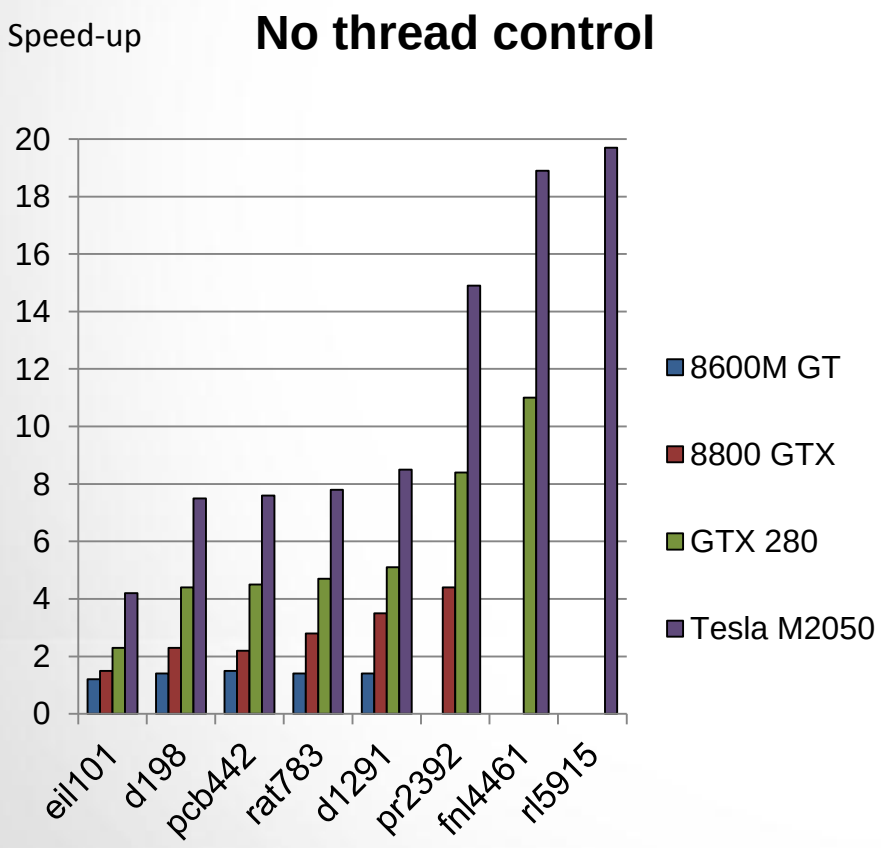
Thread control (3)

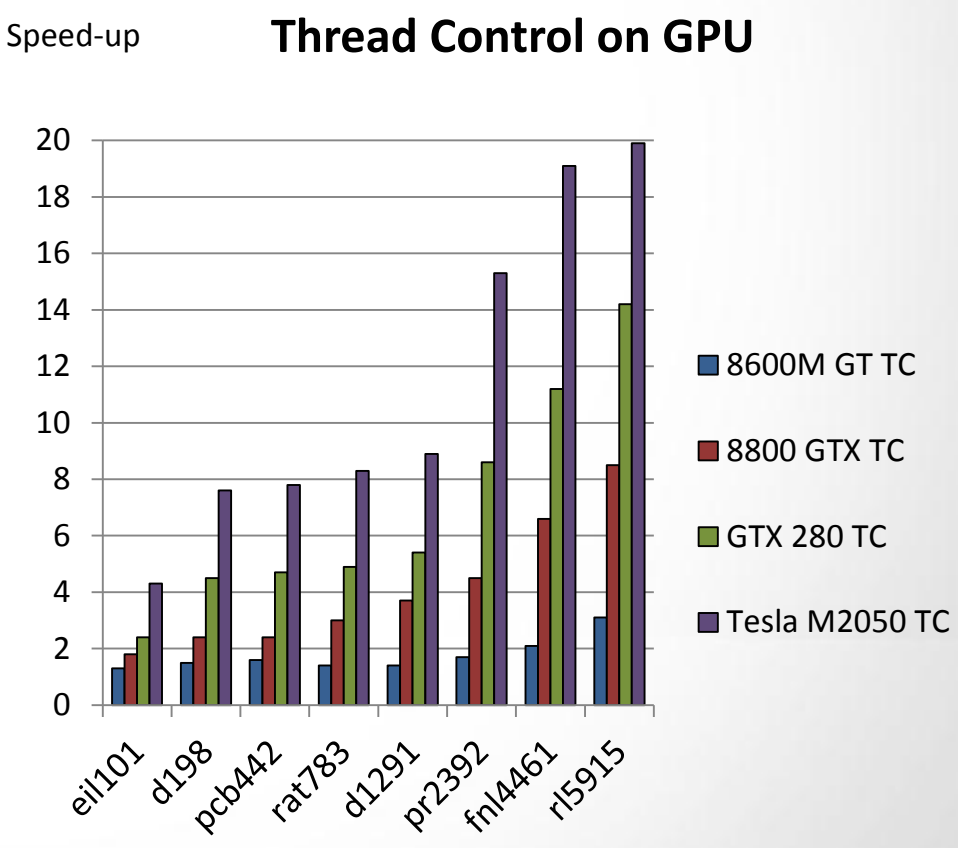


- Dynamic heuristic for parameters auto-tuning
  - To prevent the program from crashing
  - To obtain extra performance

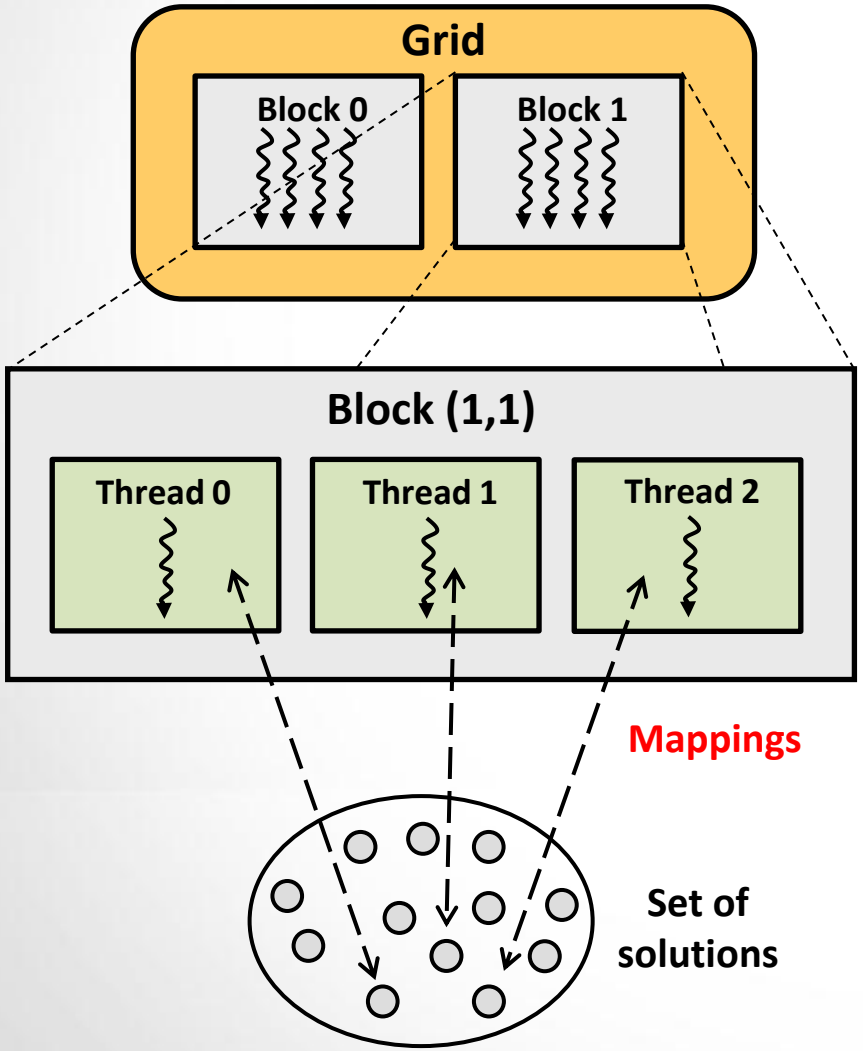
Thread control (4)

- Application to the traveling salesman problem (tabu search)





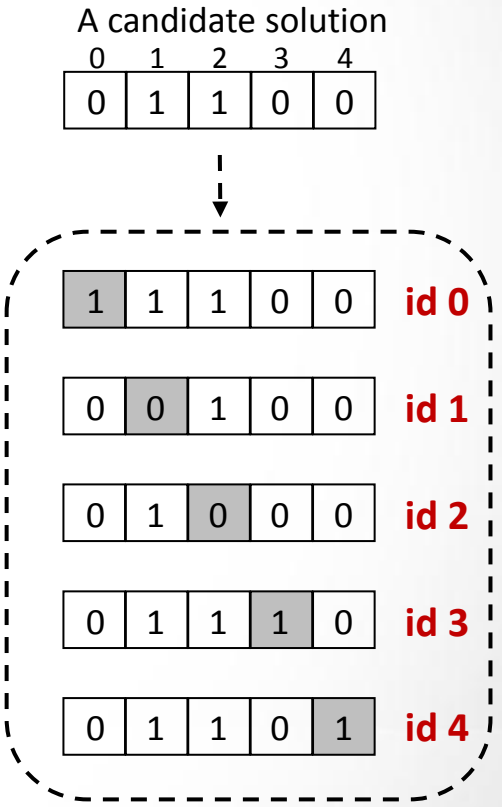
Mapping working unit -> thread id (1)



- According to the threads spatial organization …
  - … a **unique id** must be assigned to each thread to compute on different data
- For S-metaheuristics, the challenging issue is to say …
  - … which neighbor is assigned to which thread *id* (required for the generation of the neighborhood on GPU)
- Representation-dependent

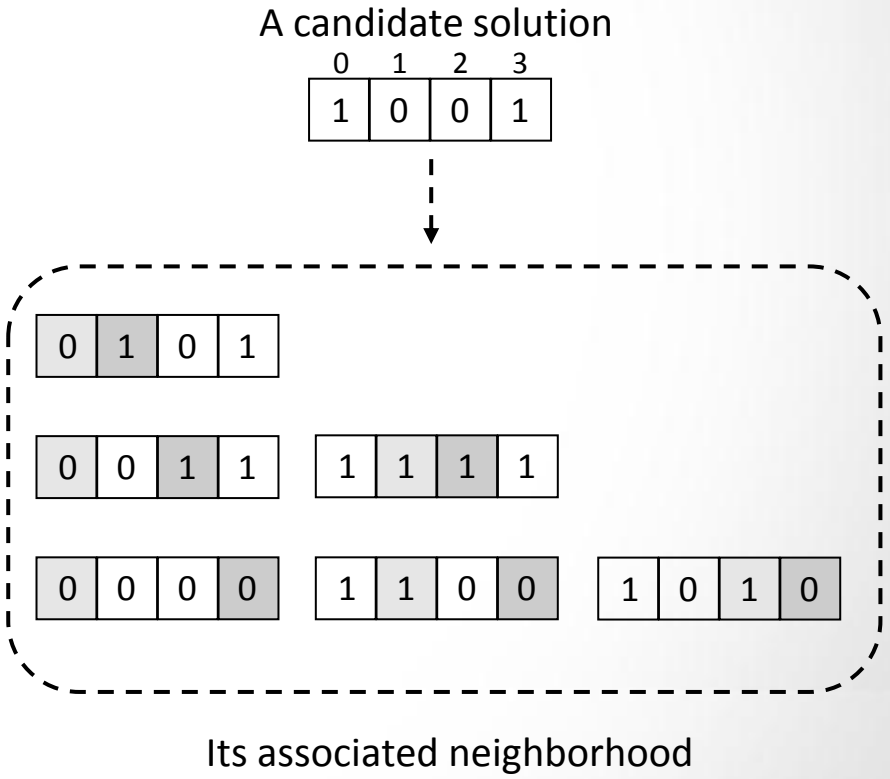
Mapping working unit -> thread id (2)

- Mappings are proposed for 4 well-known representations (binary, discrete, permutation, real vector)
- Neighborhood based on a Hamming distance of one
  - The thread with *id=i* generates and evaluates a candidate solution by flipping the *bit number i* of the initial solution
  - *At most, n* threads are generated for a solution of size *n*



Mapping working unit -> thread id (3)

- Finding a mapping can be challenging
- Neighborhood based on a Hamming distance of two
  - A thread *id* is associated with two indexes *i* and *j*
  - *At most, n x (n-1) / 2* threads are generated for a solution of size *n*



GPU computing for large neighborhoods

- The increase of the neighborhood size may improve the quality of the obtained solutions [Ahuja *et al.* 2007] …
- … but mostly CPU-time consuming. This mechanism is not often fully exploited in practice.
- Large neighborhoods are unusable because of their high computational cost …
  - … GPU computing might allow to exploit parallelism in such algorithms.
- Application to the permuted perceptron problem (configuration 3: GTX 280)

Problem	73 x 73	81 x 81	101 x 101	101 x 117
Fitness	9.8	11.3	20.7	16.8
# iterations	59891	72345	166650	260130
# solutions	11/50	4/50	0/50	0/50
CPU time	4 s	6 s	16 s	29 s
GPU time	9 s	13 s	33 s	57 s
Acceleration	x 0.44	x 0.46	x 0.48	x 0.51

Neighborhood based on a Hamming distance of one

Tabu search
n x (n-1) x (n-2) / 6 iterations

Problem	73 x 73	81 x 81	101 x 101	101 x 117
Fitness	15.5	16.2	13.1	12.7
# iterations	42143	65421	133211	260130
# solutions	22/50	17/50	13/50	0/50
CPU time	81 s	174 s	748 s	1947 s
GPU time	10 s	16 s	44 s	105 s
Acceleration	x 8.2	x 11.0	x 17.0	x 18.5

Neighborhood based on a Hamming distance of two

Tabu search
n x (n-1) x (n-2) / 6 iterations

Problem	73 x 73	81 x 81	101 x 101	101 x 117
Fitness	2.5	3.2	5.8	7.1
# iterations	19341	40636	100113	214092
# solutions	39/50	33/50	22/50	**3/50**
CPU time	1202 s	3730 s	*24657 s*	*88151 s*
GPU time	50 s	146 s	955 s	3551 s
Acceleration	x 24.2	x 25.5	x 25.8	x 26.3

Neighborhood based on a Hamming distance of three

Tabu search
n x (n-1) x (n-2) / 6 iterations

3. Efficient Memory Management

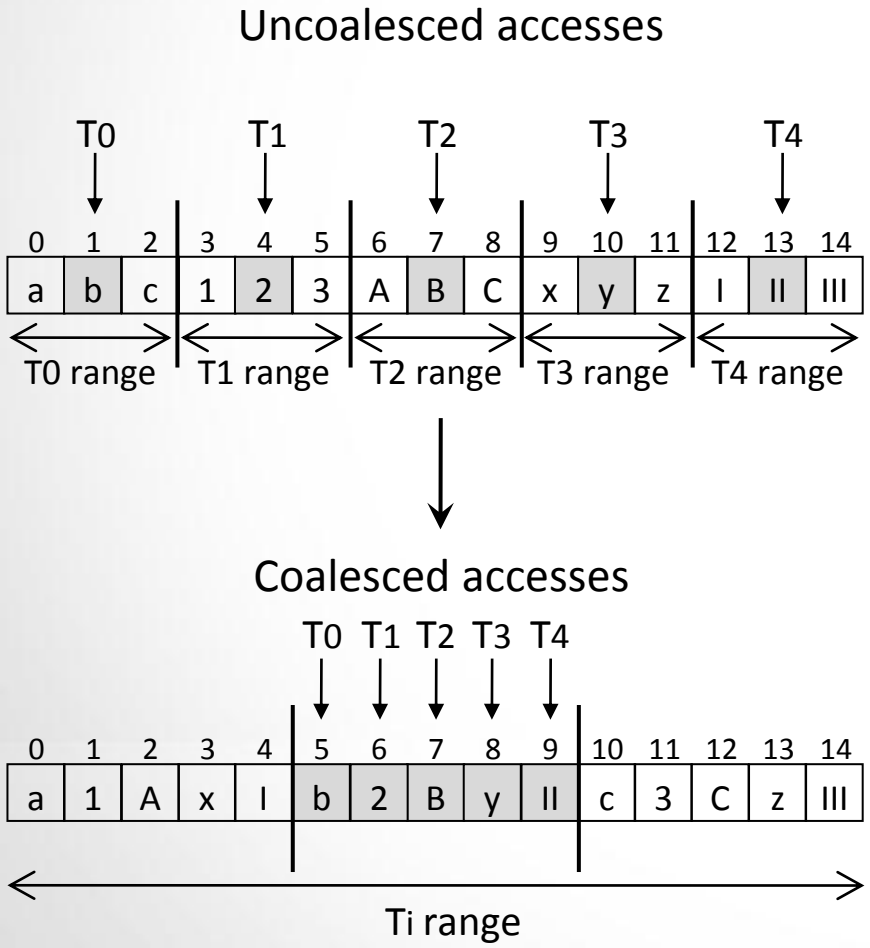
- Thé Van Luong, Nouredine Melab, El-Ghazali Talbi. **GPU-based Island Model for Evolutionary Algorithms.** Genetic and Evolutionary Computation Conference (GECCO), Portland, US, 2010

- Thé Van Luong, Nouredine Melab, El-Ghazali Talbi. **GPU-based Parallel Hybrid Evolutionary Algorithms.** IEEE Congress on Evolutionary Computation (CEC), Barcelona, Spain, 2010

Objective and challenging issues

- Re-think the parallel models of metaheuristics to take into account the characteristics of GPU
  - The focus on: iteration-level (MW) and algorithmic-level (PC)
  - Three major challenges …
- Challenge1: efficient CPU-GPU cooperation
  - Work partitioning between CPU and GPU, data transfer optimization
- Challenge2: efficient parallelism control
  - Threads generation control (memory constraints)
  - Efficient mapping between work units and threads Ids
- **Challenge3: efficient memory management**
  - Which data on which memory (latency and capacity constraints) ?

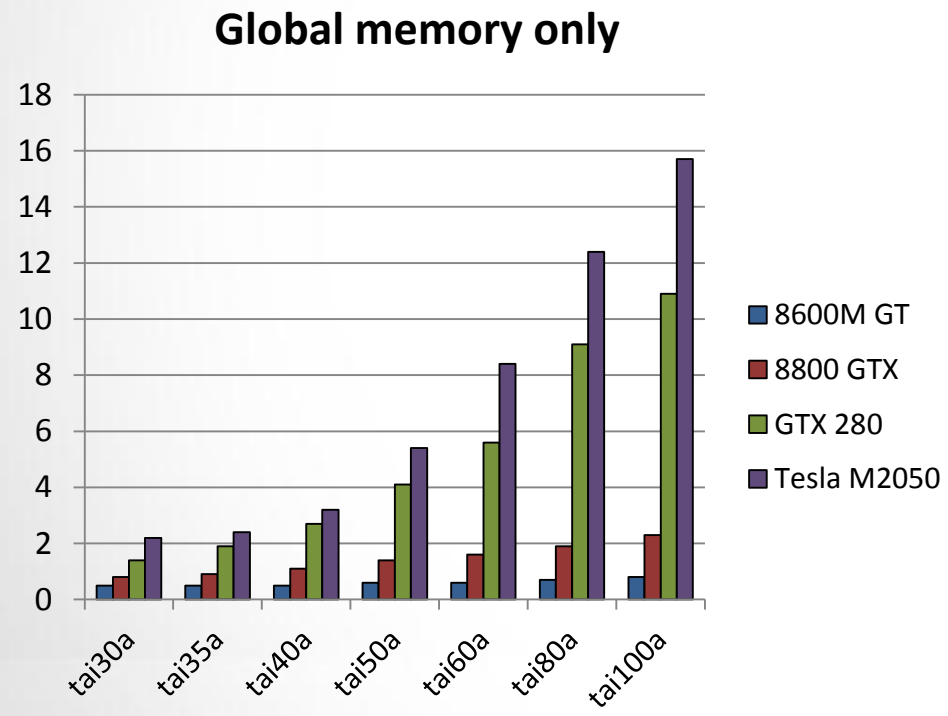
Memory coalescing and texture memory

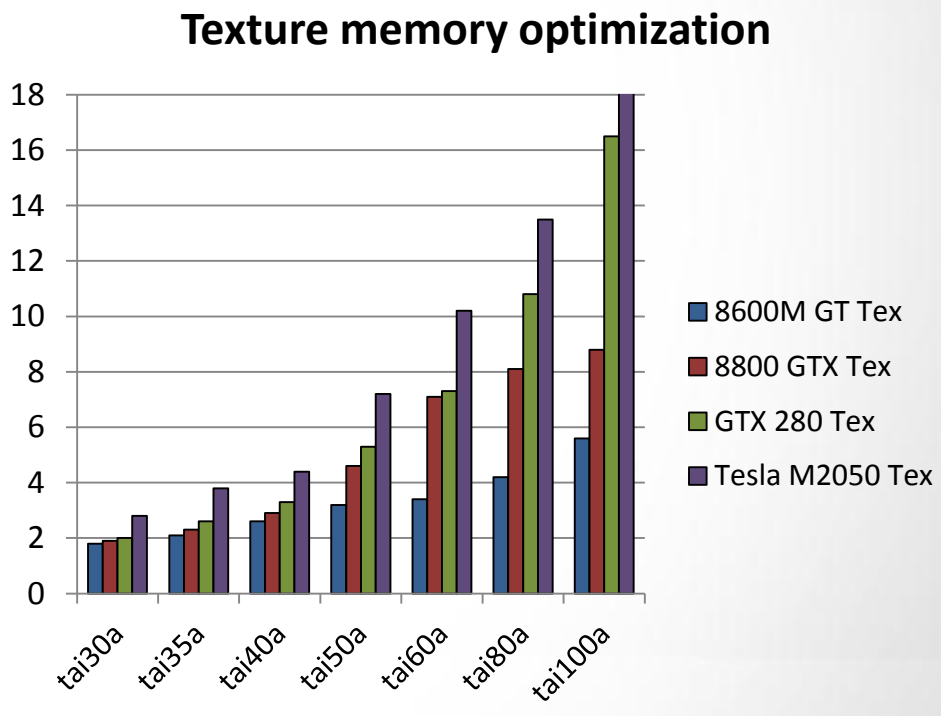


- Memory coalescing is not always feasible for structures in optimization problems …
- … use of texture memory as a data cache
  - Frequent reuse of data accesses in evaluation functions
  - 1D/2D access patterns in optimization problems (e.g. matrices or vectors)

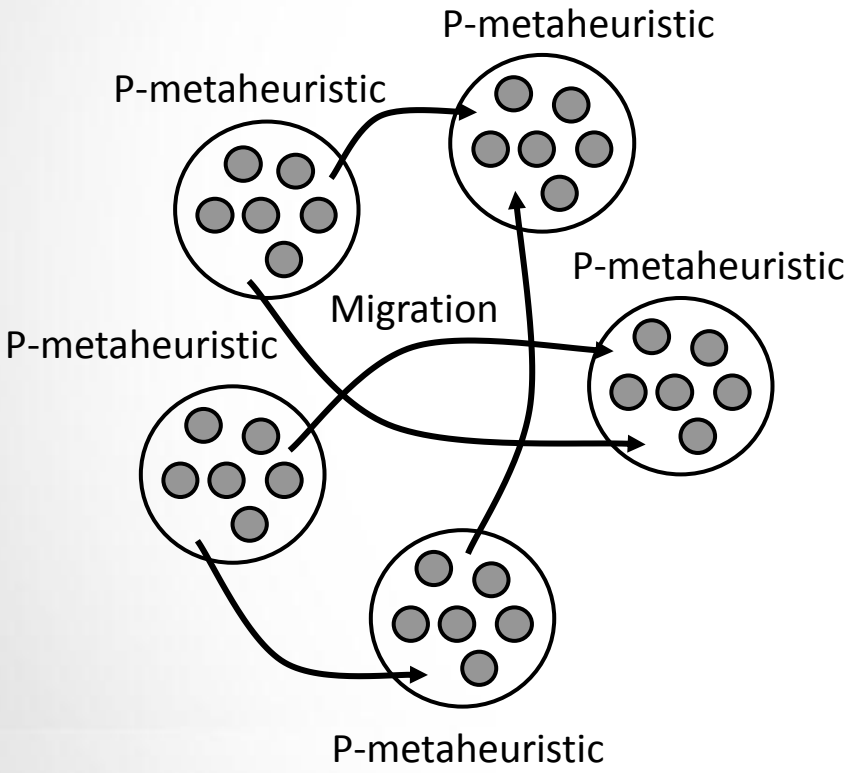
Application to the QAP

- Iterated local search with a tabu search



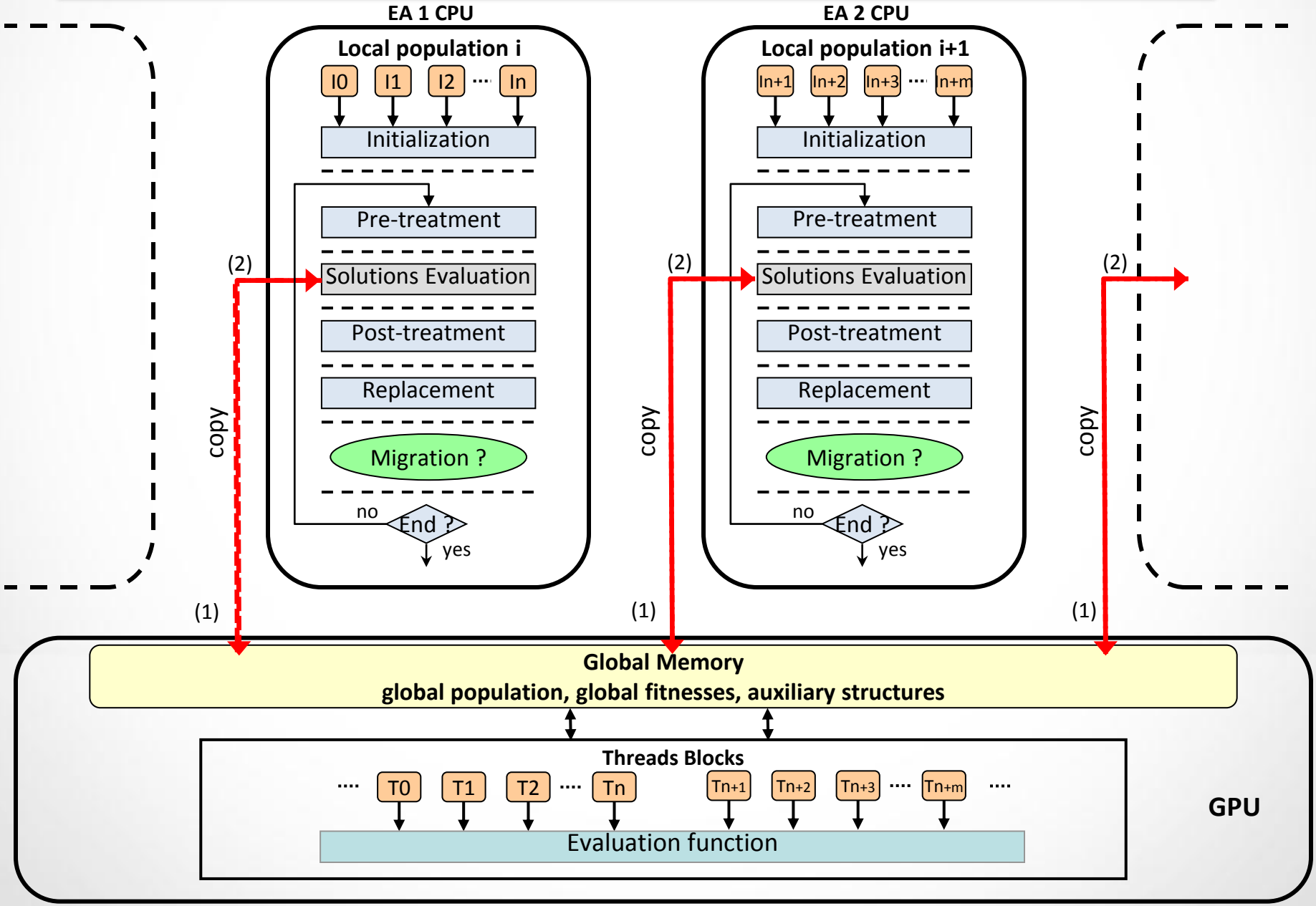


Algorithmic-level model: PC

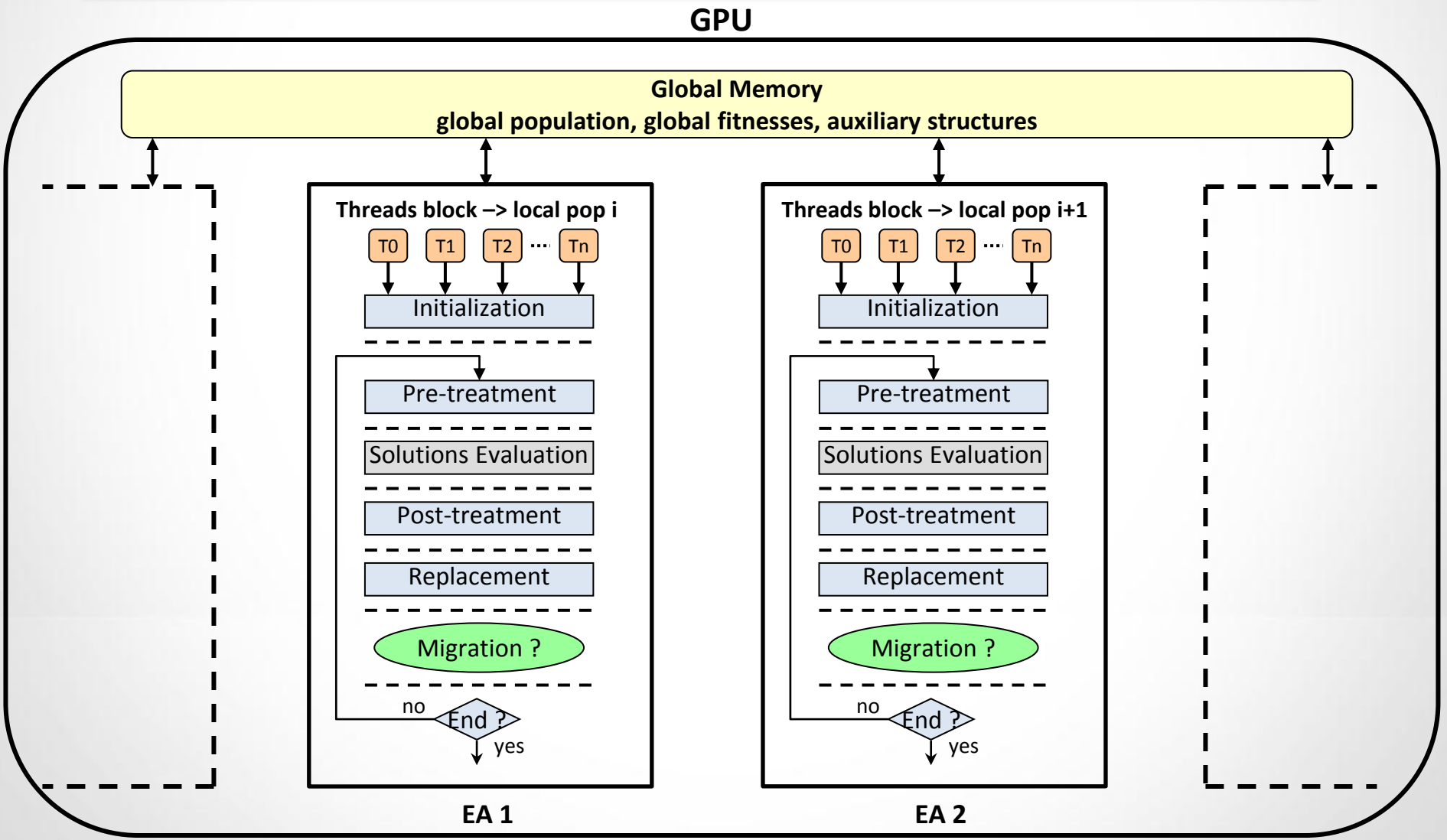


- PC model:
  - Emigrants selection policy
  - Replacement/Integration policy
  - Migration decision criterion
  - Exchange topology

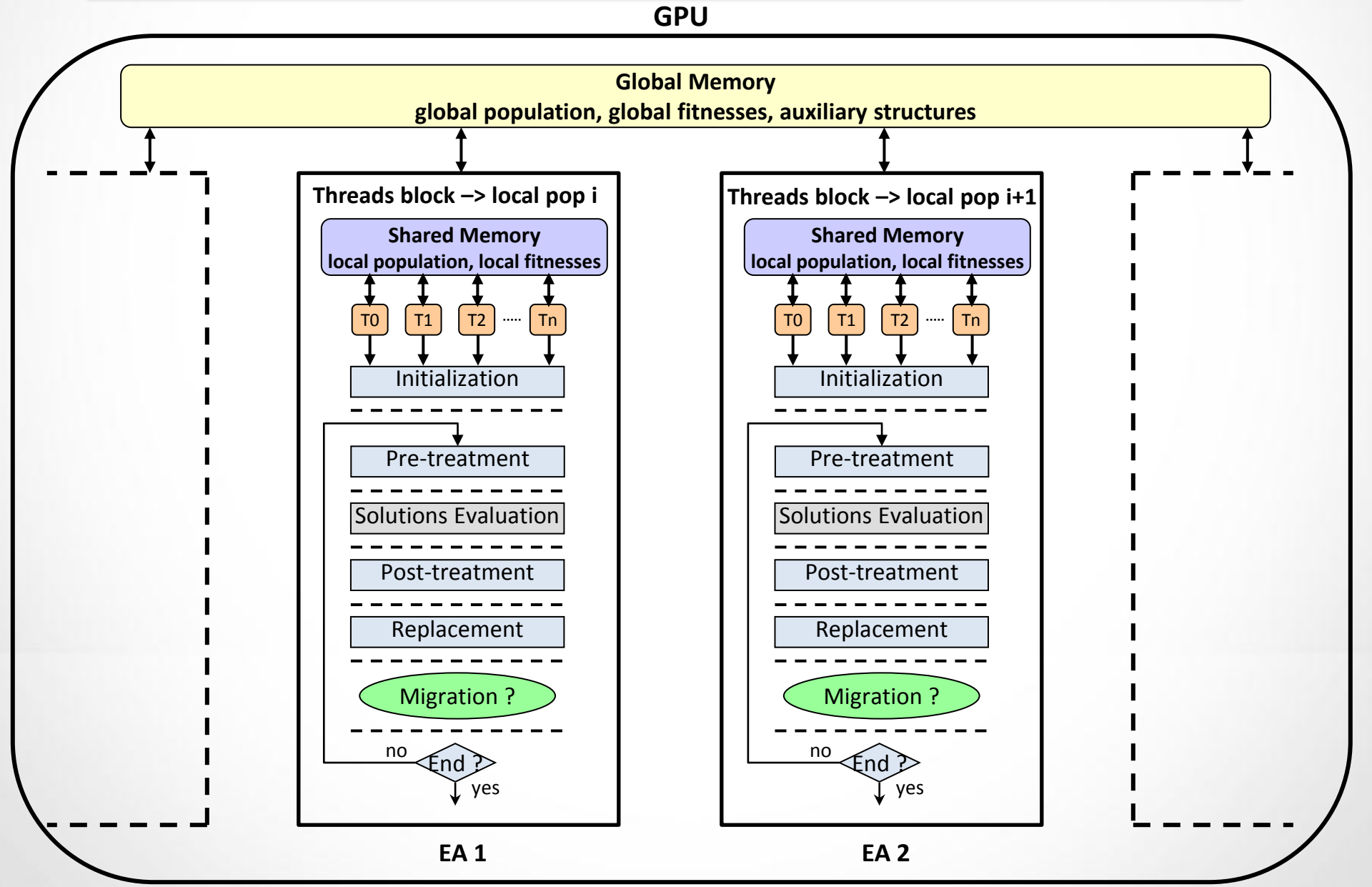
Scheme 1: Parallel evaluation of the population



Scheme 2: Full distribution on GPU



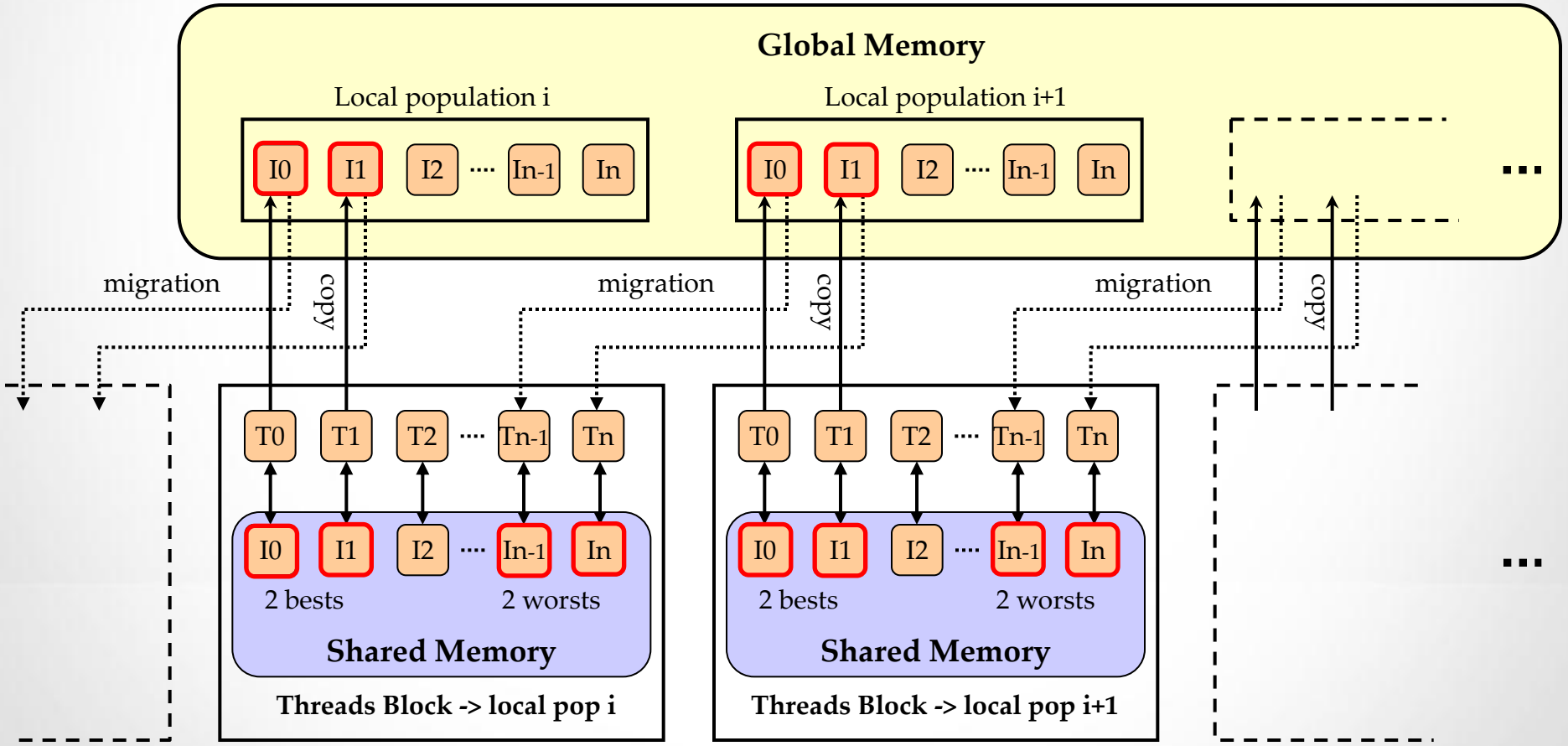
Scheme 3: Full distribution on GPU using shared memory



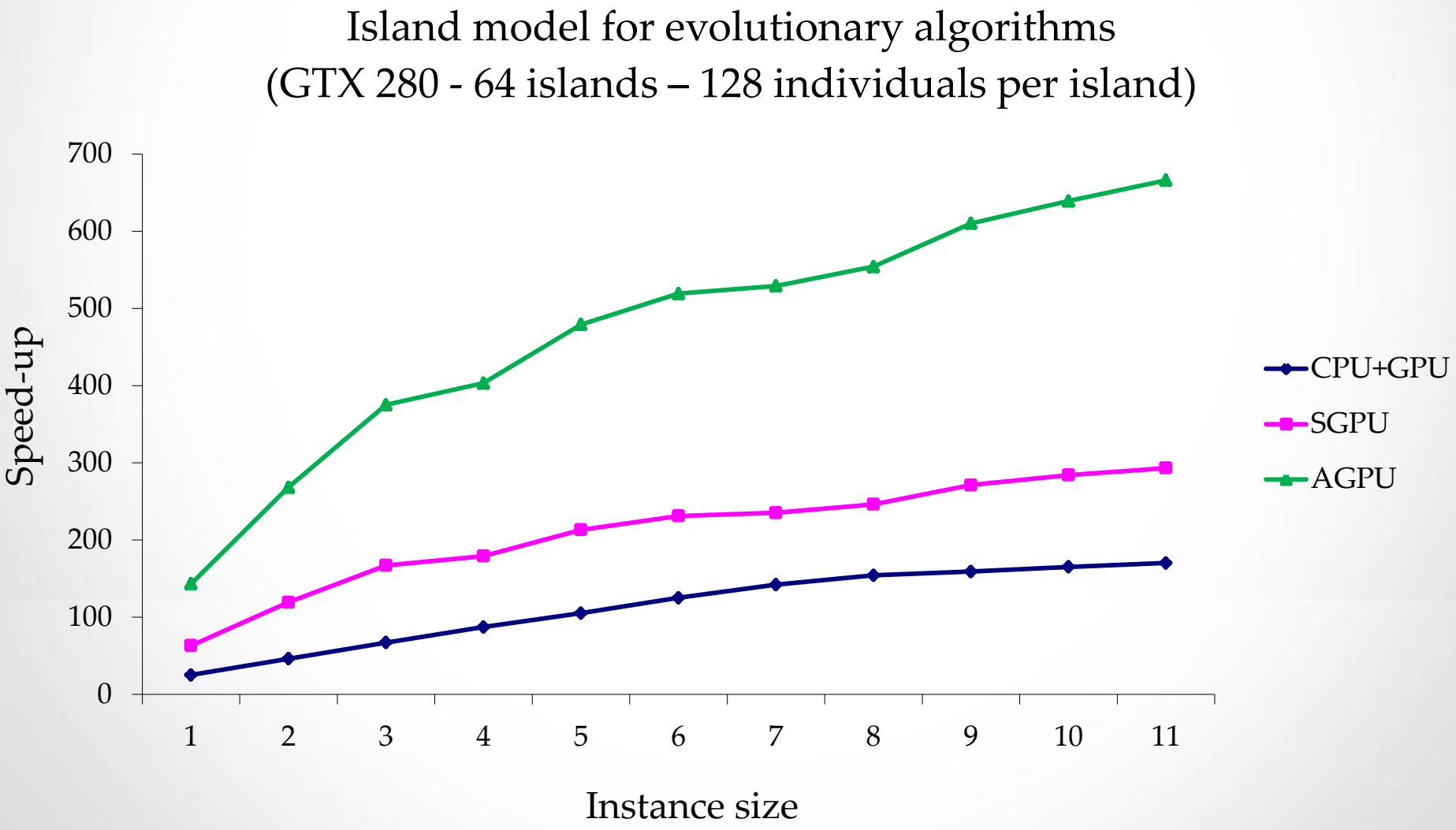
Issues of distributed schemes

- Sort each local population on GPU (bitonic sort)
- Find the minimum of each local population on GPU (parallel reduction)
- Local threads synchronization for interacting solutions
- Mechanisms of global synchronization of threads if a synchronous migration is needed
- Find efficient topologies between the different local populations according to the threads block organization

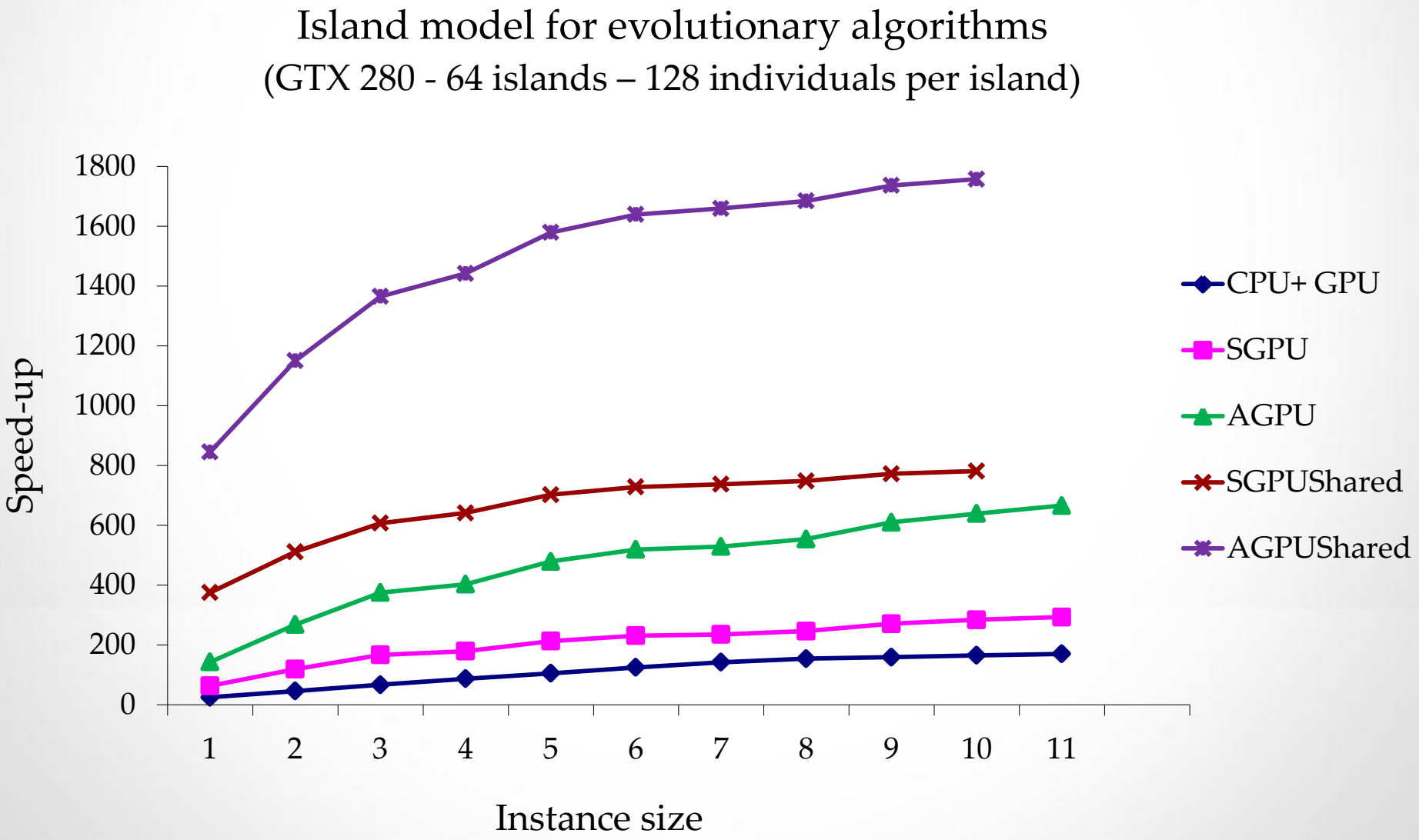
Migration on GPU for distributed schemes



Application to the Weierstrass function (1)



Application to the Weierstrass function (2)



4. Extension of ParadisEO for GPU-based metaheuristics

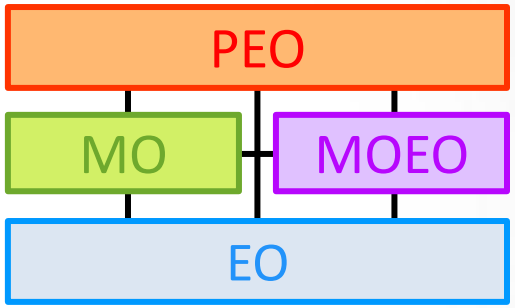
- Nouredine Melab, Thé Van Luong, Karima Boufaras, El-Ghazali Talbi. **Towards ParadisEO-MO-GPU: a Framework for GPU-based Local Search Metaheuristics**. 11th International Work-Conference on Artificial Neural Networks, IWANN 2011, Torremolinos-Málaga, Spain, 2011

- Thé Van Luong, Nouredine Melab, El-Ghazali Talbi. **Neighborhood Structures for GPU-based Local Search Algorithms.** Parallel Processing Letters, Vol. 20, No. 4, pp. 307-324, December 2010

Software framework

Paradiseo

- EO: Design and implementation of population-based metaheuristics
- MO: Design and implementation of solution-based metaheuristics
- MOEO: Design and implementation of multi-objective metaheuristics
- PEO: Design and implementation of parallel models for metaheuristics



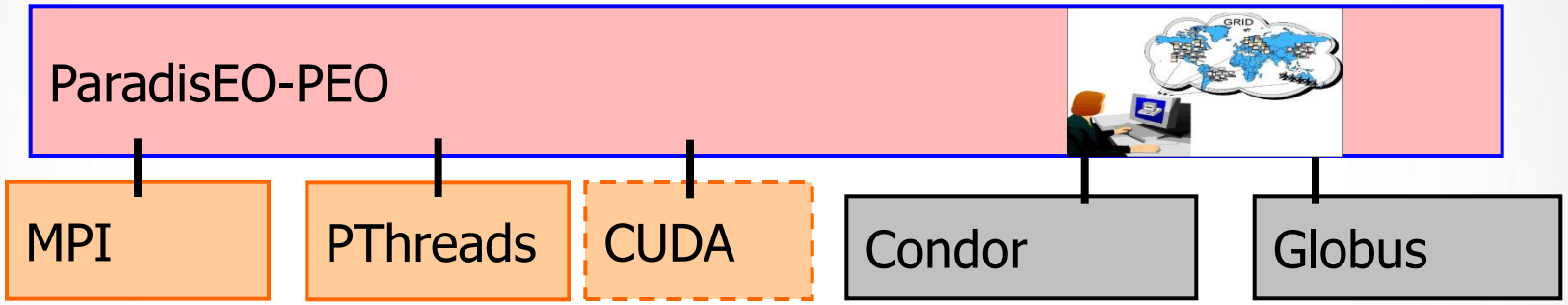
- 14871 downloads
- 121544 visitors
- 239 user-list subscribers

➢ Conceptual objectives

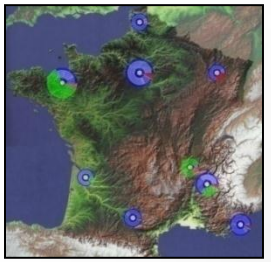
- Clear separation between resolution methods and problems at hand, maximum code reuse, flexibility, large panels of methods and portability

S. Cahon, N. Melab and E-G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. Journal of Heuristics, Vol.10(3), ISSN:1381-1231, pages 357-380, May 2004.

Parallel and distributed deployment

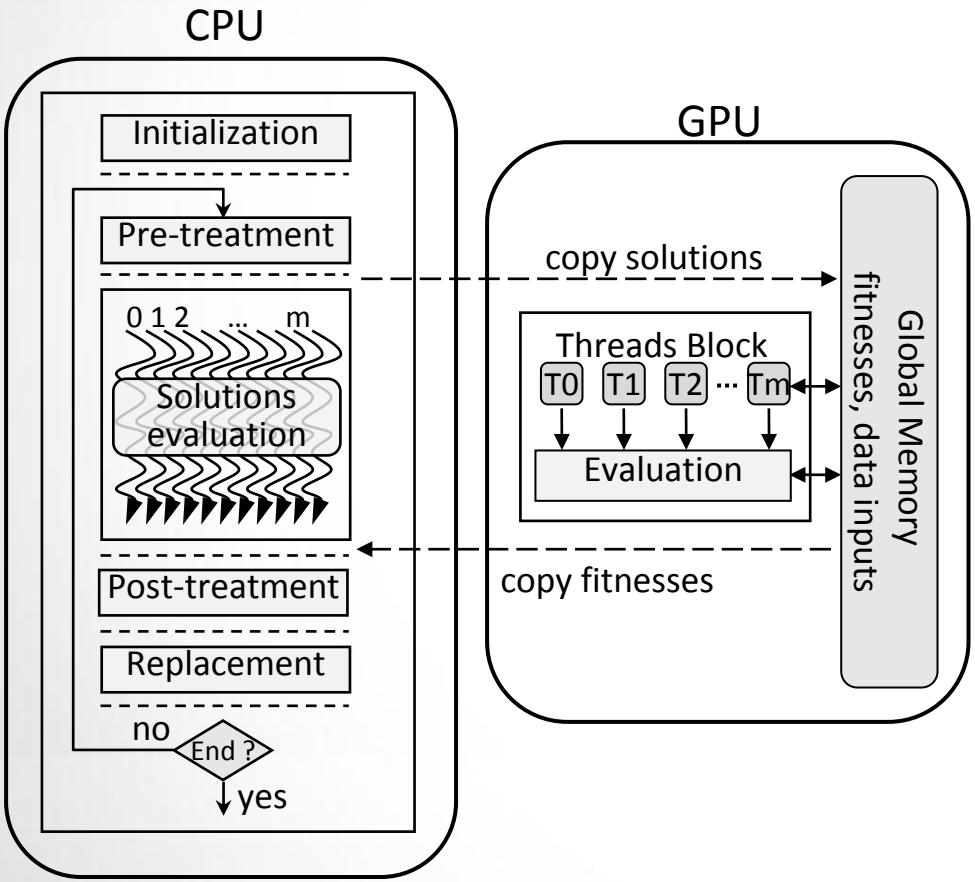


- **Transparent parallelization and distribution**
  - Clusters and networks of workstations: Communication library MPI
  - SMP and Multi-core: Multi-threading Pthreads
  - Grid computing:
    - High-performance Grids: Globus, MPICH-G
    - Desktop Grids: Condor (checkpointing & Recovery)
- GPU computing



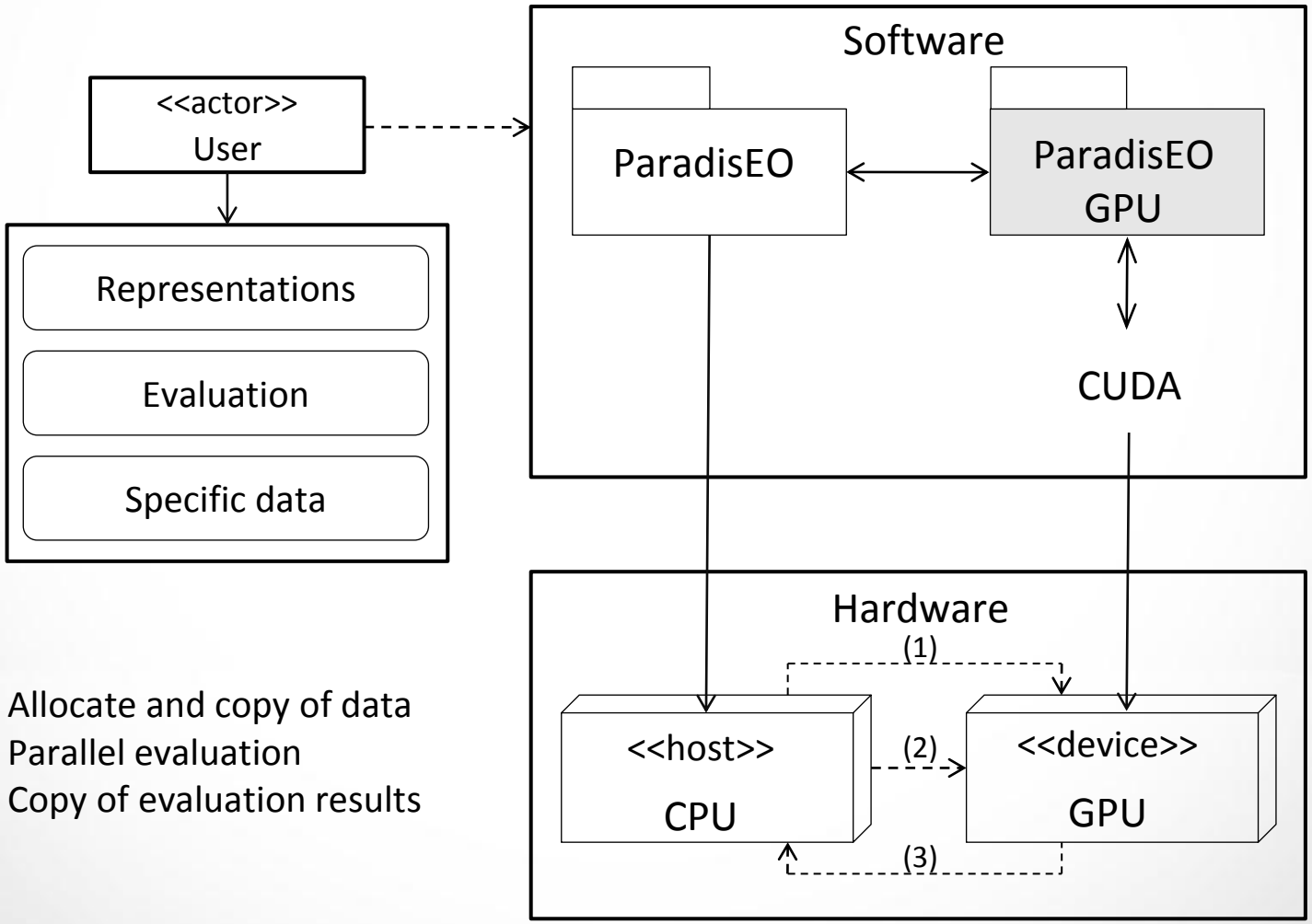
N. Melab, S. Cahon and E-G. Talbi. Grid Computing for Parallel Bioinspired Algorithms. Journal of Parallel and Distributed Computing (JPDC), Elsevier Science, Vol. 66(8), Pages 1052-1061, Aug. 2006.

Iteration-level implementation

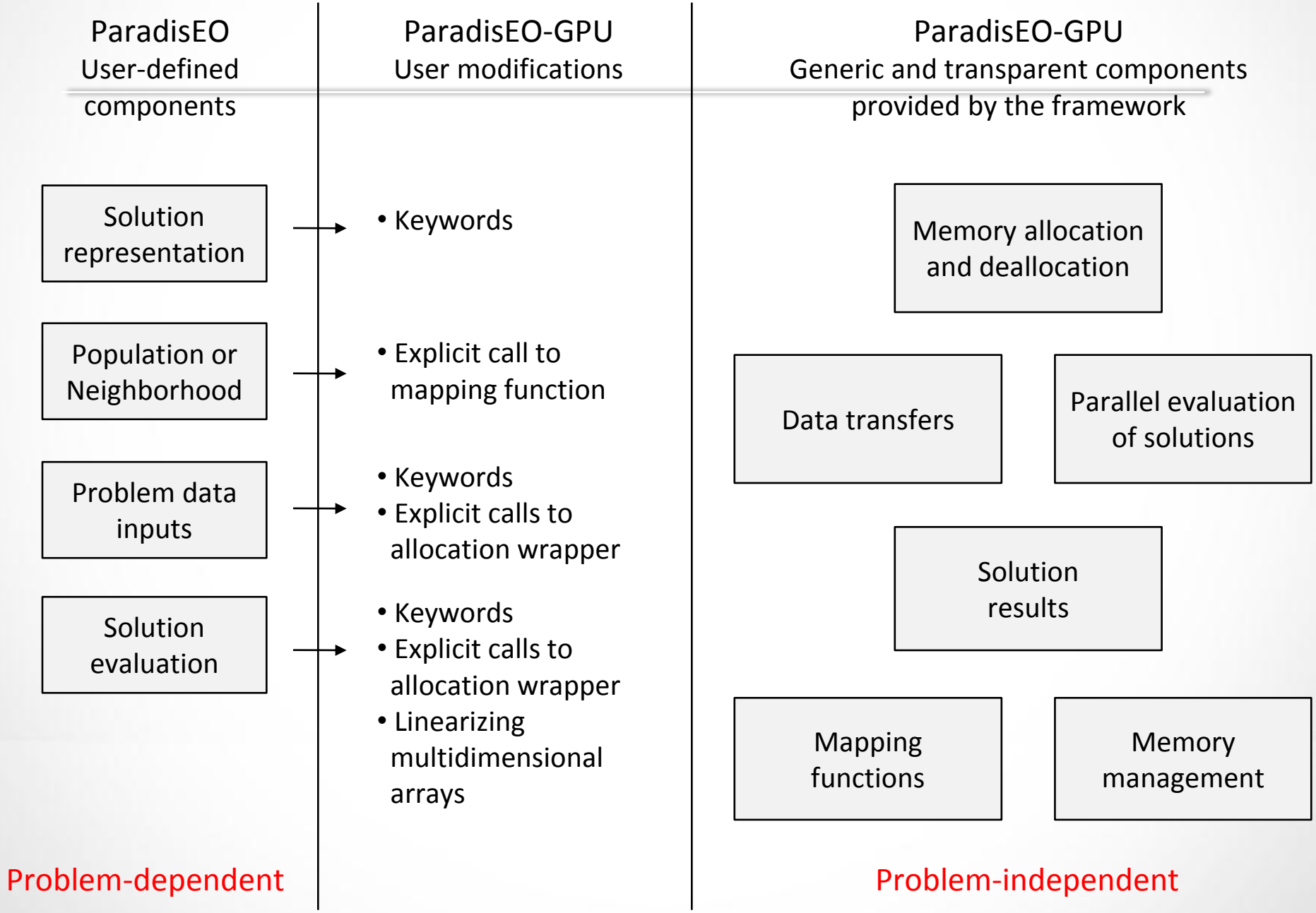


- Parallel model which provides generic concepts …
- … transparent and parallel evaluation of solutions on GPU
- MW model which does not change the original semantics of algorithms

Layered architecture of ParadisEO-GPU

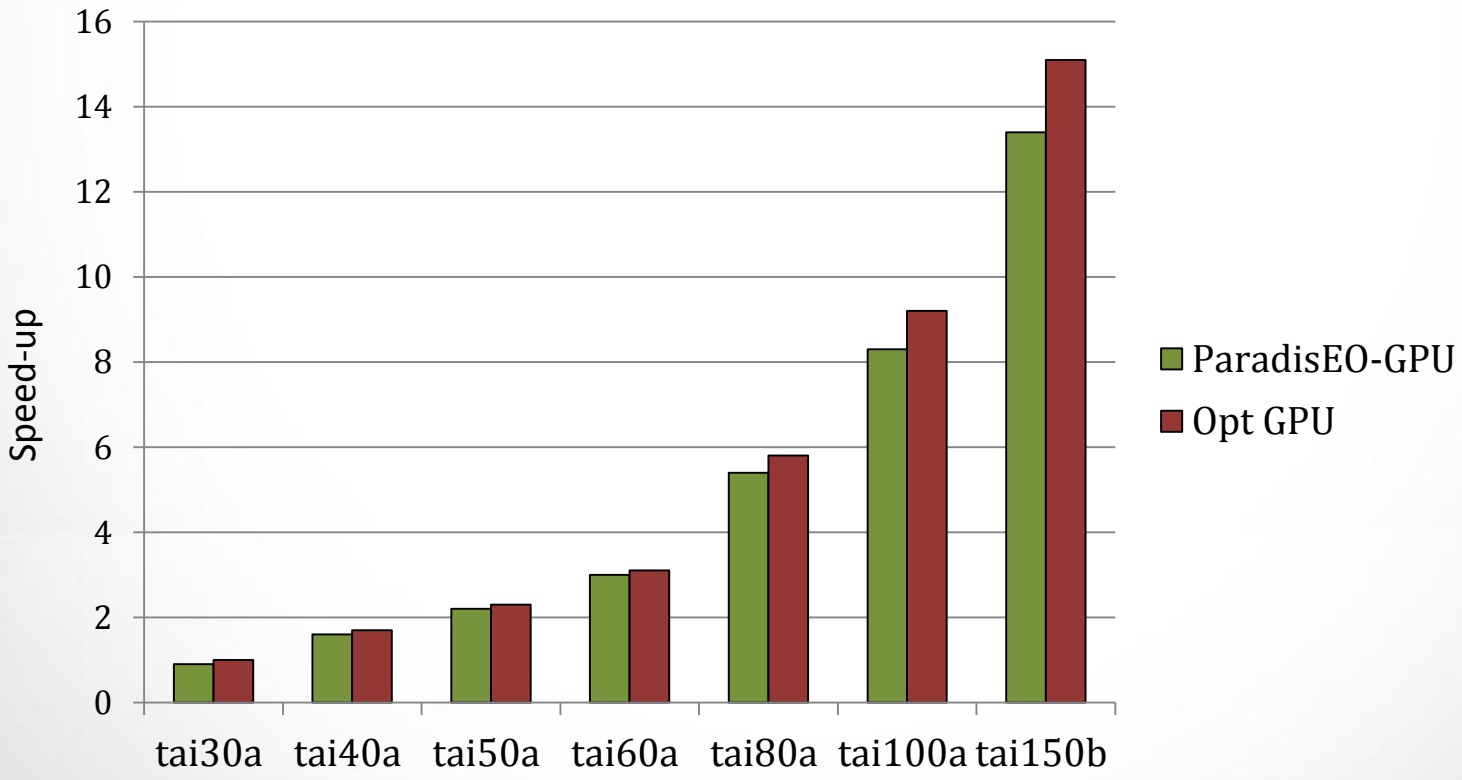


(1) Allocate and copy of data
(2) Parallel evaluation
(3) Copy of evaluation results


ParadisEO
User-defined components
ParadisEO-GPU
User modifications
ParadisEO-GPU
Generic and transparent components provided by the framework
Solution representation
• Keywords
Population or Neighborhood
• Explicit call to mapping function
Problem data inputs
• Keywords
• Explicit calls to allocation wrapper
Solution evaluation
• Keywords
• Explicit calls to allocation wrapper
• Linearizing multidimensional arrays
Memory allocation and deallocation
Data transfers
Parallel evaluation of solutions
Solution results
Mapping functions
Memory management
Problem-dependent
Problem-independent

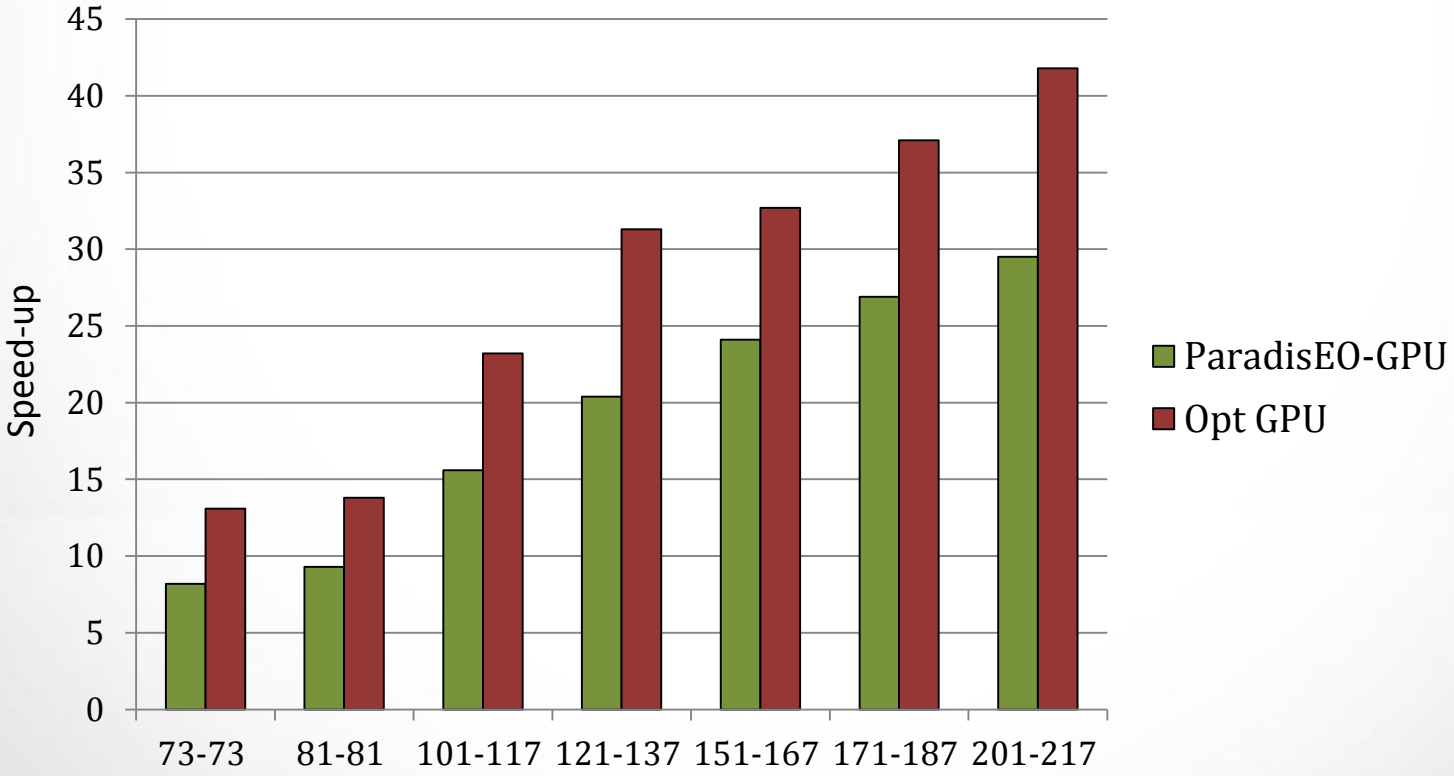
Performance of ParadisEO-GPU (1)

- Application to the quadratic assignment problem
- Tabu search using a neighborhood based on pairwise-exchange operator
- Intel Core i7 3.2 Ghz + GTX 480



Performance of ParadisEO-GPU (2)

- Application to the permuted perceptron problem
- Tabu search using a neighborhood based on a Hamming distance of two
- Intel Core i7 3.2 Ghz + GTX 480



Outline

Conclusion

- GPU-based metaheuristics require to re-design existing parallel models: iteration-level (MW) and algorithmic-level (PC)
- Efficient CPU-GPU cooperation
  - Task repartition, optimization of data transfers for S-metaheuristics (generation of the neighborhood on GPU and reduction)
- Efficient parallelism control
  - Mapping between work units and threads Ids, thread control for the threads generation (parameters tuning, fault-tolerance)
- Efficient memory management
  - Use of texture memory for optimization structures, parallelization schemes for parallel cooperative P-metaheuristics (global and shared memory)

Perspectives

- Heterogeneous computing for metaheuristics
  - Efficient exploitation of all available resources at disposal (CPU cores and many GPU cards)
  - Arrival of GPU resources in COWs and grids …
  - … conjunction of GPU computing and distributed computing to fully exploit the hierarchy of parallel model of metaheuristics
- Multiobjective optimization
  - Parallel archiving of non-dominated solutions represents a prominent issue in the design of multiobjective metaheuristics …
  - … SIMD parallel archiving on GPU with additional synchronizations, non-concurrent writing operations and dynamic allocations on GPU

Publications

- 2 international journals: IEEE Transactions on Computers, parallel processing letters
- 9 international conference proceedings: GECCO, EVOCOP, IPDPS …
- 1 national conference proceeding.
- 2 conference abstracts
- 8 workshops and talks.
- 1 research report.

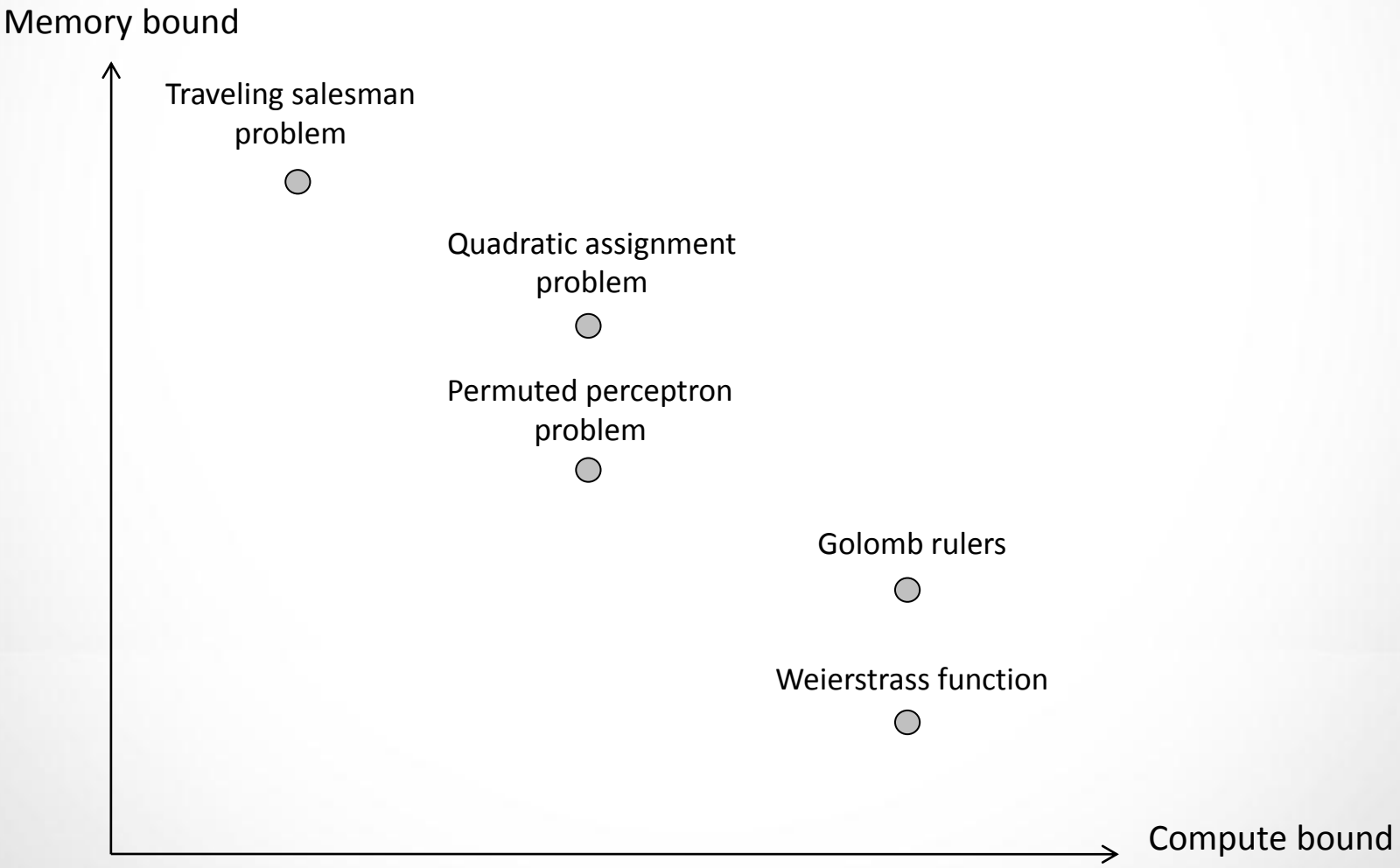
THANK YOU FOR YOUR ATTENTION

Additional slides

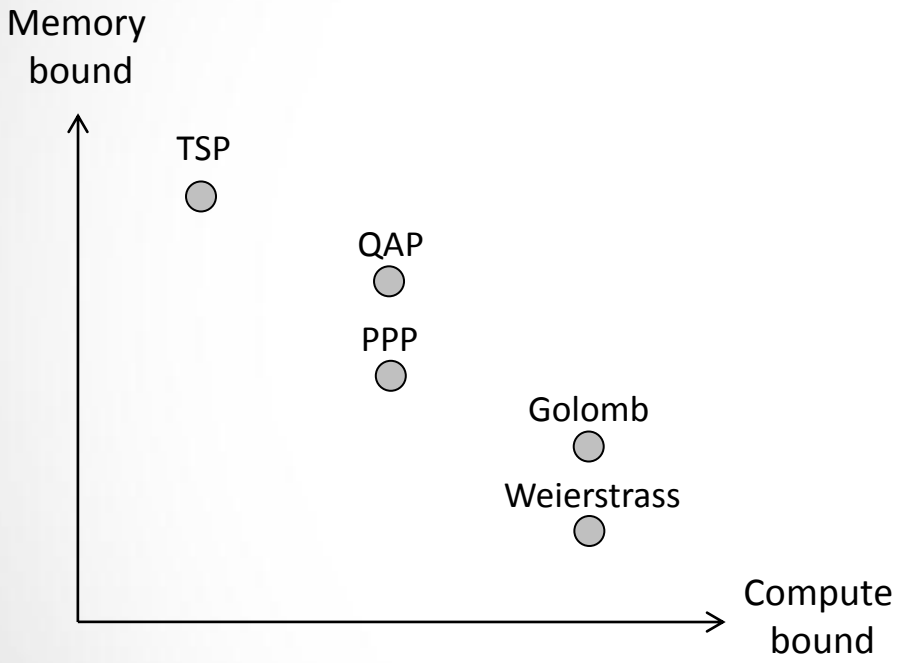
Performances of optimization problems (1)

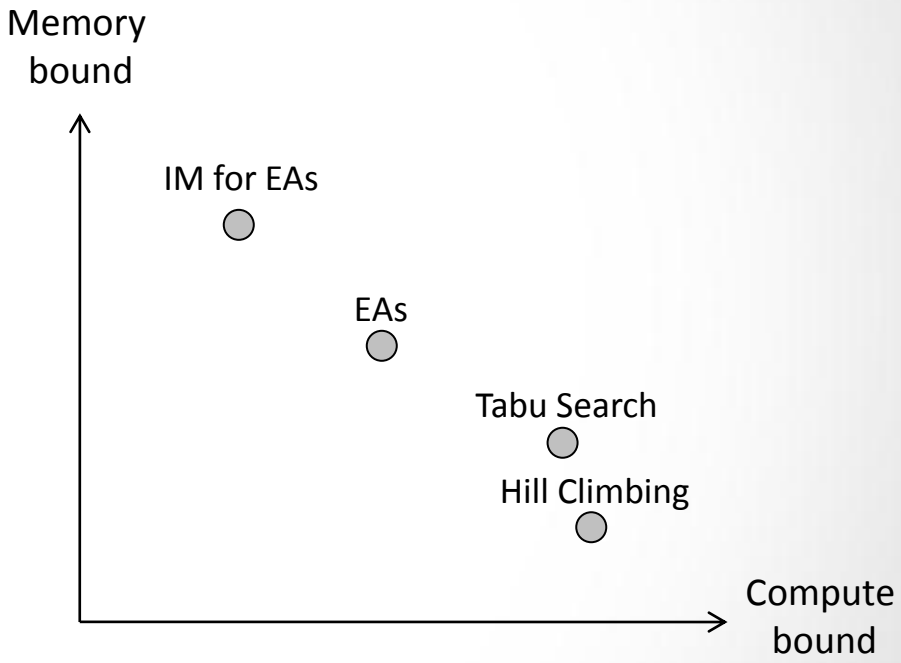
Problem	Data inputs	Evaluation	Δ-evaluation		Performance
		Time complexity	Time complexity	Space complexity	
Permuted perceptron	One matrix	$O(n^2)$	$O(n)$	$O(n)$	++
Quadratic assignment	Two matrices	$O(n^2)$	$O(1)$ / $O(n)$	$O(n^2)$	+
Weierstrass function	-	$O(n^2)$	$O(n^2)$	-	++++
Traveling salesman	One matrix	$O(n)$	$O(1)$	-	+
Golomb rulers	-	$O(n^3)$	$O(n^2)$	$O(n^2)$	+++

Performances of optimization problems (2)



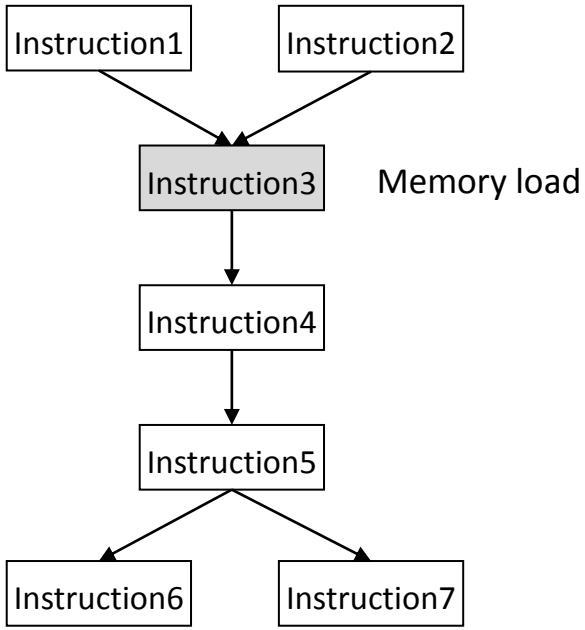
Performances of optimization problems (3)

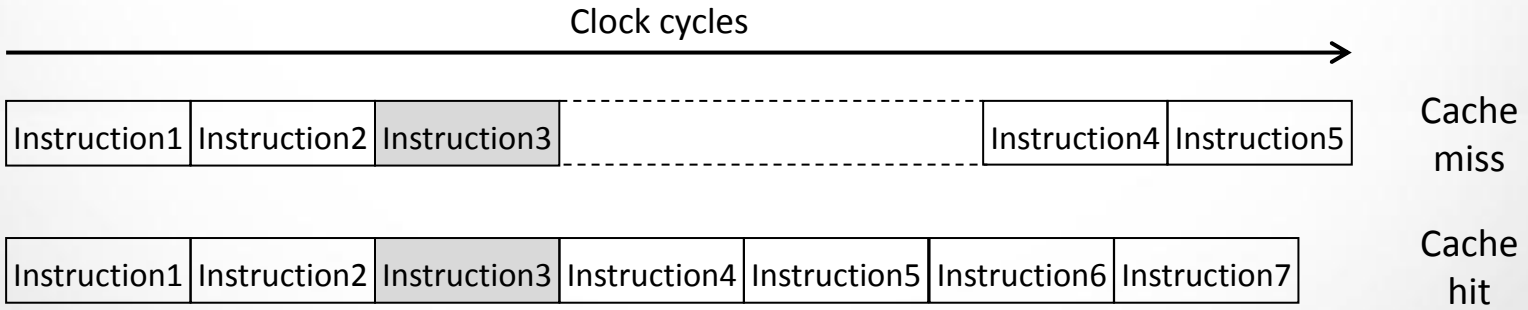




For the Weierstrass function

Performances of optimization problems (4)





Performances of optimization problems (5)

Analysis of data cache

Island model for evolutionary algorithms

GTX 280 – Instance 10 – 64 islands – 128 individuals

- CPU implementation
  - Valgrind + cachegrind
  - L1 cache misses: 84% (around 10 clock cycles per miss)
  - L2 cache misses: 71% (around 200 clock cycles per miss)

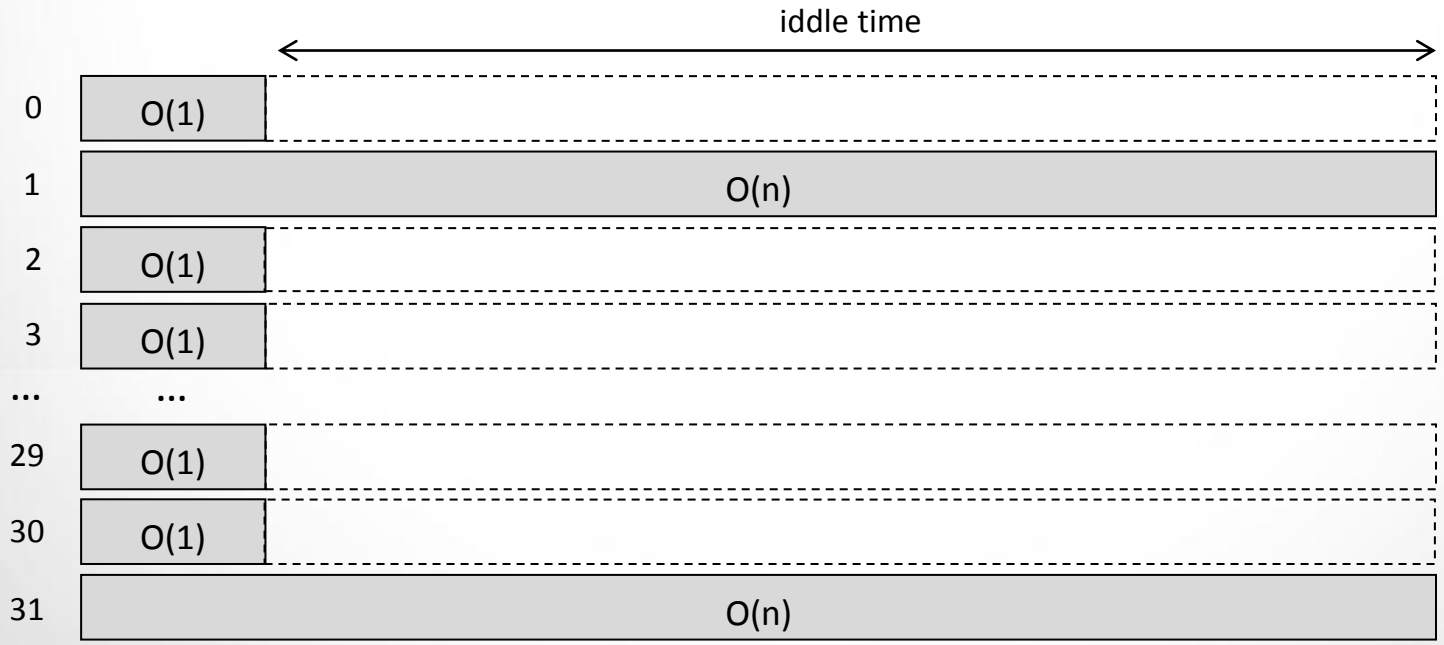
Memory bound

- AGPUShared implementation
  - CUDA profiler
  - Shared memory (around 10 clock cycles per access)
  - 16KB per multiprocessor (30 multiprocessors)
  - Population fit into the shared memory: 128 x 10 x 4 ≈ 5 KB per island

Compute bound

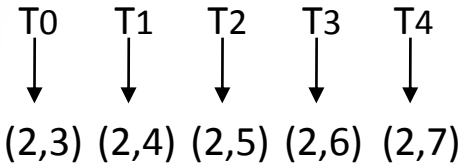
Irregular application (1)

- Evaluation function of the quadratic assignment
  - Weakly irregular
- S-metaheuristics based on a pair-wise exchange operator
  - 2n-3 neighbors can be evaluated in O(n)
  - (n-2) x (n-3) / 2 neighbors can be evaluated in O(1)

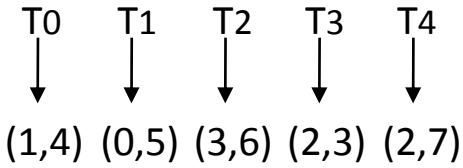


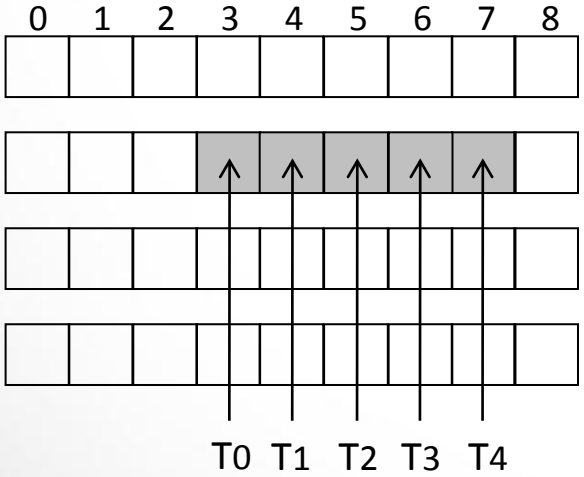
Irregular application (2)

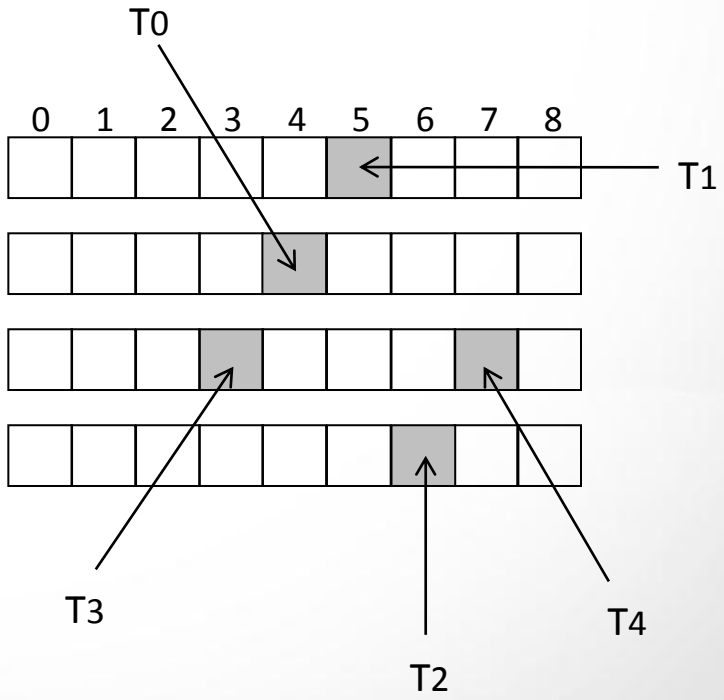
Local search based on best improvement



Local search based on first improvement







Cryptanalysis techniques PPP

0	1	2	3	4	5	6	7	8
1	-1	1	1	-1	-1	1	1	1
1	1	1	1	-1	-1	-1	1	1
1	-1	1	1	1	-1	1	-1	1
1	1	-1	1	1	-1	1	1	1
1	-1	1	1	-1	-1	-1	1	1
1	1	-1	1	-1	-1	-1	1	1

1			1		-1			1

Majority vector

1	0,5	0,66	1	0,33	0	0,5	0,83	1

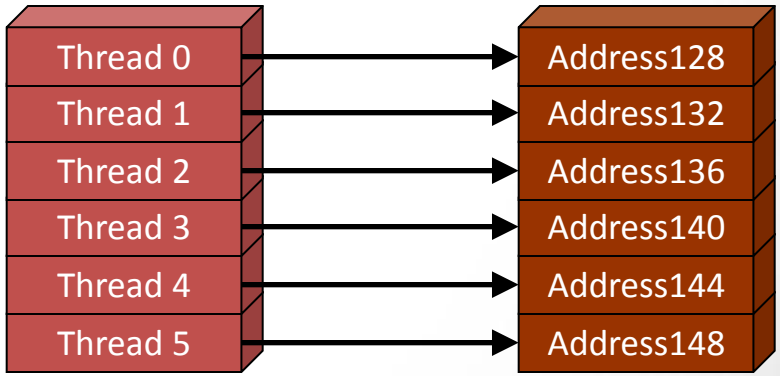
Probabilistic vector (EDAs)

Memory coalescing (1)

Coalescing accesses to global memory (matrix vector product)

```
sum[id] = 0;
for (int i = 0; i < m; i++) {
   sum[id] += A[i * n + id] * B[id];
}
```

```
sum[0] = A[i * n + 0] * B[0]
sum[1] = A[i * n + 1] * B[1]
sum[2] = A[i * n + 2] * B[2]
sum[3] = A[i * n + 3] * B[3]
sum[4] = A[i * n + 4] * B[4]
sum[5] = A[i * n + 5] * B[5]
```



Memory access pattern

SIMD: 1 memory transaction

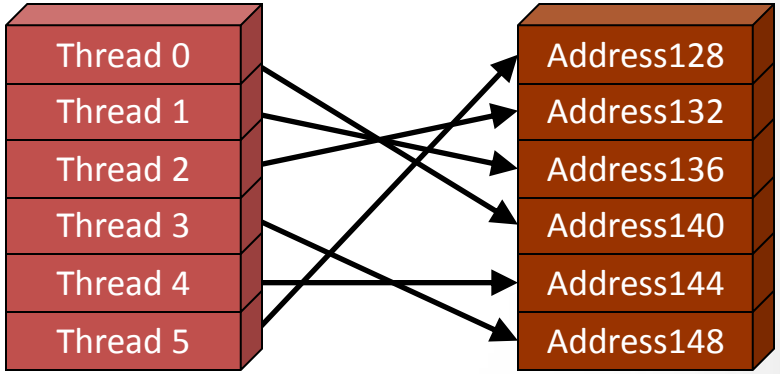
Memory coalescing (2)

Uncoalesced accesses to global memory for evaluation functions

	0	1	2	3	4	5
p	3	2	1	5	4	0

```
sum[id] = 0;
for (int i = 0; i < m; i++) {
  sum[id] += A[i * n + id ] * B[p[id]];
}
```

6 memory transactions



Memory access pattern

Because of LS methods structures, memory coalescing is difficult to realize

➔ it can lead to a significantly performance decrease.

Pros and cons of parallel (memory management)

Algorithm	Parameters	Limitation of the local population size	Limitation of the instance size	Limitation of the total population size	Speed
CPU	Heterogeneous	Not limited	Very Low	Very Low	Slow
CPU+GPU	Heterogeneous	Not limited	Low	Low	Fast
GPU	Homogeneous	Size of a threads block	Low	Medium	Very Fast
GPU Shared Memory	Homogeneous	Limited to shared memory	Limited to shared memory	Medium	Lightning Fast