Metaheuristics on GPU

Thé Van Luong, Nouredine Melab and El-Ghazali Talbi

DOLPHIN Project Team

April 2010

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche LILLE - NORD EUROPE

Local search on GPU: From design to implementation

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

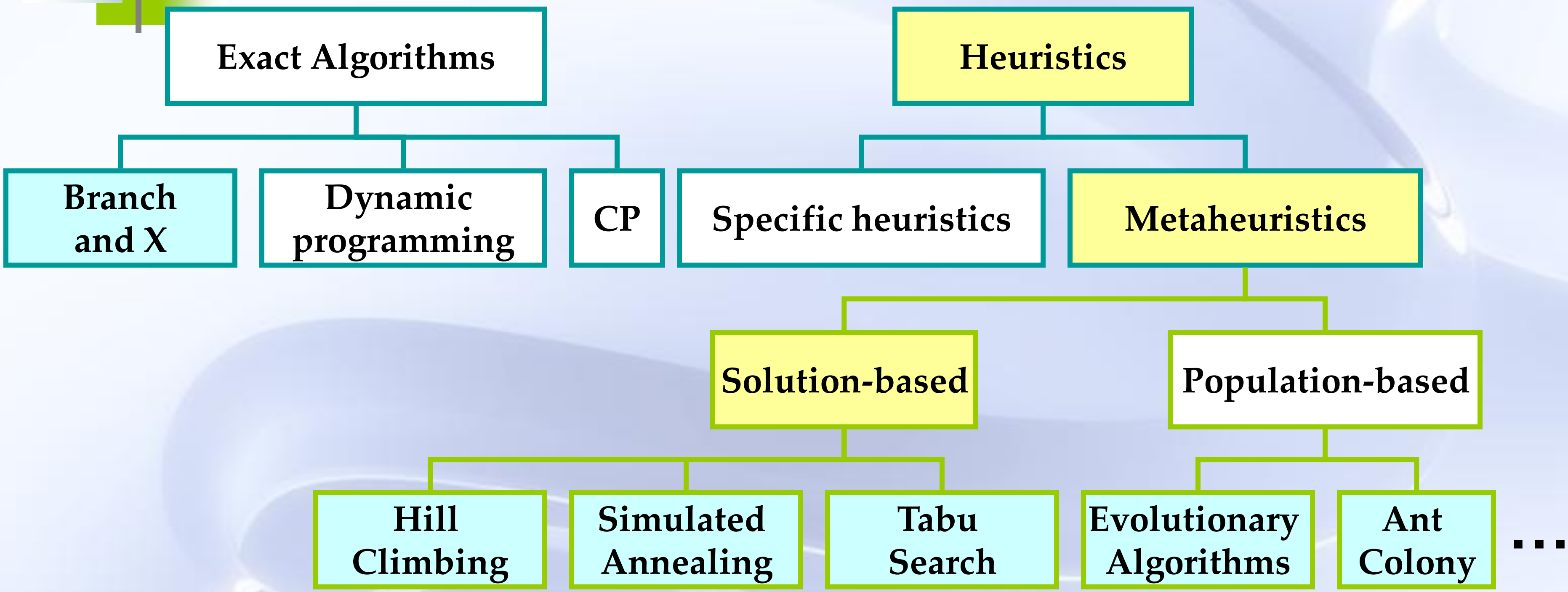
INRIA

centre de recherche
LILLE - NORD EUROPE

Outline

- **Parallel Local Search Metaheuristics (PLSM)**
- GPU-based Design and Implementation of PLSM
- Application to the Permuted Perceptron Problem (PPP)
- Conclusion and Future Work

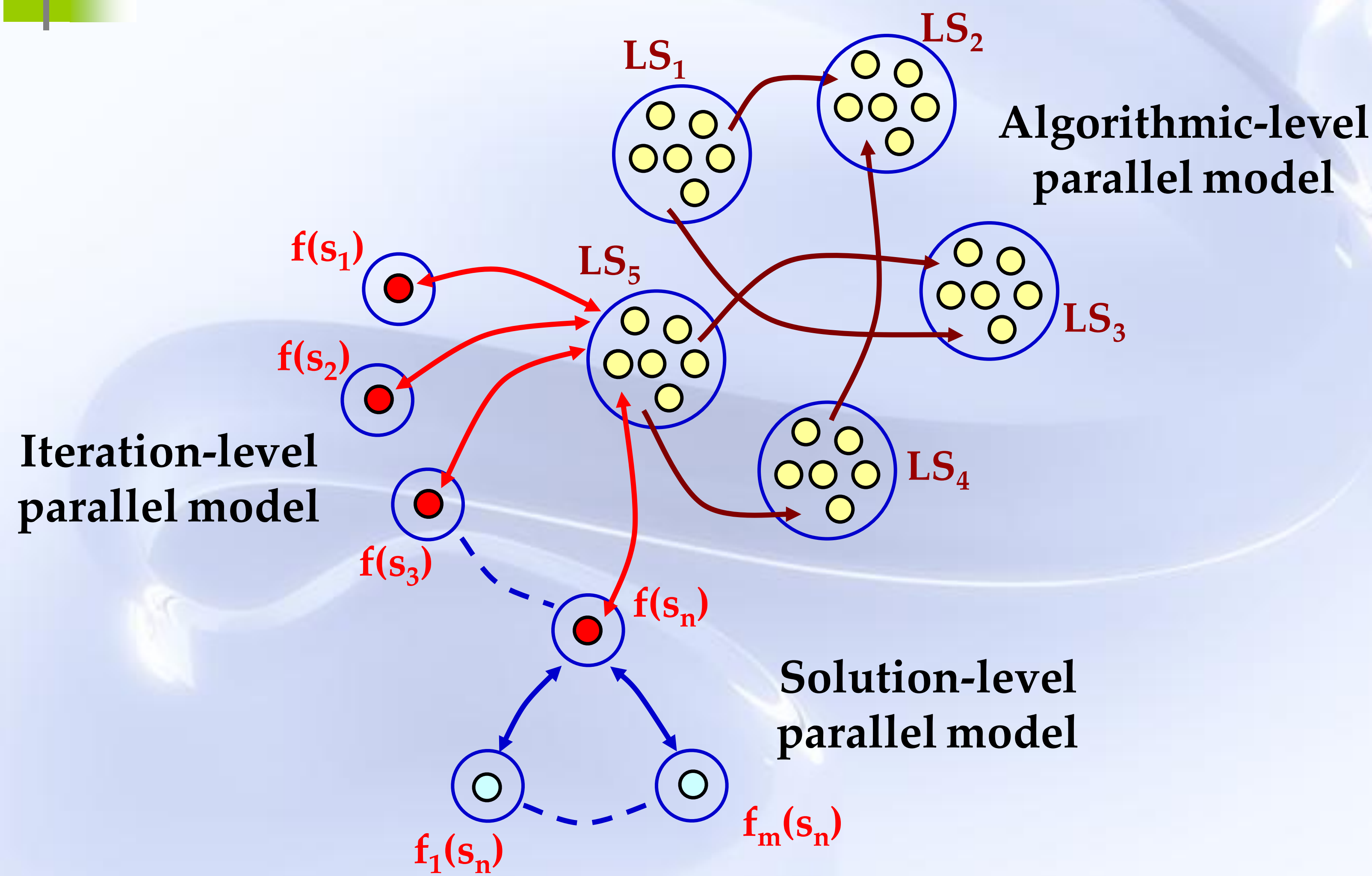
A taxonomy of optimization methods



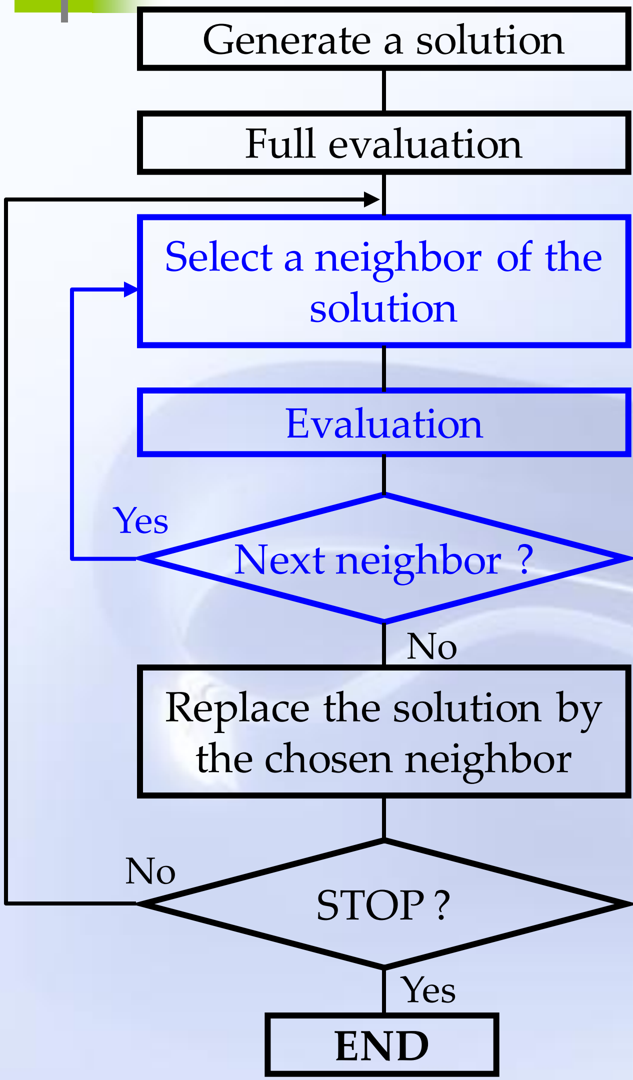
Exploitation-oriented Exploration-oriented

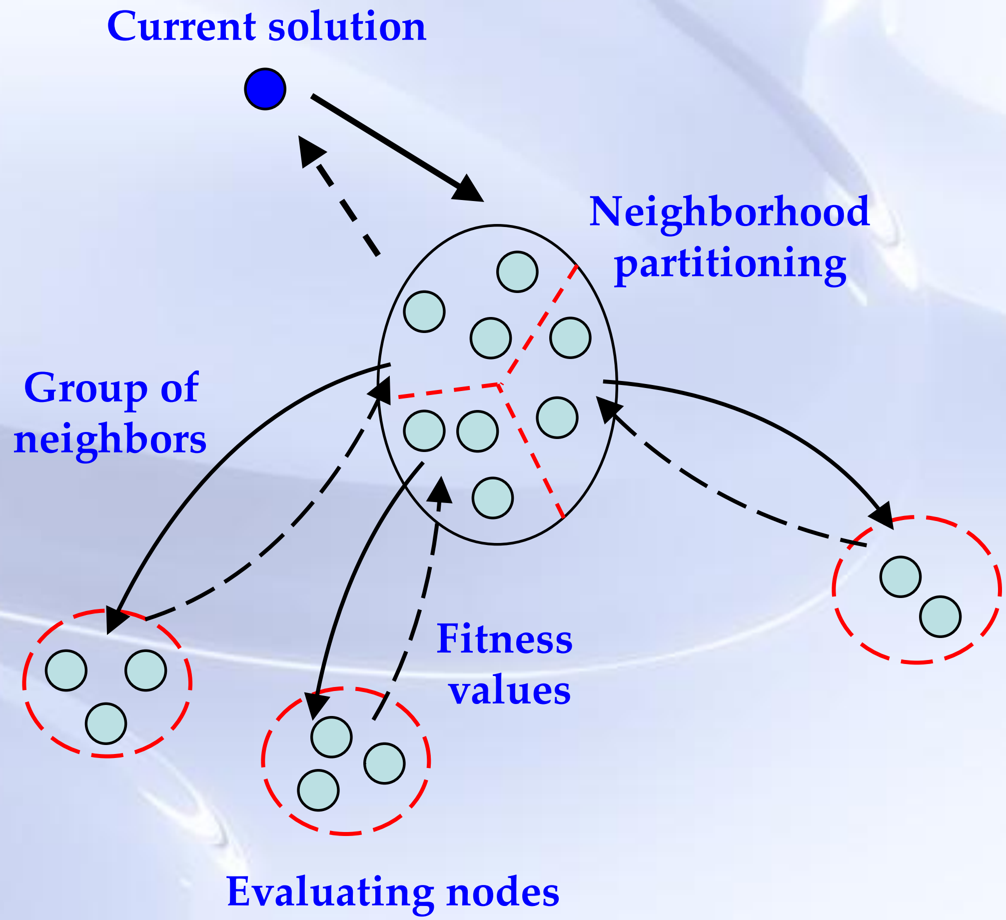
- Exact methods : optimality but exploitation on small size problem instances
- Metaheuristics : Near-optimality on larger problem instances, but …
  - … Need of **massively parallel computing** on very **large** instances

Parallel models for LSM



Iteration-level parallel model





- … Need of **massively parallel computing** on very **large** neighborhoods

Outline

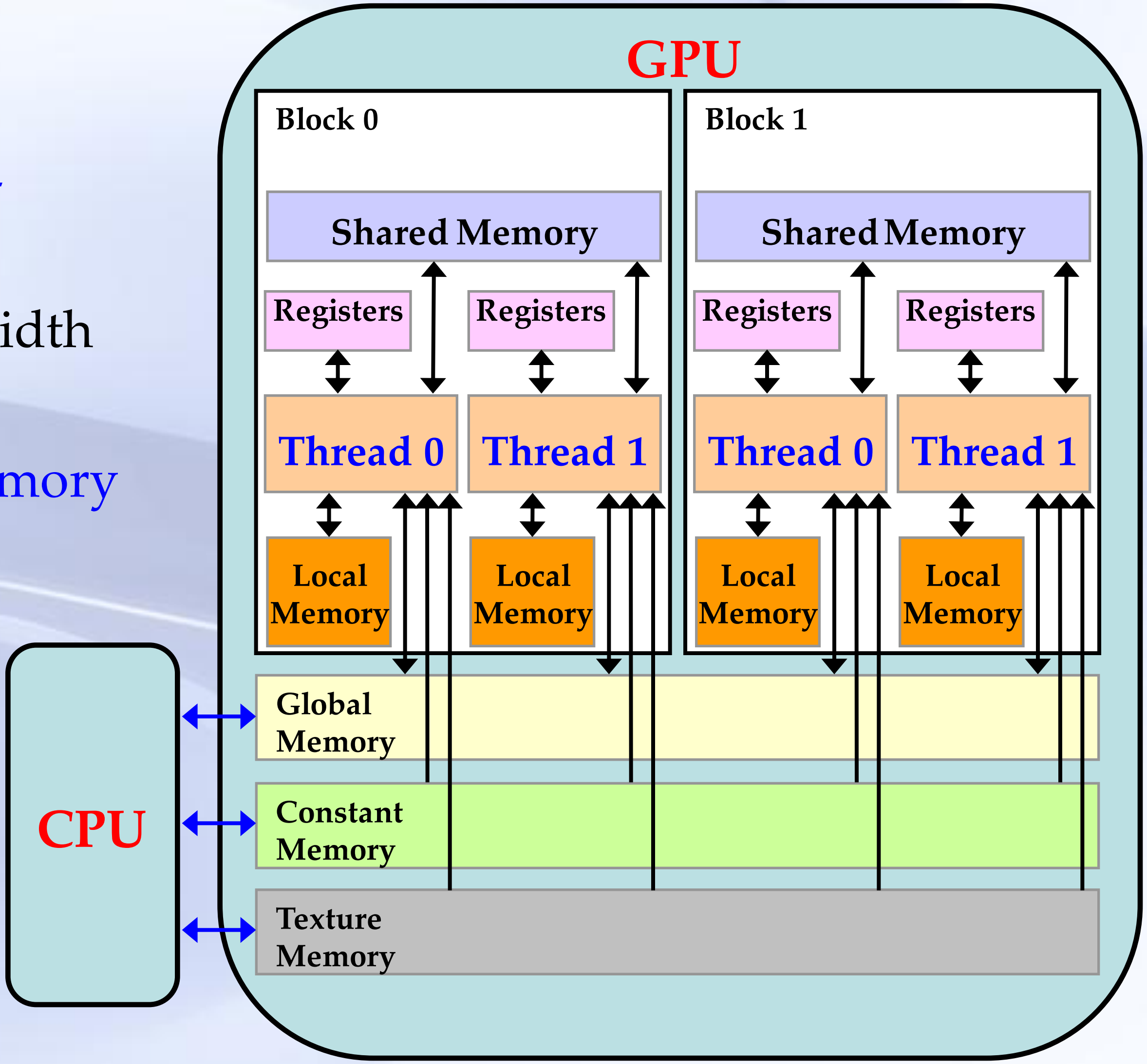
- Parallel Local Search Metaheuristics (PLSM)
- **GPU-based Design and Implementation of PLSM**
- Application to the Permuted Perceptron Problem (PPP)
- Conclusion and Future Work

GPU Computing

- Used in the past for graphics and video applications …
- … but now popular for many other applications such as scientific computing [Owens *et al*. 2008]
- Publication of the CUDA development toolkit that allows GPU programming in a C-like language [Garland *et al*. 2008]
- In the metaheuristics field:
  - Several existing works (Genetic algorithms [Wong 2006], Genetic programming [Harding *et al.* 2009], …)
  - A very light tentative for the Tabu search algorithm [Zhu *et al*. 2009]

GPU Characteristics

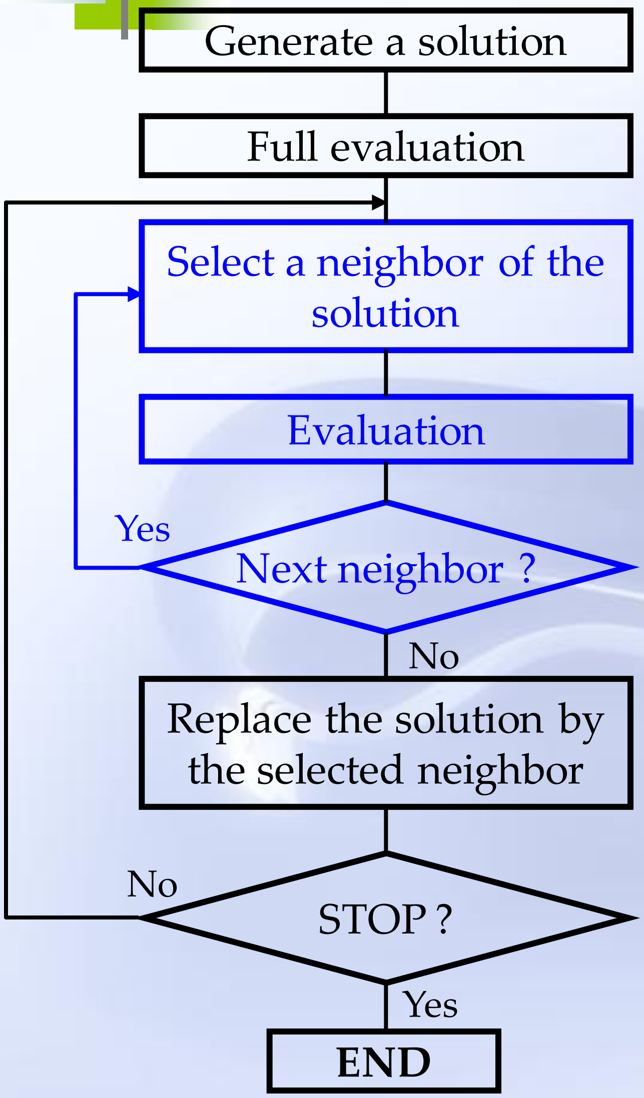
- Highly parallel multi-threaded many-core
- High memory bandwidth compared to CPU
- Different levels of memory (different latencies)



Objective and challenges

- Re-think the iteration-level parallel model to take into account the characteristics of GPU
  - Challenges at three layers …
- **CPU-GPU cooperative layer**
  - **Work partitioning between CPU and GPU**
  - **Data transfer optimization**
- Parallelism control layer
  - Neighborhood generation control (memory capacity constraints)
  - Efficient mapping between candidate solutions and threads ids
- Memory management layer
  - Which data on which memory (latency and capacity constraints) ?

Cooperation layer: CPU→ GPU data transfer



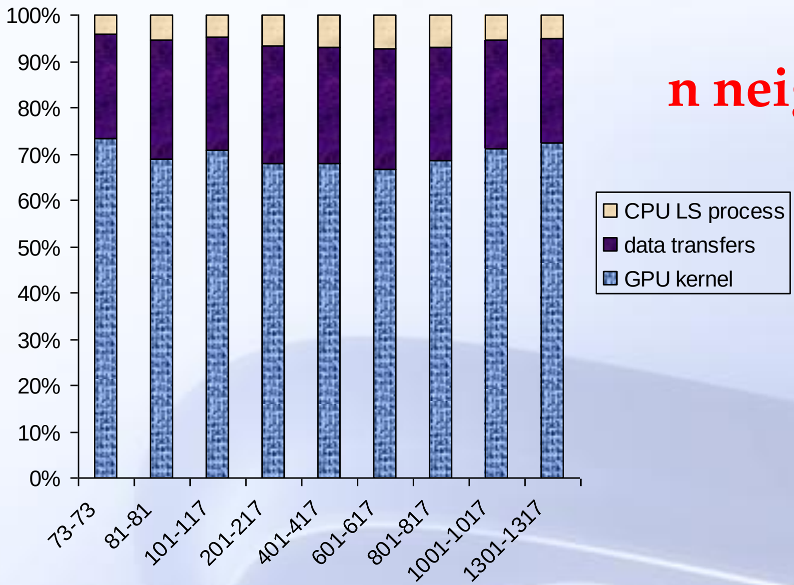
- CPU (host) controls the whole sequential part of LSM
- GPU evaluates the neighborhood
- Objective
  - Optimizing the CPU→ GPU data transfer
- Issues
  - Where the neighborhood is generated ?
  - Two approaches:
    - **Approach 1**: generation on CPU and evaluation on GPU
    - **Approach 2**: generation and evaluation on GPU

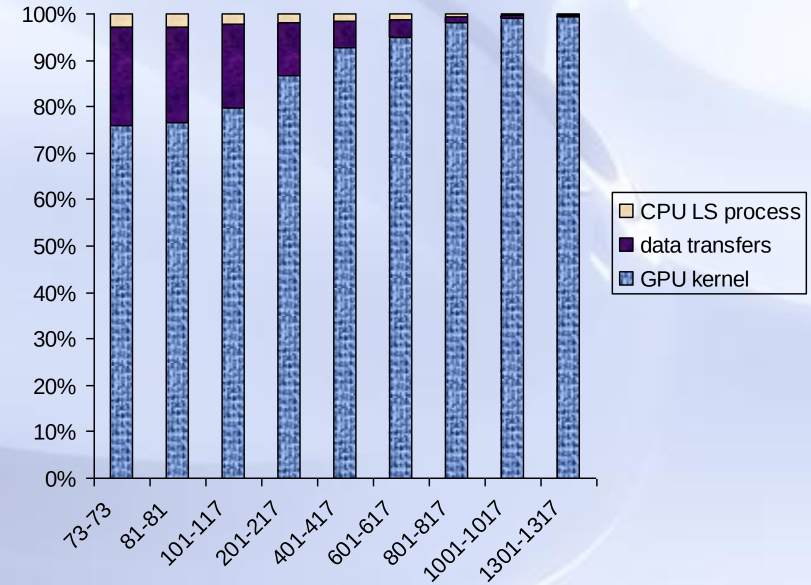
Performance of the two approaches

Approach 1

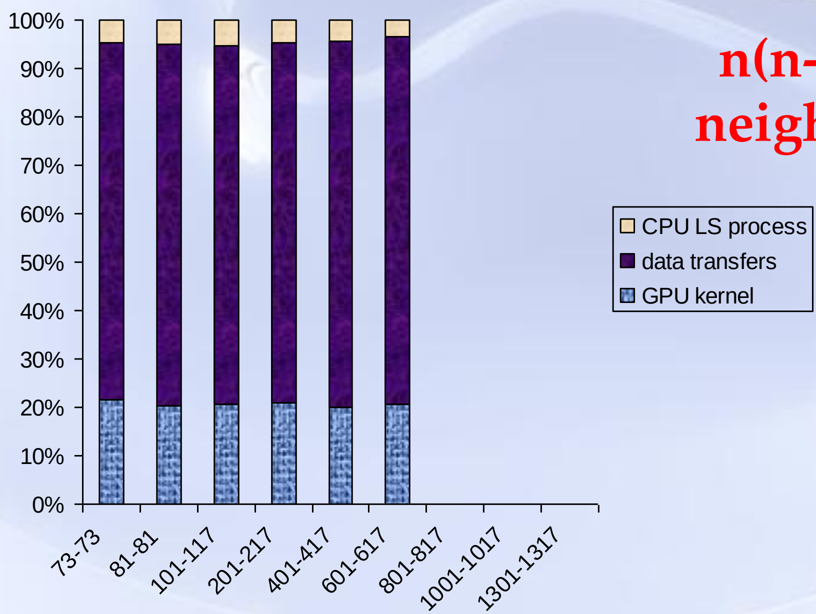
Approach 2

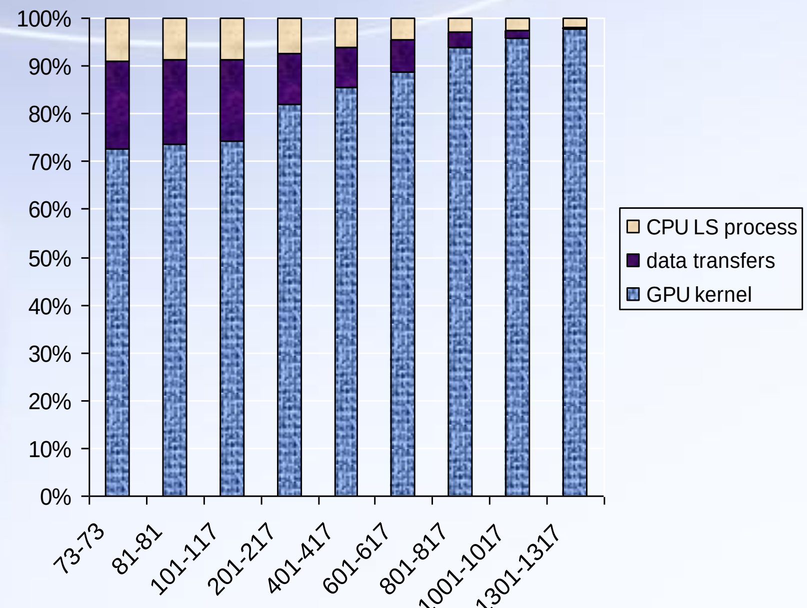
n neighbors



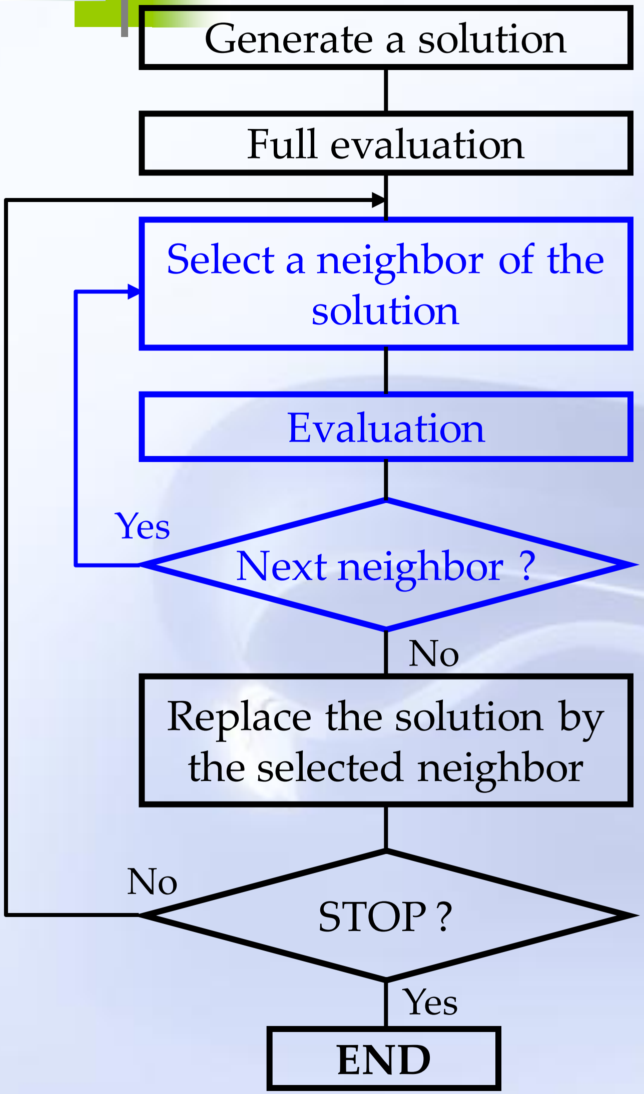


n(n-1)/2 neighbors



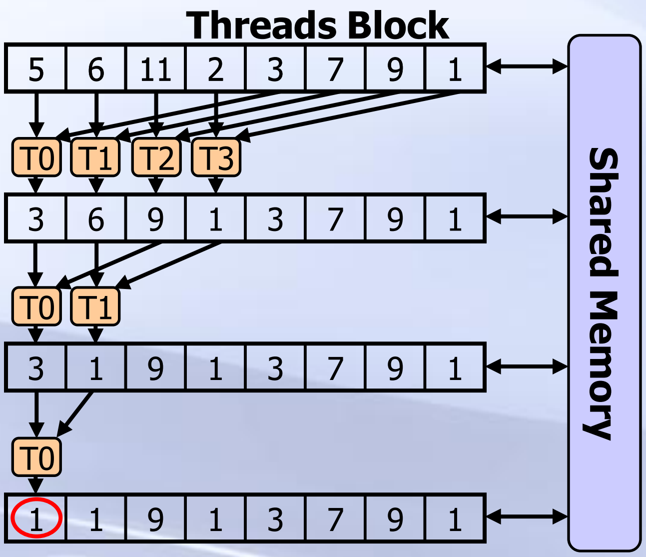


Cooperation layer: GPU→ CPU data transfer



- Objective
  - Optimizing the GPU→ CPU data transfer
- Issues
  - Where is done the selection of the best neighbors ?
  - Two approaches:
    - **Approach 1**: on CPU i.e. transfer of the data structure storing the fitnesses associated with the solutions
    - **Approach 2**: on GPU i.e. use of the reduction operation to select the best solution

GPU reduction to select the best solution



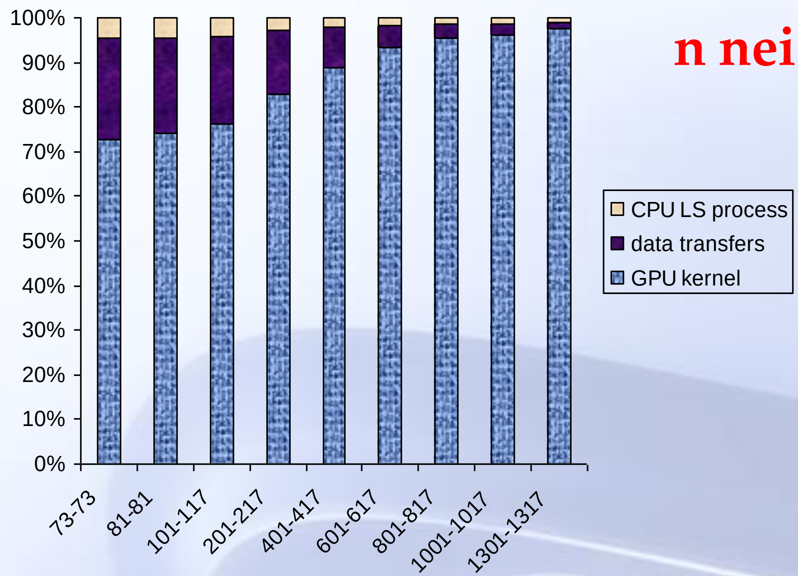
- GPU reduction kernel to find the minimum of each block of threads
- Complexity: $O(\log_2(n))$
- Cooperation of threads of a same block through the shared memory (latency: ~10 cycles)
- Performing iterations on reduction kernels allows to find the minimum of all neighbors

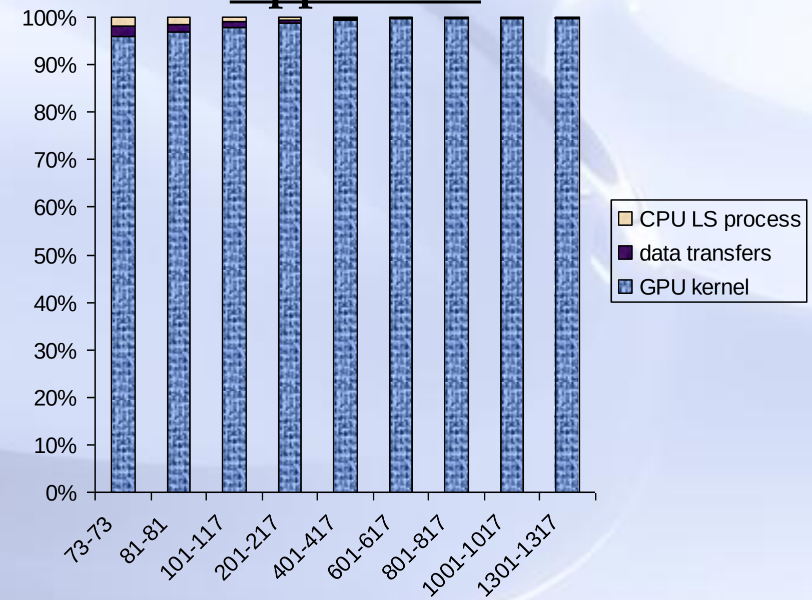
Performance of the two approaches

Approach 1

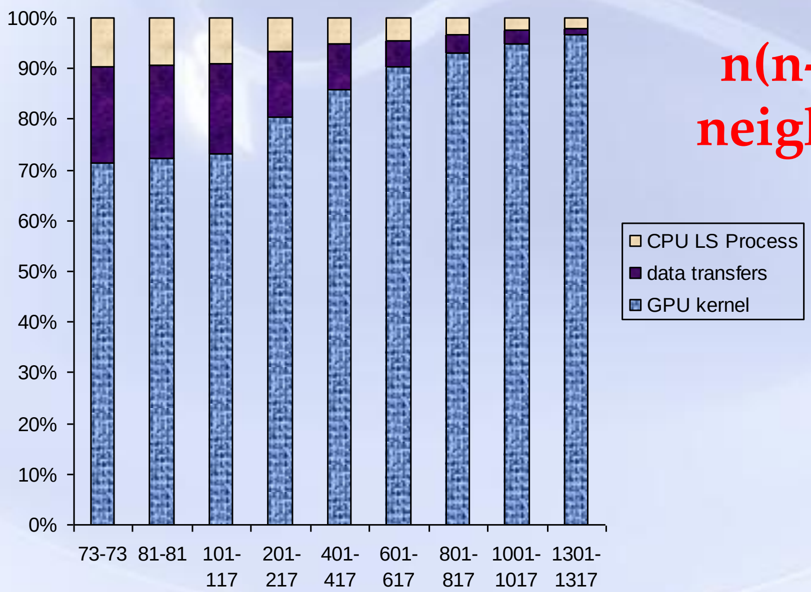
Approach 2

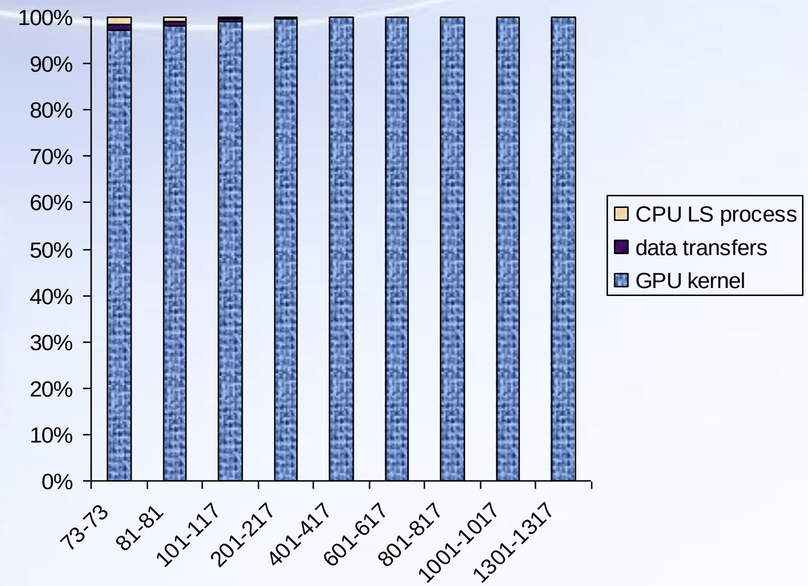
n neighbors





n(n-1)/2 neighbors





Recommendation

- Optimizing the CPU-GPU data transfer is a must to improve the efficiency of GPU-based LSM
- CPU→GPU data transfer
  - The neighborhood must be generated on GPU
  - Issue: defining an efficient mapping between the neighboring solutions and threads ids
- GPU→CPU data transfer
  - Avoid, if possible, the transfer of the whole data structure storing the neighboring fitnesses
  - Use of the thread reduction mechanism

Objective and challenges

- Re-think the iteration-level parallel model to take into account the characteristics of GPU
  - Challenges at three layers …
- CPU-GPU cooperative layer
  - Work partitioning between CPU and GPU
  - Data transfer optimization
- **Parallelism control layer**
  - **Neighborhood generation control (memory capacity constraints)**
  - **Efficient mapping between candidate solutions and threads ids**
- Memory management layer
  - Which data on which memory (latency and capacity constraints) ?

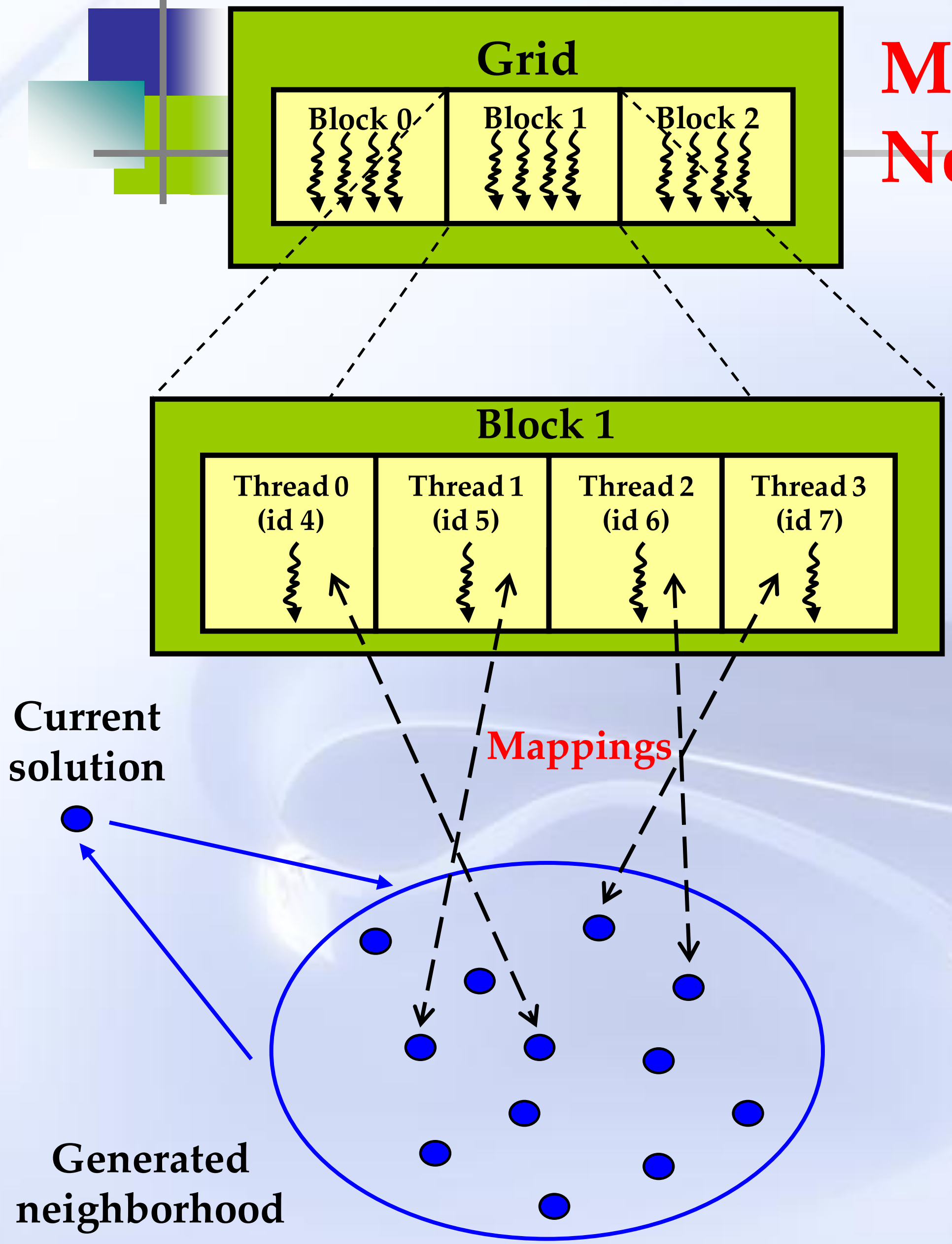
Parallelism control layer

- The parallelism control layer focuses on the neighborhood generation and evaluation on GPU
- The kernel handling is dependent of the general-purpose language
- The GPGPU paradigm introduces a model of threads which provides an easy abstraction for SIMD architecture
- CUDA and OpenCL provide an application programming interface for GPU architectures

Neighborhood generation control

- A kernel is launched with a large number of threads (SPMD model)
- The major issue is …
  - … to control the generation of the neighborhood to meet the memory capacity constraints
- Full evaluation
  - Additional duplication of the original solution for each thread dealing with a neighbor
- → Use incremental evaluation as possible
  - No additional allocated memory for each thread

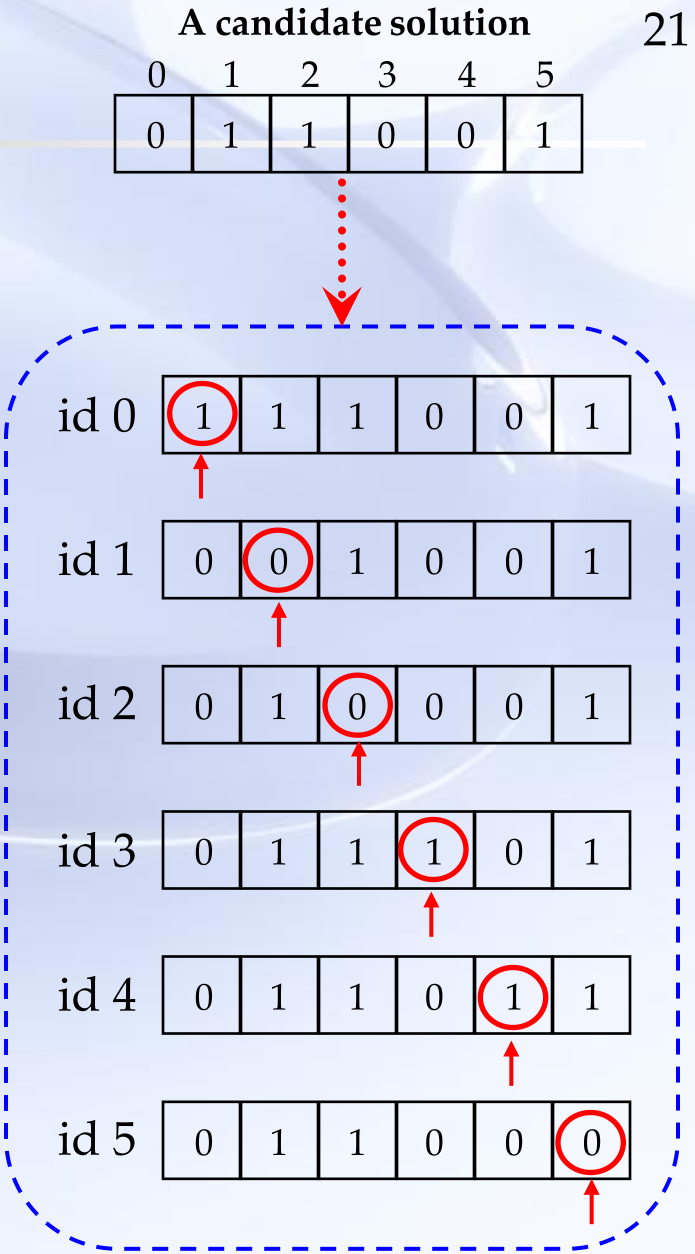
Mapping Neighbor Id → Thread Id (1)



- According to the threads spatial organization, a unique id must be assigned to each thread to compute on different data
- The challenging issue is to find efficient mappings between a thread *id* and a particular neighbor
    - Representation-dependent

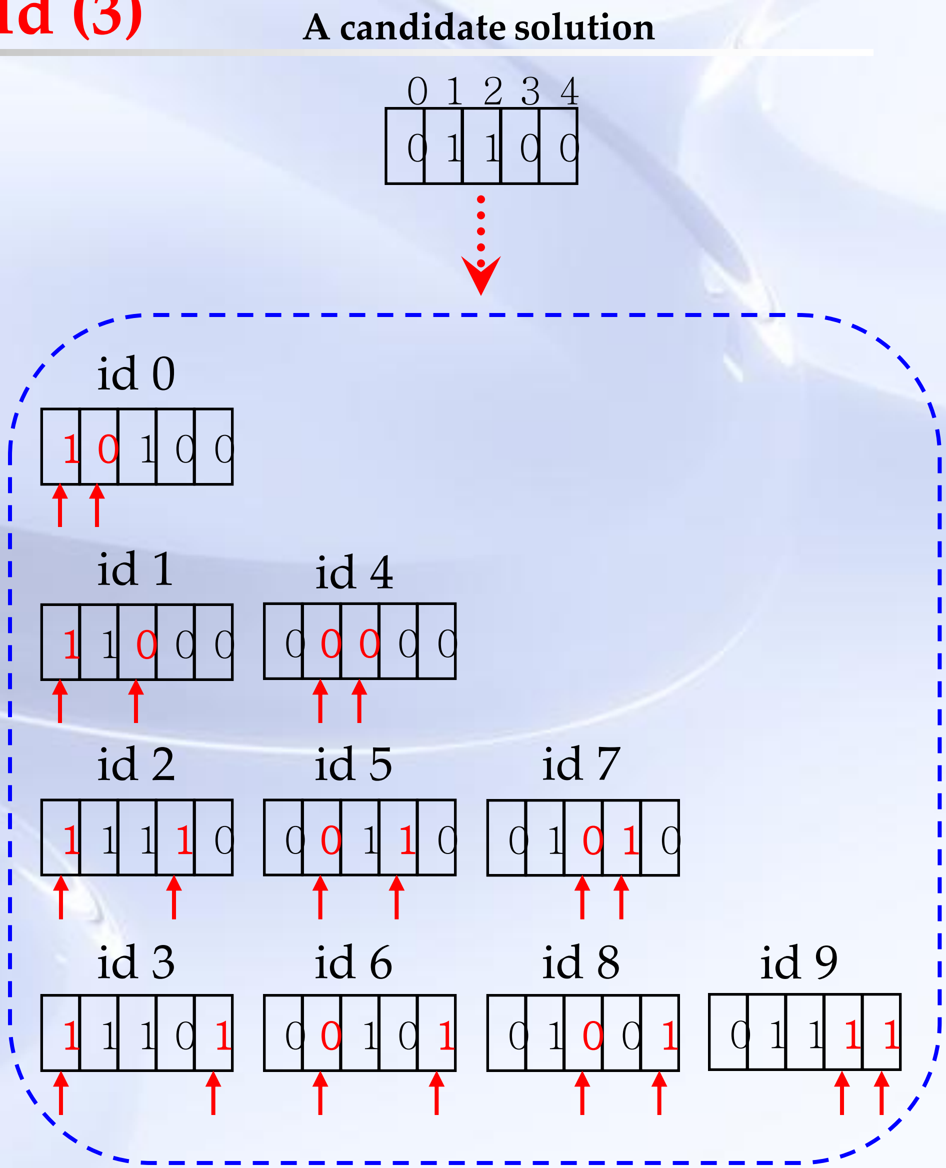
Mapping Neighbor Id → Thread Id (2)

- A mapping is proposed for 3 well-known representations (binary, discrete, permutation)
- Binary representation
  - The thread with *id=i* generates and evaluates a candidate solution by flipping the *bit number i* of the initial solution
  - *n* threads are generated for a solution of size *n*
  - Fitness data structure size = *n*



Mapping Neighbor Id → 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*
  - *n x (n-1) / 2* threads are generated for a solution of size *n*
  - Fitness data structure size = *n x (n-1) / 2*

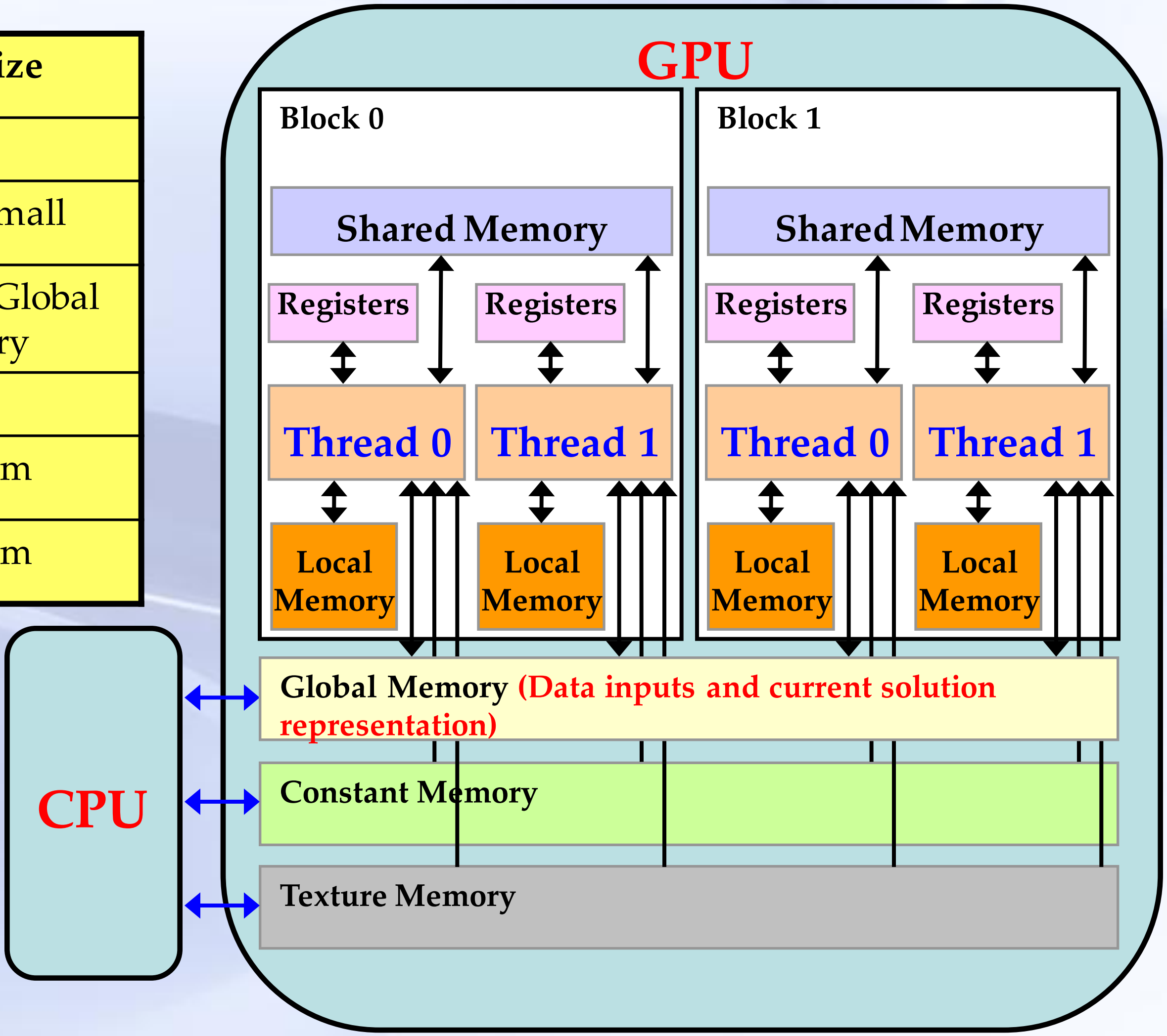


Objective and challenges

- Re-think the iteration-level parallel model to take into account the characteristics of GPU
  - Challenges at three layers …
- CPU-GPU cooperative layer
  - Work partitioning between CPU and GPU
  - Data transfer optimization
- Parallelism control layer
  - Neighborhood generation control (memory capacity constraints)
  - Efficient mapping between candidate solutions and threads Ids
- **Memory management layer**
  - **Which data on which memory (latency and capacity constraints) ?**

Memory management layer

Memory type	Speed	Size
Global	Slow	Big
Registers	Very fast	Very small
Local	Slow	Up to Global memory
Shared	Fast	Small
Constant	Fast (cached)	Medium
Texture	Fast (cached)	Medium



Memory management layer

- Threads SPMD model (shared generation and evaluation function code)
- Global Memory is not cached

  → Accesses (read/write operations) must be minimized
- Non-coalesced accesses to Global Memory

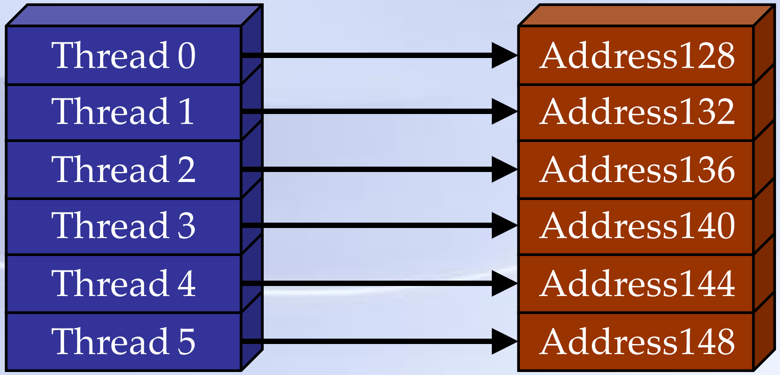
  → Use of Texture Memory

Memory coalescing

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

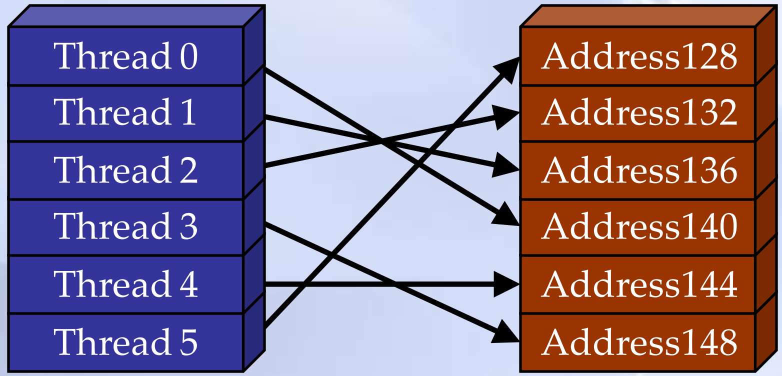
Uncoalesced accesses to global memory for evaluation functions

	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]];
}
```

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

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.

Use of Texture Memory

- Graphic cards provide also read-only texture memory to accelerate operations such as 2D or 3D mapping.
- In the case of LS algorithms, binding texture on global memory can provide an alternative optimization.

- Conditions of use
  - Read-only input data problems.
  - Read-only candidate solution for generating neighborhood.
  - Small amount of memory of input data structures to take advantage of the 8KB cache per multiprocessor of texture units.

Outline

- Parallel Local Search Metaheuristics (PLSM)
- GPU-based Design and Implementation of PLSM
- **Application to the Permuted Perceptron Problem (PPP)**
- Conclusion and Future Work

Permuted Perceptron Problem (PPP)

- An *ε-vector* (resp. *ε-matrix)* is a vector (resp. matrix) with all entries being either +1 or -1

- Definition of PPP
  - Given an *ε-matrix A* of size *m x n* and a multi-set *S* of non negative integers of size *m* …
  - find an *ε-vector V* of size *n* such that {{(AV)j / j = {1,…,m}}} = S

Illustration

- **Cryptographic identification scheme**
- **Protocol well-suited for resource constrained devices such as smart cards**

$$V = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{pmatrix}$$

Private key

$$A = \begin{pmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

Public key

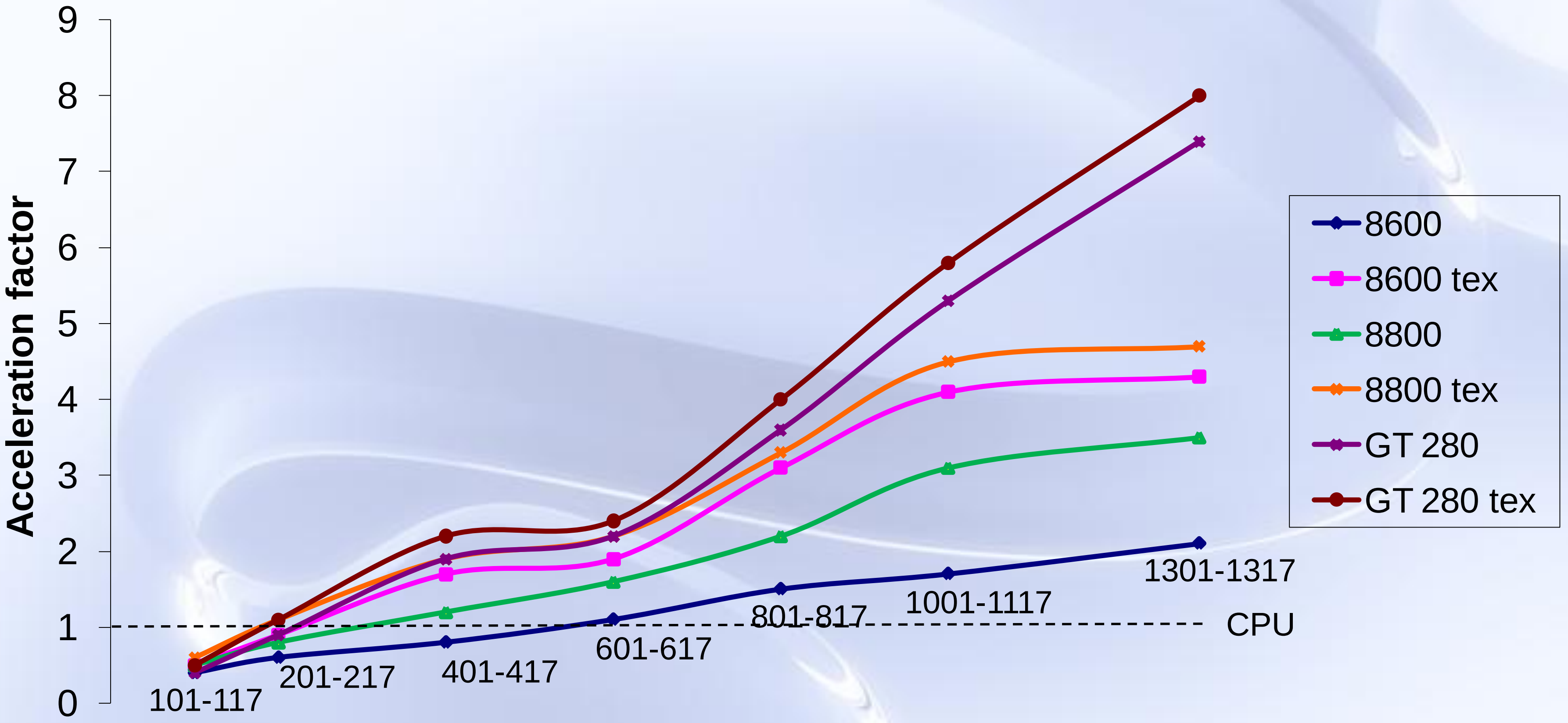
$$S = \begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$$

Message

Parameter settings

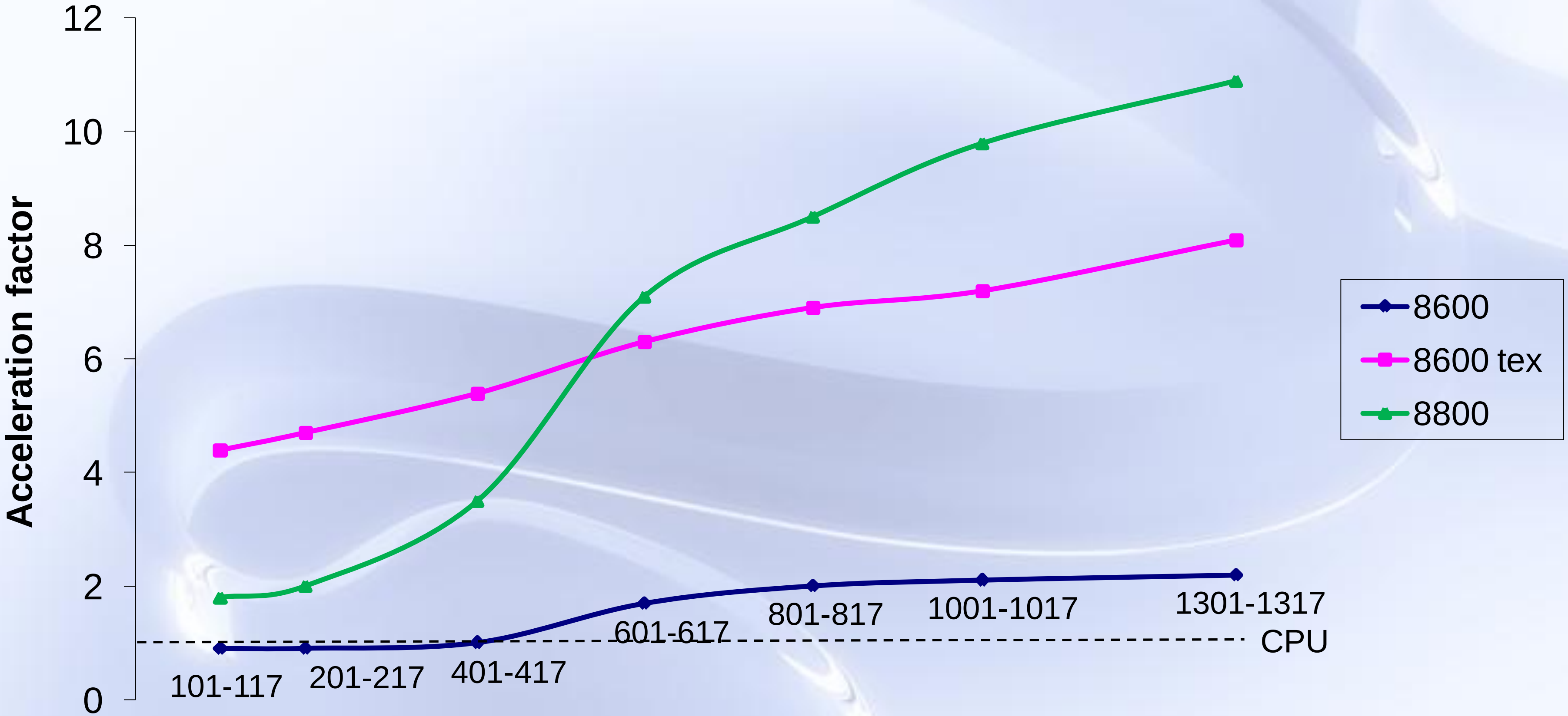
- Hardware configurations
  - Configuration 1: laptop
    - Core 2 Duo 2 Ghz + 8600M GT 4 multiprocessors (32 cores)
  - Configuration 2 : desktop-computer
    - Core 2 Quad 2.4 Ghz + 8800 GTX 16 multiprocessors (128 cores)
  - Configuration 3 : video games computer
    - Intel Xeon 3 Ghz + GTX 280 30 multiprocessors (240 cores)
- Tabu Search and PPP parameters
  - Neighborhood generation and evaluation on GPU
  - Binary representation for PPP
  - 100.000 iterations, 10 runs

Neighborhood based on a Hamming distance of one



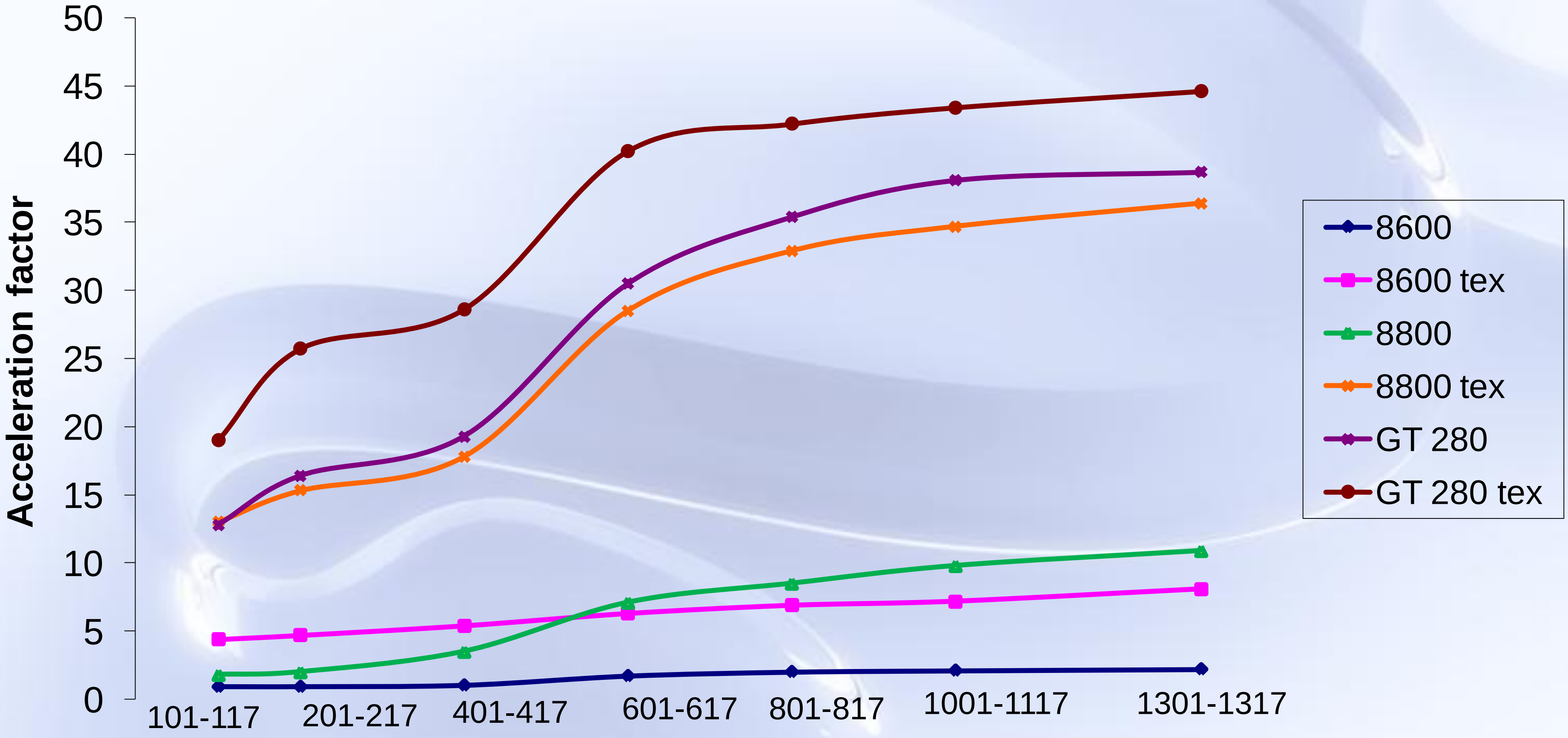
- Size of the neighborhood = size of V = *n*
- Number of threads/block is not enough to cover the memory access latency

Neighborhood based on a hamming distance of two



- Size of the neighborhood = *n x (n-1) / 2*
- Better acceleration

Neighborhood based on a Hamming distance of two



- Size of the neighborhood = *n x (n-1) / 2*
- Better acceleration

Outline

- Parallel Local Search Metaheuristics (PLSM)
- GPU-based Design and Implementation of PLSM
- Application to the Permuted Perceptron Problem (PPP)
- **Conclusion and Future Work**

Conclusion and Future Work (1)

- GPU-based LSM requires to re-design the parallel models (e.g. Iteration-level parallel model)
  - Generation of the neighborhood on the GPU side to minimize the CPU→GPU data transfer
  - If possible, thread reduction for the best solution selection to minimize the GPU→CPU data transfer
  - Efficient thread control: mapping neighboring onto threads ids, efficient kernel for fitness evaluation – incremental evaluation
  - Efficient memory management (e.g. use of texture memory)
- For problem instances with costly evaluation function and a large neighborhood set …
- … speed-ups from experiments provide promising results (up to x45 with texture memory)

Conclusion and Future Work (2)

- Extensions for LSM
  - Other problems such as TSP, QAP, Q3AP or Golomb rulers (up to x15, x20, x30 and x40)
  - Other data representations and mappings
  - Other memory and thread optimizations

- Integration of the contribution in our ParadisEO software framework (http://paradiseo.gforge.inria.fr)

Conclusion and Future Work (3)

<u>Thé Van Luong</u>, 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 (nominated for the best paper award)

Other works on GPU

Extensions of PPP

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

- Extensions for LSM
  - Larger neighborhoods
  - Other mappings between threads and neighbors
  - Measures of the effectiveness (quality of the solutions)

Problem	73 x 73	81 x 81	101 x 101	101 x 117
Fitness	10.3	10.8	20.2	16.4
# iterations	59184	77321	166650	260130
# solutions	10/50	6/50	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	16.4	15.5	14.2	13.8
# iterations	43031	67462	138349	260130
# solutions	19/50	13/50	12/50	0/50
CPU time	81 s	174 s	748 s	1947 s
GPU time	8 s	16 s	44 s	105 s
Acceleration	x 9.9	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	2.4	3.5	6.2	7.7
# iterations	21360	43231	117422	255337
# solutions	35/50	28/50	18/50	1/50
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 24.8

Neighborhood based on a Hamming distance of three

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

- Perspectives
  - Variable neighborhood search for an arbitrary number of neighborhoods
    - Issue: find a mapping between threads and neighbors …
      - … construct efficient lookup tables on global memory for mappings

Global memory
Lookup table

	move_id	move_first	move_second
T0	0	0	1
T1	1	0	2
T2	2	0	3
T3	3	1	2
T4	4	1	3
T5	5	2	3

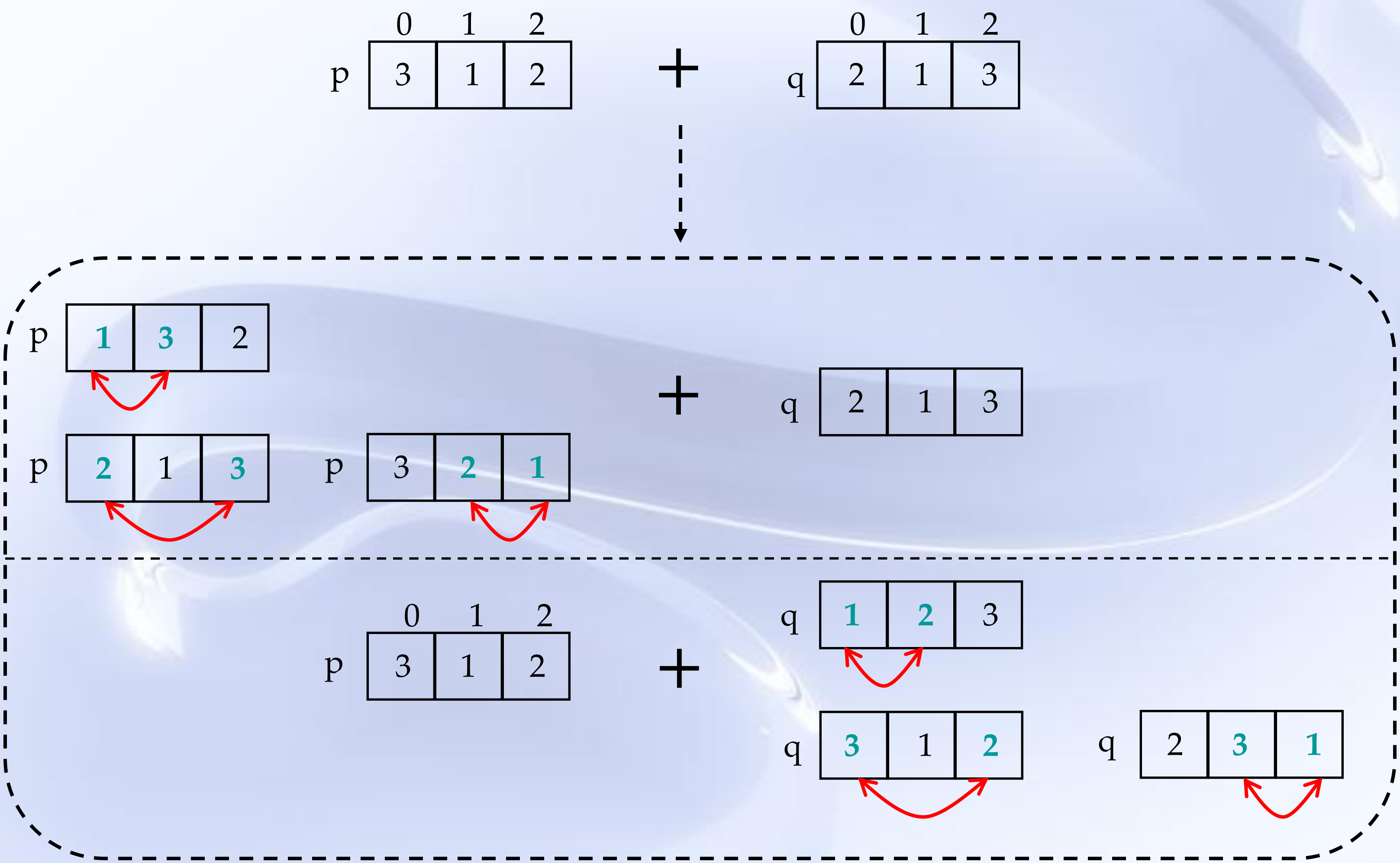
Application to the Q3AP

Thé Van Luong, Lakhdar Loukil, Nouredine Melab, El-Ghazali Talbi. **A GPU-based Iterated Tabu Search for Solving the Quadratic 3-dimensional Assignment Problem**. ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Workshop on Parallel Optimization in Emerging Computing Environments (POECE), Hammamet, Tunisia, 2010

- Extensions for LSM
  - Other data representations and mappings
  - Other memory and thread optimizations
  - The GPU allows the design of an efficient and large neighborhood

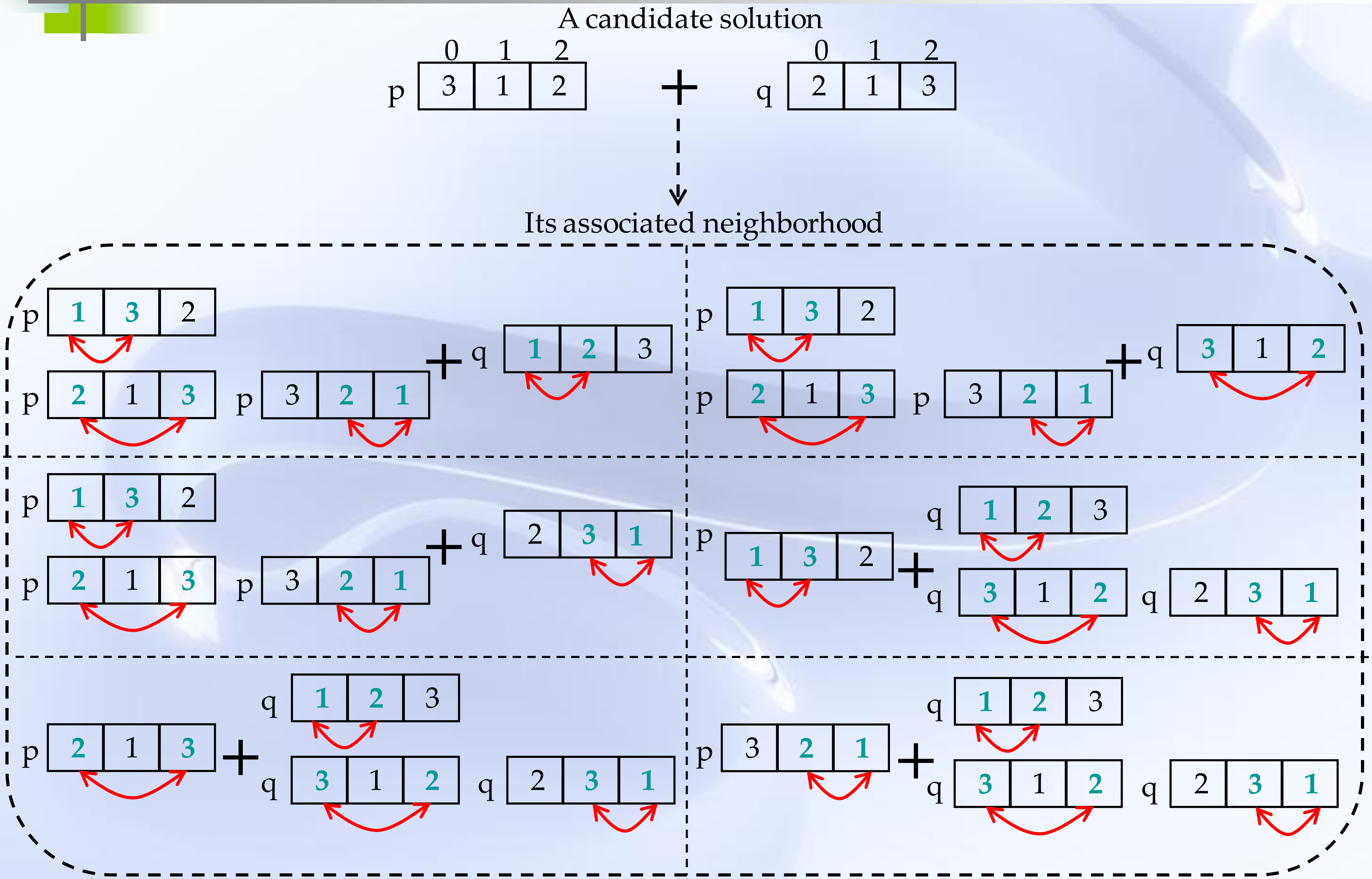
State-of-the-art neighborhood

A candidate solution

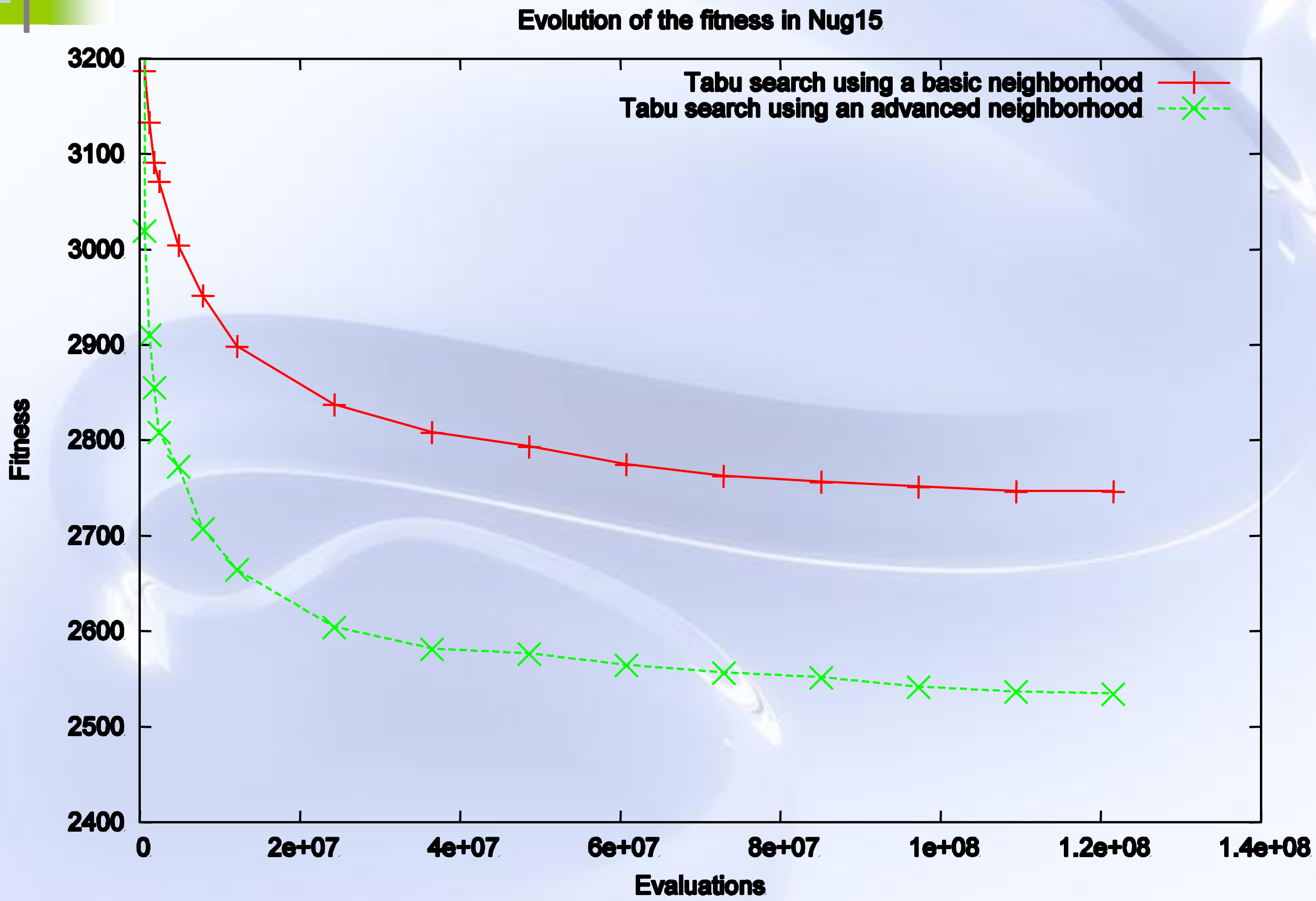


Its associated neighborhood

Advanced neighborhood



Comparison of the neighborhoods

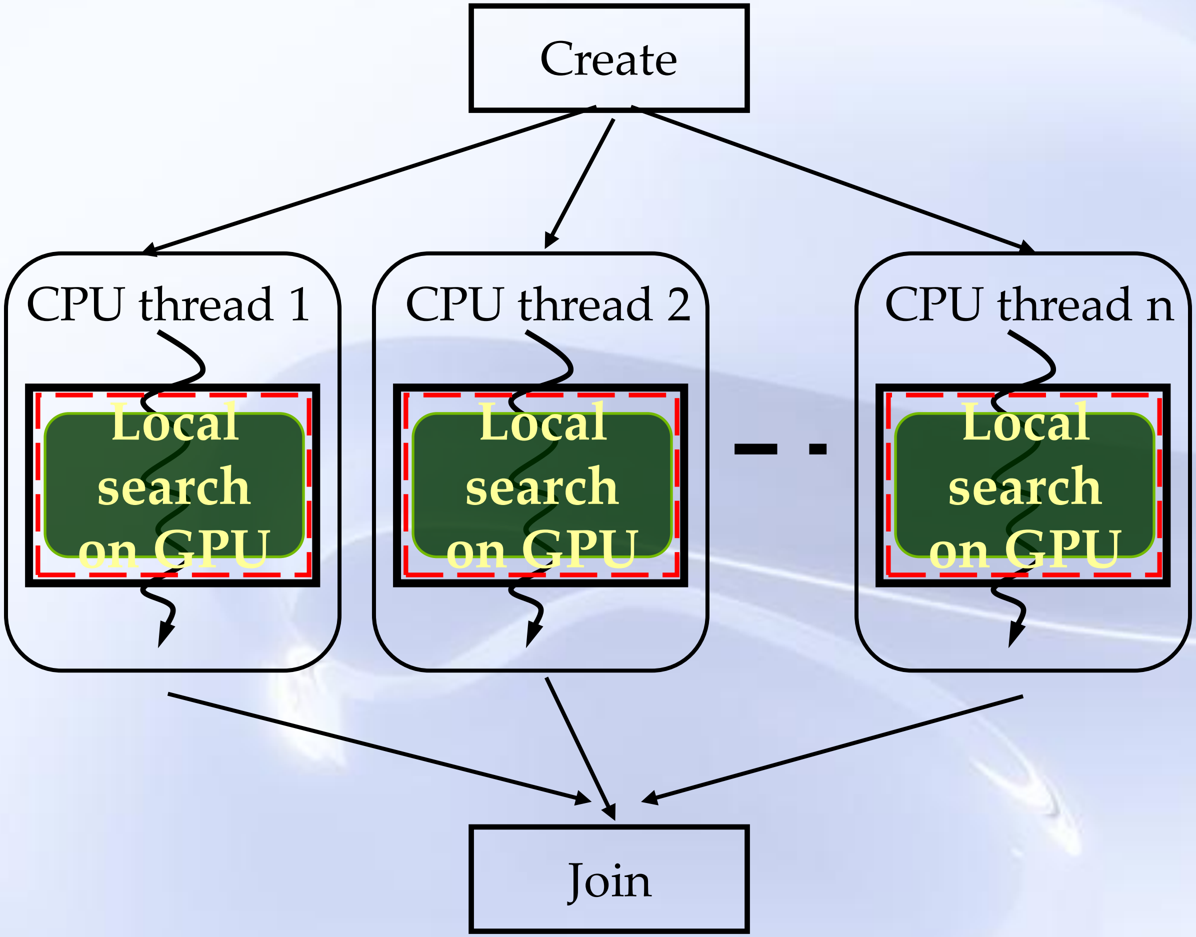


Results of Q3AP

Instance	Nug12	Nug13	Nug15	Nug18	Nug22
Best known value	**580**	**1912**	2230	17836	42476
Average value	580	1918	2230	17874	42476
Max value	604	1974	2230	18026	42476
# solutions	49/50	37/50	50/50	31/50	50/50
CPU time	256 s	1879 s	1360 s	17447 s	16147 s
GPU time	15 s	64 s	38 s	415 s	353 s
Acceleration	x 17.3	x 29.2	x 36.0	x 42.0	x 45.7
ILS iteration	18	57	15	59	15

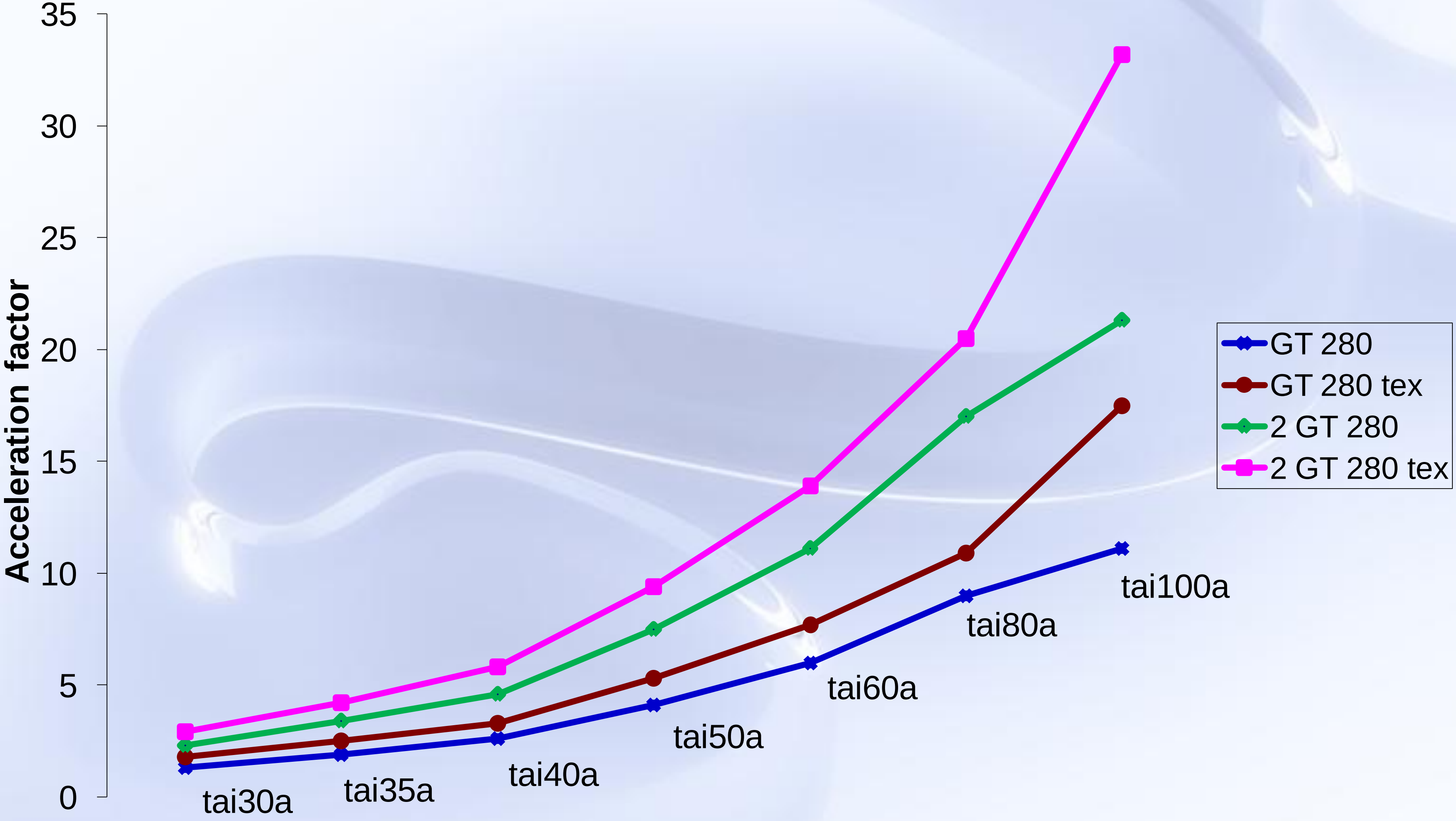
- Iterative local search (100 iters) + tabu search (5000 iters)
- Competitive algorithm
- Unpractical on CPU

Algorithmic-level: multi-GPUs

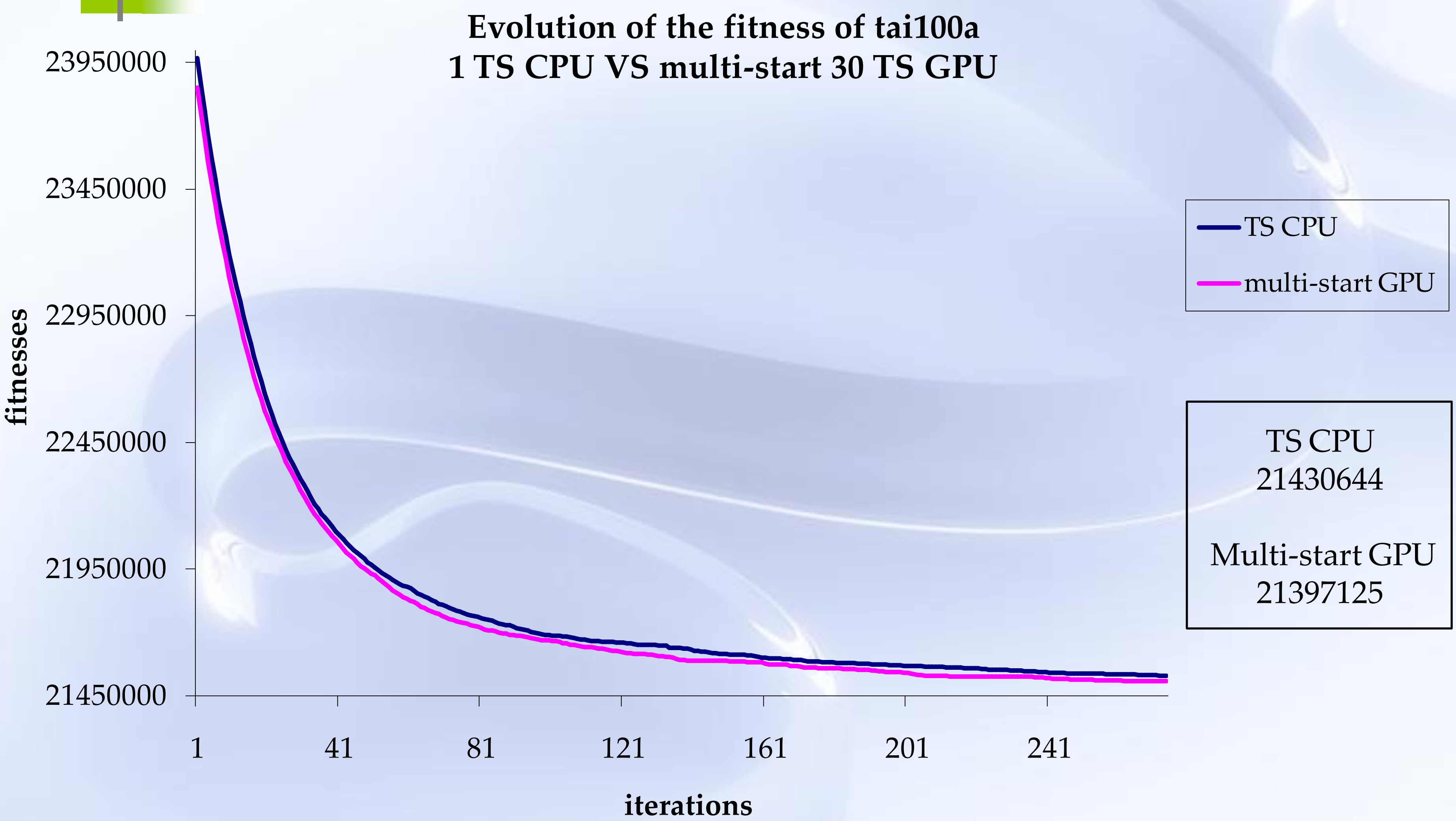


- Multi-core: OpenMP and posix threads
- Distributed: MPI

Measures of the efficiency on the quadratic assignment problem
20 multi-start tabu search – 100000 iterations



Measures of the effectiveness on the quadratic assignment problem



TS CPU
21430644

Multi-start GPU
21397125

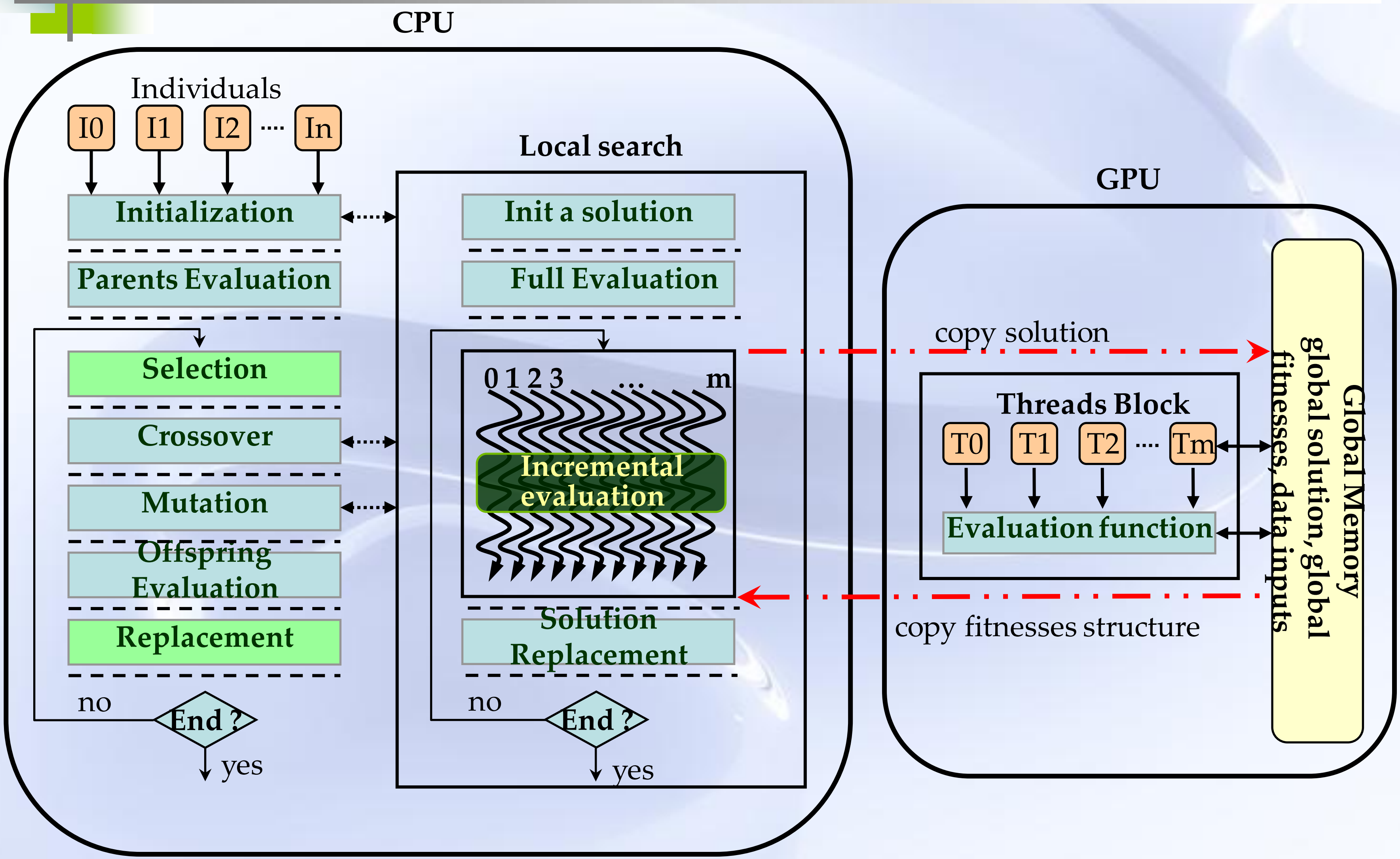
- Perspectives
  - To be submitted …
  - Need a cluster of GPUs for testing MPI experiments
  - Full distribution of the algorithmic-level on GPU
    - one GPU thread = one local search (hill-climbing, simulated annealing, …)
    - Issue for the tabu search algorithm (management of the tabu list on GPU)

Hybrid evolutionary algorithms

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

- Extensions for LSM
  - Combination of local searches and evolutionary algorithms
  - The GPU allows to design sophisticate algorithms

Hybridization scheme

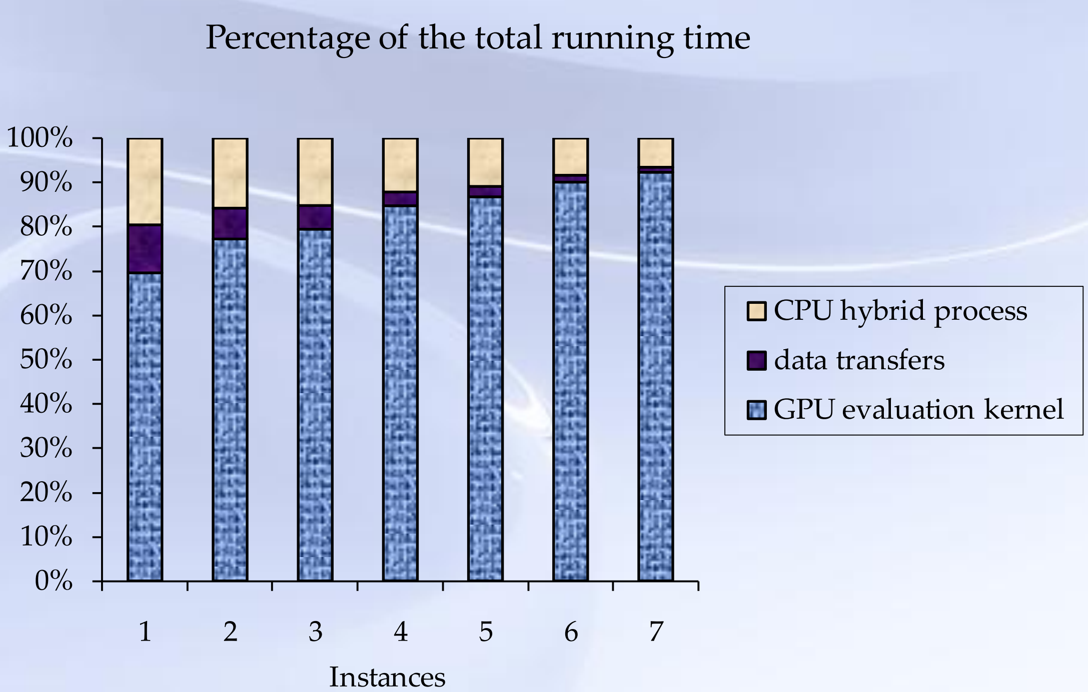


Results on QAP

Instance	tai30a	tai35a	tai40a	tai50a	tai60a	tai80a	tai100a
Best known value	**1818146**	**2522002**	3139370	4938796	7205962	13511780	21052466
Average value	1818442	2422437	3146480	4961202	7241224	13605896	21190794
# solutions	27/30	23/30	18/30	10/30	6/30	4/30	2/30
CPU time	1h15min	2h24min	3h54min	10h2min	20h17min	66h	177h
GPU time	8min50s	12min56s	18min16s	45min	1h30min	4h45min	12h6min
Acceleration	x 8.5	x 11.1	x 12.8	x 13.2	x 13.4	x 13.8	x 14.6

- 10 individuals – 10 generations
- Evolutionary algorithm + iterative local search (3 iters) + tabu search (10000 iters)
- Neighborhood based on a 3-exchange operator
- Competitive algorithm

- Perspectives
  - Full distribution of the hybrid evolutionary algorithm on GPU
    - Issue for the tabu search algorithm (management of the tabu list on GPU)
    - Does it worth parallelizing ?

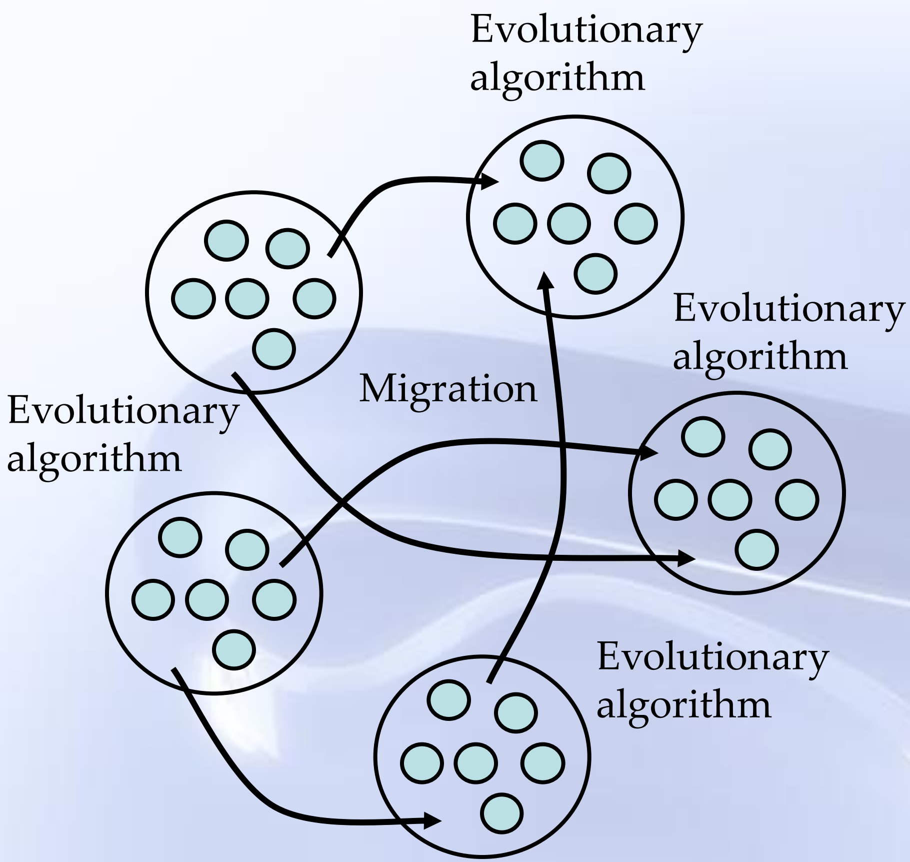


GPU-based island model for EAs

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

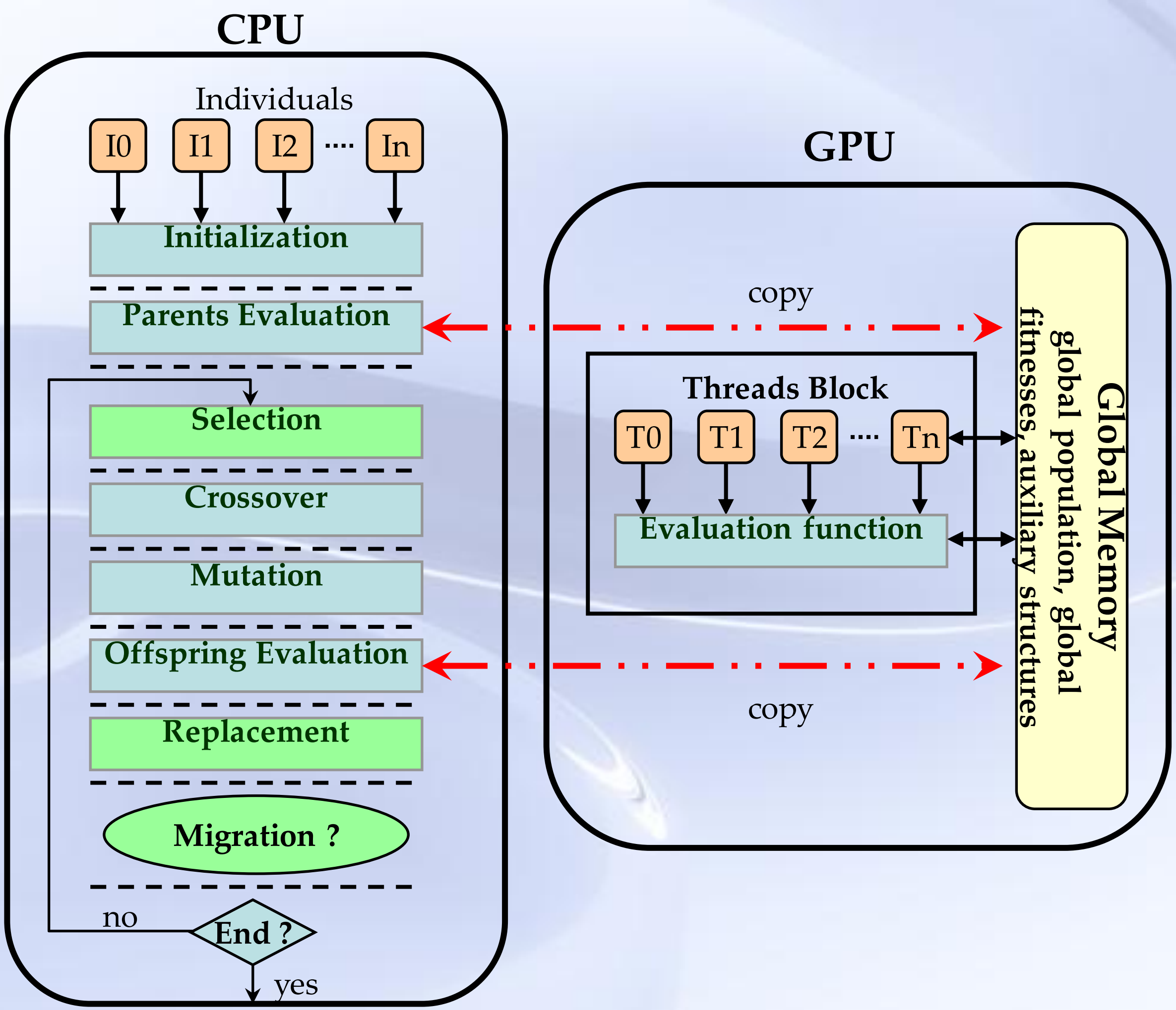
- Extensions for EAs
  - 3 schemes of the island model for evolutionary algorithms
  - EAs well-suited for continous optimization problems

Island model for EAs

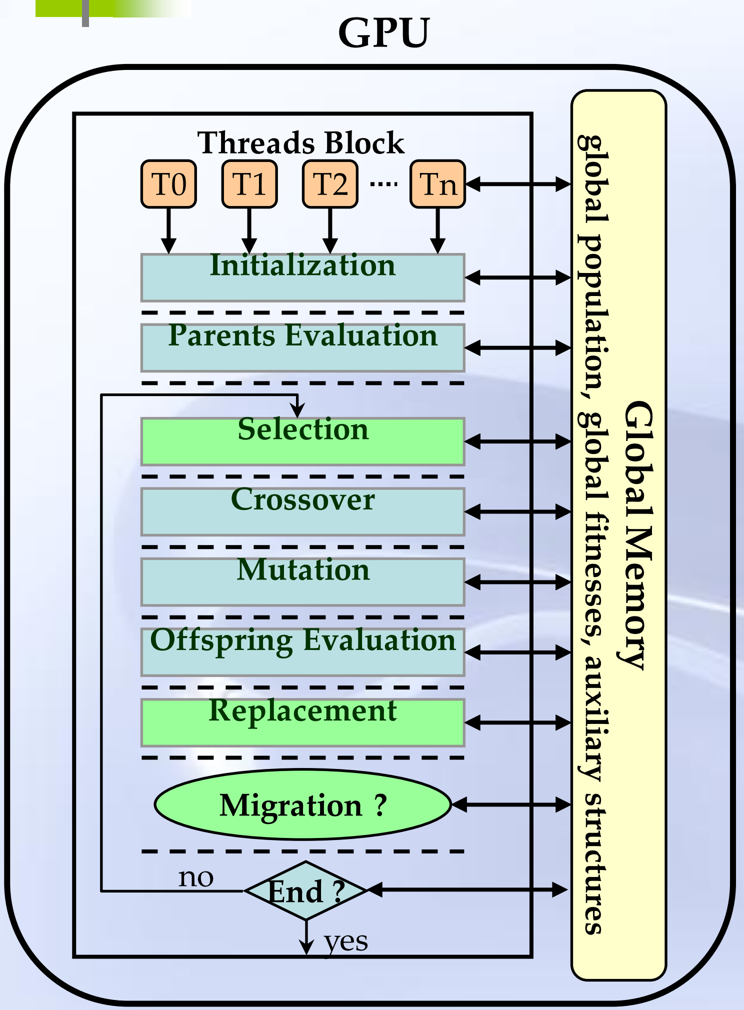


- Need to re-design on GPU:
  - Exchange topology
  - Emigrants selection policy
  - Replacement/Integration policy
  - Migration decision criterion

Parallel evaluation of each island

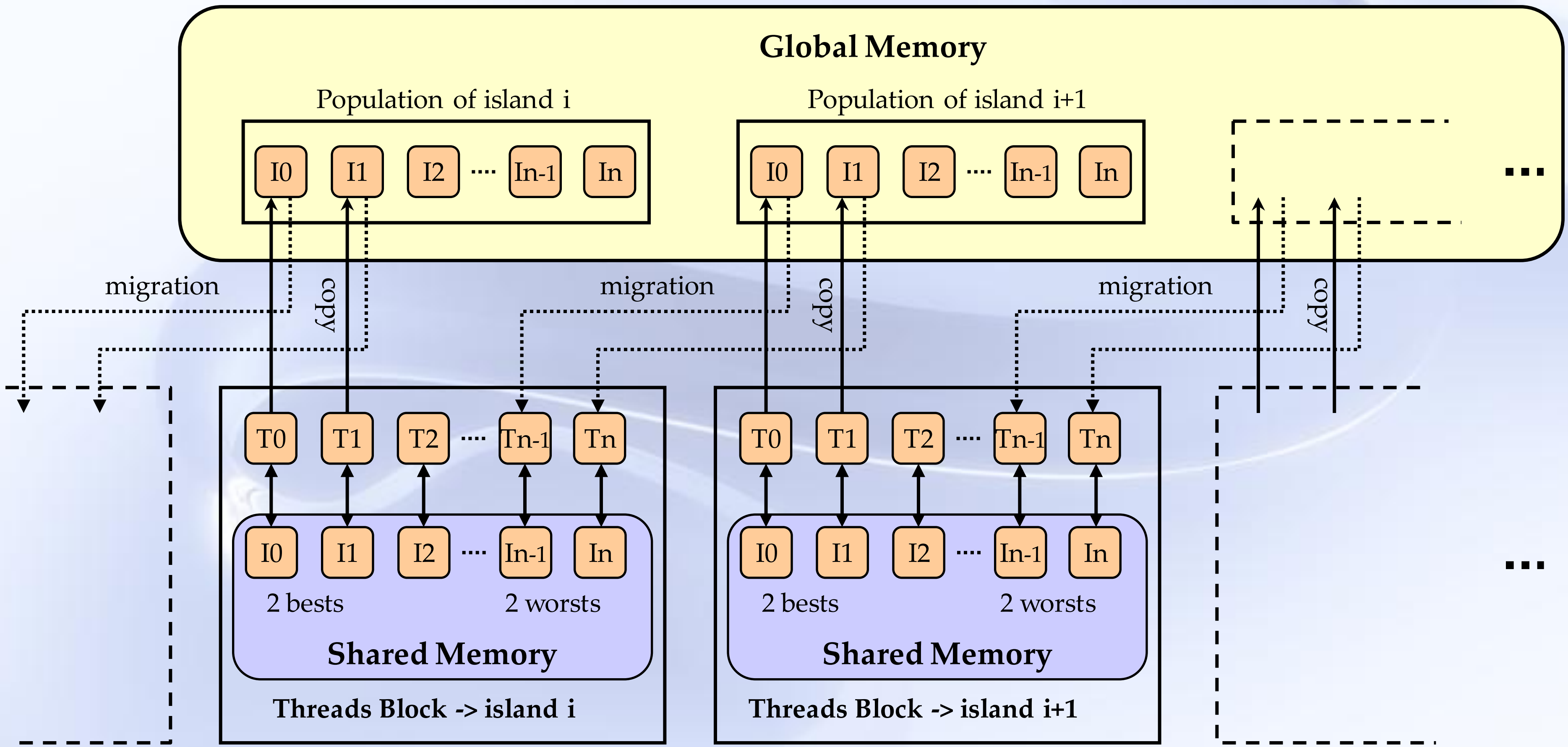


Full distribution on GPU



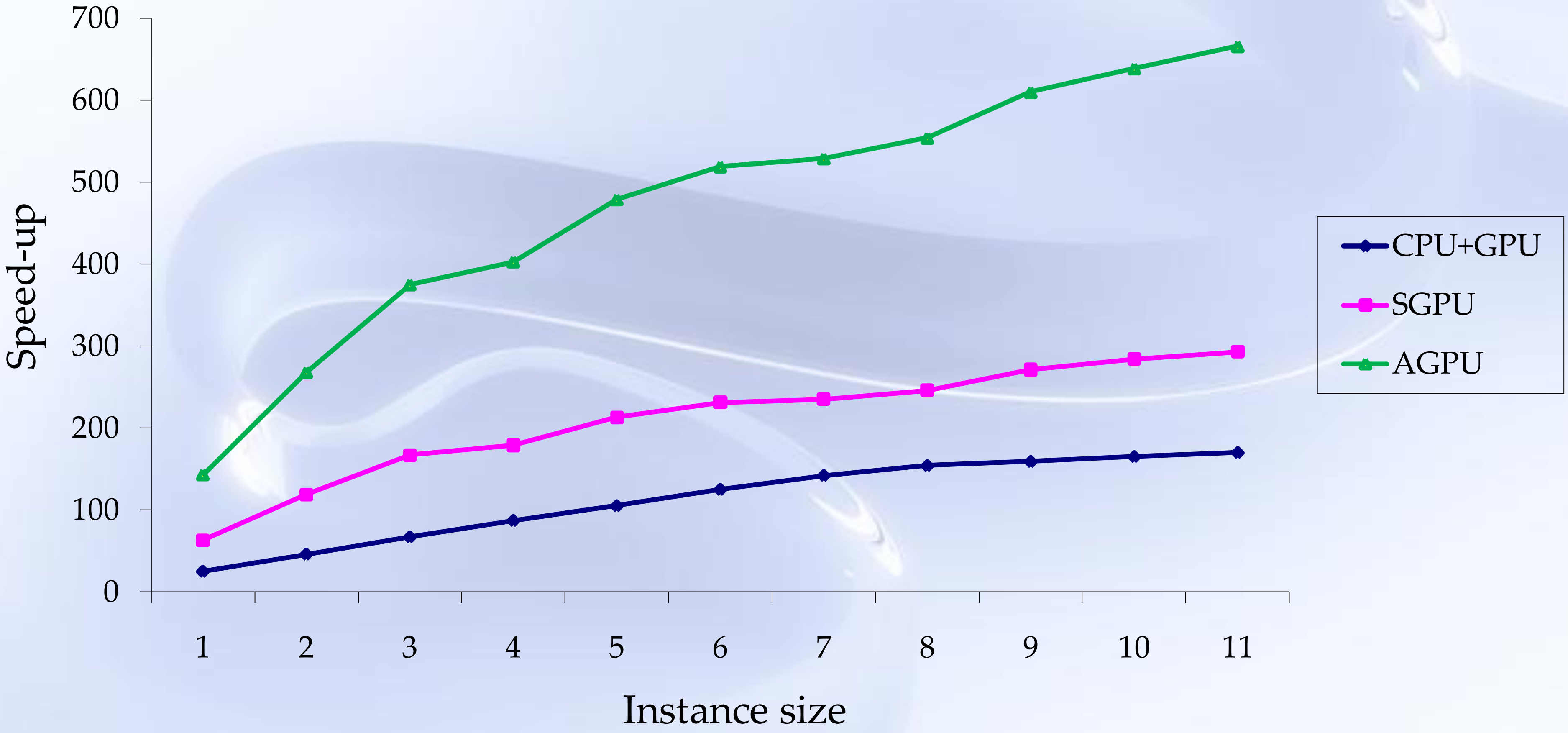
- One threads block represents one island
- Possible issues
  - Sort the population of each island
  - Find the minimum of the population of each island
  - Threads synchronization (synchronous migration)
  - Generation of random numbers

Migration on GPU



Results for the Weierstrass function (1)

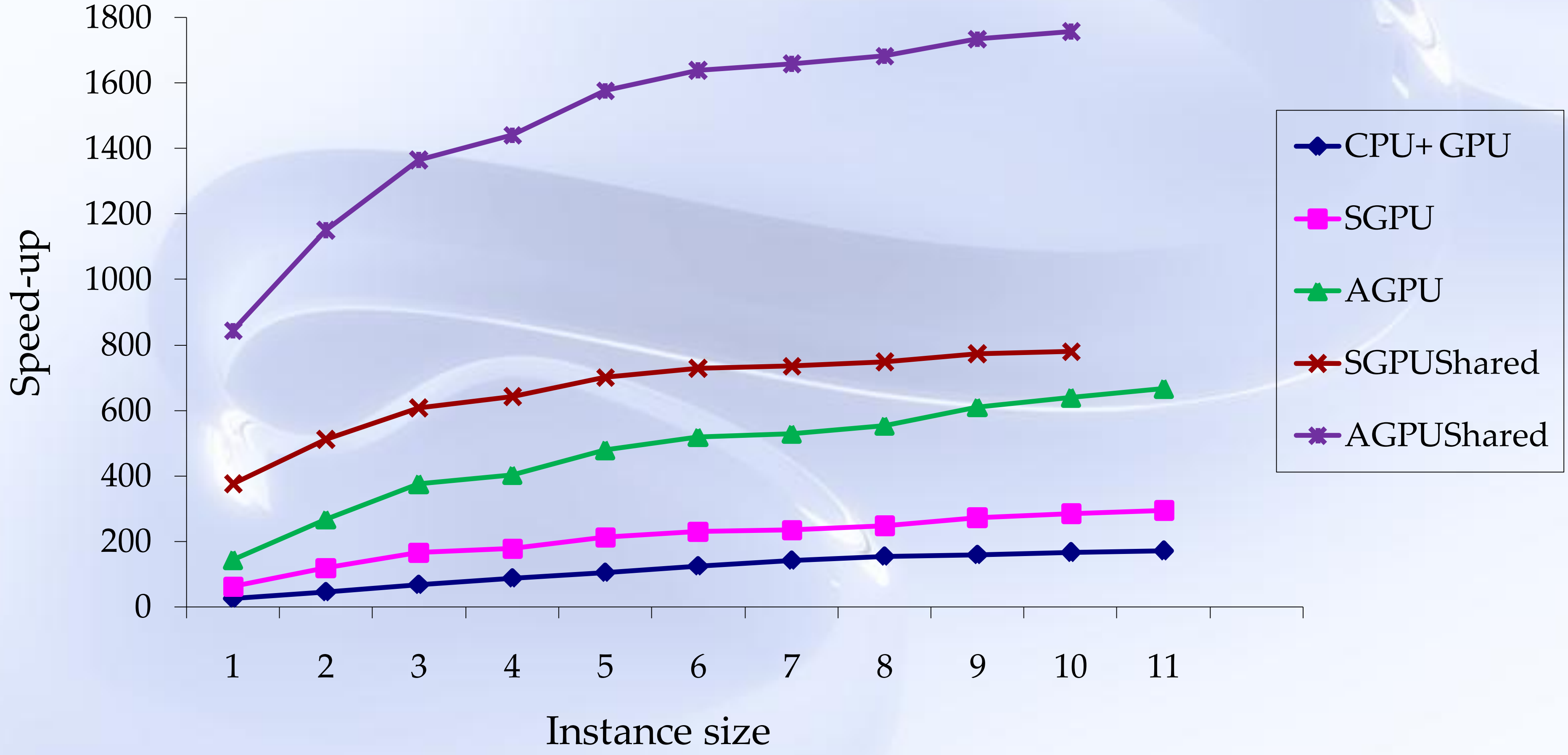
Varying the dimension of the problem
(64 islands – 128 individuals per island)



Results for the Weierstrass function (1)

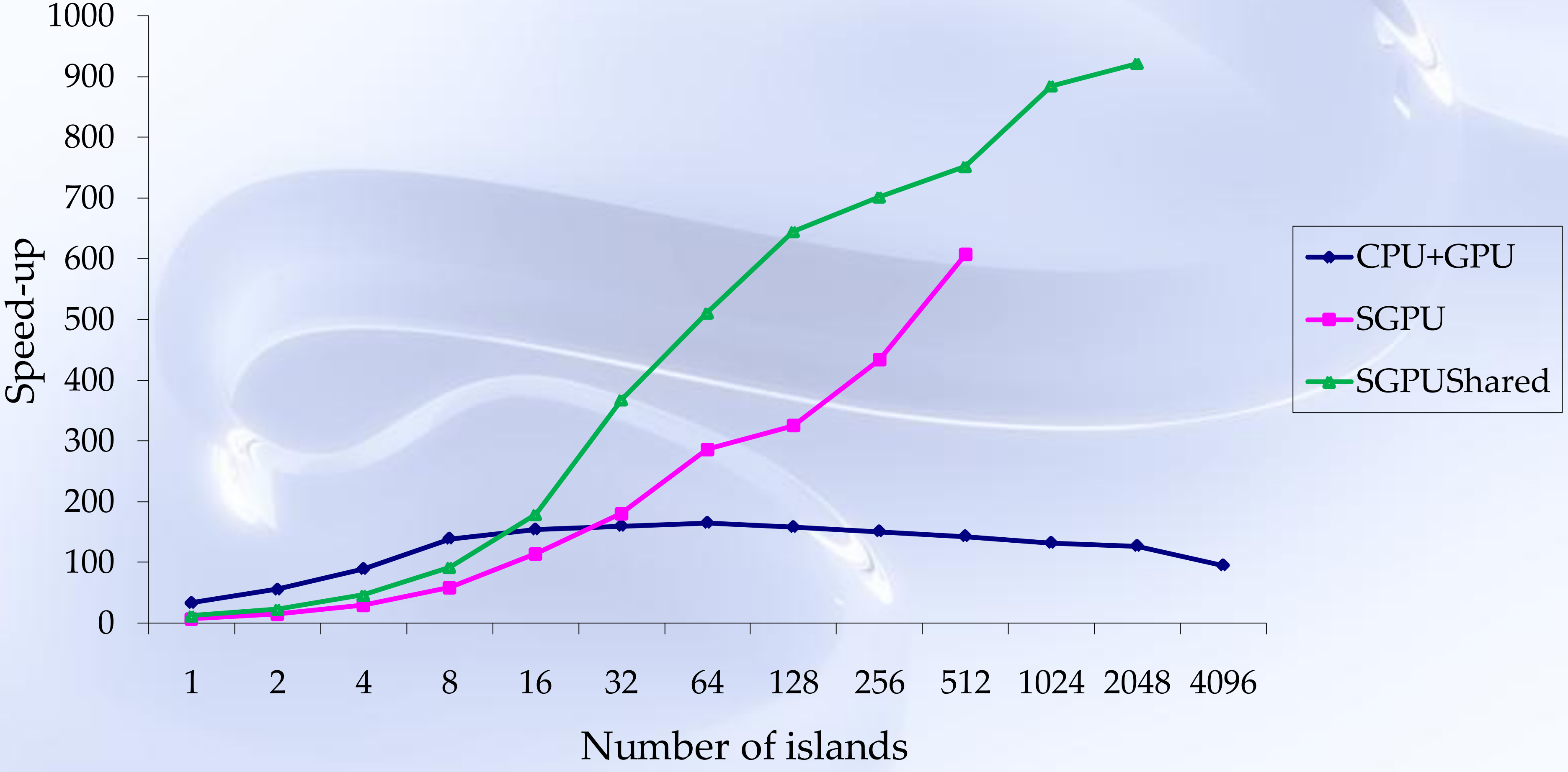
Varying the dimension size of the problem

(64 islands – 128 individuals per island)



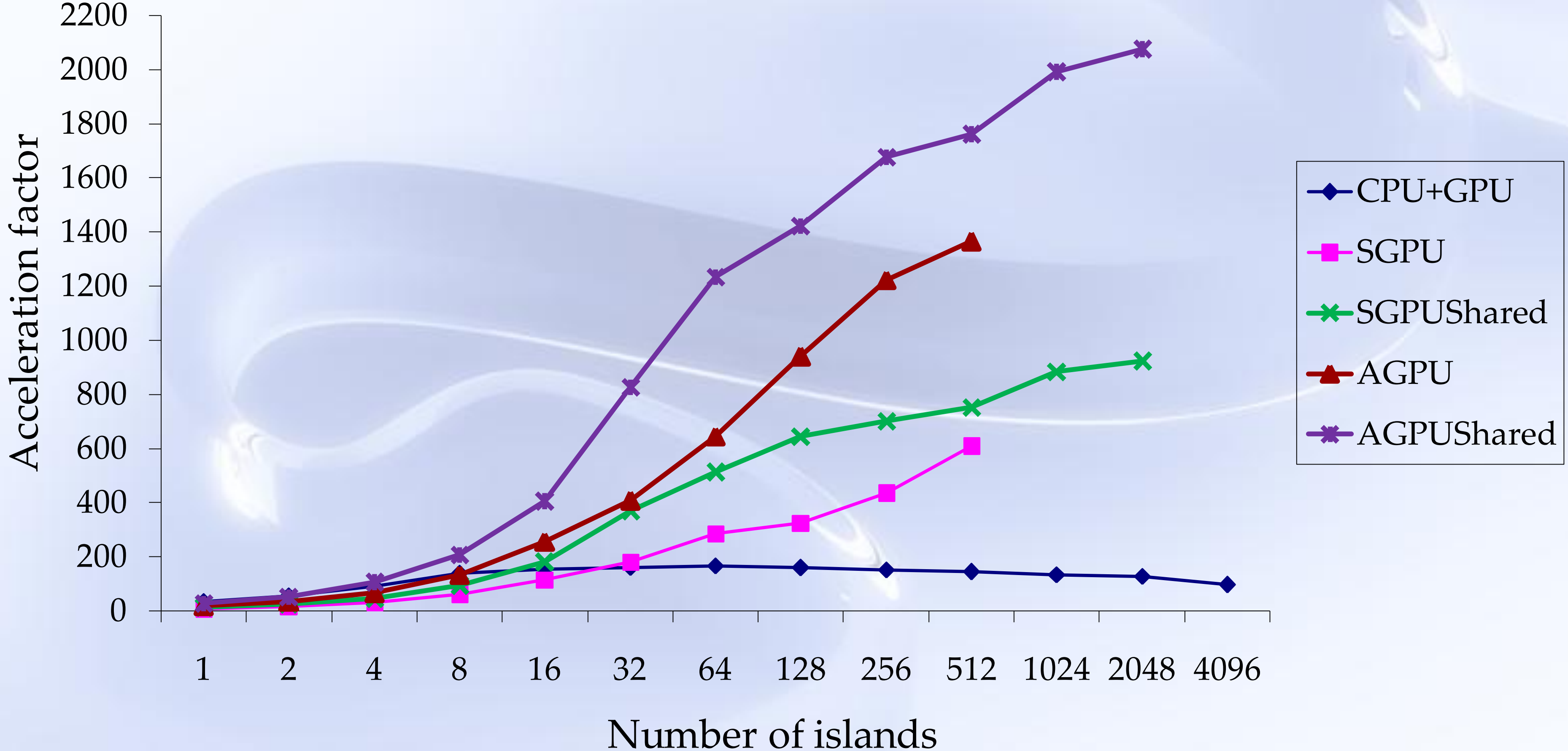
Results for the Weierstrass function (2)

Varying the number of islands
(dimension of the problem: 2 – 128 individuals per island)



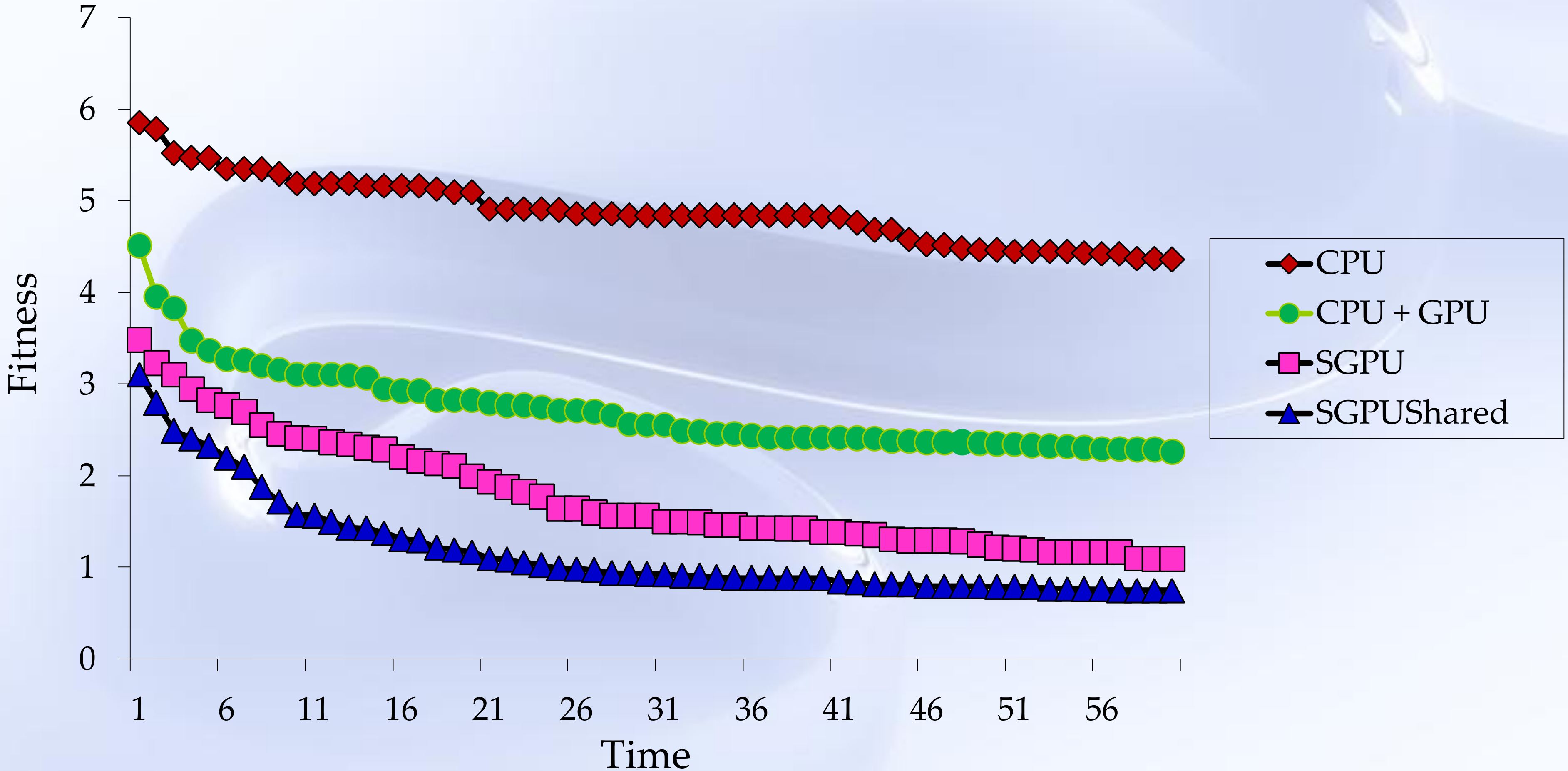
Results for the Weierstrass function (2)

Varying the number of islands
(dimension of the problem: 2 – 128 individuals per island)



Weierstrass-Mandelbrot function (3)

Measures of the quality of the solutions
(dimension of the problem: 10 – 64 islands – 128 individuals per island)



Pros and cons

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	Homogeneous	Limited to shared memory	Limited to shared memory	Medium	Lightning Fast

- Perspectives
  - Define sophisticate island topologies
  - Multi-GPU approach for the island model for EAs
  - Extension of the island model for estimation of distribution algorithm (EDA) and particle swarm optimization (PSO)

THANK YOU FOR YOUR ATTENTION