Metaheuristics on GPU

Thé Van Luong, Nouredine Melab and El-Ghazali Talbi

DOLPHIN Project Team

April 2010
Local search on GPU: From design to implementation
Outline

- Parallel Local Search Metaheuristics (PLSM)
- GPU-based Design and Implementation of PLSM
- Application to the Permutated Perceptron Problem (PPP)
- Conclusion and Future Work
A taxonomy of optimization methods

Exact Algorithms

- Branch and X
- Dynamic programming

Heuristics

- CP
- Specific heuristics

Metaheuristics

- Solution-based
- Population-based

- Hill Climbing
- Simulated Annealing
- Tabu Search
- Evolutionary Algorithms
- Ant Colony

Exploitation-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

Algorithmic-level parallel model

Solution-level parallel model

\[ f(s_1) \]
\[ f(s_2) \]
\[ f(s_3) \]
\[ f(s_n) \]
\[ f_1(s_n) \]
\[ f_m(s_n) \]
Iteration-level parallel model

1. Generate a solution
2. Full evaluation
3. Select a neighbor of the solution
   - Evaluation
     - Yes: Next neighbor?
     - No: Replace the solution by the chosen neighbor
4. STOP?
   - Yes: END
   - No: Evaluate nodes

- 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

- Generate a solution
- Full evaluation
- Select a neighbor of the solution
- Evaluation
  - Yes: Next neighbor?
  - No: Replace the solution by the selected neighbor
- No: STOP?
  - Yes: END

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

n neighbors

Approach 2

n(n-1)/2 neighbors
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 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

n neighbors

Approach 2

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
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
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
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
Finding a mapping can be challenging

Neighborhood based on a Hamming distance of two

- A thread \( id \) is associated with two indexes \( i \) and \( j \)
- \( n \times (n-1) / 2 \) threads are generated for a solution of size \( n \)
- Fitness data structure size = \( n \times (n-1) / 2 \)
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

<table>
<thead>
<tr>
<th>Memory type</th>
<th>Speed</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>Slow</td>
<td>Big</td>
</tr>
<tr>
<td>Registers</td>
<td>Very fast</td>
<td>Very small</td>
</tr>
<tr>
<td>Local</td>
<td>Slow</td>
<td>Up to Global memory</td>
</tr>
<tr>
<td>Shared</td>
<td>Fast</td>
<td>Small</td>
</tr>
<tr>
<td>Constant</td>
<td>Fast (cached)</td>
<td>Medium</td>
</tr>
<tr>
<td>Texture</td>
<td>Fast (cached)</td>
<td>Medium</td>
</tr>
</tbody>
</table>

## GPU

![GPU diagram with memory types and connections](image)

- **Block 0**
  - Shared Memory
  - Registers
  - Thread 0
  - Local Memory
  - Thread 1
  - Local Memory

- **Block 1**
  - Shared Memory
  - Registers
  - Thread 0
  - Local Memory
  - Thread 1
  - Local Memory

## CPU

- Global Memory (Data inputs and current solution representation)
- Constant Memory
- Texture Memory
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]
```

SIMD: 1 memory transaction
Memory coalescing

Uncoalesced accesses to global memory for evaluation functions

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
p & 3 & 2 & 1 & 5 & 4 & 0 \\
\end{array}
\]

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]

6 memory transactions

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 $\varepsilon$-vector (resp. $\varepsilon$-matrix) is a vector (resp. matrix) with all entries being either +1 or -1

- Definition of PPP
  - Given an $\varepsilon$-matrix $A$ of size $m \times n$ and a multi-set $S$ of non negative integers of size $m$ ...
  - find an $\varepsilon$-vector $V$ of size $n$ such that $\{(AV)_j / j = \{1,\ldots,m\}\} = S$
• Cryptographic identification scheme
• Protocol well-suited for resource constrained devices such as smart cards

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>v₁</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v₂</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v₃</td>
<td></td>
</tr>
<tr>
<td></td>
<td>v₄</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Public key

Private key

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
- Size of the neighborhood = size of $V = n$
- Number of threads/block is not enough to cover the memory access latency
- Size of the neighborhood = $n \times (n-1) / 2$
- Better acceleration
- Size of the neighborhood = \( n \times (n-1) / 2 \)
- 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 thread 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)

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)
<table>
<thead>
<tr>
<th>Problem</th>
<th>73 x 73</th>
<th>81 x 81</th>
<th>101 x 101</th>
<th>101 x 117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness</td>
<td>10.3</td>
<td>10.8</td>
<td>20.2</td>
<td>16.4</td>
</tr>
<tr>
<td># iterations</td>
<td>59184</td>
<td>77321</td>
<td>166650</td>
<td>260130</td>
</tr>
<tr>
<td># solutions</td>
<td>10/50</td>
<td>6/50</td>
<td>0/50</td>
<td>0/50</td>
</tr>
<tr>
<td>CPU time</td>
<td>4 s</td>
<td>6 s</td>
<td>16 s</td>
<td>29 s</td>
</tr>
<tr>
<td>GPU time</td>
<td>9 s</td>
<td>13 s</td>
<td>33 s</td>
<td>57 s</td>
</tr>
<tr>
<td>Acceleration</td>
<td>x 0.44</td>
<td>x 0.46</td>
<td>x 0.48</td>
<td>x 0.51</td>
</tr>
</tbody>
</table>

Neighborhood based on a Hamming distance of one
Tabu search
n x (n-1) x (n-2) / 6 iterations

<table>
<thead>
<tr>
<th>Problem</th>
<th>73 x 73</th>
<th>81 x 81</th>
<th>101 x 101</th>
<th>101 x 117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness</td>
<td>16.4</td>
<td>15.5</td>
<td>14.2</td>
<td>13.8</td>
</tr>
<tr>
<td># iterations</td>
<td>43031</td>
<td>67462</td>
<td>138349</td>
<td>260130</td>
</tr>
<tr>
<td># solutions</td>
<td>19/50</td>
<td>13/50</td>
<td>12/50</td>
<td>0/50</td>
</tr>
<tr>
<td>CPU time</td>
<td>81 s</td>
<td>174 s</td>
<td>748 s</td>
<td>1947 s</td>
</tr>
<tr>
<td>GPU time</td>
<td>8 s</td>
<td>16 s</td>
<td>44 s</td>
<td>105 s</td>
</tr>
<tr>
<td>Acceleration</td>
<td>x 9.9</td>
<td>x 11.0</td>
<td>x 17.0</td>
<td>x 18.5</td>
</tr>
</tbody>
</table>

Neighborhood based on a Hamming distance of two
Tabu search
n x (n-1) x (n-2) / 6 iterations

<table>
<thead>
<tr>
<th>Problem</th>
<th>73 x 73</th>
<th>81 x 81</th>
<th>101 x 101</th>
<th>101 x 117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fitness</td>
<td>2.4</td>
<td>3.5</td>
<td>6.2</td>
<td>7.7</td>
</tr>
<tr>
<td># iterations</td>
<td>21360</td>
<td>43231</td>
<td>117422</td>
<td>255337</td>
</tr>
<tr>
<td># solutions</td>
<td>35/50</td>
<td>28/50</td>
<td>18/50</td>
<td>1/50</td>
</tr>
<tr>
<td>CPU time</td>
<td>1202 s</td>
<td>3730 s</td>
<td>24657 s</td>
<td>88151 s</td>
</tr>
<tr>
<td>GPU time</td>
<td>50 s</td>
<td>146 s</td>
<td>955 s</td>
<td>3551 s</td>
</tr>
<tr>
<td>Acceleration</td>
<td>x 24.2</td>
<td>x 25.5</td>
<td>x 25.8</td>
<td>x 24.8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>move_id</th>
<th>move_first</th>
<th>move_second</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
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

\[
\begin{array}{ccc}
0 & 1 & 2 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 1 & 2 \\
q & 2 & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 3 & 2 \\
p & \rightarrow & 1 & 3 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 1 & 3 \\
p & \rightarrow & 3 & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 1 & 2 \\
p & \rightarrow & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
q & \rightarrow & 1 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
3 & 1 & 2 \\
q & \rightarrow & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 3 & 1 \\
q & \rightarrow & 2 & 3 & 1 \\
\end{array}
\]

Its associated neighborhood
**Advanced neighborhood**

A candidate solution

\[
\begin{array}{ccc}
0 & 1 & 2 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 1 & 2 \\
q & 2 & 1 & 3 \\
\end{array}
\]

Its associated neighborhood

\[
\begin{array}{ccc}
1 & 3 & 2 \\
p & 1 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 1 & 3 \\
p & 3 & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 3 & 2 \\
p & 3 & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 3 & 2 \\
p & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 1 & 3 \\
p & 2 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 3 & 2 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
p & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
3 & 1 & 2 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 3 & 1 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
p & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 3 & 1 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 3 & 1 \\
p & 2 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
p & 2 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 3 & 1 \\
p & 1 & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
2 & 3 & 1 \\
p & 3 & 1 & 2 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
p & 3 & 1 & 2 \\
\end{array}
\]
Comparison of the neighborhoods

Evolution of the fitness in Nug15

Tabu search using a basic neighborhood
Tabu search using an advanced neighborhood

Fitness

Evaluations
## Results of Q3AP

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

- 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

Evolution of the fitness of tai100a
1 TS CPU VS multi-start 30 TS GPU

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)
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

CPU

- Initialization
- Parents Evaluation
- Selection
- Crossover
- Mutation
- Offspring Evaluation
- Replacement

GPU

- Incremental evaluation
- Solution Replacement
- Incremental evaluation
- Global Memory
- Threads Block
- Evaluation function
- Global solution, global fitnesses, data inputs

- Init a solution
- Full Evaluation
- Copy solution
- Copy fitnesses structure

End

I₀ I₁ I₂ ... Iₙ

Init a solution

End

0 1 2 3 ... m

T₀ T₁ T₂ ... Tₘ

Global Memory
## Results on QAP

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

- 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?

Percentage of the total running time

- CPU hybrid process
- Data transfers
- GPU evaluation kernel
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 continuous 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

**CPU**
- Initialization
- Parents Evaluation
- Selection
- Crossover
- Mutation
- Offspring Evaluation
- Replacement
- Migration ?

**GPU**
- Global Memory
  - Evaluation function
  - Threads Block
    - Global population, global fitnesses, auxiliary structures

**Evaluation function**
- I0
- I1
- I2
- In

**Global Memory**
- T0
- T1
- T2
- Tn

- **CPU**
  - Individuals
    - I0
    - I1
    - I2
    - In

- **End ?**
  - no
  - yes
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

Population of island $i$

Population of island $i+1$

Shared Memory

Threads Block $\rightarrow$ island $i$

Shared Memory

Threads Block $\rightarrow$ island $i+1$

Global Memory

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
(dimensions 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters</th>
<th>Limitation of the local population size</th>
<th>Limitation of the instance size</th>
<th>Limitation of the total population size</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Heterogeneous</td>
<td>Not limited</td>
<td>Very Low</td>
<td>Very Low</td>
<td>Slow</td>
</tr>
<tr>
<td>CPU+GPU</td>
<td>Heterogeneous</td>
<td>Not limited</td>
<td>Low</td>
<td>Low</td>
<td>Fast</td>
</tr>
<tr>
<td>GPU</td>
<td>Homogeneous</td>
<td>Size of a threads block</td>
<td>Low</td>
<td>Medium</td>
<td>Very Fast</td>
</tr>
<tr>
<td>GPU Shared</td>
<td>Homogeneous</td>
<td>Limited to shared memory</td>
<td>Limited to shared memory</td>
<td>Medium</td>
<td>Lightning Fast</td>
</tr>
</tbody>
</table>
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