Solving Combinatorial Problems on HPC with BOBPP

Tarek Menouer, Bertrand Le Cun and Pascal Vander-Swalmen

University of Versailles Saint-Quentin-en-Yvelines, France
PRiSM laboratory

November 16, 2012
WOLFHPC 2012
Outline

1. Scientific Context
2. Parallel Frameworks for Exact Methods
3. Experiments
4. Conclusion and Perspectives
Outline

1 Scientific Context

2 Parallel Frameworks for Exact Methods

3 Experiments

4 Conclusion and Perspectives
Combinatorial Optimization

- Find an assignment to some variables such that
  - The value of a certain function is minimized (or maximized)
  - Satisfying some constraints

Academic examples

- Vehicle Routing Problem
- Knapsack Problem
- Travelling Salesman Problem
Combinatorial Optimization Problems

Combinatorial Explosion: The Travelling Salesman Problem

- Brut Force: TSP, 20 cities, 20! tours, $10^{-9}$ s/tour $= 77.14$ years
- Parallelization with 1,000 cores and 50% efficiency same problem: $10^{-9}$ seconds/tour $= 56.31$ days

Combinatorial Explosion

- Operational Research (OR)
  Use OR methods to reduce the size of search space
  Evaluation/Prunning of partial solutions
- Parallelization
  Parallelization to solve quicker the problem with greater size

Exact resolution up to 15112 cities

- Specific Parallel Branch & Cut: Concorde
- 22.6 years equivalent sequential time by Applegate, Bixby, Chvátal and Cook
Combinatorial Optimization Problems

Combinatorial Explosion: The Travelling Salesman Problem

- Brut Force: TSP, 20 cities, 20! tours, \(10^{-9}\) s/tour = 77.14 years
- Parallelization with 1,000 cores and 50 % efficiency same problem: \(10^{-9}\) seconds/tour = 56,31 days

Combinatorial Explosion

- Operational Research (OR)
  Use OR methods to reduce the size of search space
  Evaluation/Prunning of partial solutions
- Parallelization
  Parallelization to solve quicker the problem with greater size

Exact resolution up to 15112 cities

- Specific Parallel Branch & Cut: Concorde
- 22.6 years equivalent sequential time by Applegate, Bixby, Chvátal and Cook
## Combinatorial Optimization Problems

### Combinatorial Explosion: The Travelling Salesman Problem
- Brut Force: TSP, 20 cities, 20! tours, $10^{-9}$ s/tour $= 77.14$ years
- Parallelization with 1,000 cores and 50 % efficiency same problem: $10^{-9}$ seconds/tour $= 56,31$ days

### Combinatorial Explosion
- Operational Research (OR)
  - Use OR methods to reduce the size of search space
  - Evaluation/Prunning of partial solutions
- Parallelization
  - Parallelization to solve quicker the problem with greater size

### Exact resolution up to 15112 cities
- Specific Parallel Branch & Cut: Concorde
- 22,6 years equivalent sequential time by Applegate, Bixby, Chvátal and Cook
Methods

Exact Algorithms
- Implicit total enumeration
- Tree search-space
- Divide&Conquer, Branch&Bound, Branch&Cut, ...
- Constraint programming

Heuristic Algorithms
- Not exhaustive
- Greedy heuristics
- Local search
- Meta : Simulated Annealing, Tabu search, Genetic algorithms, ...

Scientific Context
Solving Combinatorial Problems on HPC with BOBPP

Scientific Context

Methods

**Exact Algorithms**
- Implicit total enumeration
- Tree search-space
- Divide & Conquer, Branch & Bound, Branch & Cut, ...
- Constraint programming

**Heuristic Algorithms**
- Not exhaustive
- Greedy heuristics
- Local search
- Meta: Simulated Annealing, Tabu search, Genetic algorithms, ...
The user must **define** the problem, mainly the data stored in the node.
The user describes how a node is generated from a parent node (the child generation) according to the branching strategy.
The user describes a work on node: evaluation function, constraint propagation, ...
Usefull functionalities for exact methods

The user may **choose** between different exploration strategies
(best first, depth-first search, etc)
Useful functionalities for exact methods

The user may choose what is the goal of the search: the best solution, the number of feasible/best solutions. That implicates the stop criteria.
Outline

1. Scientific Context
2. Parallel Frameworks for Exact Methods
3. Experiments
4. Conclusion and Perspectives
Related Works

Frameworks Solving Exact Combinatorial Optimization problems
- Space search based algorithms
- Parallel Programming environments

Searches
- Divide & Conquer
- Branch & X
- A*
- ...

Programming Environments
- PThreads
- MPI
- KAAPI
- ...

Usual Frameworks
Designed for one algorithm and for one specific programming environment
Related Works

Frameworks Solving Exact Combinatorial Optimization problems
- Space search based algorithms
- Parallel Programming environments

Searches
- Divide & Conquer
- Branch & X
- A*
- ...

Programming Environments
- PThreads
- MPI
- KAAPI
- ...

Usual Frameworks
Designed for one algorithm and for one specific programming environment
Frameworks issues

Several parallelizations
- Two algorithms may have two different behaviours
- There does not exist only one ultimate parallelization

Several parallel environments
Different parallel architectures ⇒ different parallel algorithms and libraries
- Shared memory: threads and mutex
- Distributed memory: processes and messages
- Cluster of SMPs: threads and processes
- ...
### BOBPP

#### Objectives
- **Solve Combinatorial Optimization Problems based on search-trees**
  - Divide & Conquer
  - Branch & Bound
  - Branch & Cut
  - Branch & Price
- Framework which proposes an interface to write such algorithms in sequential and parallel

#### Functionalities for solving combinatorial problems
- Search strategies: depth-first, breadth-first, best-first, "best of the deepest"-first, ...
- Aims: best solution, number of best solutions, ...
Solving Combinatorial Problems on HPC with BOBPP
Parallel Frameworks for Exact Methods

BOBPP

Objectives

- Solve Combinatorial Optimization Problems based on search-trees
  - Divide & Conquer
  - Branch & Bound
  - Branch & Cut
  - Branch & Price

- Framework which proposes an interface to write such algorithms in sequential and parallel

Functionalities for solving combinatorial problems

- Aims: best solution, number of best solutions, . . .
Objectives

- Solve Combinatorial Optimization Problems based on search-trees
  - Divide & Conquer
  - Branch & Bound
  - Branch & Cut
  - Branch & Price
- Framework which proposes an interface to write such algorithms in sequential and parallel

Functionalities for solving combinatorial problems

- Aims: best solution, number of best solutions, . . .
Solving Combinatorial Problems on HPC with BOBPP

Parallel Frameworks for Exact Methods

BOBPP

Constraint Problem

CP Solver

D&C

QAP

B&P

B&B

B&C

A*

User algorithm

BOBPP

Programming environment

Sequential

Atheyascan

Pox threads

Threads/MPI
## Frameworks comparison

<table>
<thead>
<tr>
<th>Framework</th>
<th>B&amp;B</th>
<th>B&amp;B+LP</th>
<th>B&amp;P</th>
<th>B&amp;C</th>
<th>D&amp;C</th>
<th>THR</th>
<th>MPI</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPBB</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCP</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Symphony</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PEBBL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBC</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>BOBPP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

### BOBPP sources

http://forge.prism.uvsq.fr/projects/bobpp
BOBPP, Objects

Classes Involved

Common functionalities:

- **Class Node**: * Represents a node of the search-tree
- **Class Genchild**: * Methods generating the children of the nodes
- **Class Instance**: * Stores all the global data used during the search
- **Class Goal**: Stores the solution
- **Class Algo**: Method for the main loop
- **Class Priority Queue**: Stores all the nodes during the search

Redefined by the user for his specific problem

Parallelism

- Each thread executes the Algo main loop
- Each thread access asynchronously to the **Global Priority Queue** and to update the **Goal**
# BOBPP, Objects

## Classes Involved

### Common functionalities:

- **Class Node:** * Represents a node of the search-tree
- **Class Genchild:** * Methods generating the children of the nodes
- **Class Instance:** * Stores all the global data used during the search
- **Class Goal:** Stores the solution
- **Class Algo:** Method for the main loop
- **Class Priority Queue:** Stores all the nodes during the search

Redefined by the user for his specific problem

## Parallelism

- Each thread executes the Algo main loop
- Each thread access asynchronously to the **Global Priority Queue** and to update the **Goal**
BOBPP, Objects

Classes Involved

Common functionalities:

- **Class Node**: * Represents a node of the search-tree
- **Class Genchild**: * Methods generating the children of the nodes
- **Class Instance**: * Stores all the global data used during the search
- **Class Goal**: Stores the solution
- **Class Algo**: Method for the main loop
- **Class Priority Queue**: Stores all the nodes during the search
  
  Redefined by the user for his specific problem

Parallelism

- Each thread executes the Algo main loop
- Each thread access asynchronously to the **Global Priority Queue** and to update the **Goal**
BOBPP, Priority Queue Principle

Steps
1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue
Solving Combinatorial Problems on HPC with BOBPP
Parallel Frameworks for Exact Methods

BOBPP, Priority Queue Principle

**Steps**

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue
BOBPP, Priority Queue Principle

**Steps**

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue
BOBPP, Priority Queue Principle

Steps

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue
BOBPP, Priority Queue Principle

Steps

1. Nodes of the search-tree
2. Each thread selects one node
3. Each thread generates the children
4. If a solution is found, the goal is updated
5. The other nodes are inserted in the Priority Queue
Outline

1. Scientific Context
2. Parallel Frameworks for Exact Methods
3. Experiments
4. Conclusion and Perspectives
Protocols

Problems

- Quadratic Assignment Problem (QAP) (B&B algorithm)
- Golomb ruler Problem (D&C algorithm + OR-Tools Constraint Programming Solver)

Figure: QAP

Figure: Golomb Ruler

Computers

- Machine 1: AMD, 48 cores
- Machines 2 and 3: Intel, (with HT), 12 cores
Mean computation time and speed-up for the QAP on Machine 1
Mean computation time and speedup for the QAP on machine 2

*Break on the curves beginning at 12 threads (Hyper-Threading)*
## Preliminary results using hybrid MPI/Pthreads

<table>
<thead>
<tr>
<th>Machine</th>
<th># MPI proc</th>
<th># thr./proc</th>
<th>total # thr.</th>
<th>time (s.)</th>
<th>speedup</th>
<th>% sended/explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>12</td>
<td>48</td>
<td>85.46</td>
<td>19.95</td>
<td>0.265</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>7</td>
<td>49</td>
<td>68.93</td>
<td>24.73</td>
<td>0.290</td>
</tr>
<tr>
<td>2&amp;3</td>
<td>2</td>
<td>12</td>
<td>24</td>
<td>137.38</td>
<td>12.41</td>
<td>0.178</td>
</tr>
<tr>
<td>2&amp;3</td>
<td>4</td>
<td>6</td>
<td>24</td>
<td>133.42</td>
<td>12.78</td>
<td>0.226</td>
</tr>
<tr>
<td>all</td>
<td>16</td>
<td>6</td>
<td>96</td>
<td>41.50</td>
<td>41.07</td>
<td>0.455</td>
</tr>
</tbody>
</table>

### Informations
- Speedup computed according to our best seq. time: 1,704.44 s.
- Very small nodes
- Mean number of nodes explored: 248,732,621
- Mean % of sended nodes compared to explored nodes: 0.256 %
- MPI/Threads version is more efficient than the threads one due to memory allocation contention.
Using a Constraint Programming Solver

OR-Tools
- Open source library and solver developed by Google
- Constraint programming methods we used are exact methods that handle a search tree.

Porting OR-Tools on top of BOBPP
- An OR-Tools solver handles its own search tree
- One OR-Tools solver per BOBPP search Algo
- Migration of sub-trees between OR-Tools solvers to perform load balancing

PAJERO project funded by OSEO a public-sector institution dedicated to economic development
OR-Tools/BOBPP results

Execution time and speedup solving the Golomb of size 13

Good speedup
OR-Tools/BOBPP results

Load balancing on 48 threads solving the Golomb-13
Good load balancing
Outline

1. Scientific Context
2. Parallel Frameworks for Exact Methods
3. Experiments
4. Conclusion and Perspectives
Conclusion

The BOBPP Framework manages the search-tree

- Whatever the final machine
- For a large variety of combinatorial problems

Advantages

- One algorithm developed by the user
- Easy to use
- Easy to test and develop better node generations
- Easy to find the best strategy
Perspectives

<table>
<thead>
<tr>
<th>BOBPP Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI/Threads need more experiments</td>
</tr>
<tr>
<td>Perform test on larger machines (Grid5000)</td>
</tr>
<tr>
<td>Reduce the memory allocation bottleneck using recursive search with a branch of preallocated nodes since a fixed depth.</td>
</tr>
<tr>
<td>Perform tests with industrial problems developed by other members of the lab</td>
</tr>
<tr>
<td>Crew Scheduling: Branch&amp; Price,</td>
</tr>
<tr>
<td>Power minimization of wireless sensor network: Branch&amp; Cut,</td>
</tr>
<tr>
<td>Nurse planning: Constraint Programming</td>
</tr>
<tr>
<td>Restaurant waiters planning: Constraint Programming</td>
</tr>
</tbody>
</table>
End of presentation

Thank you

BOBPP sources

http://forge.prism.uvsq.fr/projects/bobpp