PhD defense - Computer Science

PGAS-based Parallel Branch-and-Bound for Ultra-Scale GPU-powered Supercomputers

Guillaume HELBECQUE

Jointly supervised by:
Prof. Dr. Nouredine MELAB and Prof. Dr. Pascal BOUVRY

UNIVERSITÉ DU LUXEMBOURG

Université de Lille

Outline

Context and objectives

Combinatorial Optimization Problems (COPs)

$$\text{Optimization problem}: \begin{cases} \min / \max f(x), \\ \text{subject to } x \in X \end{cases}$$

Optimization problems are increasingly big in many application areas:

- High-dimensionality, *e.g.*, number of decision variables
- Time-demanding objectives

Combinatorial Optimization Problems (COPs)

$$\text{Optimization problem}: \begin{cases} \min / \max f(x), \\ \text{subject to } x \in X \end{cases}$$

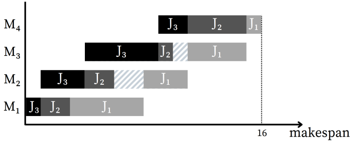
Optimization problems are increasingly big in many application areas:

- High-dimensionality, *e.g.*, number of decision variables
- Time-demanding objectives

Motivating example: Flowshop scheduling problem

- Big instance `ta056` (50 jobs, 20 machines): 10^{64} potential subproblems
- 22 years using a single-core processor [Mezmaz et al., 2007]
- **We need supercomputers to solve big COPs to optimality!**

	M_1	M_2	M_3	M_4
J_1	5	3	4	1
J_2	2	2	1	4
J_3	1	3	5	2

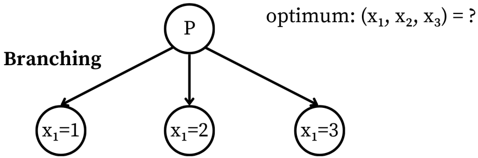


Branch-and-Bound (B&B) algorithms

B&B is a search algorithm based on **implicit enumeration** of candidate solutions, explored by constructing a **tree**.

Branch-and-Bound (B&B) algorithms

B&B is a search algorithm based on **implicit enumeration** of candidate solutions, explored by constructing a **tree**.

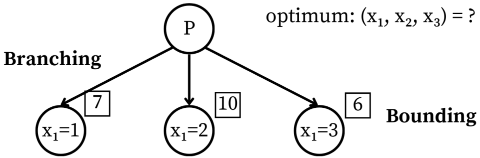


Four operators:

- **Branching**

Branch-and-Bound (B&B) algorithms

B&B is a search algorithm based on **implicit enumeration** of candidate solutions, explored by constructing a **tree**.



Four operators:

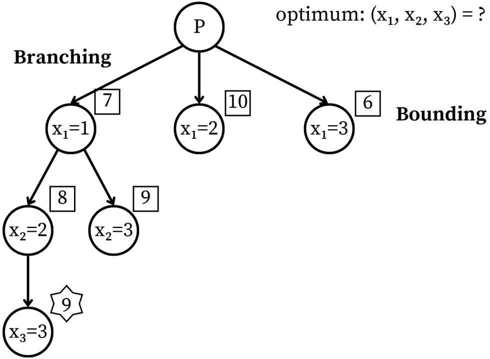
- **Branching**
- **Bounding**

Branch-and-Bound (B&B) algorithms

B&B is a search algorithm based on **implicit enumeration** of candidate solutions, explored by constructing a **tree**.

Four operators:

- **Branching**
- **Bounding**
- **Selection**

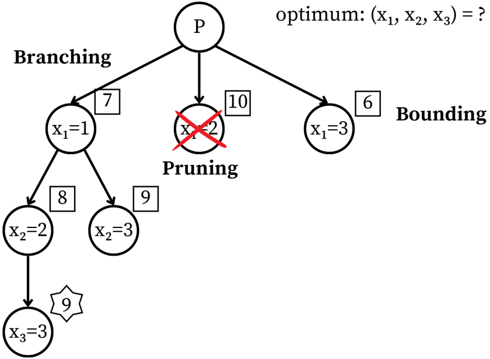


Branch-and-Bound (B&B) algorithms

B&B is a search algorithm based on **implicit enumeration** of candidate solutions, explored by constructing a **tree**.

Four operators:

- **Branching**
- **Bounding**
- **Selection**
- **Pruning**



Branch-and-Bound (B&B) algorithms

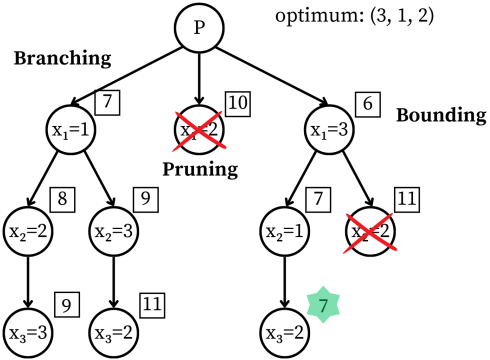
B&B is a search algorithm based on **implicit enumeration** of candidate solutions, explored by constructing a **tree**.

Four operators:

- **Branching**
- **Bounding**
- **Selection**
- **Pruning**

Main properties:

- **Huge** and **unpredictable** trees
- **Dynamic** and **irregular** loads



Architecture of modern supercomputers

TOP500 list (Nov. 2024): 9 out of TOP10 supercomputers are **heterogeneous**

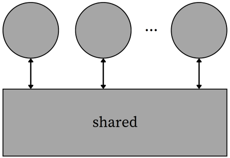
Rank	Name	System	Country	Cores	Rmax [PFlop/s]
1	El Capitan	AMD 4th Gen EPYC 24C 1.8GHz, **AMD Instinct MI300A**, Slingshot-11	United States	11,039,616	**1,742.00**
2	Frontier	AMD Optimized 3rd Generation EPYC 64C 2GHz, **AMD Instinct MI250X**, Slingshot-11	United States	9,066,176	**1,353.00**
3	Aurora	Xeon CPU Max 9470 52C 2.4GHz, **Intel Data Center GPU Max**, Slingshot-11	United States	9,264,128	**1,012.00**
4	Eagle	Xeon Platinum 8480C 48C 2GHz, **NVIDIA H100**, NVIDIA Infiniband NDR	United States	2,073,600	561.20
5	HPC6	AMD Optimized 3rd Generation EPYC 64C 2GHz, **AMD Instinct MI250X**, Slingshot-11	Italy	3,143,520	477.90
6	Supercomputer Fugaku	A64FX 48C 2.2GHz, Tofu interconnect D	Japan	7,630,848	442.01
7	Alps	NVIDIA Grace 72C 3.1GHz, **NVIDIA GH200 Superchip**, Slingshot-11	Switzerland	2,121,600	434.90
8	LUMI	AMD Optimized 3rd Generation EPYC 64C 2GHz, **AMD Instinct MI250X**, Slingshot-11	Finland	2,752,704	379.70
9	Leonardo	Xeon Platinum 8358 32C 2.6GHz, **NVIDIA A100 SXM4 64 GB**, Quad-rail NVIDIA HDR100 Infiniband	Italy	1,824,768	241.20
10	Tuolumne	AMD 4th Gen EPYC 24C 1.8GHz, **AMD Instinct MI300A**, Slingshot-11	United States	1,161,216	208.10

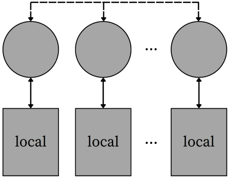
Architecture of modern supercomputers

TOP500 list (Nov. 2024): devices by three **different vendors** are present

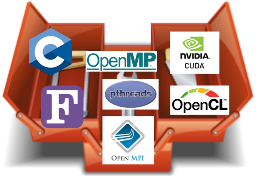
Rank	Name	System	Country	Cores	Rmax [PFlop/s]
1	El Capitan	AMD 4th Gen EPYC 24C 1.8GHz, **AMD Instinct MI300A**, Slingshot-11	United States	11,039,616	**1,742.00**
2	Frontier	AMD Optimized 3rd Generation EPYC 64C 2GHz, **AMD Instinct MI250X**, Slingshot-11	United States	9,066,176	**1,353.00**
3	Aurora	Xeon CPU Max 9470 52C 2.4GHz, **Intel Data Center GPU Max**, Slingshot-11	United States	9,264,128	**1,012.00**
4	Eagle	Xeon Platinum 8480C 48C 2GHz, **NVIDIA H100**, NVIDIA Infiniband NDR	United States	2,073,600	561.20
5	HPC6	AMD Optimized 3rd Generation EPYC 64C 2GHz, **AMD Instinct MI250X**, Slingshot-11	Italy	3,143,520	477.90
6	Supercomputer Fugaku	A64FX 48C 2.2GHz, Tofu interconnect D	Japan	7,630,848	442.01
7	Alps	NVIDIA Grace 72C 3.1GHz, **NVIDIA GH200 Superchip**, Slingshot-11	Switzerland	2,121,600	434.90
8	LUMI	AMD Optimized 3rd Generation EPYC 64C 2GHz, **AMD Instinct MI250X**, Slingshot-11	Finland	2,752,704	379.70
9	Leonardo	Xeon Platinum 8358 32C 2.6GHz, **NVIDIA A100 SXM4 64 GB**, Quad-rail NVIDIA HDR100 Infiniband	Italy	1,824,768	241.20
10	Tuolumne	AMD 4th Gen EPYC 24C 1.8GHz, **AMD Instinct MI300A**, Slingshot-11	United States	1,161,216	208.10

PGAS alternative to MPI+X

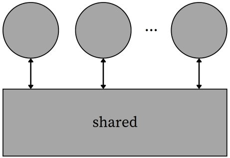


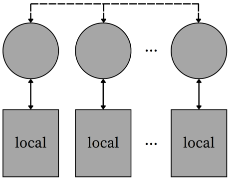


MPI+X
"Evolutionary" approach

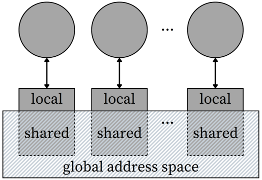


PGAS alternative to MPI+X



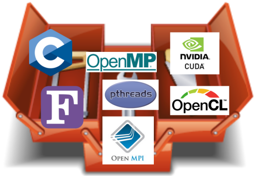


VS.



MPI+X

“Evolutionary” approach



PGAS

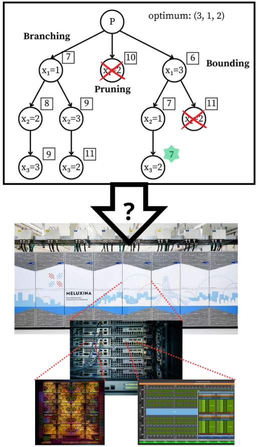
“Revolutionary” approach

Objectives

- Huge and unpredictable trees
- Dynamic and irregular loads

Challenges

- Efficient operators with scalable data structures
- Efficient multi-level parallelism



- Bigger than ever
- Heterogeneous
- Complex programming
- Unreliable (MTBF<1h)

Overall objective: Revisit the design and implementation of parallel B&B for solving complex problems on ultra-scale supercomputers

Related work

- Limitation of existing parallel B&B algorithms, *e.g.* [Lalami et al., 2012]; [Gmys et al., 2017]; [Chakroun et al., 2013]; [Alba et al., 2002]:
  - Focus only on performance
  - Combine low-level programming environments
- Focus on **holistic** PGAS-based approaches:
  - GPU computing in this context is at its infancy
  - Some initiatives to support GPU exist [Cunningham et al., 2011]; [Chen et al., 2011]; [Hayashi et al., 2023]
  - GPU-native supports recently became available, *e.g.*, **Chapel** [Milthorpe et al., 2024]

    → Chapel (HPE/Cray) is considered in this thesis
- Few works explore PGAS-based GPU-accelerated tree search approaches [Carneiro et al., 2021]

The Chapel programming language

High-level PGAS-based language:

- Portable & scalable
- Abstractions for data and task parallelism, concurrency, and nested parallelism
- Open-source & **collaborative**, *e.g.*, ChapelCon

GPU-native support:

- CPU parallelism features also target GPU
- **Vendor-neutral**, through the LLVM compiler framework:
  - PTX for NVIDIA GPUs
  - AMDGCN for AMD GPUs

More at: `https://chapel-lang.org/`.

Contributions

- Scalable data structure for billions of subproblems
  → The `distBag_DFS` data structure
- Efficient mechanisms to deal with dynamic and irregular loads
- Efficient implementation of these mechanisms
  → PGAS-based parallel B&B for CPU-based systems
  → PGAS-based parallel B&B for GPU-based systems
  → Software platform in Chapel

Collaboration with HPE/Cray (Spring, Texas, USA)

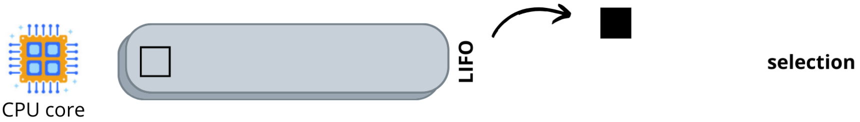
PGAS-based B&B for CPU-based systems

Incremental design of the `distBag_DFS` data structure

Single-core pool-based B&B:

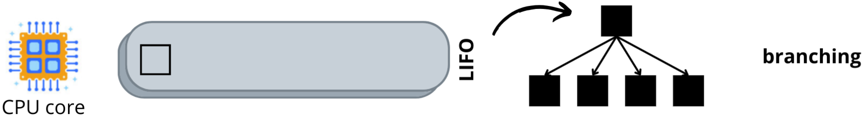
Incremental design of the `distBag_DFS` data structure

Single-core pool-based B&B:



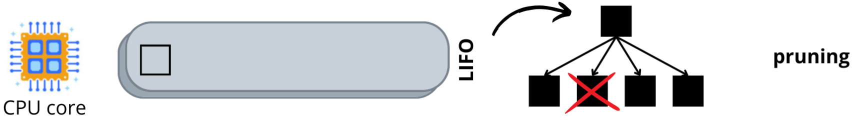
Incremental design of the `distBag_DFS` data structure

Single-core pool-based B&B:



Incremental design of the `distBag_DFS` data structure

Single-core pool-based B&B:

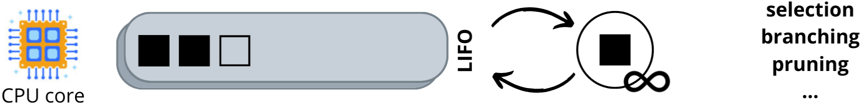


Incremental design of the `distBag_DFS` data structure

Single-core pool-based B&B:

Incremental design of the `distBag_DFS` data structure

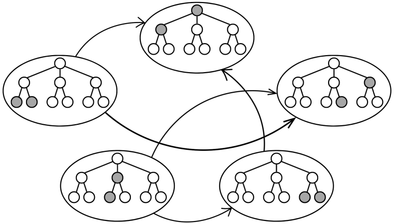
Single-core pool-based B&B:



We iterate until the pool is empty $\rightarrow$ termination

Model for parallel B&B on CPU

Parallel tree exploration [Melab, 2005]



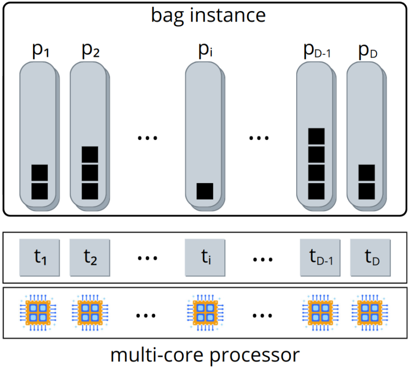
Massively parallel and generic, but highly irregular

Challenges: single-pool → multi-pool

- Efficient load balancing mechanism
- Efficient termination detection

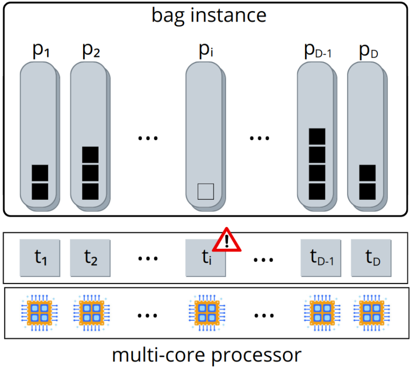
Incremental design of the distBag_DFS data structure

Parallel multi-pool B&B:



Incremental design of the `distBag_DFS` data structure

Parallel multi-pool B&B:

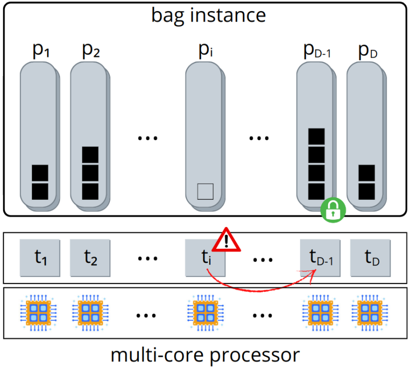


Dynamic load balancing based on **Work Stealing (WS):**

$$\text{WS} = \text{victim selection} + \text{granularity policy}$$

Incremental design of the `distBag_DFS` data structure

Parallel multi-pool B&B:

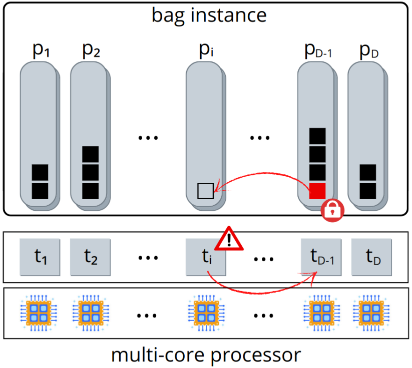


Dynamic load balancing based on **Work Stealing (WS)**:

- Victim selection: **random**

Incremental design of the `distBag_DFS` data structure

Parallel multi-pool B&B:

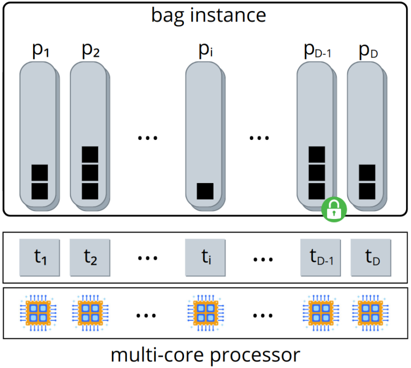


Dynamic load balancing based on **Work Stealing (WS):**

- Victim selection: **random**
- Granularity: **steal-one**

Incremental design of the `distBag_DFS` data structure

Parallel multi-pool B&B:

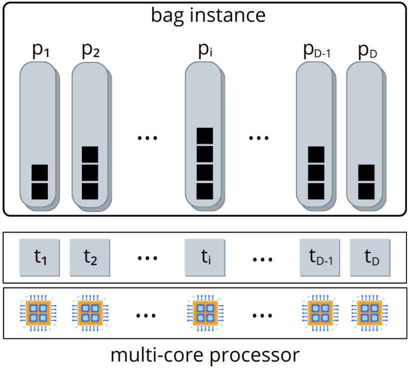


Dynamic load balancing based on **Work Stealing (WS):**

- Victim selection: **random**
- Granularity: **steal-one**

Incremental design of the distBag_DFS data structure

Parallel multi-pool B&B:

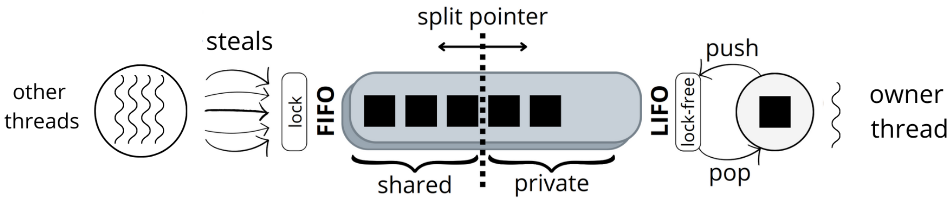


Dynamic load balancing based on **Work Stealing (WS):**

- Victim selection: **random**
- Granularity: **steal-one**

Incremental design of the `distBag_DFS` data structure

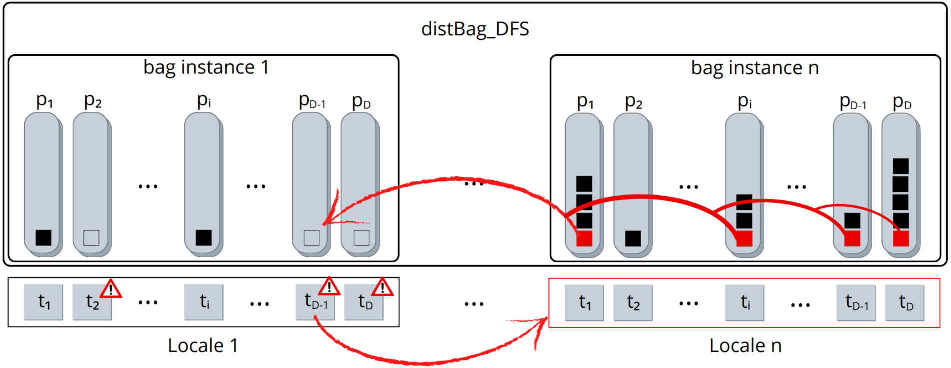
Scalable synchronization mechanism: non-blocking split double-ended queues [Dijk et al., 2014]



- **Concurrent accesses** from both ends
- **Lock-free** local access
- **Copy-free transfer** from shared to private parts

Incremental design of the distBag_DFS data structure

Locality-aware WS at the inter-node level: local, then global



- Victim locale selection: **random**
- Granularity: **steal-one** in each pool (if possible)

Global termination detection

"Wave algorithm" [Matocha et al., 1998];

- Each thread owns a **state variable** (`IDLE` or `BUSY`)
- The initiator thread checks all other states
- Termination = all threads are `IDLE`

Algorithm 1: Pseudo-code of PGAS-based termination detection

input : *allThreadStates*: global array of thread states

```
for localeID from 0 to numLocales do
    for threadID from 0 to D-1 do
        if allThreadStates[localeID][threadID]=BUSY then
            return false; // At least one thread is still working
return true; // Triggers termination
```

Experimental testbed

- MeluXina - Cluster module (ranked 460^{th} in June 2024 TOP500)
  - Up to 400 compute nodes × 2 AMD EPYC Rome 7H12 64 cores @ 2.6 GHz CPUs and 512 GB of RAM
  - InfiniBand HDR high-speed fabric
- Chapel release 2.1.0

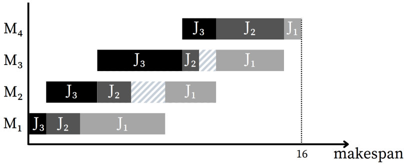
MELUXINA
HIGH PERFORMANCE
COMPUTING IN LUXEMBOURG

Benchmark problems: Permutation Flowshop Scheduling

PFSP
0/1-Knapsack
N-Queens
UTS

Objective: Minimize the completion time of the last job on the last machine.

	M_1	M_2	M_3	M_4
J_1	5	3	4	1
J_2	2	2	1	4
J_3	1	3	5	2



Main characteristics:

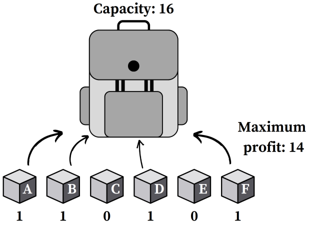
- NP-hard COP
- Minimization problem
- Permutation-based
- Coarse-grained bounding

Benchmark problems: Binary Knapsack

Objective: Maximize total profit while satisfying capacity constraint.

PFSP
0/1-Knapsack
N-Queens
UTS

	A	B	C	D	E	F
Profit	3	5	7	2	1	4
Weight	3	4	12	4	7	2



Main characteristics:

- NP-hard COP
- Maximization problem
- Binary decision variables
- Medium-grained bounding

Benchmark problems: N-Queens

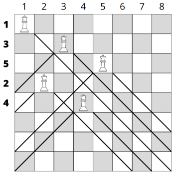
Objective: Determine the total number of valid configurations for placing N non-attacking queens on an $N \times N$ chessboard.

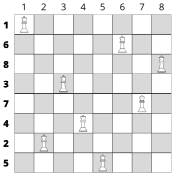
PFSP

0/1-Knapsack

N-Queens

UTS





Main characteristics:

- Constraint satisfaction problem
- Permutation-based
- Fine-grained evaluation

Benchmark problems: Unbalanced Tree Search

Objective: Count the number of nodes in an implicitly constructed tree that is parameterized in shape, depth, size, and imbalance.

PFSP
0/1-Knapsack
N-Queens
UTS

- **Binomial:** q child nodes with probability p and no children with probability $1 - p$
- **Geometric:** Each node has a branching factor that follows a geometric distribution with an expected value that is specified by the parameter $b_0 > 1$

Main characteristics:

- Performance benchmark
- Trees follow a given distribution
- Fine-grained evaluation
- Focus on load balancing

Evaluation of dynamic load balancing

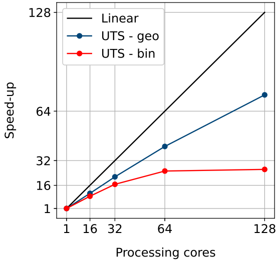
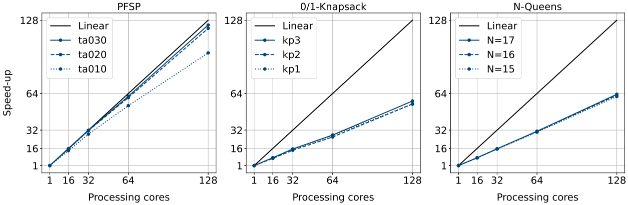


Fig. 1: Speed-up achieved solving geometrical and binomial synthetic UTS trees.

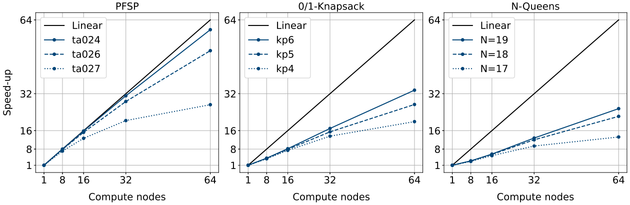
- 59% of the ideal speed-up solving `UTS-geo`
- Only 29% solving `UTS-bin` → branching factor
- High WS success rate

Instance	Nb. of nodes (10^6)	Time (s)	nodes/s (10^3)	WS attempts (% success)
`UTS-geo`	91.4	36.06	2,534.6	48,433 (99.0%)
`UTS-bin`	131.7	36.30	3,628.1	1,473,048 (96.8%)

Strong scaling efficiency on three different problems



(a) Intra-node parallel level.



(b) Inter-node parallel level.

Comparison against MPI+X baseline on PFSP

Baseline: MPI-PBB, state-of-the-art MPI+PThreads implementation [Gmys, 2017].

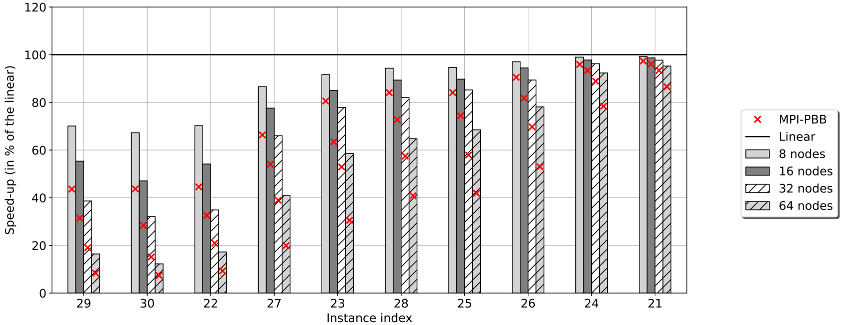
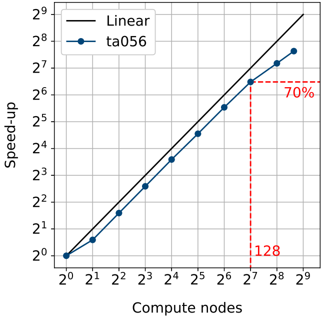


Fig. 3: Speed-up achieved by `P3D-DFS` and `MPI-PBB` up to 64 compute nodes compared to the execution on one node.

Solving hard PFSP instances at scale

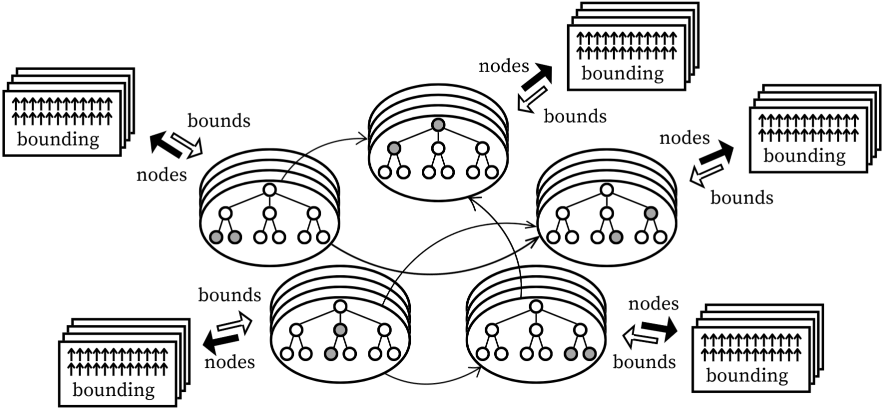
Proof of optimality for hard PFSP instances:

Instance	# CPU cores	Time (s)	Core-hour	# tree nodes (10^9)	Optimum
`ta056`	51,200	18.1	257.4 (∼ 11 days)	173.3	3,679
`ta052`	8,192	7,960.5	18,114.6 (∼ 2 years)	17,117.8	3,699
`ta057`	51,200	2,017.6	28,694.8 (∼ 4 years)	28,340.7	3,704
`ta053`	8,192	43,605.5	99,226.7 (∼ 12 years)	94,885.1	3,640

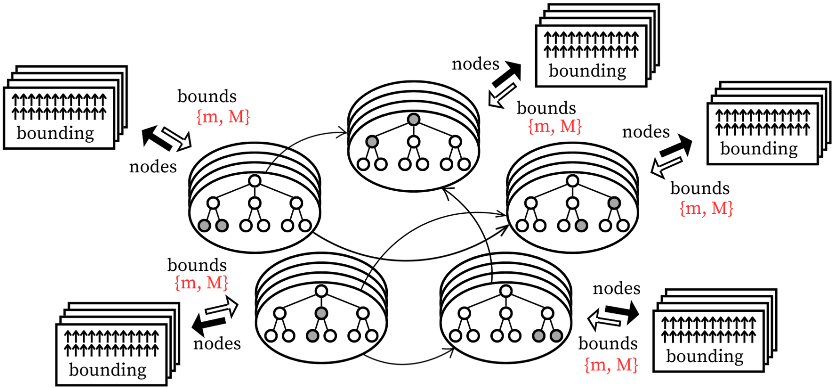


PGAS-based B&B for GPU-based systems

Model for parallel B&B on GPU



Model for parallel B&B on GPU



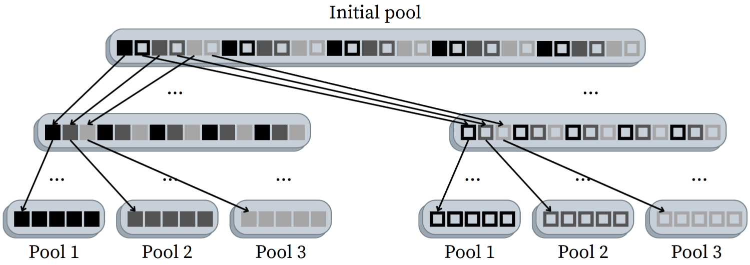
Challenges:

- Tune parameters: $\{m, M\}$
- Manage initialization
- Redesign WS mechanism
- Ensure portability

Initial search and static distribution

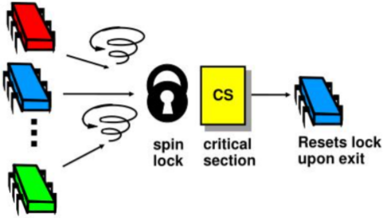
1. Initial search on CPU (assuming D GPUs):
   - Sequential (or weakly parallel) B&B
   - Until $numNodes * D * m$ pending nodes

2. Static workload distribution:



Redesign of WS and synchronization

- Revisit of WS mechanism:
  - Victim selection: **random**
  - Granularity: **steal-half**, only if at least $2 * m$ available nodes
- Revisit of synchronization mechanism: **spin-locks**



Source for image: `https://www.slideserve.com/deiondre/spin-locks-and-contention`.

Experimental testbed

- Evaluating code portability:
    - NVIDIA P100
    - NVIDIA V100
    - NVIDIA A100
    - AMD MI50
    - AMD MI250X
    - AMD MI300X
- Evaluating scalability at intra-node level: LUMI, ranked 8^{th} in Nov. 2024 TOP500 ranking
    - 64-core AMD EPYC 7A53 "Trento" CPU and four AMD Instinct MI250X GPUs
- Chapel release 2.1.0

Grid'5000

LUMI

Code performance and portability

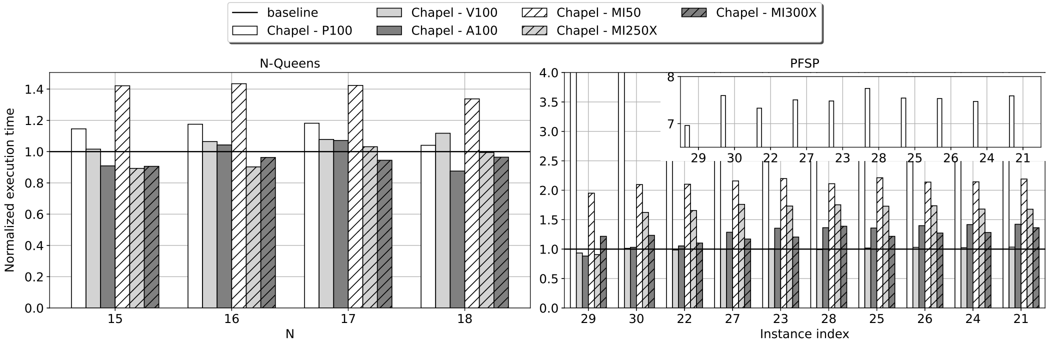


Fig. 4: Normalized execution time, Chapel *vs.* C+CUDA/HIP, single-GPU.

Loop-Invariant Code Motion (LICM) compiler optimization

Algorithm 2: Example of LICM optimization in PGAS-based languages.

input : n: number of iterations
A, B: arrays of constant data

Aptr $\leftarrow$ constPtr($A[0]$);
Bptr $\leftarrow$ constPtr($B[0]$);
for i ***from*** *0* ***to*** n **do**
```
    // arithmetic operations, accessing Aptr and Bptr
```

- LICM optimization not always triggered by Chapel compiler
  $\rightarrow$ **Bug report**
- Performance improvement with manual LICM:

GPU architecture	NVIDIA P100	NVIDIA V100	NVIDIA A100	AMD MI50	AMD MI250X	AMD MI300X
Execution time	-10%	-17%	-26%	-26%	-11%	-7%

Calibration of parameters

Solving PFSP `ta030`:

Table 1: Calibration of (m, M) parameters on AMD MI250X. Whiter is better.

m \ M	50,000	100,000	200,000	300,000	400,000	500,000	600,000
10	9.78	9.55	9.41	9.42	9.36	9.36	9.42
20	9.78	9.52	9.4	9.39	9.38	9.36	9.38
30	9.78	9.56	9.41	9.4	9.36	9.36	9.38
40	9.78	9.56	9.4	9.38	9.37	9.37	9.37
50	9.78	9.55	9.41	9.38	9.37	9.36	9.36
60	9.79	9.56	9.4	9.38	9.37	9.36	9.36
70	9.78	9.56	9.42	9.4	9.36	9.36	9.37
80	9.78	9.55	9.41	9.38	9.36	9.37	9.38
90	9.79	9.56	9.41	9.37	9.37	9.36	9.38
100	9.8	9.56	9.41	9.39	9.36	9.36	9.39

Calibration of parameters

Solving PFSP `ta030`:

Table 1: Calibration of (m, M) parameters on AMD MI250X. Whiter is better.

m \ M	50,000	100,000	200,000	300,000	400,000	500,000	600,000
10	9.78	9.55	9.41	9.42	9.36	9.36	9.42
20	9.78	9.52	9.4	9.39	9.38	9.36	9.38
30	9.78	9.56	9.41	9.4	9.36	9.36	9.38
40	9.78	9.56	9.4	9.38	9.37	9.37	9.37
50	9.78	9.55	9.41	9.38	9.37	9.36	9.36
60	9.79	9.56	9.4	9.38	9.37	9.36	9.36
70	9.78	9.56	9.42	9.4	9.36	9.36	9.37
80	9.78	9.55	9.41	9.38	9.36	9.37	9.38
90	9.79	9.56	9.41	9.37	9.37	9.36	9.38
100	9.8	9.56	9.41	9.39	9.36	9.36	9.39

Optimal values: $m = 50$ and $M = 500,000$

Strong scaling efficiency: intra-node

Table 2: Strong scaling efficiency at the intra-node level.

Instance	GPU×1	GPU×2		GPU×4		GPU×8	
	kn/s	*kn/s*	speed-up	*kn/s*	speed-up	*kn/s*	speed-up
15-Queens	34,744.0	57,970.7	1.67	82,106.1	2.36	88,475.6	2.54
16-Queens	34,884.4	60,163.2	1.72	95,487.7	2.74	119,120.2	3.41
17-Queens	34,940.7	64,458.7	1.84	93,966.3	2.69	124,720.3	3.57
AVG	**34,856.4**	**60,864.2**	**1.74**	**90,553.4**	**2.60**	**110,768.8**	**3.17**
`ta028`	1,366.0	2,092.4	1.53	3,569.3	2.61	5,993.0	4.39
`ta025`	1,361.2	2,102.2	1.54	3,673.6	2.70	6,468.7	4.75
`ta026`	1,393.4	2,166.3	1.55	3,814.3	2.74	6,745.3	4.84
`ta024`	1,393.6	2,167.5	1.56	3,973.0	2.85	8,086.0	5.80
`ta021`	1,421.1	2,175.4	1.53	3,992.2	2.81	7,170.0	5.05
AVG	**1381.1**	**1762.8**	**1.54**	**3804.5**	**2.7**	**6892.6**	**5.0**

Chapel on average 60% slower than C+HIP considering GPU×1 on PFSP

Software platform for parallel B&B

Scalable code development

Motivations:

- Reducing the costs of **software development**
- Targeting as wider **community** as possible
- Ensuring maintainability and **portability** across diverse architectures

Tools:

- Object-oriented programming
- High-level abstraction for parallelism
- Distribution (*e.g.*, GitHub), *etc.*

The Chapel's DistributedBag module

Package module for importing the `distBag_DFS` data structure.

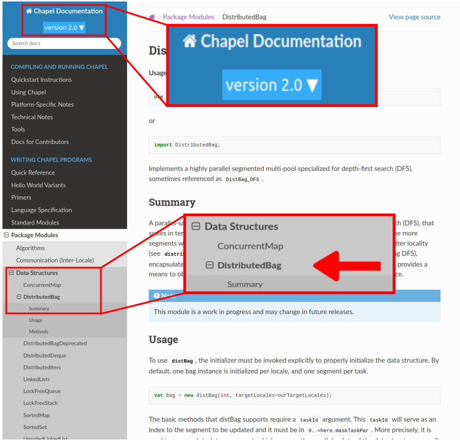
Available from Chapel 2.0 release

Usage:

```
use DistributedBag;

var bag = new distBag(int);
...
```

Code demo at ChapelCon '24 (broadcast on YouTube).



Software platform

`Node` type + `Problem` concrete class = set of parallel B&B skeletons (`--mode`)

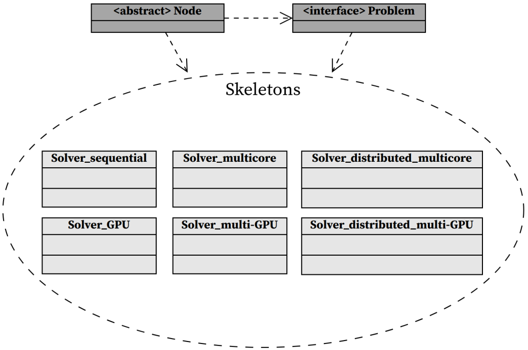


Fig. 5: UML diagram of the `pBB-chpl` software platform.

The `Problem` interface

Template for problem-specific operators (branching, bounding, *etc.*)

Algorithm 3: The `Problem` interface.

```
class Problem
    // CORE PROCEDURES
    proc copy() {}
    proc decompose(/*args*/) {}
    proc getInitSolution(): int {}
    // UTILITY PROCEDURES
    proc print_settings(): void {}
    proc print_results(/*args*/): void {}
    proc output_filepath(): string {}
    proc help_message(): void {}
```

Skeletons for parallel B&B: example

Algorithm 4: Chapel-based B&B skeleton for sequential execution.

```
use List;
use Node, Problem;
proc search_sequential(Node, problem)
    var best = problem.getInitSolution();
    problem.print_settings();
    // INITIALIZATION
    var pool: list(Node);
    var root = new Node (problem);
    pool.pushBack(root);
    // TREE EXPLORATION
    while !pool.isEmpty() do
        var parent: Node = pool.popBack();
        var children = problem.decompose(Node, parent, best);
        pool.pushBack(children);
    // OUTPUT
    problem.print_results(best);
```

Conclusions and perspectives

Conclusions (1)

- Design and implementation of the PGAS-based `distBag_DFS` data structure
  - → Highly parallel and scalable
  - → Hierarchical and locality-aware
  - → Packaged into Chapel

Conclusions (1)

- Design and implementation of the PGAS-based `distBag_DFS` data structure
  - → Highly parallel and scalable
  - → Hierarchical and locality-aware
  - → Packaged into Chapel
- Design and implementation of a `distBag_DFS`-based parallel B&B
  - → High level of abstraction
  - → Genericity w.r.t. the problem solved
  - → Performance depends on characteristics of problem
  - → Competitive against MPI+X implementation
  - → Proof of optimality for hard PFSP instances

Conclusions (2)

- Design and implementation of a PGAS-based GPU-accelerated B&B
  - → Unification of all parallel levels
  - → Genericity w.r.t. the problem solved
  - → Portability on both NVIDIA and AMD
  - → High relative performance at the intra-node level
  - → Limited compiler optimization

Conclusions (2)

- Design and implementation of a PGAS-based GPU-accelerated B&B
  - → Unification of all parallel levels
  - → Genericity w.r.t. the problem solved
  - → Portability on both NVIDIA and AMD
  - → High relative performance at the intra-node level
  - → Limited compiler optimization
- Distribution of a Chapel software platform for parallel B&B
  - → Open-source and freely available
  - → Extensible to other problems
  - → B&B skeletons targeting various parallel systems

Perspectives

- Extend/improve proposed approaches
  → Adaptive load balancing mechanism [Chakroun et al., 2012]

Within the framework of the ANR/FNR UltraBO research project

Perspectives

- Extend/improve proposed approaches
  → Adaptive load balancing mechanism [Chakroun et al., 2012]
- Design **fault-tolerance** mechanisms
  → Checkpoint-and-restart [Bendjoudi et al., 2014]

Within the framework of the ANR/FNR UltraBO research project

Perspectives

- Extend/improve proposed approaches
  → Adaptive load balancing mechanism [Chakroun et al., 2012]
- Design **fault-tolerance** mechanisms
  → Checkpoint-and-restart [Bendjoudi et al., 2014]
- **Hybridize B&B** with metaheuristics
  → Low-level and high-level teamwork hybrids [Talbi, 2009]

Within the framework of the ANR/FNR UltraBO research project

Perspectives

- Extend/improve proposed approaches
  → Adaptive load balancing mechanism [Chakroun et al., 2012]
- Design **fault-tolerance** mechanisms
  → Checkpoint-and-restart [Bendjoudi et al., 2014]
- **Hybridize B&B** with metaheuristics
  → Low-level and high-level teamwork hybrids [Talbi, 2009]
- Solve **open instances** of hard COPs
  → Some PFSP Taillard's instances remain open, more than 30 years after their release [Gmys, 2022] (*e.g.*, `ta051`, `ta054`, `ta055`, *etc.*)

Within the framework of the ANR/FNR UltraBO research project

Main publications

International journals (2):

- **G. Helbecque**, J. Gmys, N. Melab, T. Carneiro, and P. Bouvry. Parallel distributed productivity-aware tree-search using Chapel. *Concurrency Computat Pract Exper* (**CCPE**). 35(27):e7874, 2023. DOI: 10.1002/cpe.7874.
- **G. Helbecque**, E. Krishnasamy, T. Carneiro, N. Melab, and P. Bouvry. Portable PGAS-based GPU-accelerated Branch-and-Bound Algorithms at Scale. *Concurrency Computat Pract Exper* (**CCPE**). 2025. **[Invited for publication - submitted].**

International conferences and workshops (5):

- **G. Helbecque**, J. Gmys, T. Carneiro, N. Melab, and P. Bouvry. A performance-oriented comparative study of the Chapel high-productivity language to conventional programming environments. In: *Proceedings of the Thirteenth International Workshop on Programming Models and Applications for Multicores and Manycores* (**PMAM**). pp. 21–29, 2022. DOI: 10.1145/3528425.3529104.
- **G. Helbecque**, E. Krishnasamy, N. Melab, and P. Bouvry. GPU-Accelerated Tree-Search in Chapel versus CUDA and HIP. In: *14th IEEE Workshop Parallel / Distributed Combinatorics and Optimization* (**PDCO**). 2024. DOI: 10.1109/IPDPSW63119.2024.00156.
- **G. Helbecque**, T. Carneiro, N. Melab, J. Gmys, and P. Bouvry. PGAS Data Structure for Unbalanced Tree-Based Algorithms at Scale. In: *Computational Science – ICCS 2024* (**ICCS**). vol 14834, 2024. DOI: 10.1007/978-3-031-63759-9_13.
- T. Carneiro, E. Kayraklioglu, **G. Helbecque**, N. Melab. Investigating Portability in Chapel for Tree-based Optimization on GPU-powered Clusters. In: *Euro-Par 2024: Parallel Processing* (**EuroPar**). LNCS, vol. 14803, pp. 386-399, 2024. DOI: 10.1007/978-3-031-69583-4_27.
- **G. Helbecque**, E. Krishnasamy, T. Carneiro, N. Melab, and P. Bouvry. A Chapel-based Multi-GPU Branch-and-Bound Algorithm: Application to the Flowshop Scheduling Problem. In: *22nd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Platforms* (**HeteroPar**). 2024.

Software

- The Chapel's `DistributedBag` module:
  `https://github.com/chapel-lang/chapel` (Chapel release $\geq$ 2.0)
- Chapel-based parallel B&B skeletons for CPU-based systems:
  `https://github.com/Guillaume-Helbecque/P3D-DFS`
- Chapel-based parallel B&B skeletons for GPU-based systems:
  `https://github.com/Guillaume-Helbecque/GPU-accelerated-tree-search-Chapel`

References I

Alba, E. et al. (2002). "MALLBA: A Library of Skeletons for Combinatorial Optimisation". In: *Euro-Par 2002 Parallel Processing*, pp. 927–932. DOI: 10.1007/3-540-45706-2_132.

Bendjoudi, A., N. Melab, and E.-G. Talbi (2014). "FTH-B&B: A Fault-Tolerant Hierarchical Branch and Bound for Large Scale Unreliable Environments". In: *IEEE Transactions on Computers* 63.9, pp. 2302–2315. DOI: 10.1109/TC.2013.40.

Carneiro, T. et al. (2021). "Towards Chapel-based Exascale Tree Search Algorithms: dealing with multiple GPU accelerators". In: *HPCS 2020 - The 18th International Conference on High Performance Computing & Simulation*.

Chakroun, I. and N. Melab (2012). "An Adaptative Multi-GPU Based Branch-and-Bound. A Case Study: The Flow-Shop Scheduling Problem". In: *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, pp. 389–395. DOI: 10.1109/HPCC.2012.59.

References II

Chakroun, I. et al. (2013). “Combining multi-core and GPU computing for solving combinatorial optimization problems”. In: *Journal of Parallel and Distributed Computing* 73.12, pp. 1563–1577. DOI: 10.1016/j.jpdc.2013.07.023.

Chen, L. et al. (2011). “Unified Parallel C for GPU Clusters: Language Extensions and Compiler Implementation”. In: *Languages and Compilers for Parallel Computing*, pp. 151–165. DOI: 10.1007/978-3-642-19595-2_11.

Cunningham, D., R. Bordawekar, and V. Saraswat (2011). “GPU programming in a high level language: compiling X10 to CUDA”. In: *Proceedings of the 2011 ACM SIGPLAN X10 Workshop*. DOI: 10.1145/2212736.2212744.

Dijk, T. van and J. C. van de Pol (2014). “Lace: Non-blocking Split Deque for Work-Stealing”. In: *Euro-Par 2014: Parallel Processing Workshops*, pp. 206–217. DOI: 10.1007/978-3-319-14313-2_18.

Gmys, J. (2017). *Heterogeneous cluster computing for many-task exact optimization - Application to permutation problems*. PhD thesis, Université de Mons and Université de Lille.

References III

Gmys, J. (2022). "Exactly Solving Hard Permutation Flowshop Scheduling Problems on Peta-Scale GPU-Accelerated Supercomputers". In: *INFORMS Journal on Computing* 34.5, pp. 2502–2522. DOI: 10.1287/ijoc.2022.1193.

Gmys, J. et al. (2017). "IVM-based parallel branch-and-bound using hierarchical work stealing on multi-GPU systems". In: *Concurrency and Computation: Practice and Experience* 29.9, e4019. DOI: 10.1002/cpe.4019.

Hayashi, A., S. R. Paul, and V. Sarkar (2023). "A Multi-Level Platform-Independent GPU API for High-Level Programming Models". In: *High Performance Computing. ISC High Performance 2022 International Workshops*, pp. 90–107. DOI: 10.1007/978-3-031-23220-6_7.

Lalami, M. E. and D. El-Baz (2012). "GPU Implementation of the Branch and Bound Method for Knapsack Problems". In: *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pp. 1769–1777. DOI: 10.1109/IPDPSW.2012.219.

References IV

Matocha, J. and T. Camp (1998). “A taxonomy of distributed termination detection algorithms”. In: *Journal of Systems and Software* 43.3, pp. 207–221. DOI: 10.1016/S0164-1212(98)10034-1.

Melab, N. (2005). *Contributions à la résolution de problèmes d’optimisation combinatoire sur grilles de calcul*. Thèse HDR, Université des Sciences et Technologies de Lille.

Mezmaz, M., N. Melab, and E-G. Talbi (2007). “A Grid-enabled Branch and Bound Algorithm for Solving Challenging Combinatorial Optimization Problems”. In: *2007 IEEE International Parallel and Distributed Processing Symposium*, pp. 1–9. DOI: 10.1109/IPDPS.2007.370217.

Milthorpe, J., X. Wang, and A. Azizi (2024). “Performance Portability of the Chapel Language on Heterogeneous Architectures”. In: *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 6–13. DOI: 10.1109/IPDPSW63119.2024.00011.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley Publishing. ISBN: 978-0-470-27858-1.

Thank you for your attention

Contact:

`guillaume.helbecque@uni.lu`

`guillaume.helbecque@univ-lille.fr`

Work supported by:

Luxembourg National Research Fund

agence nationale de la recherche

Appendices

GPU approach, strong scaling efficiency inter-node

Table 3: Strong scaling efficiency at the inter-node level.

Instance	node×1	node×16		node×32		node×64		node×128	
	kn/s	kn/s	speed-up	kn/s	speed-up	kn/s	speed-up	kn/s	speed-up
17-Queens	124,720.3	408,788.8	3.28	708,250.0	5.68	1,288,628.5	10.33	2,362,483.6	18.94
18-Queens	80,571.4	526,908.5	6.54	914,021.0	11.34	1,520,257.5	18.87	2,913,535.2	36.16
19-Queens	79,308.8	631,282.6	7.96	1,205,458.2	15.20	2,029,609.9	25.59	3,210,848.4	40.49
AVG	**94,866.8**	**522,326.6**	**5.93**	**942,576.4**	**10.74**	**1,612,832.0**	**18.26**	**2,828,955.7**	**31.86**
`ta028`	5,993.0	26,172.8	4.37	38,768.2	6.47	44,693.5	7.46	40,969.0	6.84
`ta025`	6,468.7	30,628.0	4.86	42,100.1	6.69	58,218.3	9.25	63,865.0	10.15
`ta026`	6,745.3	33,022.3	4.89	48,921.9	7.25	64,821.5	9.60	93,878.2	13.92
`ta024`	8,086.0	40,293.7	5.01	68,731.1	8.54	100,030.1	12.44	141,005.3	17.54
`ta021`	7,170.0	37,642.5	5.25	67,637.6	9.43	93,249.8	13.00	131,768.6	18.37
AVG	**6,892.6**	**33,551.8**	**4.88**	**53,231.7**	**7.68**	**72,202.6**	**10.35**	**94,297.2**	**13.36**

Need to investigate larger instances

Execution statistics

Table 4: Execution statistics of the largest instance solved for each problem using 128 CPU cores.

Instance	kn/s	Percentage of total execution time			
		Remove	Decompose	Insert	Termination
`ta030`	2,204.7	$< 1\%$	98%	$< 1\%$	$< 1\%$
17-Queens	269,751.7	18%	62%	10%	8%
`kp3`	161,505.5	42%	47%	5%	4%