Productivity-aware parallel cooperative combinatorial optimization for ultra-scale supercomputers

Guillaume HELBECQUE

PCOG Talks

April 18th, 2023

PhD supervisors:
Pr. Pascal BOUVRY, Université du Luxembourg
Pr. Nouredine MELAB, Université de Lille

Context

- Beginning of the exascale *era*[1] (June 2022);

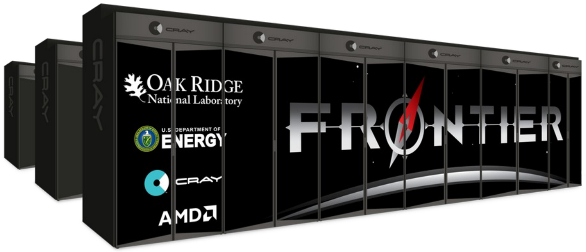


Fig. 1: The Frontier system at Oak Ridge National Laboratory.

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,730,112	1,102.00	1,685.65	21,100

Fig. 2: Frontier is the new No. 1 system in the Top500[1].

- Increasingly large (millions of cores), heterogeneous (CPU-GPU, *etc.*) and less and less reliable (Mean Time Between Failures – MTBF < 1h) systems[1];

- Evolutionary school (MPI+X) *vs.* revolutionary school (Partitioned Global Address Space (PGAS) - based environments).

1. Top500 ranking (edition of June 2022), https://www.top500.org/.

Context

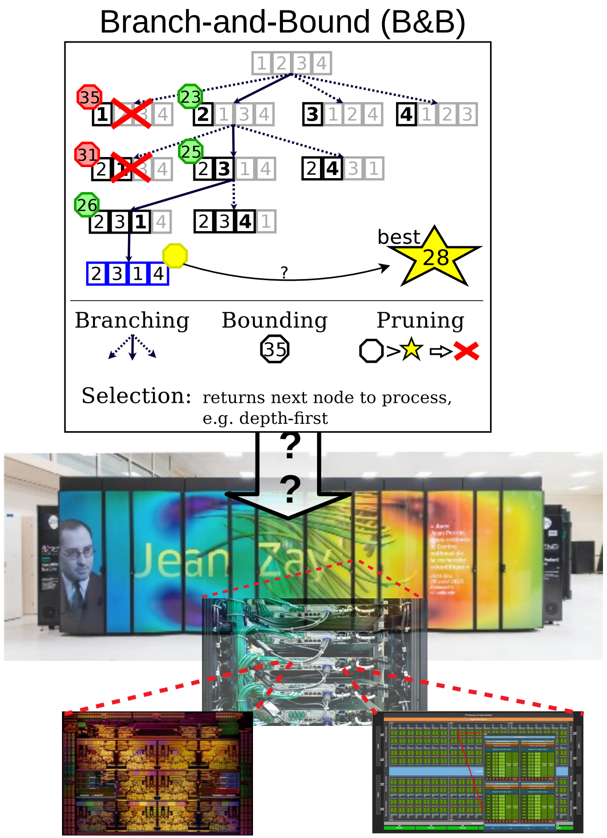


Fig. 3: Mapping B&B to hardware.

- Focus on exact Branch-and-Bound (B&B) optimization methods to solve combinatorial optimization problems:
  - Large tree size → Efficient data structure;
  - High irregularity → Efficient load balancing mechanism.
- **Motivating example:** Permutation Flowshop Scheduling Problems (PFSP). Search trees for very hard PFSP instances contain up to 10^{15} nodes.

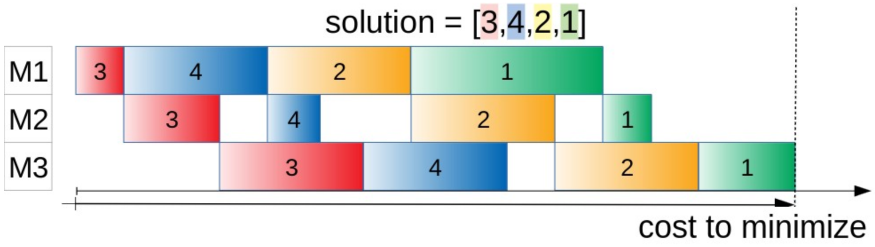


Fig. 4: Solution of a PFSP of size n=4.

State-of-the-art

- Most of existing parallel B&B algorithms are only guided by performance and benefit from problem-specific optimizations:
    - Multi-core CPUs: *[Mezmaz2014]*, *[Gmys2016];*
    - GPU and many-core: *[Chakroun2013a]*, *[Melab2018];*
    - Clusters of GPUs: *[Vu2016];*
    - Grid computing: *[Mezmaz2007]*, *[Drozdowski2011].*

- Few studies investigate the PGAS-oriented approach in the parallel optimization setting: *[Machado2013]*, *[Munera2013]*.

- Rise of the PGAS-based Chapel productivity-aware parallel programming language (HPE/Cray) *[Callahan2004]*.

- Many issues have to be investigate:
    - Dealing with scalability;
    - Handling GPU-based heterogeneity;
    - Address fault tolerance using checkpointing;
    - Combine parallel B&B with metaheuristics.

PhD outline

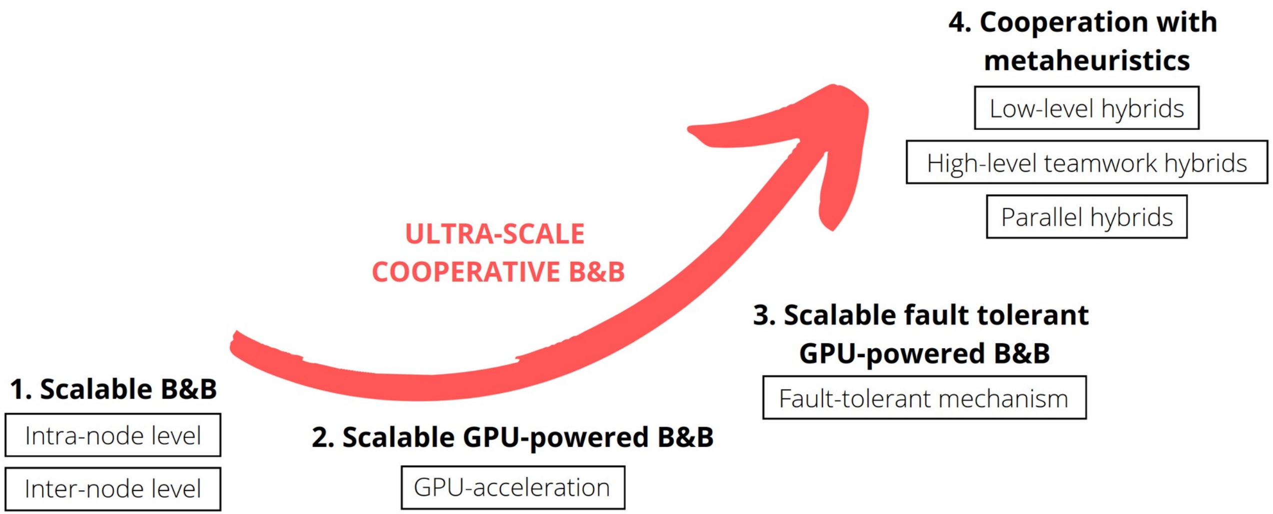


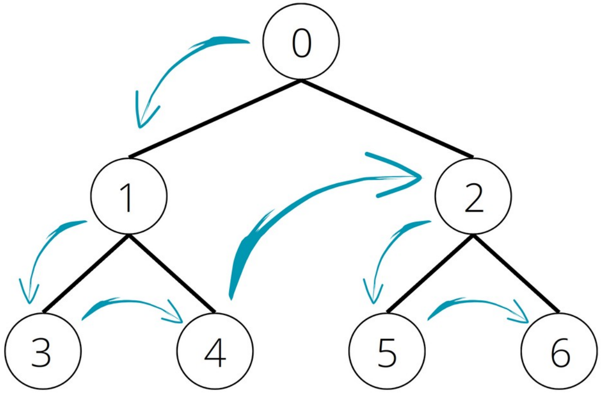
Fig. 6: PhD outline.

- Extensive experiments using the Jean Zay (FR) and Meluxina (EuroHPC/LU) petascale supercomputers.
- Support from the Chapel's team (HPE/Cray).

Scalable B&B - Parallel design and implementation

- Asynchronous parallel tree exploration model:
  - Unpredictable communications;
  - Unbalanced work units → need Work Stealing (WS).

- Depth-First tree-Search (DFS):
  - Memory Efficiency;
  - Stack (LIFO).



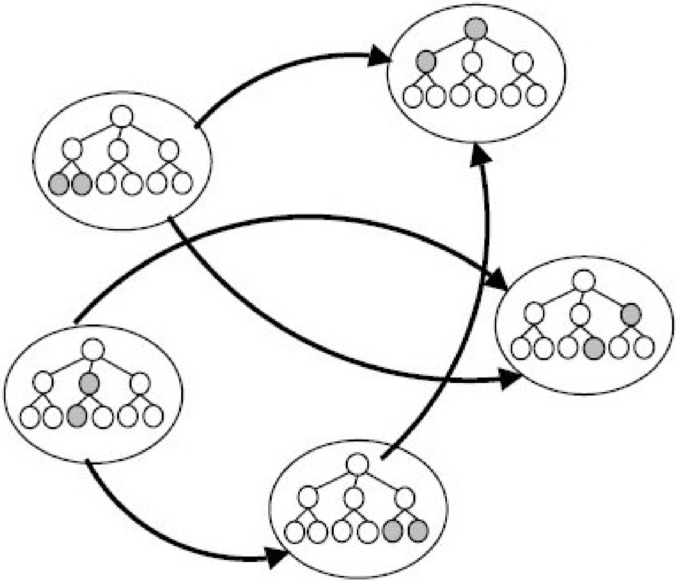
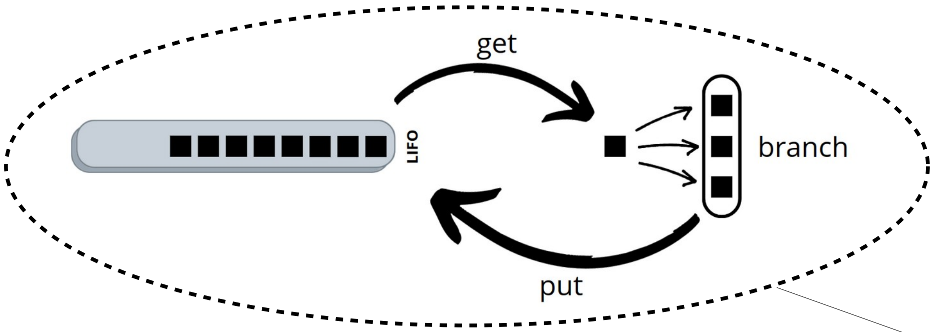


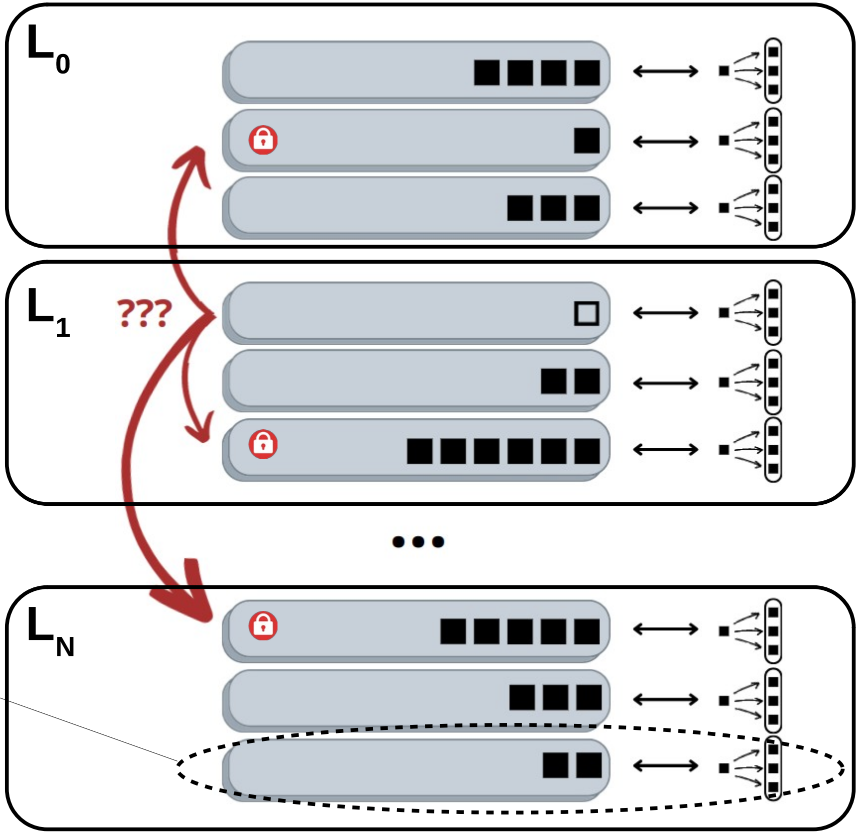
Fig. 7: Illustration of the parallel tree exploration model.
(from [Chakroun2013b])

Scalable B&B - Parallel design and implementation

Collegial multi-pool approach *[Gendron1994]*.

At the process level





Scalable B&B - The DistBag-DFS data structure

- DistBag[2] ("distributed bag"): user-defined parallel-safe distributed multi-set implementation.

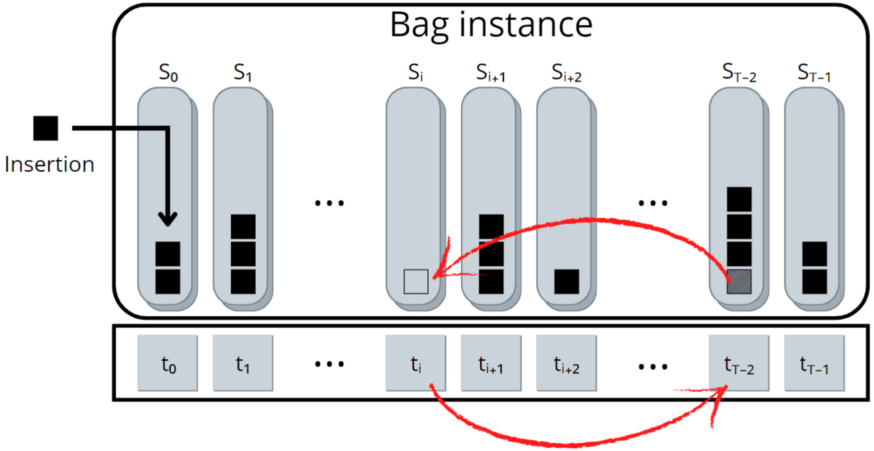


Fig. 8: The DistBag data structure.

→ not suitable for DFS tree-search.

2. The DistBag data structure, https://chapel-lang.org/docs/modules/packages/DistributedBag.html.

Scalable B&B - The DistBag-DFS data structure

Revisited into DistBag-DFS:

- Work pools → non-blocking split deque;

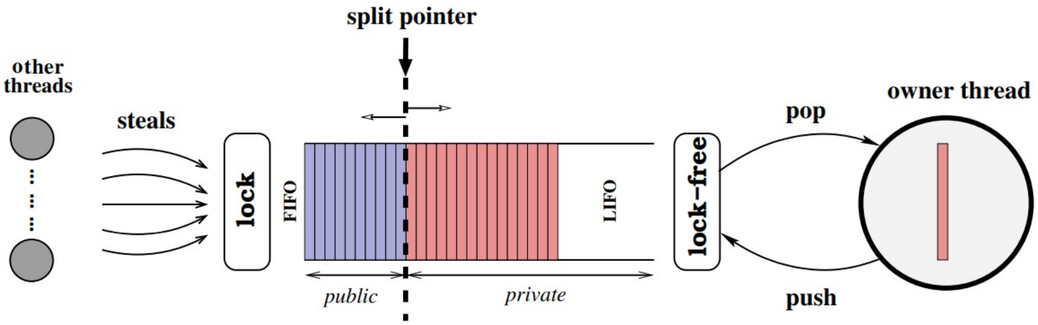


Fig. 9: Simplified view of a non-blocking split deque.
(from [Vu2016])

- New WS mechanism:
  - Bi-level (locality-aware);
  - Random victim selection;
  - Steal half.

Scalable B&B - Productivity-awareness

Sequential *vs.* Distributed parallel[3]:

```
proc tree_search_sequential(type Node, Problem problem){
  var root = new Node(problem); /* problem-specific */
  var pool = new Pool(Node);
  pool.add(root);

  while true {
    var (hasWork, parent): (int, Node) = pool.remove();
    /* Check termination condition */
    var children = problem.decompose(parent); /* problem-specific */
    pool.addBulk(children);
  }
}
```

```
proc tree_search_distributed(type Node, Problem problem){
  var root = new Node(problem); /* problem-specific */
  var bag = new DistBag_DFS(Node, Locales);
  bag.add(root, 0);

  coforall locId in 0..#numLocales do on Locales[locId] {
    coforall taskId in 0..#here.maxTaskPar {
      while true {
        var (hasWork, parent): (int, Node) = bag.remove(taskId);
        /* Check termination condition */
        var children = problem.decompose(parent); /* problem-specific */
        bag.addBulk(children, taskId);
        /* Sharing of global knowledge - problem-specific */
    }
  }
}
```

Support for:

- PFSP;
- 0/1-Knapsack;
- Unbalanced Tree-Search benchmark (UTS);
- N-Queens.

3. P3D-DFS, https://github.com/Guillaume-Helbecque/P3D-DFS.

Scalable B&B - Experimental results at the intra-node level

- P3D-DFS *vs.* OMP-PBB (OpenMP);
- Resolution of large PFSP instances;
- Aion cluster[4]: up to 128 processing cores;
- P3D-DFS outperforms its counterpart:
  - ≠ data structures;
  - ≠ synchronization mechanisms;
- On UTS, P3D-DFS is outperformed for finest-grained instances.

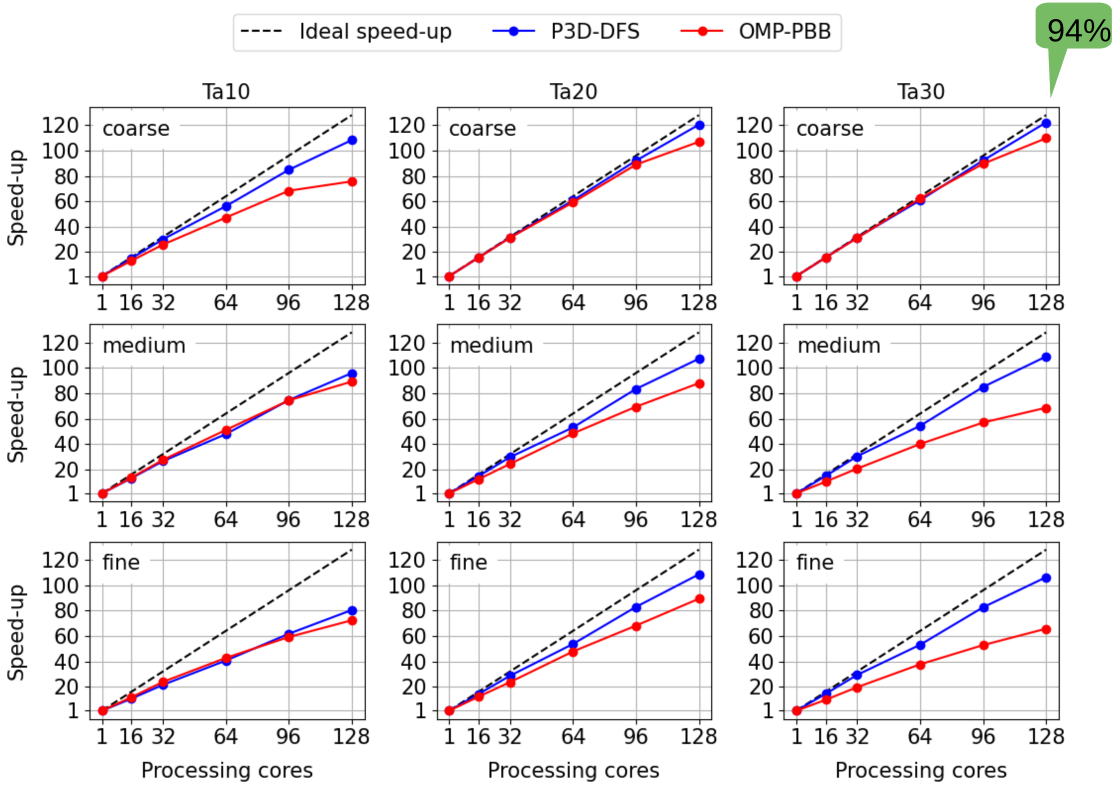


Fig. 10: Speed-up P3D-DFS *vs.* OMP-PBB in shared-memory experiments, Aion cluster.

4. ULHPC supercomputers - Aion system: https://hpc-docs.uni.lu/systems/aion/.

Scalable B&B - Experimental results at the inter-node level

- P3D-DFS *vs.* MPI-PBB (MPI+pthread);
- Resolution of large PFSP instances;
- Aion cluster[4]: up to 64 computer nodes (8192 processing cores);
- P3D-DFS competitive against its counterpart:
  - ≠ data structures;
  - ≠ WS mechanisms;
- Similar results on UTS.

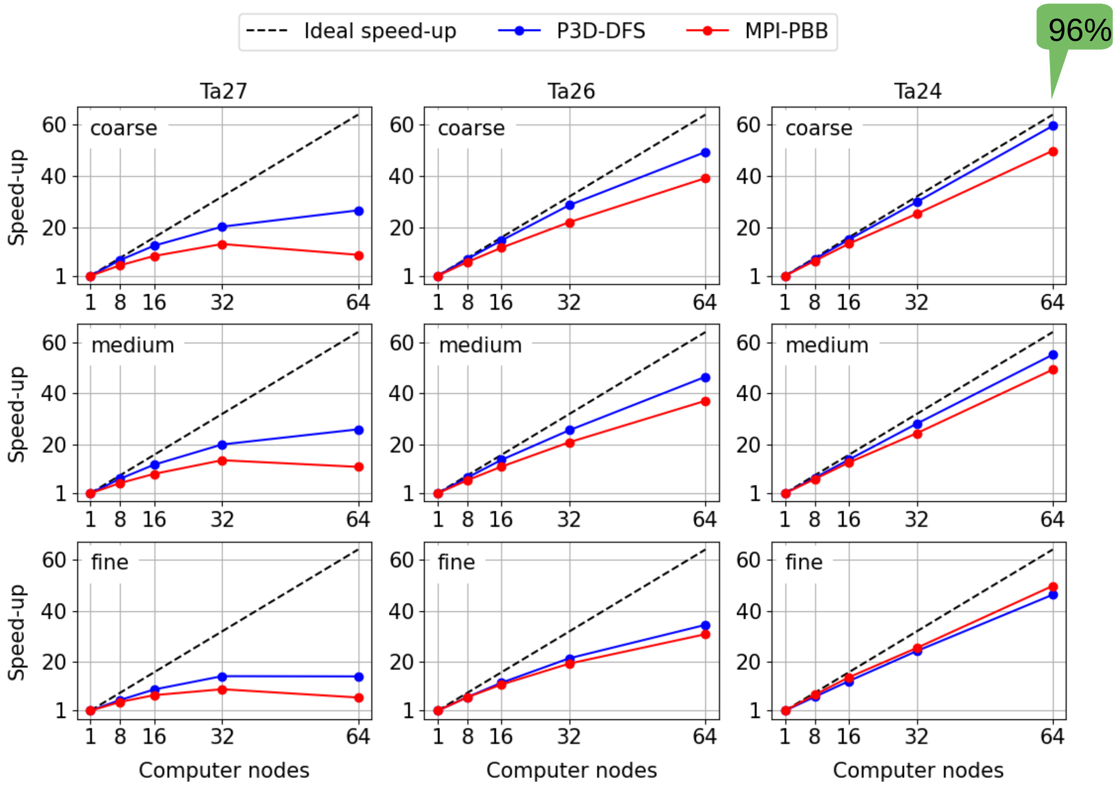


Fig. 11: Speed-up P3D-DFS *vs.* MPI-PBB in distributed-memory experiments, Aion cluster.

4. ULHPC supercomputers - Aion system: https://hpc-docs.uni.lu/systems/aion/.

Scalable GPU-powered B&B – Design

Generation of nodes → CPU / Evaluation of nodes → GPU

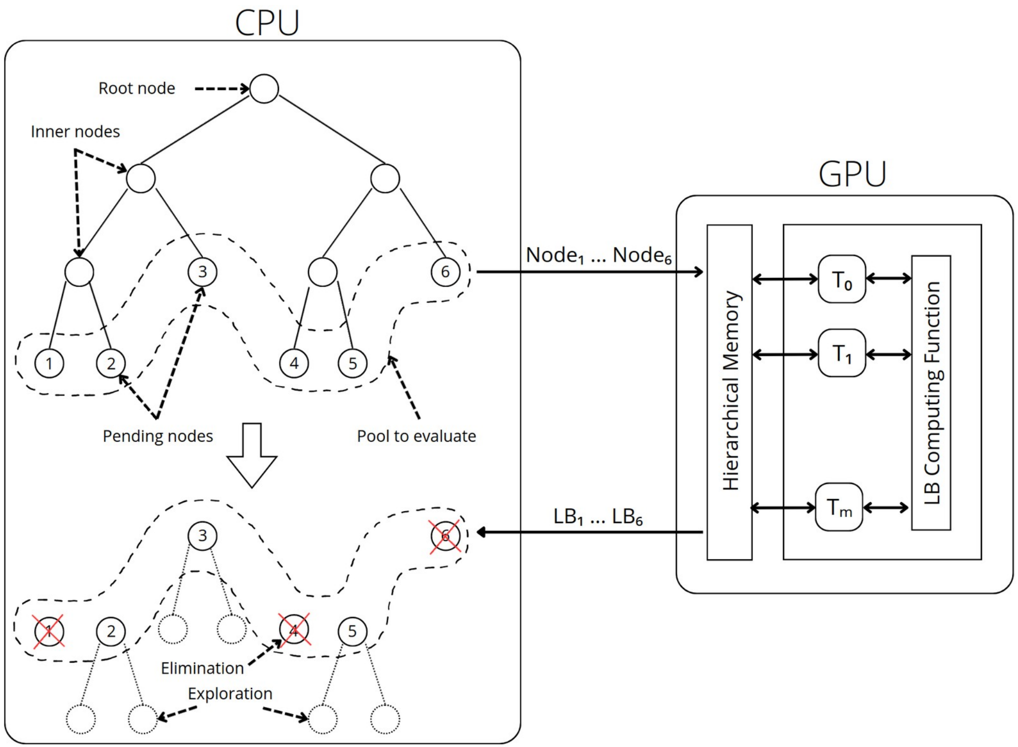


Fig. 12: Illustration of the parallel evaluation of bounds model for GPU-accelerated B&B.
(from [Chakroun2013b])

Scalable GPU-powered B&B – Implementation

Two main approaches:

- Chapel's native GPUs features:
  - Since Chapel 1.26.0 (March 2022);
  - Target Nvidia and AMD GPUs;
  - Generate and launch GPU kernels;
  - Under active development.

```
coforall gpu in here.gpus do on gpu {
  foreach i in 1..n {
    // do work on GPUs
  }
}
```

- GPUIterator and GPUAPI[4] modules:
  - User-defined package (Georgia Tech);
  - Target Nvidia, AMD and Intel GPUs;
  - Automate work distribution across CPUs and GPUs;
  - Need to handle Cuda/OpenCL kernels explicitly.

```
var GPUCallBack(/* args */){
  CudaKernel(/* args */);
}

forall i in GPU(1..n, GPUCallBack, CPUPercent){
  // do work on CPUs and/or GPUs
}
```

4. Chapel-GPU: GPUIterator and GPUAPI module for Chapel, https://github.com/ahayashi/chapel-gpu.

Conclusion & future works

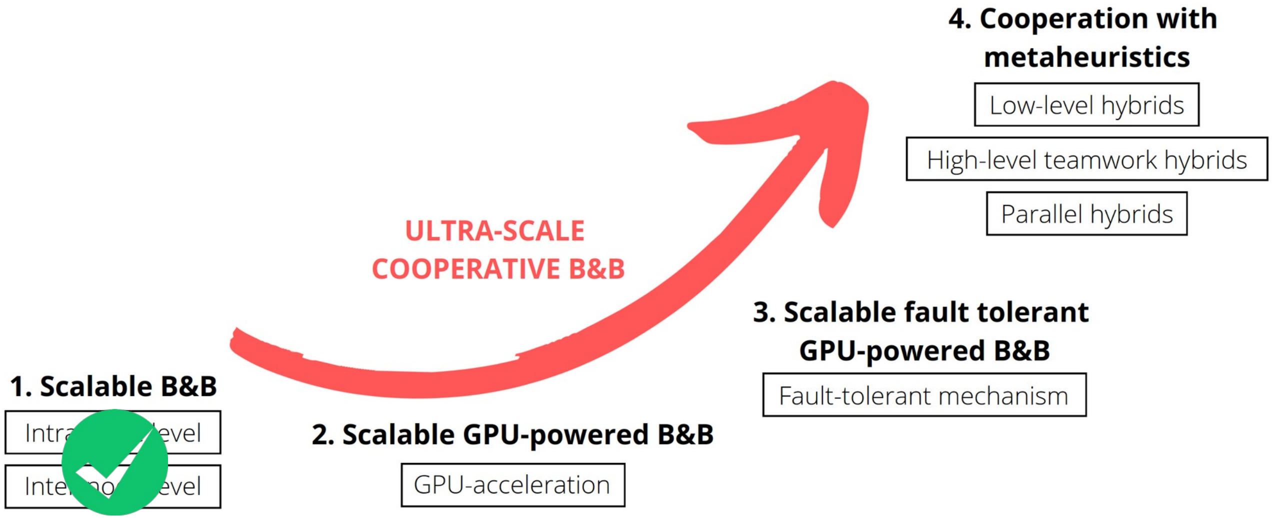


Fig. 13: PhD outline.

- Extension of P3D-DFS to other combinatorial optimization problems (QAP, TSP, *etc.*)
- Validations through extensive experiments
- Discussion with the Chapel’s team to incorporate DistBag-DFS in the language

Some references

[Callahan2004] D. Callahan, *et al*. The cascade high productivity language. In 9th *International Workshop on High-Level Parallel Programming Models and Supportive Environments*, 52–60, 2004.

[Chakroun2013a] I. Chakroun, *et al*. Combining multi-core and GPU computing for solving combinatorial optimization problems. *Journal of Parallel and Distributed Computing*, 73(12):1563–1577, 2013.

[Chakroun2013b] I. Chakroun. Parallel heterogeneous Branch and Bound algorithms for multi-core and multi-GPU environments. PhD dissertation, Université de Lille, 2013.

[Drozdowski2011] M. Drozdowski, *et al*. Grid branch-and-bound for permutation flowshop. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics - Volume Part II*, 21–30, Berlin, 2011.

[Gendron1994] B. Gendron, *et al*. Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 42(6):1042–1066, 1994.

[Gmys2016] J. Gmys, *et al*. Work stealing with private integer–vector–matrix data structure for multi-core branch-and-bound algorithms. *Concurrency and Computation: Practice and Experience*, 28(18):4463–4484, 2016.

Some references

[Machado2013] R. Machado, *et al*. Parallel local search: Experiments with a PGAS-based programming model. Abs/1301.7699, 2013.

[Melab2018] N. Melab, *et al*. Multi-core versus many-core computing for many-task branch-and-bound applied to big optimization problems. *Future Generation Computer Systems*, 82:472–481, 2018.

[Mezmaz2007] M. Mezmaz, *et al*. A grid-enabled branch and bound algorithm for solving challenging combinatorial optimization problems. In *2007 IEEE International Parallel and Distributed Processing Symposium*, 1–9, 2007.

[Mezmaz2014] M. Mezmaz, *et al*. A multi-core parallel branch-and-bound algorithm using factorial number system. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 1203–1212, 2014.

[Munera2013] D. Munera, *et al*. Experimenting with X10 for parallel constraint-based local search. Abs/1307.4641, 2013.

[Vu2016] T. Vu, *et al*. Parallel branch-and-bound in multi-core multi-CPU multi-GPU heterogeneous environments. *Future Generation Computer Systems*, 56:95–109, 2016.

Thank you for your attention.

guillaume.helbecque@uni.lu
MNO E02 0225-060

https://github.com/Guillaume-Helbecque