Towards a scalable load balancing for productivity-aware tree-search

G. Helbecque[†], J. Gmys[†], T. Carneiro*, N. Melab[†], P. Bouvry*

[†]Université de Lille, CNRS/CRIStAL UMR 9189, Centre Inria de l'Université de Lille, France
*Université du Luxembourg, DCS-FSTM/SnT, Luxembourg

10th annual Chapel Implementers and Users Workshop (CHIUW)
June 2, 2023

Context

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

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	8,699,904	1,194.00	1,679.82	22,703

Fig. 1: Frontier is the No. 1 system in the Top500[1] (since June 2022).

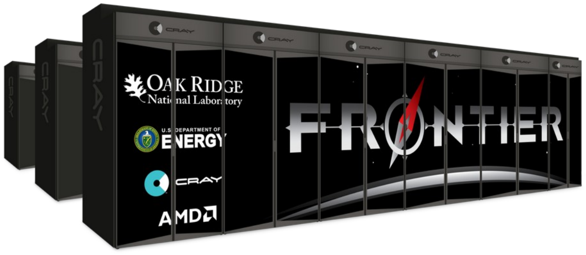


Fig. 2: The Frontier system at Oak Ridge National Laboratory.

- 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: https://www.top500.org/.

Context

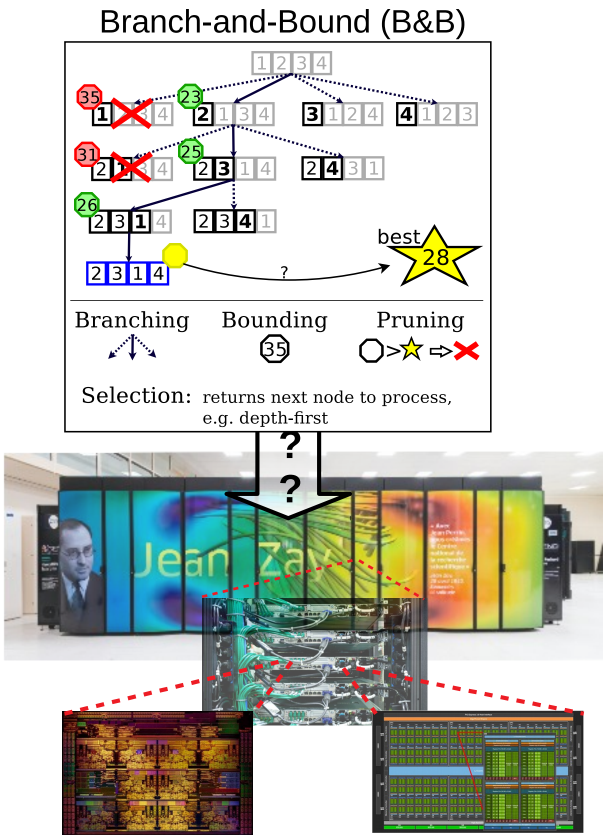


Fig. 3: Mapping B&B to hardware.

- Focus on exact Branch-and-Bound (B&B) optimization methods to solve combinatorial optimization problems:
  - Large tree size → Efficient data structure;
  - High irregularity → Efficient dynamic load balancing mechanism.

- **Motivating example:** Permutation Flowshop Scheduling Problem (PFSP). Search trees for very hard PFSP instances contain up to 10^{15} explored nodes.

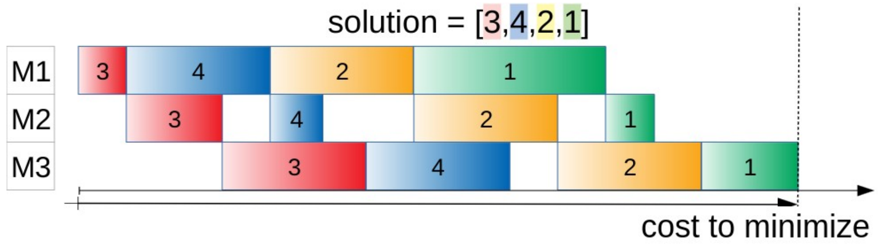


Fig. 4: Solution of a PFSP instance of 4 jobs and 3 machines.

State-of-the-art overview

- 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: *[Chakroun2013]*, *[Melab2018]*;
  - Clusters of GPUs: *[Vu2016]*;
  - Grid computing: *[Mezmaz2007]*, *[Drozdowski2011]*.

- Emergence of the PGAS-based Chapel productivity-aware parallel programming language (HPE/Cray): *[Callahan2004]*, *[Carneiro2020]*.

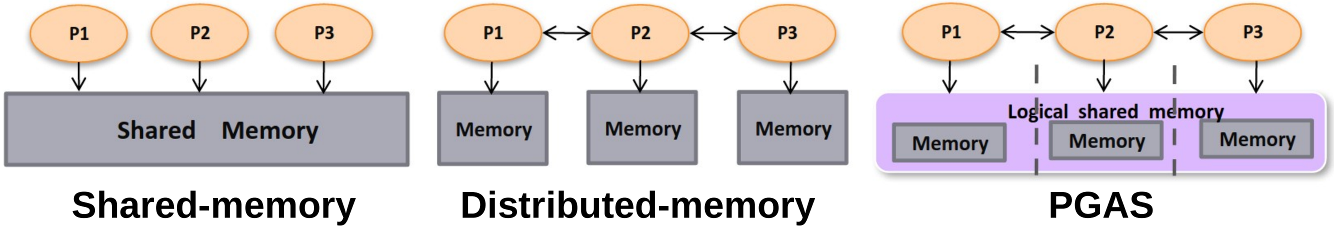


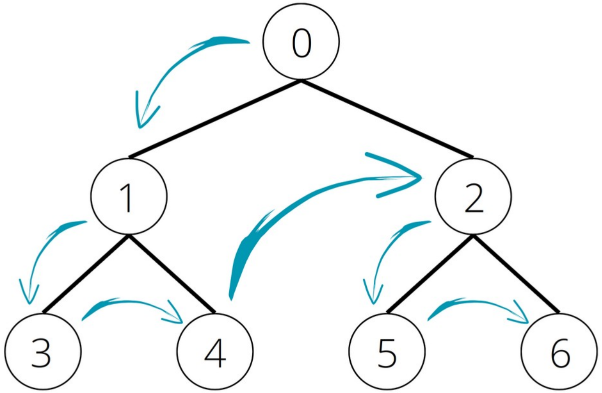
Fig. 5: Parallel programming models.

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

Parallel design

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

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



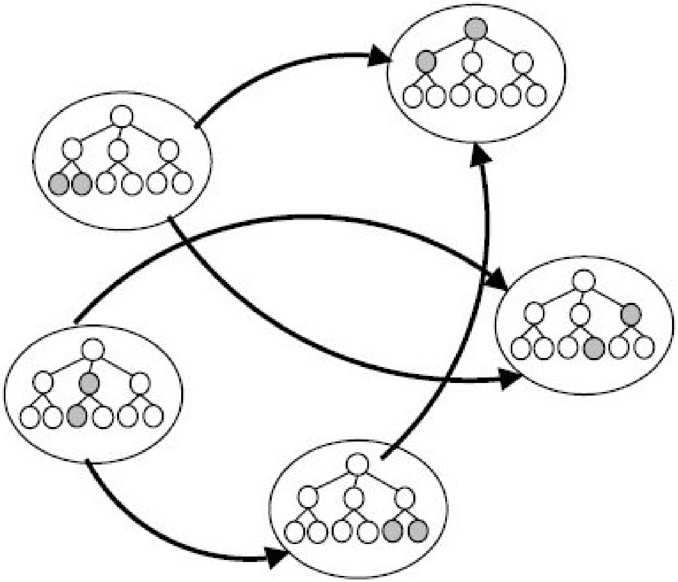
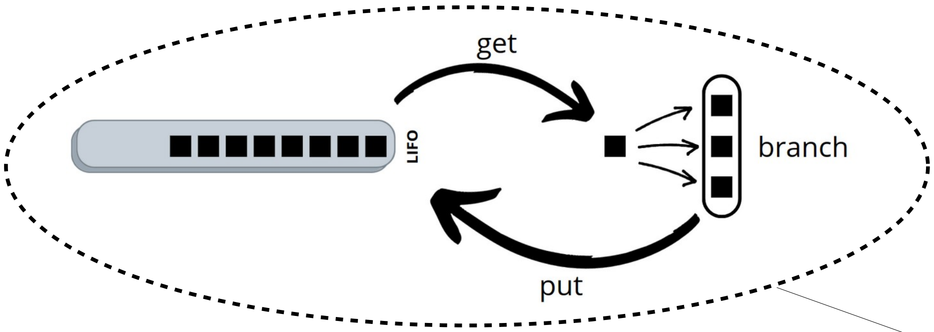


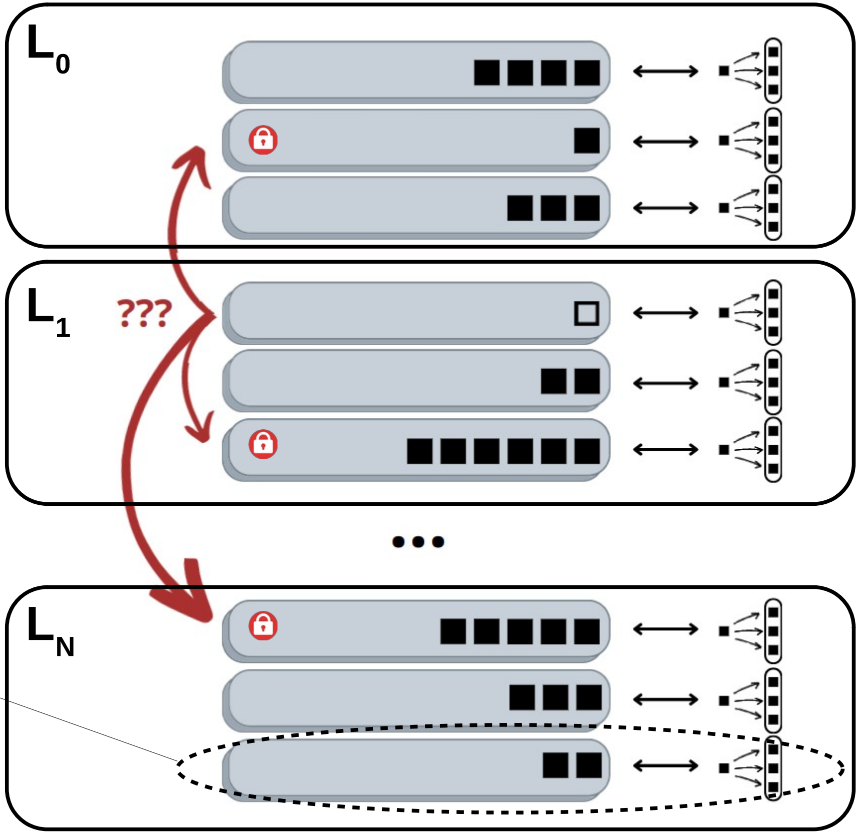
Fig. 6: Illustration of the parallel tree exploration model.

Parallel design

Collegial multi-pool *[Gendron1994]*.

At the process level





Parallel implementation

`DistBag`[2] ("distributed bag"): parallel-safe distributed multi-pool implementation;

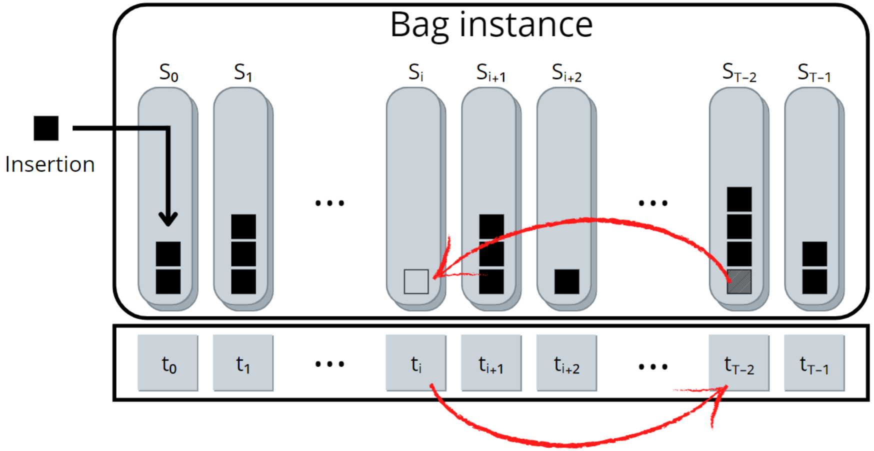


Fig. 7: The `DistBag` data structure.

```
var bag = new DistBag(int, Locales);
bag.addBulk(1..N);

coforall taskId in 0..#here.maxTaskPar {
  var (status, i) = bag.remove();
  // ...
  bag.add(j);
}
```

→ not suitable for DFS.

2. The Chapel's `DistributedBag` module: https://chapel-lang.org/docs/modules/packages/DistributedBag.html.

Parallel implementation

Revisited into `DistBag-DFS`[3]:

- Work pools → non-blocking split deques *[vanDijk2014]*, *[Dinan2009]*;

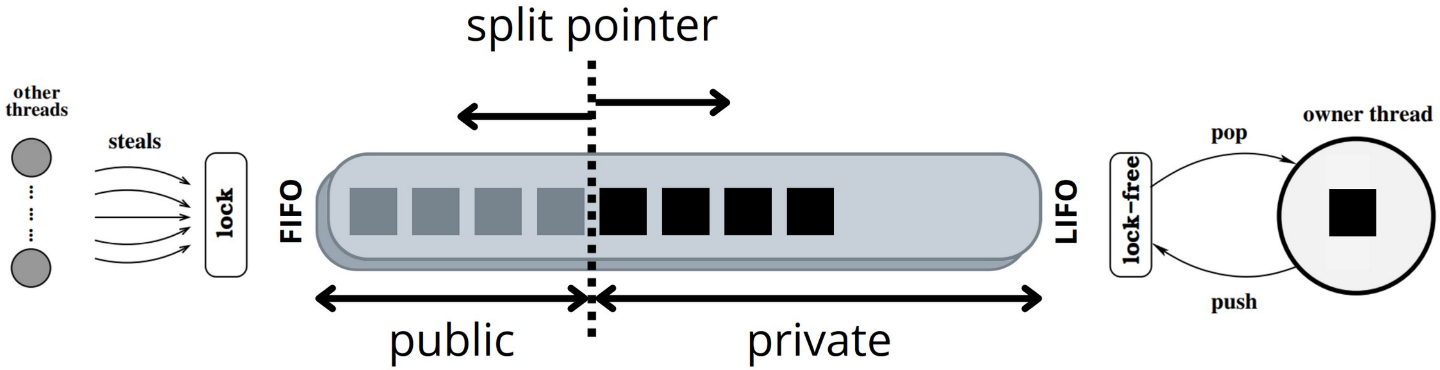


Fig. 8: Simplified view of a non-blocking split deque.

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

3. The `DistBag-DFS` data structure: https://github.com/Guillaume-Helbecque/P3D-DFS/DistBag-DFS.

Productivity-awareness

Sequential *vs.* distributed parallel[4]:

```
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 */
    }
  }
}
```

→ Few more lines of code are required;
→ Generic approach: *e.g.* PFSP, Unbalanced Tree-Search benchmark (UTS), N-Queens.

4. G. Helbecque, *el al*. Productivity- and Performance-aware Parallel Distributed Depth-First Search (P3D-DFS), 2023. https://github.com/Guillaume-Helbecque/P3D-DFS.

Experimental protocol

- **Hardware:** ULHPC facilities[5]
  2x AMD Epyc ROME 7H12 @ 2.6 GHz (64 cores), 256 GB RAM;
  Fast InfiniBand HDR100 network.

- **Applications:**
  - PFSP → Taillard's instances (20 jobs x 20 machines) *[Taillard1993]*;
  - UTS → synthetic trees.

- **Experiments:**
  - Memory consumption of `DistBag` *vs.* `DistBag-DFS`;
  - P3D-DFS *vs.* MPI+X best known counterparts (≠ approaches).

Fig. 9: The Aion system.

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

DistBag *vs.* DistBag-DFS

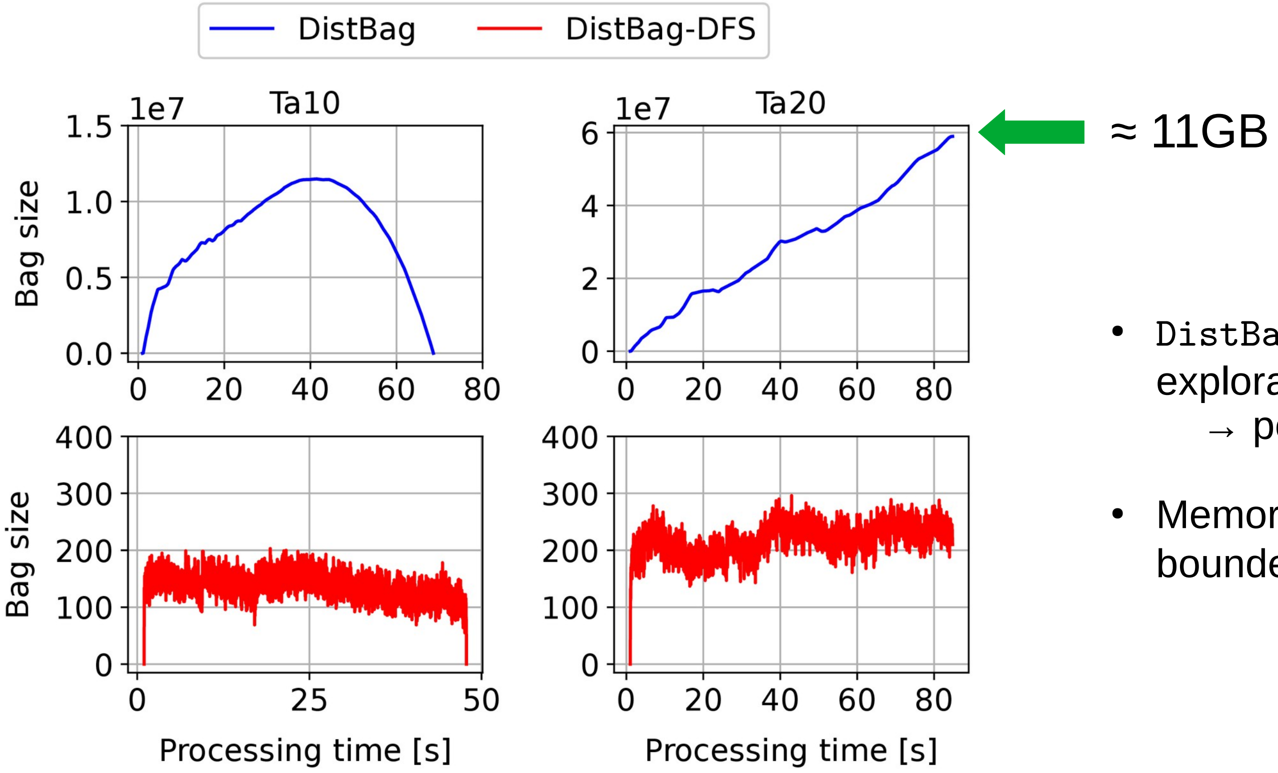


Fig. 10: Bag size according to the processing time when solving PFSP instances in DFS.

- `DistBag` cannot ensure DFS exploration order
  → poor memory efficiency;
- Memory consumption remains bounded using `DistBag-DFS`.

Experimental results at the inter-node level

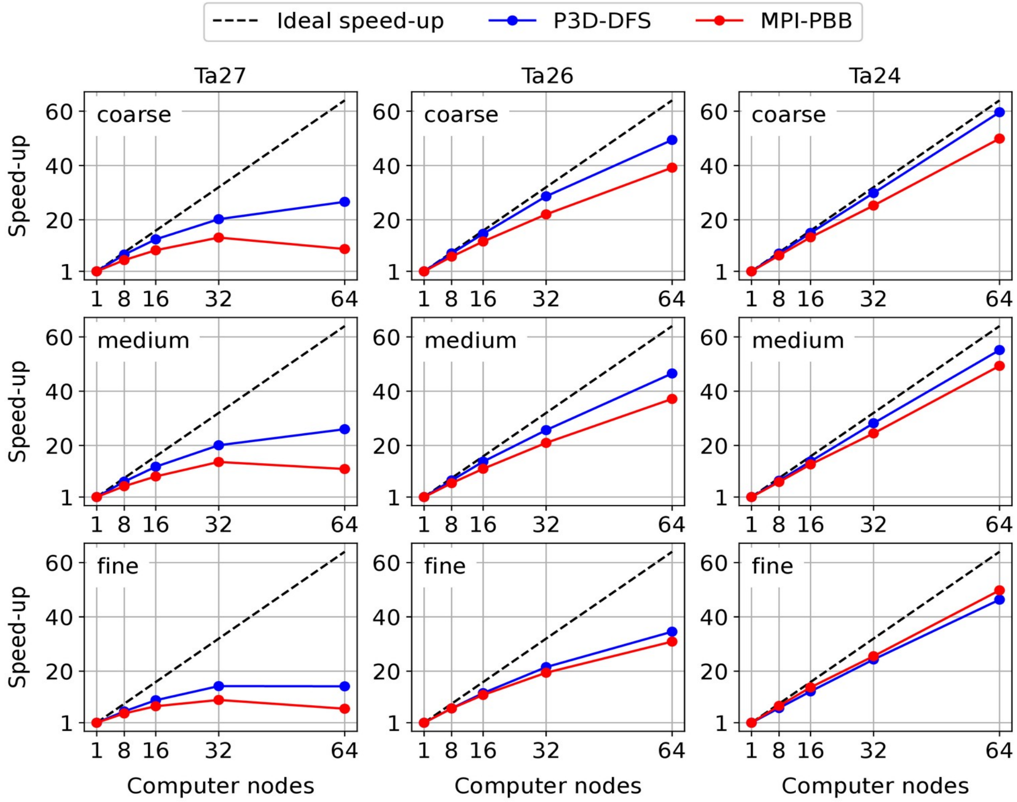


Fig. 11: Absolute speed-up P3D-DFS *vs.* MPI-PBB in distributed-memory experiments.

94% of ideal speed-up

- P3D-DFS *vs.* MPI-PBB[6] (MPI+pthread);
- P3D-DFS competitive against its counterpart:
  - ≠ WS mechanisms.
- MPI-PBB performs better solving the largest instance with the finest granularity:
  - `DistBag-DFS` overheads (?).

6. J. Gmys. Parallel Branch-and-Bound for permutation-based optimization, 2023. https://doi.org/10.5281/zenodo.7674826.

Experimental results at the inter-node level

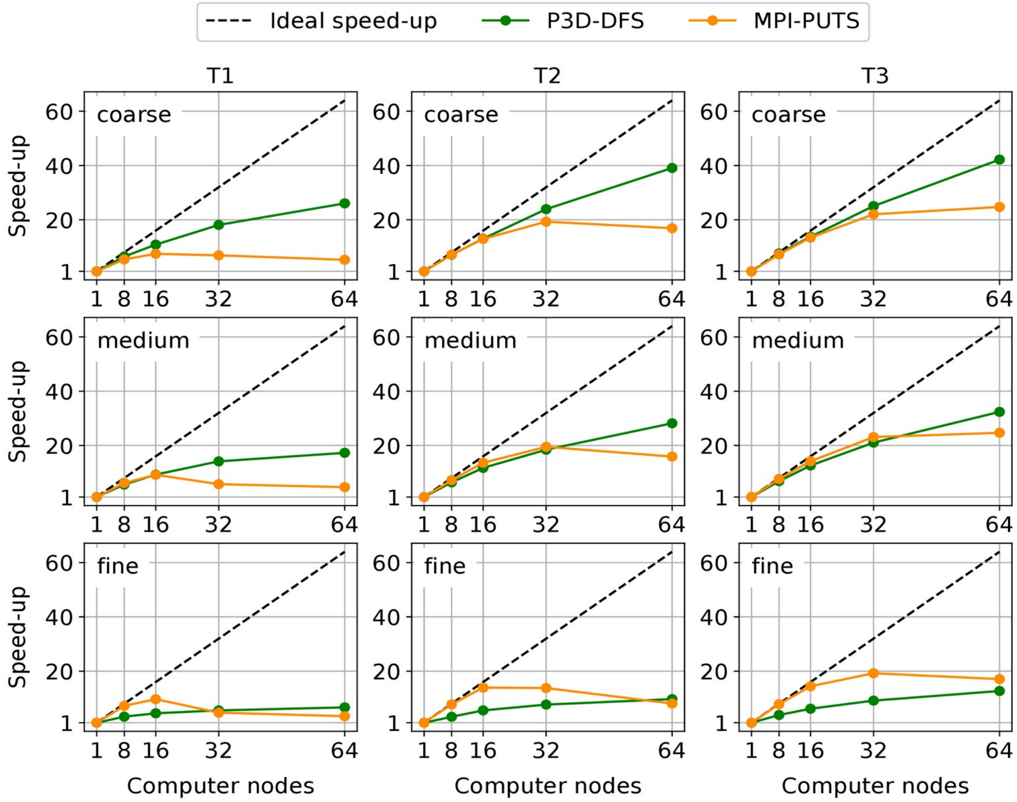


Fig. 12: Absolute speed-up P3D-DFS *vs.* MPI-PUTS in distributed-memory experiments.

66% of ideal speed-up

- P3D-DFS *vs.* MPI-PUTS[7] (MPI+MPI);
- P3D-DFS outperforms its counterpart, at medium- and coarse-grain:
  - ≠ WS mechanisms.
- P3D-DFS outperformed at fine-grain:
  - `DistBag-DFS` overheads (?);
  - poor intra-node speed-up.

7. J. Dinan, *et al*. The Unbalanced Tree-Search benchmark, 2022. https://doi.org/10.5281/zenodo.7328332.

Conclusion & future works

- `DistBag-DFS` allows high-productivity and good performance for coarser-grained applications;
  → P3D-DFS competitive to MPI+X baselines, in terms of performance and productivity-awareness.

- Investigate and benchmark `DistBag-DFS` low-level mechanisms;

- Extend P3D-DFS to other combinatorial optimization problems:
  - *e.g.* Quadratic assignment problems, Traveling salesman problems.

- Extend experiments to larger systems;

- Develop a Chapel's `DistributedBag-DFS` package module (?).

Suggestions are welcomed!

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.

[Carneiro2020] T. Carneiro, *et al*. Towards ultra-scale Branch-and-Bound using a high-productivity language. *Future Generation Computer Systems*, 105:196-209, 2020.

[Chakroun2013] 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.

[Dinan2009] J. Dinan, *el al*. Scalable Work Stealing. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, 2009.

[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.

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

Some references

[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.

[Taillard1993] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[vanDijk2014] T. van Dijk, *et al*. Lace: Non-blocking Split Deque for Work-Stealing. In *Euro-Par 2014: Parallel Processing Workshops*, 206–217, 2014.

[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.

Contact:
Guillaume HELBECQUE
guillaume.helbecque@univ-lille.fr

https://github.com/Guillaume-Helbecque