24th International Conference on Computational Science

PGAS Data Structure for Unbalanced Tree-Based Algorithms at Scale

G. Helbecque[1,2], T. Carneiro[3], N. Melab[2], J. Gmys[2], P. Bouvry[1]

[1]University of Luxembourg, DCS-FSTM/SnT, Luxembourg
[2]Université de Lille, CNRS/CRIStAL UMR 9189, Centre Inria de l'Université de Lille, France
[3]Interuniversity Microelectronics Centre, Belgium

UNIVERSITÉ DU LUXEMBOURG

Université de Lille

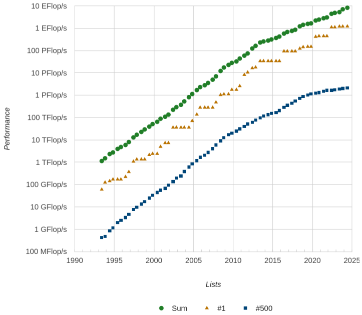
imec

2-4 July 2024
Málaga, Spain

Outline

Motivation

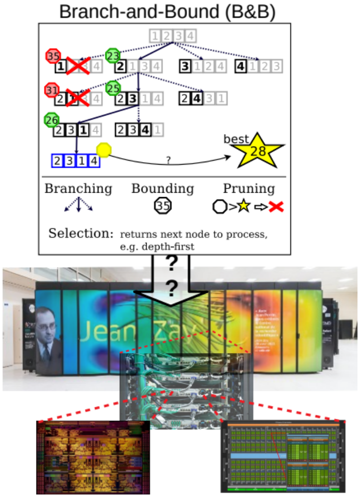
- Exascale era of computation;



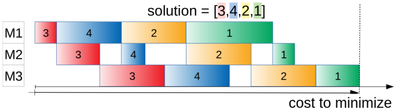
- Increasingly large (millions of cores), heterogeneous (CPU-GPU, etc.), and less and less reliable (Mean Time Between Failures – MTBF $<$ 1h) systems[1];
- "Evolutionary approaches" (MPI+X) vs. "revolutionary approaches" (e.g., Partitioned Global Address Space (PGAS) -based environments).

[1] Bi-annual TOP500 ranking, `https://www.top500.org/`.

Motivation



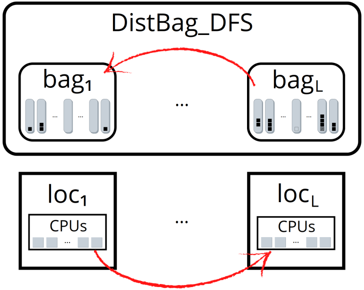
- Focus on parallel tree-search methods for solving combinatorial problems, e.g., Backtracking and Branch-and-Bound (B&B):
  - Large trees → efficient data structure
  - Irregular trees → efficient load balancing
- Motivating example: Permutation Flowshop Scheduling Problem (PFSP). Search trees for hard PFSP instances contain up to 10^{15} explored nodes.



Related work

- Limitations of existing MPI+X data structures and load balancing for parallel tree-search algorithms, e.g. [1, 2]:
  - focus only on performance
  - combine low-level programming environments
- PGAS-based load balancing techniques also exist [3, 4, 5], but none in Chapel.
- In PGAS Chapel, we introduced the `DistBag_DFS` distributed data structure [6], but ...
  - The description of the data structure could be extended
  - Load balancing mechanism not evaluated
  - Lack of performance evaluation at scale
  - Not included in the language (user-defined library)

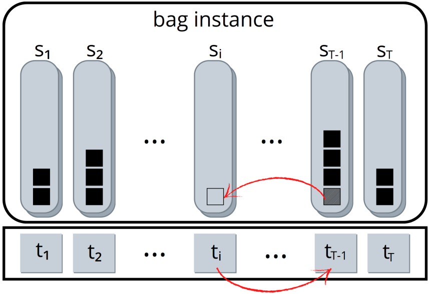
DistBag_DFS's components: bag instances



- One bag instance per Chapel *locales*
  $\rightarrow$ Exploit inter-node level of parallelism

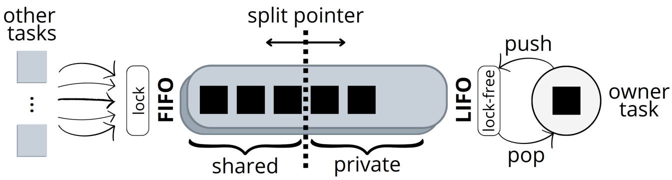
- Each bag instance maintains a multi-pool

DistBag_DFS's components: multi-pools



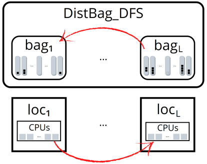
- One pool per Chapel *tasks*
  $\rightarrow$ Exploit intra-node level of parallelism
- Each pool is indexed by a task ID
  $\rightarrow$ Ensure DFS

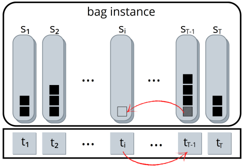
DistBag_DFS's components: pools

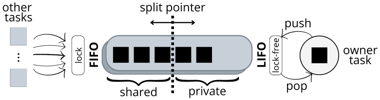


- Non-blocking double-ended queues (deques) [7]
  → lock-free local access to the private portion
  → copy-free transfer between shared and private portions
- Dynamic-sized: 1024×2^k

DistBag_DFS's components: dynamic load balancing







Dynamic Work Stealing (WS):

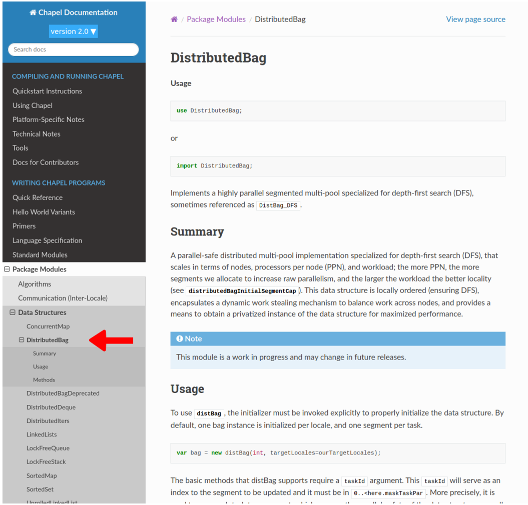
- Locality-aware: local, then global
- Random victim selection
- Steal-one strategy locally, steal-half otherwise

WS fails when all pools have been visited and no work has been stolen.

DistBag_DFS's user interface

- Two initialization variables: `eltType` and `targetLocales`
- Three local procedures:
  - `add`: insert an element
  - `addBulk`: insert elements in bulk
  - `remove`: remove an element (contains WS)
- Four global procedures:
  - `clear`: clear `DistBag_DFS`
  - `these`: iterate over `DistBag_DFS`
  - `contains`: check if a given element is in `DistBag_DFS`
  - `getSize`: get the global size of `DistBag_DFS`

Integration to Chapel


Chapel Documentation
version 2.0
Search docs
COMPILING AND RUNNING CHAPEL
Quickstart Instructions
Using Chapel
Platform-Specific Notes
Technical Notes
Tools
Docs for Contributors
WRITING CHAPEL PROGRAMS
Quick Reference
Hello World Variants
Primers
Language Specification
Standard Modules
Package Modules
Algorithms
Communication (Inter-Locale)
Data Structures
ConcurrentMap
DistributedBag
Summary
Usage
Methods
DistributedBagDeprecated
DistributedDeque
DistributedIters
LinkedLists
LockFreeQueue
LockFreeStack
SortedMap
SortedSet

Package Modules / DistributedBag   View page source

DistributedBag

Usage

```
use DistributedBag;
```

or

```
import DistributedBag;
```

Implements a highly parallel segmented multi-pool specialized for depth-first search (DFS), sometimes referenced as `DistBag_DFS`.

Summary

A parallel-safe distributed multi-pool implementation specialized for depth-first search (DFS), that scales in terms of nodes, processors per node (PPN), and workload; the more PPN, the more segments we allocate to increase raw parallelism, and the larger the workload the better locality (see `distributedBagInitialSegmentCap`). This data structure is locally ordered (ensuring DFS), encapsulates a dynamic work stealing mechanism to balance work across nodes, and provides a means to obtain a privatized instance of the data structure for maximized performance.

Note
This module is a work in progress and may change in future releases.

Usage

To use `distBag`, the initializer must be invoked explicitly to properly initialize the data structure. By default, one bag instance is initialized per locale, and one segment per task.

```
var bag = new distBag(int, targetLocales=ourTargetLocales);
```

The basic methods that distBag supports require a `taskId` argument. This `taskId` will serve as an index to the segment to be updated and it must be in `0..<here.maskTaskPar`. More precisely, it is

Released in Chapel 2.0 (March 2024) in the `DistributedBag` package module:

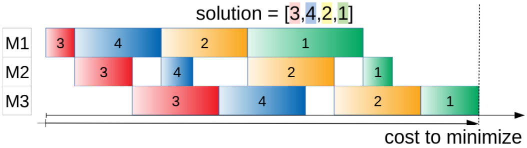
Usage

```
use DistributedBag;

var bag = new distBag(int);
// your code ...
```

Experimental protocol

- Applications:
  - Backtracking applied to the Unbalanced Tree-Search (UTS) benchmark [8], with binary- and geometric-shaped trees
  - B&B applied to the Permutation Flowshop Scheduling Problem (PFSP)



- Testbed: MeluXina - Cluster module (`https://docs.lxp.lu/`)
  - 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 1.31.0

Speed-up solving UTS instances

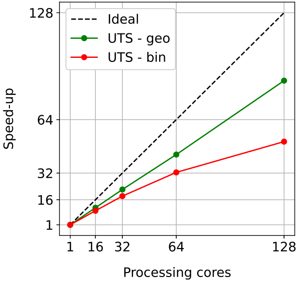


Fig. 1: Speed-up achieved solving geometrical and binomial synthetic UTS trees.

- 68% of the ideal speed-up solving `UTS-geo`
- 40% more than `UTS-bin`
- High ratio of WS success

Inst.	**Nb. of nodes** (10^6)	**Time (s)**	**nodes/s** (10^3)	**WS attempts (% success)**
`UTS-geo`	171.1	37.38	4,577	48,433 (99.0%)
`UTS-bin`	131.7	37.11	3,548	1,473,048 (96.8%)

Load balancing solving the UTS-bin instance

Workload distribution solving **UTS-bin** using 16, 32, 64, and 128 Chapel tasks:

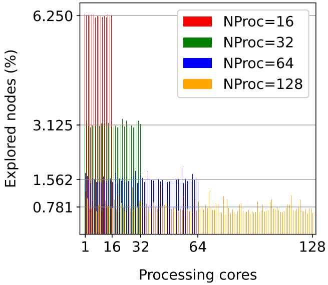


Fig. 2: Percentage of explored nodes per Chapel tasks solving the UTS-bin instance.

Even workload distribution for each experiment, i.e., 100/NbTasks.

Strong scaling efficiency solving PFSP instances

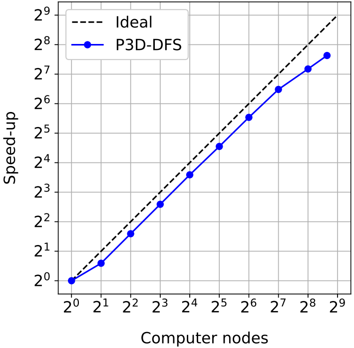


Fig. 3: Speed-up achieved solving `ta056`, compared to a multi-core version.

50% of strong scaling efficiency using 400 compute nodes (51,200 CPU cores) solving `ta056`

Conclusions

In the context of tree-search methods for combinatorial problems:

- `DistBag_DFS` provides high-level abstractions for unbalanced tree-search at scale
- 68% of the linear speed-up on a fine-grain backtracking application in single-node setting
- 50% of strong scaling efficiency using 400 compute nodes on a B&B application

Future works

- Pursue `DistBag_DFS` development:
  - Investigate ways to remove the required task ID in insertion/retrieval operations
  - Track its performance along Chapel's releases
  - Improve existing features and/or add new ones
  - Collect users feedbacks
- Further experiment `DistBag_DFS`:
  - Solve unsolved PFSP instances
  - Solve other problems (e.g., 0/1-Knapsack)
  - Extend our `DistBag_DFS`-based algorithms with a fault-tolerance mechanism

References

[1] T. Carneiro Pessoa, J. Gmys, F. H. de Carvalho Júnior, and et al., “GPU-accelerated backtracking using CUDA Dynamic Parallelism,” *Concurrency and Computation: Practice and Experience*, vol. 30, no. 9, p. e4374, 2018.

[2] J. Gmys, R. Leroy, M. Mezmaz, and et al., “Work stealing with private integer–vector–matrix data structure for multi-core branch-and-bound algorithms,” *Concurrency and Computation: Practice and Experience*, vol. 28, no. 18, pp. 4463–4484, 2016.

[3] J. Dinan, D. B. Larkins, P. Sadayappan, and et al., “Scalable Work Stealing,” in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Association for Computing Machinery, 2009.

[4] R. Machado, C. Lojewski, S. Abreu, and et al., “Unbalanced tree search on a manycore system using the GPI programming model,” *Computer Science - Research and Development*, vol. 26, no. 3, pp. 229–236, 2011.

[5] S. Olivier and J. Prins, “Scalable Dynamic Load Balancing Using UPC,” in *37th International Conference on Parallel Processing*, pp. 123–131, 2008.

[6] G. Helbecque, J. Gmys, N. Melab, and et al., “Parallel distributed productivity-aware tree-search using Chapel,” *Concurrency and Computation: Practice and Experience*, vol. 35, no. 27, p. e7874, 2023.

[7] T. van Dijk and J. C. van de Pol, “Lace: Non-blocking Split Deque for Work-Stealing,” in *Euro-Par 2014: Parallel Processing Workshops*, pp. 206–217, 2014.

[8] S. Olivier, J. Huan, J. Liu, and et al., “UTS: An Unbalanced Tree Search Benchmark,” in *Languages and Compilers for Parallel Computing*, pp. 235–250, 2007.

Thank you for your attention.

Contact:
Guillaume HELBECQUE
`guillaume.helbecque@uni.lu`

Work supported by the Agence Nationale de la Recherche (ref. ANR-22-CE46-0011) and the Luxembourg National Research Fund (ref. INTER/ANR/22/17133848), under the UltraBO project.

Luxembourg National Research Fund

anr agence nationale de la recherche