14th IEEE Workshop Parallel / Distributed Combinatorics and Optimization

GPU-Accelerated Tree Search in Chapel versus CUDA and HIP

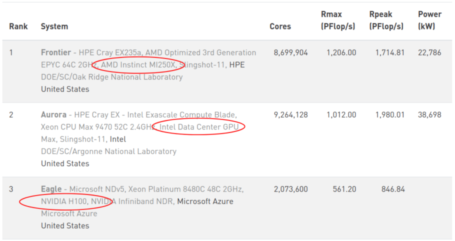
G. Helbecque[1,2], E. Krishnasamy[1], N. Melab[2], P. Bouvry[1]

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

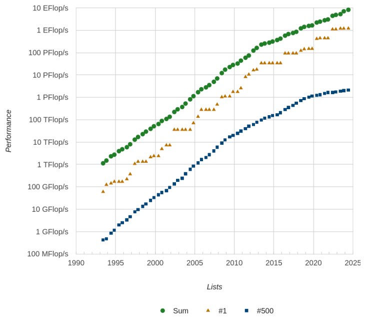
31 May, 2024
San Francisco, USA

Motivation

- Exascale era of computation;



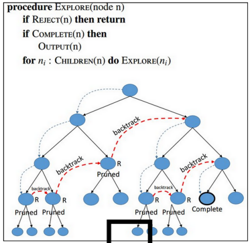
Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	1,714.81	22,786
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	



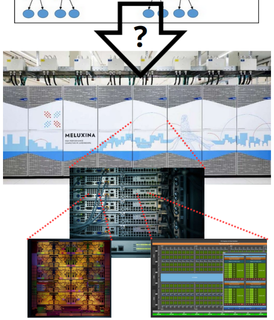
- Increasingly large (millions of cores), heterogeneous (CPU-GPU, etc.), and less and less reliable (Mean Time Between Failures – MTBF < 1h) systems[1];
- "Evolutionary approaches" (MPI+X) vs. "revolutionary approaches" (e.g., Partitioned Global Address Space (PGAS) -based environments).

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

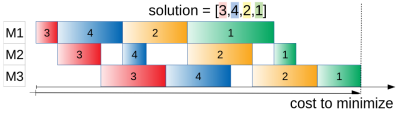
Motivation



```
procedure EXPLORE(node n)
    if REJECT(n) then return
    if COMPLETE(n) then
        OUTPUT(n)
    for $n_i$ : CHILDREN(n) do EXPLORE($n_i$)
```



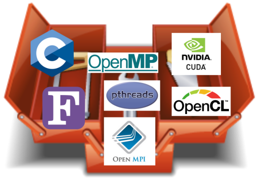
- Focus on GPU-accelerated tree search methods for solving combinatorial problems, e.g., Backtracking and Branch-and-Bound (B&B).
  → Large and irregular trees
- Motivating example: Permutation Flowshop Scheduling Problem (PFSP). Search trees for hard PFSP instances contain up to 10^{15} explored nodes.



- We first provide a proof-of-concept based on the Backtracking method.

Related work

- Most of existing GPU-accelerated tree search algorithms, e.g. [1, 2, 3], ...
  - focus only on performance;
  - combine low-level programming environments.



- Emergence of GPU supports for PGAS-based environments [4, 5, 6].

- Few works explore GPU-accelerated PGAS-based tree search approaches [7].

About Chapel

- Portable & scalable;
- High-level abstractions for data parallelism, task parallelism, concurrency, and nested parallelism;
- Open-source & collaborative.

GPU-native support:

- CPU parallelism features also target GPUs;
- Vendor-neutral, through the LLVM compiler framework:
  - PTX for Nvidia GPUs;
  - AMDGCN for AMD GPUs.

See more at: `https://chapel-lang.org/`.

GPU programming in Chapel

- Data placement: the `on`-clause

Code sample

```chapel
var A: [1..10] int; // memory allocation on host
on here.gpus[0] {
    var A_d: [1..10] int; // memory allocation on device
    A_d = A; // host-to-device copy
}
```

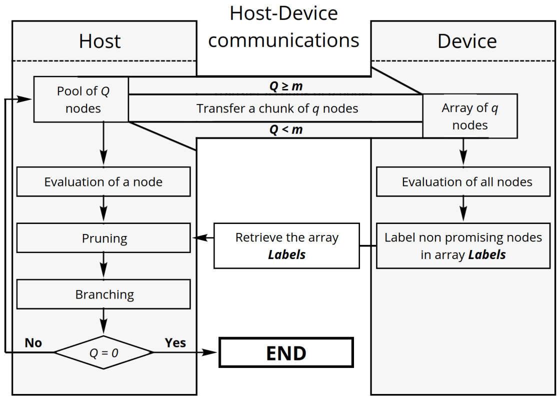
- GPU-eligible loops → GPU kernels

Code sample

```chapel
on here.gpus[0] {
    forall i in 0..#N {
        // do something ...
    }
}
```

Algorithm in single-GPU setting

Parallel evaluation of nodes:



1 GPU device;

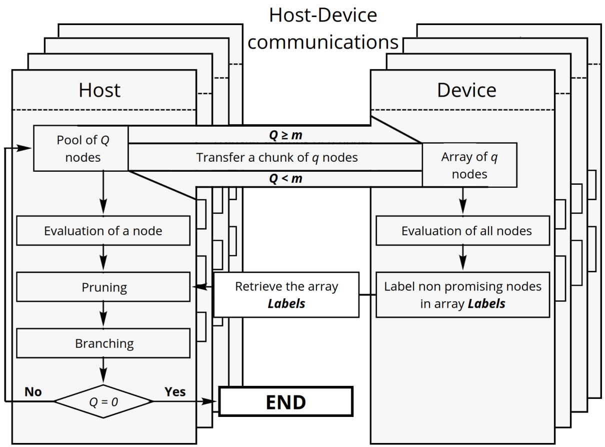
Single work pool;

Parameters:

- m;
- $q = \min(Q, M)$.

Algorithm in multi-GPU setting

Parallel evaluation of nodes + parallel tree exploration:



D GPU devices;

D work pools

$\rightarrow$ load balancing

Static load balancing mechanism

1. Initial search on CPU:

Pseudo-code

```
while (pool.size < D*m) {
    var parent: Node = getNode(pool);
    var children: [] Node = decompose(parent);
    insertNodes(children, pool); // bulk insertion
}
```

2. Static workload distribution:

Pseudo-code

```
for i in 0..#pool.size {
    var node = pool[i];
    insertNode(node, multiPool[i%D]); // cyclic distribution
}
```

GPU-accelerated backtracking in Chapel vs. CUDA-based

Pseudo-code in Chapel

```
var pool = new Pool();
var root = new Node();
insertNode(root, pool);

// do partial search ...

coforall deviceID in 0..#D with (ref pool) {
    var pool_d = new Pool();
    // do static load balancing ...

    while (pool_d.size != 0) {
        if (pool_d.size < m) {
            var parent = getNode(pool_d);
            var children = decompose(parent);
            insertNodes(children, pool_d);
        } else {
            var parents: [] Node = getNodes(pool_d);
            var labels: [] int;
            on here.gpus[deviceID] {
                var parents_d = parents;
                var labels_d: [] int;
                evaluateNodes(parents_d, labels_d);
                labels = labels_d;
            }
            generateNodes(parents, labels, pool_d);
        }
    }
}
```

Pseudo-code in C+OpenMP+CUDA

OpenMP

CUDA

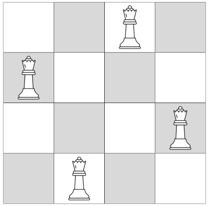
```
Pool pool;
Node root;
insertNode(root, pool);

// do partial search ...

#pragma omp parallel for num_thread(D) shared(pool)
for (int deviceID = 0, deviceID < D; deviceID++) {
    cudaSetDevice(deviceID);
    Pool pool_d;
    // do static load balancing ...

    while (pool_d.size != 0) {
        if (pool_d.size < m) {
            Node parent = getNode(pool_d);
            Node* children = decompose(parent);
            insertNodes(children, pool_d);
        } else {
            Node* parents = malloc();
            parents = getNodes(pool_d);
            Node* parents_d;
            int* labels_d;
            cudaMalloc(parents_d);
            cudaMalloc(labels_d);
            cudaMemcpy(parents_d, parents, HostToDevice);
            evaluateNodes<<<nBlocks, blockSize>>>(parents_d, labels_d);
            cudaMemcpy(labels_d, labels, DeviceToHost);
            cudaFree(parents_d);
            cudaFree(labels_d);
            generateNodes(parents, labels, pool_d);
            free(parents);
            free(labels);
        }
    }
}
```

Experimental protocol

- N-Queens problem, with artificial granularity;

Evaluations are repeated g times

```
foreach node to evaluate do
    for i in 1..g do
        evaluate node;
```



- Instances from N = 14 to 17;
- Comparison Chapel vs. . . .
  - C+CUDA in single-GPU setting;
  - C+OpenMP+CUDA in multi-GPU setting;

  } on a Nvidia-powered system

  - C+HIP in single-GPU setting;
  - C+OpenMP+HIP in multi-GPU setting.

  } on an AMD-powered system

Testbed

Grid'5000 testbed (`https://www.grid5000.fr/`):

- **Nvidia Tesla V100**: Intel Xeon E5-2698 v4 (Broadwell) @ 2.2 GHz, 512 GiB, equipped with 8 Nvidia Tesla V100 SXM2 (32 GiB);
- **AMD Radeon Instinct MI50**: AMD EPYC 7642 (Zen 2) @ 2.3 GHz, 512 GiB, equipped with 8 AMD Radeon Instinct MI50 (32 GiB).

Software	**Version**
Chapel	1.33.0
C compiler	10.4.0
CUDA	11.7.1
HIP/HIP compiler	4.5.0

Normalized execution time in single-GPU setting

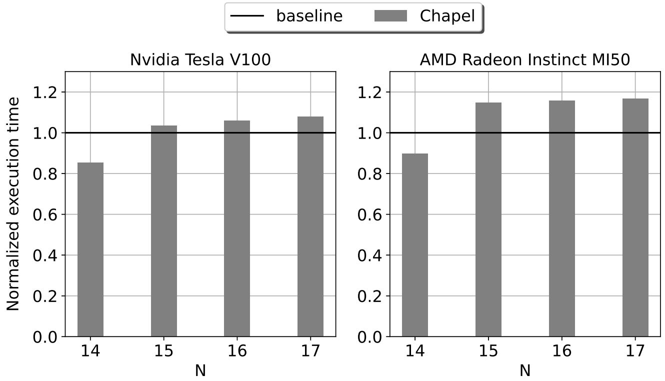
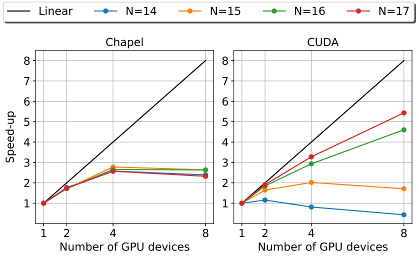


Fig. 1: Normalized execution time: Chapel vs. baselines.

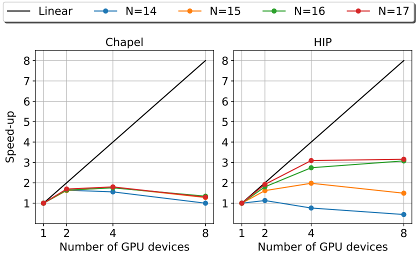
Support for AMD GPU architectures more recent than Nvidia one (March, 2023).

Strong scaling efficiency

Fine-grained experiments: $g = 1$



(a) Chapel vs. CUDA.



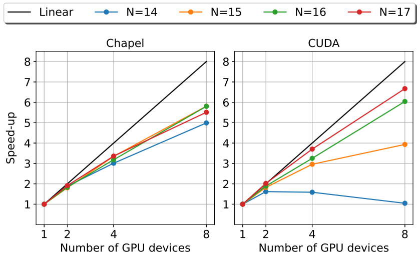
(b) Chapel vs. HIP.

Fig. 2: Strong scaling efficiency: Chapel vs. baselines, setting $g = 1$.

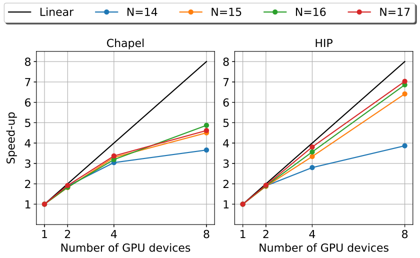
We achieve 45% (resp. 48%) of the CUDA (resp. HIP) baseline strong scaling efficiency solving the largest instance using 8 GPUs.

Strong scaling efficiency

Coarser-grained experiments: $g = 10,000$



(a) Chapel vs. CUDA.



(b) Chapel vs. HIP.

Fig. 3: Strong scaling efficiency: Chapel vs. baselines, setting $g = 10,000$.

We achieve 82% (resp. 66%) of the CUDA (resp. HIP) baseline strong scaling efficiency solving the largest instance using 8 GPUs.

Conclusions

In the context of tree search methods for combinatorial problems:

- Chapel provides high-level features for portable GPU-programming;
- The performance loss is only 8% (resp. 16%) on a Nvidia V100 (resp. AMD MI50) GPU device;
- 82% (resp. 66%) of the CUDA (resp. HIP) baseline strong scaling efficiency is achieved solving the coarser-grained largest instance using 8 GPUs.

Future works

- Extension to combinatorial optimization problems, e.g., B&B applied to PFSP:
  - More irregular workload;
  - External libraries;
  - Communications between B&B workers (e.g., best solution found so far), etc.

- Extension to larger systems (e.g., the LUMI supercomputer, #5 of TOP500).
  - Use of a scalable data structure, e.g., the Chapel's `distBag` data structure.

References

[1] M. E. Lalami and D. El-Baz, “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, 2012.

[2] J. Gmys, M. Mezmaz, N. Melab, and D. Tuyttens, “IVM-based parallel branch-and-bound using hierarchical work stealing on multi-GPU systems,” *Concurrency and Computation: Practice and Experience*, vol. 29, no. 9, p. e4019, 2017.

[3] I. Chakroun, N. Melab, M. Mezmaz, and D. Tuyttens, “Combining multi-core and GPU computing for solving combinatorial optimization problems,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 12, pp. 1563–1577, 2013.

[4] D. Cunningham, R. Bordawekar, and V. Saraswat, “GPU programming in a high level language: compiling X10 to CUDA,” in *Proceedings of the 2011 ACM SIGPLAN X10 Workshop*, pp. 1–10, 2011.

[5] L. Chen, L. Liu, S. Tang, L. Huang, Z. Jing, S. Xu, D. Zhang, and B. Shou, “Unified Parallel C for GPU Clusters: Language Extensions and Compiler Implementation,” in *Languages and Compilers for Parallel Computing*, pp. 151–165, 2011.

[6] A. Hayashi, S. R. Paul, and V. Sarkar, “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, 2022.

[7] T. Carneiro, N. Melab, A. Hayashi, and V. Sarkar, “Towards Chapel-based Exascale Tree Search Algorithms: dealing with multiple GPU accelerators,” in *18th International Conference on High Performance Computing & Simulation*, 2021.

Thank you for your attention.

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

Open-source code on GitHub:
`https://github.com/Guillaume-Helbecque/GPU-accelerated-tree-search-Chapel`

Krishnasamy is supported by FNR CORE (ref. U-AGR-7213-00-V), while the other authors are 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.