Investigating Portability in Chapel for Tree-based Optimization on GPU-powered Clusters

EUROPAR 2024 – The 30th International European Conference on Parallel and Distributed Computing
26-30 August 2024 – Madrid, Spain

Tiago Carneiro[1], Engin Kayraklioglu[2], Guillaume Helbecque[3] and Nouredine Melab[4]

IMEC - Leuven[1], Hewlett Packard Enterprise[2], University of Luxembourg[3], University of Lille[3 4]

imec

Outline

- Context, motivations and objectives
- Distributed Tree-based Search in Chapel
- The Complexity of Mixing CUDA/HIP with Chapel
- Performance Portability Experiments
- Performance Portability Results
- Programming Effort
- Conclusions and Perspectives

Introduction

Context, motivations and objectives

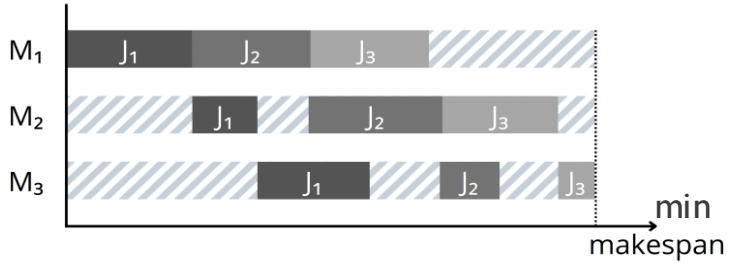
Context and motivations

Combinatorial Optimization Problems

- Optimization problems are increasingly big in many application areas
  - High-dimensionality (#decision variables, #objectives)
  - Time-demanding objectives

$$(O.P) \begin{cases} \min f(x) = (f_1(x), f_2(x), \ldots, f_n(x)) \\ \textbf{s.t.} \quad x \in S \end{cases}$$

- **Motivating Example:** Flow-shop scheduling problem
  - Big instance Ta056 (50 jobs, 20 machines) 10^64 sub-problems
  - 22 years using a single-core processor [M. Mezmaz et al., 2006]
  - **We do need supercomputers to solve big COPs!**



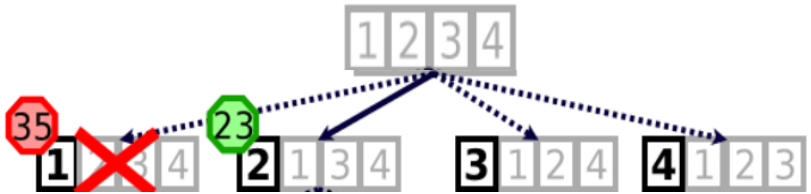
Context and motivations (cont.)

Tree-based Search Algorithms – Branch-and-Bound and Backtracking

- 4 Operators:
  - Branching
  - Bounding
  - Pruning and Selection (DFS, BFS,…)
- Major properties:
  - Huge tree
  - Very dynamic and irregular loads
  - Load balancing is required

Context and motivations (cont.)

Tree-based Search Algorithms – Branch-and-Bound and Backtracking

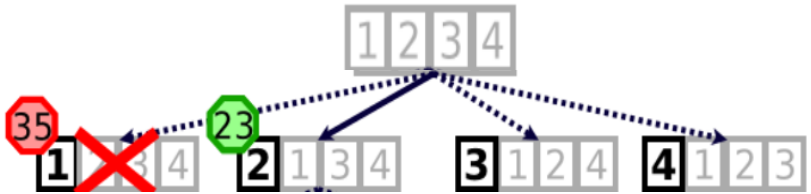


- 4 Operators:
  - Branching
  - Bounding
  - Pruning and Selection (DFS, BFS,…)
- Major properties:
  - Huge tree
  - Very dynamic and irregular loads
  - Load balancing is required

Branching

Context and motivations (cont.)

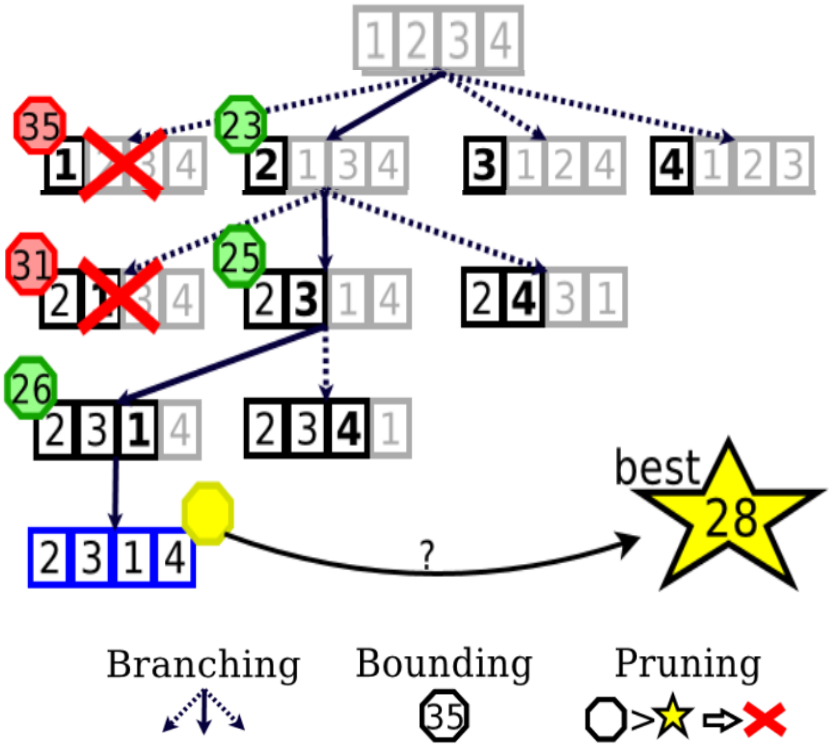
Tree-based Search Algorithms – Branch-and-Bound and Backtracking



- 4 Operators:
  - Branching
  - Bounding
  - Pruning and Selection (DFS, BFS,…)
- Major properties:
  - Huge tree
  - Very dynamic and irregular loads
  - Load balancing is required

Context and motivations (cont.)

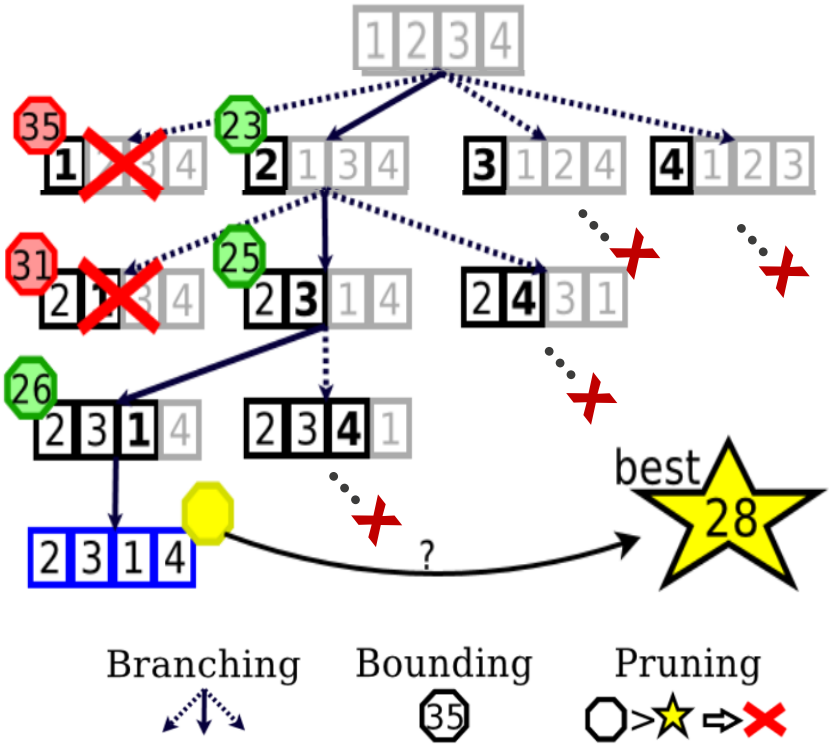
Tree-based Search Algorithms – Branch-and-Bound and Backtracking



- 4 Operators:
  - Branching
  - Bounding
  - Pruning and Selection (DFS, BFS,…)
- Major properties:
  - Huge tree
  - Very dynamic and irregular loads
  - Load balancing is required

Context and motivations (cont.)

Tree-based Search Algorithms – Branch-and-Bound and Backtracking



- 4 Operators:
  - Branching
  - Bounding
  - Pruning and Selection (DFS, BFS,…)
- **Major properties:**
  - Huge tree
  - **Very dynamic and irregular loads**
  - **Load balancing is required**

Objectives

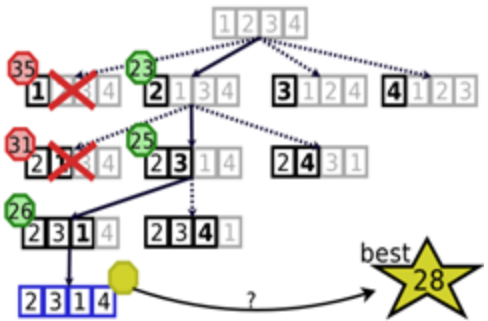
Productivity-aware Ultra-scale Optimization

- Revisit the design and implementation of parallel tree-based search for solving COPs on large-scale supercomputers dealing with …
  - Productivity awareness
  - Parallel efficiency

Objectives

Productivity-aware Ultra-scale Optimization

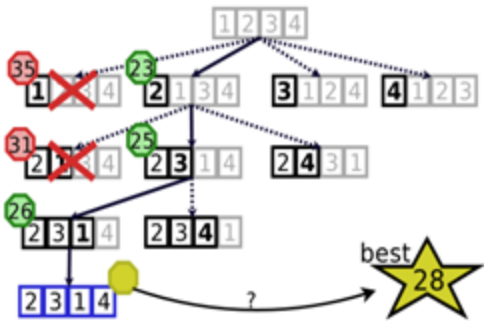
- Revisit the design and implementation of parallel tree-based search for solving COPs on large-scale supercomputers dealing with …
  - Productivity awareness
  - Parallel efficiency

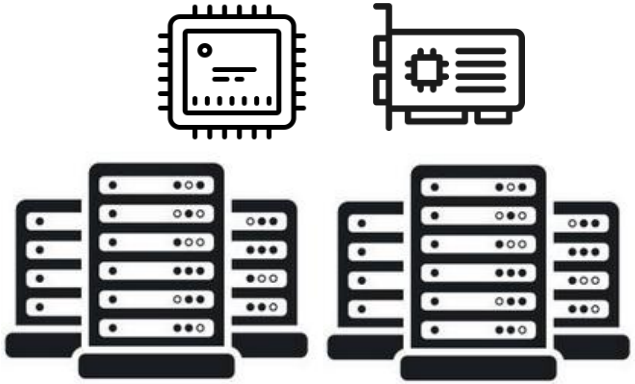


Objectives

Productivity-aware Ultra-scale Optimization

- Revisit the design and implementation of parallel tree-based search for solving COPs on large-scale supercomputers dealing with …
  - Productivity awareness
  - Parallel efficiency
  - Heterogeneity

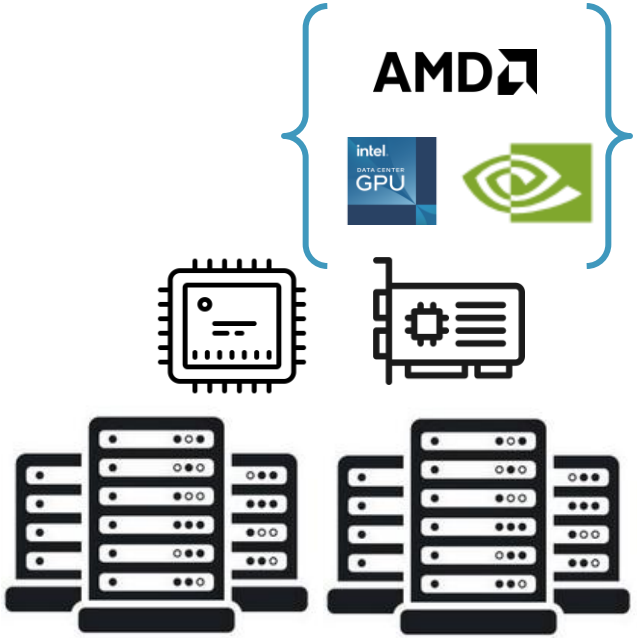




Objectives

Productivity-aware Ultra-scale Optimization

- Revisit the design and implementation of parallel tree-based search for solving COPs on large-scale supercomputers dealing with …
  - Productivity awareness
  - Parallel efficiency
  - Heterogeneity
  - **Code and performance portability**



Objectives

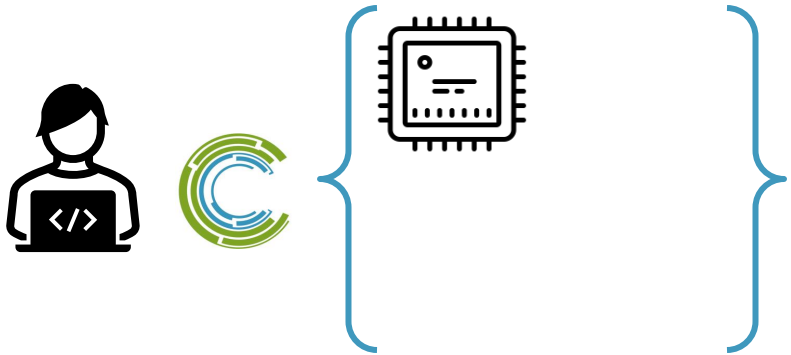
The Chapel Language – Productivity Awareness

- The Chapel language, designed for productive parallel computing at scale, shows to be a candidate for revisiting distributed tree-based search in the PGAS model.

Objectives

The Chapel Language – Productivity Awareness

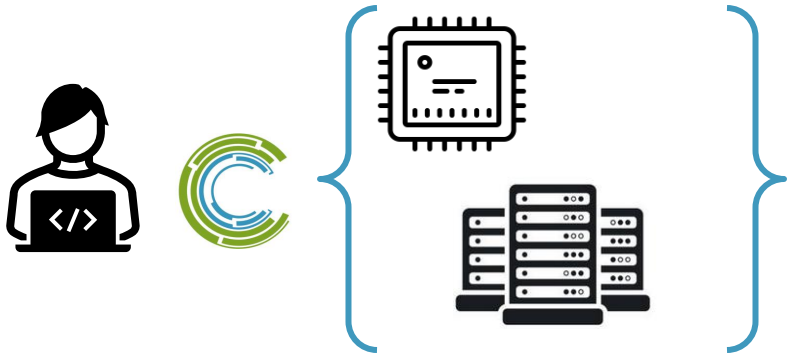
- The Chapel language, designed for productive parallel computing at scale, shows to be a candidate for revisiting distributed tree-based search in the PGAS model.



Objectives

The Chapel Language – Productivity Awareness

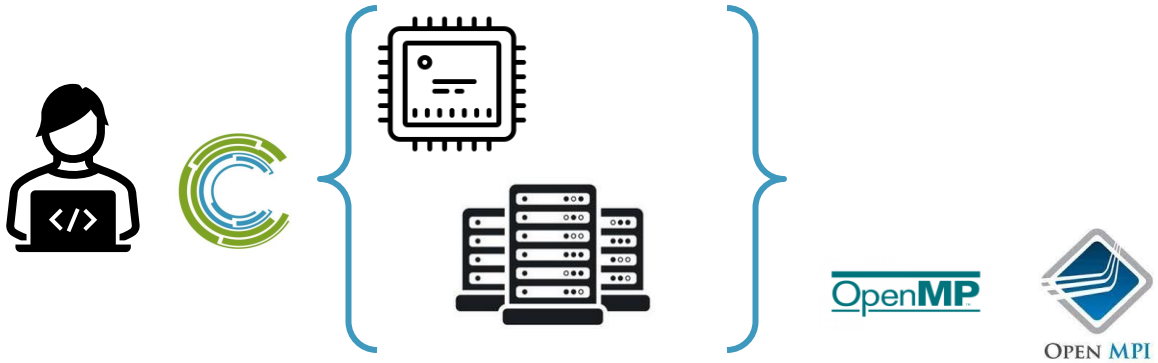
- The Chapel language, designed for productive parallel computing at scale, shows to be a candidate for revisiting distributed tree-based search in the PGAS model.



Objectives

The Chapel Language – Productivity Awareness

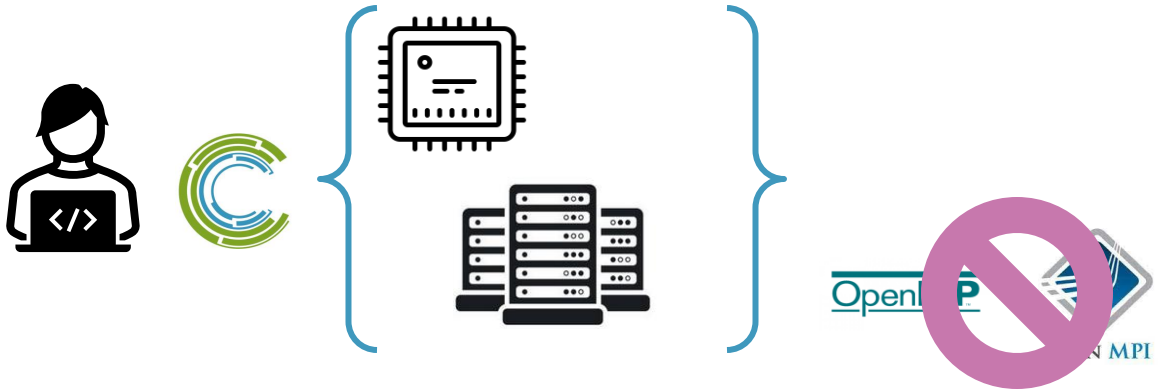
- The Chapel language, designed for productive parallel computing at scale, shows to be a candidate for revisiting distributed tree-based search in the PGAS model.



Objectives

The Chapel Language – Productivity Awareness

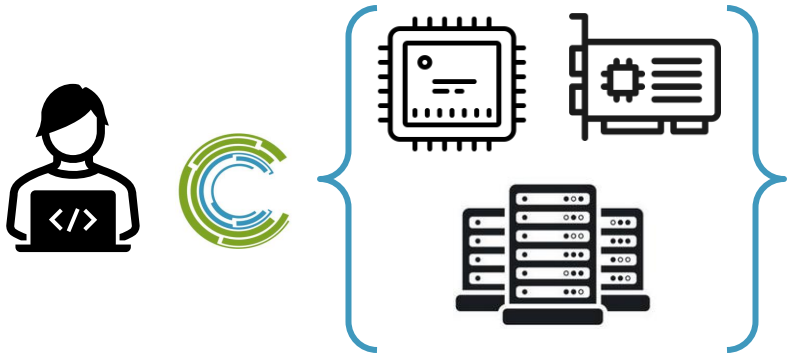
- The Chapel language, designed for productive parallel computing at scale, shows to be a candidate for revisiting distributed tree-based search in the PGAS model.



Objectives

The Chapel Language – Productivity Awareness

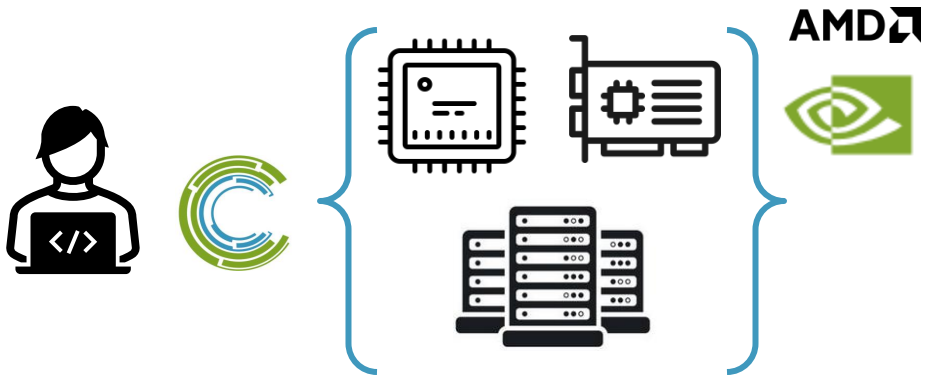
- The Chapel language, designed for productive parallel computing at scale, shows to be a candidate for revisiting distributed tree-based search in the PGAS model.



Objectives

The Chapel Language – Productivity Awareness

- The Chapel language, designed for productive parallel computing at scale, shows to be a candidate for revisiting distributed tree-based search in the PGAS model.



Distributed Tree-based Search in Chapel

Chapel-based Optimization (ChOp)

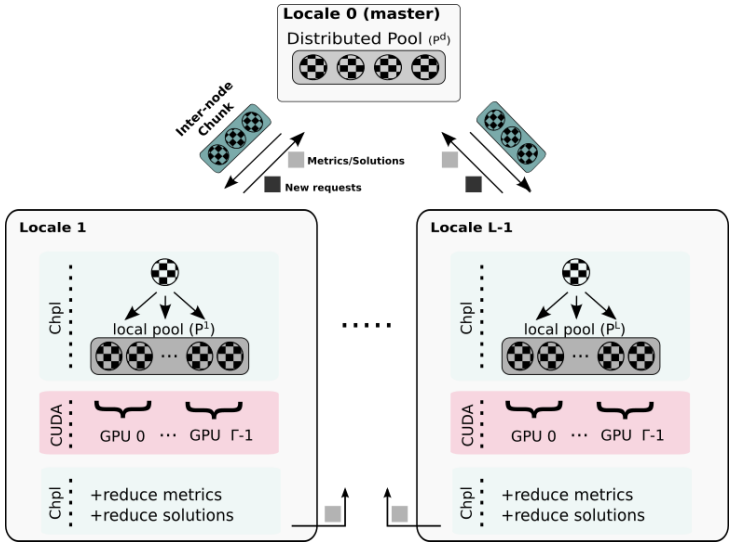
Distributed Tree-based Search in Chapel

- Chapel-based Optimization (ChOp)
  - The `DistributedIters` module, encapsulates a complex distributed PGAS-based master-worker load balancing scheme.

Chapel-based Optimization (ChOp)

Distributed Tree-based Search in Chapel

- Chapel-based Optimization (ChOp)
  - The `DistributedIters` module, encapsulates a complex distributed PGAS-based master-worker load balancing scheme.

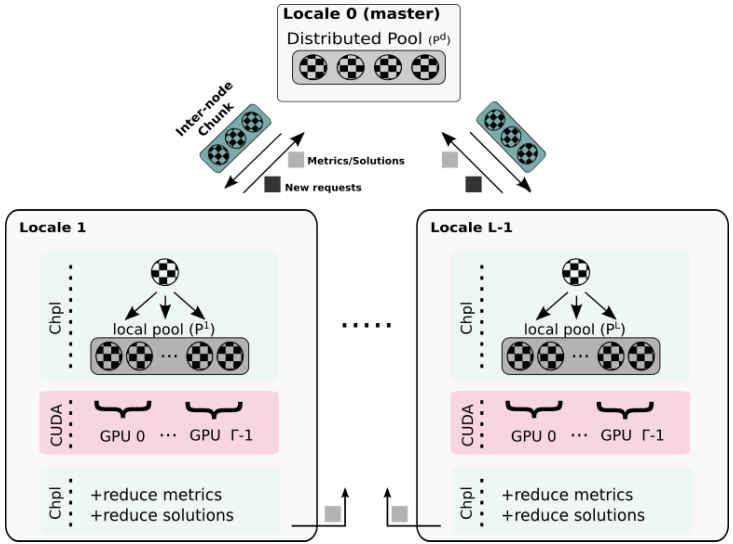


Chapel-based Optimization (ChOp)

Distributed Tree-based Search in Chapel

- Chapel-based Optimization (ChOp)
  - The `DistributedIters` module, encapsulates a complex distributed PGAS-based master-worker load balancing scheme.

Branch-and-Bound/Backtracking
To solve big COP

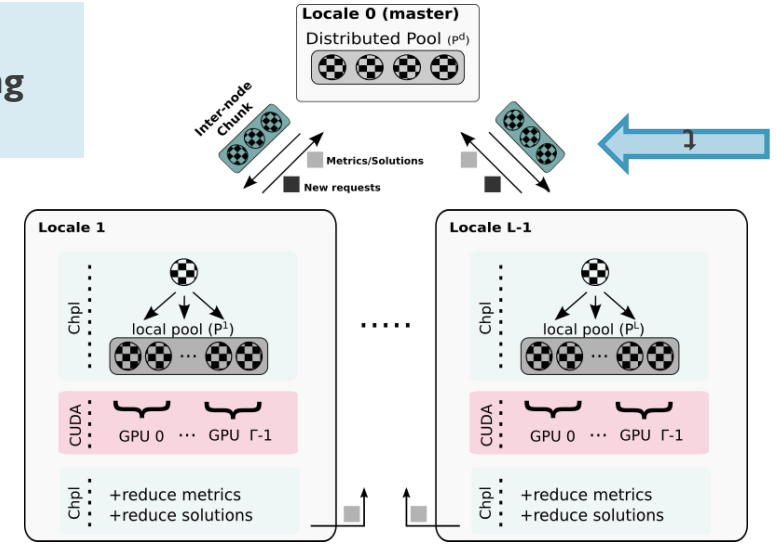


Chapel-based Optimization (ChOp)

Distributed Tree-based Search in Chapel

- Chapel-based Optimization (ChOp)
  - The `DistributedIters` module, encapsulates a complex distributed PGAS-based master-worker load balancing scheme.

Branch-and-Bound/Backtracking
To solve big COP



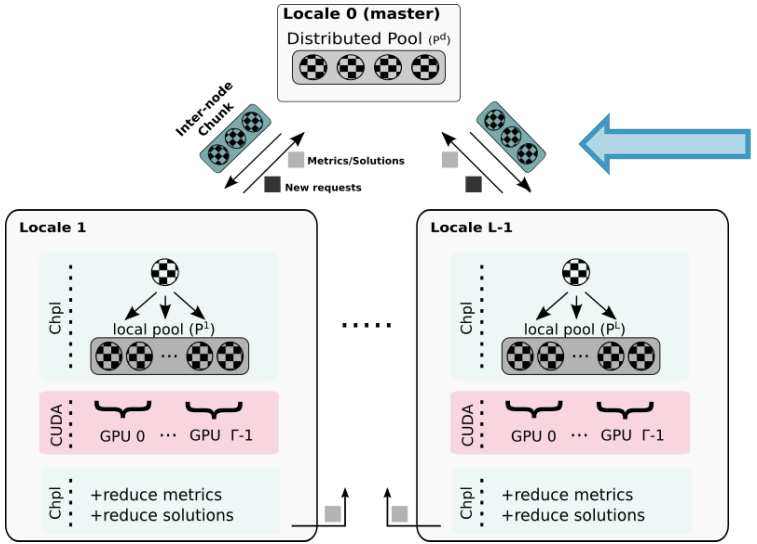
DistributedIters:
distributed load balancing/work distribution
Metrics reduction
Termination criteria

imec

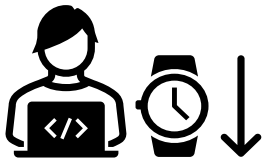
Chapel-based Optimization (ChOp)

Distributed Tree-based Search in Chapel

- Chapel-based Optimization (ChOp)
  - The `DistributedIters` module, encapsulates a complex distributed PGAS-based master-worker load balancing scheme.



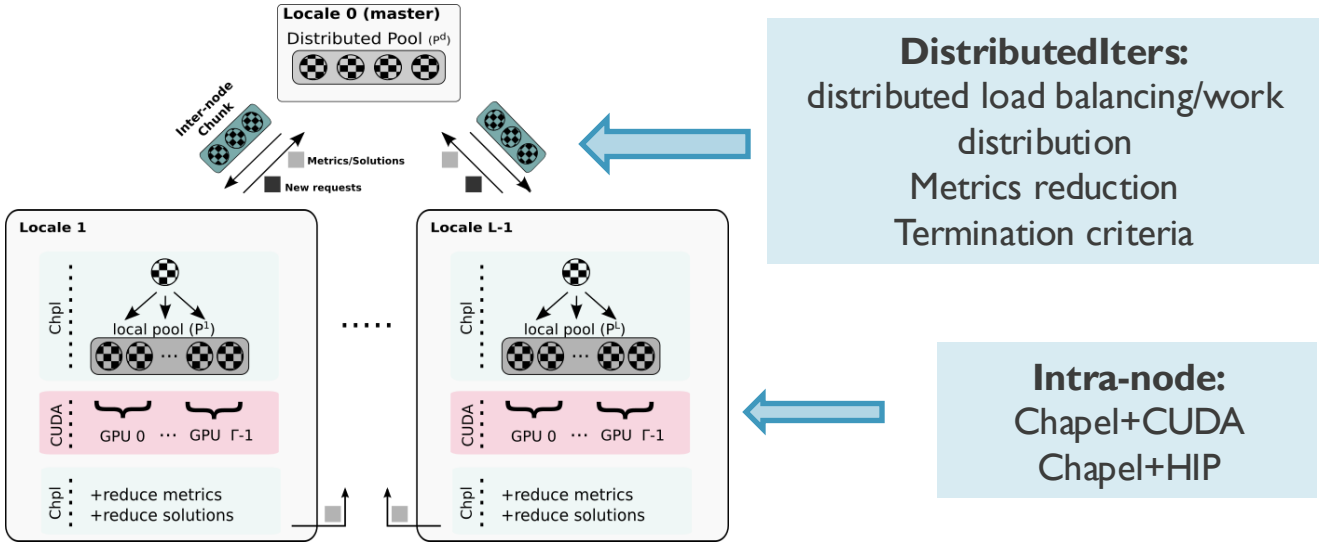
DistributedIters:
Increases productivity in the context of distributed tree-based search!



Chapel-based Optimization (ChOp)

Distributed Tree-based Search in Chapel

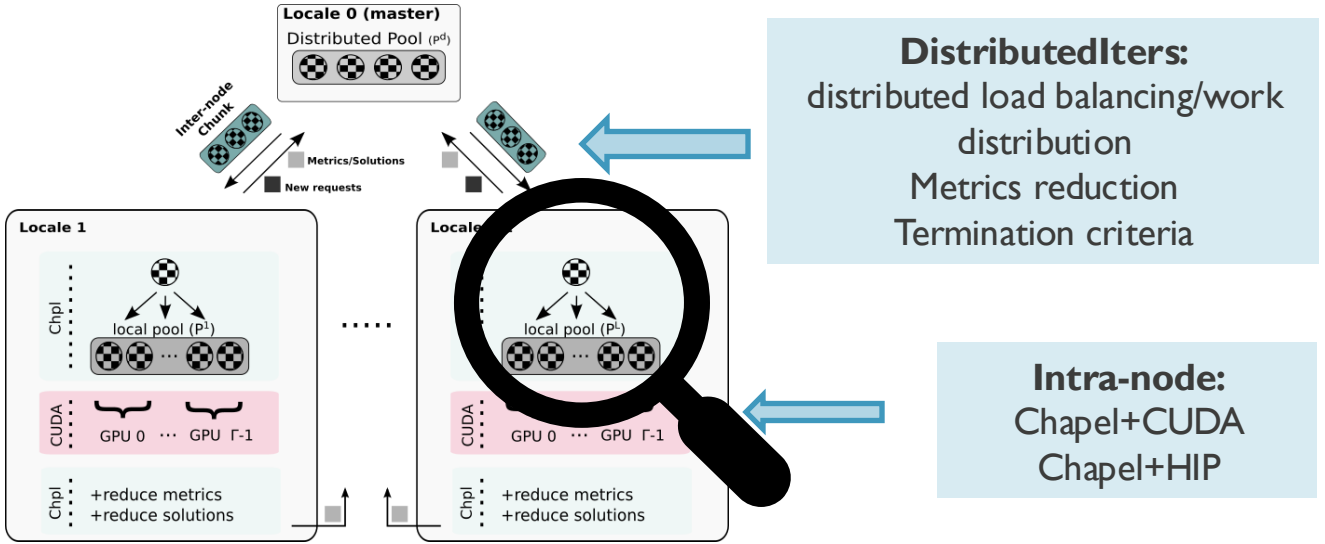
- Chapel-based Optimization (ChOp)
  - The `DistributedIters` module, encapsulates a complex distributed PGAS-based master-worker load balancing scheme.



Chapel-based Optimization (ChOp)

Distributed Tree-based Search in Chapel

- Chapel-based Optimization (ChOp)
  - The `DistributedIters` module, encapsulates a complex distributed PGAS-based master-worker load balancing scheme.

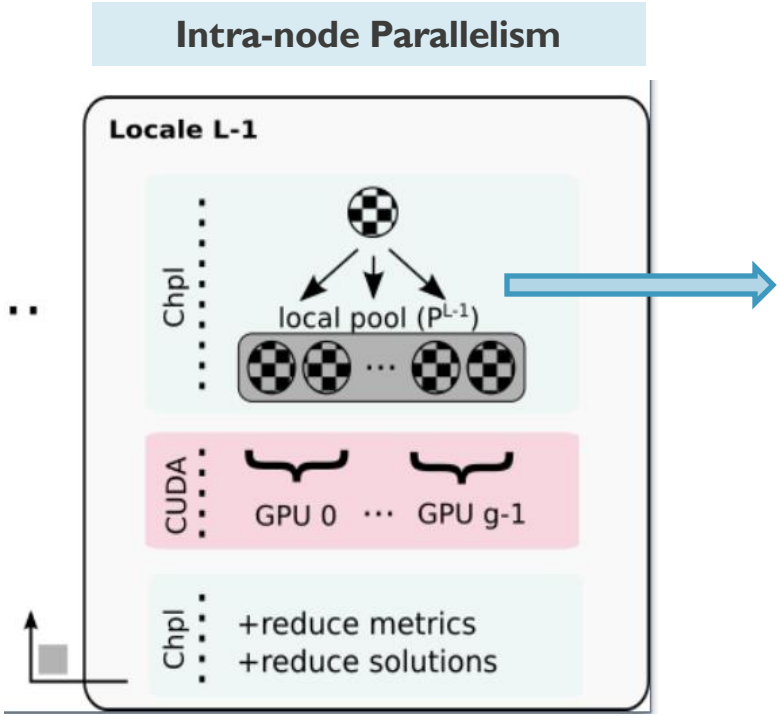


imec

The Complexity of Mixing CUDA/HIP with Chapel

The Complexity of Using C-Interoperability

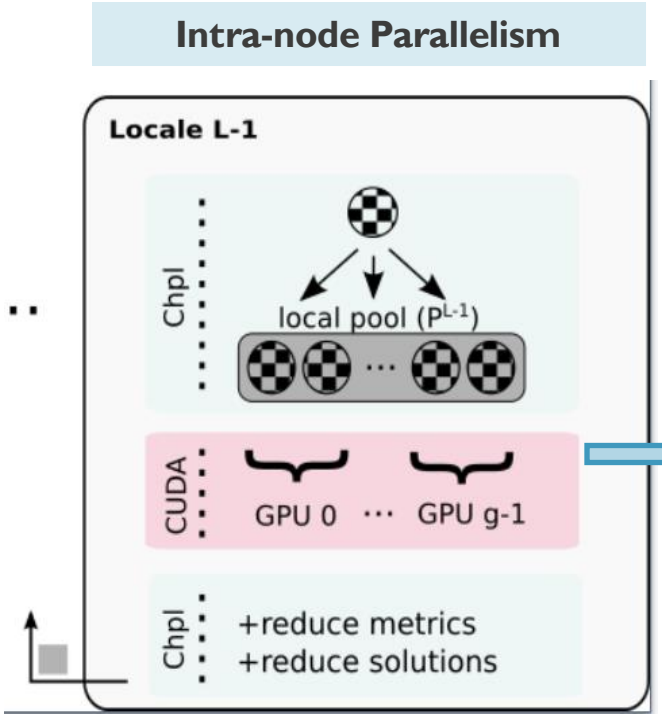
Mix of Programming Languages and Programming Models



From a subproblem in the chunk:
Generates load for the GPUs
Improves locality

The Complexity of Using C-Interoperability

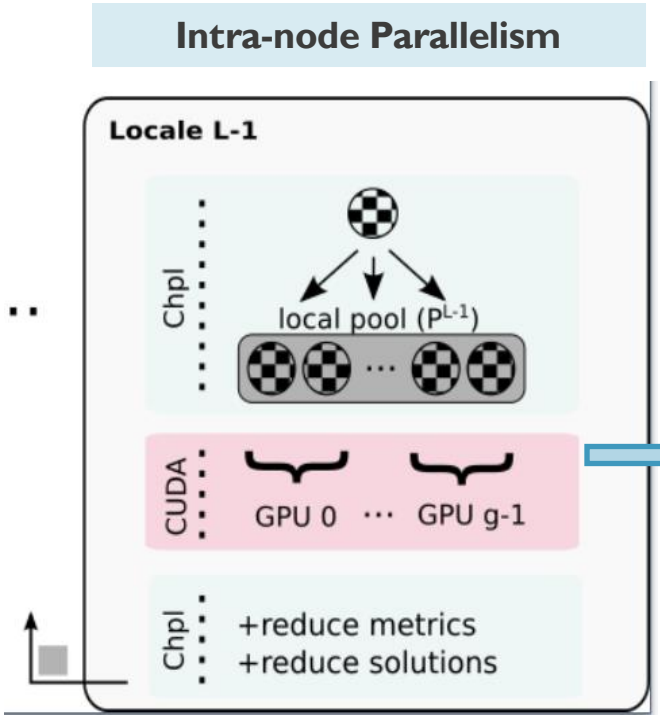
Mix of Programming Languages and Programming Models



Calling the kernels and load distribution:
Tasks in Chapel – 1 task per GPU
C-Interoperability types and functions
(`c_ulonglong`, `CPtrTo(pool+chunk)`)

The Complexity of Using C-Interoperability

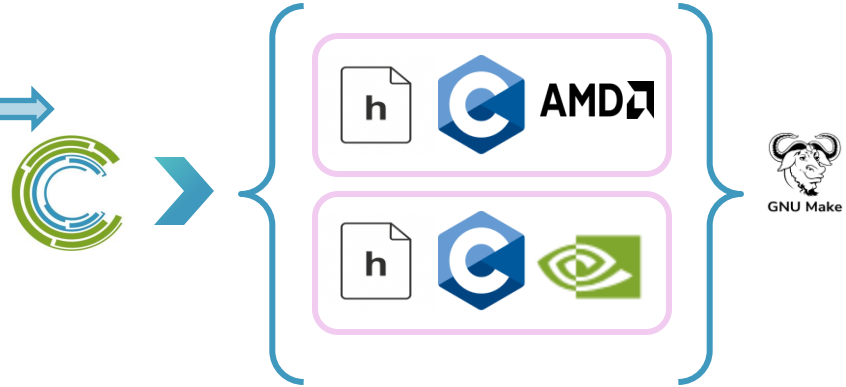
Redundant Kernel Implementations



Redundant kernels in CUDA and HIP

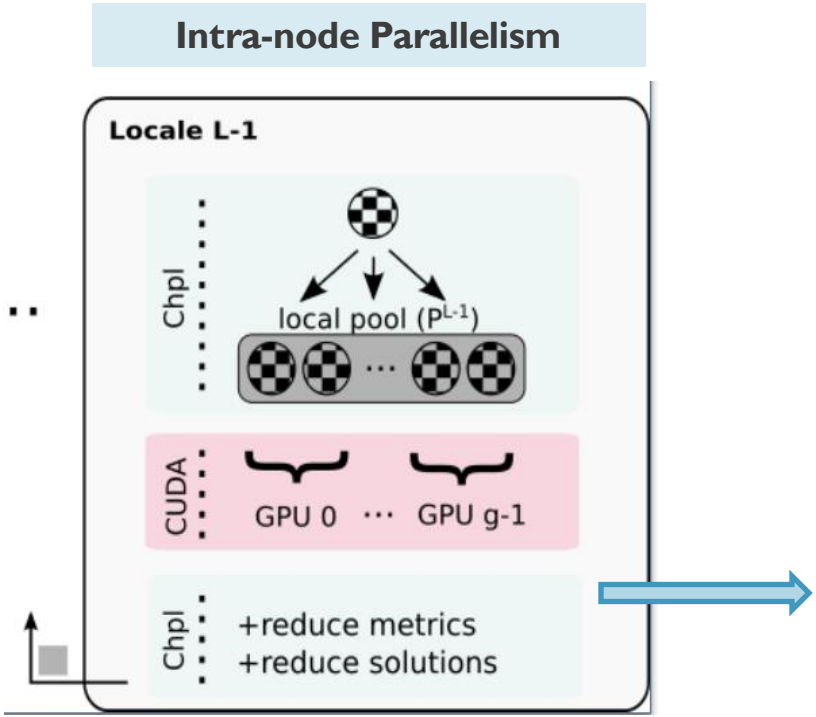
Wrapper function in C

Different Makefile rules



The Complexity of Using C-Interoperability

Redundant Kernel Implementations

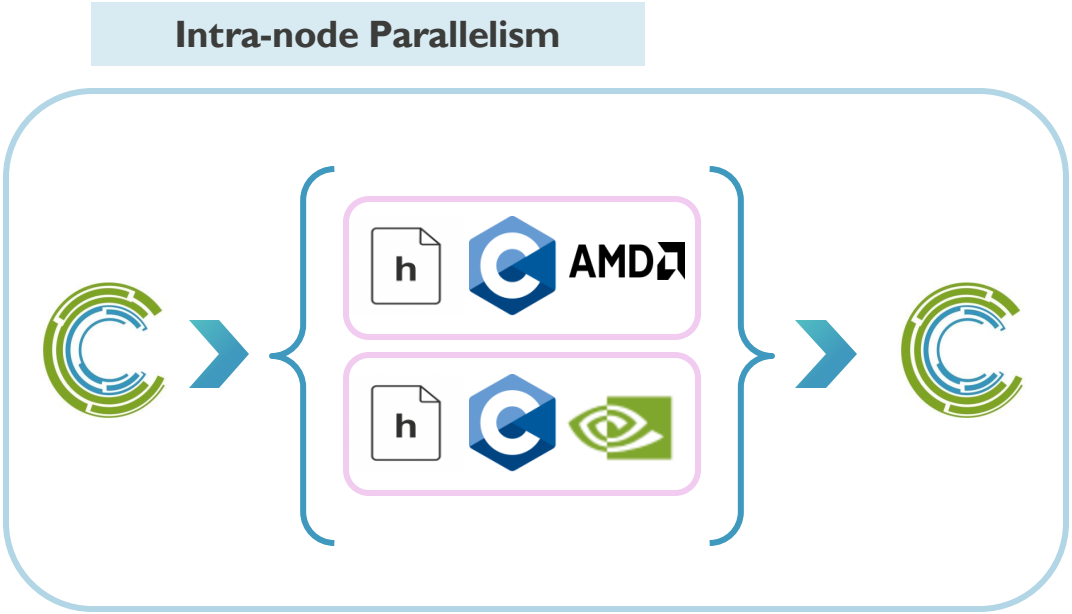


Gets the result back from the C-Wrapper

Uses C-Interoperability types

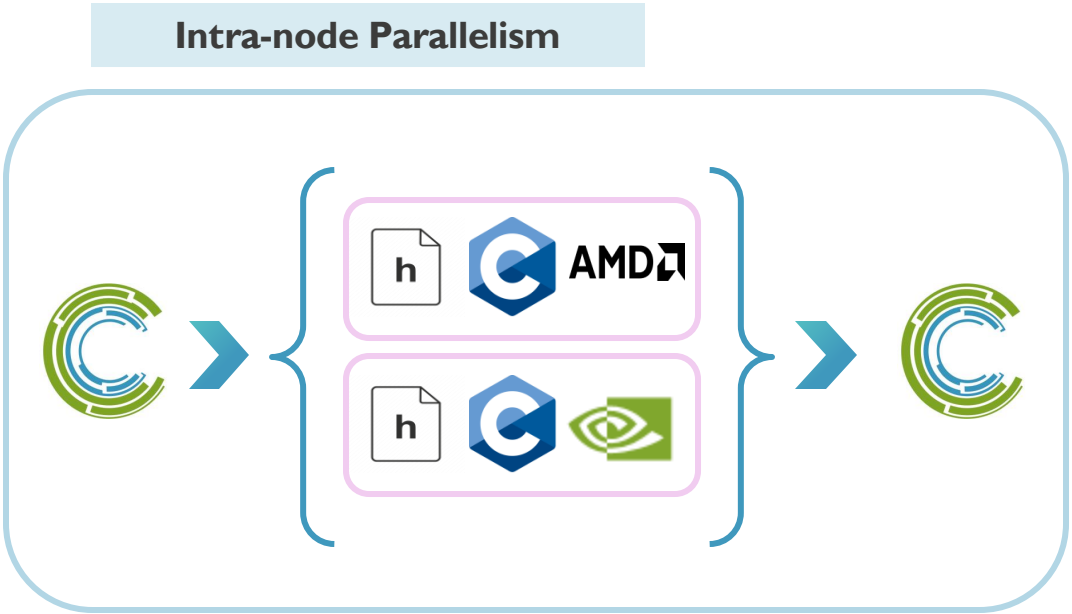
The Complexity of Using C-Interoperability

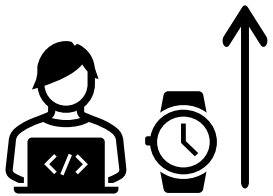
Programming Complexity and Productivity Loss



The Complexity of Using C-Interoperability

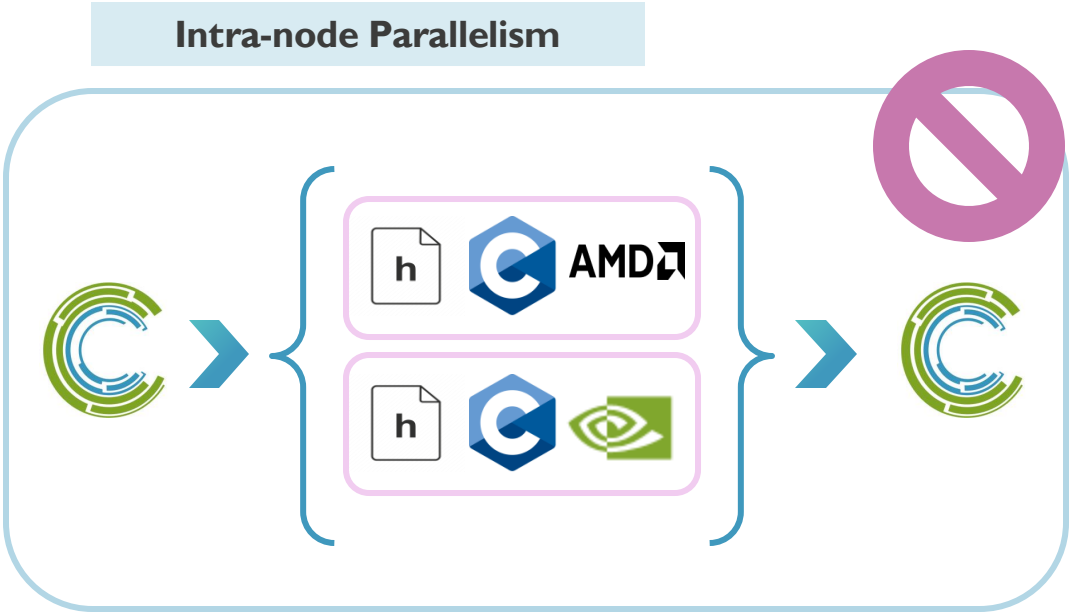
Programming Complexity and Productivity Loss

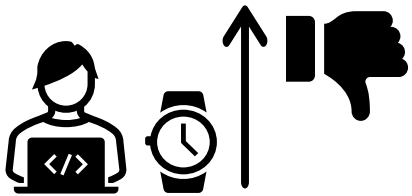




The Complexity of Using C-Interoperability

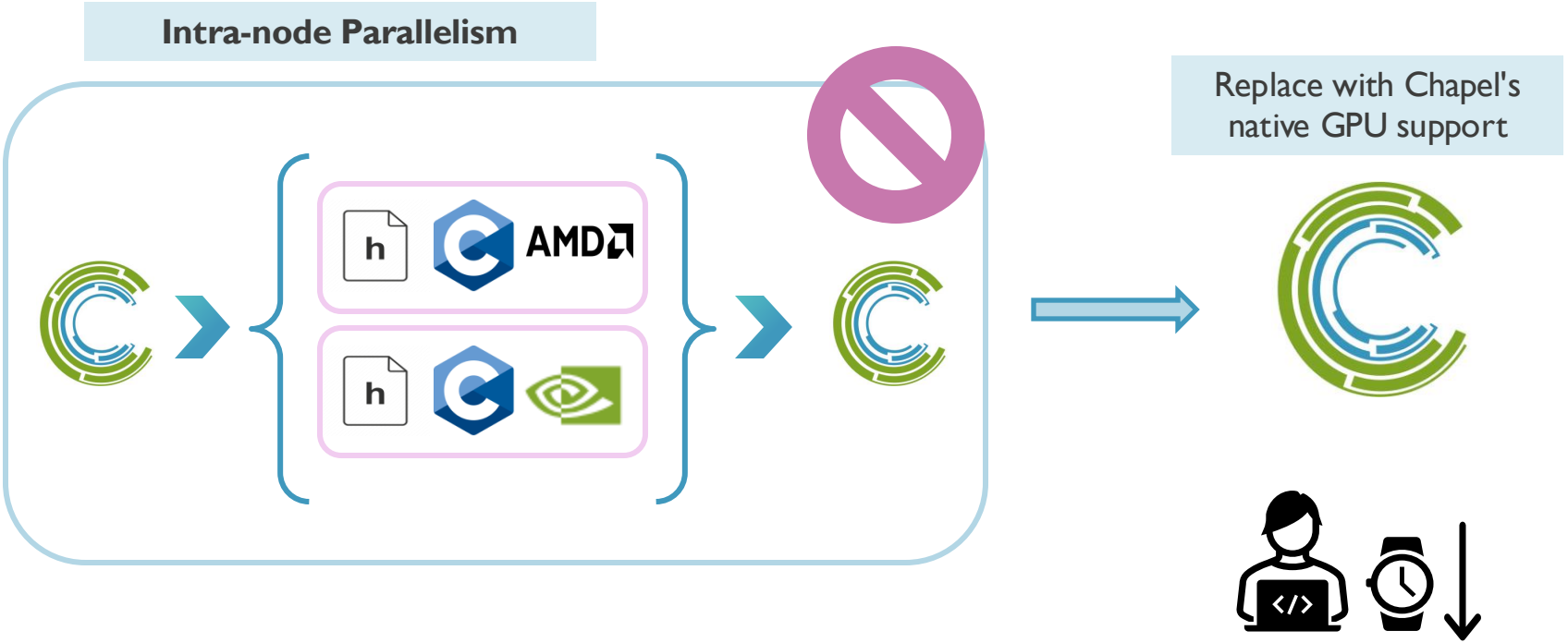
Programming Complexity and Productivity Loss





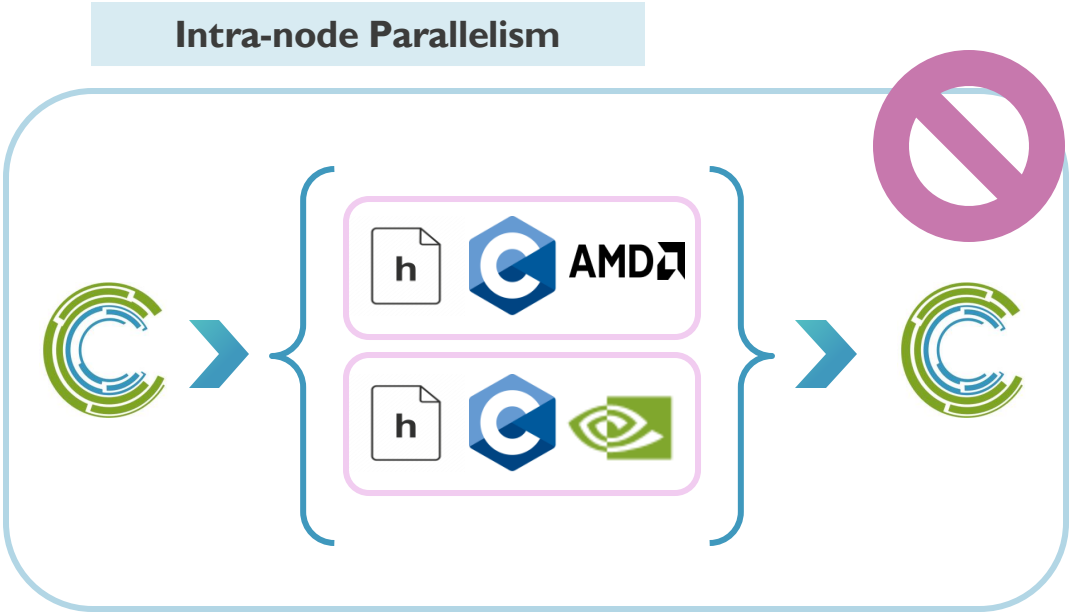
The Complexity of Using C-Interoperability

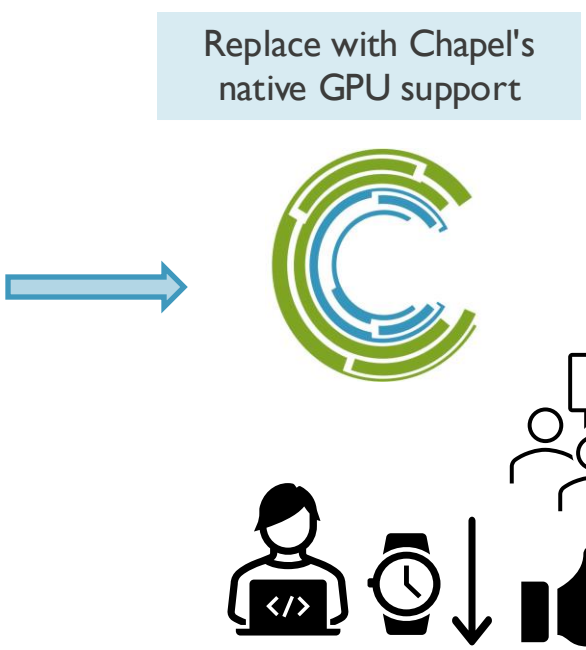
Programming Complexity and Productivity Loss



The Complexity of Using C-Interoperability

Programming Complexity and Productivity Loss





Research Questions

1. Is it possible to achieve both code and performance portability in distributed tree-based search with Chapel's native GPU support?
2. Is it worth in terms of programming effort?

Performance Experiments

Performance Experiments

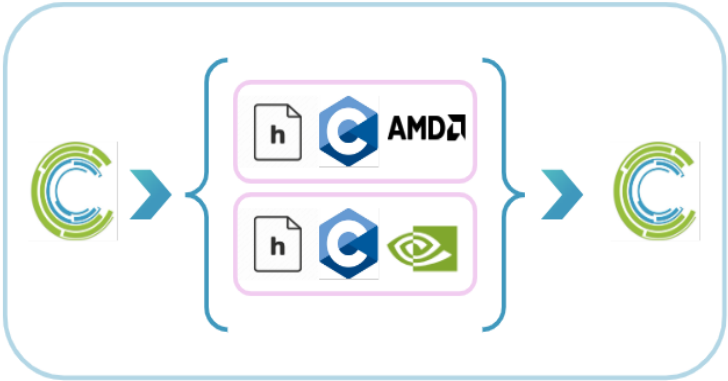
Baselines

- The following applications for enumerating all valid complete solutions of the N-Queens problem are considered: (Backtracking)
  - **Single-node Multi-GPU *baselines*:** HIP+OpenMP **and** CUDA+OpenMP
  - **Single-node Multi-GPU:** Chapel-GPU
  - **Distributed Hybrid:** Chapel+CUDA **and** Chapel+HIP
  - **Distributed Chapel-GPU:** Chape's native GPU support

Performance Experiments

Baselines

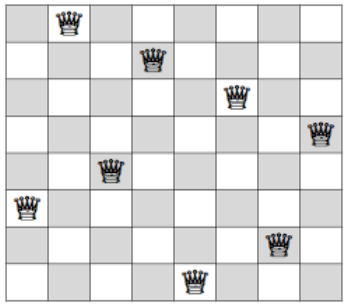
- The following applications for enumerating all valid complete solutions of the N-Queens problem are considered: (Backtracking)
  - **Single-node Multi-GPU *baselines*:** HIP+OpenMP **and** CUDA+OpenMP
  - **Single-node Multi-GPU:** Chapel-GPU
  - **Distributed Hybrid:** Chapel+CUDA **and** Chapel+HIP
  - **Distributed Chapel-GPU:** Chape's native GPU support

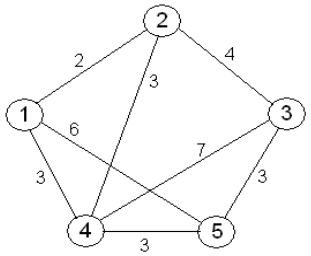


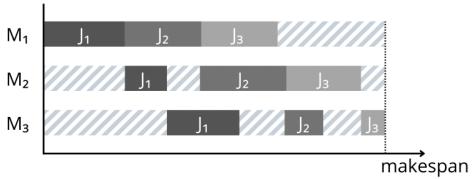
Performance Experiments

Baselines

- The following applications for enumerating all valid complete solutions of the N-Queens problem are considered: (Backtracking)
  - **Single-node Multi-GPU *baselines*:** HIP+OpenMP **and** CUDA+OpenMP
  - **Single-node Multi-GPU:** Chapel-GPU
  - **Distributed Hybrid:** Chapel+CUDA **and** Chapel+HIP
  - **Distributed Chapel-GPU:** Chape's native GPU support



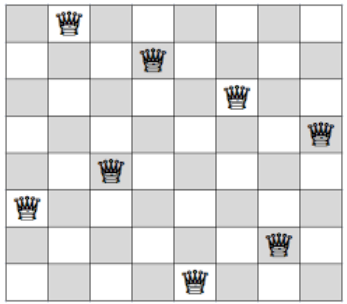




Performance Experiments

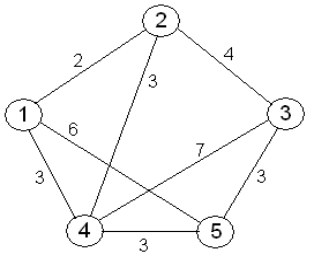
Baselines

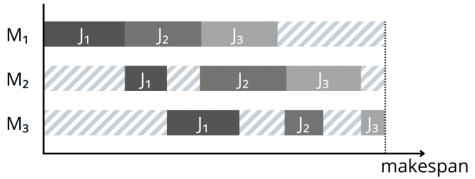
- The following applications for enumerating all valid complete solutions of the N-Queens problem are considered: (Backtracking)
  - **Single-node Multi-GPU *baselines*:** HIP+OpenMP **and** CUDA+OpenMP
  - **Single-node Multi-GPU:** Chapel-GPU
  - **Distributed Hybrid:** Chapel+CUDA **and** Chapel+HIP
  - **Distributed Chapel-GPU:** Chape's native GPU support



N-Queens – proof of concept

C. Optimization Problems – B&B





Performance Experiments

Testbeds and Parameters - Problems

- **N-Queens:**
  - Problem sizes (board) ranging from N=18 **to** N=22
    - **N=18:**
      - $6.6 * 10^{8}$ solutions
      - $2.96 * 10^{10}$ valid placements of queens on the board
    - **N=22**
      - $2.69 * 10^{12}$ solutions
      - $1.43 * 10^{14}$ valid placements of queens on the board
    - From ~10 seconds (N=18) to ~8 hours (N=22) on 8 GPUs.

Performance Experiments

Testbeds and Parameters - Computers

- NVIDIA System:
  - Perlmutter supercomputer (#14 of TOP500)
  - 1 to 128 computer nodes
  - 4 to 512 GPUs (A100 SXM4)
- AMD System:
  - Frontier supercomputer (#1 of TOP500)
  - 1 to 128 computer nodes
  - 8 to 1024 GPUs (MI250As)
  - (only two hours of execution)

Performance Experiments

Testbeds and Parameters - Computers

- **NVIDIA System:**
  - Perlmutter supercomputer **(#14 of TOP500)**
  - 1 to 128 computer nodes
  - 4 to 512 GPUs (A100 SXM4)
- AMD System:
  - Frontier supercomputer (#1 of TOP500)
  - 1 to 128 computer nodes
  - 8 to 1024 GPUs (MI250As)
  - (only two hours of execution)

Performance Experiments

Testbeds and Parameters - Computers

- NVIDIA System:
  - Perlmutter supercomputer (#14 of TOP500)
  - 1 to 128 computer nodes
  - 4 to 512 GPUs (A100 SXM4)
- **AMD System:**
  - Frontier supercomputer (#1 of TOP500)
  - 1 to 128 computer nodes
  - 8 to 1024 GPUs (MI250As)
  - (only two hours of execution)

Performance Experiments

Testbeds and Parameters - Computers

- NVIDIA System:
  - Perlmutter supercomputer (#14 of TOP500)
  - 1 to 128 computer nodes
  - 4 to 512 GPUs (A100 SXM4)
- **AMD System:**
  - Frontier supercomputer **(#1 of TOP500)**
  - 1 to 128 computer nodes
  - 8 to 1024 GPUs (MI250As)
  - (only two hours of execution)

Performance Experiments

Testbeds and Parameters - Computers

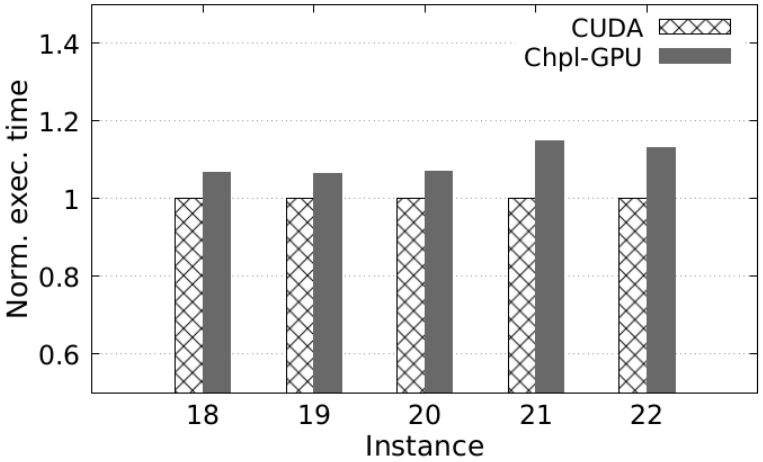
- NVIDIA System:
  - Perlmutter supercomputer (#14 of TOP500)
  - 1 to 128 computer nodes
  - 4 to 512 GPUs (A100 SXM4)
- **AMD System:**
  - Frontier supercomputer (#1 of TOP500)
  - 1 to 128 computer nodes
  - 8 to 1024 GPUs (MI250As)
  - **(only two hours of execution)**

Performance Portability Results

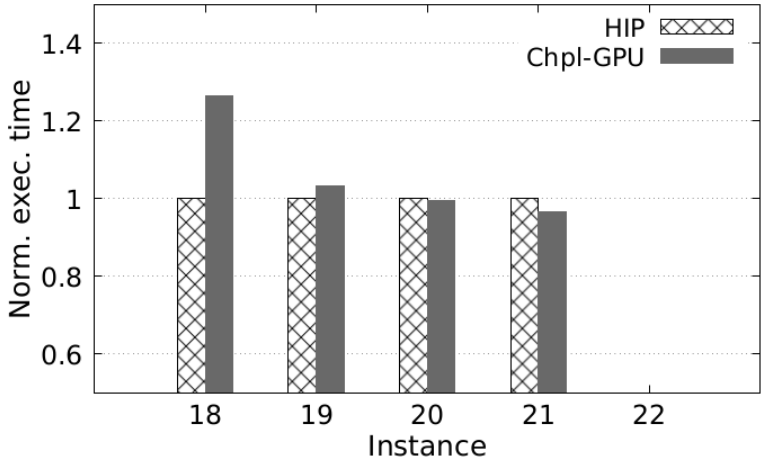
Performance Results

Single-node Application: execution time

- N-Queens of sizes ranging from N=18 to N=22
- 4 GPUS – NVIDIA, 8 GPUs – AMD, Chapel's native GPU support *vs.* Baseline



(a) vs. CUDA - NVIDIA-based System

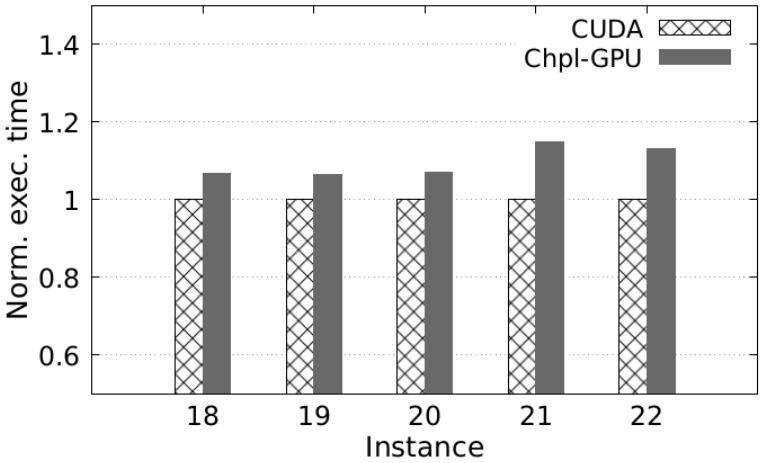


(b) vs. HIP - AMD-based System

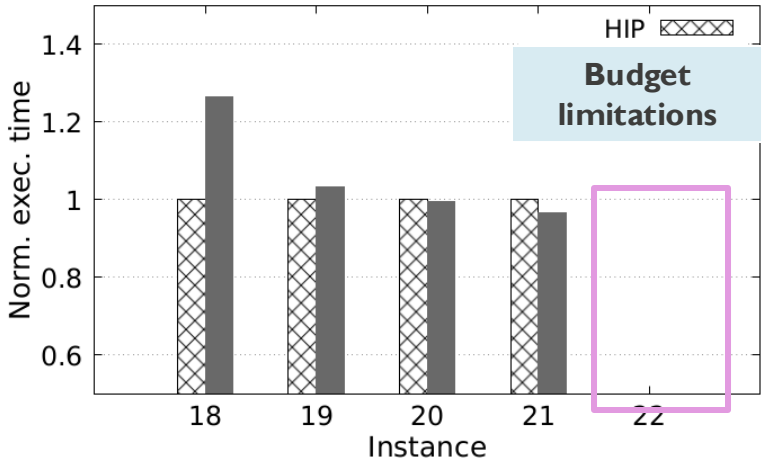
Performance Results

Single-node Application: execution time

- N-Queens of sizes ranging from N=18 to N=22
- 4 GPUS – NVIDIA, 8 GPUs – AMD, Chapel's native GPU support *vs.* Baseline



(a) vs. CUDA - NVIDIA-based System



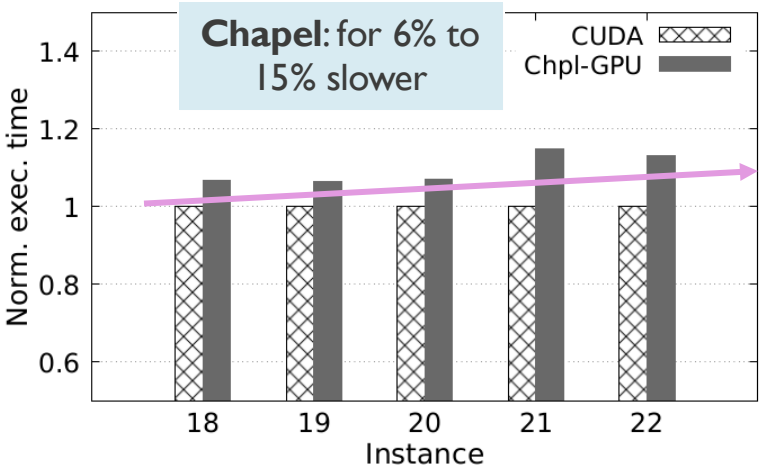
(b) vs. HIP - AMD-based System

Performance Results

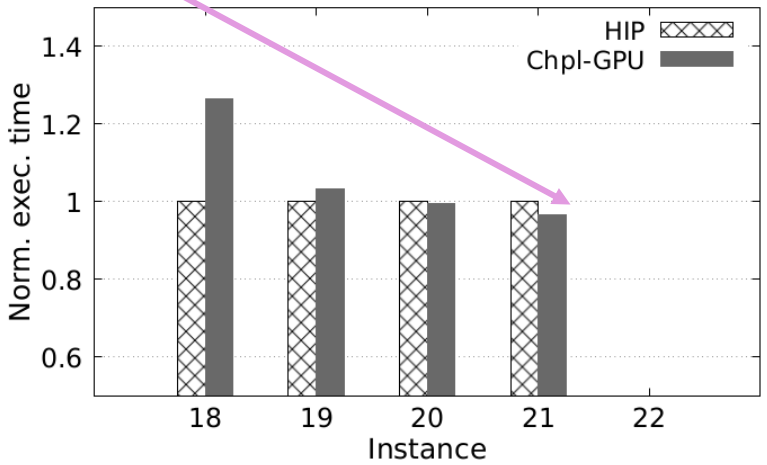
Single-node Application: execution time

- N-Queens of sizes ranging from N=18 to N=22
- 4 GPUS – NVIDIA, 8 GPUs - AMD

The **bigger** the load - the **better** the Chapel performance



(a) vs. CUDA - NVIDIA-based System

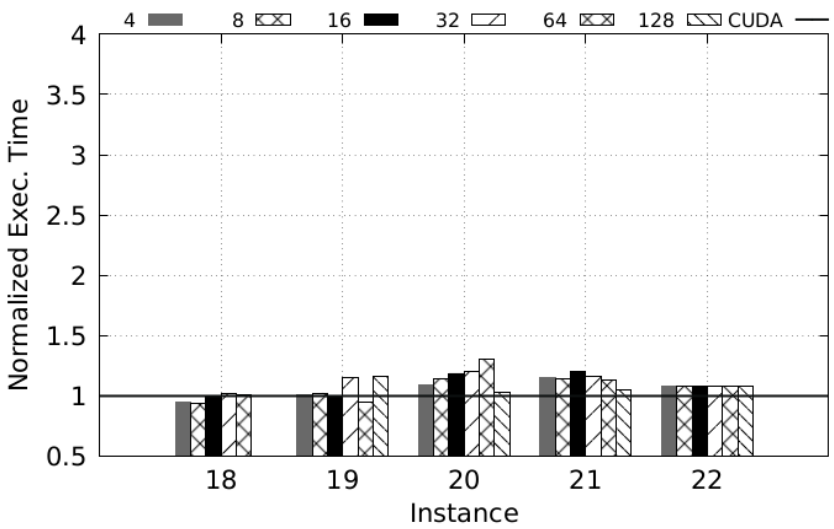


(b) vs. HIP - AMD-based System

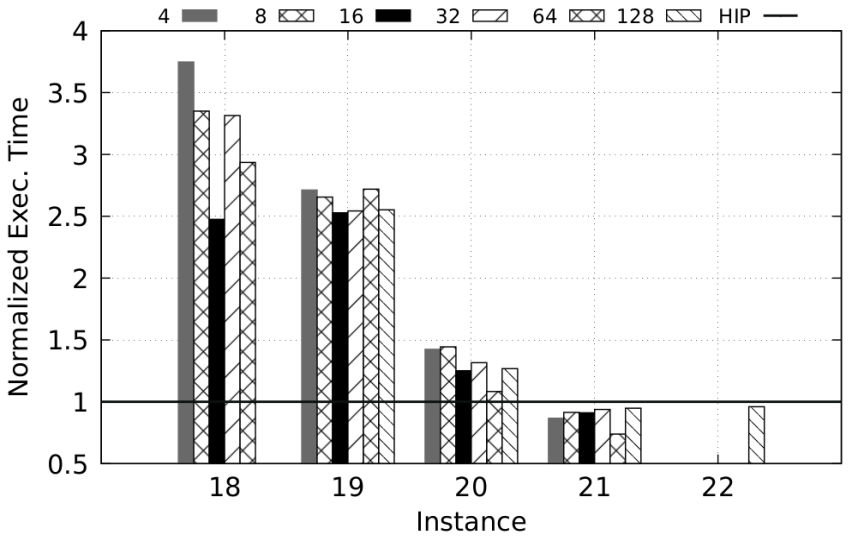
Performance Results

Distributed Application: execution time

- N-Queens of sizes ranging from N=18 to N=22
- 4 to 128 computer nodes, Chapel's native GPU support *vs.* Hybrid



(a) vs. CUDA - NVIDIA-based System

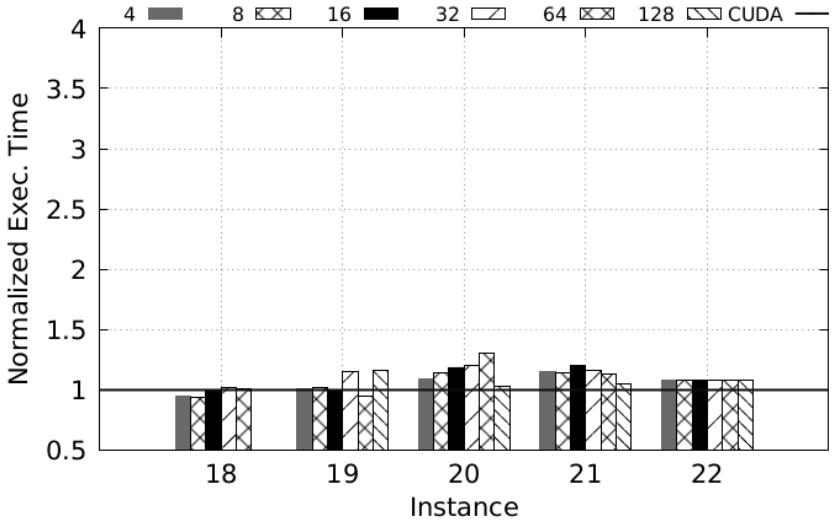


(b) vs. HIP - AMD-based System

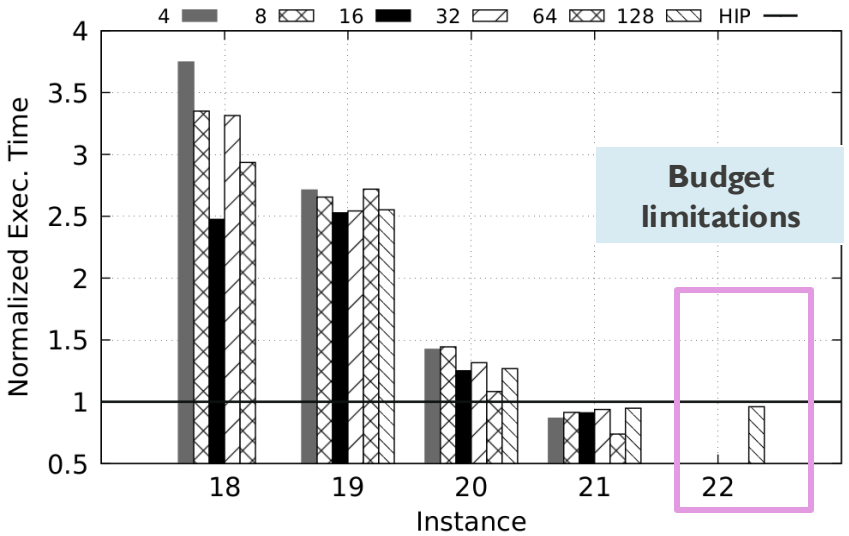
Performance Results

Distributed Application: execution time

- N-Queens of sizes ranging from N=18 to N=22
- 4 to 128 computer nodes, Chapel's native GPU support *vs.* Hybrid



(a) vs. CUDA - NVIDIA-based System

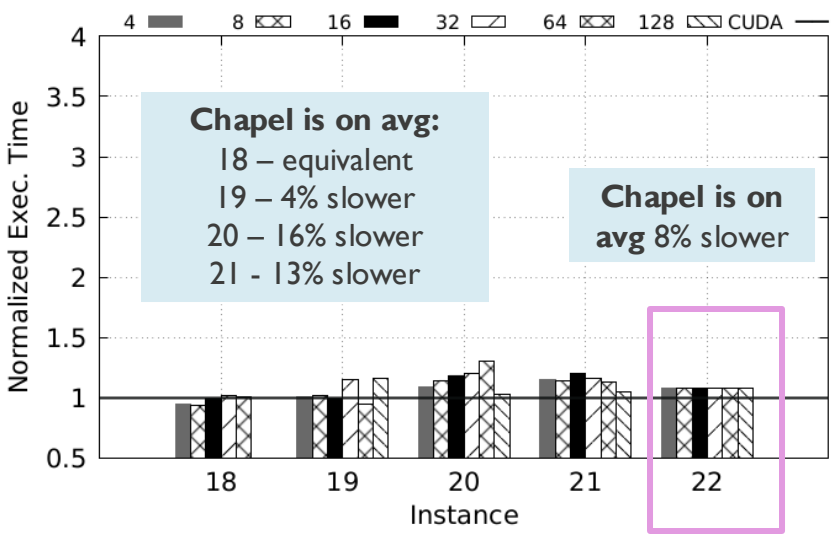


(b) vs. HIP - AMD-based System

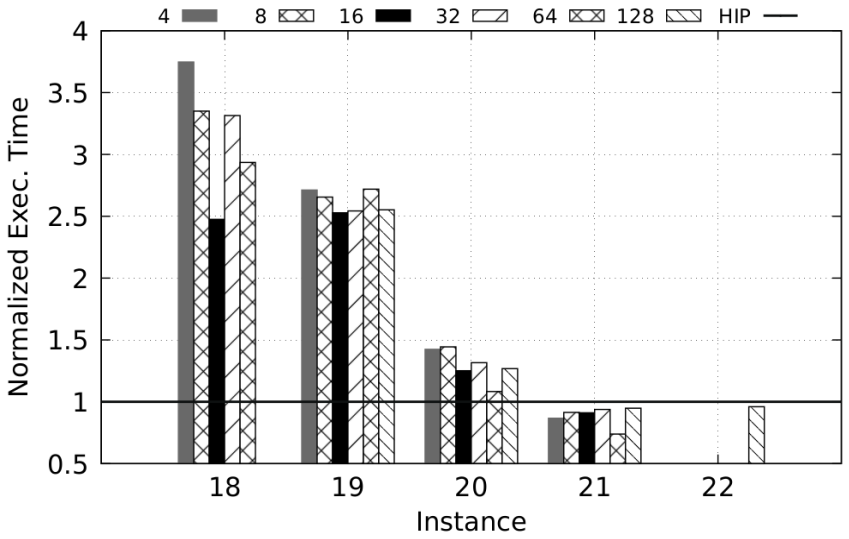
Performance Results

Distributed Application: execution time

- N-Queens of sizes ranging from N=18 to N=22
- 4 to 128 computer nodes, Chapel's native GPU support *vs.* Hybrid



(a) vs. CUDA - NVIDIA-based System



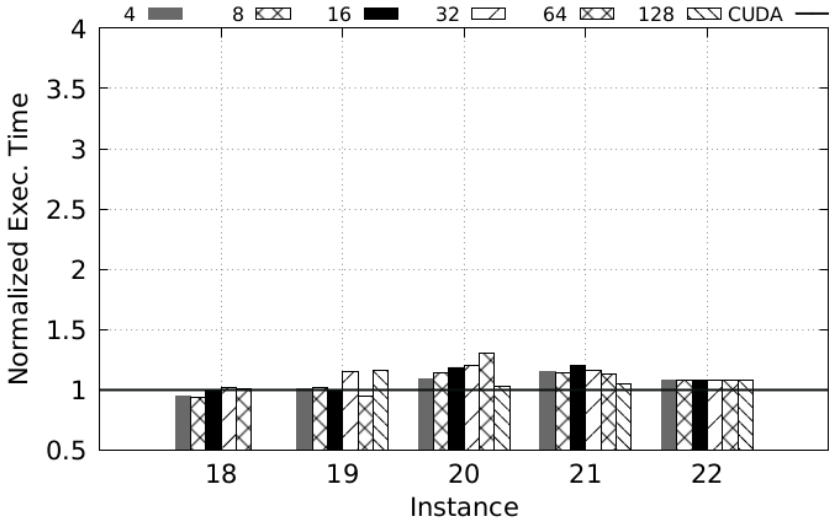
(b) vs. HIP - AMD-based System

imec

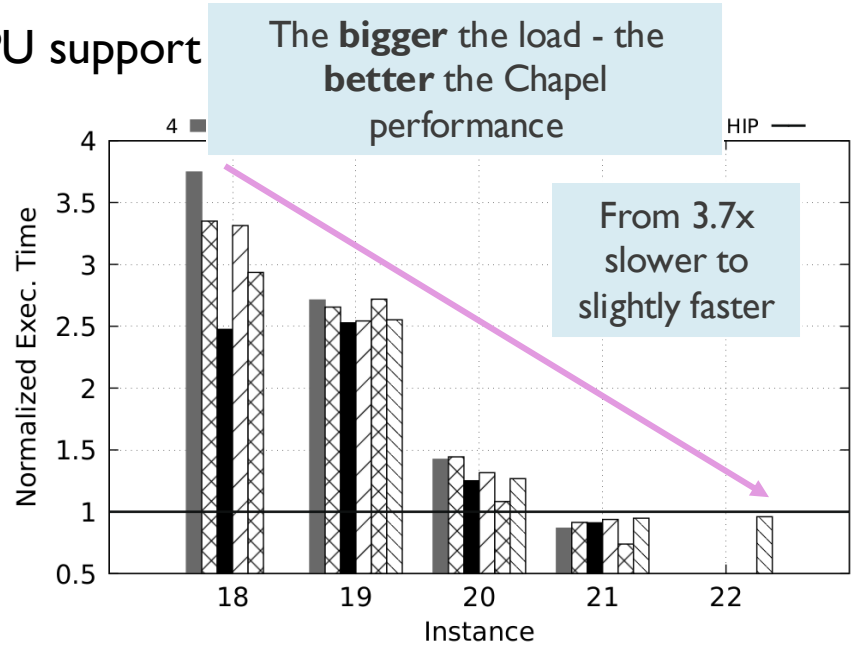
Performance Results

Distributed Application: execution time

- N-Queens of sizes ranging from N=18 to N=22
- 4 to 128 computer nodes, Chapel's native GPU support



(a) vs. CUDA - NVIDIA-based System



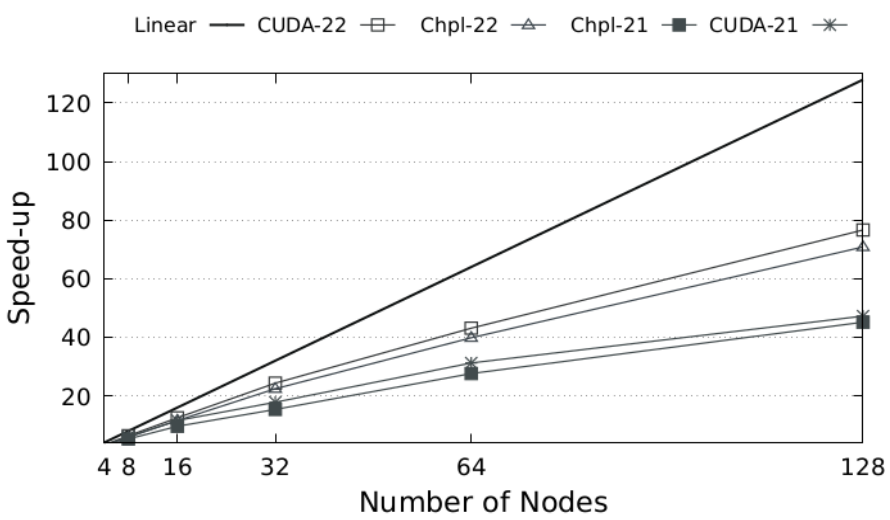
(b) vs. HIP - AMD-based System

imec

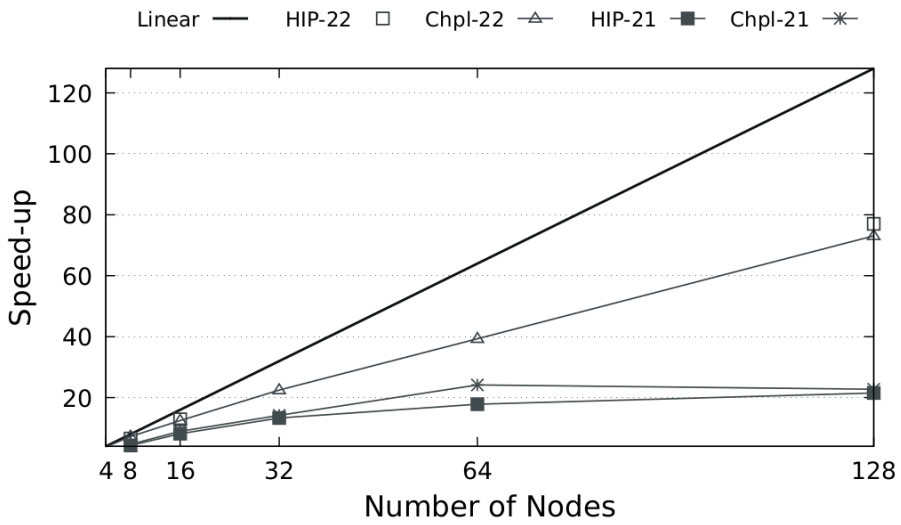
Performance Results

Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline



(a) NVIDIA-based System



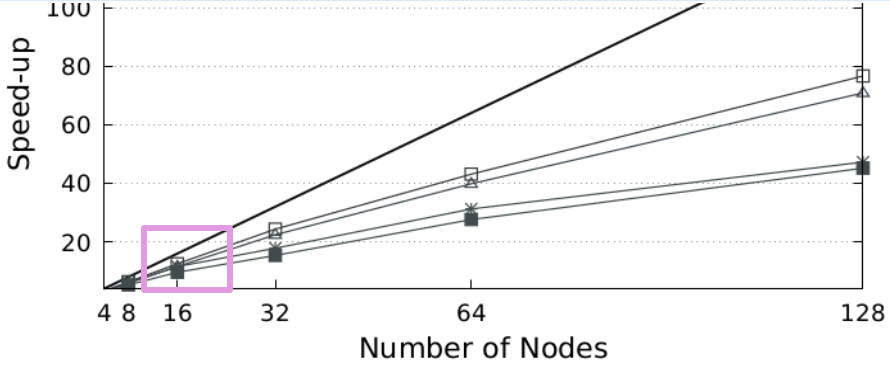
(b) AMD-based system

Performance Results

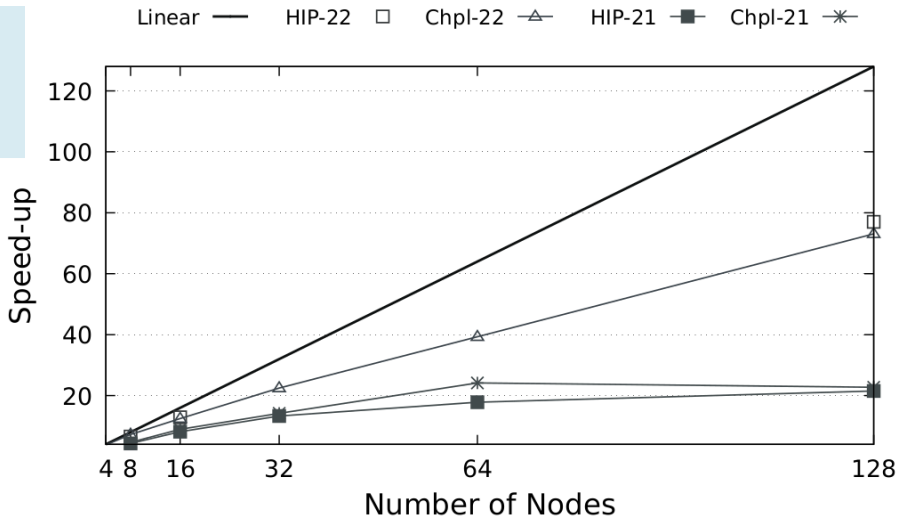
Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline

Load is tiny for 512 GPUs:
Hybrid CUDA 72% *vs.* **Chapel** 60%
of the **linear speedup** on 16 nodes (64 GPUs)



(a) NVIDIA-based System



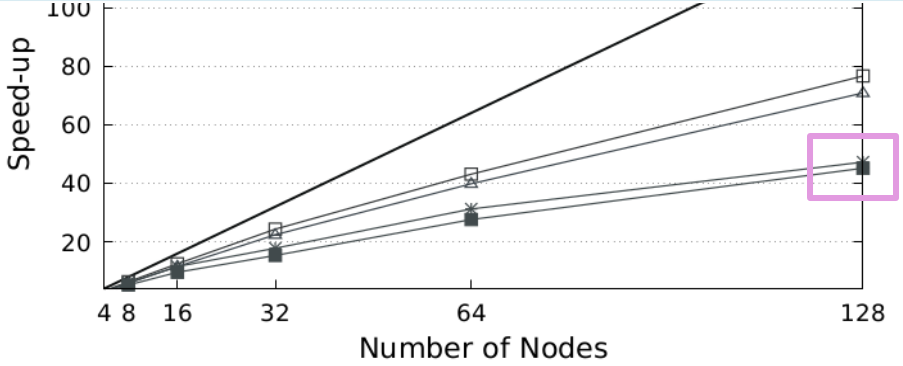
(b) AMD-based system

Performance Results

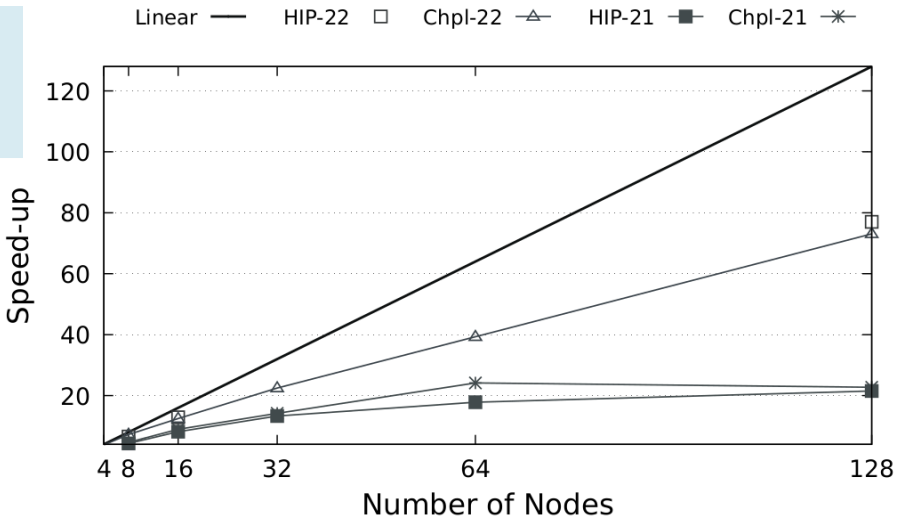
Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline

Load is tiny for 512 GPUs:
Drops to **37%** *vs.* **35%**
of the **linear speedup** on **128** nodes (**512** GPUs)



(a) NVIDIA-based System

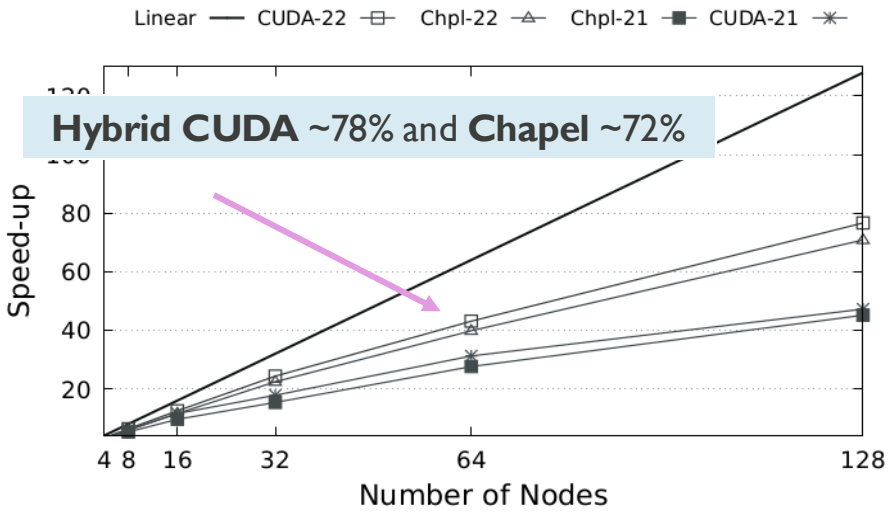


(b) AMD-based system

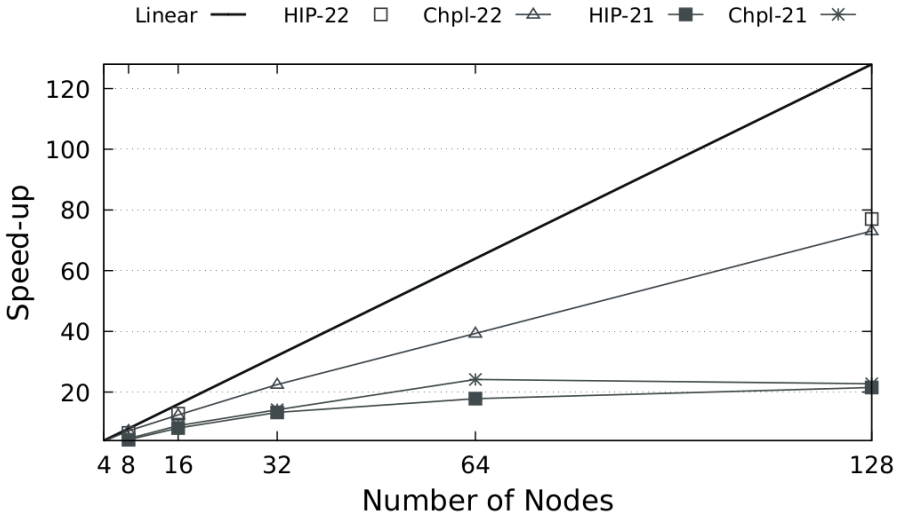
Performance Results

Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline



(a) NVIDIA-based System



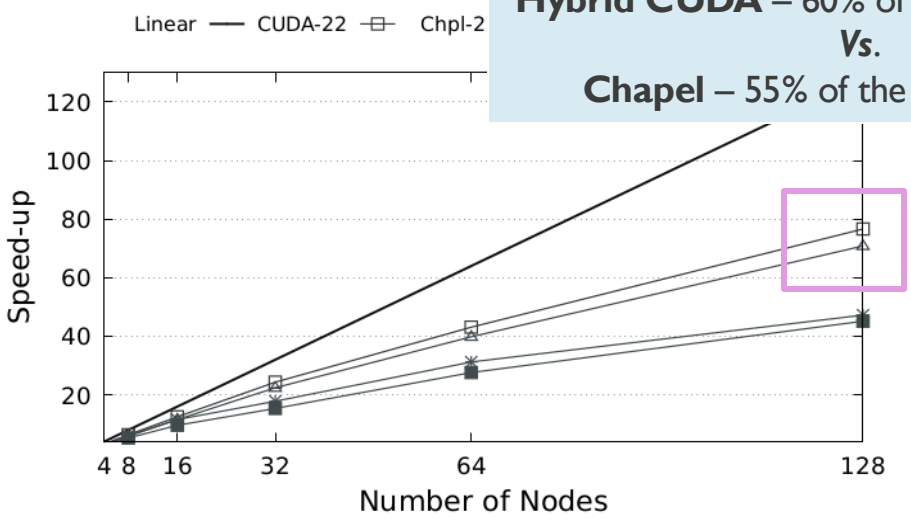
(b) AMD-based system

Performance Results

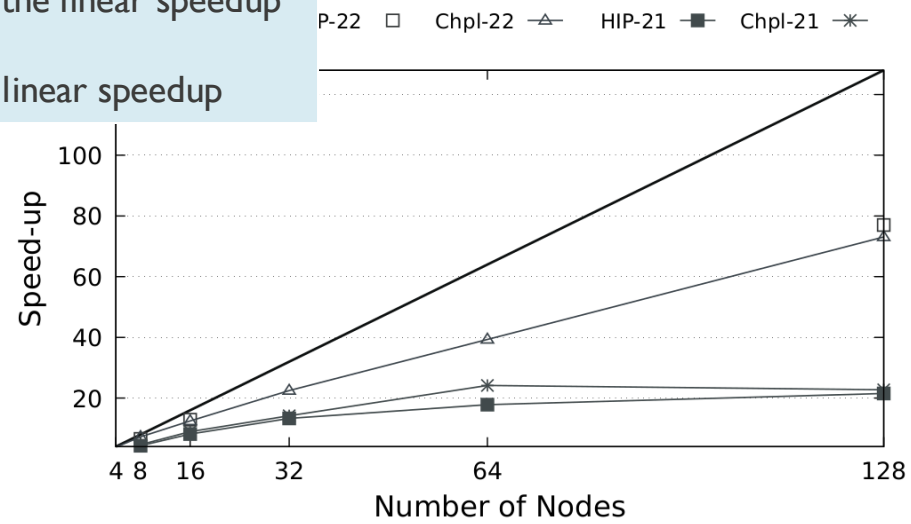
Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline

Hybrid CUDA – 60% of the linear speedup
Vs.
Chapel – 55% of the linear speedup



(a) NVIDIA-based System

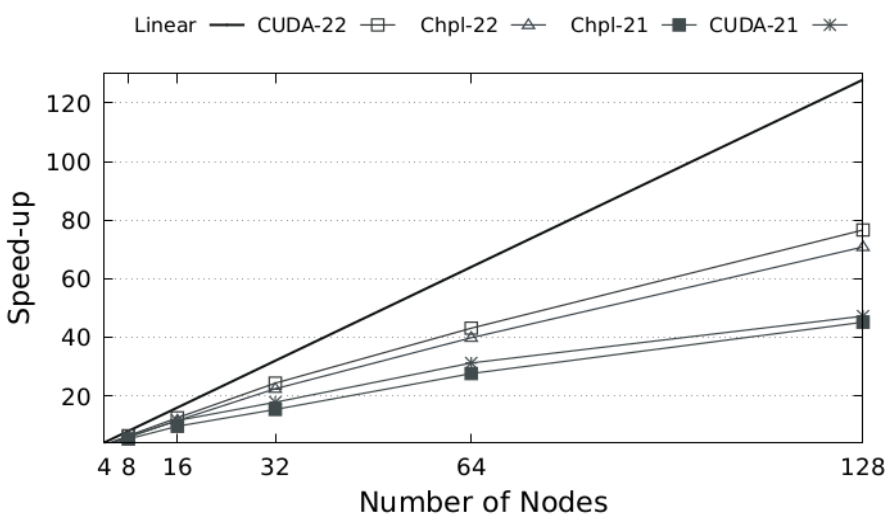


(b) AMD-based system

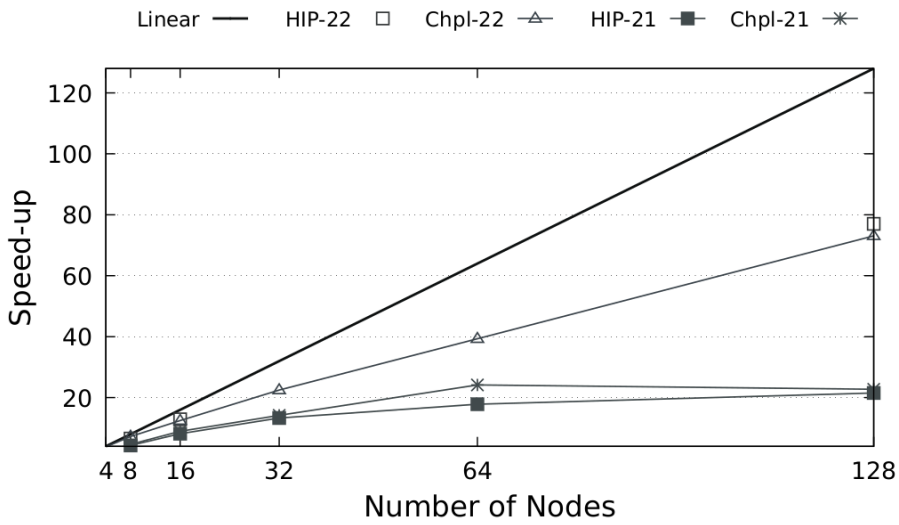
Performance Results

Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline



(a) NVIDIA-based System

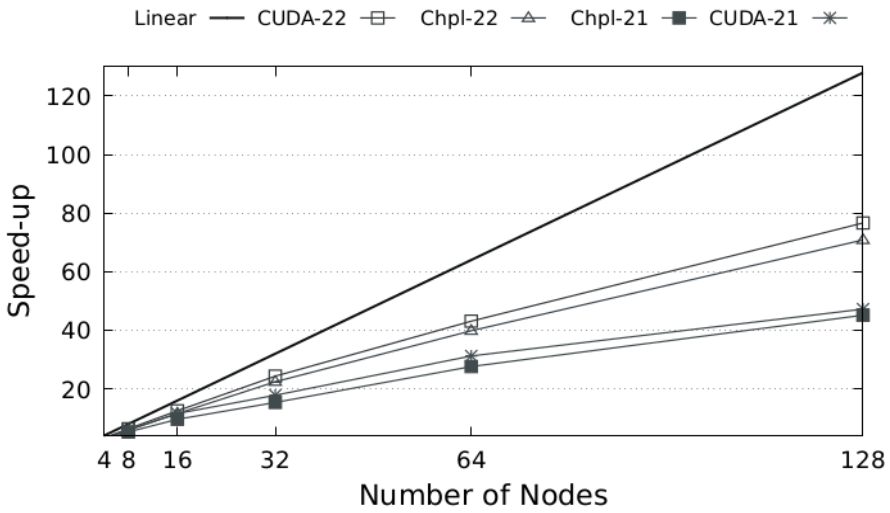


(b) AMD-based system

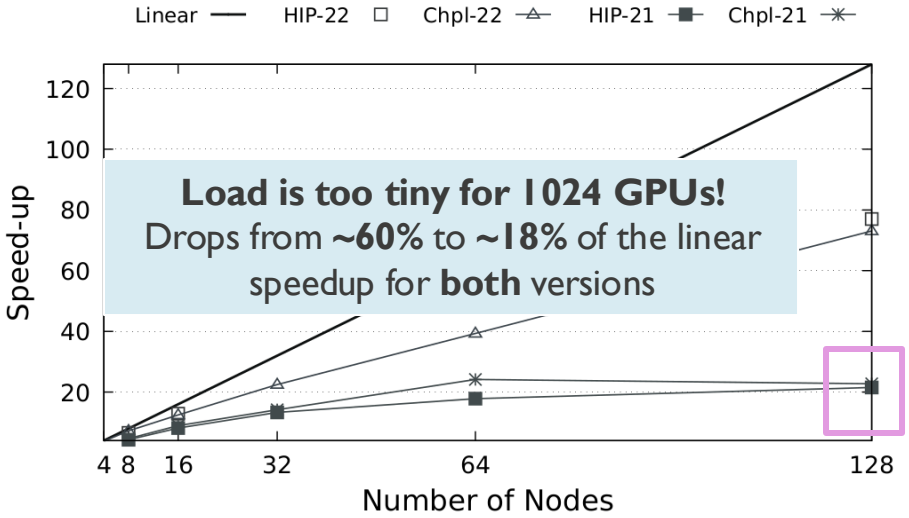
Performance Results

Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline



(a) NVIDIA-based System

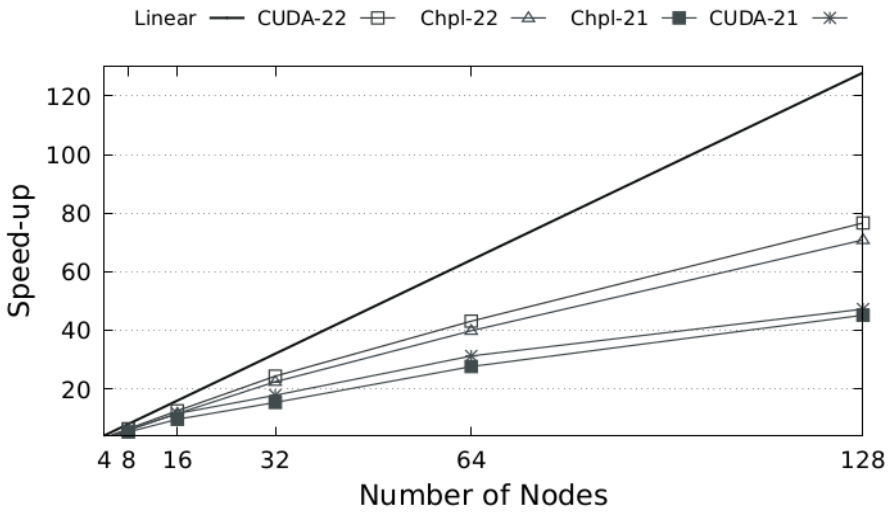


(b) AMD-based system

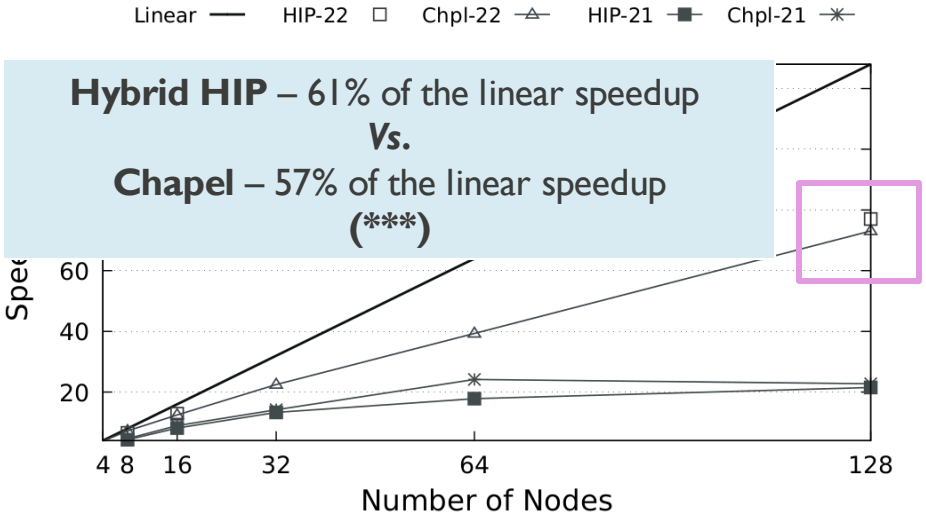
Performance Results

Strong Scaling

- N-Queens of sizes N=21 and N=22
- 4-128 computer nodes, speedup compared to the Baseline



(a) NVIDIA-based System



Hybrid HIP – 61% of the linear speedup

Vs.

Chapel – 57% of the linear speedup

(***)

(b) AMD-based system

Research Questions

1. Is it possible to achieve both code and performance portability in distributed tree-based search with Chapel's native GPU support?
    1. **Yes!**
    2. **Minor performance loss** on NVIDIA GPUs and equivalent performance on AMD for big problems.
    3. **Chapel scales similarly** on both systems – up to 128 nodes.

Programming Effort

Programming Effort

Single-node – Multi-GPU

- The HIP, CUDA and Chapel-GPU applications are equivalent in terms of SLOC
  - Single-node – Multi-GPU
  - One thread per GPU
  - N-Queens kernel

Programming Effort

Single-node – Multi-GPU

- The HIP, CUDA and Chapel-GPU applications are equivalent in terms of SLOC
  - Single-node – Multi-GPU
  - One thread per GPU
  - N-Queens kernel
- **Chapel-GPU:** the benefit is to get rid of both HIP and CUDA applications
  - Low/no performance loss for the medium/big problems

Programming Effort

Single-node – Multi-GPU

- The HIP, CUDA and Chapel-GPU applications are equivalent in terms of SLOC
  - Single-node – Multi-GPU
  - One thread per GPU
  - N-Queens kernel
- **Chapel-GPU:** the benefit is to get rid of both HIP and CUDA applications
  - Low/no performance loss for the medium/big problems
  - Only one application

Programming Effort

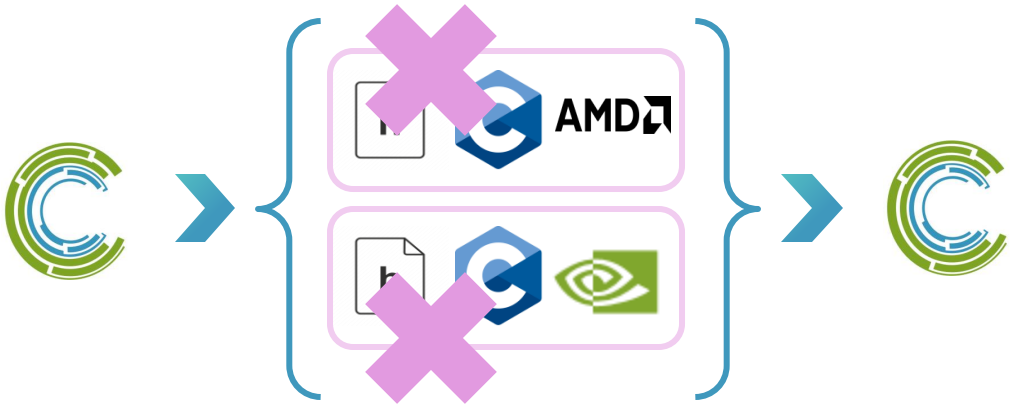
Distributed

- Replacing one hybrid version (CUDA or HIP) with Chapel's native GPU support results in an application 30% shorter.
  - Getting rid of both interoperability codes (CUDA and HIP wrappers) results in a final code 65% shorter.

Programming Effort

Distributed

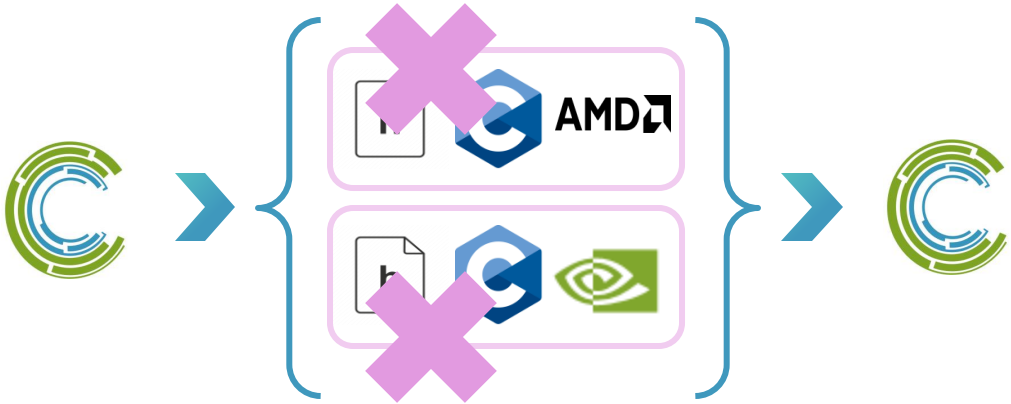
- Replacing one hybrid version (CUDA or HIP) with Chapel's native GPU support results in an application 30% shorter.
  - Getting rid of both interoperability codes (CUDA and HIP wrappers) results in a final code 65% shorter.



Programming Effort

Distributed

- Replacing one hybrid version (CUDA or HIP) with Chapel's native GPU support results in an application 30% shorter.
  - Getting rid of both interoperability codes (CUDA and HIP wrappers) results in a final code 65% shorter.
  - Only one kernel version to keep
  - Almost no performance loss for distributed execution - biggest loads



Research Questions

1.

2. Is it worth in terms of programming effort?
   1. **Yes!**
   2. **Single-node:** a single application for two different vendors
   3. **Distributed:** getting rid of the kernels and its wrappers results in a much shorter application – small performance loss

Conclusions and Perspectives

Conclusions

- Chapel efficiently unifies the different parallel levels of GPU-powered clusters: inter-node and intra-node (CPUs and GPUs).
  - This allows one to holistically deal with the two major challenges of exascale computing in a unified/productive way:
    - Intra-node and inter-node scalability
    - GPU-aware heterogeneity

Conclusions

- The Chapel-GPU *vs.* its hybrid distributed counterparts achieved:
  - Similar parallel performance
  - Similar strong scaling efficiency on 128 nodes.
- It is possible to achieve both code portability and performance portability in distributed tree-based search with Chapel's native GPU support.
- Using Chapel's Native GPU support instead of interoperability results in a distributed application 65% shorter.

Conclusions

- The main benefit Chapel brings: the **incremental parallelism**:
  - Consider the scenario when the application is developed from scratch.
    - **A first version would likely be written for CPUs**:
      - Serial then multicore
    - **From multicore to distributed:**
      - Due to the unified memory model (PGAS), a distributed application in Chapel can be very similar to its multicore counterpart.
    - **From CPU to GPU:** how to write and compile the mix of OpenMP+CUDA or MPI+CUDA?
- In Chapel, all these steps are significantly easier than other paradigms:
  - One language for CPUs, GPUs, single-node and distributed computing
  - No mix of programming models

Perspectives

- Study the generated target code to better understand the results
- **From the N-Queens to Combinatorial Optimization Problems**:
  - Branch-and-Bound search instead of distributed Backtracking
  - Variants of the N-Queens, FSP, VRP, MinLA, etc.
    - (e.g.; FSP Ta057 ~30-years PFSP instance)
  - Checkpointing for solving to the optimality larger problems
- More efficient ways of exploiting intra-node parallelism?
  - The present algorithm was conceived for small-sized clusters
  - Is there a more efficient way of exploiting intra-node parallelism?
    - One locale/sublocale per GPU?
    - Is it possible to use all GPUs/CPU cores to decrease execution time?

Acknowledgments

- This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254) -- Project EU2022D08-197.
- It is also 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.
- This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231 using NERSC award ASCR-ERCAP-mp215. In addition, this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Thank you!

Questions?

https://github.com/tcarneirop/ChOp

Chapel-based Optimization on Github

tiago.carneiropessoa@imec.be


imec
embracing a better life