Modeling and DVFS for the Energy Optimisation of HPC I/Os

Louis-Marie Nicolas[*]
Philippe Couvée[+]
Salim Mimouni[+]
Jalil Boukhobza[*]

[*]Lab-STICC, CNRS UMR 6285, ENSTA | IP Paris, 29806 Brest, France
[+]Atos BDS R&D Data Management, 38100 Grenoble, France

Context

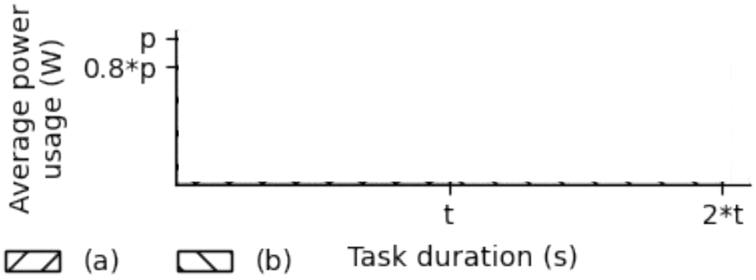
- The scale and power usage of HPC clusters is growing.
  The 10 most powerful clusters used 63 MW in 2014, 156 MW in 2024 [1].
- Energy has a cost, both economical and environmental, with HPC projected to be responsible for up to 8% of the worldwide CO_2 emissions in 2030 [2].
- While storage consume less energy than compute, the gap of performance between persistent storage and memory means storage can be a performance bottleneck [3], lengthening the application duration and wasting energy.
- Multiple techniques to balance energy and performance, amongst which Dynamic Voltage and Frequency Scaling (DVFS);

1: TOP500. Online; accessed 16. Jan. 2025. https://top500.org/lists/top500/
2: Li, Baolin, et al. "Toward sustainable hpc: Carbon footprint estimation and environmental implications of hpc systems.", SC'23
3: Lüttgau, Jakob, et al. "Survey of storage systems for high-performance computing." Supercomputing Frontiers and Innovations 5.1 (2018)

Background

DVFS for Energy Optimization

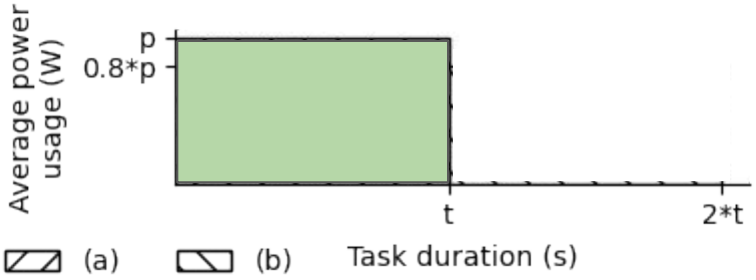
- **Reduced CPU frequency : lower power usage, lower performance.**
- When CPU performance has a low impact on the running task duration, energy can be saved by lowering the CPU frequency.
- When CPU performance has a high impact on the running task duration, reducing the frequency can lead to an increased energy cost.



Background

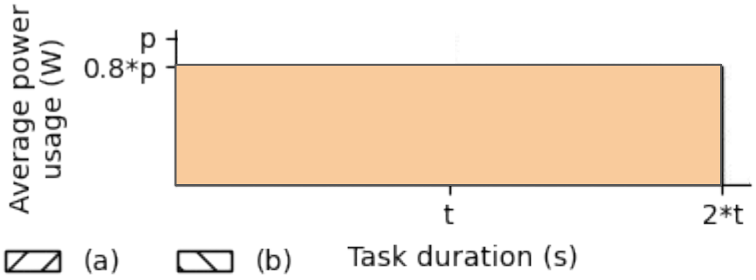
DVFS for Energy Optimization

- **Reduced CPU frequency : lower power usage, lower performance.**
- When CPU performance has a low impact on the running task duration, energy can be saved by lowering the CPU frequency.
- When CPU performance has a high impact on the running task duration, reducing the frequency can lead to an increased energy cost.



Background
DVFS for Energy Optimization

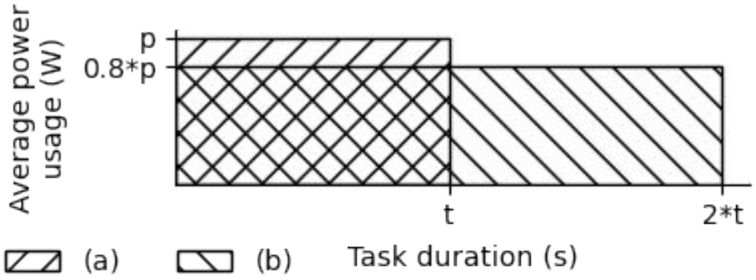
- **Reduced CPU frequency : lower power usage, lower performance.**
- When CPU performance has a low impact on the running task duration, energy can be saved by lowering the CPU frequency.
- When CPU performance has a high impact on the running task duration, reducing the frequency can lead to an increased energy cost.



Background

DVFS for Energy Optimization

- **Reduced CPU frequency : lower power usage, lower performance.**
- When CPU performance has a low impact on the running task duration, energy can be saved by lowering the CPU frequency.
- When CPU performance has a high impact on the running task duration, reducing the frequency can lead to an increased energy cost.



Background

DVFS in HPC

- Reducing the CPU frequency on compute tasks was shown to lead to an increased total energy consumption and a worse performance [1].
- Reducing the CPU frequency on memory-bound tasks or some MPI tasks was shown to lead to a reduced energy consumption, at the cost of a slightly worse performance [1].
- However, to the best of our knowledge, **the effect of DVFS on HPC I/Os was not covered by the literature**

1: Calore, Enrico, et al. “Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications.”, CCPE’17

Background

I/O Modeling

- In order to precisely apply DVFS, an I/O model of the HPC applications is necessary.

Background

I/O Modeling

- 3 main approaches to I/O modeling and prediction in the litterature:

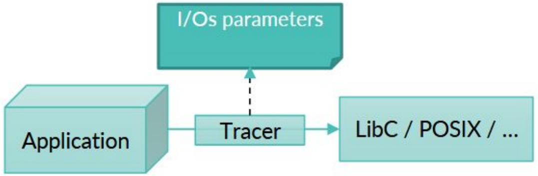
Background
I/O Modeling

- 3 main approaches to I/O modeling and prediction in the litterature:
  - White-box
    - Access and/or modification of an application source code.
    - Adding hints or prefetching primitives to the application code

Background
I/O Modeling

- 3 main approaches to I/O modeling and prediction in the litterature:
  - White-box
    - Access and/or modification of an application source code.
    - Adding hints or prefetching primitives to the application code

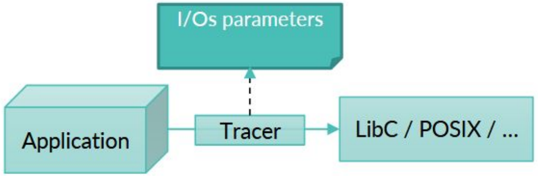
  - Black-box
    - Intercepting I/Os.
    - Pattern matching, probabilistic models



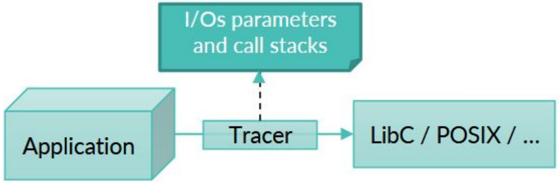
Background
I/O Modeling

- 3 main approaches to I/O modeling and prediction in the litterature:
  - White-box
    - Access and/or modification of an application source code.
    - Adding hints or prefetching primitives to the application code

  - Black-box
    - Intercepting I/Os.
    - Pattern matching, probabilistic models



  - Grey-box [1]
    - Intercepting I/Os call stacks.
      Extracting knowledge about an application I/O structure using I/O call stacks.



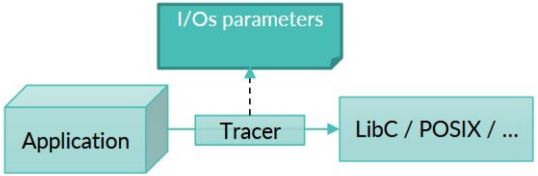
1: Dorier, Matthieu, et al. "Omnisc'IO: A Grammar-Based Approach to Spatial and Temporal I/O Patterns Prediction", SC'14

Background

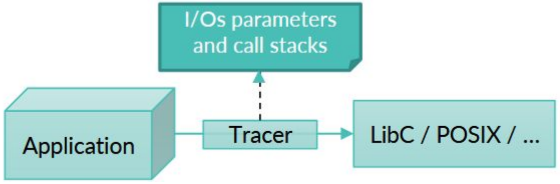
I/O Modeling

- 3 main approaches to I/O modeling and prediction in the litterature:
  - White-box → **need source code**
    - Access and/or modification of an application source code.
    - Adding hints or prefetching primitives to the application code

  - Black-box → **scaling issues**
    - Intercepting I/Os.
    - Pattern matching, probabilistic models



  - Grey-box [1] → **deterministic I/Os only**
    - Intercepting I/Os call stacks.
      Extracting knowledge about an application I/O structure using I/O call stacks.



1: Dorier, Matthieu, et al. "Omnisc'IO: A Grammar-Based Approach to Spatial and Temporal I/O Patterns Prediction", SC'14

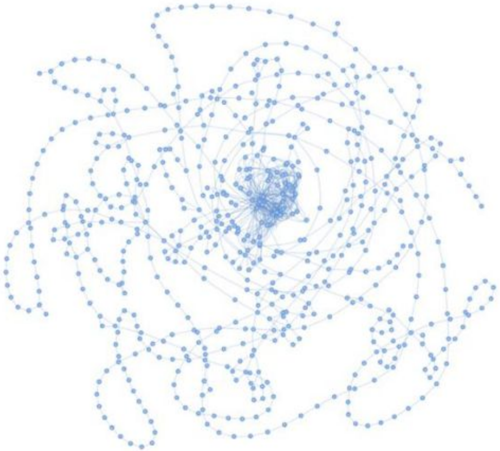
Problem Statements

Hence the problem statements:

- What is the effect of Dynamic Voltage and Frequency Scaling on HPC I/Os?
- How to create a low-overhead I/O model for both deterministic and non-deterministic I/Os without access to the application source code?

I/O Modeling
Overview

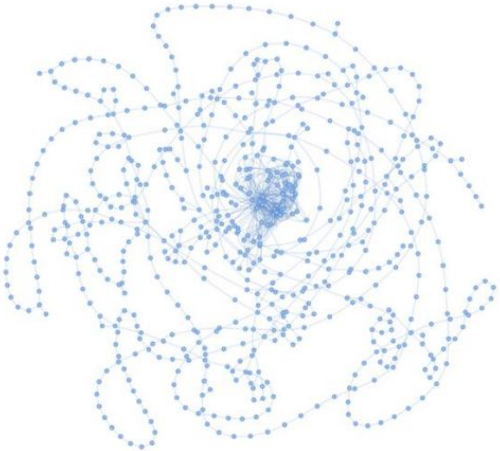
- We created GrIOt, a grey-box approach based on a directed graph of call stacks.
  - Bounded size, depending on the number of unique call stacks.
  - Near O(1) prediction and update thanks to an hash map
  - Can support non-deterministic I/Os by adding metadata to nodes and edges



I/O Modeling

Overview

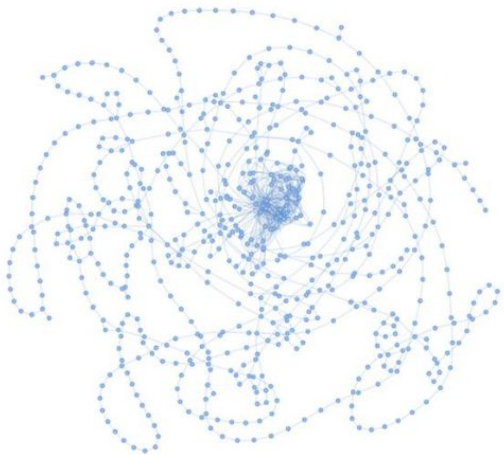
- We created GrIOt, a grey-box approach based on a directed graph of call stacks.
  - Bounded size, depending on the number of unique call stacks.
  - Near O(1) prediction and update thanks to an hash map
  - Can support non-deterministic I/Os by adding metadata to nodes and edges



- 1 node = 1 or more I/O call stacks
- 1 outgoing edge = 1 possible “next” I/O call stack

I/O Modeling
Overview

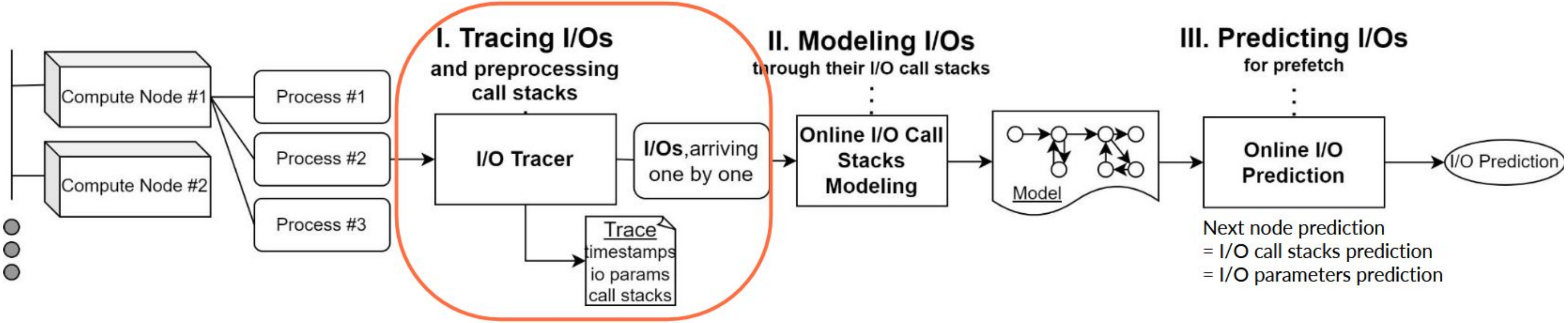
- We created GrIOt, a grey-box approach based on a directed graph of call stacks.
  - Bounded size, depending on the number of unique call stacks.
  - Near O(1) prediction and update thanks to an hash map
  - Can support non-deterministic I/Os by adding metadata to nodes and edges



  - 1 node = 1 or more I/O call stacks
  - 1 outgoing edge = 1 possible “next” I/O call stack
  - **New:** 1 graph per process, or 1 graph per file.

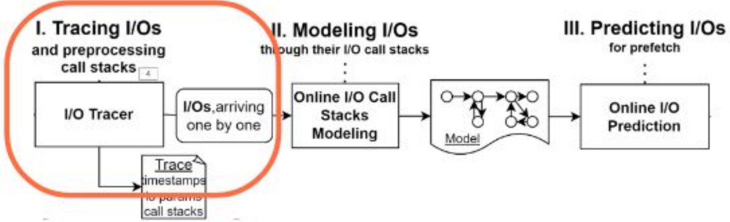
I/O Modeling

Overview



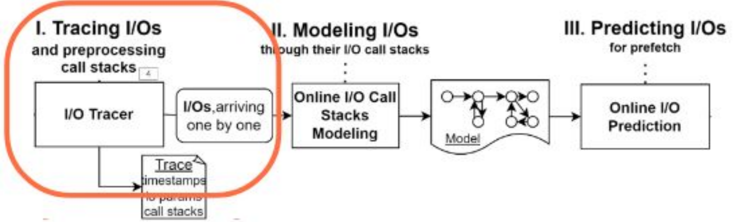
I/O Modeling

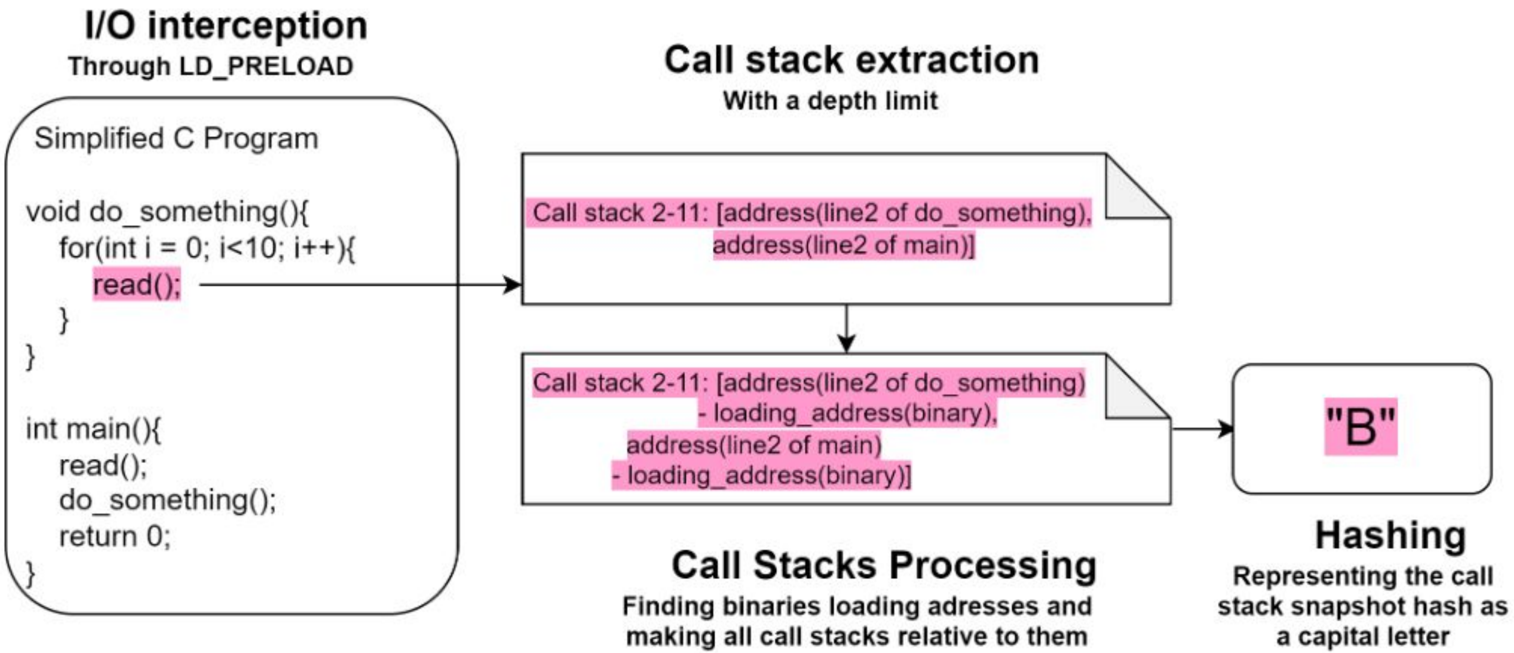
Tracing I/Os

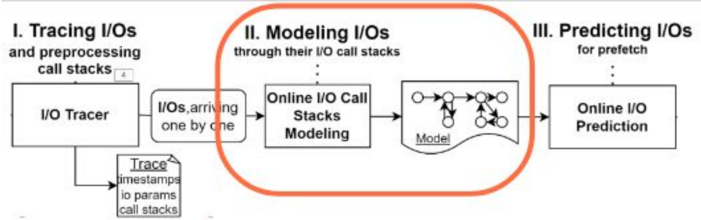


- POSIX and Lib-C I/O function call interception through LD_PRELOAD
    - Indirect support of libraries such as HDF5 or MPI-IO
- Obtain the (relative) call stack and I/O parameters of every I/O
- Optional tracing of I/O call stacks for debug, as existing tracers did not support them

I/O Modeling
Tracing I/Os





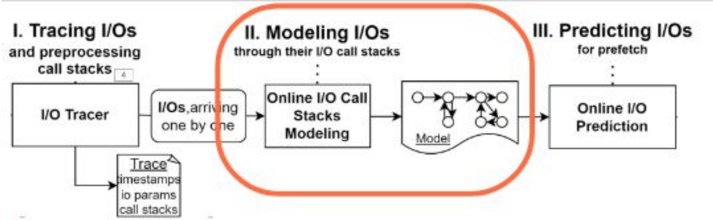


I/O Modeling

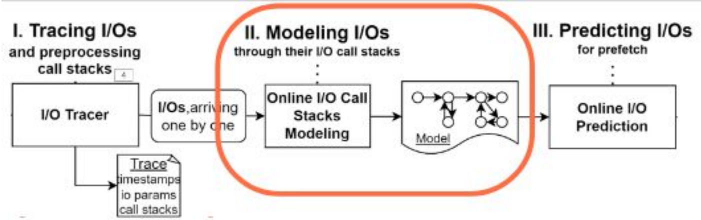
Modeling

- I/O after I/O, GrIOt creates an I/O call stack graph
- When a new I/O call stack “A” is discovered, a graph node is created.
- When a new I/O call stack transition “A→B” is discovered, an edge is created.

I/O Modeling
Modeling



- I/O after I/O, GrIOt creates an I/O call stack graph
- When a new I/O call stack “A” is discovered, a graph node is created.
- When a new I/O call stack transition “A→B” is discovered, an edge is created.
- 2 modeling granularities:
  - 1 graph per process
  - 1 graph per “open” call stack

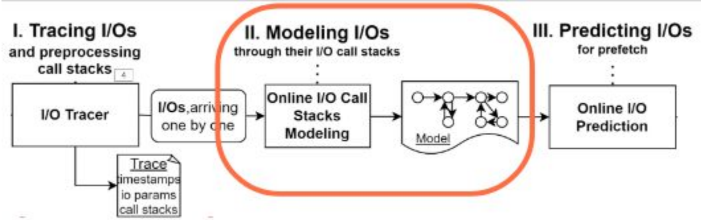


I/O Modeling

Modeling

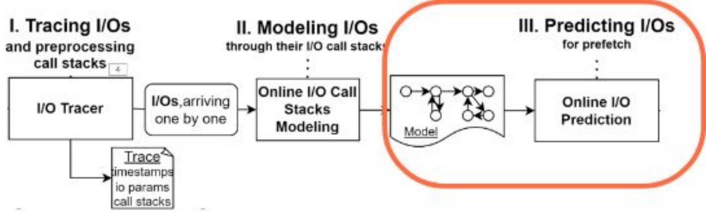
- I/O after I/O, GrIOt creates an I/O call stack graph
- When a new I/O call stack “A” is discovered, a graph node is created.
- When a new I/O call stack transition “A→B” is discovered, an edge is created.
- 2 modeling granularities:
  - 1 graph per process (previous version of GrIOt)
  - 1 graph per “open” call stack

I/O Modeling
Modeling

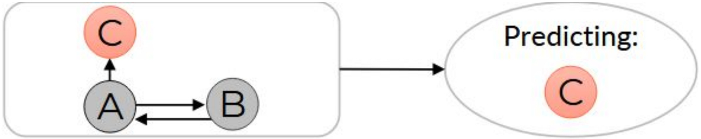


- I/O after I/O, GrIOt creates an I/O call stack graph
- When a new I/O call stack “A” is discovered, a graph node is created.
- When a new I/O call stack transition “A→B” is discovered, an edge is created.
- 2 modeling granularities:
  - 1 graph per process (previous version of GrIOt)
  - 1 graph per “open” call stack → enables per-file I/O prediction & model reuse

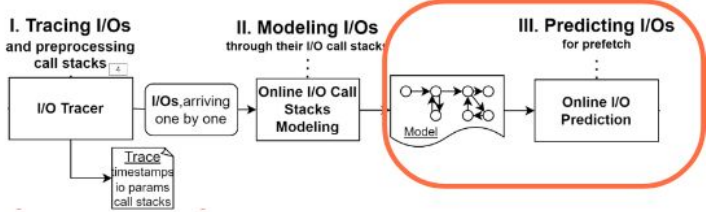
I/O Modeling
Predicting



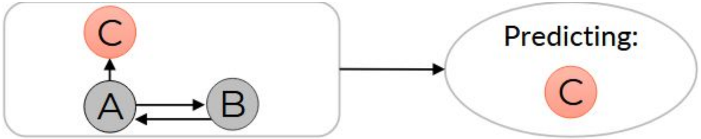
- If the node corresponding to the previous I/O call stack has no outgoing edge:



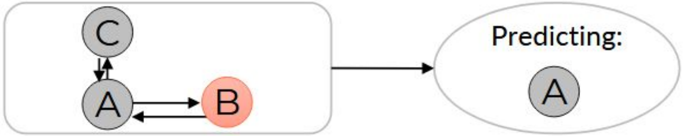
I/O Modeling
Predicting

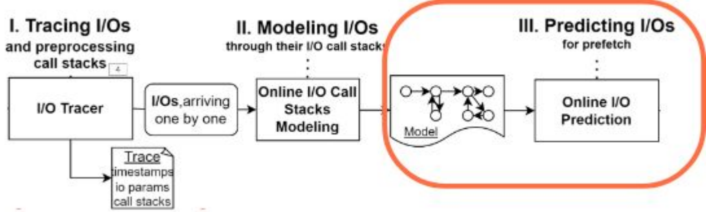


- If the node corresponding to the previous I/O call stack has no outgoing edge:



- If the node has a single outgoing edge:

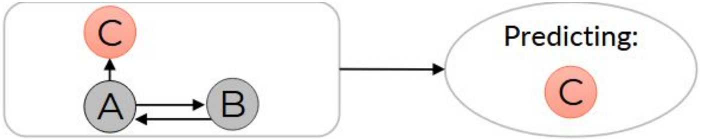




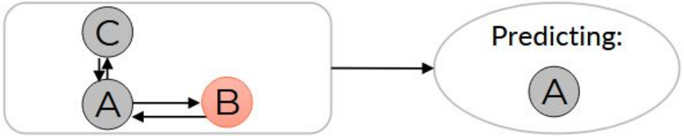
I/O Modeling

Predicting

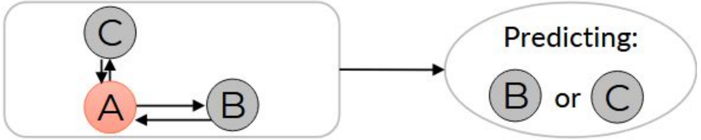
- If the node corresponding to the previous I/O call stack has no outgoing edge:



- If the node has a single outgoing edge:



- If the node has more than a single edge:



I/O Modeling
Methodology

5 applications:

- NAMD: Molecular dynamics
- LAMMPS: Molecular dynamics
- Xcompact3d: Navier Stokes solver
- LQCD: Quantic chromodynamics
- Nemo: Ocean simulation

I/O Modeling

Methodology

Application	Description	Nodes	Processes	I/O volume	# unique call stacks	# unique call stacks transitions	% of repeating call stacks
NAMD	Molecular Dynamics, 1.1M Atoms: STMV 210M	12	12	81.5GB	371	718	9.32%
LAMMPS	Molecular Dynamics, 10k Atoms: 3NIR Crambin	14	896	13.1GB	39	52	80.18%
Xcompact3d	Navier-Stokes solver	10	640	13.8GB	85	110	0.28%
LQCD	Quantic chromodynamics	16	3072	73.0GB	319	643	97.91%
Nemo	Ocean simulation	8	256	22.8GB	229	312	32.52%

I/O Modeling
Methodology

Purpose: Evaluating the overhead and accuracy of both model granularities.

- We run all 5 applications with both model granularities
- We compare GrIOt the state-of-the-art, Omnisc'IO
- We run all 5 applications again with only the I/O call stack instrumentation, with varying call stack depth, and compare POSIX *backtrace* with *libunwind*

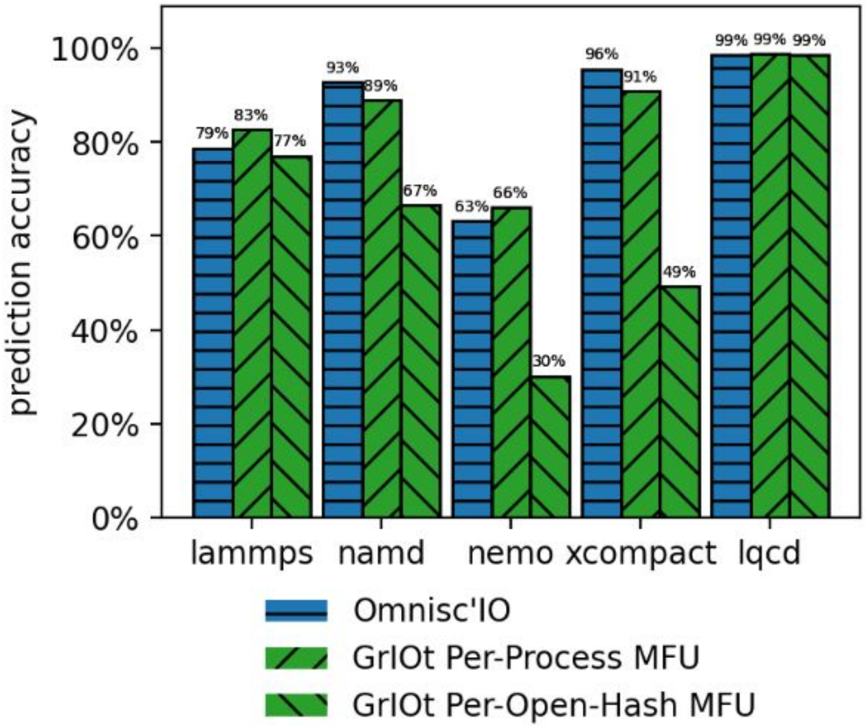
I/O Modeling

Experimental Evaluation

- 20x computes nodes, with 2x AMD EPYC 7282 16-Core Processor each.
- Each CPU core supports only 3 CPU-frequencies: 1.5Ghz, 2.0Ghz, 2.8Ghz
- A GPFS file system is used. It is under GPFS v5.1.8.0, with 8 volumes of 50TB, for a total volume of 400TB.
- The Linux page cache and GPFS page pool are cleared between experiments

I/O Modeling

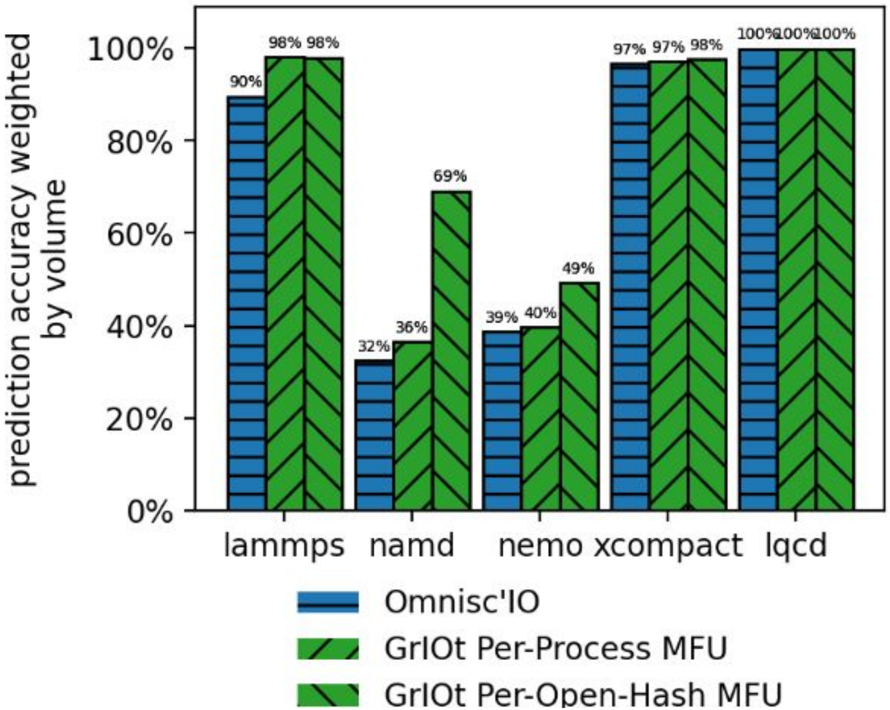
Experimental Evaluation: GrIOt VS Omnisc'IO, Accuracy



GrIOt per-process is similar to Omnisc'IO in performance. GrIOt per open call stack is either similar or worse, depending on the application.

I/O Modeling

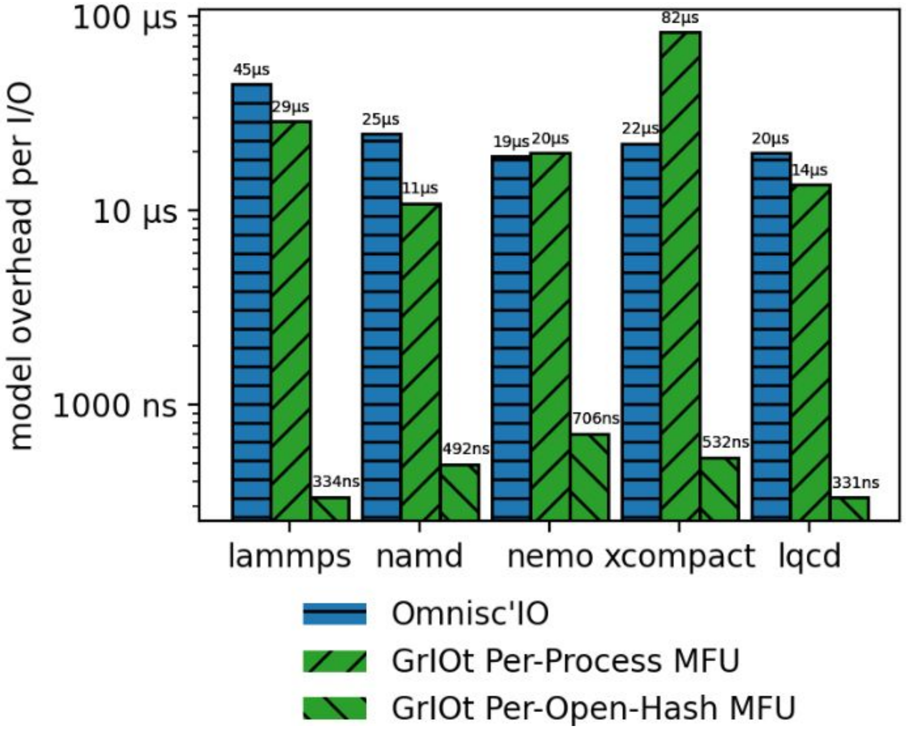
Experimental Evaluation: GrIOt VS Omnisc'IO, Weighted Accuracy



When accuracy is weighted by volume, it's the opposite: GrIOt per open call stack has similar or better performance on all applications.

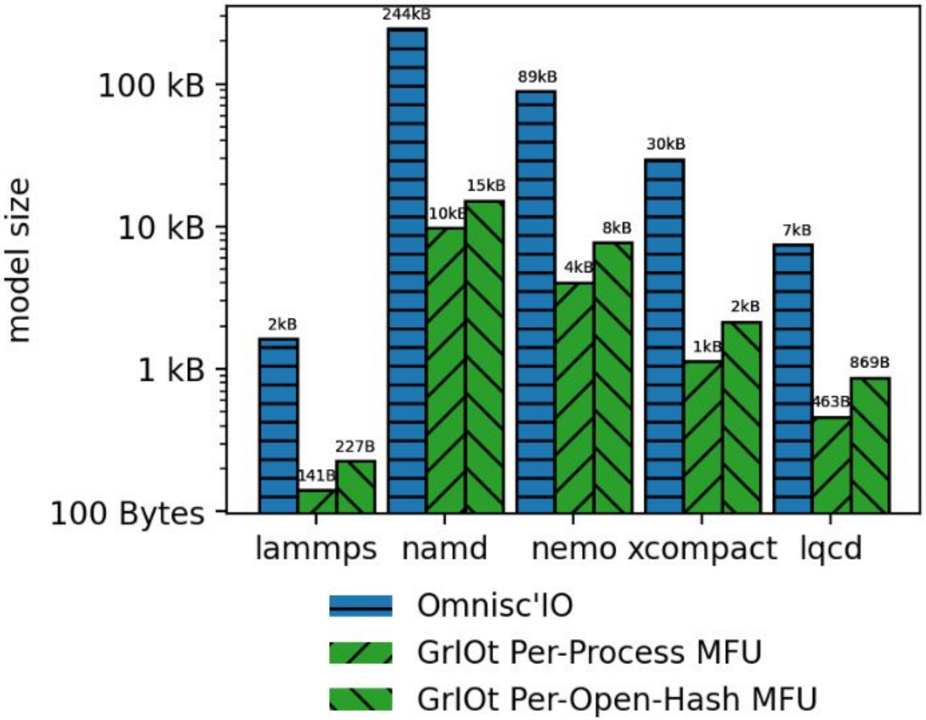
I/O Modeling

Experimental Evaluation: GrIOt VS Omnisc'IO, Model Overhead



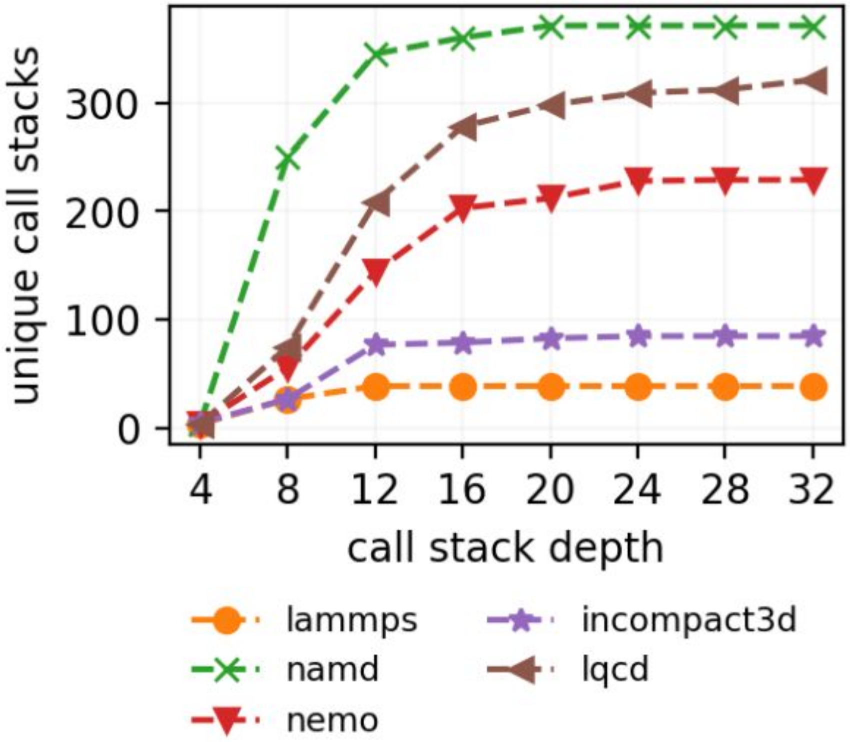
I/O Modeling

Experimental Evaluation, GrIOt VS Omnisc'IO, Model size



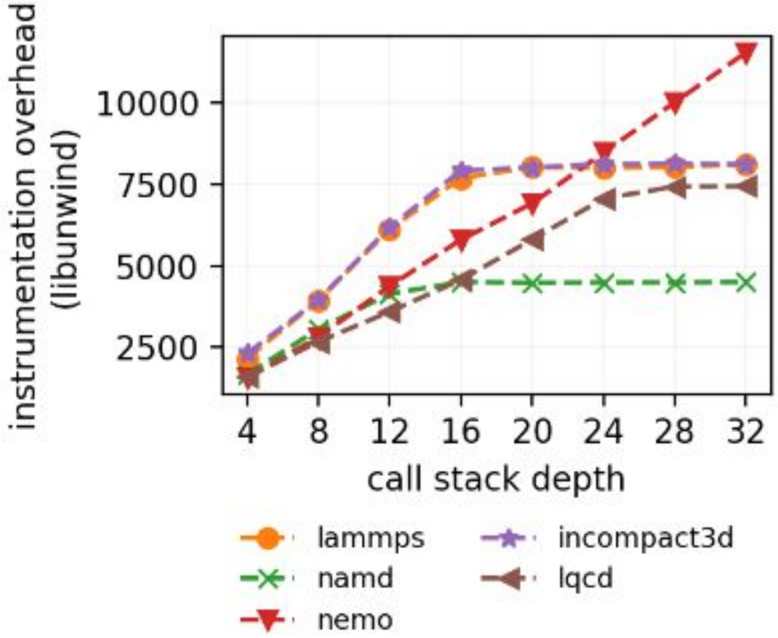
I/O Modeling

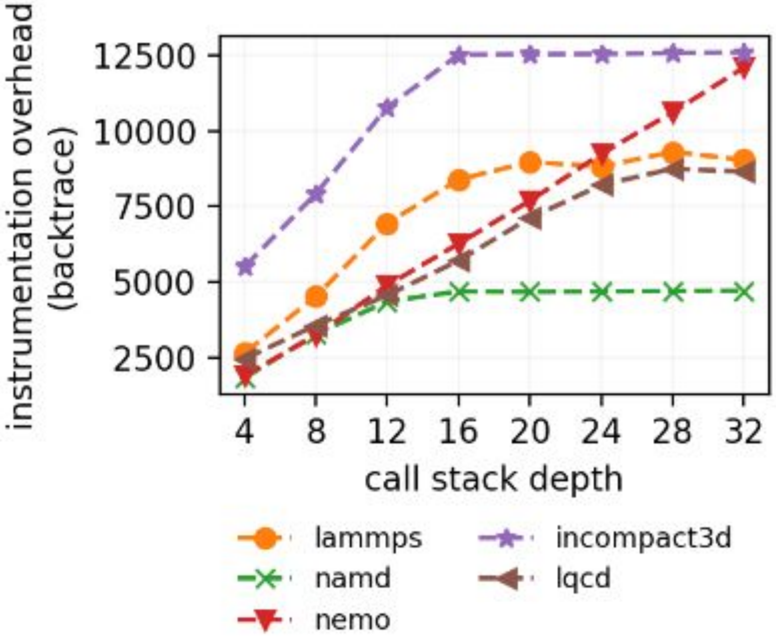
Experimental Evaluation: Call stack depth VS Call stack differentiation



I/O Modeling

Experimental Evaluation: POSIX backtrace VS libunwind





I/O Modeling
Key Takeaways

- While GrIOt have a similar accuracy to Omnisc'IO, it has a better weighted accuracy (up to +90% on NAMD)

I/O Modeling

Key Takeaways

- While GrIOt have a similar accuracy to Omnisc'IO, it has a better weighted accuracy (up to +90% on NAMD)
- GrIOt with its per open call stack granularity has a much lower overhead as well

I/O Modeling
Key Takeaways

- While GrIOt have a similar accuracy to Omnisc'IO, it has a better weighted accuracy (up to +90% on NAMD)
- GrIOt with its per open call stack granularity has a much lower overhead as well
- Both GrIOt granularity have a much lower model size

I/O Modeling

Key Takeaways

- While GrIOt have a similar accuracy to Omnisc'IO, it has a better weighted accuracy (up to +90% on NAMD)
- GrIOt with its per open call stack granularity has a much lower overhead as well
- Both GrIOt granularity have a much lower model size
- It is not possible to reduce call stack depth to gain performance without losing information

I/O Modeling

Key Takeaways

- While GrIOt have a similar accuracy to Omnisc'IO, it has a better weighted accuracy (up to +90% on NAMD)
- GrIOt with its per open call stack granularity has a much lower overhead as well
- Both GrIOt granularity have a much lower model size
- It is not possible to reduce call stack depth to gain performance without losing information
- libunwind seems to have a better performance than POSIX backtrace

DVFS
Overview

In order to characterize DVFS for HPC, we provide an experimental methodology :

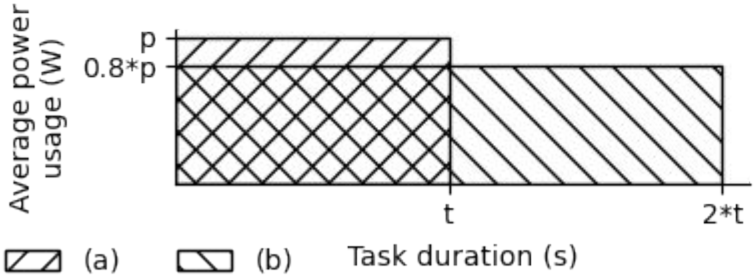
- Selecting the evaluation metrics
- Selecting the synthetic workloads
- Executing the workloads with varying CPU frequencies

DVFS

Methodology

3 metrics for performance and energy:

- Application duration (s)
- Average power (W, that is $J.s^{-1}$)
- Energy consumption (J)



We use an out-of-band energy monitoring tool, that communicates with the Baseboard Management Controllers. As such, energy instrumentation includes every physical component on the instrumented compute nodes.

DVFS
Methodology

3 parallel configurable synthetic workloads, with one process per core:

- A CPU-bound compute task
- A Memory-bound compute task
- A sequential I/O benchmark

DVFS
Methodology

Name	Category	% CPU idle	% CPU waiting for I/Os	%CPU working	Duration (2.8 Ghz, all C-states)	Parameters
CPU-Fakeapp	Compute task (CPU-bound)	2%	0%	98%	75 s	Number of pseudo-random numbers to generate per process = 20e9
Memory-Fakeapp	Compute task (Memory-bound)	2%	0%	98%	115 s	Volume of memory to access per process = 400 GB
I/O-Fakeapp	File I/O Data dependency (MPI-IO)	97%	2%	1%	100-500 s (depending on I/O size)	I/O size, I/O type (buffered or direct), I/O count = variable

DVFS

Methodology

Name	Category	% CPU idle	% CPU waiting for I/Os	%CPU working	Duration (2.8 Ghz, all C-states)	Parameters
CPU-Fakeapp	Compute task (CPU-bound)	2%	0%	98%	75 s	Number of pseudo-random numbers to generate per process = 20e9
Memory-Fakeapp	Compute task (Memory-bound)	2%	0%	98%	115 s	Volume of memory to access per process = 400 GB
I/O-Fakeapp	File I/O Data dependency (MPI-IO)	97%	2%	1%	100-500 s (depending on I/O size)	I/O size, I/O type (buffered or direct), I/O count = variable

DVFS

Methodology

Name	Category	% CPU idle	% CPU waiting for I/Os	%CPU working	Duration (2.8 Ghz, all C-states)	Parameters
CPU-Fakeapp	Compute task (CPU-bound)	2%	0%	98%	75 s	Number of pseudo-random numbers to generate per process = 20e9
Memory-Fakeapp	Compute task (Memory-bound)	2%	0%	98%	115 s	Volume of memory to access per process = 400 GB
I/O-Fakeapp	File I/O Data dependency (MPI-IO)	97%	2%	1%	100-500 s (depending on I/O size)	I/O size, I/O type (buffered or direct), I/O count = variable

DVFS
Methodology

Name	Category	% CPU idle	% CPU waiting for I/Os	%CPU working	Duration (2.8 Ghz, all C-states)	Parameters
CPU-Fakeapp	Compute task (CPU-bound)	2%	0%	98%	75 s	Number of pseudo-random numbers to generate per process = 20e9
Memory-Fakeapp	Compute task (Memory-bound)	2%	0%	98%	115 s	Volume of memory to access per process = 400 GB
I/O-Fakeapp	File I/O Data dependency (MPI-IO)	97%	2%	1%	100-500 s (depending on I/O size)	I/O size, I/O type (buffered or direct), I/O count = variable

DVFS

Methodology

Purpose: Analyzing the effect of setting the CPU frequency during P-states

- Using the *userspace* cpufreq governor to set a CPU frequency target
- All C-states are enabled
- Running all 3 synthetic workloads 5 times with all the supported CPU frequencies

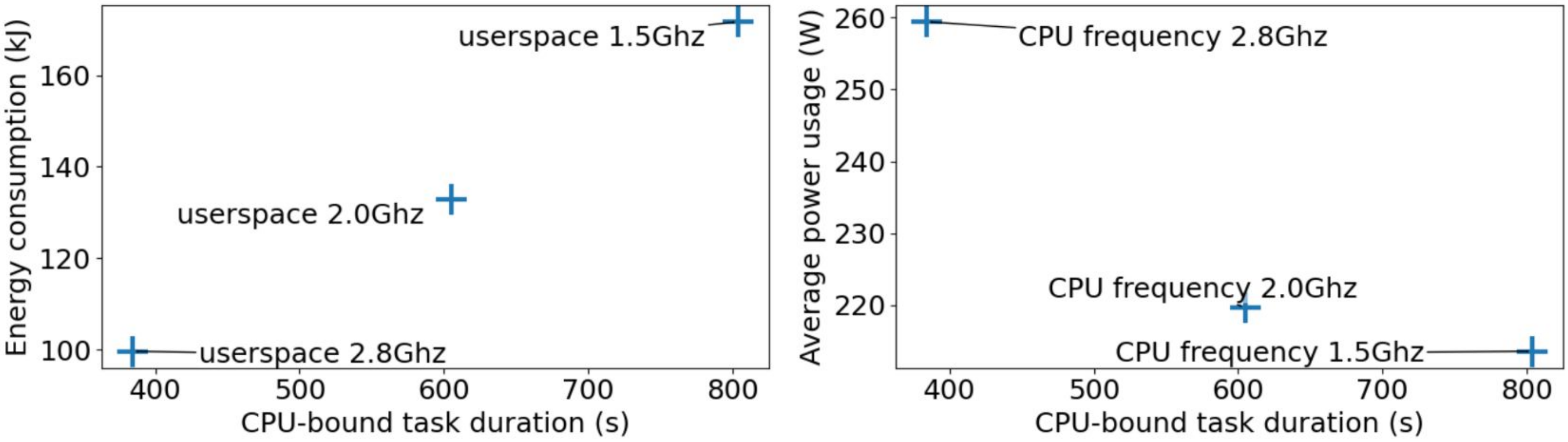
DVFS

Experimental Evaluation

- 20x computes nodes, with 2x AMD EPYC 7282 16-Core Processor each.
- Each CPU core supports only 3 CPU-frequencies: 1.5Ghz, 2.0Ghz, 2.8Ghz
- A GPFS file system is used. It is under GPFS v5.1.8.0, with 8 volumes of 50TB, for a total volume of 400TB.
- The Linux page cache and GPFS page pool are cleared between experiments

DVFS

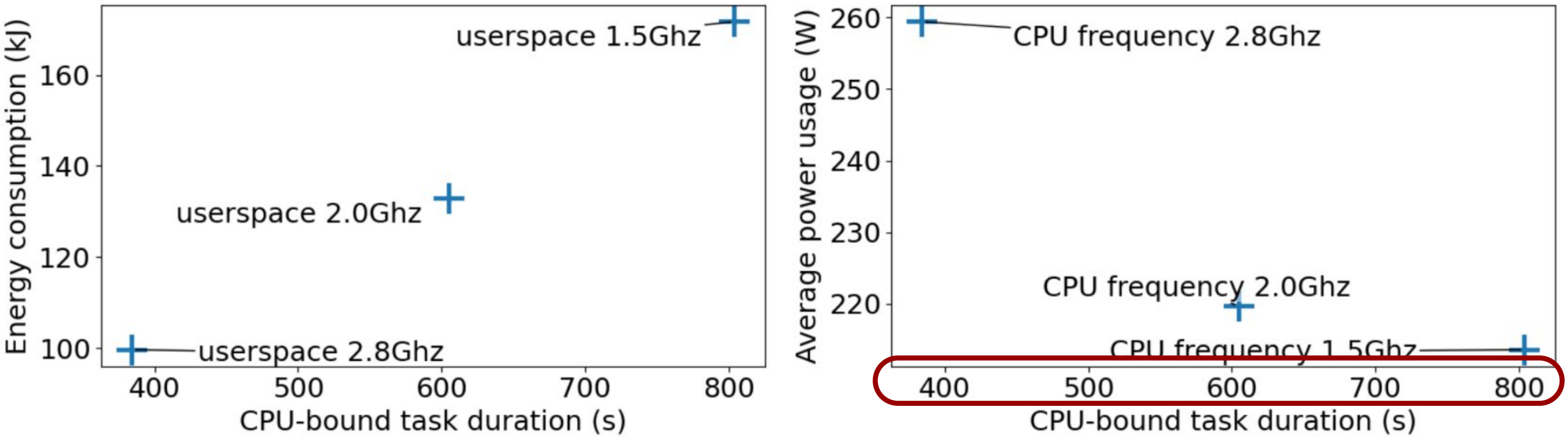
Experimental Evaluation: P-states for the CPU-bound workload



- On CPU-bound tasks, reducing the CPU frequency leads to both a lower performance AND to an increased total energy consumption (up to +70%)

DVFS

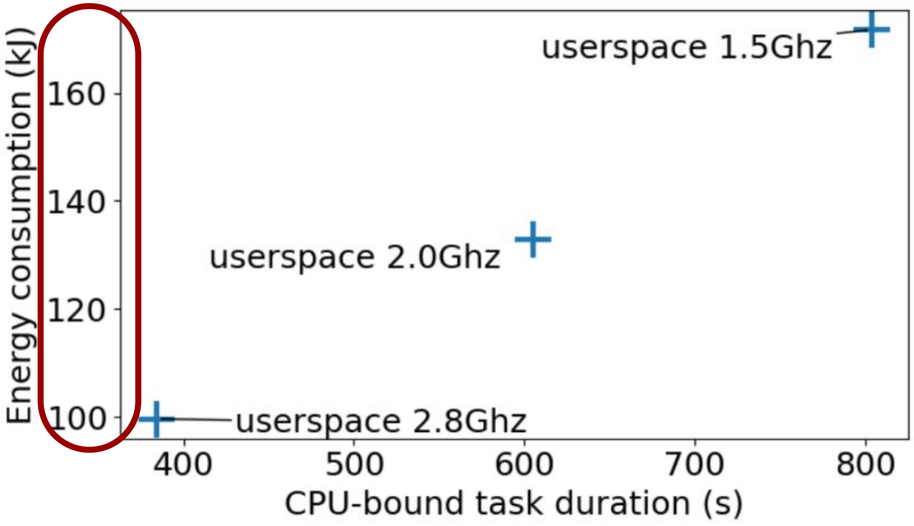
Experimental Evaluation: P-states for the CPU-bound workload

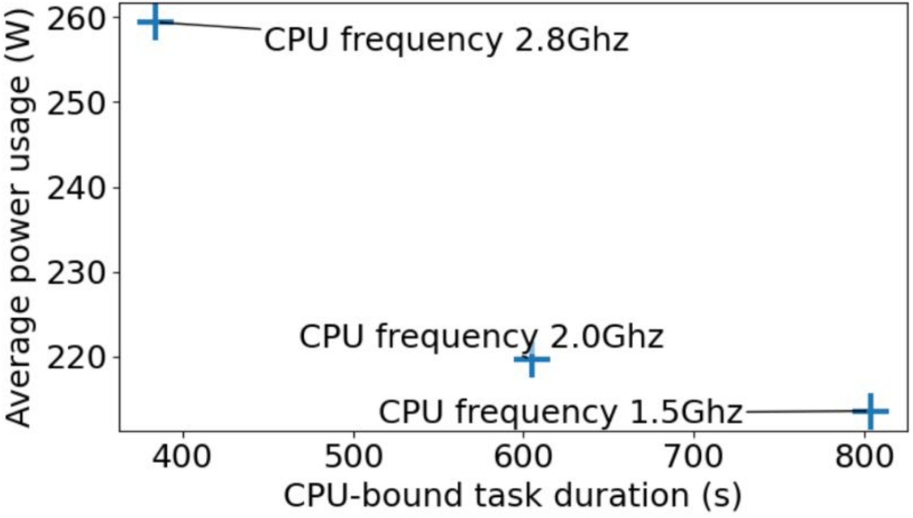


- On CPU-bound tasks, reducing the CPU frequency leads to both a lower performance AND to an increased total energy consumption (up to +70%)

DVFS

Experimental Evaluation: P-states for the CPU-bound workload

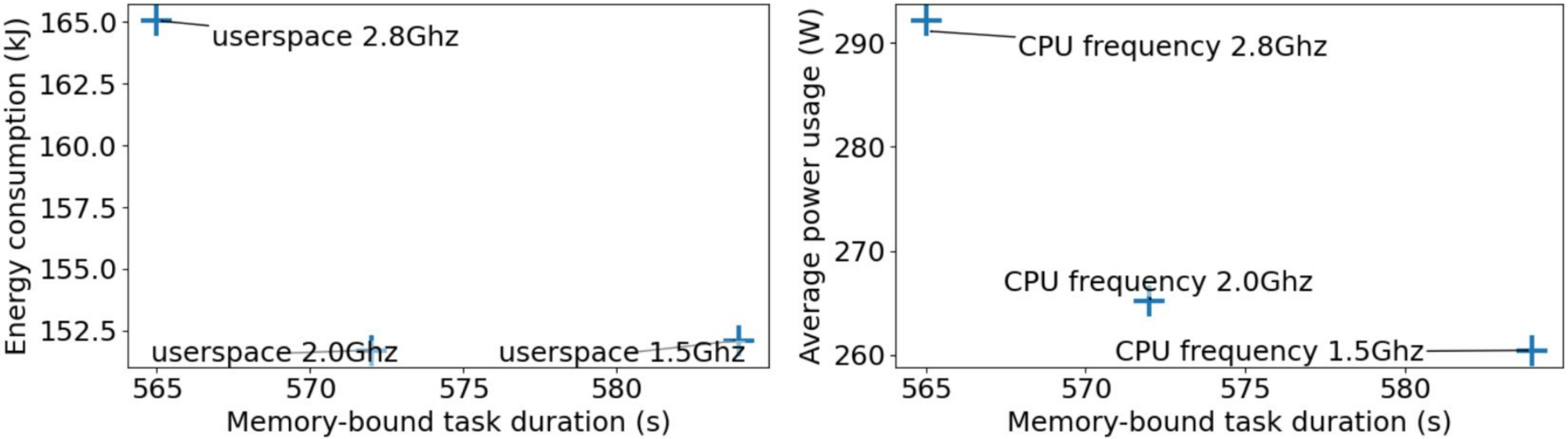




- On CPU-bound tasks, reducing the CPU frequency leads to both a lower performance AND to an increased total energy consumption (up to +70%)

DVFS

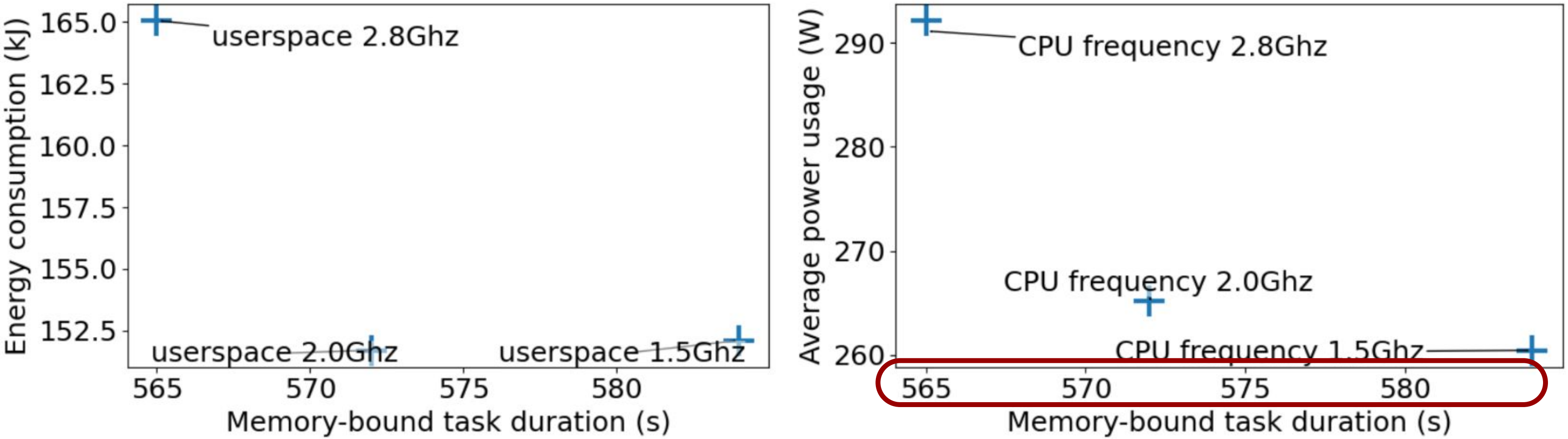
Experimental Evaluation: P-states for the memory-bound workload



- On memory-bound tasks, reducing the CPU frequency leads to a slightly lower performance (-4%) and to a reduced total energy consumption (-9%)

DVFS

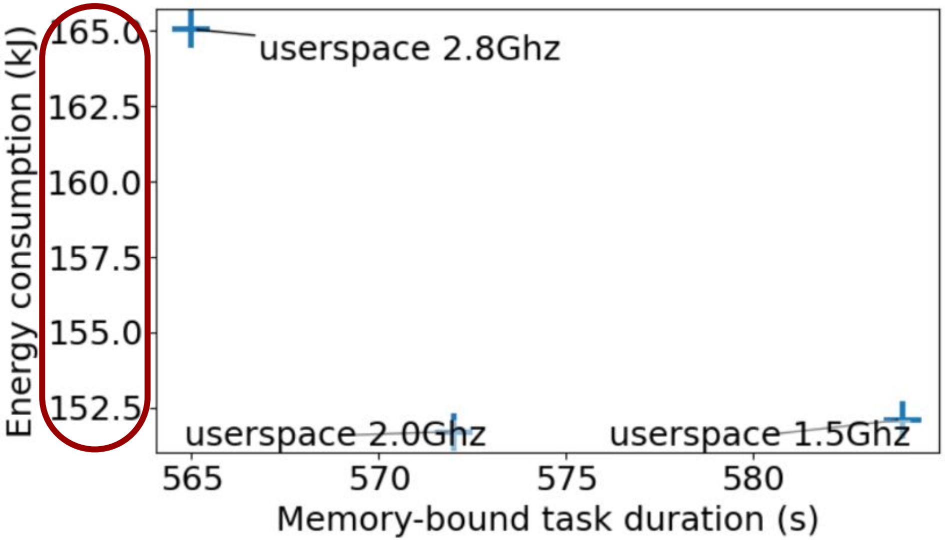
Experimental Evaluation: P-states for the memory-bound workload

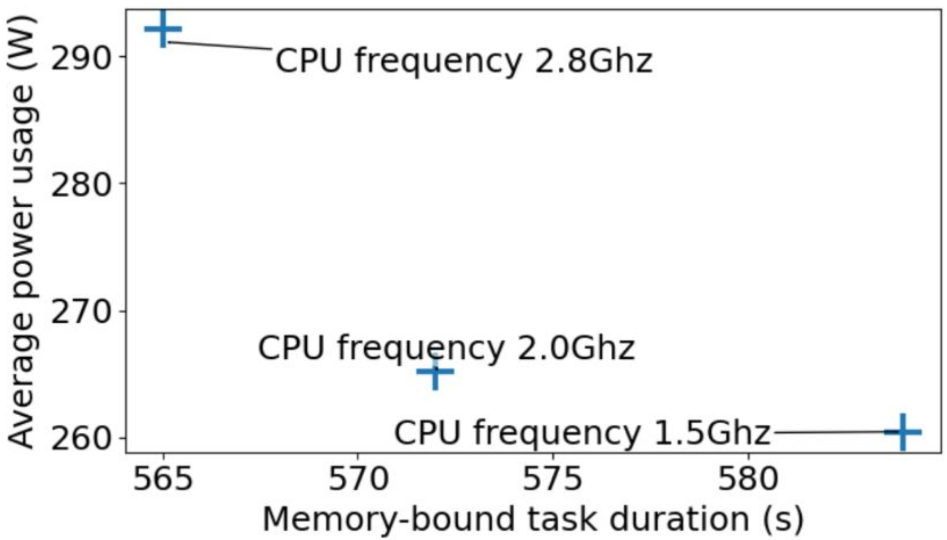


- On memory-bound tasks, reducing the CPU frequency leads to a slightly lower performance (-4%) and to a reduced total energy consumption (-9%)

DVFS

Experimental Evaluation: P-states for the memory-bound workload

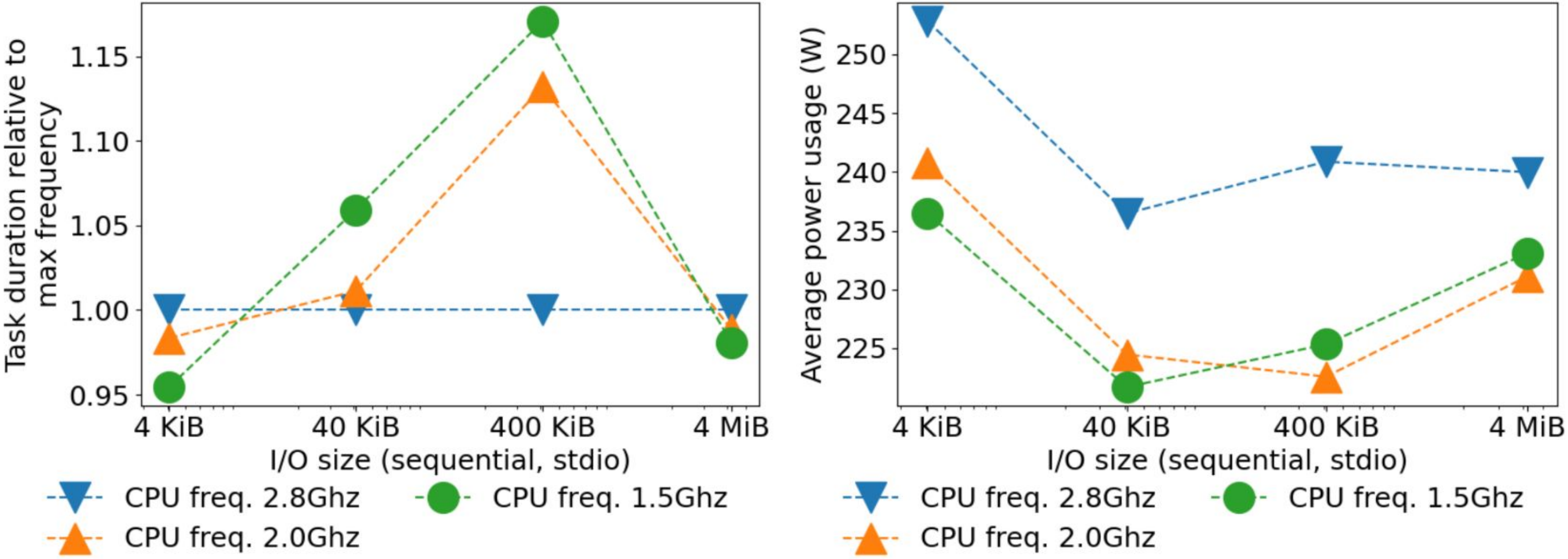




- On memory-bound tasks, reducing the CPU frequency leads to a slightly lower performance (-4%) and to a reduced total energy consumption (-9%)

DVFS

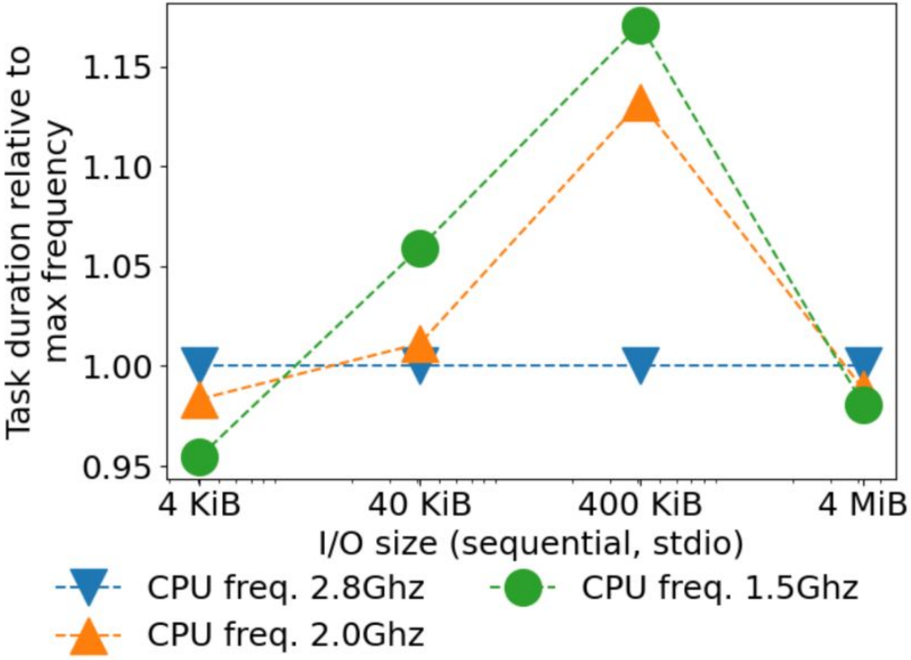
Experimental Evaluation: P-states for the buffered I/O workload

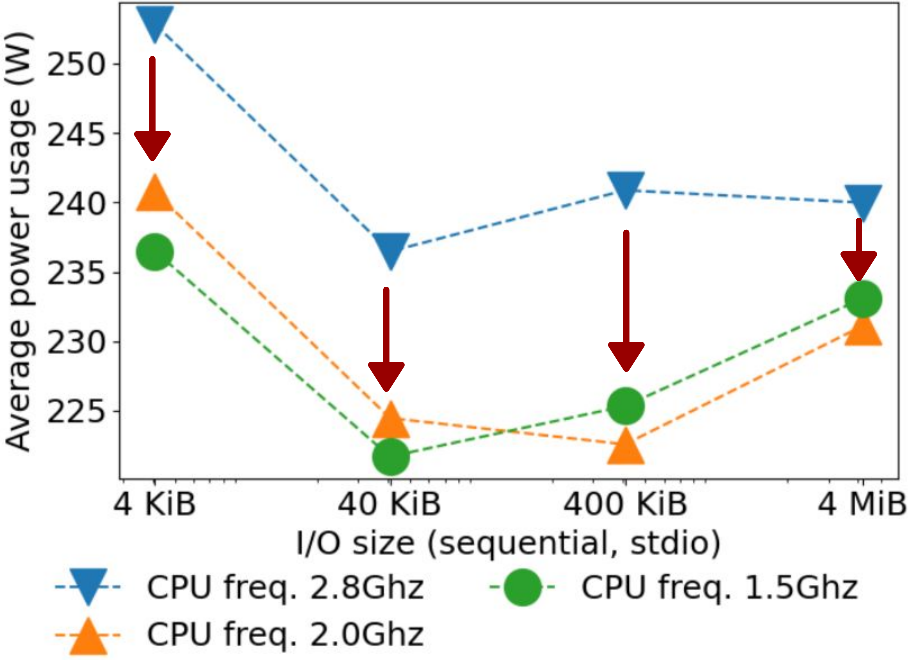


- On buffered I/O tasks, reducing the CPU frequency leads to a reduced power usage (up to -7%) at a variable performance cost (from none up to +17% task duration)

DVFS

Experimental Evaluation: P-states for the buffered I/O workload

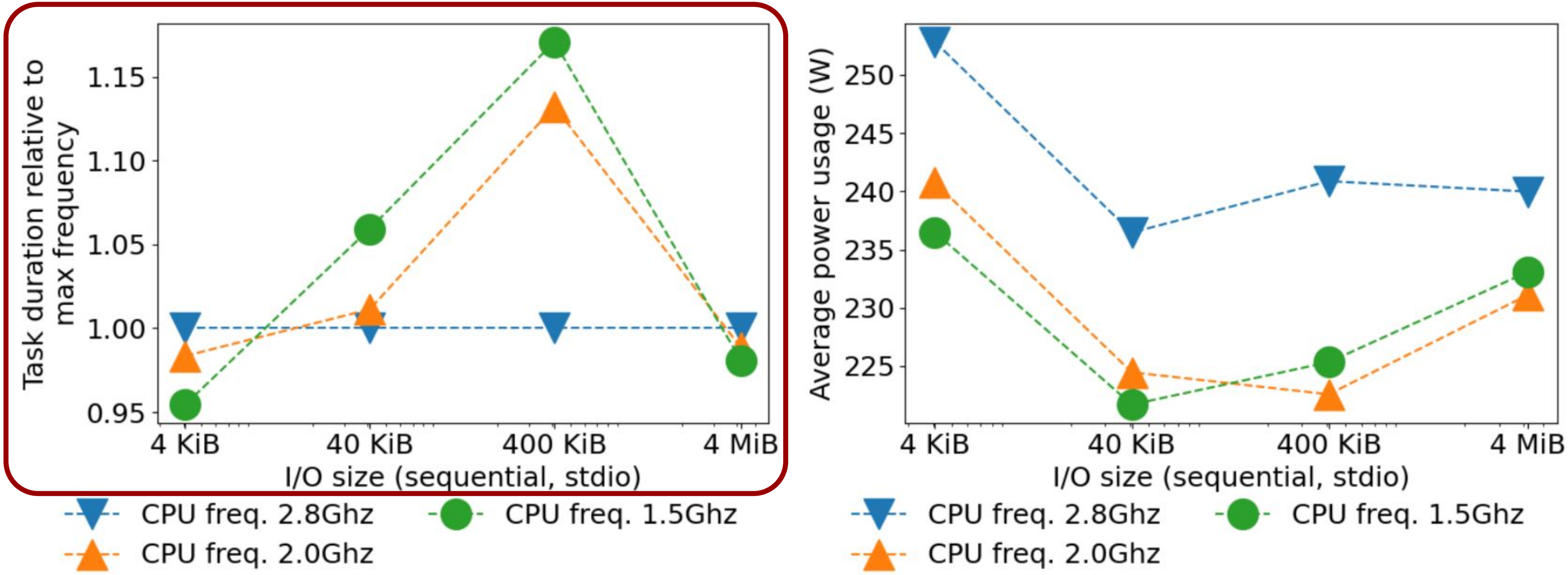




- On buffered I/O tasks, reducing the CPU frequency leads to a reduced power usage (up to -7%) at a variable performance cost (from none up to +17% task duration)

DVFS

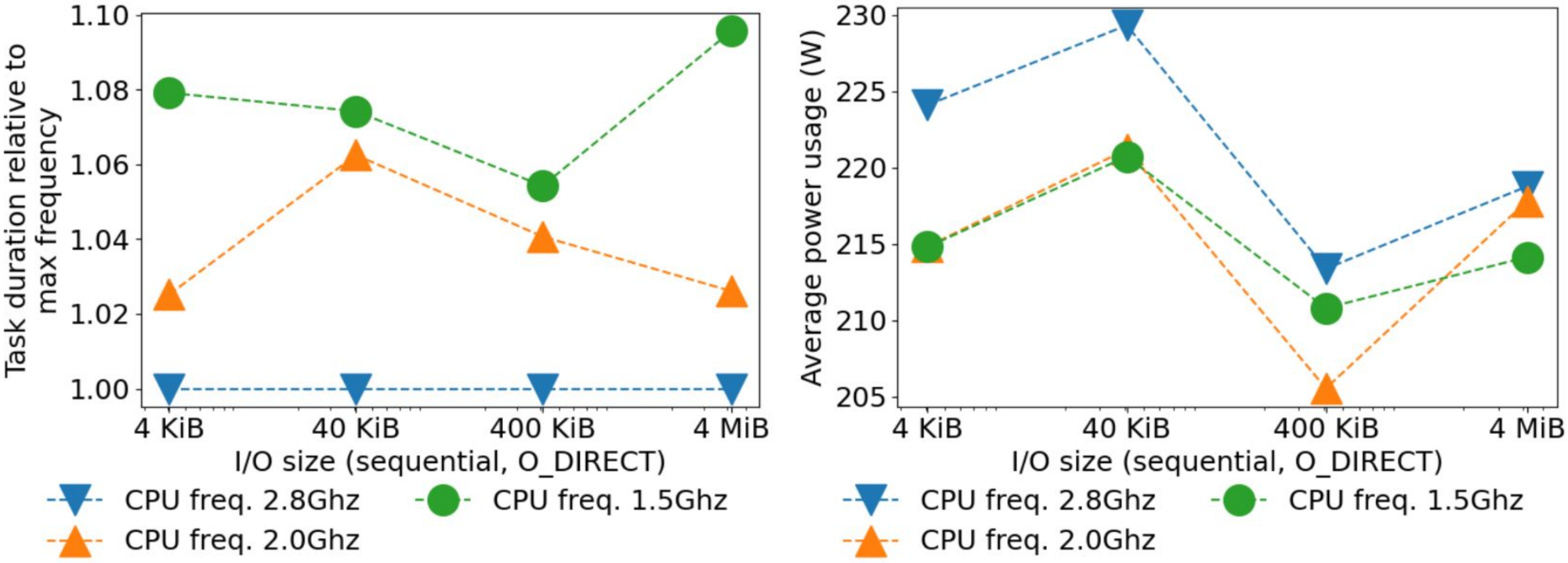
Experimental Evaluation: P-states for the buffered I/O workload



- On buffered I/O tasks, reducing the CPU frequency leads to a reduced power usage (up to -7%) at a variable performance cost (from none up to +17% task duration)

DVFS

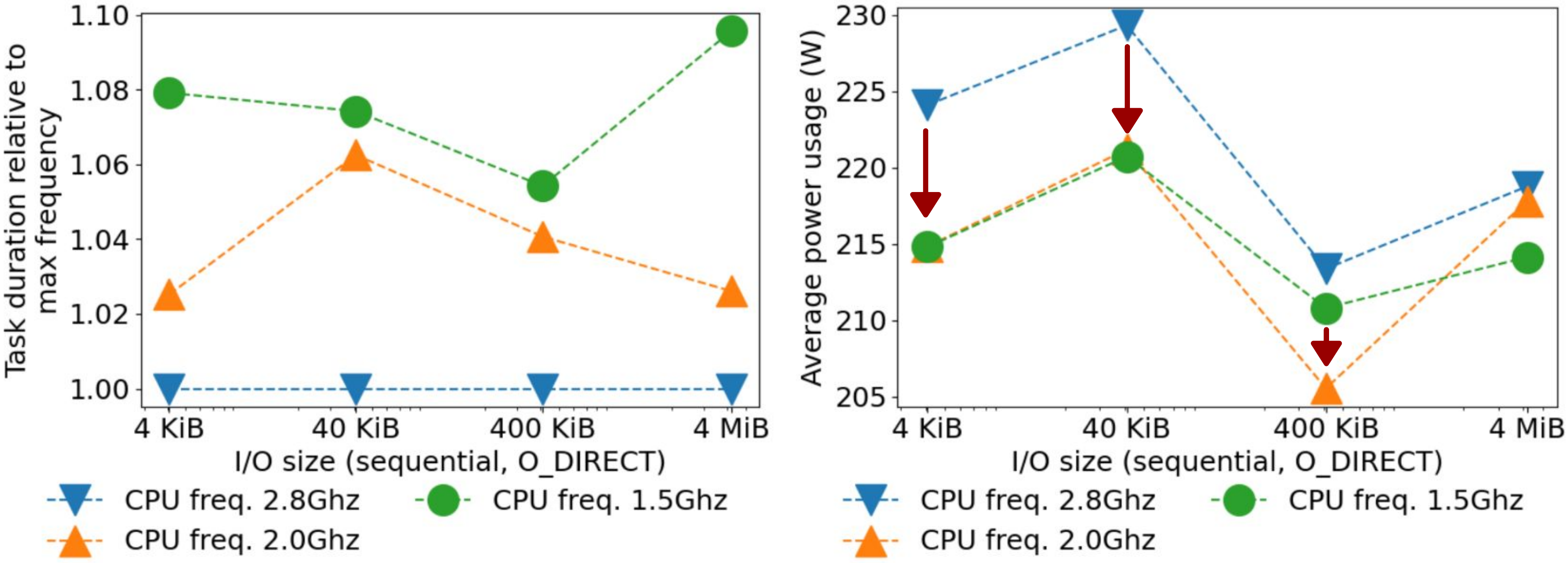
Experimental Evaluation: P-states for the direct I/O workload



- On direct I/O tasks, reducing the CPU frequency leads to a slightly reduced power usage (up to -4%) and a lower performance (up to +9% task duration)

DVFS

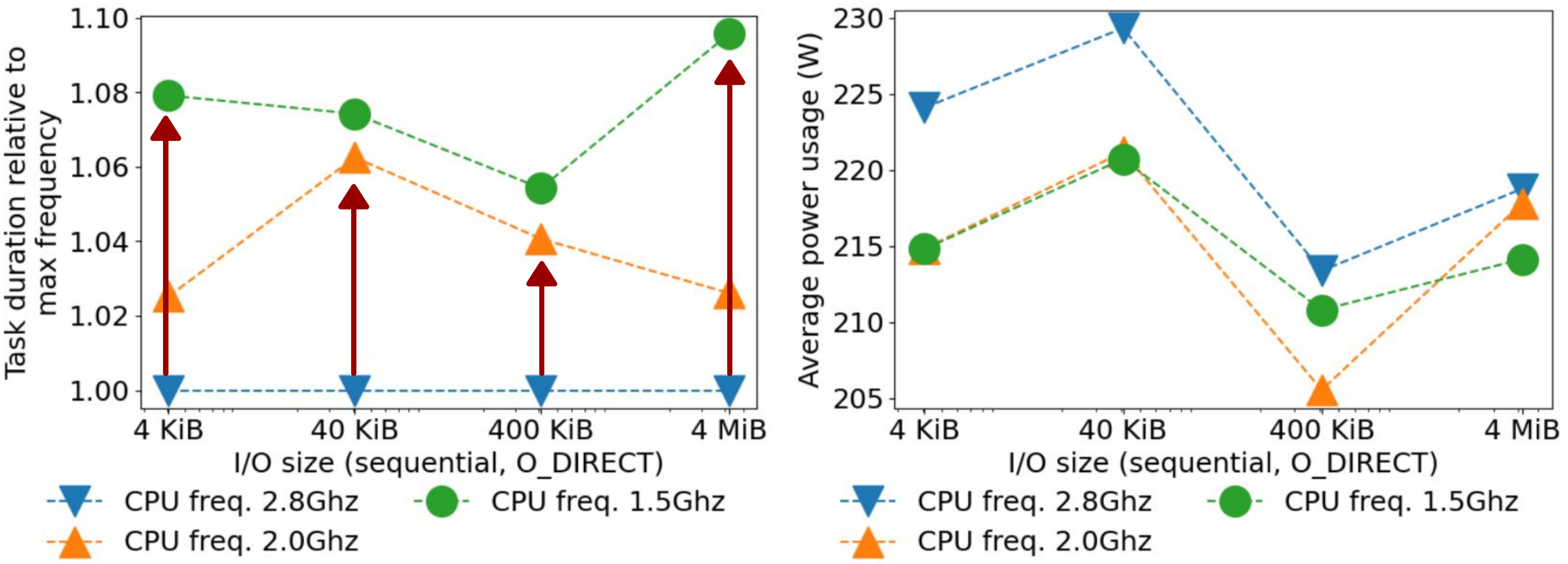
Experimental Evaluation: P-states for the direct I/O workload



- On direct I/O tasks, reducing the CPU frequency leads to a slightly reduced power usage (up to -4%) and a lower performance (up to +9% task duration)

DVFS

Experimental Evaluation: P-states for the direct I/O workload

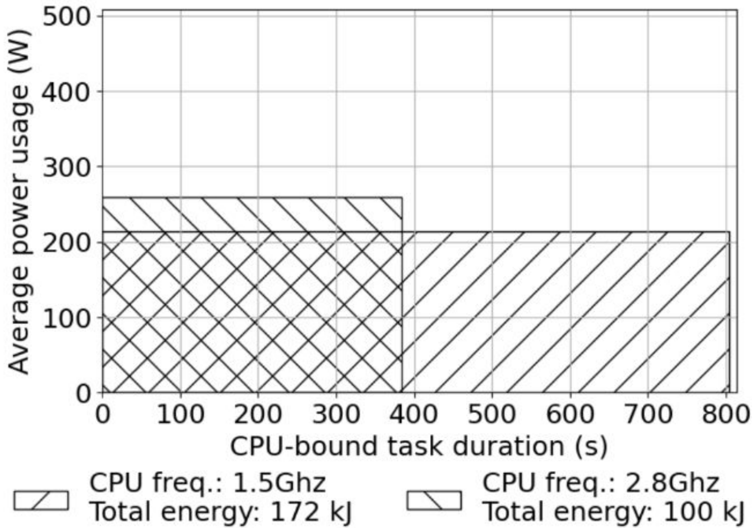


- On direct I/O tasks, reducing the CPU frequency leads to a slightly reduced power usage (up to -4%) and a lower performance (up to +9% task duration)

DVFS

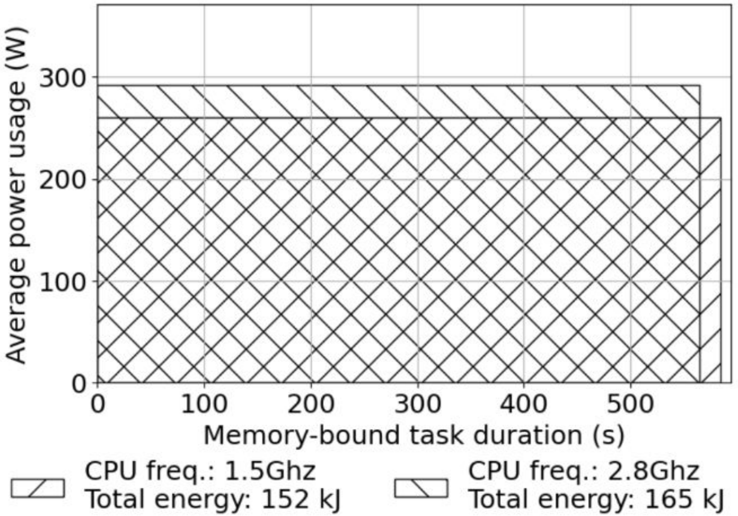
Key Takeaways

- On a CPU-bound workload, the reduced power usage is not enough to compensate for the increased duration



DVFS
Key Takeaways

- On a CPU-bound workload, the reduced power usage is not enough to compensate for the increased duration.
- On a Memory-bound workload, the reduced power usage is able to compensate for the increased duration, enabling energy optimization.



DVFS
Key Takeaways

- On a CPU-bound workload, the reduced power usage is not enough to compensate for the increased duration.
- On a Memory-bound workload, the reduced power usage is able to compensate for the increased duration, enabling energy optimization.
- On I/O workloads, we constantly observe a lower power usage with lower CPU frequencies, but also a variable performance loss.

DVFS

Key Takeaways

- On a CPU-bound workload, the reduced power usage is not enough to compensate for the increased duration.
- On a Memory-bound workload, the reduced power usage is able to compensate for the increased duration, enabling energy optimization.
- On I/O workloads, we constantly observe a lower power usage with lower CPU frequencies, but also a variable performance loss.
- Overall, while we were limited to a single CPU model and PFS in this study, we have demonstrated that there are I/O energy optimization opportunities with DVFS

Conclusion and Future Works

- GrIOt with one graph per file enables I/O modeling and prediction with a similar or better prediction accuracy than state of the art. It also has less overhead and a lower memory footprint.

Conclusion and Future Works

- GrIOt with one graph per file enables I/O modeling and prediction with a similar or better prediction accuracy than state of the art. It also has less overhead and a lower memory footprint.
- We have demonstrated that there were energy optimization opportunities using DVFS.

Conclusion and Future Works

- GrIOt with one graph per file enables I/O modeling and prediction with a similar or better prediction accuracy than state of the art. It also has less overhead and a lower memory footprint.
- We have demonstrated that there were energy optimization opportunities using DVFS.
- Future works:
  - Extending our studies on DVFS to more software and hardware resources.
  - Extending our study on DVFS to provide an I/O energy predictive model.
  - Extending GrIOt to enable federating models made on multiple compute nodes into a single application model.
  - Using GrIOt to optimize I/O energy with DVFS.