Energy-Aware Runtime Resource Harmonizer for Co-running Applications

Vanshika Jain[1], Varun Parashar[1], Vivek Kumar[1], Chiranjib Sur[2]

1 IIIT Delhi, New Delhi, India
2 Shell India Markets Pvt. Ltd, India

Outline

- Introduction
- Motivation
- Existing Approaches
- Contributions
- Implementation
- Results
- Conclusion

Resource Utilization in the Exascale Era

Increasing number of sockets and cores per node

Rank of Top500 (November 2025)	Sockets Per Node	Cores Per Node
1	**4**	96
2	1	64
3	2	104
4	**4**	**288**
5	2	96

1. https://top500.org/lists/top500/

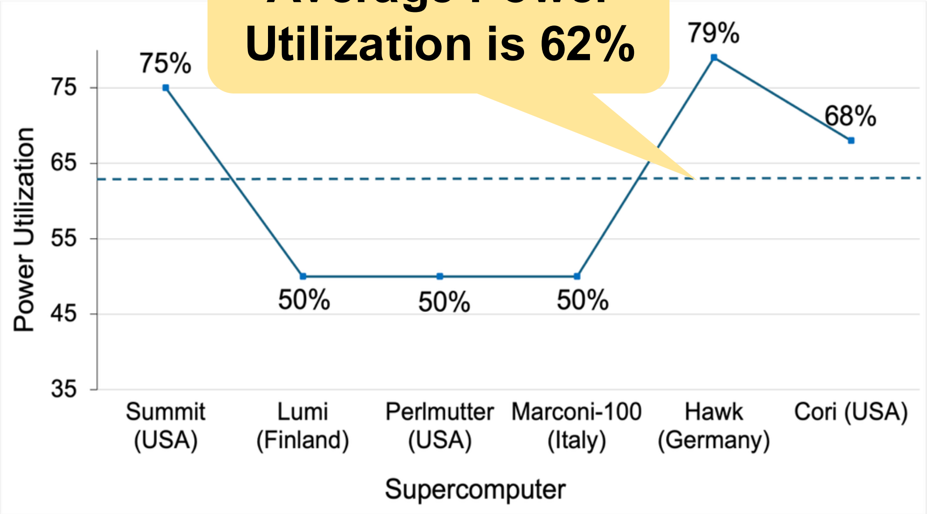
Resource Utilization in the Exascale Era

Increasing number of sockets and cores per node

Rank of Top500 (November 2025)	Sockets Per Node	Cores Per Node
1	**4**	96
2	1	64
3	2	104
4	**4**	**288**
5	2	96

Power usage at supercomputers

Average Power Utilization is 62%



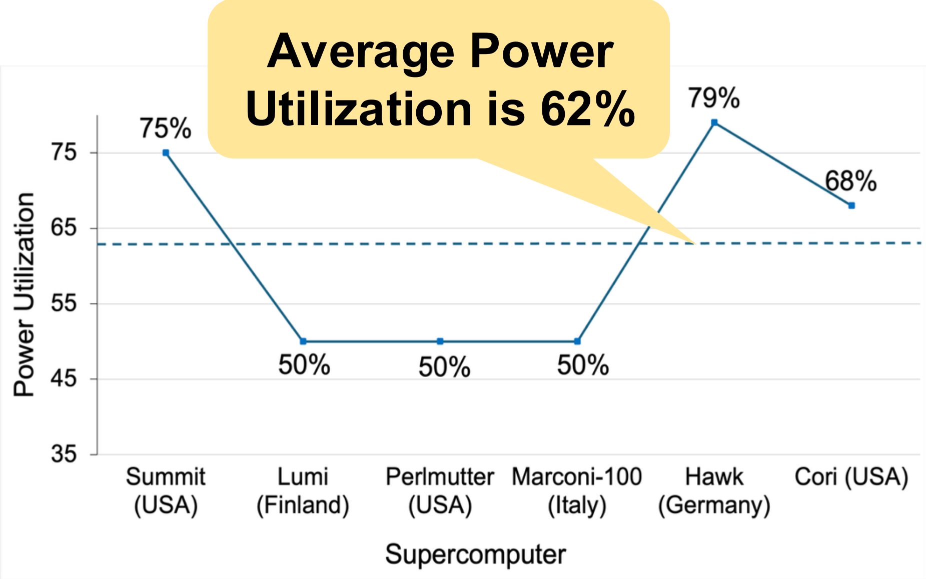
1. https://top500.org/lists/top500/ 2. Patki et.al. [ICS2025]

Resource Utilization in the Exascale Era

Increasing number of sockets and cores per node

Rank of Top500 (November 2025)	Sockets Per Node	Cores Per Node
1	**4**	96
2	1	64
3	2	104
4	**4**	**288**
5	2	96

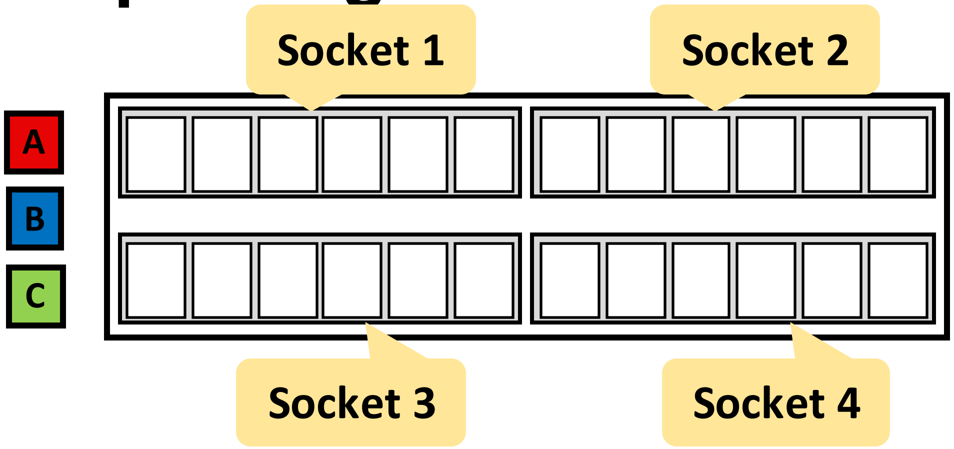
Power usage at supercomputers


Average Power Utilization is 62%

It is critical to improve resource utilization for achieving energy efficiency

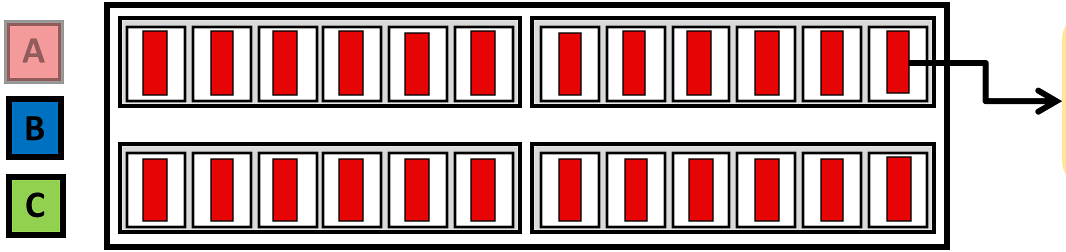
1. https://top500.org/lists/top500/ 2. Patki et.al. [ICS2025]

Improving Resource Utilization via Co-location



Applications A, B & C to be executed on a quad-socket system

Improving Resource Utilization via Co-location

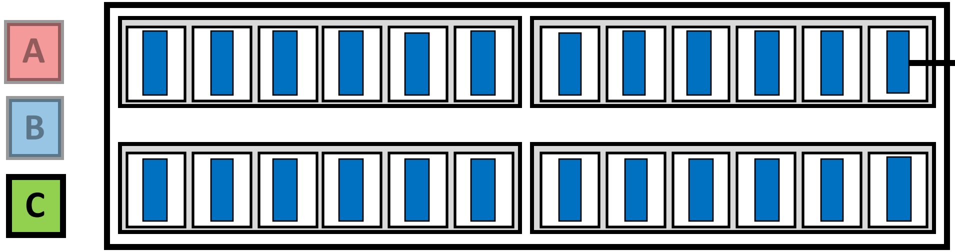


Threads of Application A

Batch Execution

Applications B & C waiting for the CPUs

Improving Resource Utilization via Co-location

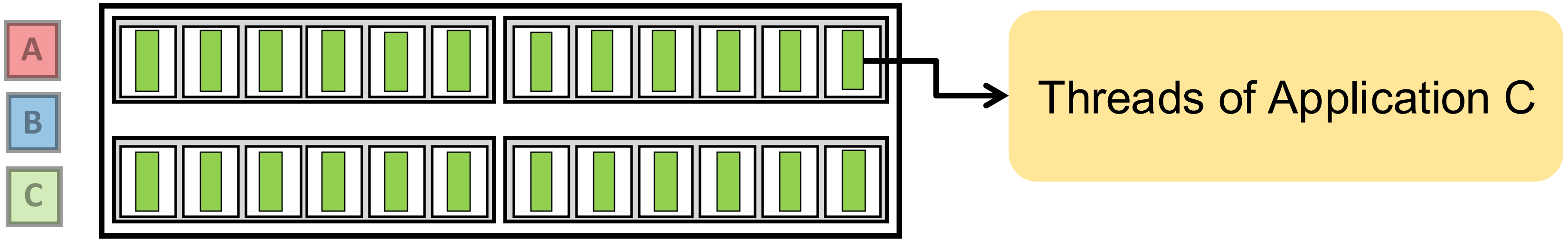


Threads of Application B

Batch Execution

Application C
waiting for the CPUs

Improving Resource Utilization via Co-location

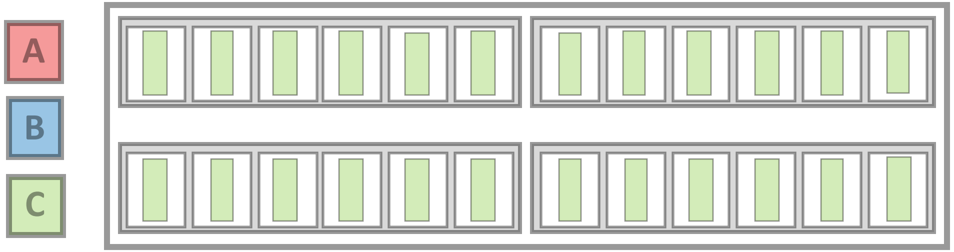


Threads of Application C

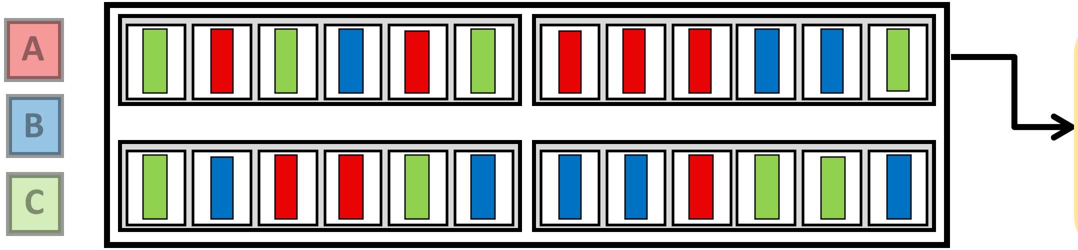
Batch Execution

Each application completed their execution one by one

Improving Resource Utilization via Co-location



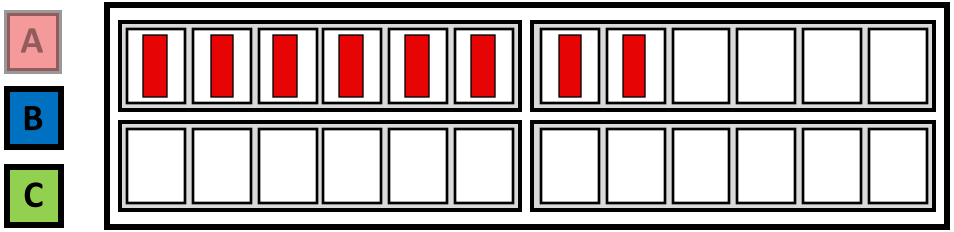
Batch Execution



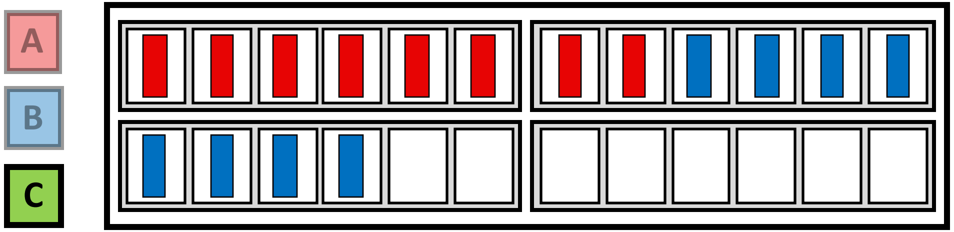
Co-running Execution

Threads of Application
A, B & C
running in parallel

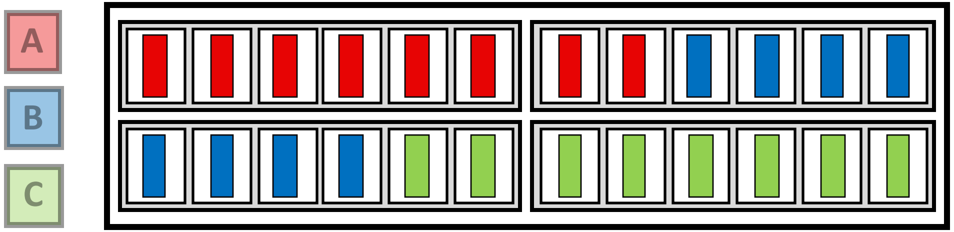
Improving Resource Utilization via Co-location



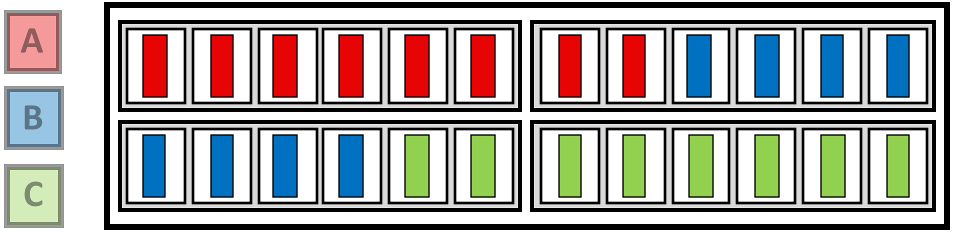
Improving Resource Utilization via Co-location



Improving Resource Utilization via Co-location

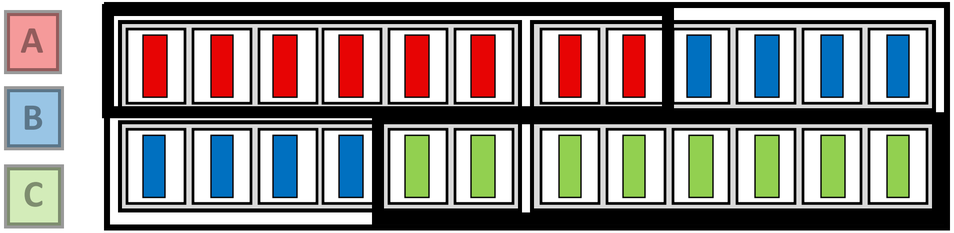


Improving Resource Utilization via Co-location



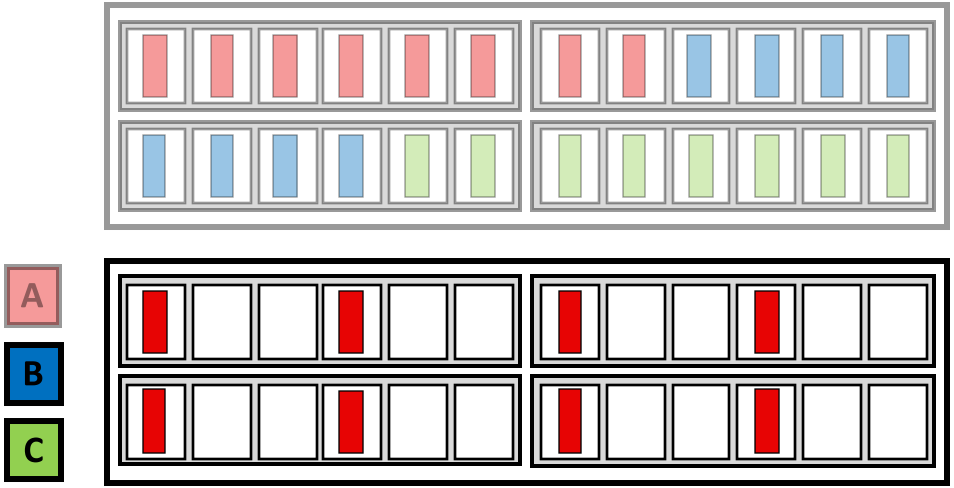
Type: Block-Cyclic

Improving Resource Utilization via Co-location

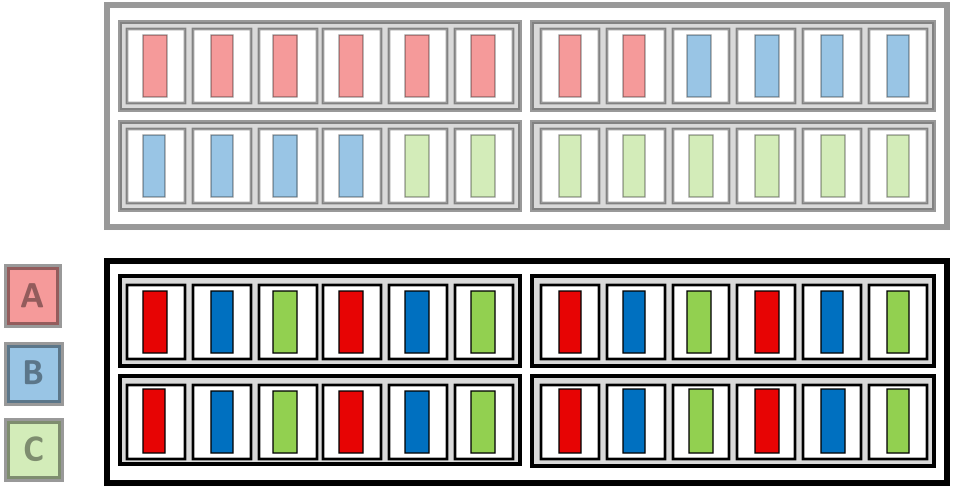


Type: Block-Cyclic

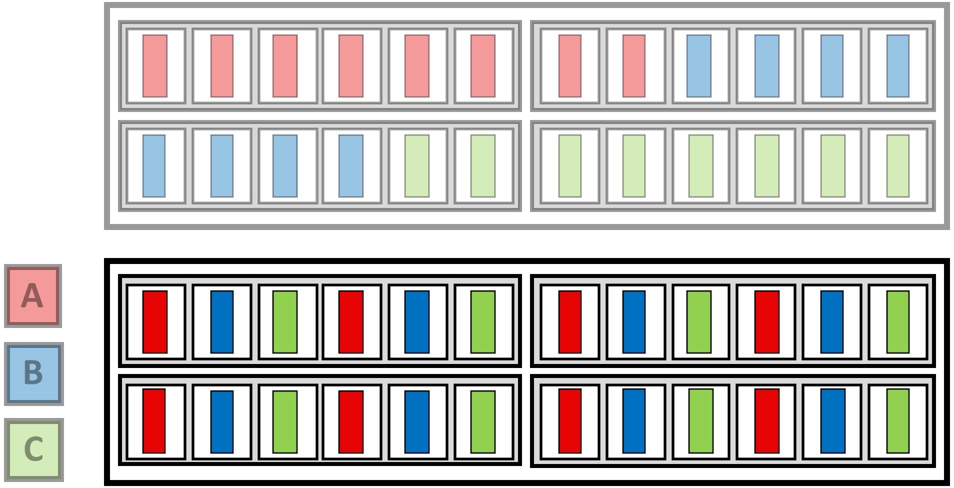
Improving Resource Utilization via Co-location



Improving Resource Utilization via Co-location

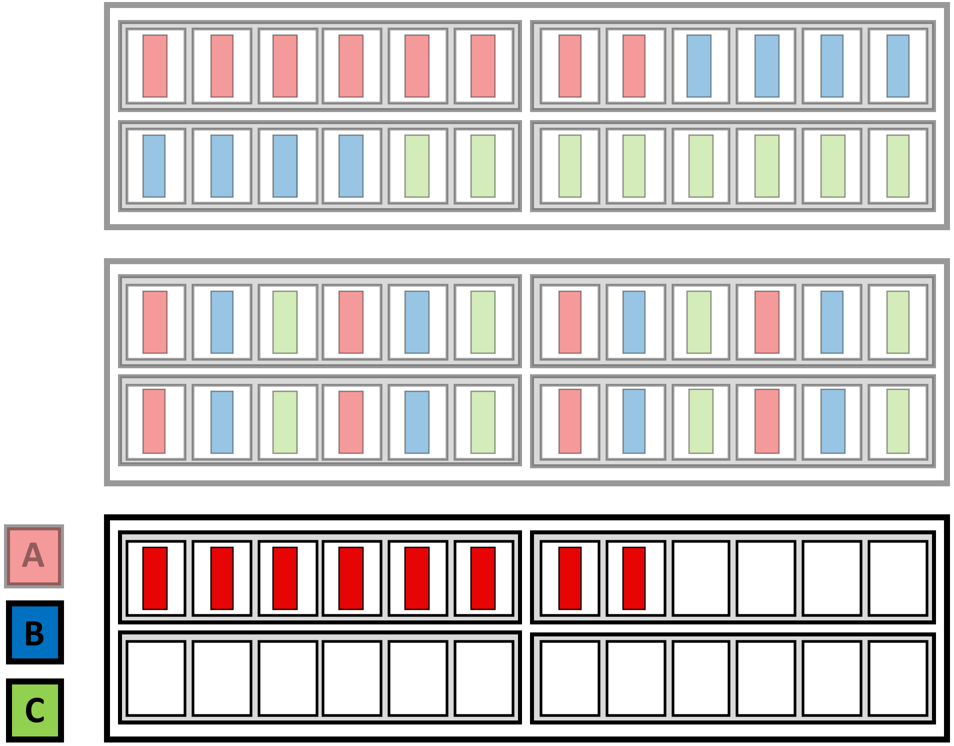


Improving Resource Utilization via Co-location

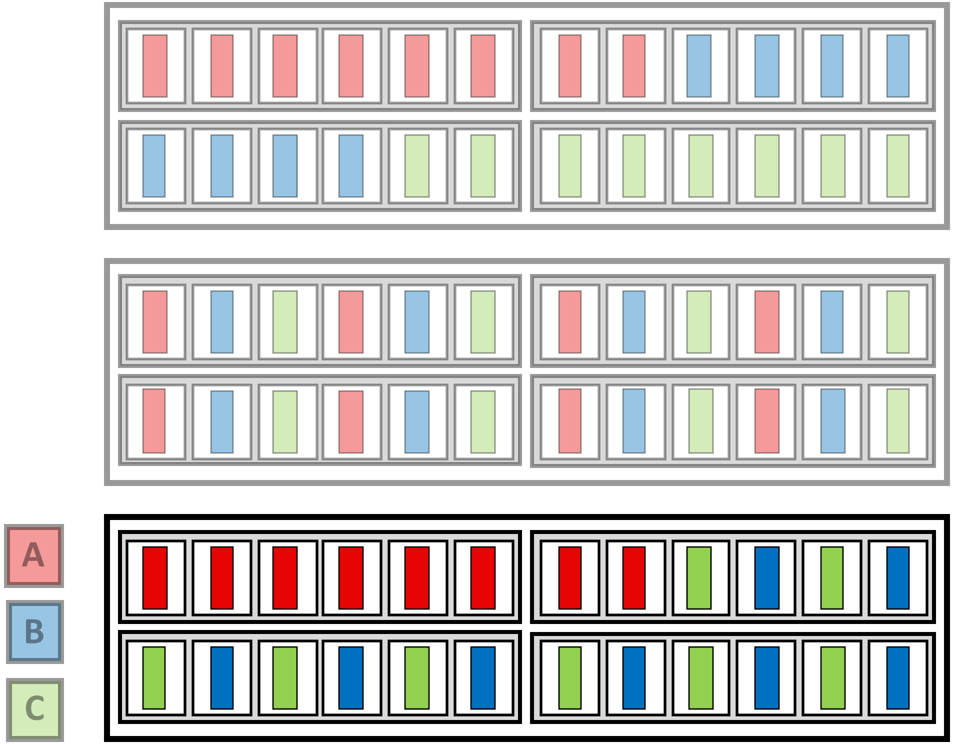


Type: Interleaved

Improving Resource Utilization via Co-location

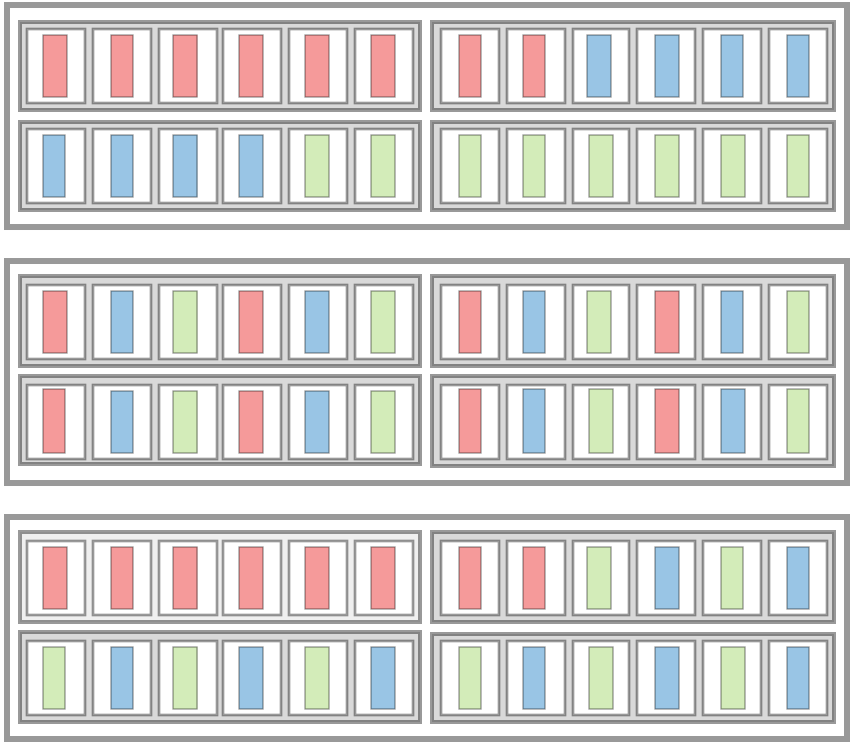


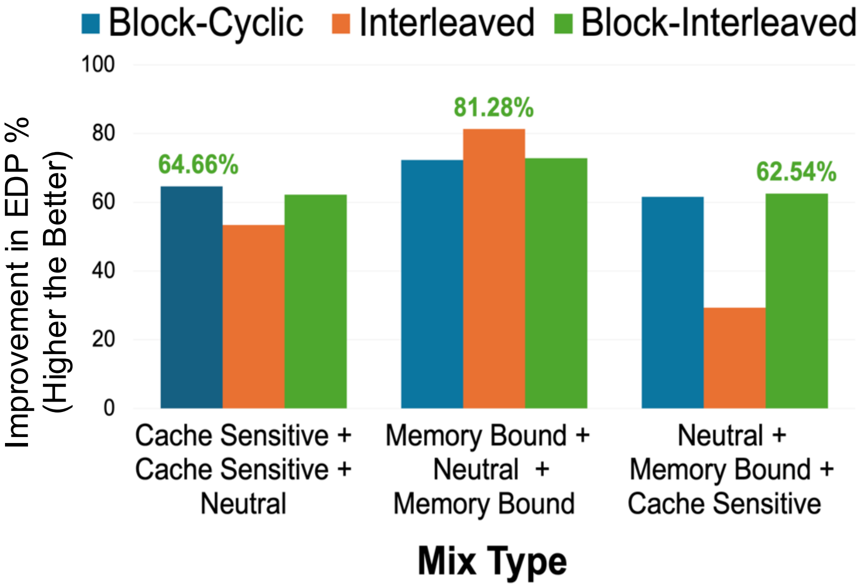
Improving Resource Utilization via Co-location



Type: Block-Interleaved

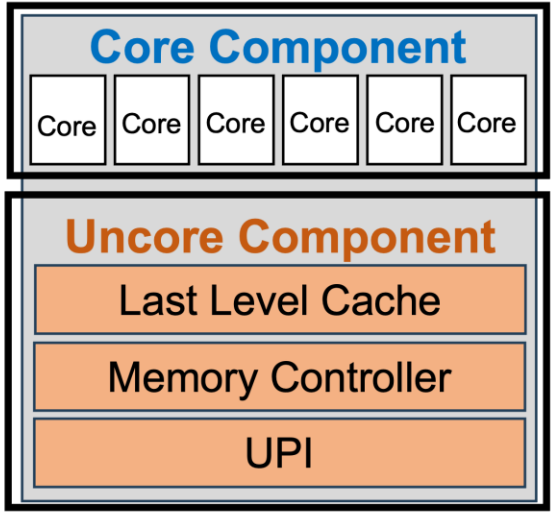
Improving Resource Utilization via Co-location



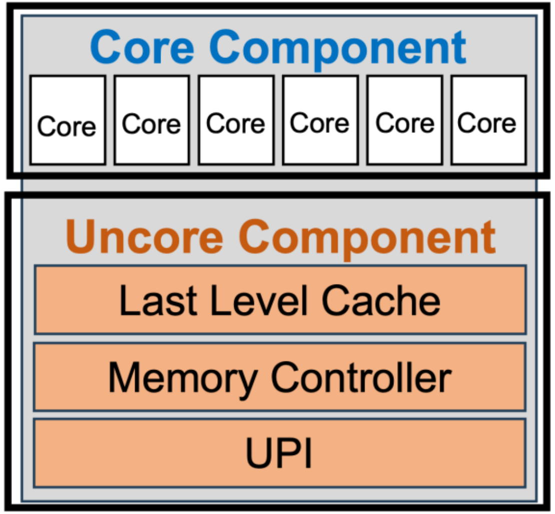

Block-Cyclic
Interleaved
Block-Interleaved
Improvement in EDP %
(Higher the Better)
100
80
60
40
20
0
64.66%
81.28%
62.54%
Cache Sensitive + Cache Sensitive + Neutral
Memory Bound + Neutral + Memory Bound
Neutral + Memory Bound + Cache Sensitive
Mix Type

Choosing optimal thread placement over Batch execution improves EDP by up to 81%

Achieving Energy Efficiency on Multicores

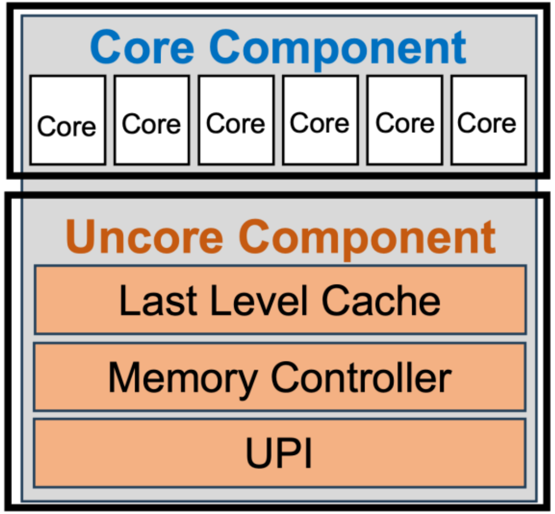


Achieving Energy Efficiency on Multicores



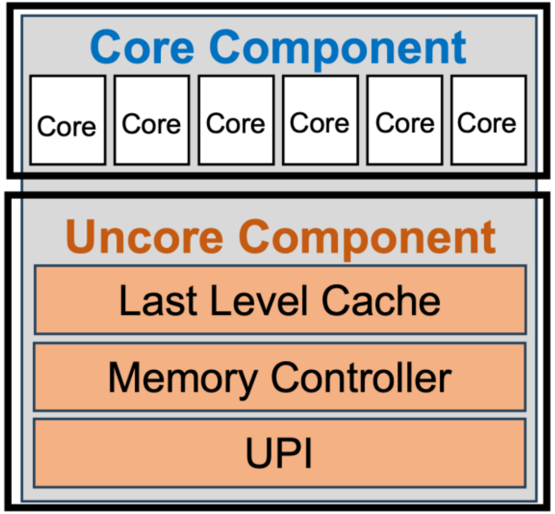
- Dynamic Voltage and Frequency Scaling (DVFS)
  - Core-level

Achieving Energy Efficiency on Multicores



- Dynamic Voltage and Frequency Scaling (DVFS)
  - Core-level
- Uncore Frequency Scaling (UFS)
  - Socket-level

Achieving Energy Efficiency on Multicores



Core Component

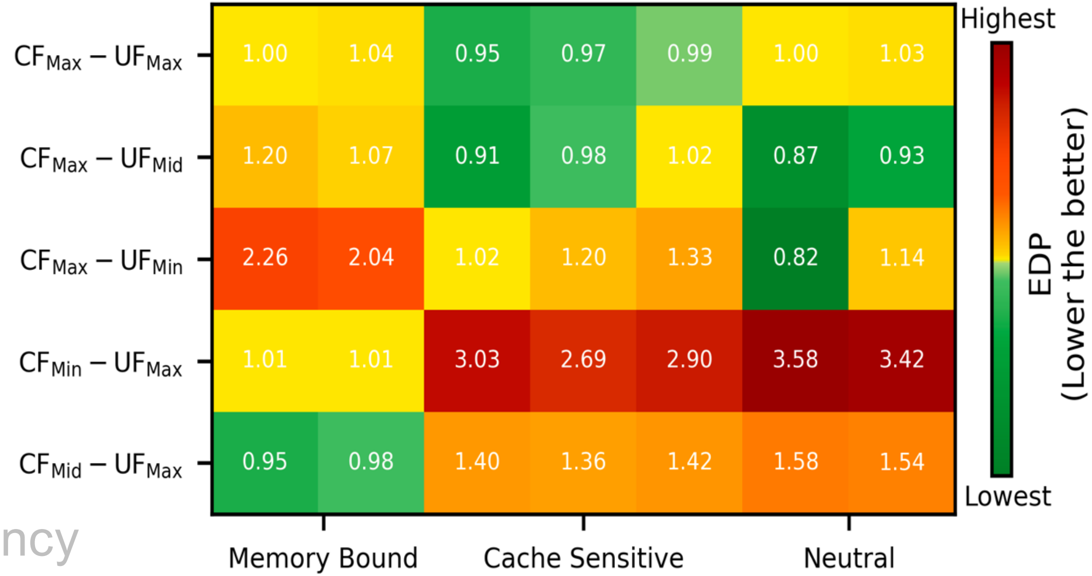
Core | Core | Core | Core | Core | Core

Uncore Component

Last Level Cache

Memory Controller

UPI

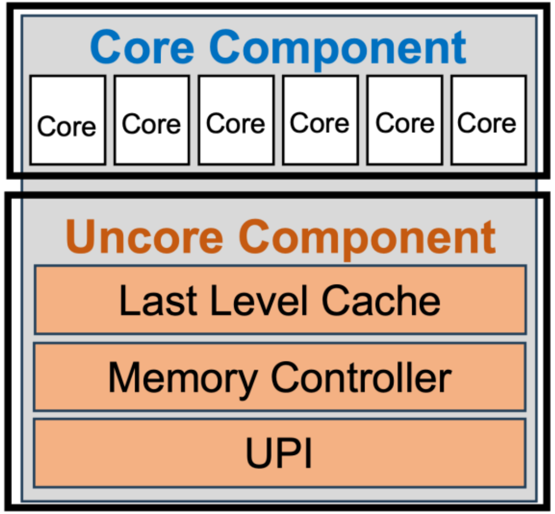


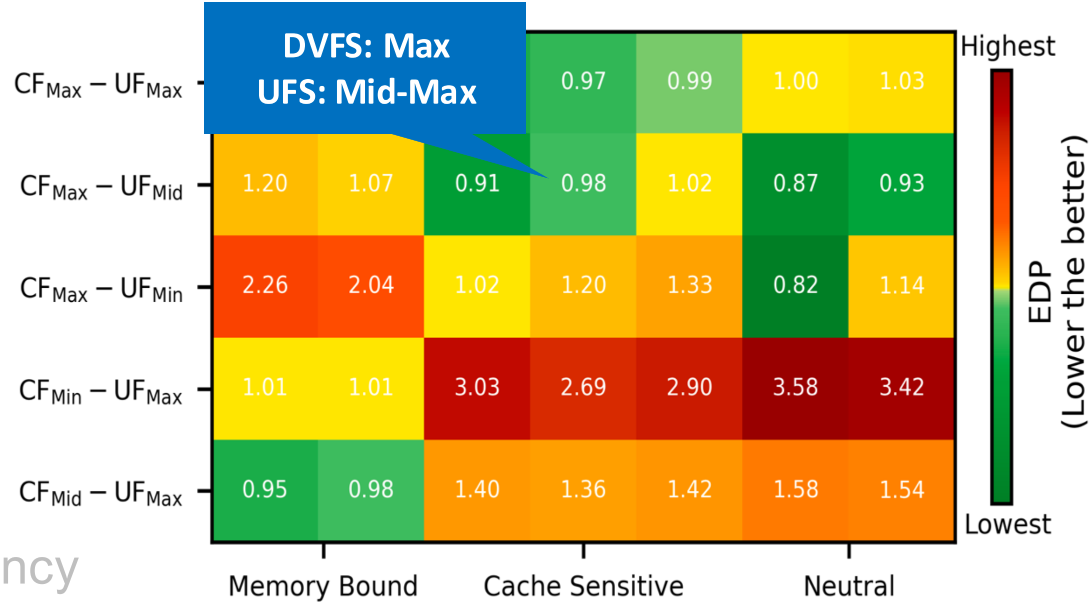
	Memory Bound		Cache Sensitive			Neutral	
$CF_{Max} - UF_{Max}$	1.00	1.04	0.95	0.97	0.99	1.00	1.03
$CF_{Max} - UF_{Mid}$	1.20	1.07	0.91	0.98	1.02	0.87	0.93
$CF_{Max} - UF_{Min}$	2.26	2.04	1.02	1.20	1.33	0.82	1.14
$CF_{Min} - UF_{Max}$	1.01	1.01	3.03	2.69	2.90	3.58	3.42
$CF_{Mid} - UF_{Max}$	0.95	0.98	1.40	1.36	1.42	1.58	1.54

- Dynamic Voltage and Frequency Scaling (DVFS)
  - Core-level
- Uncore Frequency Scaling (UFS)
  - Socket-level

Heatmap represents the change in EDP with a particular combination of core-uncore frequency relative to default settings

Achieving Energy Efficiency on Multicores

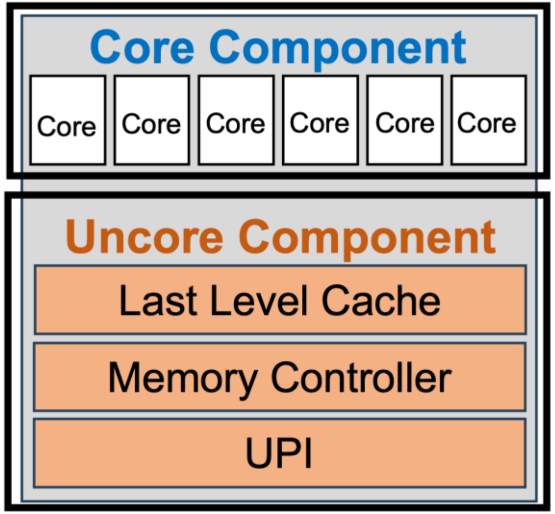


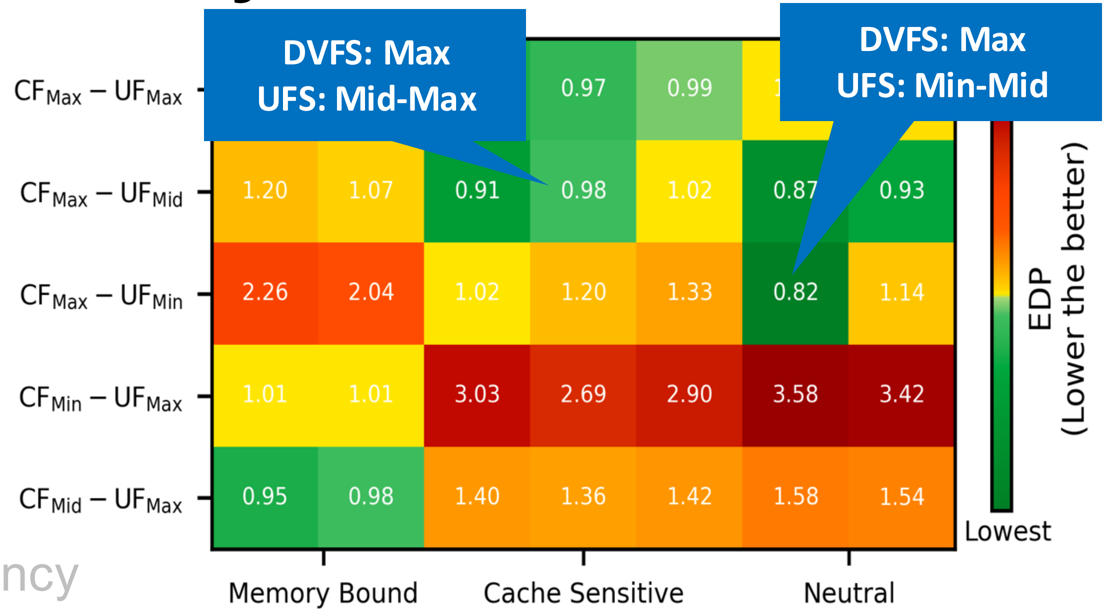


- Dynamic Voltage and Frequency Scaling (DVFS)
  - Core-level
- Uncore Frequency Scaling (UFS)
  - Socket-level

Heatmap represents the change in EDP with a particular combination of core-uncore frequency relative to default settings

Achieving Energy Efficiency on Multicores



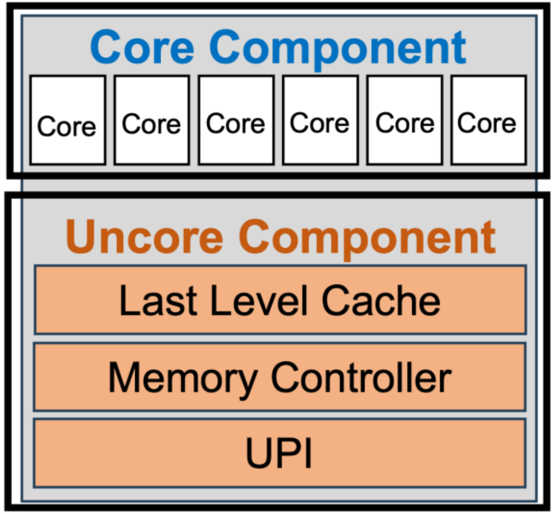


	Memory Bound			Cache Sensitive		Neutral	
$CF_{Max} - UF_{Max}$				0.97	0.99	1	
$CF_{Max} - UF_{Mid}$	1.20	1.07	0.91	0.98	1.02	0.87	0.93
$CF_{Max} - UF_{Min}$	2.26	2.04	1.02	1.20	1.33	0.82	1.14
$CF_{Min} - UF_{Max}$	1.01	1.01	3.03	2.69	2.90	3.58	3.42
$CF_{Mid} - UF_{Max}$	0.95	0.98	1.40	1.36	1.42	1.58	1.54

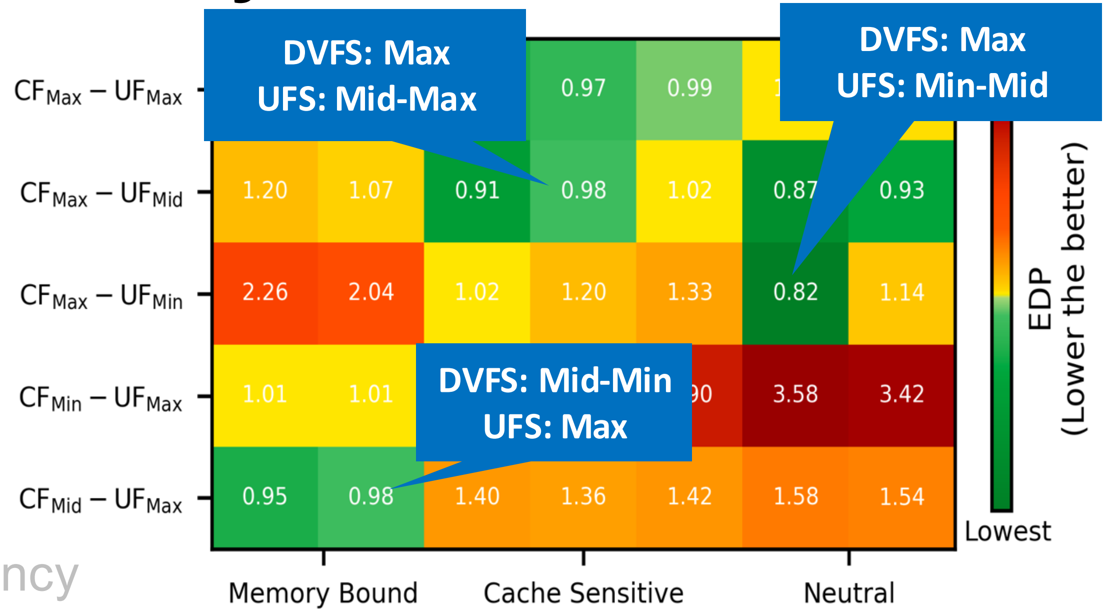
Heatmap represents the change in EDP with a particular combination of core-uncore frequency relative to default settings

- Dynamic Voltage and Frequency Scaling (DVFS)
  - ○ Core-level
- Uncore Frequency Scaling (UFS)
  - ○ Socket-level

Achieving Energy Efficiency on Multicores



- Dynamic Voltage and Frequency Scaling (DVFS)
  - Core-level
- Uncore Frequency Scaling (UFS)
  - Socket-level



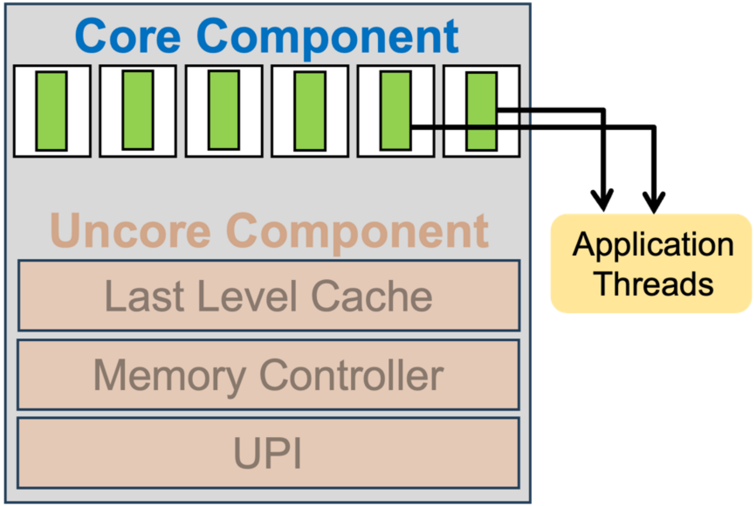
DVFS: Max
UFS: Mid-Max

DVFS: Max
UFS: Min-Mid

DVFS: Mid-Min
UFS: Max

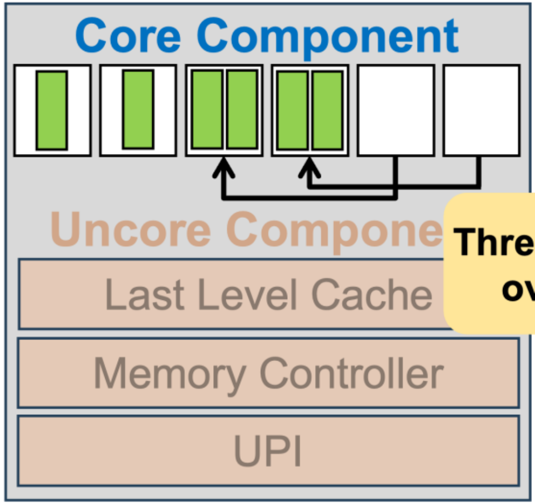
Heatmap represents the change in EDP with a particular combination of core-uncore frequency relative to default settings

Achieving Energy Efficiency on Multicores



- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation

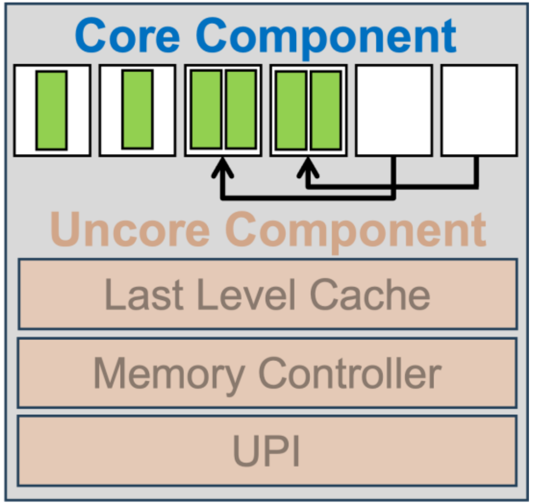
Achieving Energy Efficiency on Multicores

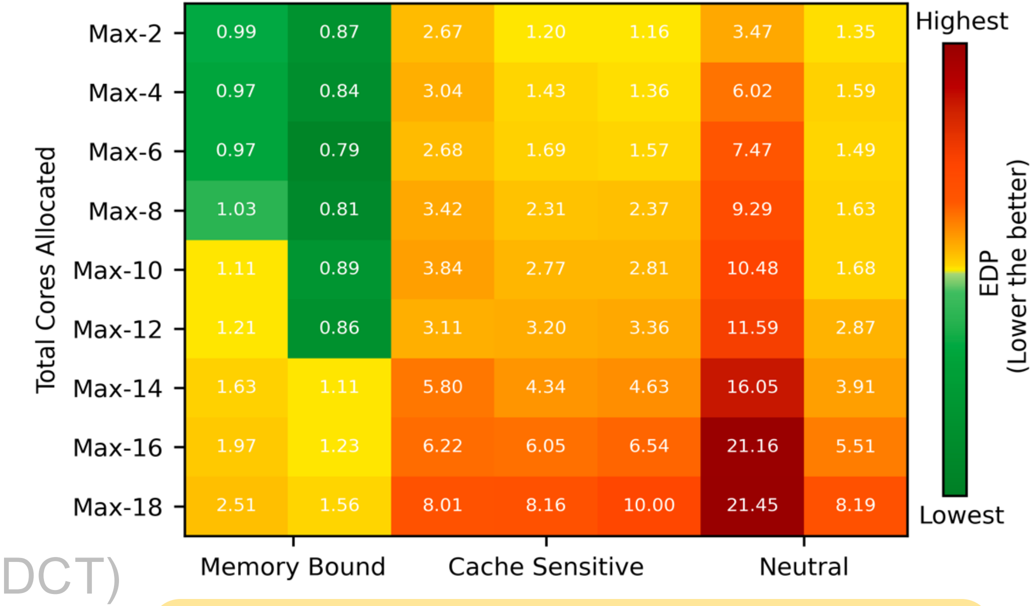


Threads packed over cores

- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation
  - Thread packing and unpacking technique provides runtime independence

Achieving Energy Efficiency on Multicores



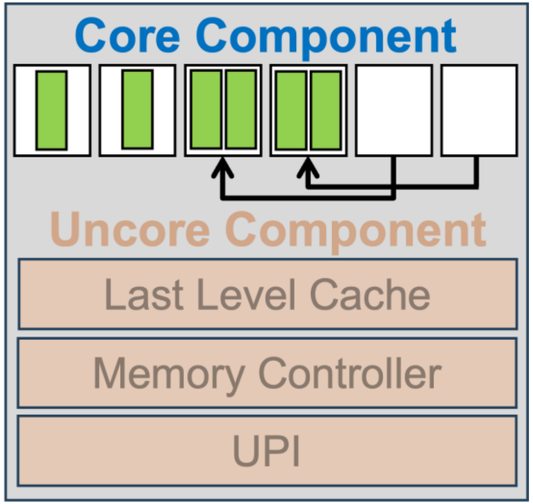


Max-2	0.99	0.87	2.67	1.20	1.16	3.47	1.35
Max-4	0.97	0.84	3.04	1.43	1.36	6.02	1.59
Max-6	0.97	0.79	2.68	1.69	1.57	7.47	1.49
Max-8	1.03	0.81	3.42	2.31	2.37	9.29	1.63
Max-10	1.11	0.89	3.84	2.77	2.81	10.48	1.68
Max-12	1.21	0.86	3.11	3.20	3.36	11.59	2.87
Max-14	1.63	1.11	5.80	4.34	4.63	16.05	3.91
Max-16	1.97	1.23	6.22	6.05	6.54	21.16	5.51
Max-18	2.51	1.56	8.01	8.16	10.00	21.45	8.19

- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation
  - Thread packing and unpacking technique provides runtime independence

Heatmap represents the change in EDP by changing the core count relative to default with maximum core allocation

Achieving Energy Efficiency on Multicores



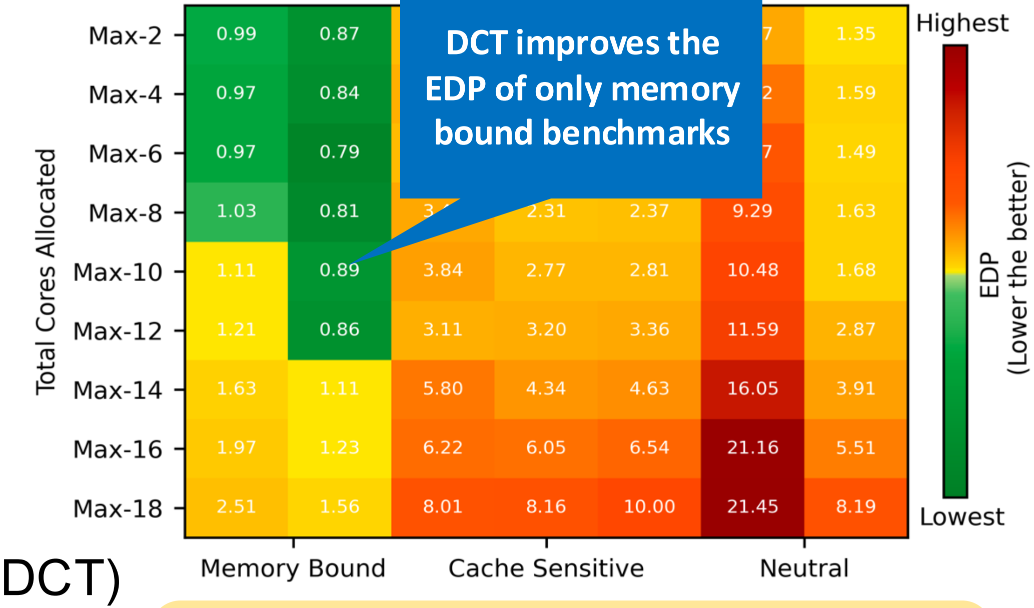
Core Component

Uncore Component

Last Level Cache

Memory Controller

UPI



DCT improves the EDP of only memory bound benchmarks

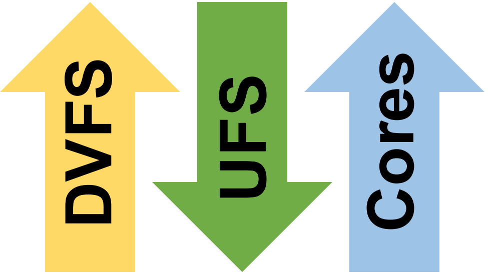
- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation
  - Thread packing and unpacking technique provides runtime independence

Heatmap represents the change in EDP by changing the core count relative to default with maximum core allocation

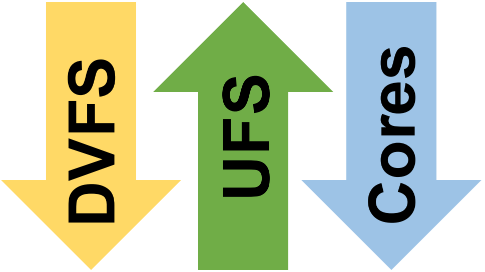
Insights

- Choosing optimal thread placement improves resource utilization for co-running applications
- There is a strong correlation between application behavior and resource requirement

Cache-Sensitive and **Neutral**



Memory-bound



Existing Approaches for Co-Running Applications

Categories of Resource Management Techniques	DCT only	DVFS only	UFS only	DVFS+ UFS	DCT+ DVFS+ UFS
Thread Placement for contention reduction	✓	✗	✗	✗	✗
Runtime Oblivious	✓	✓	✗	✗	✗
Model Free	✓	✓	✓	✓	✗

Existing Approaches for Co-Running Applications

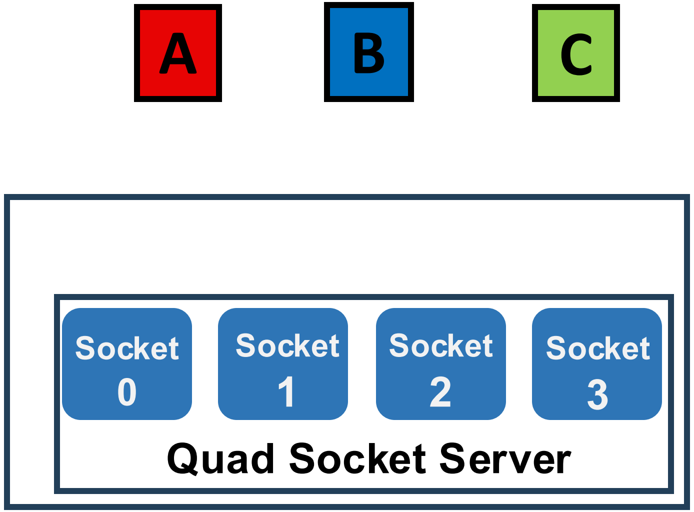
Our Focus

Categories of Resource Management Techniques	DCT only	DVFS only	UFS only	UFS	DCT+ DVFS+ UFS
Thread Placement for contention reduction	✓	✗	✗	✗	✗
Runtime Oblivious	✓	✓	✗	✗	✗
Model Free	✓	✓	✓	✓	✗

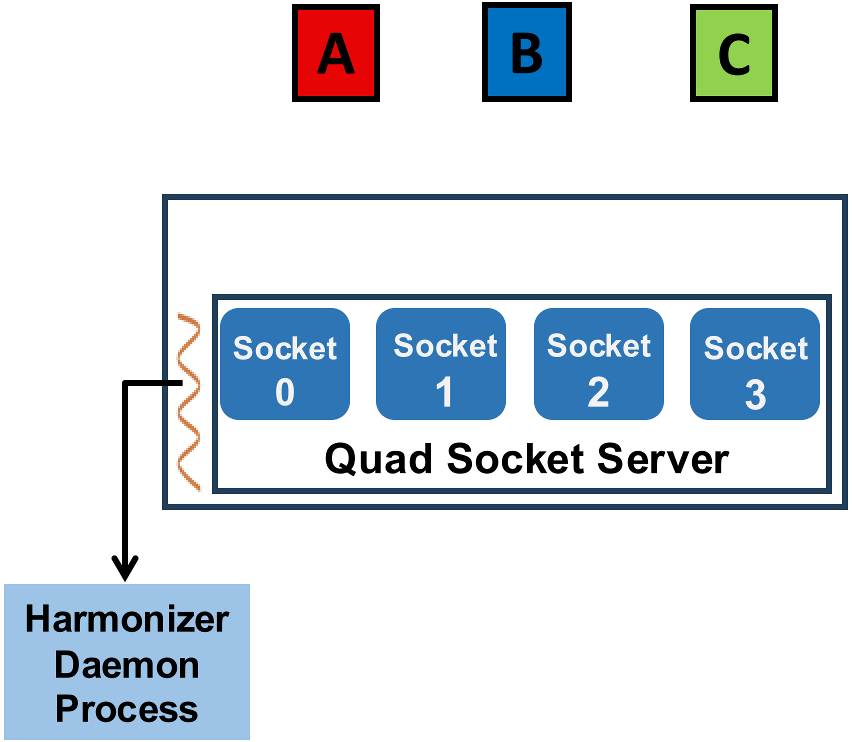
Contributions

- ✓ Harmonizer: A library-based resource management framework for co-running applications on multicore multi-socket servers
  - ✓ Model-free and runtime oblivious
- ✓ Dynamically manages thread placement, core frequency, uncore frequency and core allocation
  - ✓ Uses a lightweight daemon for online profiling of hardware PMCs
- ✓ Experimental Evaluations on a quad-socket 72-core Intel Xeon processor
  - ✓ Using several exascale proxy applications (OpenMP, Kokkos and HCLib)
- ✓ Results
  - ✓ Demonstrating substantially energy savings and performance gains

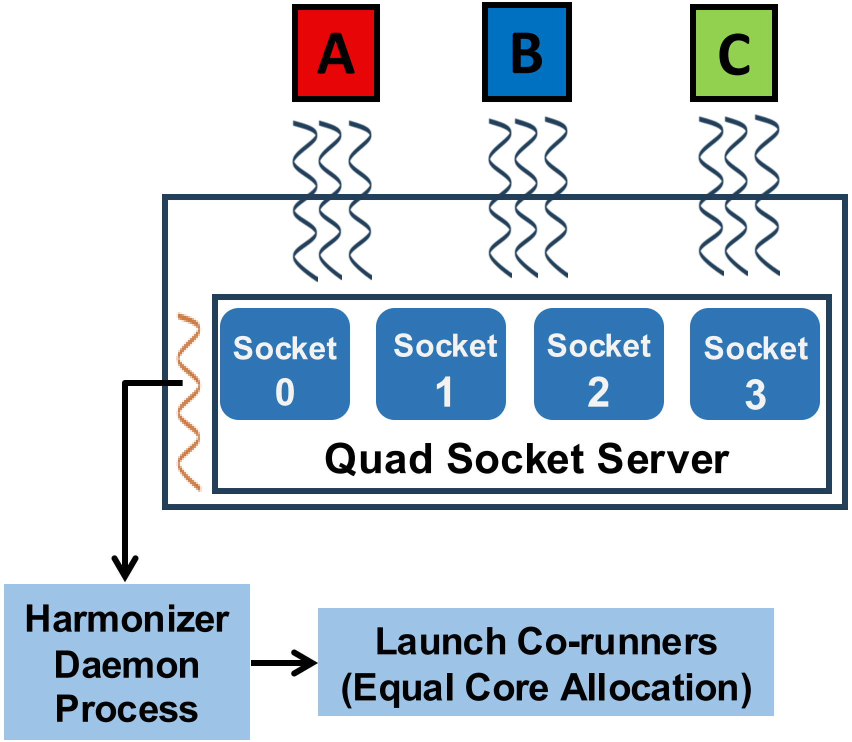
High-level Architecture of Harmonizer



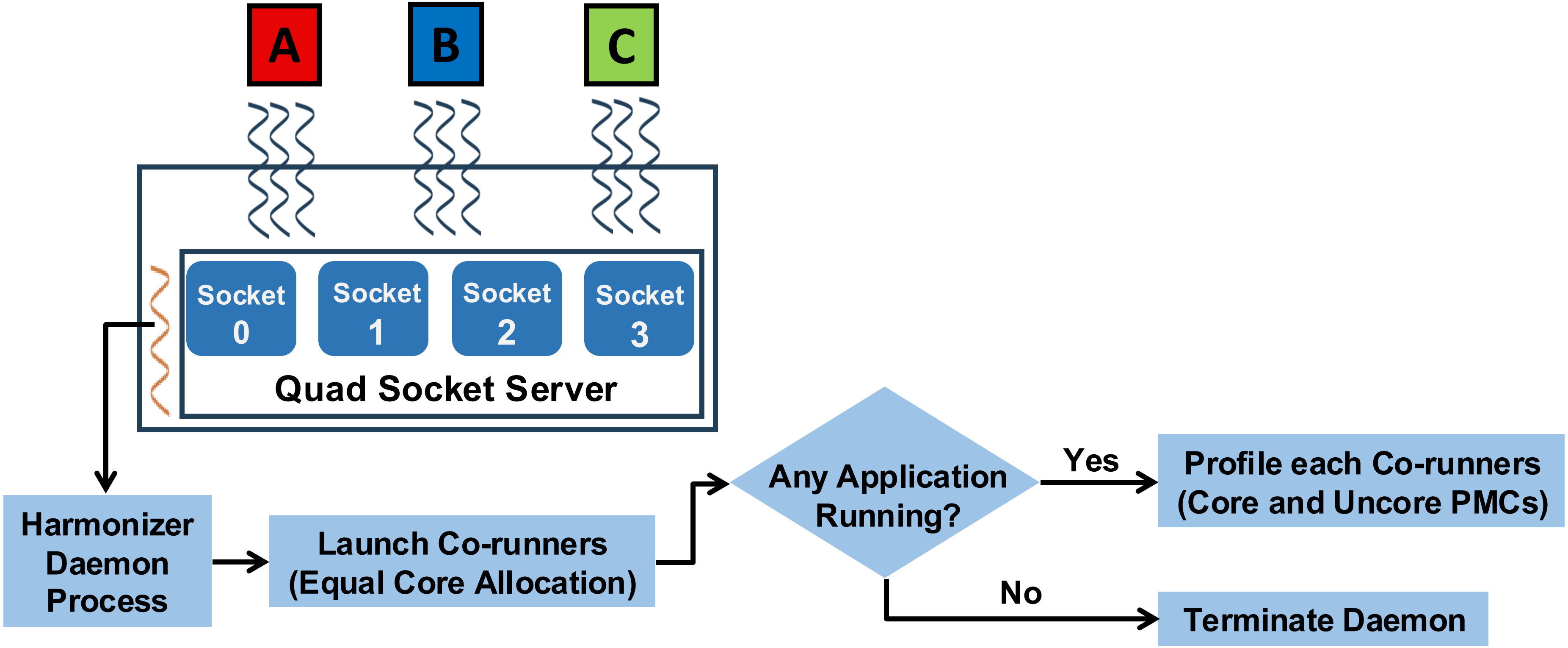
High-level Architecture of Harmonizer



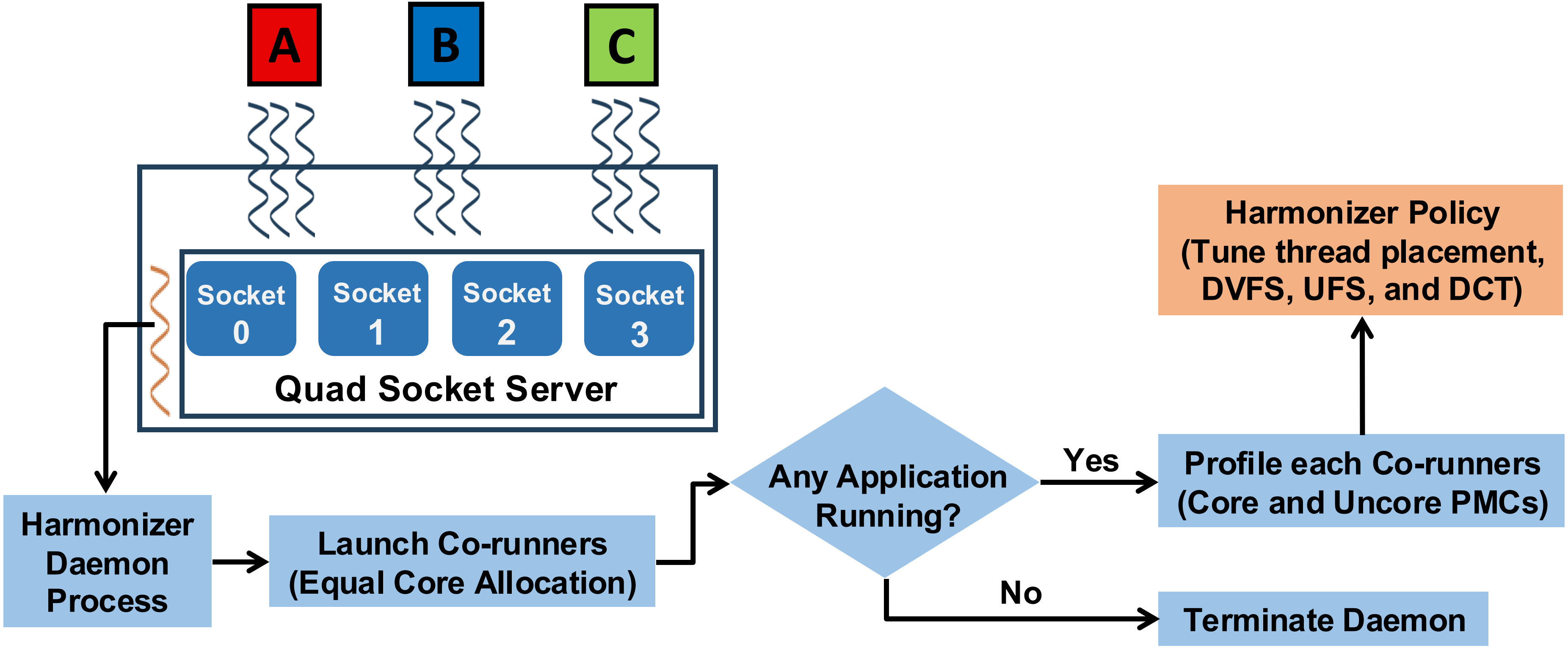
High-level Architecture of Harmonizer


A
B
C
Socket 0
Socket 1
Socket 2
Socket 3
Quad Socket Server
Harmonizer Daemon Process
Launch Co-runners (Equal Core Allocation)

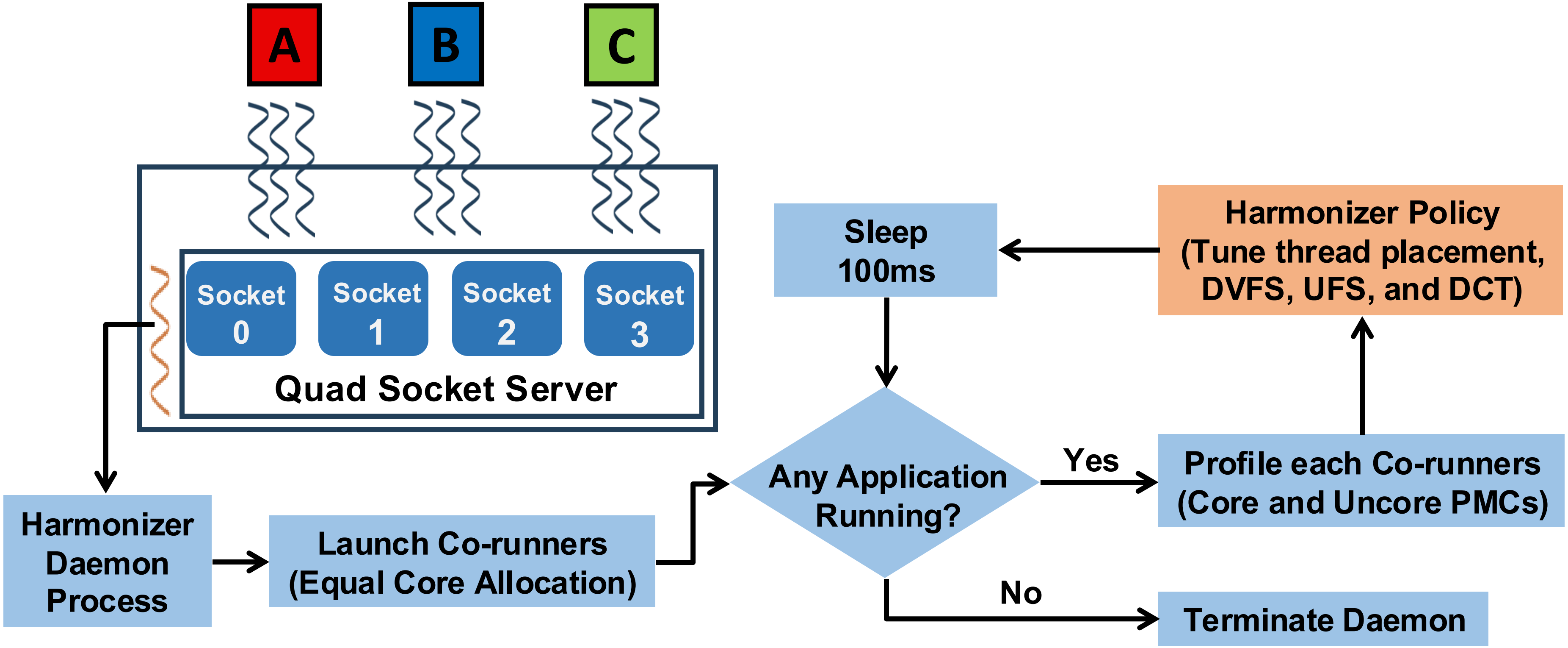
High-level Architecture of Harmonizer


A
B
C
Socket 0
Socket 1
Socket 2
Socket 3
Quad Socket Server
Harmonizer Daemon Process
Launch Co-runners (Equal Core Allocation)
Any Application Running?
Yes
Profile each Co-runners (Core and Uncore PMCs)
No
Terminate Daemon

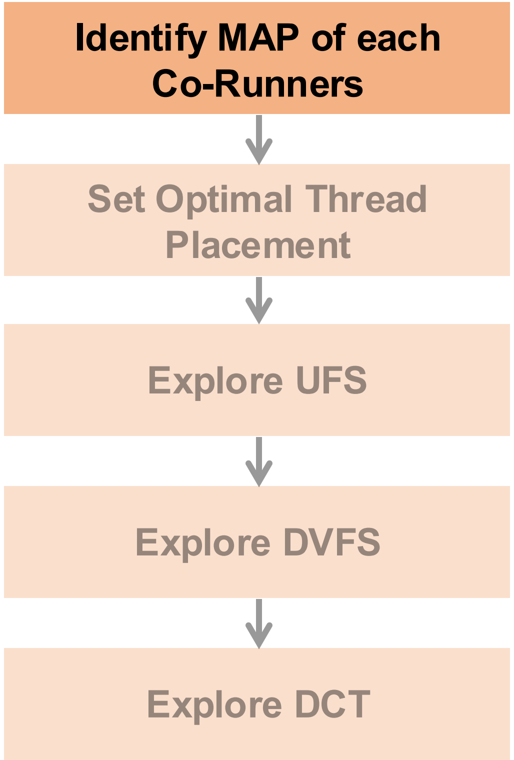
High-level Architecture of Harmonizer


A
B
C
Socket 0
Socket 1
Socket 2
Socket 3
Quad Socket Server
Harmonizer Daemon Process
Launch Co-runners (Equal Core Allocation)
Any Application Running?
Yes
Profile each Co-runners (Core and Uncore PMCs)
Harmonizer Policy (Tune thread placement, DVFS, UFS, and DCT)
No
Terminate Daemon

High-level Architecture of Harmonizer


A
B
C
Socket 0
Socket 1
Socket 2
Socket 3
Quad Socket Server
Harmonizer Daemon Process
Launch Co-runners (Equal Core Allocation)
Any Application Running?
Yes
No
Profile each Co-runners (Core and Uncore PMCs)
Harmonizer Policy (Tune thread placement, DVFS, UFS, and DCT)
Sleep 100ms
Terminate Daemon

Harmonizer Policy



Identify MAP of each Co-Runners

Set Optimal Thread Placement

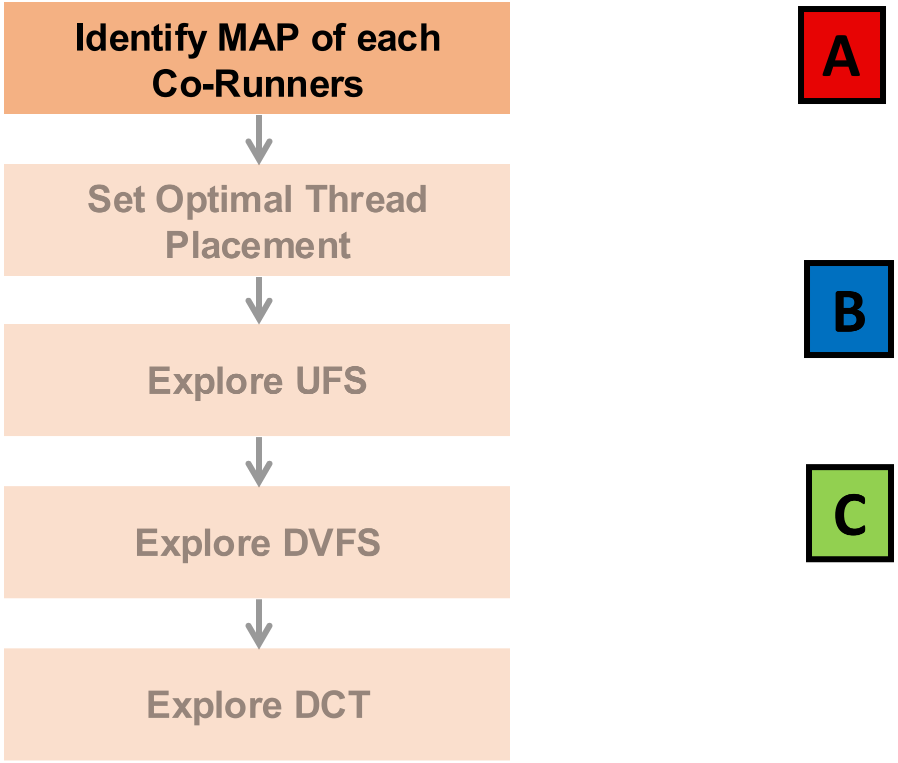
Explore UFS

Explore DVFS

Explore DCT

- Classify each application’s Memory Access Pattern (**MAP**)
  - Core-level PMCs
    - Cache misses
    - Cache accesses
  - Uncore PMCs (Socket-level)
    - Integrated Memory Controller (IMC) accesses

Harmonizer Policy



Identify MAP of each Co-Runners

Set Optimal Thread Placement

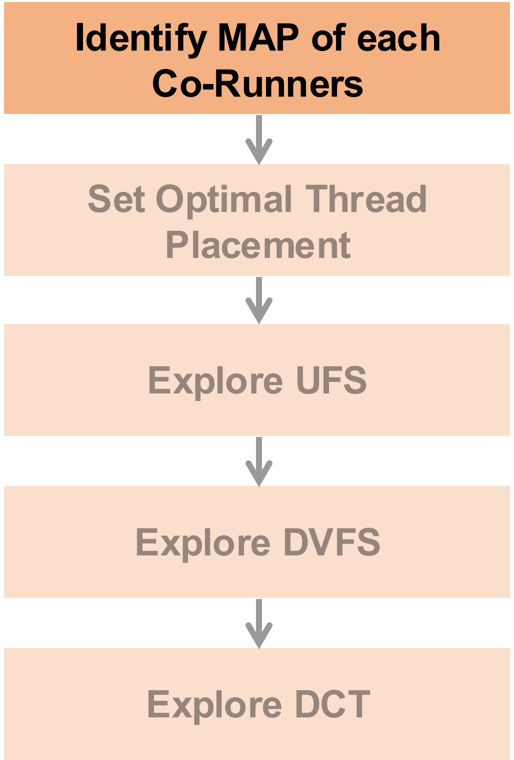
Explore UFS

Explore DVFS

Explore DCT

- Classify each application's Memory Access Pattern (**MAP**)
  - Core-level PMCs
    - Cache misses
    - Cache accesses
  - Uncore PMCs (Socket-level)
    - Integrated Memory Controller (IMC) accesses

Harmonizer Policy



Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

Explore DVFS

Explore DCT

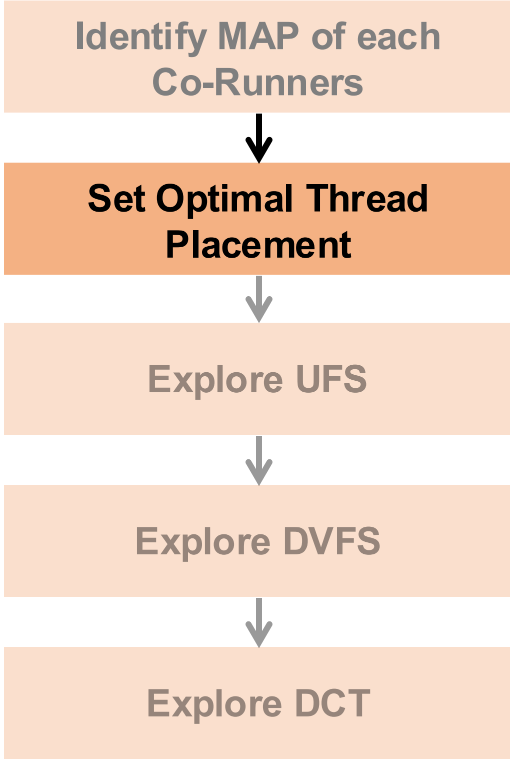
A Cache Sensitive

B Neutral

C Cache Sensitive

- Classify each application’s Memory Access Pattern (**MAP**)
  - Core-level PMCs
    - Cache misses
    - Cache accesses
  - Uncore PMCs (Socket-level)
    - Integrated Memory Controller (IMC) accesses

Harmonizer Policy



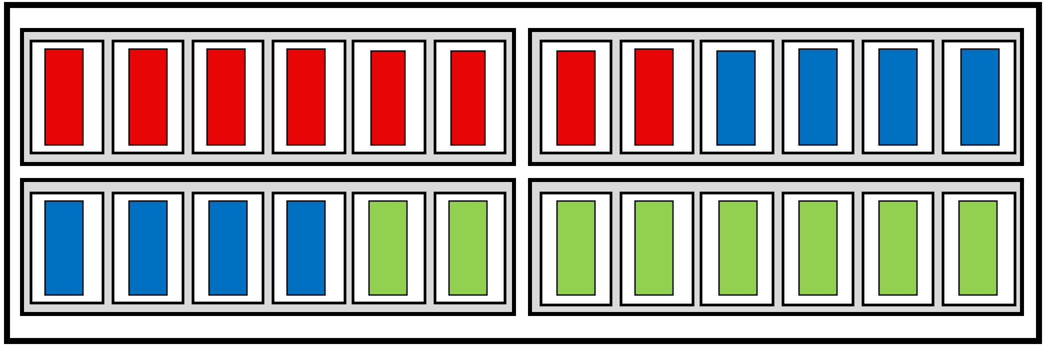
Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

Explore DVFS

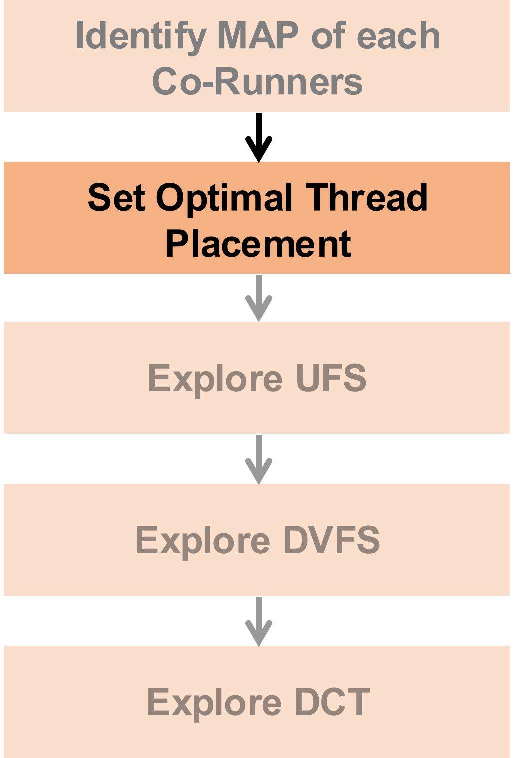
Explore DCT

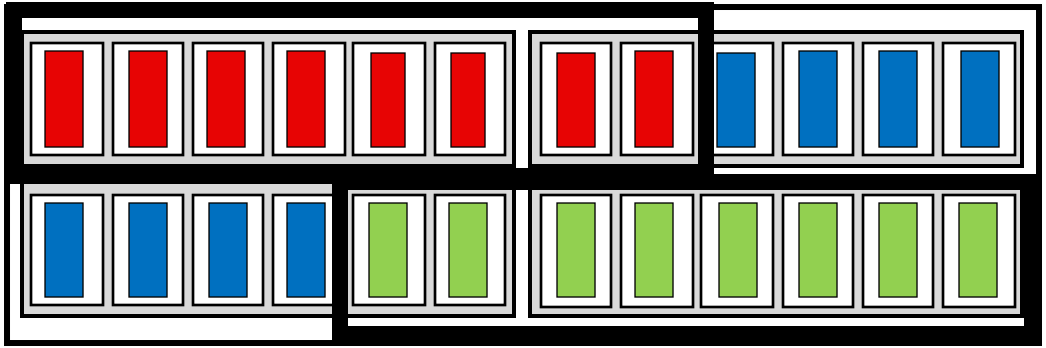


Optimal placement for a particular mix

Cache Sensitive – Neutral – Cache Sensitive

Harmonizer Policy



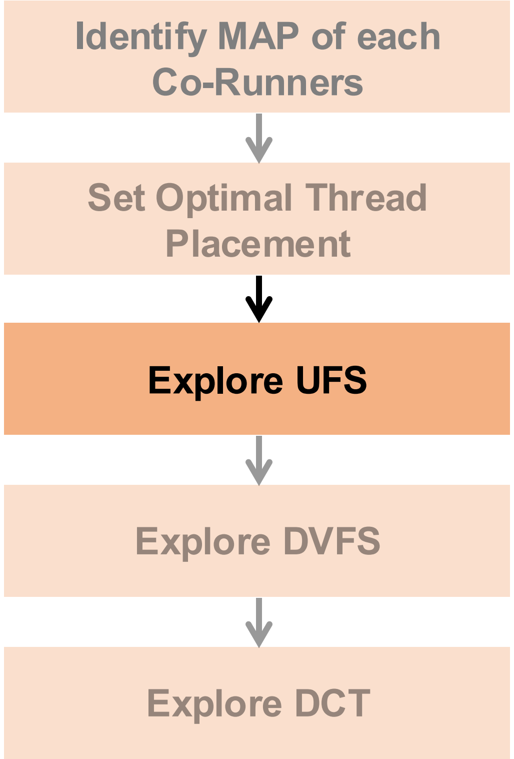


Optimal placement for a particular mix

Cache Sensitive – Neutral – Cache Sensitive

(Block-Cyclic to minimize LLC sharing)

Harmonizer Policy



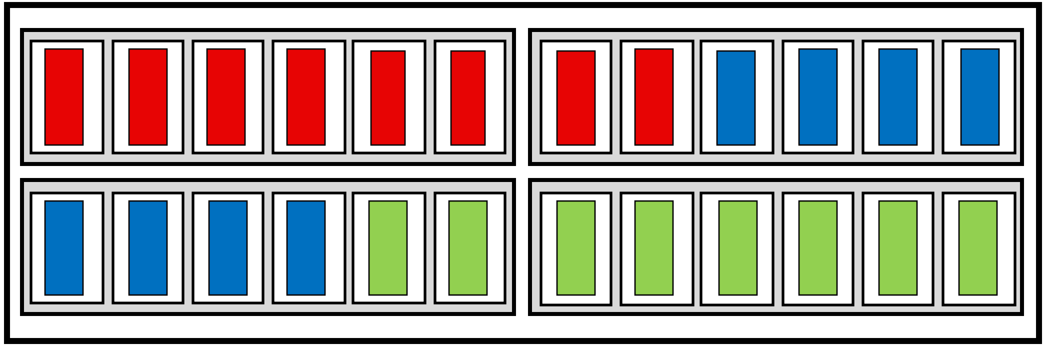
Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

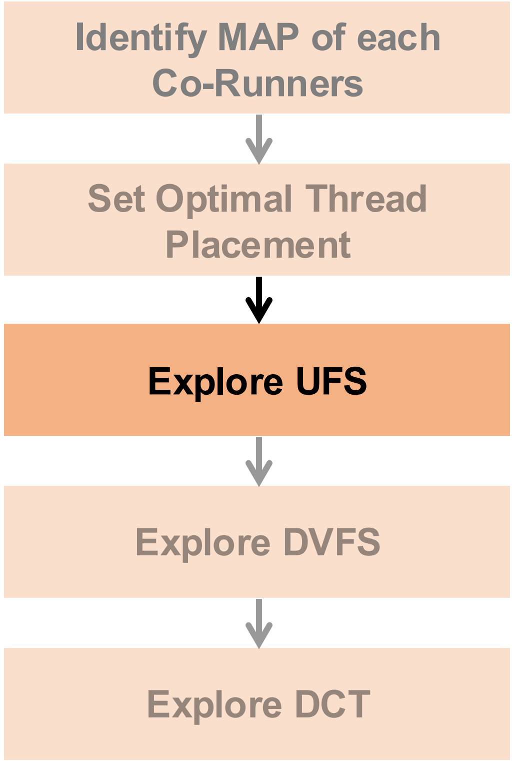
Explore DVFS

Explore DCT



UFS is used to explore optimal UF

Harmonizer Policy



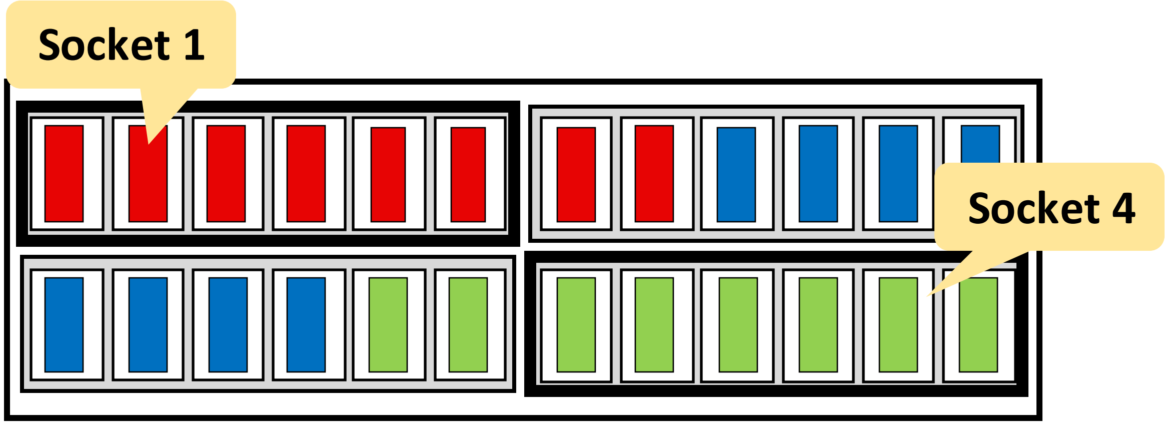
Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

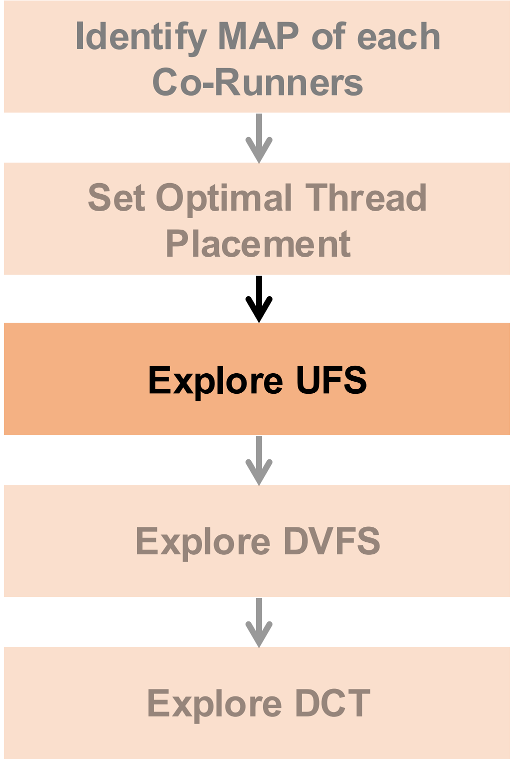
Explore DVFS

Explore DCT



UFS exploration possible only over two sockets in this mix because UFS can be applied at socket-level

Harmonizer Policy

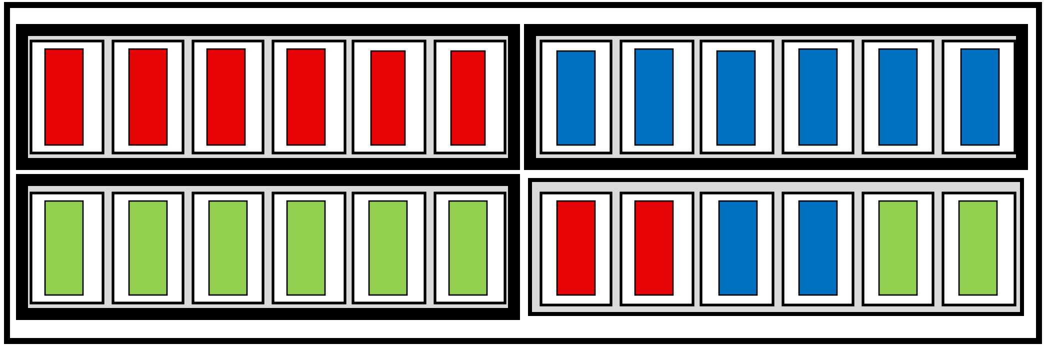

Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

Explore DVFS

Explore DCT



Harmonizer rearranges threads over sockets to maximize application isolation while retaining the behaviour of Block-cyclic placement

Harmonizer Policy

Identify MAP of each Co-Runners

↓

Set Optimal Thread Placement

↓

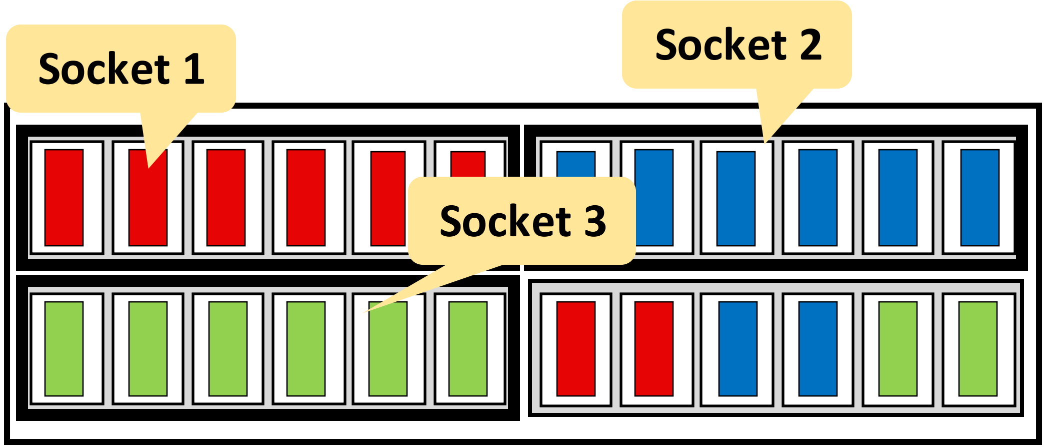
Explore UFS

↓

Explore DVFS

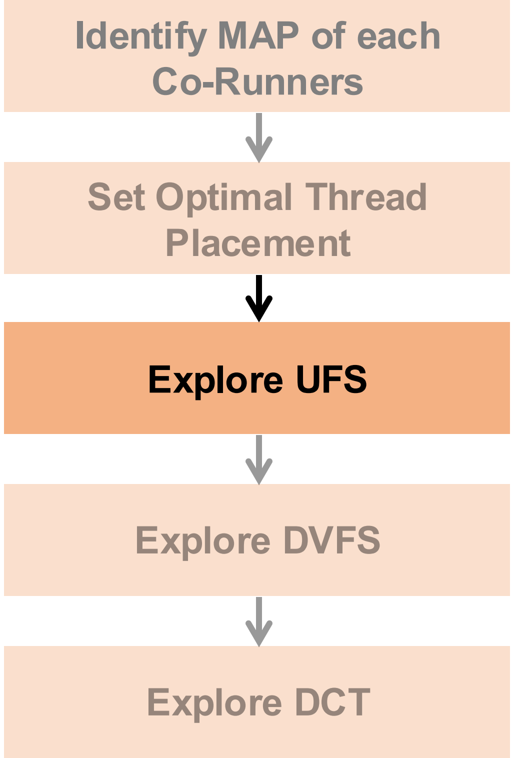
↓

Explore DCT



UFS exploration now possible on three sockets instead of two

Harmonizer Policy



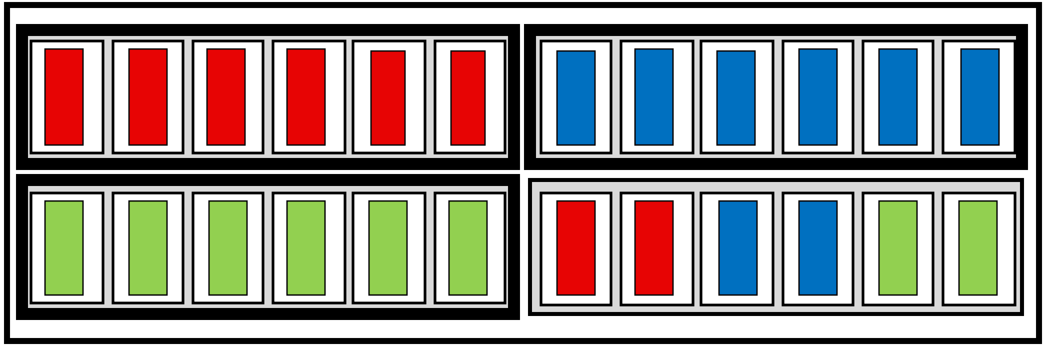
Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

Explore DVFS

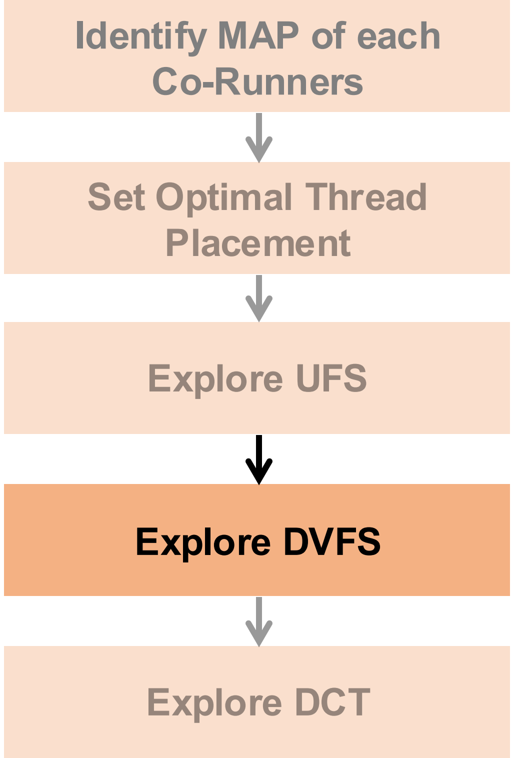
Explore DCT



UFS exploration now possible on three sockets instead of two

Reduced exploration space based on MAP identified over each socket

Harmonizer Policy



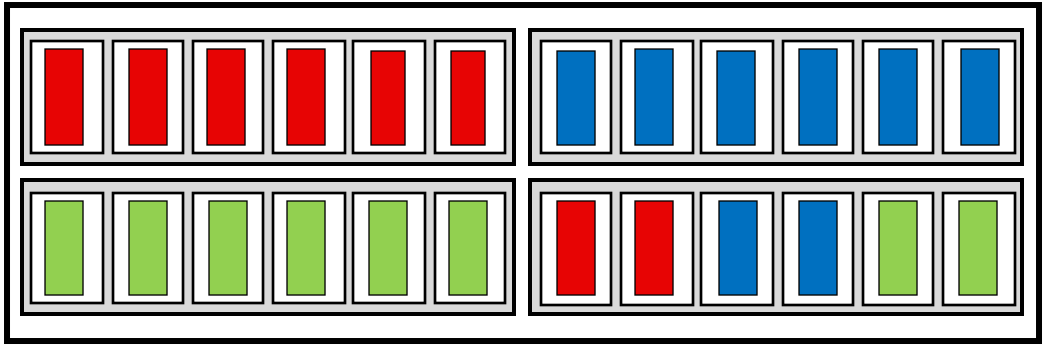
Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

Explore DVFS

Explore DCT



DVFS is used to explore optimal CF for each application

Harmonizer Policy

Identify MAP of each Co-Runners

↓

Set Optimal Thread Placement

↓

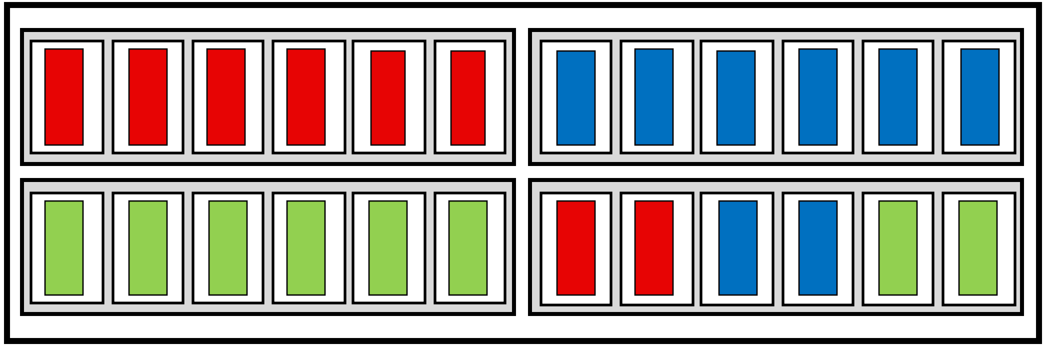
Explore UFS

↓

Explore DVFS

↓

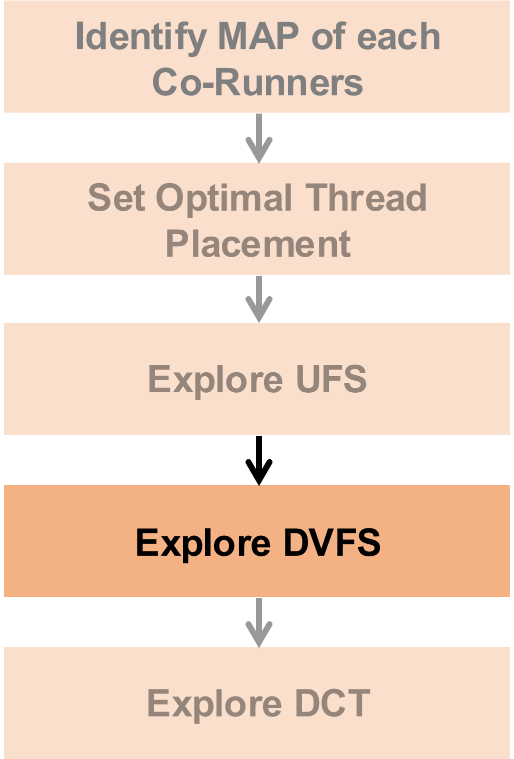
Explore DCT

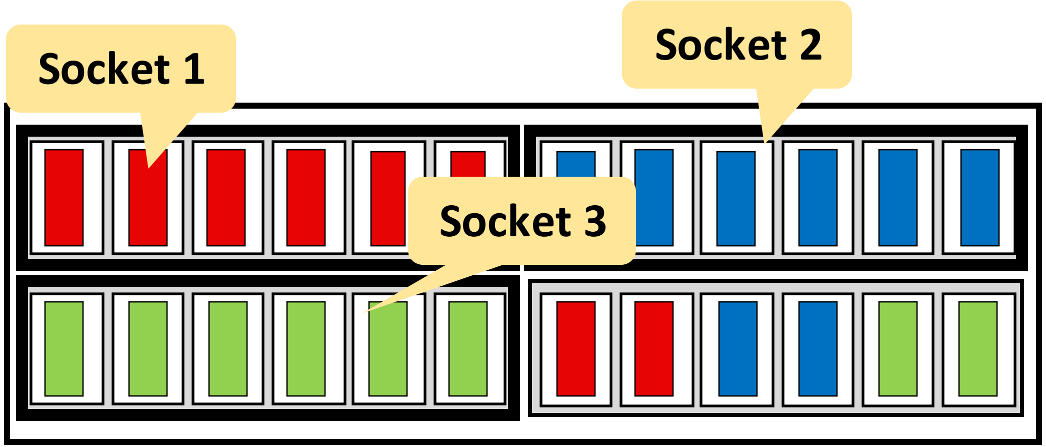


DVFS is used to explore optimal CF for each application

Reduced exploration space based on MAP identified for each application

Harmonizer Policy





Uniform DVFS settings on each core of sockets hosting a single application

Harmonizer Policy

Identify MAP of each Co-Runners

↓

Set Optimal Thread Placement

↓

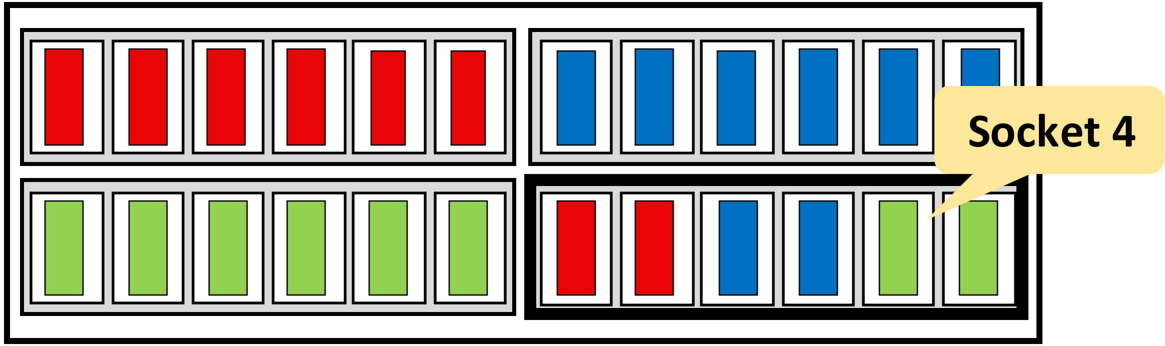
Explore UFS

↓

Explore DVFS

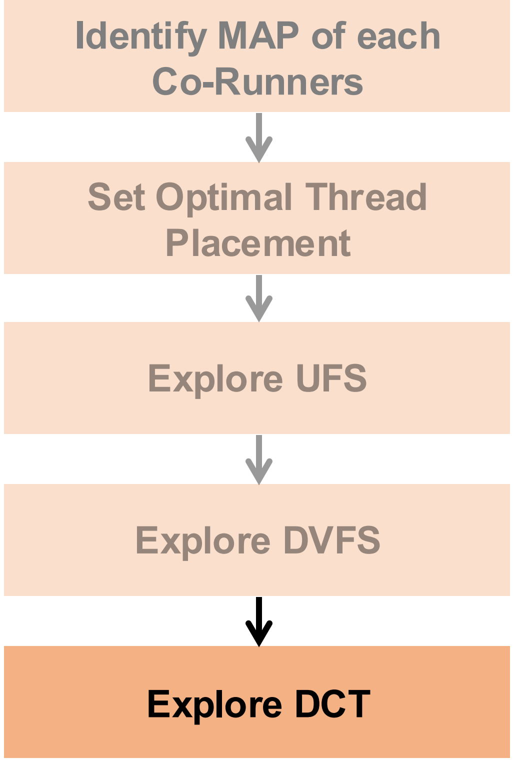
↓

Explore DCT



Non-uniform DVFS setting at socket hosting multiple application's threads

Harmonizer Policy



Identify MAP of each Co-Runners

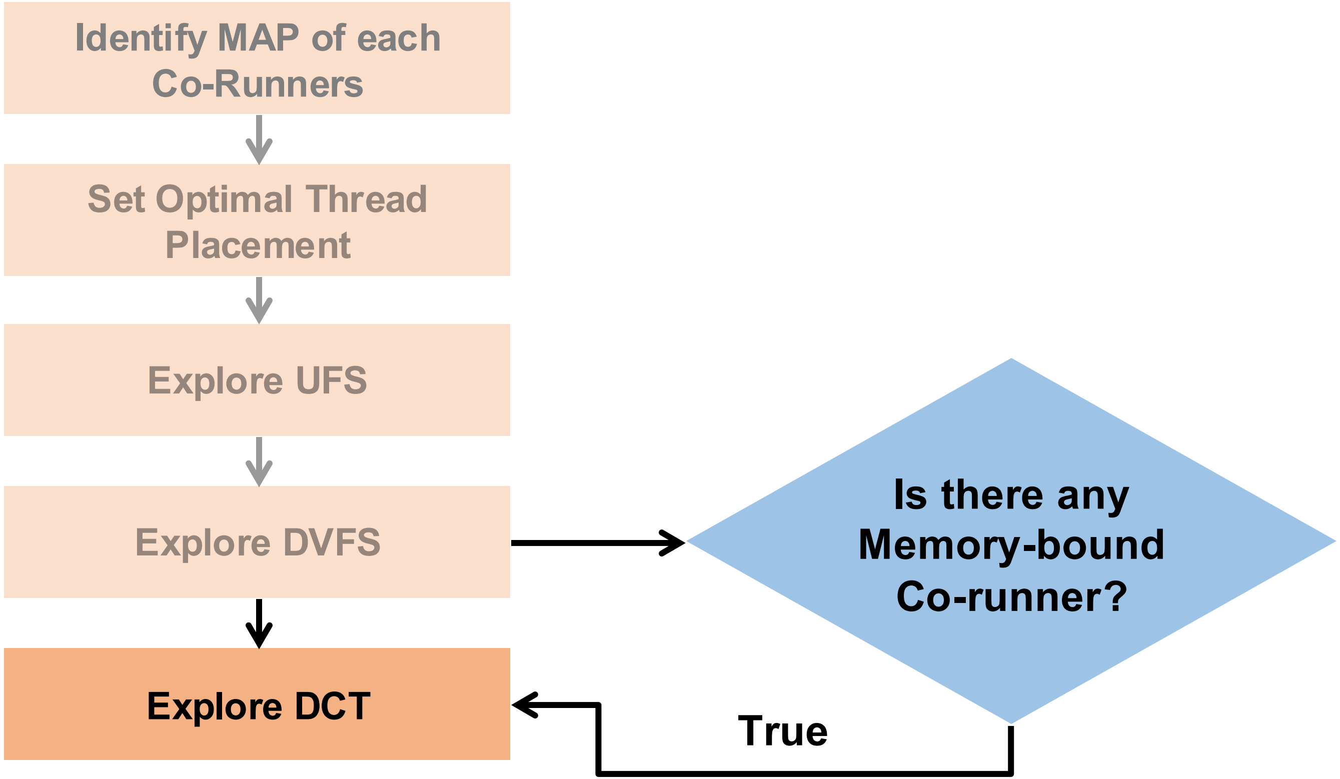
Set Optimal Thread Placement

Explore UFS

Explore DVFS

Explore DCT

Harmonizer Policy



Identify MAP of each Co-Runners

Set Optimal Thread Placement

Explore UFS

Explore DVFS

Explore DCT

Is there any Memory-bound Co-runner?

True

Experimental Methodology

Exascale proxy applications

Type of Applications	Application
Cache Sensitive	SimpleMOC (OpenMP) MinTally (OpenMP) XSBench (OpenMP)
Memory Bound	HPCCG (OpenMP) MiniFE (**Kokkos**)
Neutral	CoHMM (**HCLib**) CoMD (OpenMP)

Experimental Methodology

Exascale proxy applications

Type of Applications	Application
Cache Sensitive	SimpleMOC (OpenMP) MinTally (OpenMP) XSBench (OpenMP)
Memory Bound	HPCCG (OpenMP) MiniFE (**Kokkos**)
Neutral	CoHMM (**HCLib**) CoMD (OpenMP)

Number of Applications in a Mix	Number of Mixes
3	6
4	3

Experimental Methodology

Exascale proxy applications

Type of Applications	Application
Cache Sensitive	SimpleMOC (OpenMP) MinTally (OpenMP) XSBench (OpenMP)
Memory Bound	HPCCG (OpenMP) MiniFE (**Kokkos**)
Neutral	CoHMM (**HCLib**) CoMD (OpenMP)

Number of Applications in a Mix	Number of Mixes
3	6
4	3

Hardware Platform

- Quad socket Intel Xeon 5318H Cooper Lake
- 18 cores per socket, Total 72 cores (144 CPUs)

Experimental Methodology

Exascale proxy applications

Type of Applications	Application
Cache Sensitive	SimpleMOC (OpenMP) MinTally (OpenMP) XSBench (OpenMP)
Memory Bound	HPCCG (OpenMP) MiniFE (**Kokkos**)
Neutral	CoHMM (**HCLib**) CoMD (OpenMP)

Number of Applications in a Mix	Number of Mixes
3	6
4	3

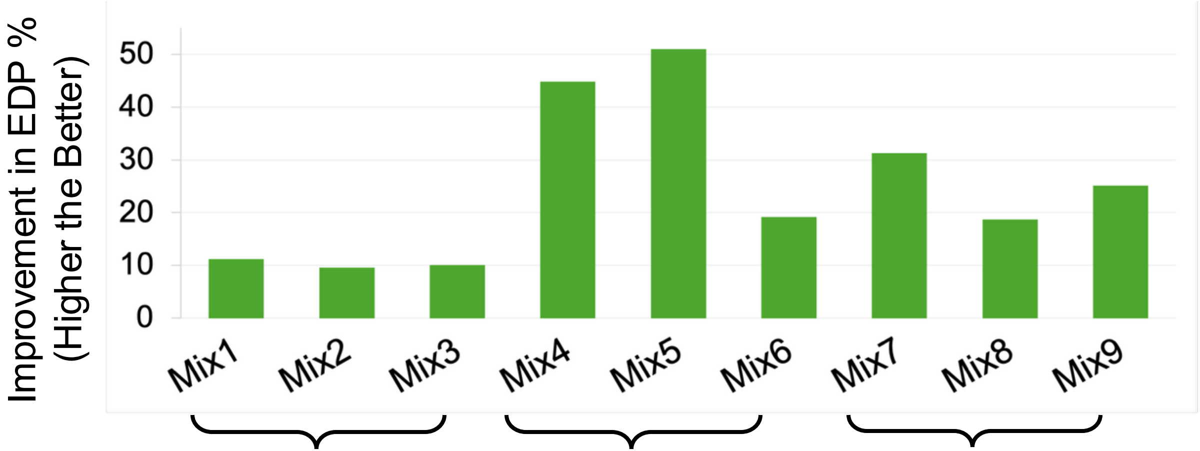
State-of-the-Art used for comparison

- Mapper (TACO’22)
- NuPoCo (PACT’18)

Hardware Platform

- Quad socket Intel Xeon 5318H Cooper Lake
- 18 cores per socket, Total 72 cores (144 CPUs)

EDP of Harmonizer Relative to Default

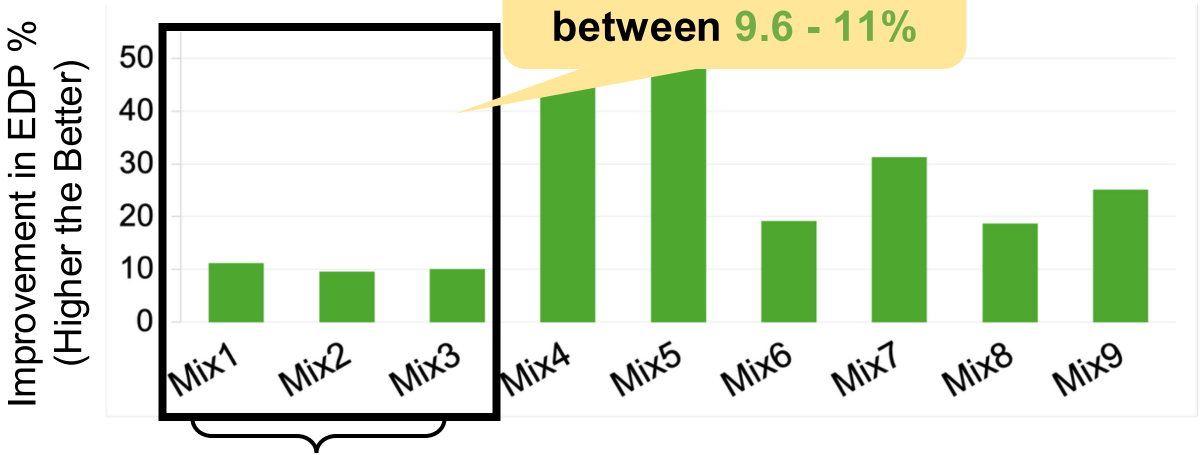


Block-Cyclic Placement

Interleaved Placement

Block-Interleaved Placement

EDP of Harmonizer Relative to Default



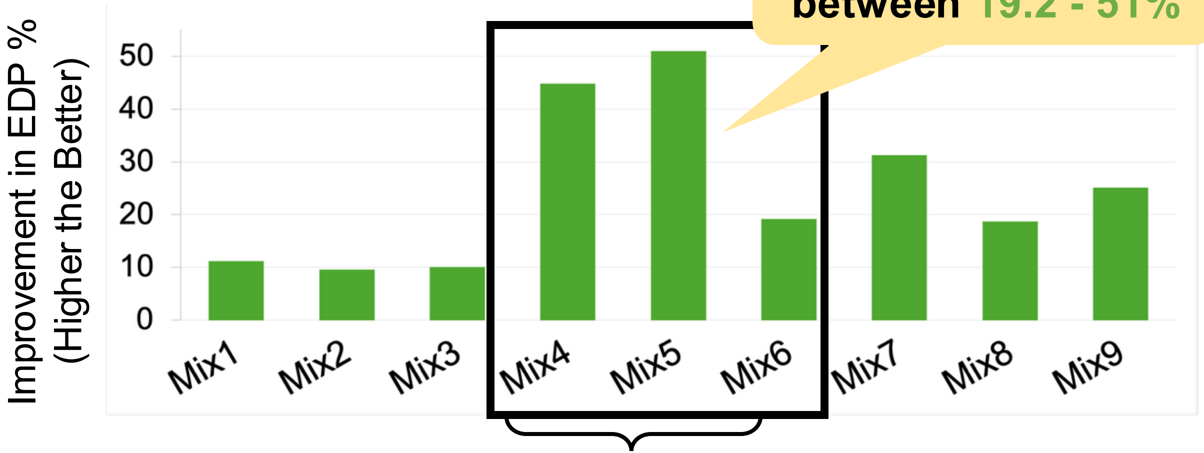
EDP improvement between 9.6 - 11%

Block-Cyclic Placement

Harmonizer Policy	Mean EDP Improvement (Mix1- Mix3)
Thread Placement	7.3%
UFS	3%
DVFS	N/A
DCT	N/A

Improvement in EDP from individual policies

EDP of Harmonizer Relative to Default



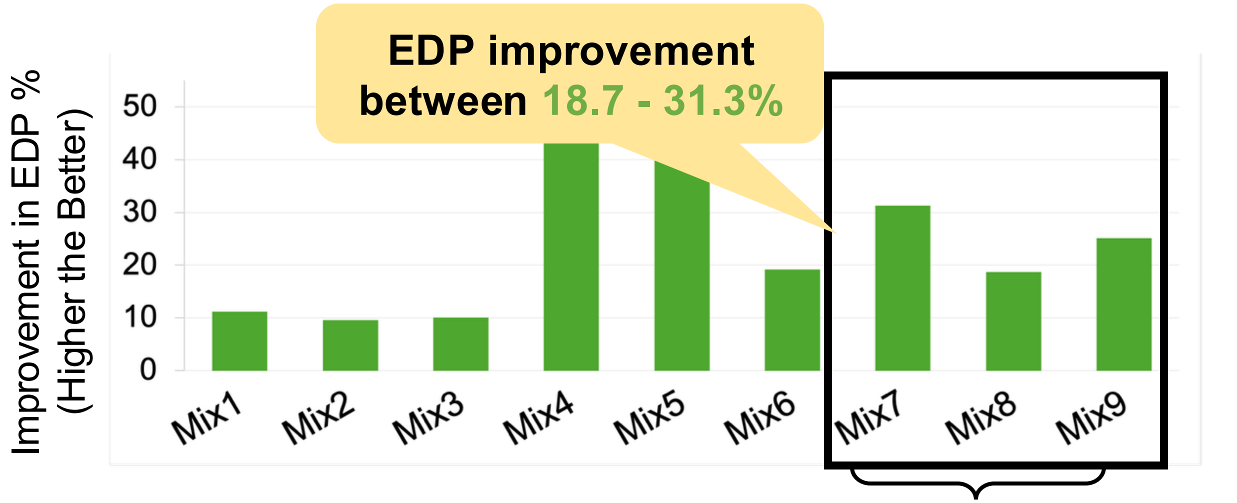
EDP improvement between 19.2 - 51%

Interleaved Placement

Harmonizer Policy	**Mean EDP Improvement (Mix4- Mix6)**
Thread Placement	**26.6%**
UFS	N/A
DVFS	**3%**
DCT	**13.7%**

Improvement in EDP from individual policies

EDP of Harmonizer Relative to Default



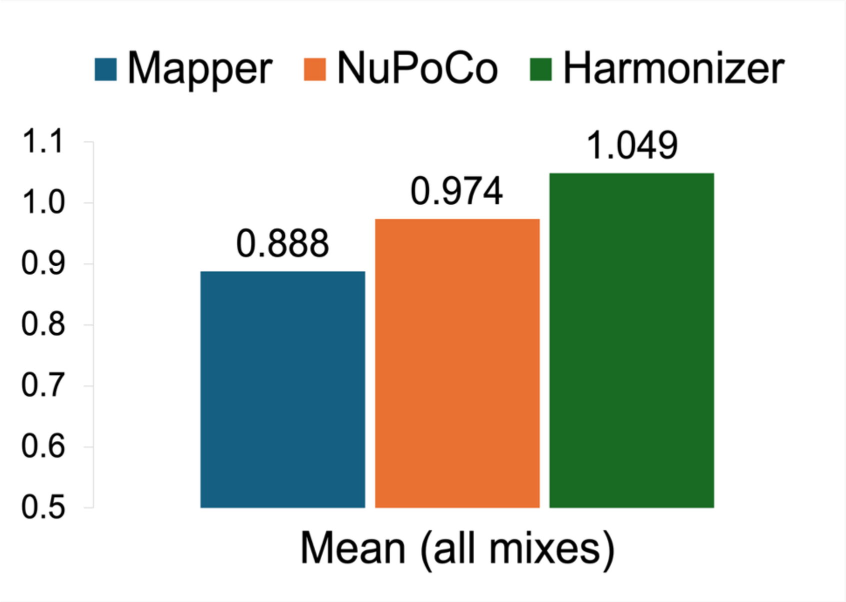
EDP improvement between 18.7 - 31.3%

Block-Interleaved Placement

Harmonizer Policy	Mean EDP Improvement (Mix7- Mix9)
Thread Placement	14.6%
UFS	N/A
DVFS	3.2%
DCT	9.3%

Improvement in EDP from individual policies

System Throughput Relative to Default



System Throughput

Geometric mean of speedup of each application

Summary

- Effective system utilization is key to improving energy efficiency in the exascale era
  - Co-running applications can improve system utilization by complementing each other’s resource requirements
- Harmonizer dynamically profiles the core and uncore PMCs to characterize the behaviour of co-running applications
  - It applies optimal thread placement for improving the system utilization
  - Dynamically tunes each socket’s core and uncore frequencies, and application level core allocation to enhance energy efficiency
- Future Work
  - We plan to extend Harmonizer to handle dynamically varying memory access patterns in applications and scale it to cluster-level environments

Thank You

Scan to access the Harmonizer artifact