RADR 2022

Performance Analysis of Multi-Containerized MD Simulations for Low-Level Resource Allocation

Shingo OKUNO
Akira HIRAI
Naoto FUKUMOTO
(FUJITSU LIMITED)

Introduction

Container-Based Virtualization

FUJITSU

Property	What can we do for drug discovery?
Portability	Distributing applications while maintaining their configurations
Repeatability	Obtaining the same analysis results even on different systems
Low overhead	Running performance-sensitive applications such as molecular dynamics (MD) simulation

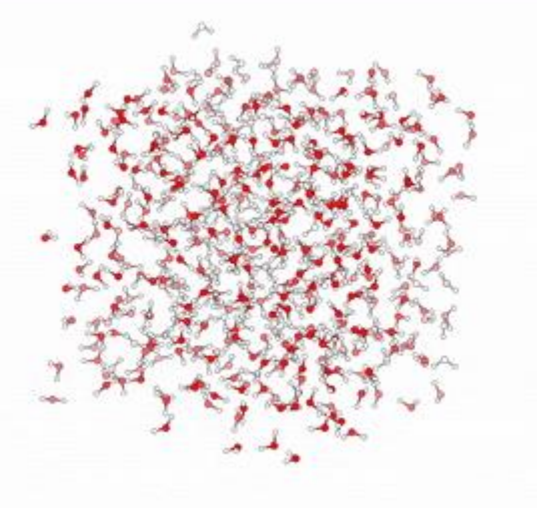
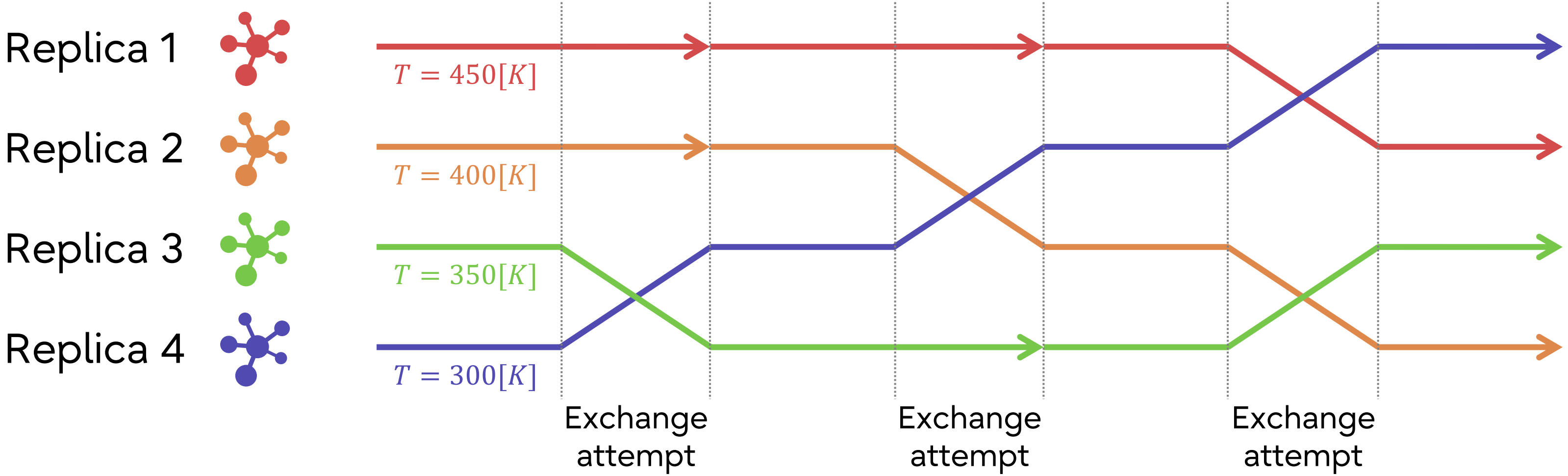


Image provided by Kmckiern (CC BY-SA 4.0)

Ensemble Runs of Containers

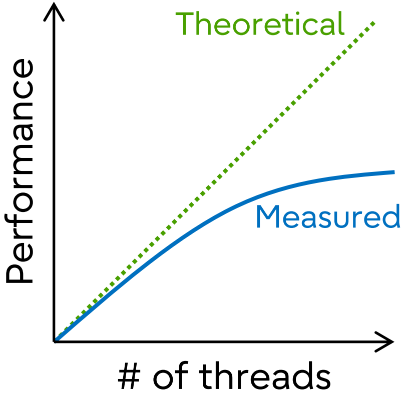
- Increase in # of containers due to microservices
- Other users' containers in multi-tenant cloud environments
- Ensemble simulation with multiple replicas

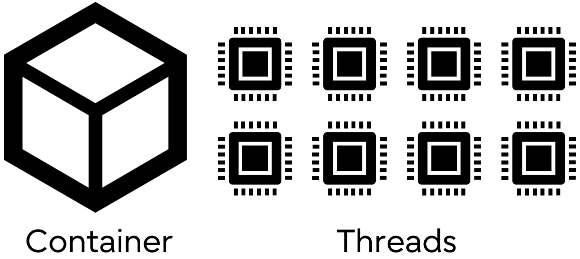
  e.g.) Containerized replica-exchange MD (REMD) simulation



Container Scheduling Problem 1

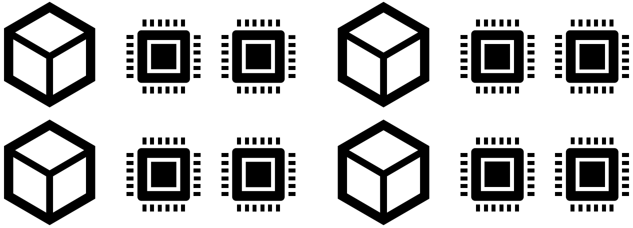
How many containers should we run simultaneously?





Large degree of parallelism

- Good speedup
- Bad throughput

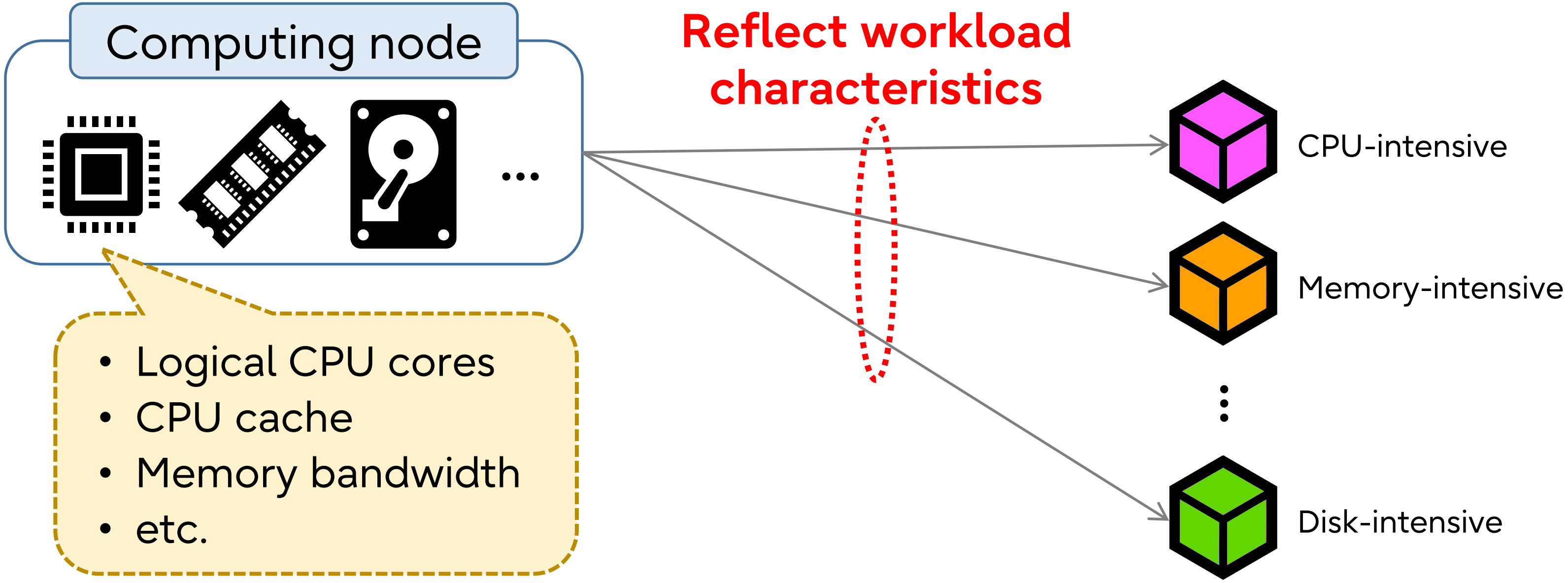


Large number of containers

- Good throughput
- Bad speedup

Container Scheduling Problem 2

How should we allocate resources to each container?



- Logical CPU cores
- CPU cache
- Memory bandwidth
- etc.

Allocate low-level resources

Solution to Container Scheduling Problem

	How many containers should we run simultaneously?	How should we allocate resources to each container?
Existing schedulers	Heuristics approach	No support for low-level resources
Our scheduler	**Automatic adjustment**	**Low-level resource allocation based on workload characteristics**

What We Did in This Research

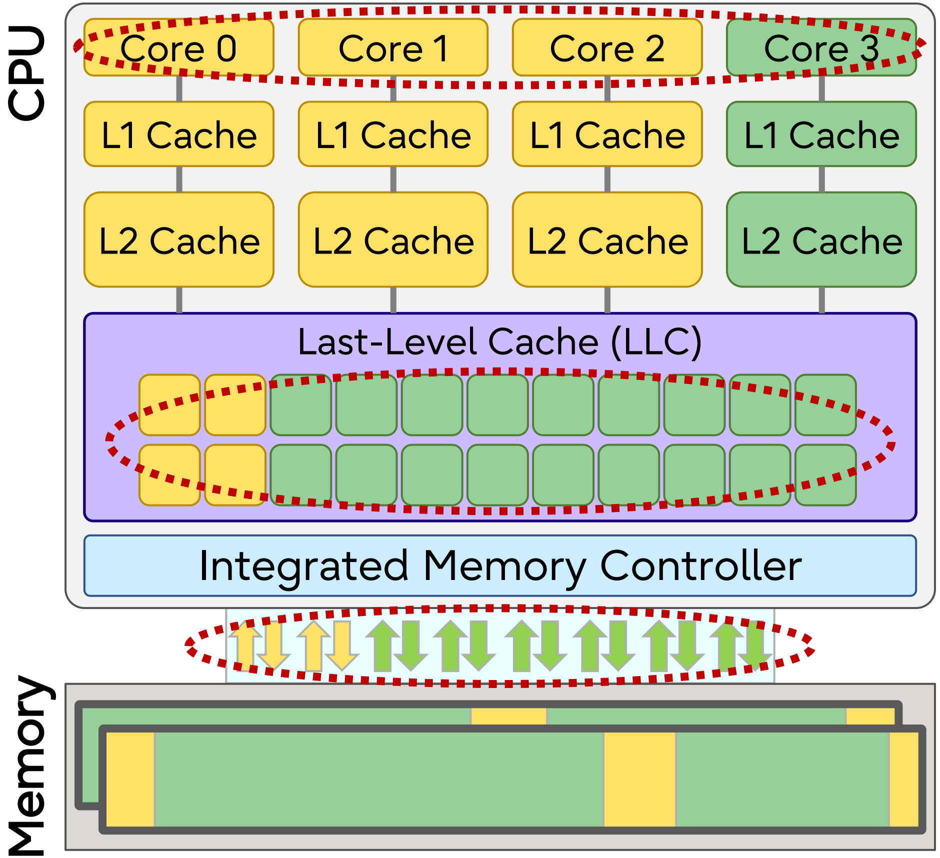
- Discussion about low-level hardware resources for container scheduling

- Evaluation with multi-containerized MD simulations
  - Examining the effect of CPU resource allocation with simultaneous multi-threading (SMT)

Concept of Our Scheduler

Low-Level Hardware Resource Allocation

FUJITSU



1. CPU core control

- Increasing CPU frequency for parallel applications with poor speedup
- Enabling simultaneous multi-threading (SMT) for applications with low resource contention

2. Cache block control

3. Memory bandwidth control

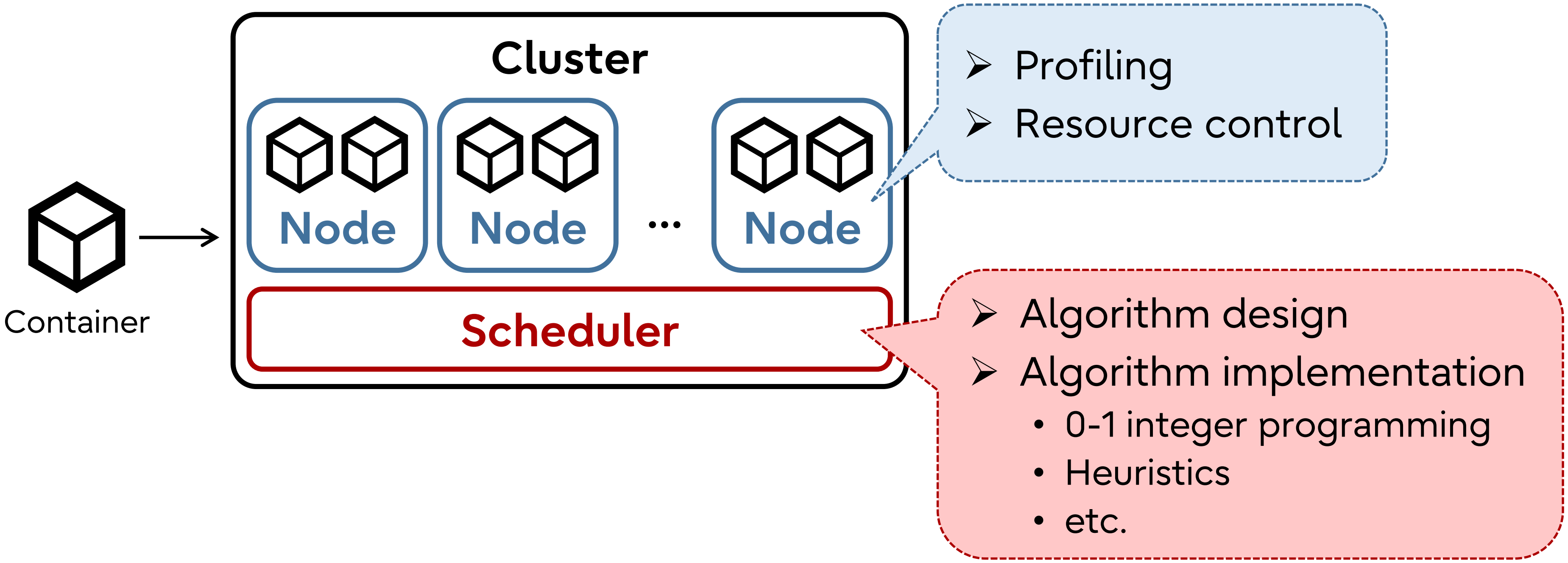
Implementation using Kubernetes Scheduler

FUJITSU

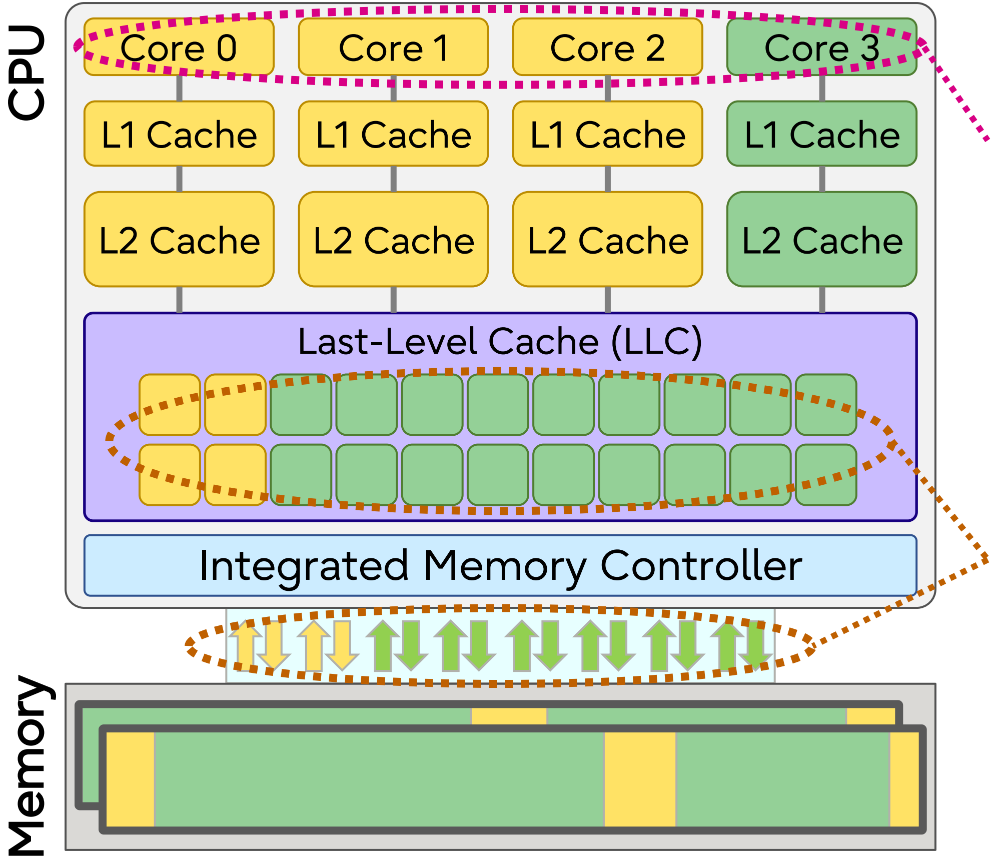
- Default resource types
  - CPU (in number of cores)
  - Memory (in bytes)
- Bin packing along with extended resources
  - 😊 Improvement in container aggregation rate
  - 😟 Difficult to consider workload characteristics

Need to develop own container scheduler

What We Need to Implement Our Scheduler



Resource Control by CRI Resource Manager*



Intel Speed Select Technology

- Power management control

Intel Resource Director Technology

- LLC control
- Memory bandwidth control

* https://github.com/intel/cri-resource-manager

Towards Algorithm Design/Implementation

FUJITSU

Workload analysis → Design → Implemen-tation → Evaluation → Refinement

Conducted preliminary experiments!

- Application
  - Multi-containerized MD simulation
- Low-level hardware resource
  - Simultaneous multi-threading (SMT)

Experiment

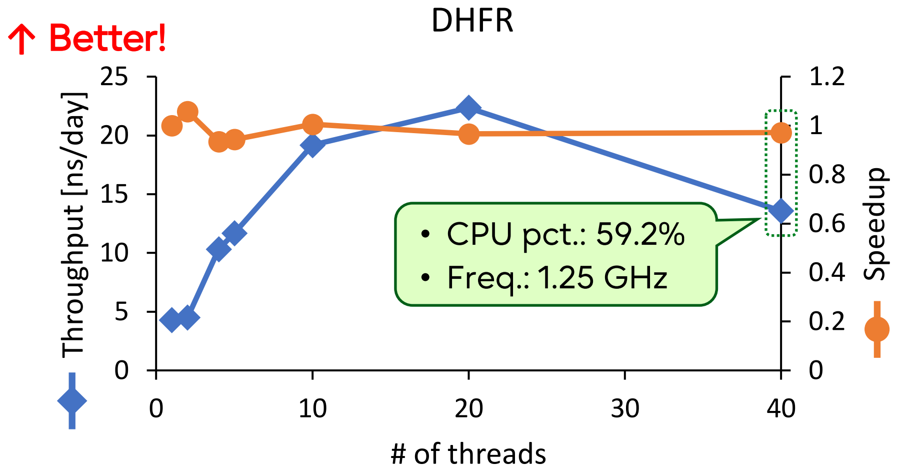
Evaluation Setup

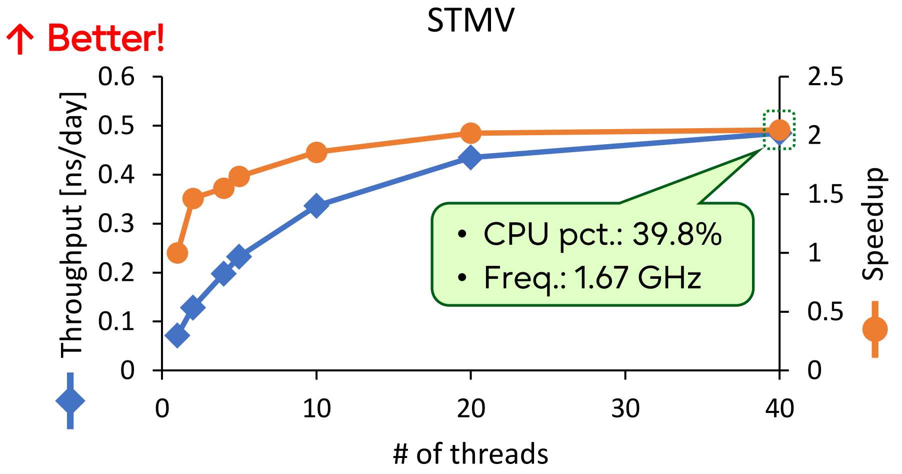
- CPU: Intel Xeon Gold 6148 2.4GHz 20-core × 2 sockets
  - Hyper-Threading Technology enabled (80 logical cores in total)
  - Turbo Boost Technology enabled (1.00–3.70GHz)
  - CPU scaling governor: powersave
- Memory: DDR4-2666 16GB × 12 (96GB / socket)
- Software
  - Container: Docker 20.10.8 with `--cpuset-cpus --cpuset-mems`
  - Compiler: GCC 9.3.0 with `-O3 -march=skylake-avx512`
  - Application: OpenMM 7.6.0 and GROMACS 2021.4
  - Input: Amber20 Benchmark Suite (DHFR and STMV)

Performance of Single Container (OpenMM)

FUJITSU

- Lower CPU percentage in parallel executions
  - Many sequential processes
  - Functions with high memory access (executed only in parallel exec.)
  - Inter-socket memory accesses (# of threads > 20)

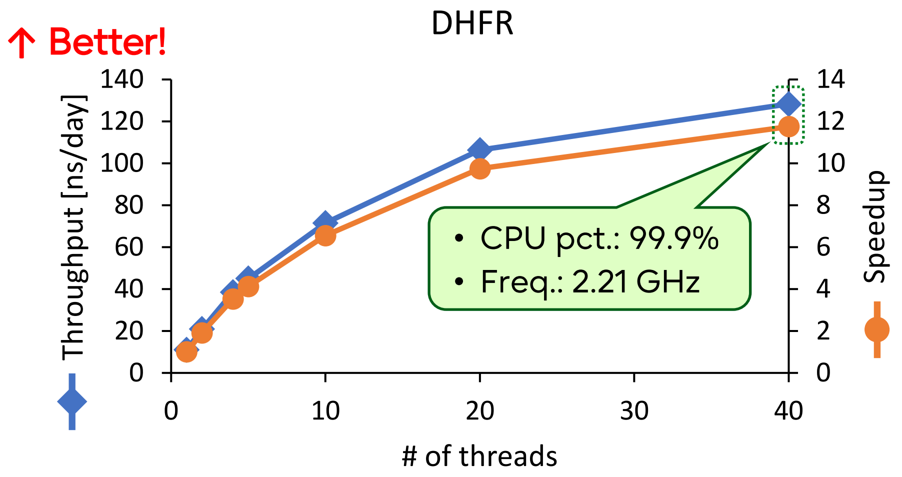


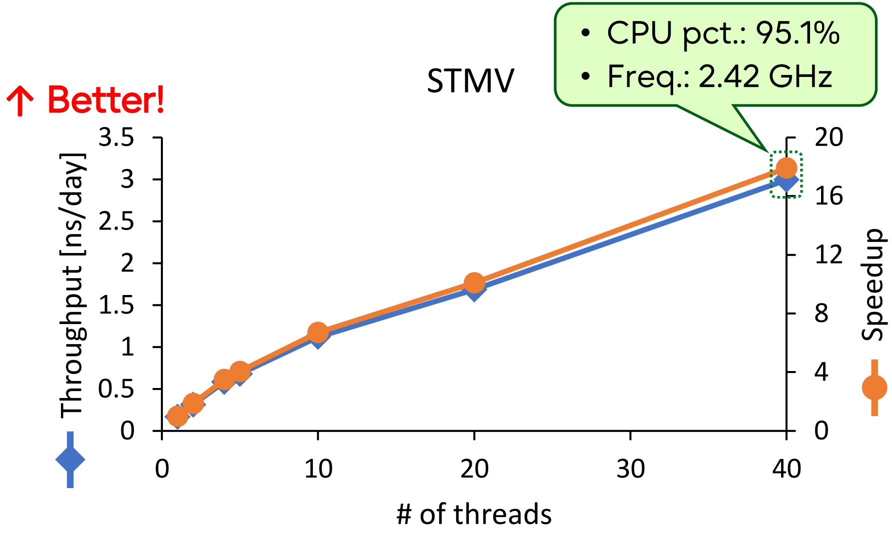


Performance of Single Container (GROMACS)

FUJITSU

- Decrease in CPU frequency with increase in # of threads
  - Turbo Boost Technology
  - AVX-512 (Max 2.20GHz with 20 active cores per socket)





Performance of Multiple Containers (STMV)

FUJITSU

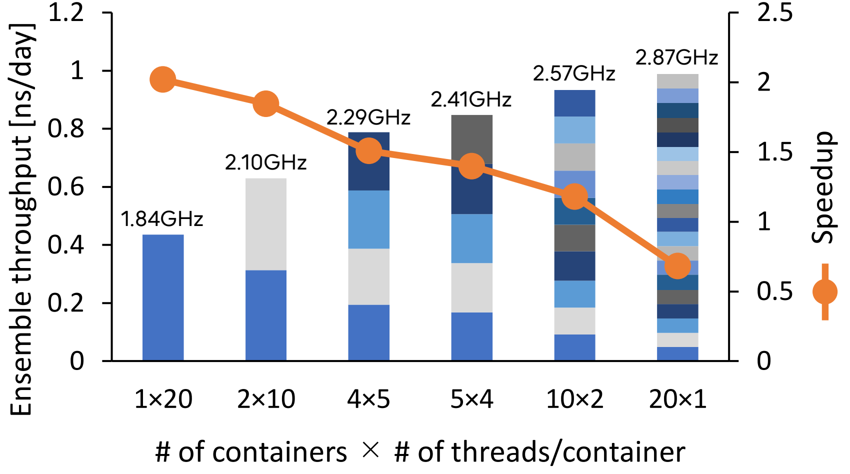
OpenMM

😊 Throughput: 0.435 → 0.987 (× 2.27)
😟 Speedup: 2.02 → 0.682 (× 0.338)

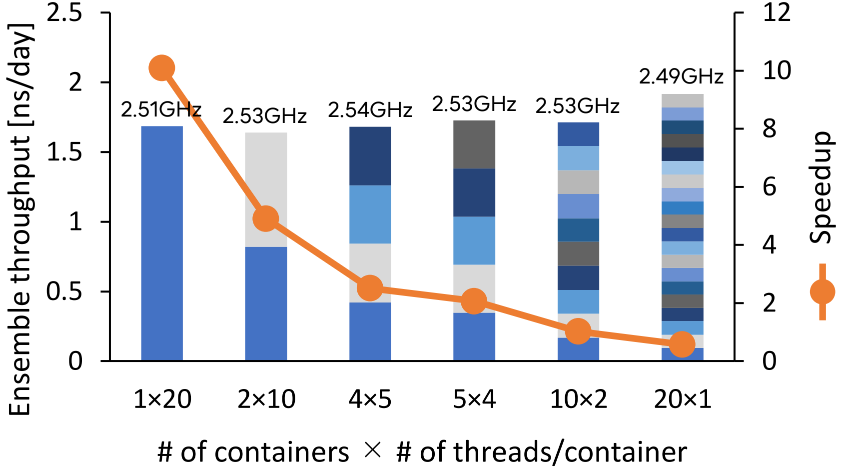
GROMACS

🙂 Throughput: 1.69 → 1.92 (× 1.14)
😱 Speedup: 10.1 → 0.573 (× 0.0568)

↑ Better!



↑ Better!



Performance with SMT (STMV)

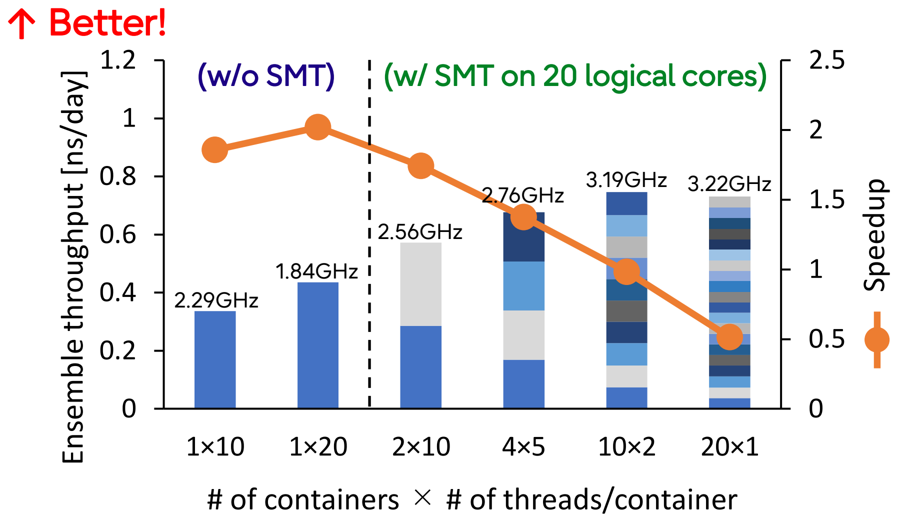
FUJITSU

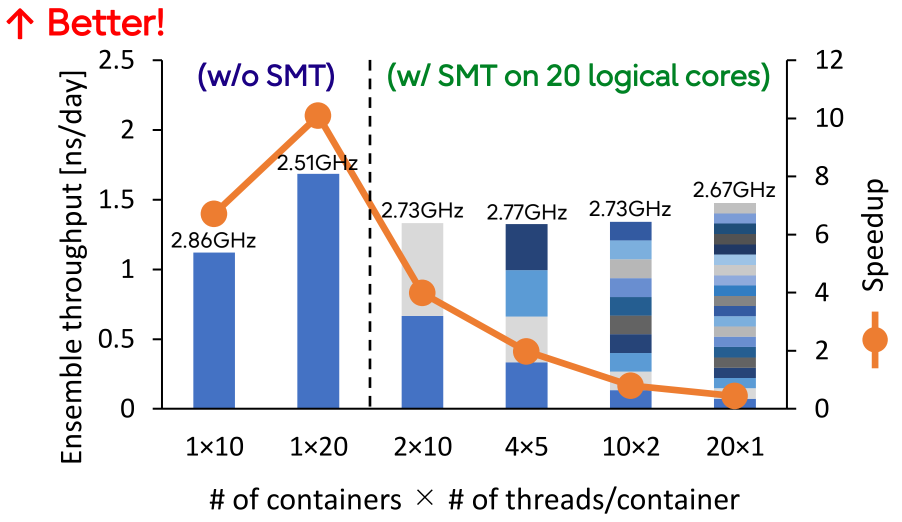
OpenMM (10 containers × 2 threads)

	Throughput	Speedup
vs. 1 cont. × 10 th.	× 2.22	× 0.530
vs. 1 cont. × 20 th.	× 1.71	× 0.487

GROMACS (20 containers × 1 thread)

	Throughput	Speedup
vs. 1 cont. × 10 th.	× 1.32	× 0.0659
vs. 1 cont. × 20 th.	× 0.875	× 0.0438





Conclusion

Conclusion

FUJITSU

Container scheduling problem

- How many containers and how many threads per container should we allocate?
- Which low-level resources should we allocate to reflect workload characteristics?

- Scheduling strategies in multi-containerized MD simulations
  - Large number of containers with fewer threads
    - Improving ensemble throughput while decreasing speedup
  - CPU core allocation considering SMT
    - OpenMM: 2.22-fold ensemble throughput with 0.530-fold speedup
  - Variable CPU frequency
- Implementing our scheduler…

Thank you