上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

DWS: Demand-aware Work-Stealing in Multi-programmed Multi-core Architectures

Quan Chen, Long Zheng, Minyi Guo

Shanghai Jiao Tong University, China

PMAM 2014

Outline

- Background
- Problem & Motivation
- Demand-aware Work-Stealing (DWS)
- Evaluation
- Conclusions

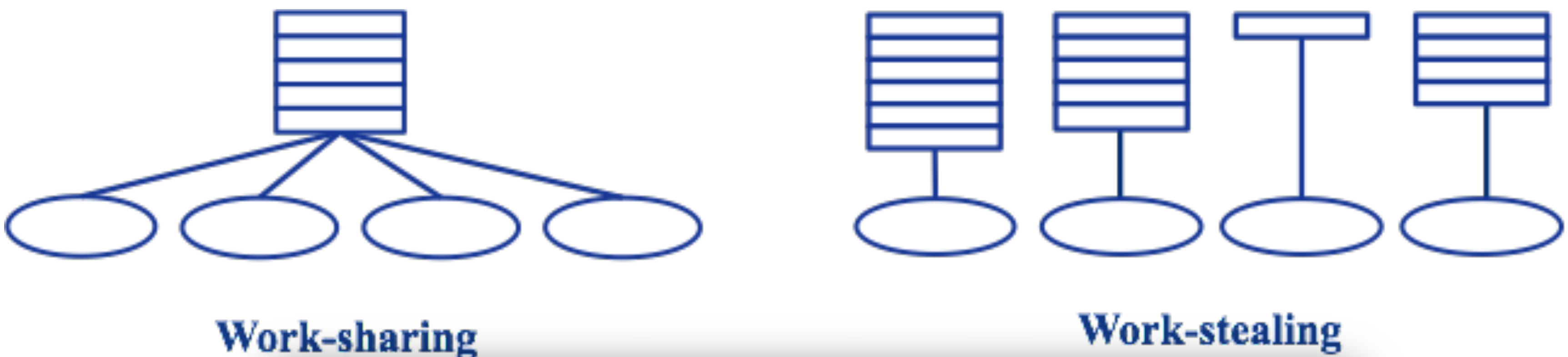
Background

- Hardware: Multi-core/Many-core Architectures

- Scenario: Multiple parallel programs

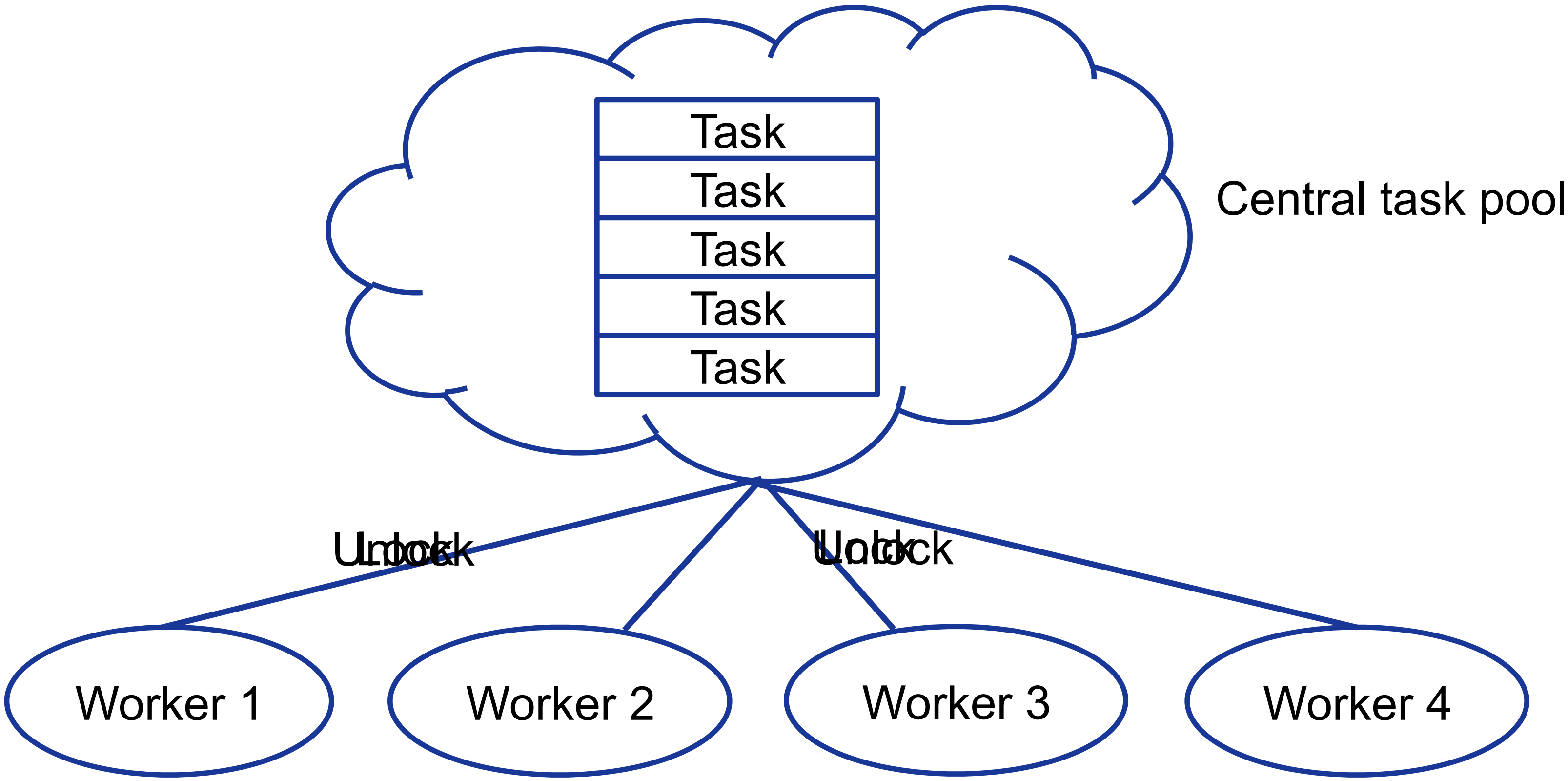
Background-parallel programs

- Traditional parallel programs
  - ***Hard*** to adjust the number of threads at runtime
- Task-based parallel programs
  - Dynamic task scheduling



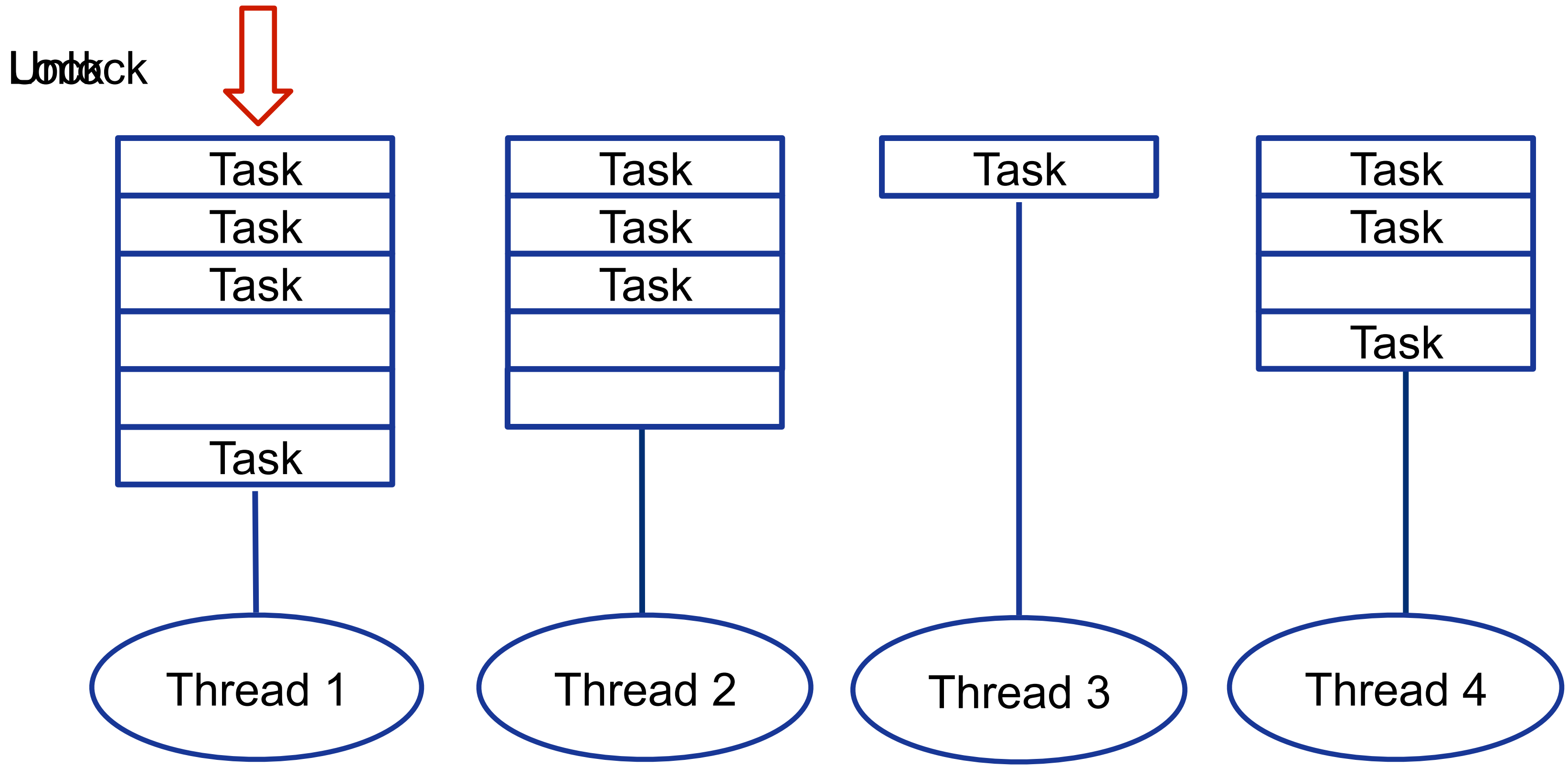
Easy to adjust the number of workers

Work-sharing



Lock the central task pool when getting a task

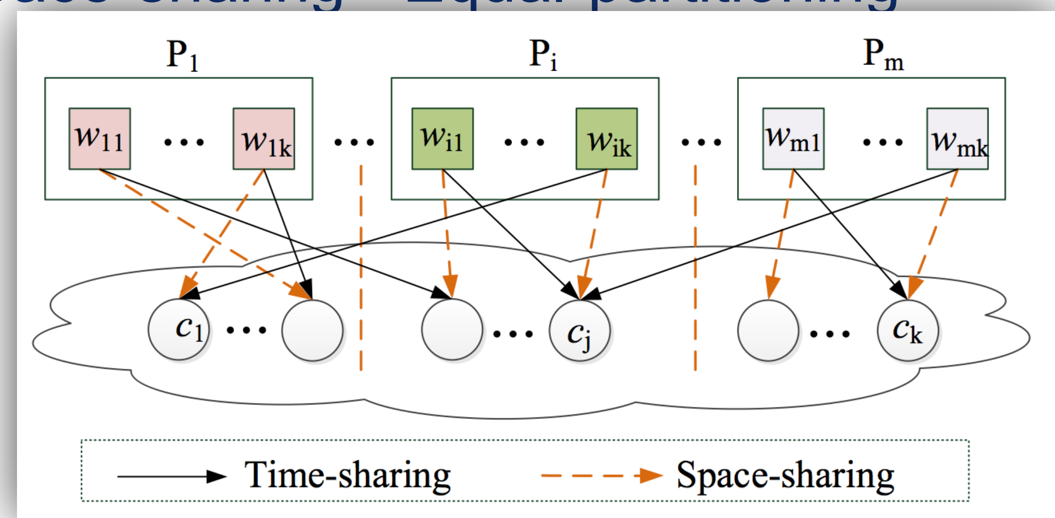
Work-stealing



Problem & Motivation

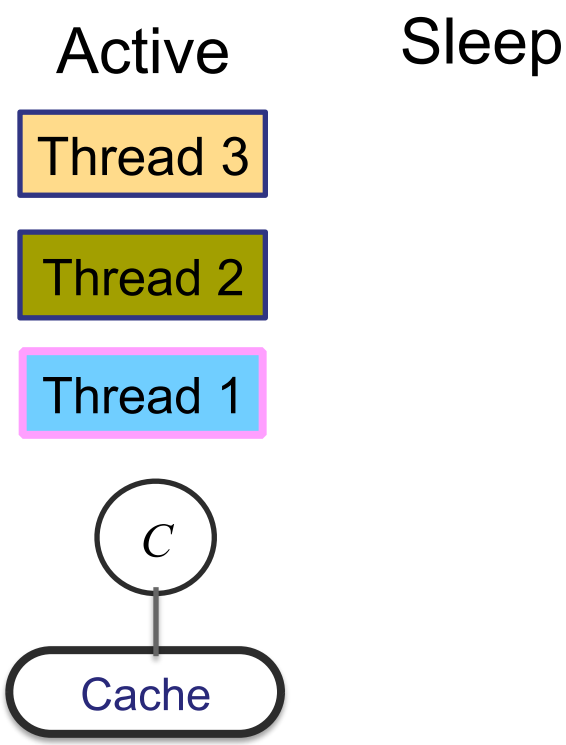
- Aggressive feature of work-stealing
  - On a *k*-core computer, *k* threads/workers are launched

- Existing solutions
  - Time-sharing - ABP yielding mechanism
  - Space-sharing - Equal-partitioning



Time-sharing

- ABP yielding mechanism
  - If a thread fails to steal a task, it goes to sleep



Unfair resource allocation

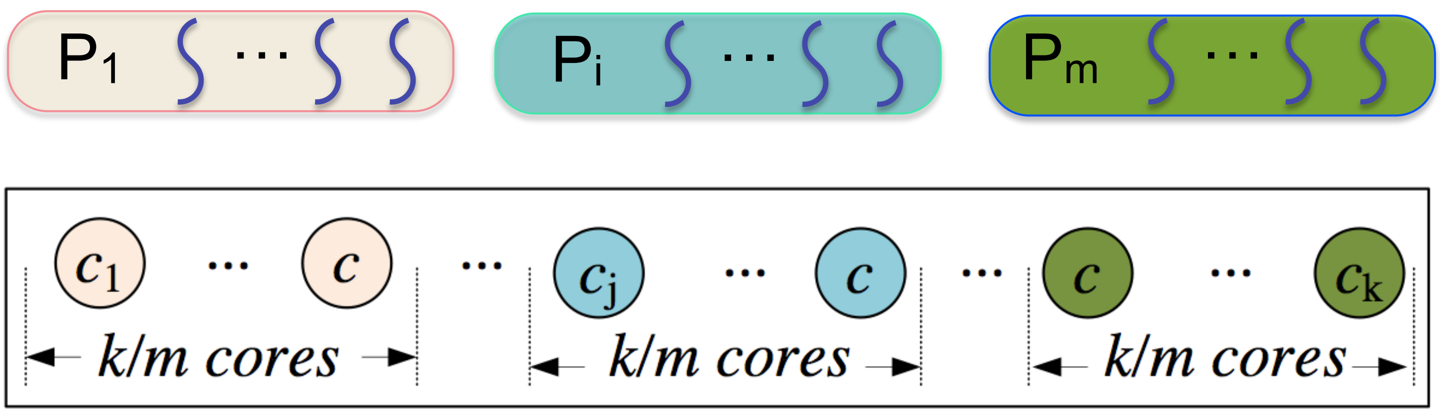
Poor data locality

SHANGHAI JIAO TONG UNIVERSITY

Space-sharing

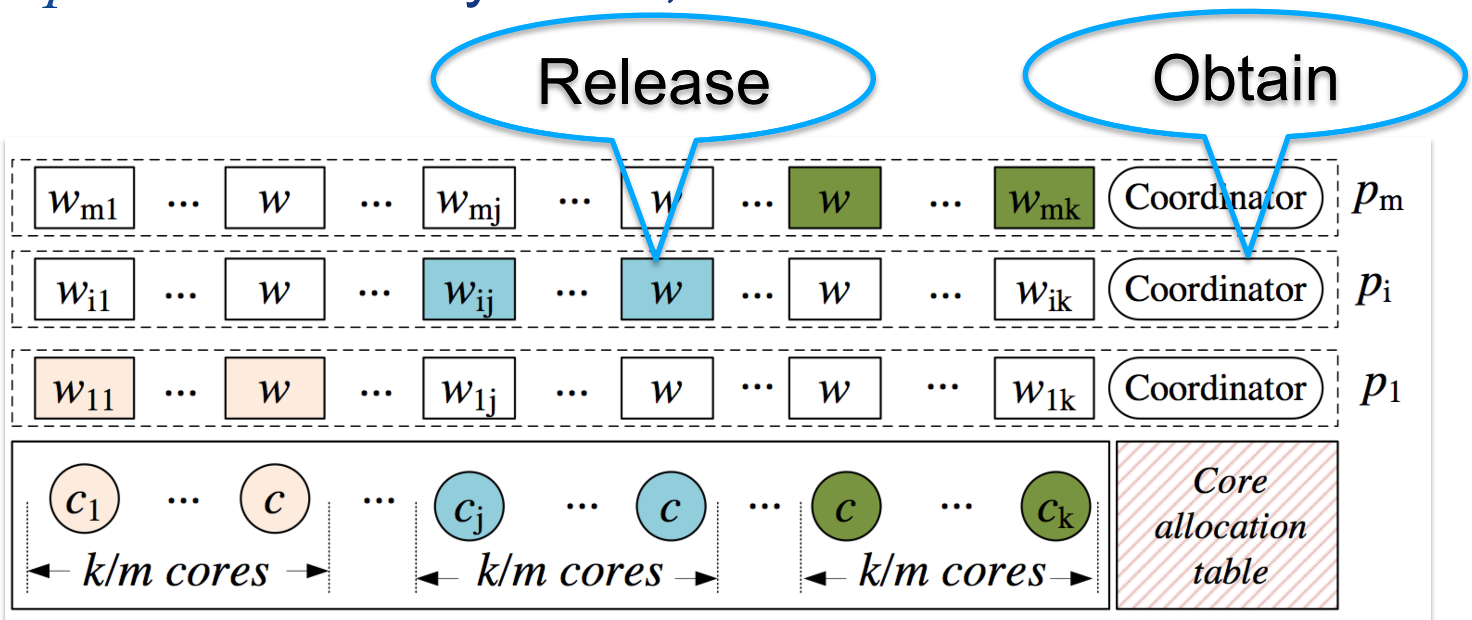
- Equal-partitioning mechanism
  - If m programs co-run on a k-core computer, each program is allocated k/m cores.

Fair but inefficient



Demand-aware Work-Stealing (DWS)

- Start from Equal-partitioning
- Dynamically balance cores at runtime
  - If p_i cannot fully-utilized a core, it release the core
  - If p_i has too many tasks, it tries to obtain more cores



Runtime Arch. of DWS

Stealing algorithm - (Release)

- A worker decides whether to release its core by itself

```
Algorithm 1: Work-stealing algorithm in DWS
  Input: w: current worker
int failed_steals = 0;            // num of failed steals
while work is not done do
    if w is free then
        if its task pool is not empty then
            w obtains a task t from its own task pool ;
            failed_steals = 0 ;
        else
            w randomly selects v as victim worker ;
            if v has a non-empty task pool then
                w steals t from v ;
                failed_steals = 0 ;
            else
                failed_steals ++ ;
                if failed_steals > T_SLEEP then
                    w goes to sleep ;
                    w waits to be woken up ;
                end if
            end if
        end if
        if t then
            w executes t ;
        end if
    end if
end while
```

If a worker fails too many times (T_SLEEP) to steal a new task, it goes to sleep

Coordinator - (Obtain)

- The coordinator decides whether to obtain more cores
  - If a program has too many queued tasks, it should try to get some free cores

How Many?

Which?

C1: The more queued tasks in a program, the more cores should the program obtain

C2: A program can take its allocated cores back

C3: A program cannot obtain the busy cores

Coordinator - How Many?

- C1: The more queued tasks in a program, the more cores should the program obtain

How many: $$N_w = \frac{N_b}{N_a}$$

Num of active workers	N_a
Num of queued tasks	N_b
Num of free cores	N_f
Num of released cores	N_r
Num of cores expected	N_w

Coordinator - Which?

- $N_w <= N_f$
  - Randomly select N_w free cores
- $N_f < N_w <= N_f + N_r$ **(C2)**
  - Select N_f free cores + its $(N_w - N_f)$ released core
- $N_w > N_f + N_r$ **(C3)**
  - N_f free cores+its N_r released cores

Num of active workers	N_a
Num of queued tasks	N_b
Num of free cores	N_f
Num of released cores	N_r
Num of cores expected	N_w

Evaluation platform

- A Dual-socket Quad-core computer with Hyper-Threading Technology

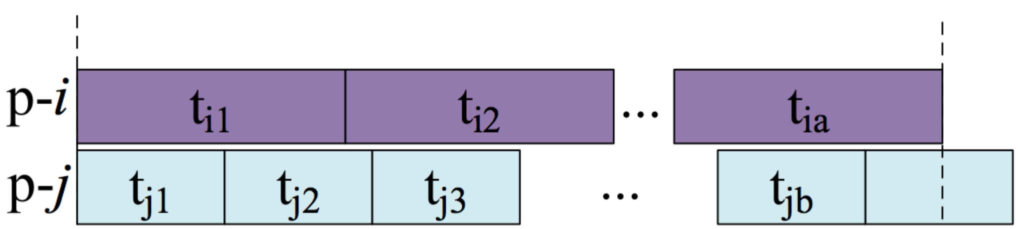
- Each socket is a Quad-Core Intel Xeon E5620

Hardware & Configuration	Size/Version
L1/L2 cache size (each core)	***256 KB/1MB***
L3 cache size (each socket)	***12 MB***
Main memory size	***32 GB***
Operation system	***Linux 2.6.32-38***

Benchmarks

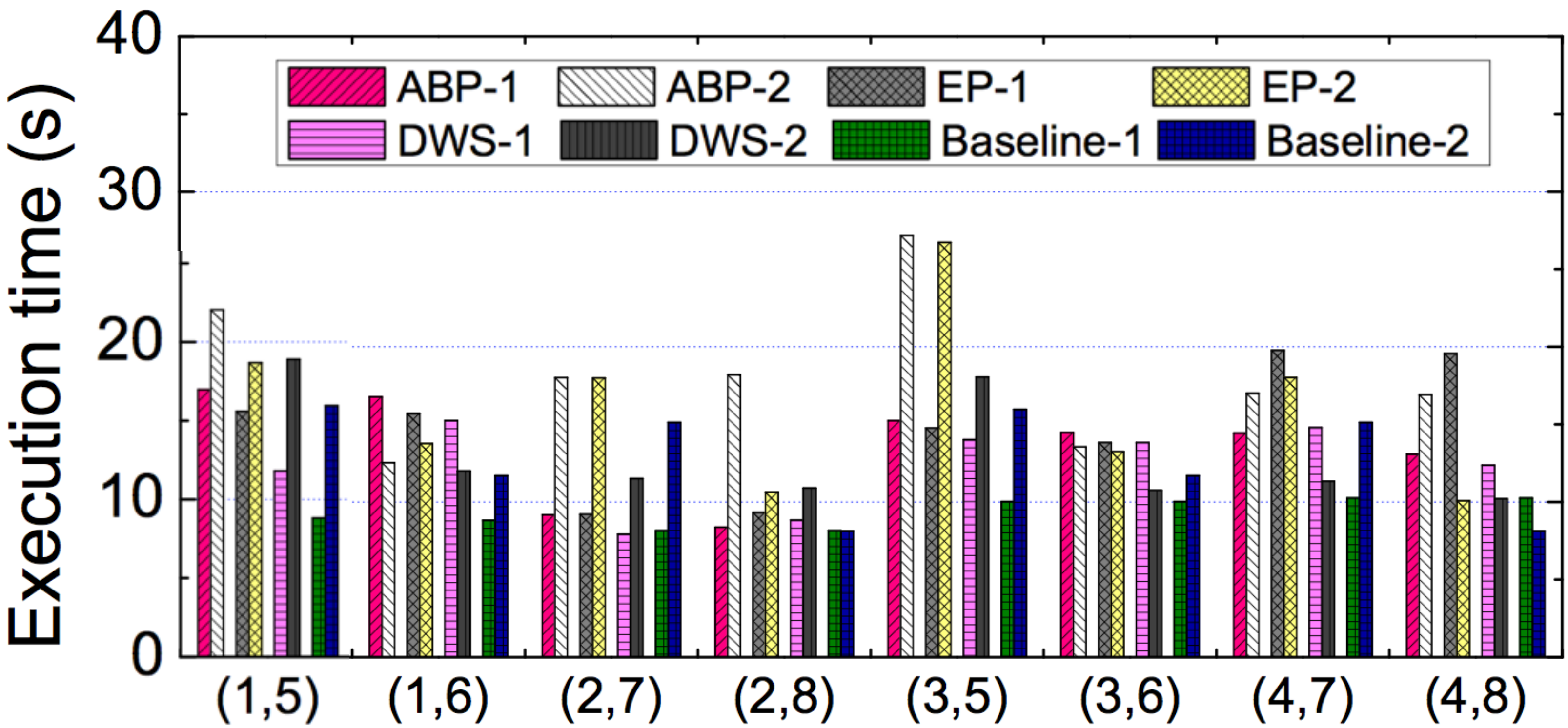
ID	Name	Description
p-1	FFT	Fast Fourier Transform
p-2	PNN	Polynomial Neural Network
p-3	Cholesky	Cholesky decomposition
p-4	LU	LU decomposition
p-5	GE	Gaussian Elimination algorithm
p-6	Heat	Five-point heat distribution
p-7	SOR	2D Successive Over-Relaxation
p-8	Mergesort	Merge sort on 4E6 numbers

Calculate execution time:



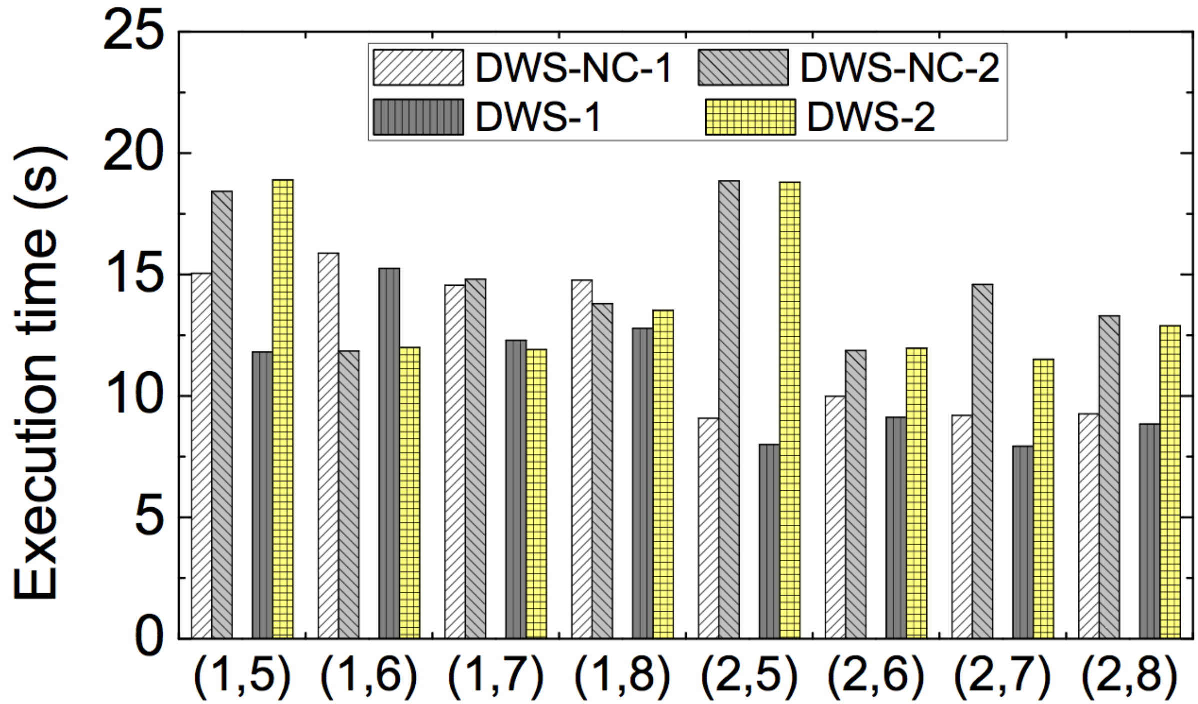
$$T_i = \frac{\sum_{r=1}^{a} t_{ir}}{a}, T_j = \frac{\sum_{r=1}^{b} t_{jr}}{b}$$

Performance of DWS



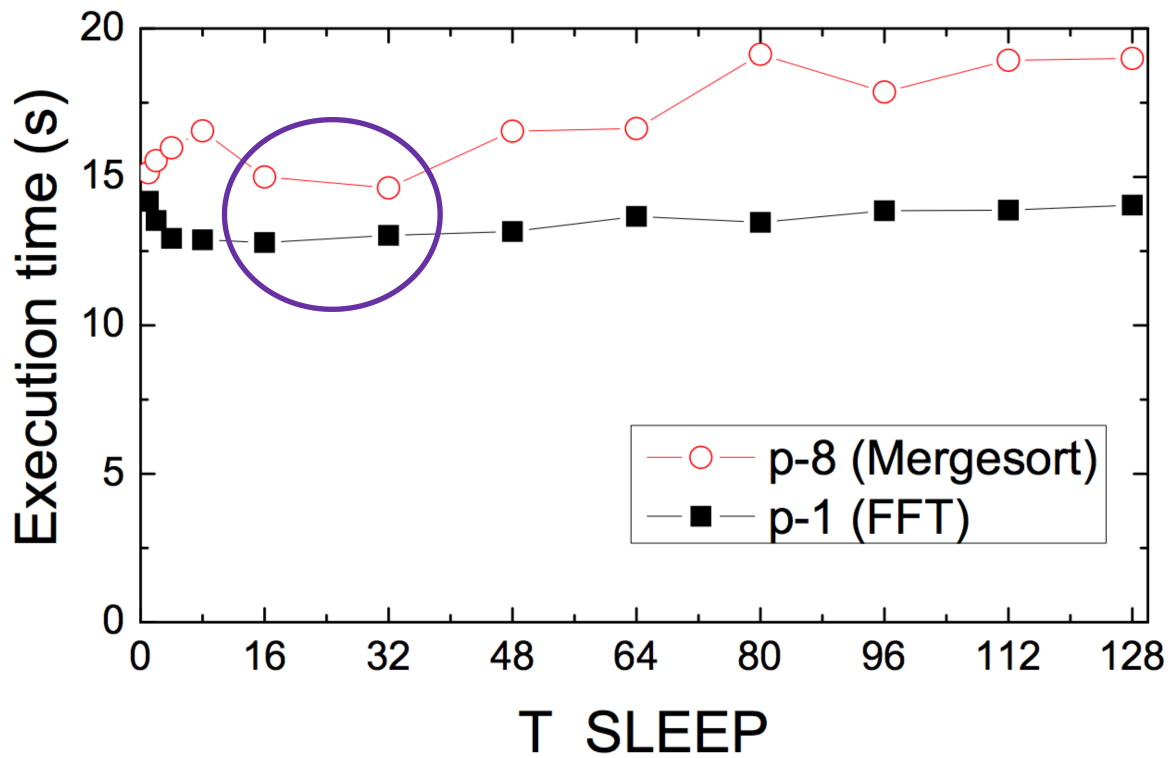
DWS can significantly improve the performance of the benchmarks

Effectiveness of the coordinator



Without the coordinator, the performance of the benchmarks is degraded

Impact of T_SLEEP



We should choose T_SLEEP = *k* or *2k* on a k-core computer

Contributions & conclusions

- A modified work-stealing algorithm that enables a program to release the under-utilized cores.
- A coordinator to manage the workers. It enables a program to grab and use the under-utilized cores released by other programs.
- We have implemented DWS, which achieves a performance gain of up to 32.3% in the best cases compared to traditional work-stealing schedulers.

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Thanks!

Questions?