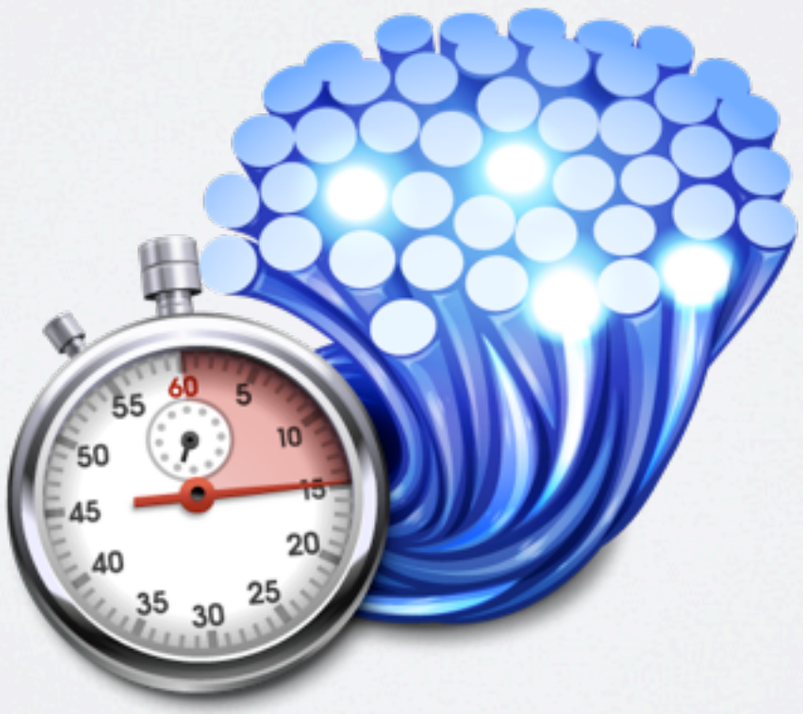
SCHEDULING THREADS FOR CONSTRUCTIVE CACHE SHARING ON CMPS

Shimin Chen et al.
CMU & Intel Pittsburgh



CONSTRUCTIVE CACHE SHARING

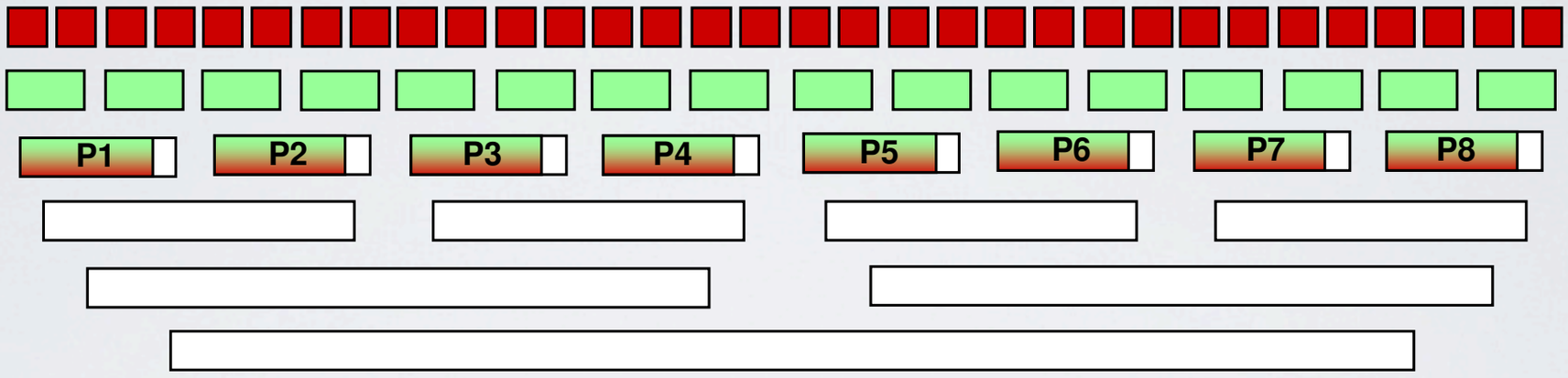
- mitigates the latency and bandwidth gap
- enables better use of on-chip real estate
- reduces power consumption

SCHEDULERS

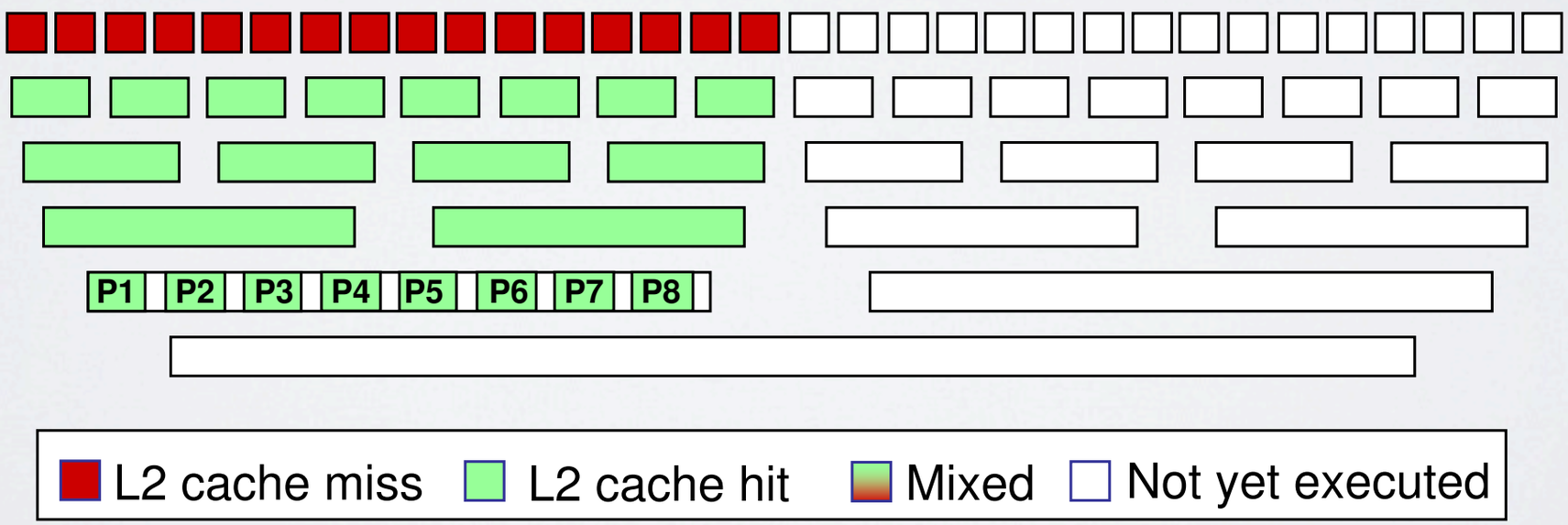
Work Stealing (WS)	Parallel Depth First (PDF)
greedy	greedy
local work queues	global work queue (?)
breadth first	depth first
good affinity among jobs executed by one core	good affinity among jobs executed simultaneously

MERGESORT

Work Stealing:

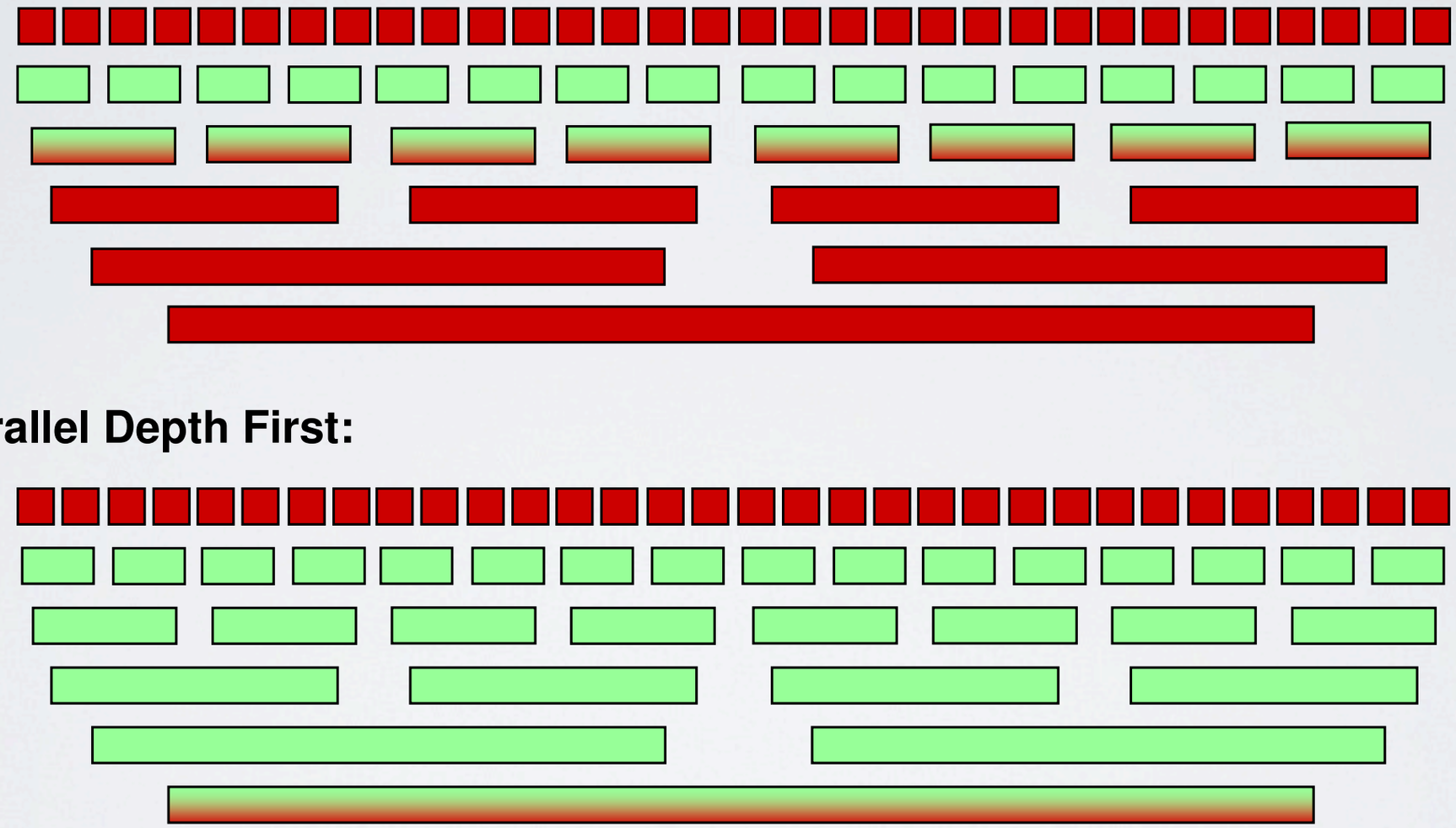


Parallel Depth First:



MERGESORT

Work Stealing:



Parallel Depth First:

BENCHMARKS

1. **LU:** matrix decomposition, scientific

   good cache reuse expected

2. **Hash Join:** database algorithm, large irregular data structure

   memory-bound

3. **Mergesort:** recursive divide-and-conquer

   depends on problem size and recursion strategy

CORE CONFIGURATIONS

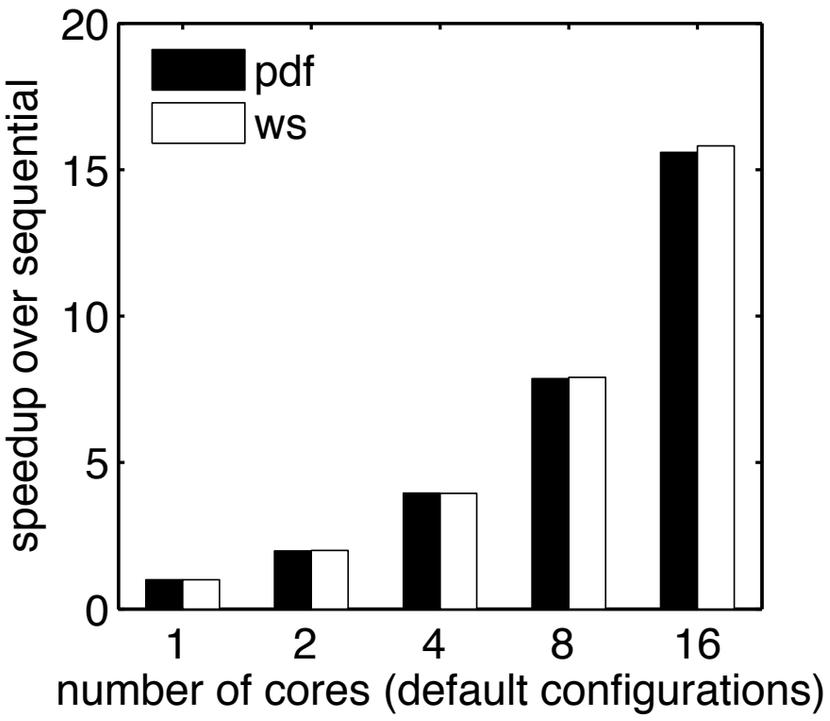
Table 1: Parameters common to all configurations.

Processor core	In-order scalar
Private L1 cache	64KB, 128-byte line, 4-way, 1-cycle hit latency
Shared L2 cache	128-byte line, configuration-dependent
Main Memory	latency: 300; service rate: 30 (cycles)

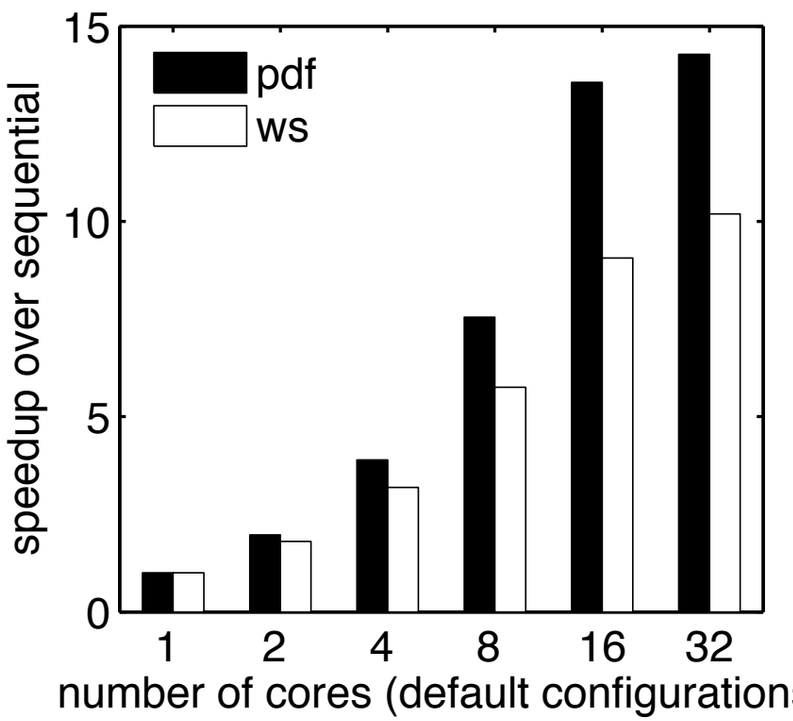
Table 2: Default configurations.

Number of cores	1	2	4	8	16	32
Technology (nm)	90	90	90	65	45	32
L2 cache size (MB)	10	8	4	8	20	40
Associativity	20	16	16	16	20	20
L2 hit time (cycles)	15	13	11	13	19	23

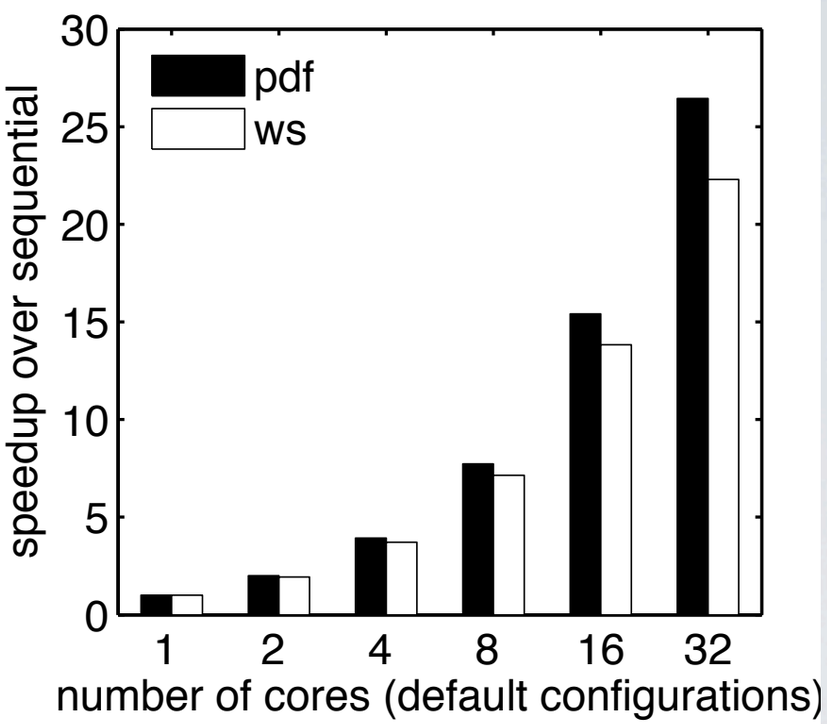
SPEEDUP



(a) LU



(c) Hash Join

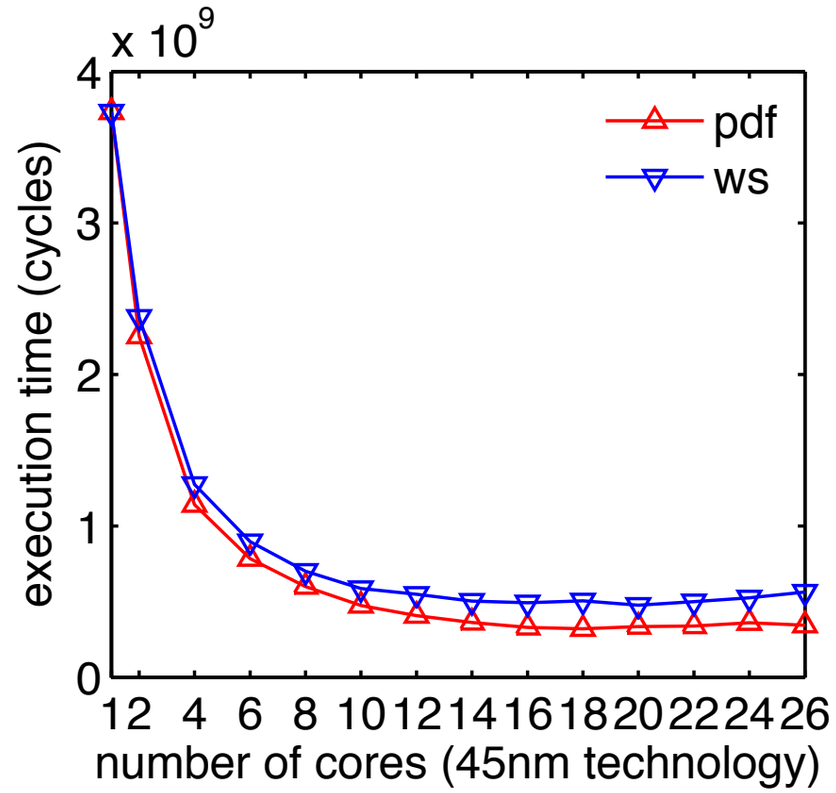


(e) Mergesort

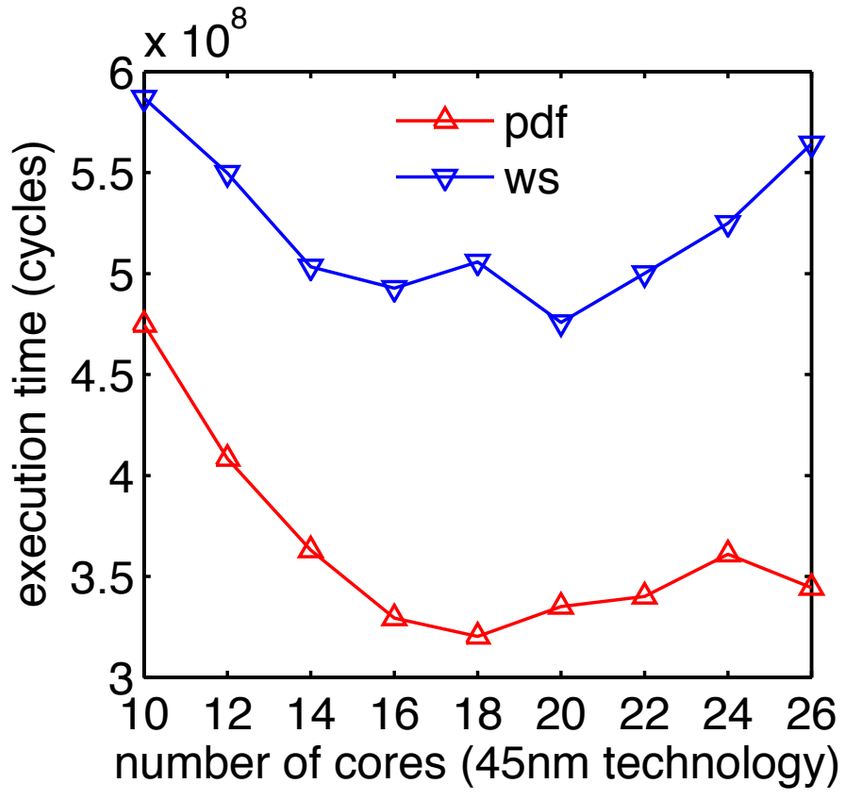
SINGLE TECHNOLOGY

Table 3: Single technology configurations with 45nm technology.

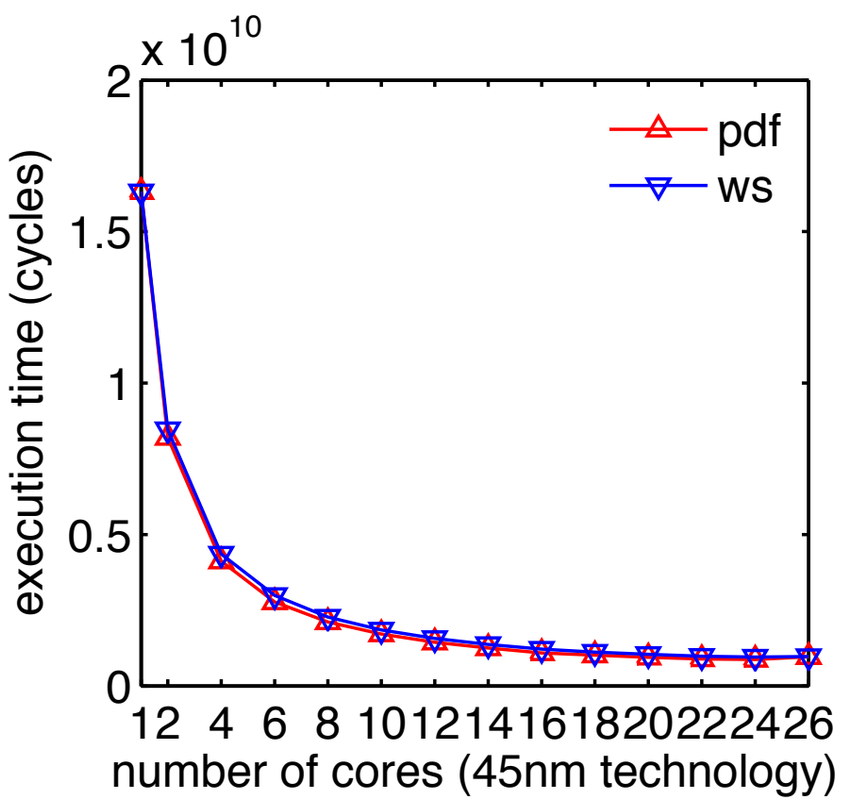
Number of cores	1	2	4	6	8	10	12	14	16	18	20	22	24	26
L2 cache size (MB)	48	44	40	36	32	32	28	24	20	16	12	9	5	1
Set associativity	24	22	20	18	16	16	28	24	20	16	24	18	20	16
L2 hit time (cycles)	25	25	23	23	21	21	21	19	19	17	15	15	13	7



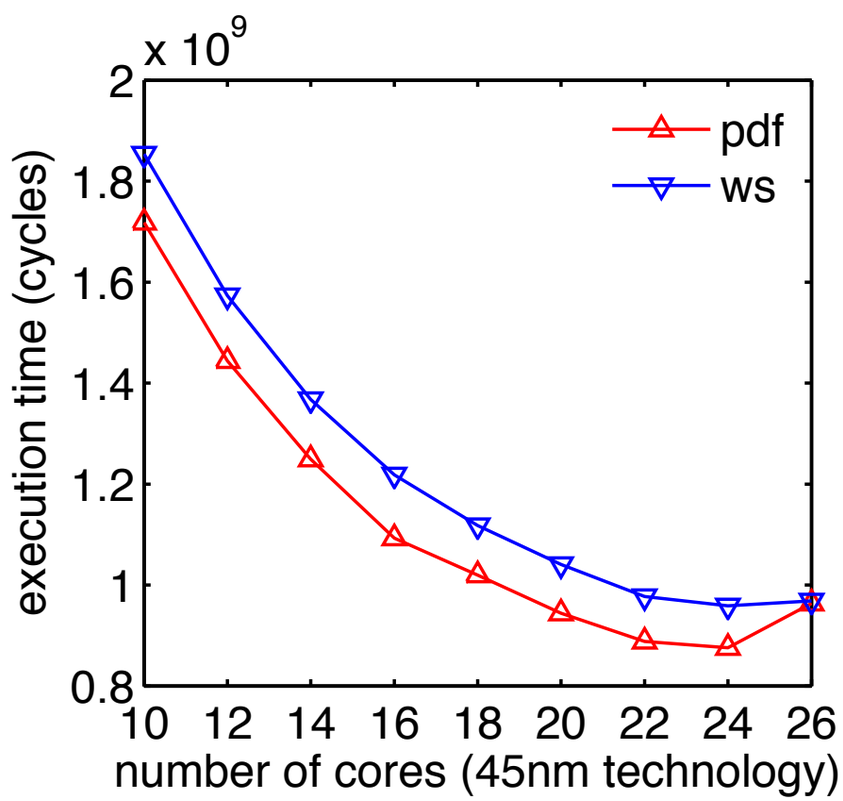
(a) Hash Join: execution time



(b) Hash Join: 10-26 cores

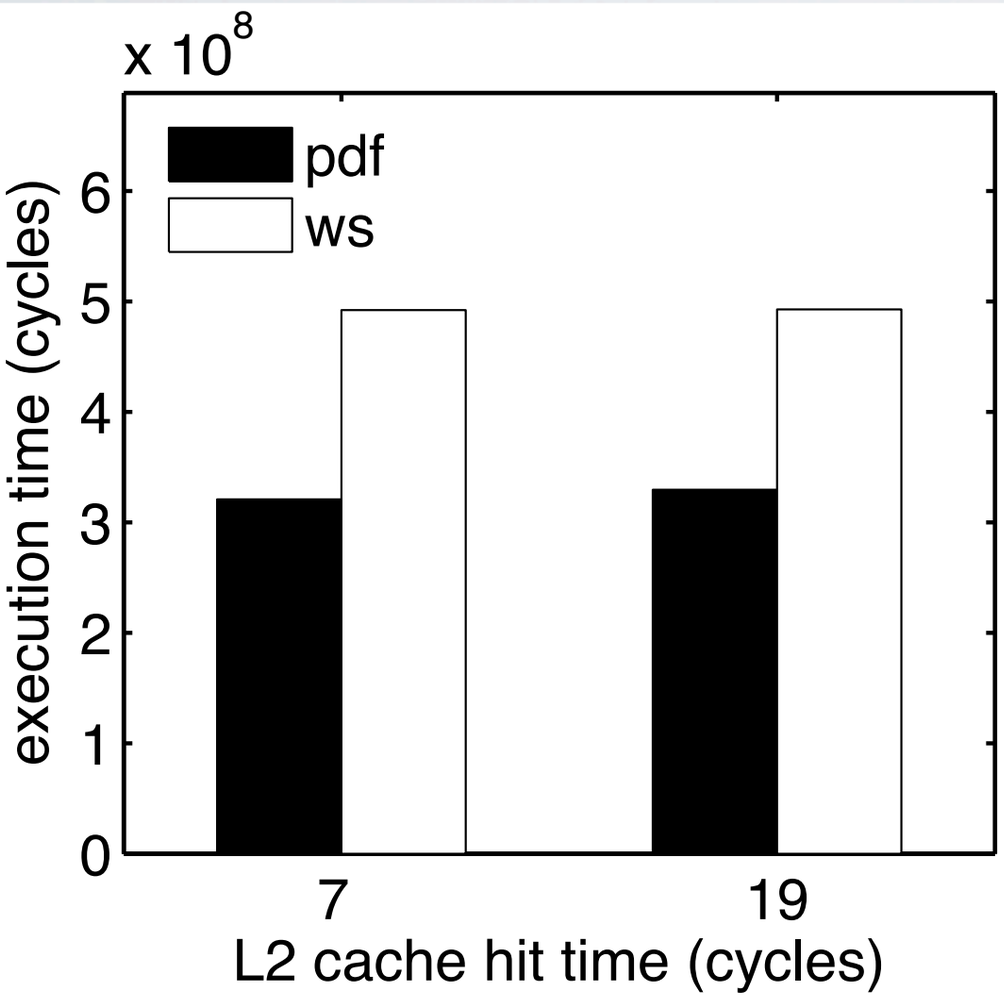


(c) Mergesort: execution time

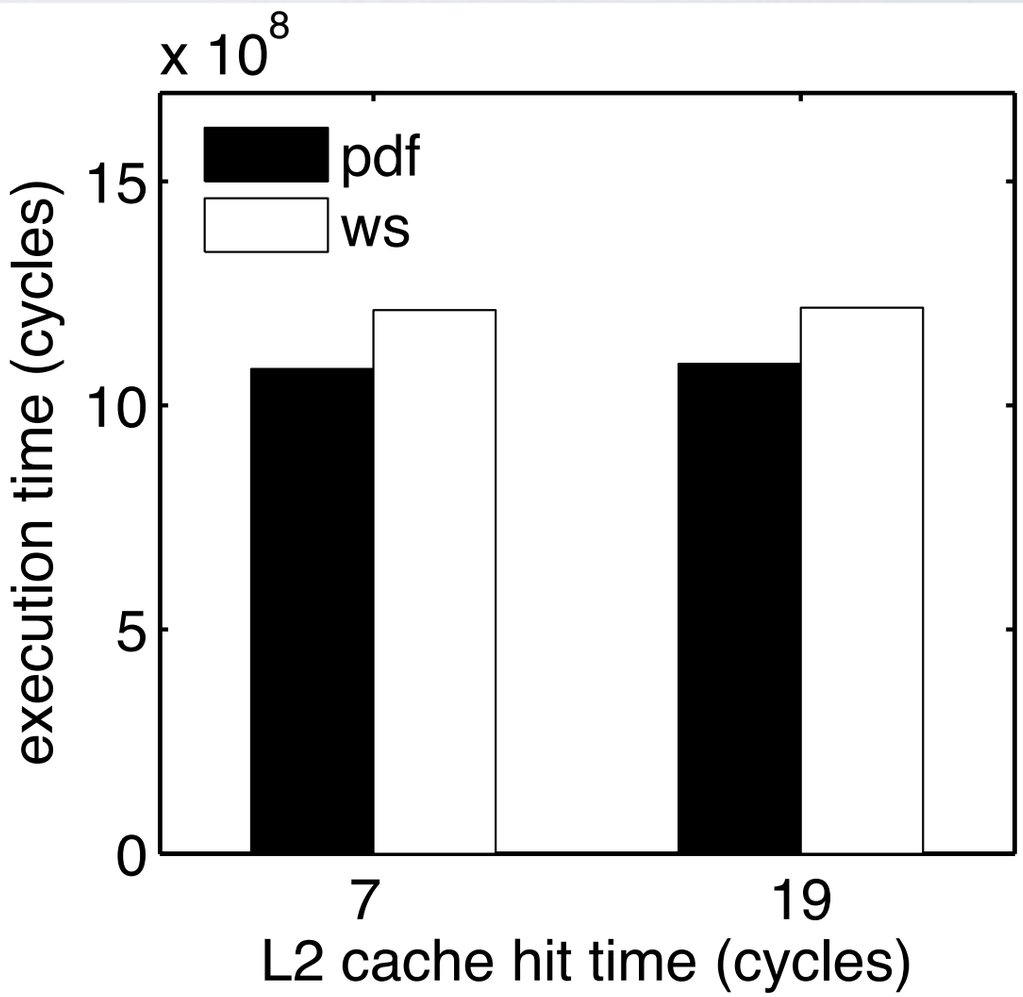


(d) Mergesort: 10-26 cores

CACHE HIT TIME SENSITIVITY

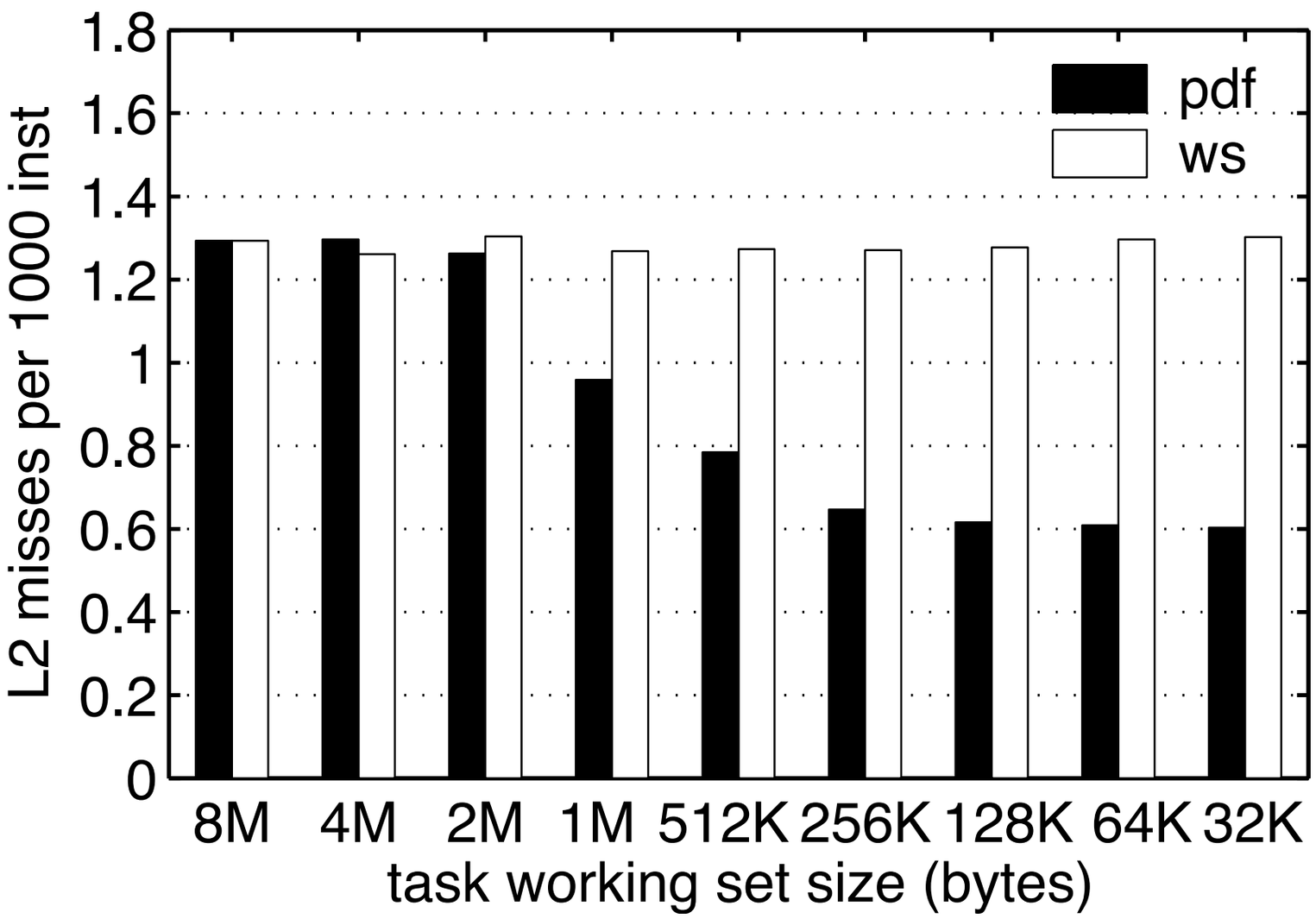


(a) Hash Join



(b) Mergesort

IMPACT OF THREAD GRANULARITY



ONE-PASS WORKING-SET PROFILING

- record a trace of all memory accesses
  - assumes an infinite cache?
- compute a 2D binning of cache accesses
  - cache age x job distance
- cache hits can be calculated for different cache sizes

AUTOMATIC TASK COARSENING

```
Function parallel_f(param) {
    If (Parallelize(param, __FILE__, __LINE__)) {
        Spawn (parallel_f(Subdivide(param, 1)));
        Spawn (parallel_f(Subdivide(param, 2)));
        ...
        Spawn (parallel_f(Subdivide(param, k)));
        Sync ();
        combine_results(param);
    } Else {
        sequential_f(param);
    }
}
```

`Parallelize()` decides based on threshold values for different cache size / core count configurations

CONCLUSION

- PDF matches or outperforms WS on a variety of configurations
- task granularity plays a key role in cache performance

DISCUSSION

- Who fully understood the automatic granularity selection?
- Is this a general approach or did the paper provide specific solution for hand-picked benchmark algorithms?
- Are there counter-examples where breadth-first is superior?
- Can we annotate jobs with metadata on their data access behavior? (Would benefit STM too.)