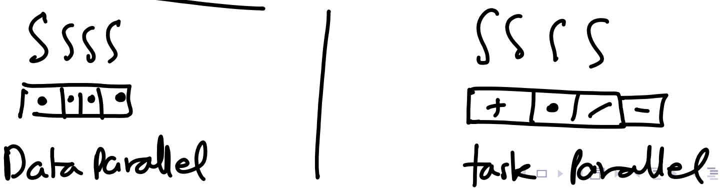
Work Sharing

- **Work Sharing**: General term describing distribution of work across threads
- Can be performed using three constructs:
  - **for** construct (for data parallelism)
  - **sections** construct (for task parallelism)
  - **tasks** construct (for irregular problems, e.g. unbounded loops, recursive codes)

Programmng models for threads. SIMD



Work Sharing

Matrix Multiplication → Posix

Matrix Multiplication → OpenMP

- **Work Sharing**: General term describing distribution of work across threads
- Can be performed using three constructs:
    - **for** construct (for data parallelism)
    - **sections** construct (for task parallelism)
    - **tasks** construct (for irregular problems, e.g. unbounded loops, recursive codes)

tasks → tricky

Data Parallelism

- Assuming that there is data independence across loop iterations, try:

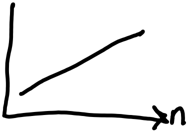
```
#pragma omp parallel [clauses]
{
    #pragma omp for [for clauses]
    for (loop control) {
        // statements
    }
}
```

→ Scheduling.

- OpenMP (or it's compilers) cannot (always) automatically identify data dependencies
- Threads "share" the iterations of the for loop
- Equivalent Code:

```
#pragma omp parallel for [for clauses]
for (loop control) {
    // statements
}
```

Data Parallelism (cont.)



2 : 20

1 : 10

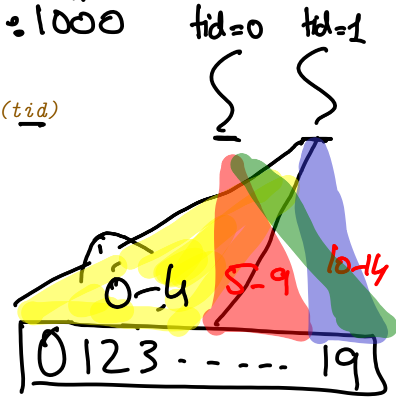
t : ω

n

1 : 1000

```
#pragma omp parallel num_threads(2) private(tid)
{
  tid = omp_get_thread_num();
  #pragma omp for
  for (i = 0; i < 20; i++) {
      printf("%3d by %3d\n", i, tid);
  }
}
```

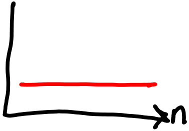
delay in picking work



Data Parallelism (cont.)

thread — workload.
@ compile time

```
int chunksize = 5;

#pragma omp parallel num_threads(2) private(tid)
{
    tid = omp_get_thread_num();
    #pragma omp for  schedule (static, chunksize)
    for (i = 0; i < 20; i++) {
        printf("%3d by %3d\n", i, tid);
    }
}
```



Data Parallelism (cont.)

Schedule Clauses (How loop iterations are mapped to threads)

Static Scheduling

- Low-overhead
- Load imbalance
- threads assigned “chunks” of iterations

Dynamic Scheduling

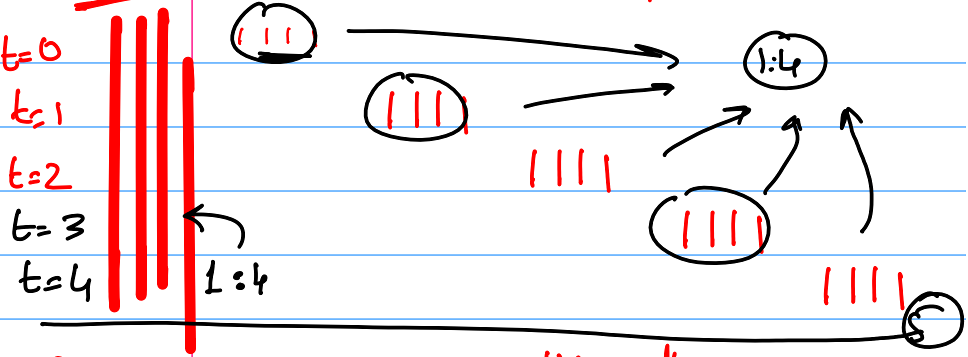
- High-overhead
- Reduces load imbalance
- Threads grab “chunks” of iterations

- *#pragma omp for schedule(static,chunksize)*
- *#pragma omp for schedule(dynamic,chunksize)*

6:24 = 1:4

Core=2

Static threds ← equally divided Workload



Dynamic threads ← Unequally dinded workload

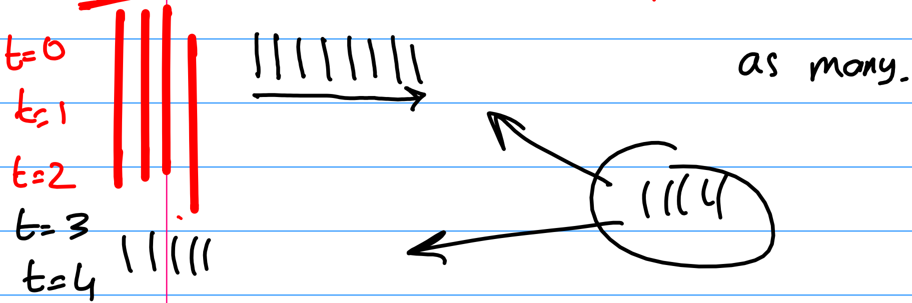
24

1 : 1

1 : 23

$\boxed{\text{Core} = 2}$

Static threads $\xleftarrow[\text{divided}]{\text{equally}}$ workload

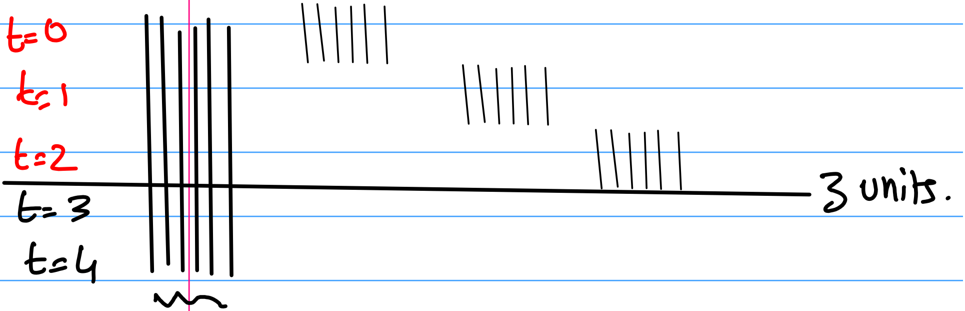


Dynamic threads $\xleftarrow[\text{divided}]{\text{unequally}}$ workload

6 : 24 = 1 : 6

$\boxed{\text{Core} = 2}$

Static threads ← equally div'd d workload



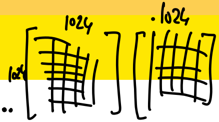
Dynamic threads ← unequally dinded workload

Static threds $\xleftarrow[\text{div'd d}]{\text{equally}}$ Workload

Dynamic threads $\xleftarrow[\text{divided}]{\text{Unequally}}$ workload

Data Parallelism (cont.)

- 1 — 1024
- 2 — 512
- 4 — 256
- 8 — 128
- 16 — 64
- 32 — 32
- 64 — 16
- 128 — 8
- 256 — 4
- 512 2
- 1024 1

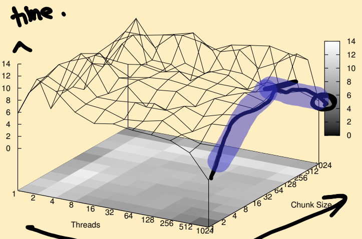


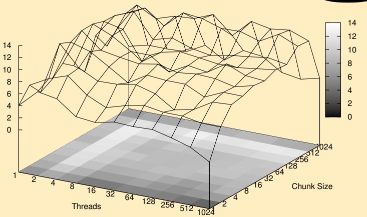
Effect of Chunk Size and Thread Quantities

- Computed 1024×1024 matrix multiplication using static and dynamic scheduling ?
- (Left) Static Scheduling (Right) Dynamic Scheduling

threads
Chunksize

better.





1024 threads
1 chunks

static

dynamic

Data Parallelism (cont.)

Is Schedule Class Really Necessary?

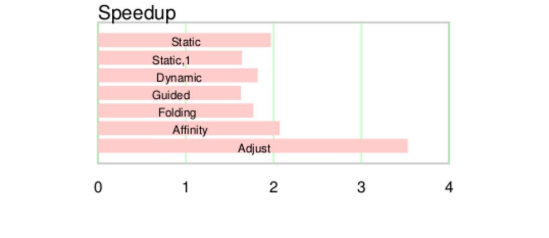


Fig. 1. Speedup for different schedules on a 4-way IBM Power4

- Ayguade et. al., Is the schedule clause really necessary in OpenMP?, Technical Report, Springer Verlag, 2003

Task Parallelism

- Considering following scenario:

```
p = pcibus();
n = networkCard(p);
w = wifiCard(p);
s = ssh(n,w);
h = http(n,w);
f = ftp(n,w);
```

- n, w can be executed in parallel
- s, h, and f can be executed in parallel
- Task Parallelism using **sections** in OpenMP:

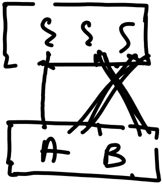
```
#pragma omp parallel [clauses]
{
    #pragma omp sections
    {
        #pragma omp section
        // Code of first task

        #pragma omp section
        // Code of second task
    }
}
```

num_threads (1)

← Sections

sharing th. work.

loop/I/O



Task Parallelism (cont.)

- **Sections** must be inside a parallel region. **Sections** itself provides enclosure for an individual **omp section**

```
p = pcibus();
#pragma omp parallel sections num_threads(2)
{
    #pragma omp section
    n = networkCard(p);
    #pragma omp section
    w = wifiCard(p);
}
#pragma omp parallel sections num_threads(3)
{
    #pragma omp section
    s = ssh(n,w);
    #pragma omp section
    h = http(n,w);
    #pragma omp section
    f = ftp(n,w);
}
```

- If no. of threads is $<$ no. of tasks, threads first attempt the beginning tasks before jumping to next ones.
- If no. of threads is $>$ no. of tasks, each task is performed by one thread, while the remaining remain idle