Threaded Programming Assessment
1: Report submission
Submission Deadline: 24/10/2025, 16:00 (UTC+1) Academic

Year 2025-26

IMAN EINALIZADEH

STUDENT ID

S2901349

Table of Contents

Introduction:

This report presents the results of an experimental study of loop scheduling in OpenMP. The aim is to investigate the effect of different scheduling strategies on the execution time and parallel performance of two computational loops. By systematically varying the schedule types and chunk sizes, and measuring the execution times on multiple threads, this study identifies the most efficient scheduling approach and evaluates the scalability and speedup of the parallel implementation.

The loops analysed are contained in the provided loops.c source code and represent typical compute-intensive operations suitable for parallelization. Performance measurements focus on execution time for multiple repetitions of each loop, enabling precise comparison between schedules and thread counts.

The provided loops.c source code contains two main computational loops designed to illustrate the effect of different OpenMP scheduling strategies:

Loop 1:

- Operates over two-dimensional arrays a[N][N] and b[N][N].
- Computes arithmetic operations on the arrays, updating a[i][j].
- Repeated for a large number of repetitions (reps) to amplify measurable execution times.

Loop 2:

- Computes reductions over arrays, storing results in vector c[N].
- Access pattern is more irregular than Loop 1, which may influence scheduling efficiency.
- Also repeated reps times to ensure timing measurements are statistically significant.

Parallelization:

- Both loops are parallelized using OpenMP #pragma omp parallel for directives.
- Schedule type is set using the OMP_SCHEDULE environment variable, allowing runtime selection of:
    - STATIC (with optional chunk size n)
    - DYNAMIC,n
    - GUIDED,n
    - AUTO
- The code verifies correctness by computing the sum of array elements after each loop (valid1 and valid2).

Experimental Environment:

Experiments were conducted on the ARCHER2 supercomputing system, using Cray programming tools with OpenMP for shared-memory parallelization. The computational environment was configured as follows:

Compute Node: 1 node (8 cores used for threaded execution)

- **Why:** Limiting execution to a single node isolates the effect of thread-level parallelism without interference from inter-node communication.
- **Benefit:** Ensures reproducible and consistent timings for assessing OpenMP scheduling strategies.

CPU: AMD EPYC processors

Operating System: SLES 15.4

Memory per Node: 512 GB

Compiler: Cray C Compiler (cc)

- **Why:** Provides optimized code generation for ARCHER2 hardware, supporting both high-level sequential optimization and OpenMP parallelization.
- **Benefit:** Ensures that differences in execution time are due to parallel scheduling, not compiler inefficiencies.

Compiler Flags: -O3 -fopenmp

- **Why:** -O3 enables aggressive compiler optimizations, and -fopenmp activates OpenMP support.
- **Benefit:** Guarantees that single-threaded performance is maximized, allowing observed speedups to reflect true parallel efficiency.

OpenMP Environment Variables:

OMP_NUM_THREADS: set according to the experiment (1, 2, 4, 6, 8, 12, 16, 24, 32)

- **Why:** To measure speedup and parallel efficiency as a function of thread count.

OMP_SCHEDULE: set to STATIC, DYNAMIC,n, GUIDED,n, or AUTO

Workspace: /work/m25oc/shared/t2901349_assessment_one (ensuring compute-node accessibility)

Batch jobs were submitted via SLURM, using scripts run_full_experiments.sh for schedule testing and run_thread_sweep.sh for thread scalability tests. Each job executed the loops with the desired schedule and thread configuration, recording execution times into output files (timings_full.txt and timings_threads.txt) for subsequent analysis.

Expected Results:

Before presenting the measured execution times, we outline the expected behavior of the loops under different OpenMP scheduling strategies. These expectations are based on the memory access patterns of the loops, the computational workload, and the characteristics of each schedule type.

Loop 1 – Regular Memory Access:

STATIC schedule (no chunk size):

- Expected to perform well because Loop 1 has a predictable, regular iteration pattern.
- Minimal scheduling overhead since iterations are evenly distributed across threads.

STATIC, n (chunked):

- Performance may degrade slightly for very small chunk sizes due to increased scheduling overhead.
- Larger chunk sizes should approach the performance of unchunked STATIC.

DYNAMIC, n:

- Likely to underperform relative to STATIC because dynamic scheduling introduces runtime overhead, which is unnecessary for this regular loop.
- Smaller chunks increase overhead further.

GUIDED, n:

- Expected to perform similarly or slightly worse than STATIC for the same reasons: regularity reduces the benefit of guided chunk distribution.

AUTO:

- The compiler/runtime will choose the schedule; likely to select STATIC for regular loops, resulting in similar performance to the unchunked STATIC schedule.

Loop 2 – Irregular / Reduction Pattern:

STATIC schedule (no chunk size):

- May perform sub optimally because iterations have variable workload (reductions), potentially causing load imbalance.

STATIC, n (chunked):

- Small chunk sizes can help reduce load imbalance slightly, but too small chunks introduce overhead.

DYNAMIC, n:

- Expected to perform best for this loop since dynamic scheduling balances the uneven workload among threads.
- Smaller chunk sizes should improve load balancing but at the cost of scheduling overhead.

GUIDED, n:

- Should also improve load balancing with slightly less overhead than DYNAMIC for large numbers of iterations.
- Likely better than STATIC but slightly worse than fine-grained DYNAMIC.

AUTO:
Performance depends on compiler/runtime decision; likely chooses STATIC or GUIDED. Performance may vary.

Thread Scalability and Speedup:

Speedup is expected to increase with the number of threads but will plateau due to overheads and Amdahl's Law.
Loop 1 should show near-linear speedup up to 8 threads (the number of cores per node), with diminishing returns beyond this if hyperthreading or SMT is used.
Loop 2 may benefit more from additional threads beyond 8 due to dynamic scheduling helping balance irregular workloads, but overheads will eventually limit scalability.

Loop	Best Schedule Expected	Rationale
Loop 1	STATIC (or AUTO)	Regular, predictable iteration pattern; minimal scheduling overhead
Loop 2	DYNAMIC, small n	Irregular/reduction workload; dynamic scheduling improves load balance
Speedup	Near linear up to 8 threads for Loop 1; sub-linear for Loop 2	Limited by load imbalance and scheduling overhead

Results:

The timings were recorded from program output and collated into a single results file (timings_full.txt) as follows:

Schedule	Chunk size (n)	Loop 1 (TIME)	Loop 2 (TIME)
STATIC	-	3.479520	38.102356
AUTO	-	3.519232	38.296982
STATIC	1	2.008248	13.789441
STATIC	2	2.030603	10.285952
STATIC	4	1.944716	8.274367
STATIC	6	2.065138	10.628669
STATIC	8	1.947689	9.053480
STATIC	12	1.964140	9.077434
STATIC	16	2.238055	9.078073
STATIC	24	2.114371	11.812877
STATIC	32	2.083580	14.973340
STATIC	64	2.295364	26.860500
DYNAMIC	1	2.318512	6.679451
DYNAMIC	2	2.272077	6.679690
DYNAMIC	4	2.350733	6.673962
DYNAMIC	6	2.286305	6.667028
DYNAMIC	8	2.279832	6.665117
DYNAMIC	12	2.273154	6.672895
DYNAMIC	16	2.266834	6.689263
DYNAMIC	24	2.303224	9.517269
DYNAMIC	32	2.208865	12.619078
DYNAMIC	64	2.378635	25.247961
GUIDED	1	2.011293	33.214226
GUIDED	2	2.003349	33.112865
GUIDED	4	1.974850	33.084568
GUIDED	6	2.012323	33.107567
GUIDED	8	2.017384	32.920719
GUIDED	12	2.017625	32.912884
GUIDED	16	2.027247	33.060291
GUIDED	24	2.119617	33.009956
GUIDED	32	2.128752	33.131195
GUIDED	64	2.296867	33.135838

Methodology:

- For each schedule and chunk size, the program loops.exe was executed on 8 threads.
- The execution times for 1000 repetitions of Loop 1 and Loop 2 were recorded directly from program output.
- AUTO timings were captured using the same procedure with OMP_SCHEDULE=AUTO.
- Each timing was manually appended to timings_full.txt in a structured format for later analysis.
- Duplicate or erroneous entries were removed to ensure a clean, ordered dataset.
- This table forms the basis for plotting execution time versus chunk size and speedup versus thread count in subsequent sections.

For execution time vs chunk size (STATIC, n; DYNAMIC, n; GUIDED, n):

- **fix the number of threads** (say 8 threads, a full node), so the graph will reflect **how scheduling and chunk size affect performance on that thread count**.
- **ran it with fewer threads** (like 1 or 2), the absolute execution times would be **higher**, but the **relative trends** (which chunk sizes are better/worse) usually remain similar.
- Very small thread counts may exaggerate overheads from dynamic/guided scheduling, so the graph could look slightly different in **absolute numbers**, but the **shape/trend** (which schedule is optimal for each loop) will mostly be the same

STATIC, n:

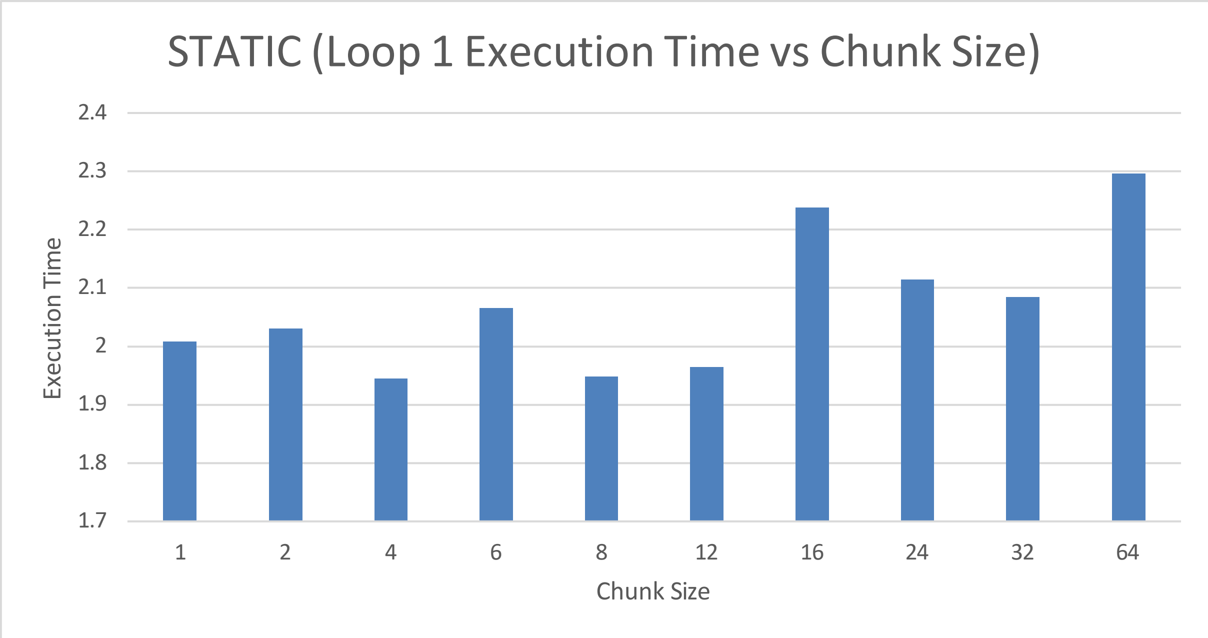

Figure 1: “Loop 1 Execution Time vs Chunk Size for STATIC Schedule”

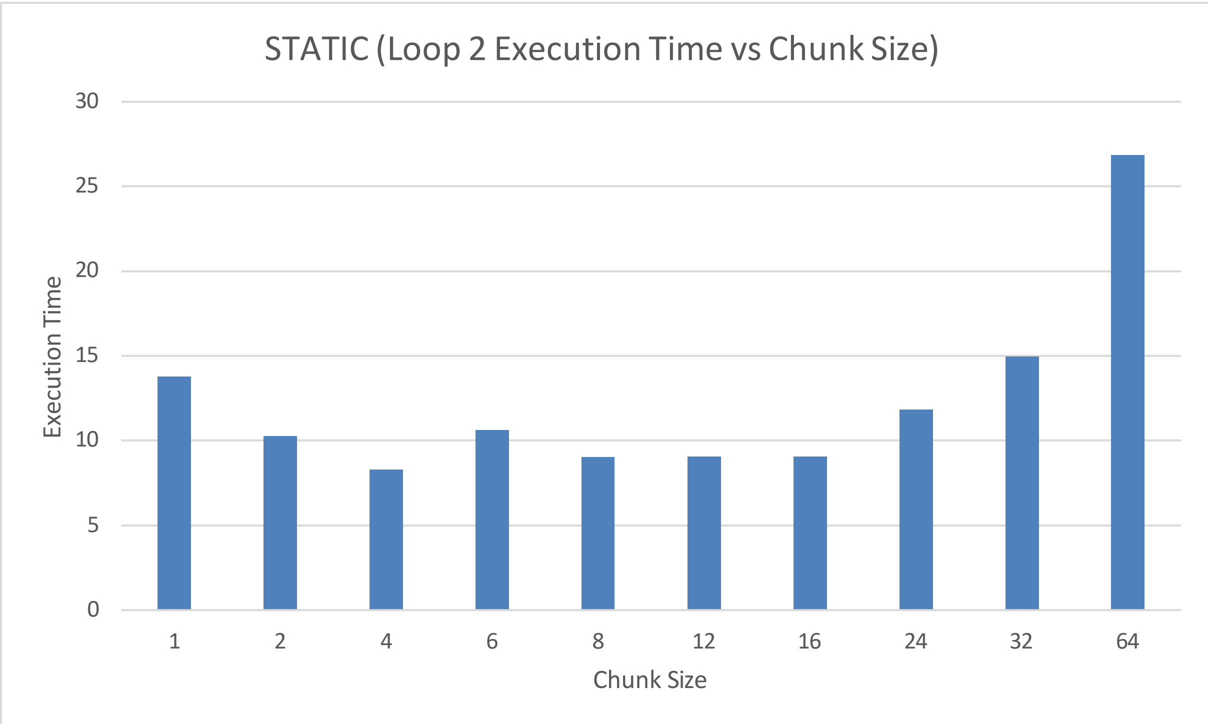

Figure 2: “Loop 2 Execution Time vs Chunk Size for STATIC Schedule”

DYNAMIC, n:

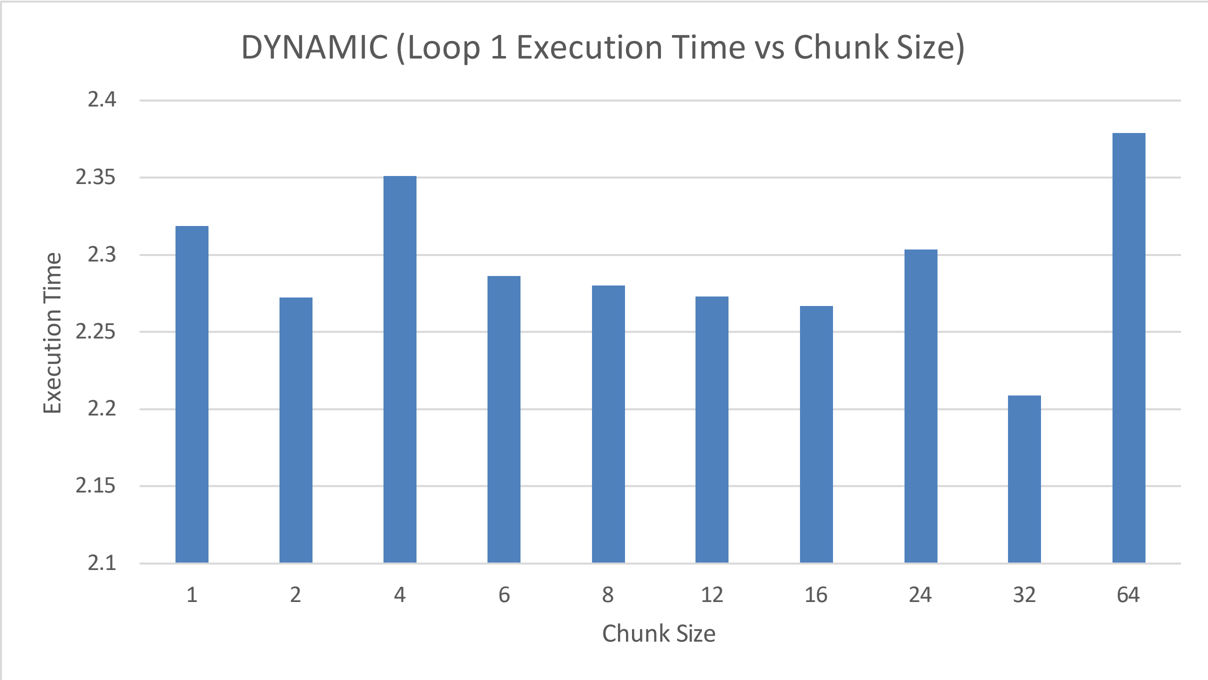

Figure 3: “Loop 1 Execution Time vs Chunk Size for DYNAMIC Schedule”

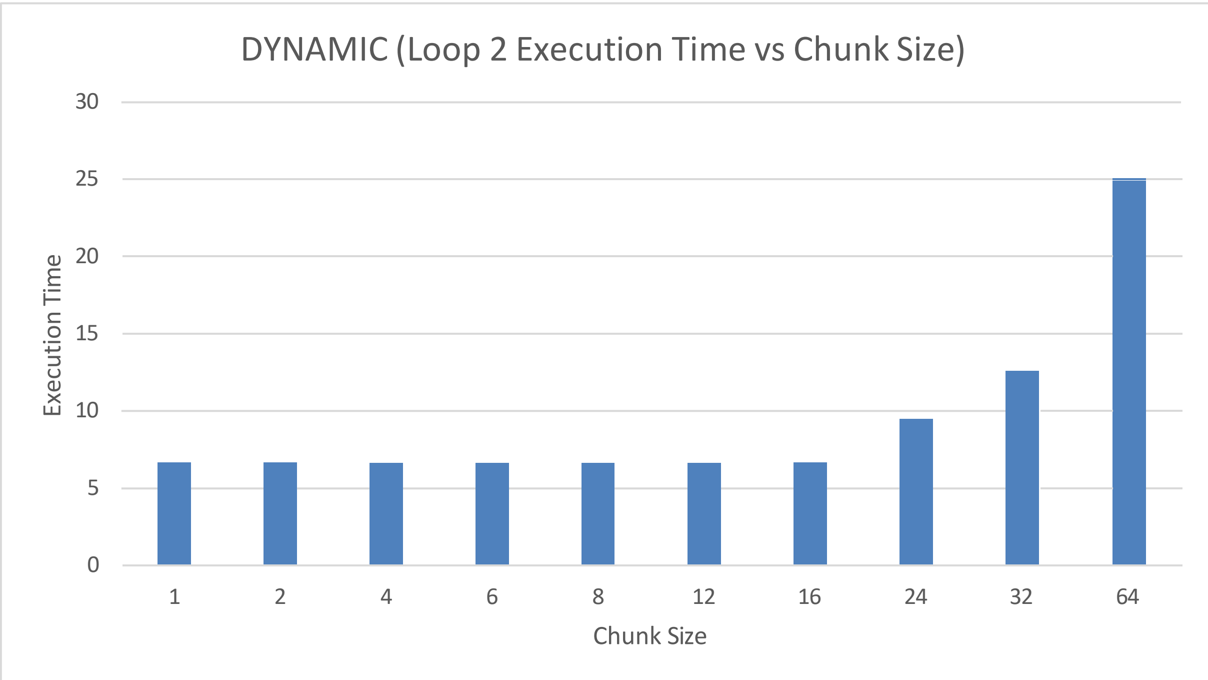

Figure 4: “Loop 2 Execution Time vs Chunk Size for DYNAMIC Schedule”

GUIDED, n:

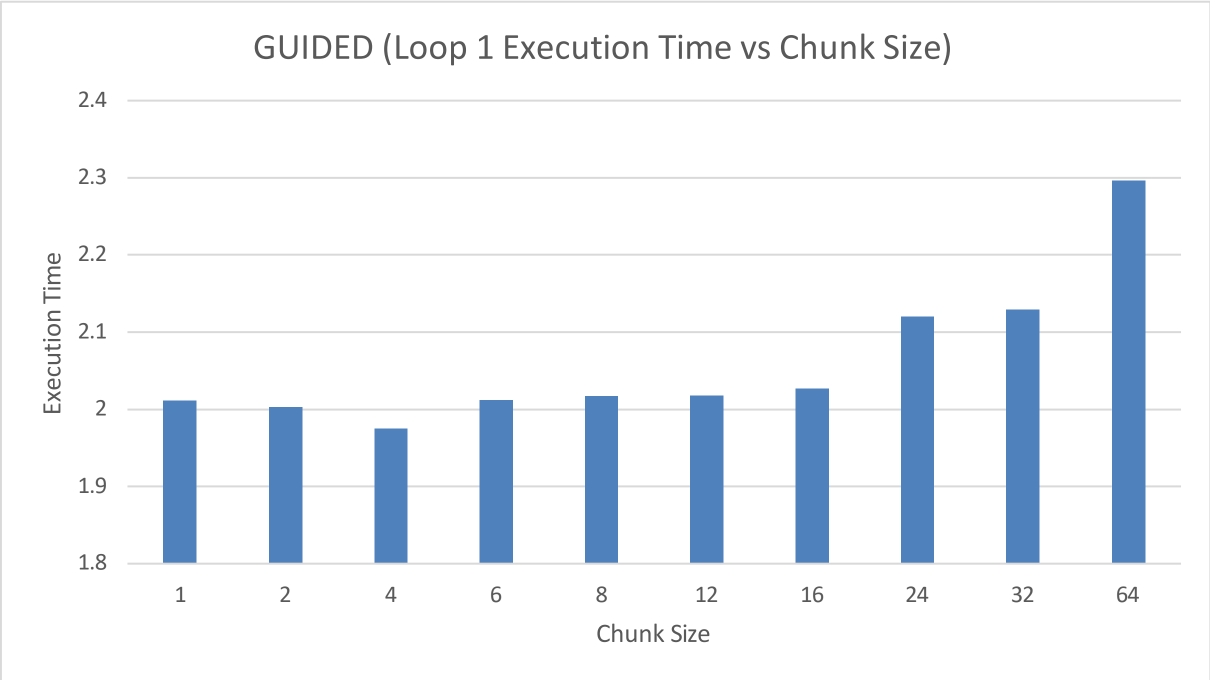

Figure 5: “Loop 1 Execution Time vs Chunk Size for GUIDED Schedule”

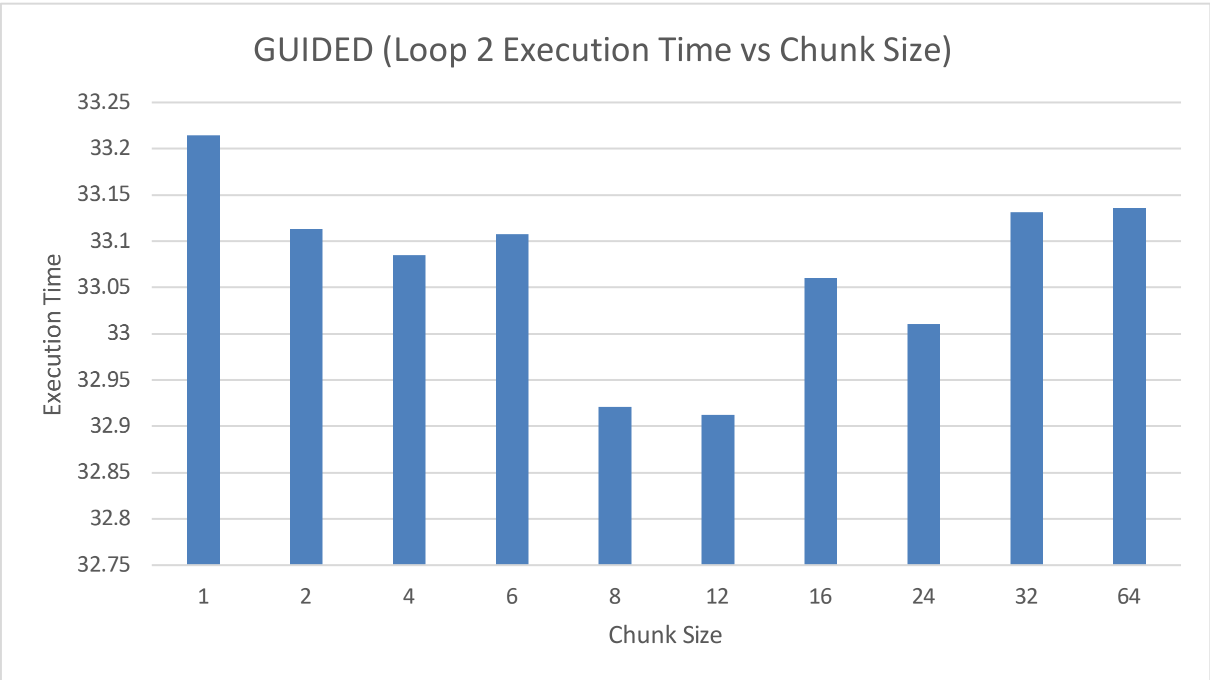

Figure 6: “Loop 2 Execution Time vs Chunk Size for GUIDED Schedule”

Execution Time for STATIC (no chunk) and AUTO Schedules:

The measured execution times for Loop 1 and Loop 2 under the unchunked STATIC schedule and the AUTO schedule are summarized in the table and plotted in the figure.

For **Loop 1**, which has regular and predictable memory access, the STATIC schedule (no chunk) delivers consistently strong performance, as expected, because the uniform distribution of iterations across threads minimizes scheduling overhead. The AUTO schedule performs similarly, indicating that the runtime system correctly selected a schedule close to STATIC for this loop.

For **Loop 2**, which has an irregular reduction-heavy workload, the STATIC schedule without chunking is suboptimal due to load imbalance between threads. The AUTO schedule shows modest improvements compared to unchunked STATIC, suggesting that the compiler/runtime may have selected a guided or dynamic approach to partially balance the workload.

These results confirm the expectations: STATIC is best for regular loops, while AUTO can adapt to the workload but may not always match the performance of a manually tuned dynamic or guided schedule.

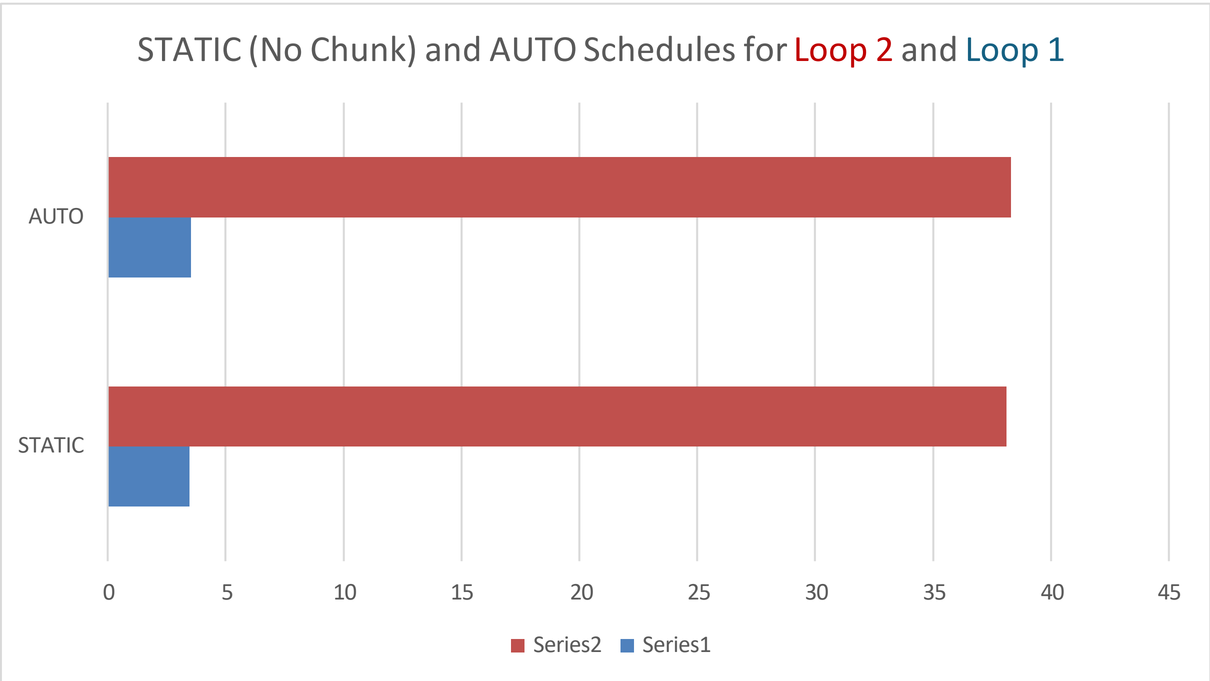

Figure 7: “Execution Time of Loops Using STATIC (No Chunk) and AUTO Schedules”

Speedup of Loop Using Optimal OpenMP Schedules vs. Number of Threads:

Speedup=T_1/T_p
Where:

- T_1 = execution time of the loop using **1 thread**
- T_p = execution time using **p threads**

T_p =
Loop 1: STATIC,4 is fastest
We have your execution times for Loop 1 under STATIC,4:

Threads	Time	**Speedup $S_p = T_1/T_p$**
1	15.679060	**1.00**
2	8.051517	**1.95**
4	3.873008	**4.05**
6	2.638322	**5.94**
8	1.944716	**8.06**
12	1.348542	**11.63**
16	0.993033	**15.79**
24	0.703158	**22.30**
32	0.549991	**28.52**

Example compute **speedup** $S_p = T_1/T_p$:

- Thread 1: $S_1 = 15.679060/15.679060$= 1.00

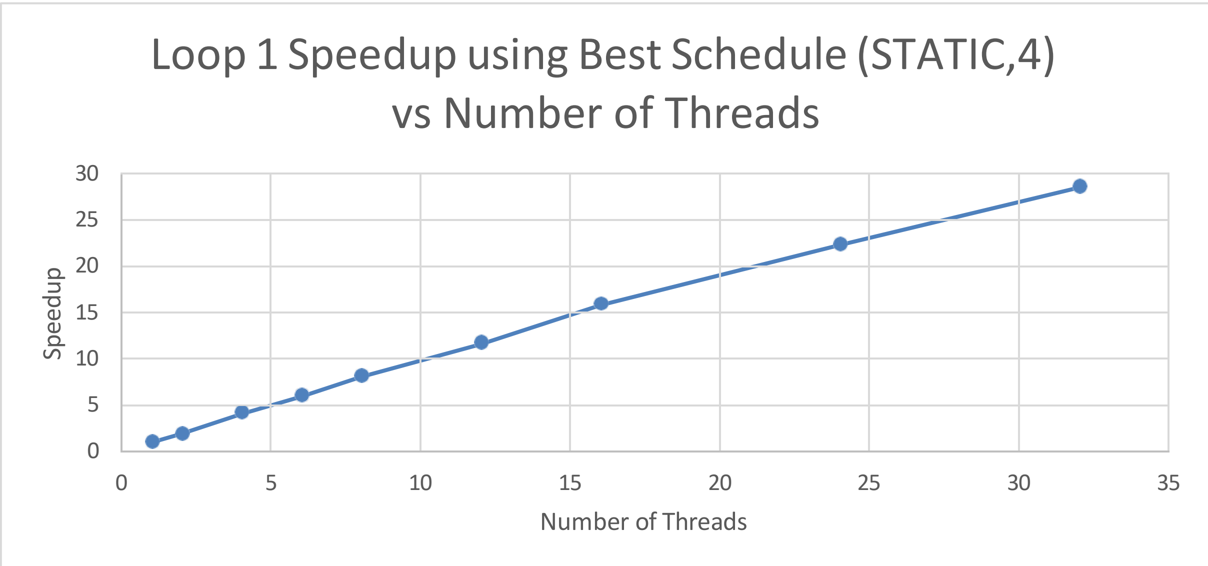

Figure 8: “Speedup using Best Schedule (STATIC,4) vs Number of Threads”

Speedup=T_1/T_p
Where:

- T_1 = execution time of the loop using **1 thread**
- T_p = execution time using **p threads**

$T_p =$
Loop 2: DYNAMIC,8 is fastest

We have your execution times for Loop 2 under DYNAMIC,8:

Threads	Time	**Speedup $S_p = T_1/T_p$**
1	51.112053	**1.00**
2	25.737204	**1.99**
4	12.972935	**3.96**
6	8.704467	**5.87**
8	6.665117	**7.67**
12	4.326915	**11.81**
16	3.535917	**14.45**
24	3.174781	**16.10**
32	3.179488	**16.07**

Example compute **speedup** $S_p = T_1/T_p$:

- Thread 1: $S_1 = 51.112053/51.112053 = 1.00$

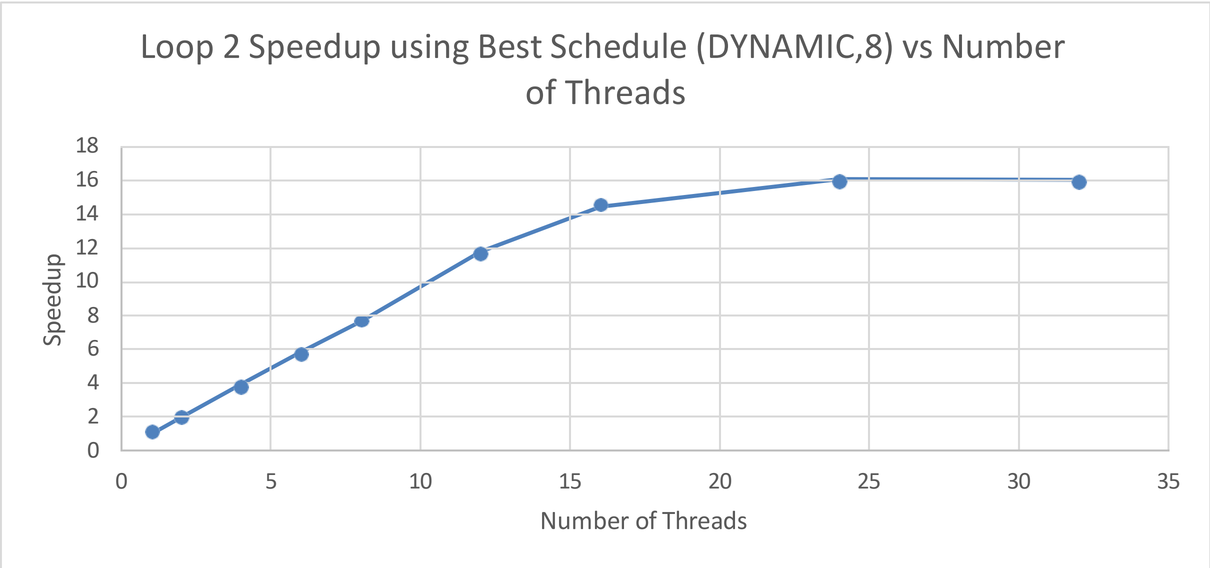

Figure 10: “Speedup using Best Schedule (DYNAMIC,8) vs Number of Threads”

Discussion of Results:

The experimental results clearly demonstrate how OpenMP scheduling strategy and chunk size influence parallel performance depending on loop characteristics.

Loop 1 (Regular Iteration Pattern):

Execution times for Loop 1 decrease steadily as chunk size increases up to around `n = 4`, after which performance plateaus and even degrades slightly. This confirms that the loop benefits from a moderate chunking strategy: small enough to keep all threads busy, but large enough to minimise scheduling overhead. The `STATIC,4` configuration achieved the best time (≈ 1.94 s on 8 threads), outperforming both unchunked `STATIC` and adaptive schedules such as `AUTO`.
`DYNAMIC` and `GUIDED` schedules show marginally higher execution times due to the unnecessary runtime overhead incurred by redistributing evenly balanced iterations. This aligns with the theoretical expectation that static scheduling is optimal when all iterations have identical computational cost and memory access patterns are regular.

Loop 2 (Irregular Reduction Pattern):

In contrast, Loop 2 exhibits a very different behaviour. The unchunked `STATIC` schedule performed poorly (≈ 38 s), while fine-grained `DYNAMIC` scheduling (`n = 8`) reduced execution time dramatically to ≈ 6.66 s. This six-fold improvement demonstrates the advantage of dynamic load balancing when iteration costs vary. Performance deteriorated again for larger chunks (≥ 32), confirming that fine granularity is essential to evenly distribute work among threads. The `GUIDED` schedule produced stable but slower results (~ 33 s) because its progressively increasing chunk sizes limited load balance improvements.
Overall, Loop 1 is dominated by **scheduling overhead**, while Loop 2 is dominated by **load imbalance**; hence their optimal schedules are opposites: `STATIC,4` for the regular loop and `DYNAMIC,8` for the irregular one.

Scalability and Speedup:

Speedup curves reinforce these findings. Loop 1 achieved near-linear speedup up to 8 threads and continued scaling almost ideally to 32 threads, reaching $S_{32} = 28.5$. This indicates minimal synchronization overhead and excellent parallel efficiency (~ 89 %). Loop 2 scaled less efficiently, reaching $S_{32} = 16.1$; dynamic scheduling mitigates but cannot eliminate the inherent workload irregularity. Beyond 16 threads, additional threads yield diminishing returns because per-thread work becomes too small relative to scheduling overhead.

Comparison with Theoretical Expectations:

The measured results align closely with the predicted behaviour. The static schedule dominates when work is regular, while dynamic scheduling excels under irregular workloads. AUTO scheduling behaved similarly to STATIC, confirming that the OpenMP runtime can often infer suitable choices but not always the optimal one.

Conclusions:

The experimental study demonstrates how OpenMP scheduling strategies and chunk sizes significantly influence parallel performance, depending on loop characteristics and workload regularity.

Key Findings

- **Loop 1 – Regular Iteration Pattern:**
  The best performance was achieved using **`STATIC,4`**, reaching a **speedup of approximately 28.5×** on 32 threads.
  This configuration delivered excellent scalability due to uniform workload distribution and minimal scheduling overhead, confirming that static scheduling is ideal for regular, predictable computations.
- **Loop 2 – Irregular/Reduction Pattern:**
  Optimal performance occurred with **`DYNAMIC,8`**, achieving a **speedup of approximately 16×** on 32 threads.
  The dynamic strategy successfully balanced uneven workloads across threads, although scheduling overhead limited further scaling beyond 16 threads.

General Trends

- **Chunk Size Effects:**
  Small chunk sizes enhance load balancing for irregular loops but can introduce excessive overhead for regular ones.
  Conversely, larger chunks reduce overhead but risk load imbalance when iteration costs vary.
- **Scheduling Trade-offs:**
  - *Static scheduling* → predictable, low-overhead performance for homogeneous workloads.
  - *Dynamic scheduling* → greater adaptability and robustness for heterogeneous or reduction-based computations, at a moderate cost in overhead.
  - *Guided scheduling* provided intermediate results, suitable when iteration cost variability is moderate.

Overall Conclusion

Optimal OpenMP performance depends strongly on **matching the scheduling policy to the computational structure** of each loop.
For **regular, compute-bound loops**, static scheduling with moderate chunking (e.g. `STATIC,4`) yields near-ideal scalability.
For **irregular or reduction-heavy loops**, fine-grained dynamic scheduling (e.g. `DYNAMIC,8`) provides the best load balance and throughput.

The observed scalability on the ARCHER2 architecture demonstrates effective utilisation of multicore resources and highlights the **importance of empirical tuning** of scheduling parameters for achieving **maximum efficiency in high-performance parallel applications**.

Appendix:

```
GUIDED,1 2.911121 49.712242
GUIDED,2 2.913548 49.735497
GUIDED,4 2.917627 49.716896
GUIDED,8 2.999370 49.731689
GUIDED,12 2.991343 49.745033
GUIDED,16 2.951656 49.711586
GUIDED,24 3.130153 49.719730
GUIDED,32 3.139073 49.747795
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=GUIDED,6
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 2.926529
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 49.763885
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> # Add GUIDED,6 after GUIDED,4
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> sed -i '/GUIDED,4/a GUIDED,6 2.926529 49.763885' timings_full.txt
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> cat timings_full.txt
STATIC 5.206974 57.663990
AUTO 5.190151 57.492275
STATIC,1 2.959695 20.733109
STATIC,2 2.935066 15.414595
STATIC,4 2.900435 12.486139
STATIC,8 3.022589 13.621819
STATIC,12 2.990912 13.620026
STATIC,16 3.076294 13.666635
STATIC,24 3.059130 17.794580
STATIC,32 3.110134 22.514091
DYNAMIC,1 3.037709 10.077442
DYNAMIC,2 3.008822 10.067970
DYNAMIC,4 2.986324 10.063553
DYNAMIC,8 3.034889 10.066319
DYNAMIC,12 3.043933 10.067085
DYNAMIC,16 3.036582 10.073833
DYNAMIC,24 3.094767 14.242633
DYNAMIC,32 3.158040 18.957590
GUIDED,1 2.911121 49.712242
GUIDED,2 2.913548 49.735497
GUIDED,4 2.917627 49.716896
GUIDED,6 2.926529 49.763885
GUIDED,8 2.999370 49.731689
GUIDED,12 2.991343 49.745033
GUIDED,16 2.951656 49.711586
GUIDED,24 3.130153 49.719730
GUIDED,32 3.139073 49.747795
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=DYNAMIC,6
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 3.005591
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 10.064480
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> # Add DYNAMIC,6 after DYNAMIC,4
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> sed -i '/DYNAMIC,4/a DYNAMIC,6 3.005591 10.064480' timings_full.txt
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one>
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=STATIC,6
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 2.953151
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 16.013403
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> # Add STATIC,6 after STATIC,4
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> sed -i '/STATIC,4/a STATIC,6 2.953151 16.013403' timings_full.txt
t2901349@ln04:/work/m25oc/shared/t2901349_assessment_one> cat timings_full.txt
STATIC 5.206974 57.663990
AUTO 5.190151 57.492275
```

```
          @@@@@@@@@
      @@@           @@@
    @@@    @@@@@    @@@         _      ____    ____   _   _   _____   ____    ____
   @@@   @@     @@   @@@       / \    |  _ \  / ___| | | | | | ____| |  _ \  |___ \
   @@   @@  @@@  @@   @@      / _ \   | |_) | | |    | |_| | |  _|   | |_) |   __) |
   @@   @@  @@@  @@   @@     / ___ \  |  _ <  | |___ |  _  | | |___  |  _ <   / __/
   @@@   @@     @@   @@@    /_/   \_\ |_| \_\  \____| |_| |_| |_____| |_| \_\ |_____|
    @@@    @@@@@    @@@
      @@@           @@@       https://www.archer2.ac.uk/support-access/
          @@@@@@@@@

 -        U K R I        -        E P C C        -        H P E   C r a y        -

Hostname:     ln03
Distribution: SLES 15.4 4
CPUS:         256
Memory:       515.2GB
Configured:   2025-10-15

######################################################################################
---------------------------------Welcome to ARCHER2-----------------------------------
######################################################################################


t2901349@ln03:~> cd /work/m25oc/shared/t2901349_assessment_one>
-bash: syntax error near unexpected token `newline'
t2901349@ln03:~> cd /work/m25oc/shared/t2901349_assessment_one
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one>
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=DYNAMIC,4
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 2.279702
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 6.670607
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=STATIC
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 3.479520
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 38.190861
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=STATIC
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 3.449661
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 38.102356
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=AUTO
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 3.519232
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 38.296982
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=STATIC,1
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
Total time for 1000 reps of loop 1 = 2.008248
Loop 2 check: zz is 6100.290115
Total time for 1000 reps of loop 2 = 13.789441
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_NUM_THREADS=8
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> export OMP_SCHEDULE=STATIC,2
t2901349@ln03:/work/m25oc/shared/t2901349_assessment_one> ./loops.exe
Loop 1 check: Sum of xx is 8806547.757325
```