

Walchand College of Engineering, Sangli
Department of Computer Science and Engineering
Name: Meet Vipul Gandhi

PRN: 2020BTECS00112

Class: Final Year (Computer Science and Engineering)

Year: 2023-24

Semester: 1

Course: High Performance Computing Lab

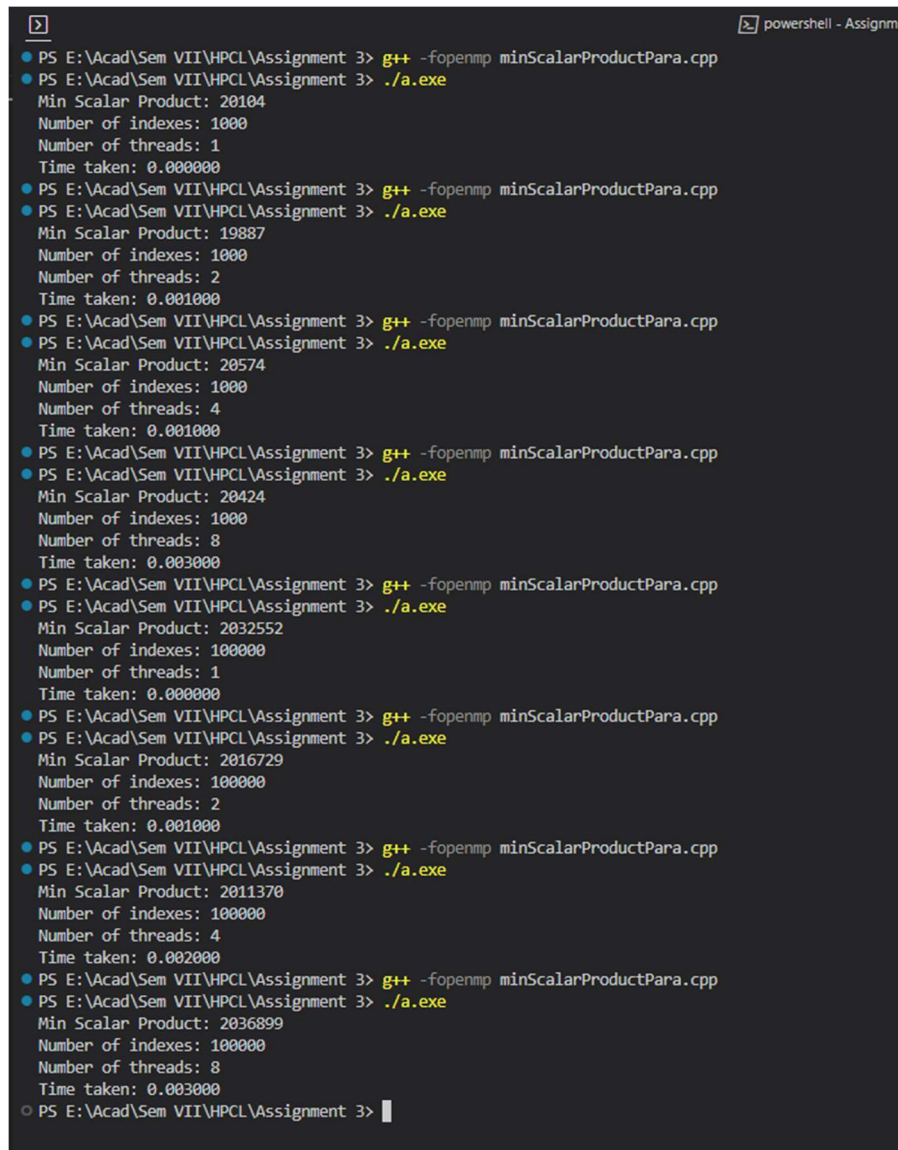
Practical No. 3

Title of practical:

Study and Implementation of schedule, nowait, reduction, ordered and collapse clauses

Problem Statement 1: Analyse and implement a Parallel code for below program using OpenMP. // C
Program to find the minimum scalar product of two vectors (dot product)

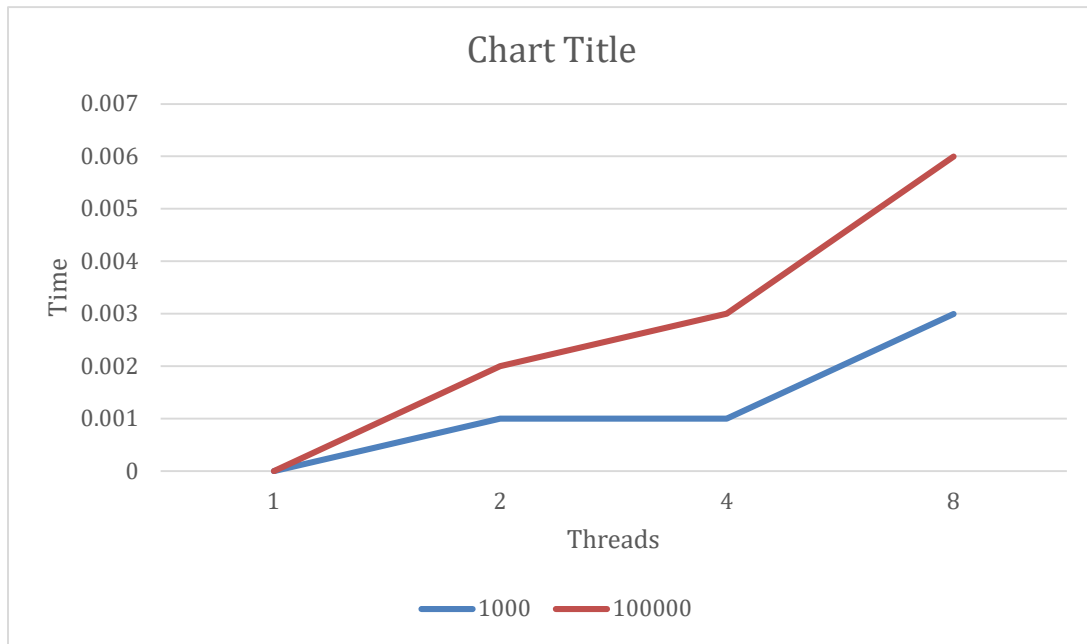
Screenshots:



```
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 20104
Number of indexes: 1000
Number of threads: 1
Time taken: 0.000000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 19887
Number of indexes: 1000
Number of threads: 2
Time taken: 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 20574
Number of indexes: 1000
Number of threads: 4
Time taken: 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 20424
Number of indexes: 1000
Number of threads: 8
Time taken: 0.003000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 2032552
Number of indexes: 100000
Number of threads: 1
Time taken: 0.000000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 2016729
Number of indexes: 100000
Number of threads: 2
Time taken: 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 2011370
Number of indexes: 100000
Number of threads: 4
Time taken: 0.002000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp minScalarProductPara.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Min Scalar Product: 2036899
Number of indexes: 100000
Number of threads: 8
Time taken: 0.003000
PS E:\Acad\Sem VII\HPCL\Assignment 3>
```

Information and analysis:

The outer loop iterates through the array, similar to the sequential bubble sort. Inside the outer loop, start is computed based on whether it's an even or odd pass. We then use ***#pragma omp parallel for*** to parallelize the inner loop. Each thread gets its own j, but they share the nums array. The inner loop logic remains the same as in the sequential bubble sort. Then we use ***#pragma omp parallel reduction(+:sum)*** to parallelize the for loop to calculate scalar product.



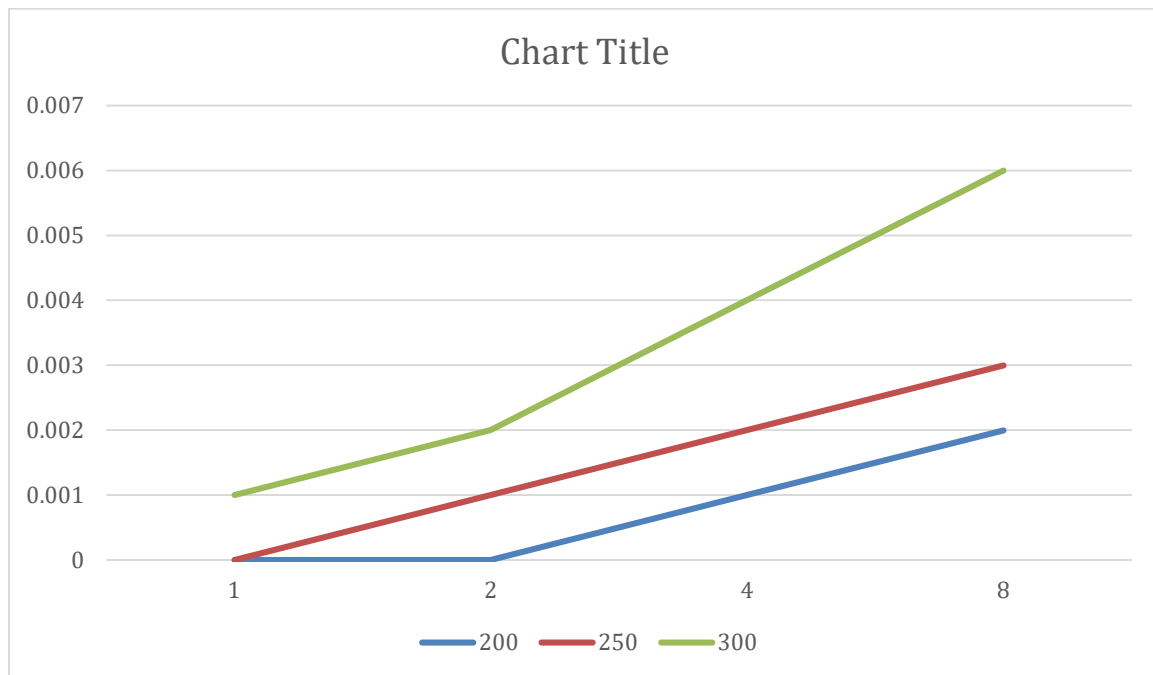
Vector Size			
Threads		1000	100000
1		0	0
2		0.001	0.001
4		0.001	0.002
8		0.003	0.003

Problem Statement 2: Write OpenMP code for two 2D Matrix addition, vary the size of your matrices from 250, 500, 750, 1000, and 2000 and measure the runtime with one thread (Use functions in C in calculate the execution time or use GPROF)

- For each matrix size, change the number of threads from 2,4,8, and plot the speedup versus the number of threads.
- Explain whether or not the scaling behaviour is as expected.

Screenshots:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp matrixAdditionSeq.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Number of Indexes: 300
Number of threads: 2
Time: 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp matrixAdditionSeq.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Number of Indexes: 300
Number of threads: 4
Time: 0.002000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp matrixAdditionSeq.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Number of Indexes: 300
Number of threads: 8
Time: 0.003000
PS E:\Acad\Sem VII\HPCL\Assignment 3> |
```



Size \ Threads	200	250	300
1	0	0	0.001
2	0	0.001	0.002
4	0.001	0.002	0.004
8	0.002	0.003	0.006

Information and analysis:

In this code, we have a function `matrix_addition` that performs the addition of two matrices. The main function iterates over different matrix sizes and different numbers of threads. It allocates memory for the matrices, initializes them with some values, and then measures the execution time of the matrix addition operation.

Collapse clause: The collapse clause is used to combine multiple loops into a single loop for parallelization. It can improve parallelization efficiency when dealing with nested loops. In this example, the two loops are combined into a single loop for parallelization.

Problem Statement 3: For 1D Vector (size=200) and scalar addition, Write an OpenMP code with the following: i. Use STATIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup. ii. Use DYNAMIC schedule and set the loop iteration chunk size to various sizes when changing the size of your matrix. Analyze the speedup. iii. Demonstrate the use of `nowait` clause.

Screenshots:

- Schedule clause with static chunk size

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_static.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Threads: 4
Time taken is 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_static.cpp
Threads: 4
Time taken is 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_static.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Indexes: 5000
Threads: 4
Time taken is 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_static.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Indexes: 5000
Threads: 8
Time taken is 0.001000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_static.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
Indexes: 5000
Threads: 1
Time taken is 0.000000
PS E:\Acad\Sem VII\HPCL\Assignment 3> |
```

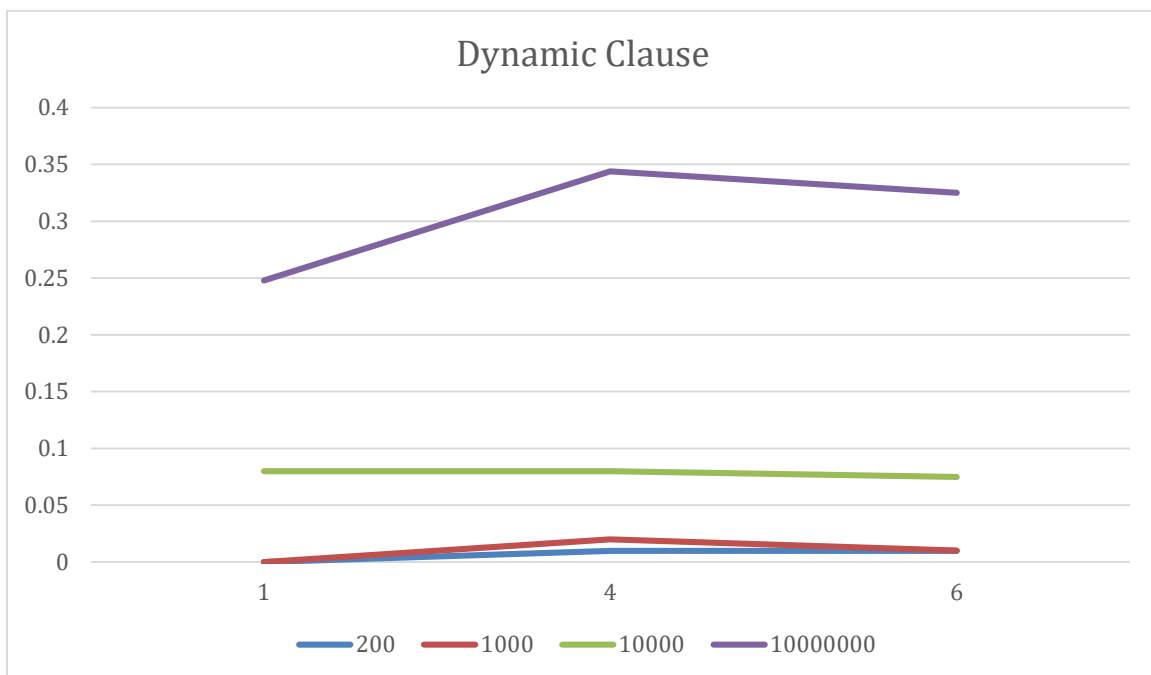
b. Schedule clause with dynamic chunk size

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_dynamic.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
• Indexes: 1000
• Threads: 1
Time taken is 0.002000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_dynamic.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
• Indexes: 200
• Threads: 4
Time taken is 0.003000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_dynamic.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
• Indexes: 200
• Threads: 8
Time taken is 0.002000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_dynamic.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
• Indexes: 1000
• Threads: 8
Time taken is 0.002000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_dynamic.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
• Indexes: 1000
• Threads: 4
Time taken is 0.004000
PS E:\Acad\Sem VII\HPCL\Assignment 3>

```



Size \ Threads	200	1000	10000	10000000
1	0	0	0.08	0.168
4	0.01	0.01	0.06	0.264
6	0.01	0	0.065	0.25

c. Nowait

```
powershell - Assignment 3
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_NoWait.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
i: 3, thread: 1
i: 4, thread: 1
i: 5, thread: 1
i: 9, thread: 3
i: 10, thread: 3
i: 11, thread: 3
i: 6, thread: 2
i: 7, thread: 2
i: 8, thread: 2
i: 0, thread: 0
i: 1, thread: 0
i: 2, thread: 0
Hello world
Hello world
Hello world
Hello world
Time: 0.002000
PS E:\Acad\Sem VII\HPCL\Assignment 3> g++ -fopenmp Q3_NoWait.cpp
PS E:\Acad\Sem VII\HPCL\Assignment 3> ./a.exe
i: 3, thread: 1
i: 4, thread: 1
i: 5, thread: 1
Hello world
i: 0, thread: 0
i: 1, thread: 0
i: 2, thread: 0
Hello world
i: 9, thread: 3
i: 10, thread: 3
i: 11, thread: 3
Hello world
i: 6, thread: 2
i: 7, thread: 2
i: 8, thread: 2
Hello world
Time: 0.003000
PS E:\Acad\Sem VII\HPCL\Assignment 3>
```

	Time
With NoWait Clause	0.003
Without NoWait Clause	0.002

Information and analysis:

- a. **Schedule Clause:** The schedule clause in OpenMP controls how loop iterations are divided among threads in a parallel loop. It has different scheduling options like *static*, *dynamic*, *guided*, and *auto*.
 - i. **Static scheduling:** The loop iterations are divided into chunks of size `chunk_size` and assigned to threads before execution. Each thread gets a fixed set of iterations at compile time.
 - ii. **Dynamic Scheduling:** In dynamic scheduling, the iterations are assigned to threads dynamically at runtime. Each thread receives a chunk of work, executes it, and then requests another chunk until all iterations are processed.
 - iii. **Guided Scheduling:** Guided scheduling is a hybrid approach that starts with large chunks and progressively reduces the chunk size as the loop progresses. It's particularly useful when you don't know the workload distribution in advance.
 - iv. **Auto Scheduling:** With auto scheduling, the compiler decides whether to use static, dynamic, or guided scheduling based on heuristics. This allows the compiler to choose the most suitable scheduling strategy for the specific loop and system.
- b. **Nowait Clause:** The nowait clause in OpenMP is used to indicate that a thread can proceed with its tasks without waiting for other threads. It's often used to overlap computations with other parallel regions. In above example we executed the program without using the nowait clause and we can see that helloworld function gets called after all the threads complete executing the for loop. In second run we add the nowait clause and we can see that helloworld function gets called as soon as the for-loop execution is completed.

GitHub Link: <https://github.com/meetgandhi692/HPC-Lab/tree/79b911b51b14aed89f30d1a35fe33b2761b4459f/Assignment%203>