CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

Data Decomposition and Reduction Using OpenMP

Please use Bridges-2 for this assignment. If you have trouble getting interactive access to a compute node using interact, you can submit batch jobs instead.

For our data decomposition example, we will parallelize Jacobi iteration using the OpenMP **parallel** and **for** directives.

Task 1. (10 pts) You are provided with a sequential Jacobi iteration code in **jacobi.c**. Compile the code by typing

gcc -fopenmp -o jacobi jacobi.c -lm

Then get on a compute node using interact and run the code by typing **./jacobi**

The reason we had to compile with -fopenmp, even though jacobi.c is a sequential code, is that we are using the OpenMP timing routines. Run three times and report the average sequential execution time. Be sure to type **exit** or **Ctrl-D** to exit the compute node when you are done.

```
[vazquezv@bridges2-login013 PA3]$ interact

A command prompt will appear when your session begins
"Ctrl+d" or "exit" will end your session

--partition RM-small,RM-shared
salloc -J Interact --partition RM-small,RM-shared
salloc: Pending job allocation 29413419
salloc: job 29413419 queued and waiting for resources
salloc: job 29413419 has been allocated resources
salloc: Granted job allocation 29413419
salloc: Waiting for resource configuration
salloc: Nodes r001 are ready for job
[vazquezv@r001 PA3]$ ./jacobi

  Jacobi iteration to solve A*x=b.

  Number of variables  N = 50000
  Number of iterations M = 10000

Time for jacobi iteration: 12.300812 seconds

  Part of final solution estimate:

        0              0
        1              0
        2              0
        3              0
        4              0
        5              0
        6              0
        7              0
        8              0
        9              0
...
    49989        45621.4
    49990        46018.4
    49991        46415.3
    49992          46813
    49993        47210.7
    49994        47608.9
    49995        48007.1
    49996        48405.7
    49997        48804.3
    49998        49203.2
    49999        49602.1

  Normal end of execution.
```

CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

```
[vazquezv@r001 PA3]$ ./jacobi

  Jacobi iteration to solve A*x=b.

  Number of variables  N = 50000
  Number of iterations M = 10000

Time for jacobi iteration: 12.531346 seconds

  Part of final solution estimate:

        0              0
        1              0
        2              0
        3              0
        4              0
        5              0
        6              0
        7              0
        8              0
        9              0
...
    49989         45621.4
    49990         46018.4
    49991         46415.3
    49992           46813
    49993         47210.7
    49994         47608.9
    49995         48007.1
    49996         48405.7
    49997         48804.3
    49998         49203.2
    49999         49602.1

  Normal end of execution.
```

```
[vazquezv@r001 PA3]$ ./jacobi

  Jacobi iteration to solve A*x=b.

  Number of variables  N = 50000
  Number of iterations M = 10000

Time for jacobi iteration: 12.468364 seconds

  Part of final solution estimate:

        0              0
        1              0
        2              0
        3              0
        4              0
        5              0
        6              0
        7              0
        8              0
        9              0
...
    49989         45621.4
    49990         46018.4
    49991         46415.3
    49992           46813
    49993         47210.7
    49994         47608.9
    49995         48007.1
    49996         48405.7
    49997         48804.3
    49998         49203.2
    49999         49602.1

  Normal end of execution.
[vazquezv@r001 PA3]$
```

The average sequential execution time is:

12.433507

CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

Task 2. (10 pts) You are provided with an OpenMP version of the Jacobi iteration code in **omp_jacobi.c**. The code has a single parallel region inside the iteration loop, with several parallel for loops inside that region. The code is correct in that all data sharing is done correctly and there are no race conditions or bad interleaving. Compile the code by typing

gcc -fopenmp -o omp_jacobi omp_jacobi.c -lm

If you are not still on a compute node, get on a compute node again by typing

interact --ntasks-per-node=2 -t 10:00

Set number of threads to 1 by typing

export OMP_NUM_THREADS=1

Run the code by typing **./omp_jacobi**
Set number of threads to 2 by typing

export OMP_NUM_THREADS=2

Run the code again by typing **./omp_jacobi**

Is the parallel code with one thread about as fast as the sequential code, or is it slower? Does the code speedup with 2 threads or does it slow down? Examine the code and explain the reason for the poor performance.

1. The parallel code with one thread is slower than the sequential version, taking 18.98s compared to ~12.4s.
2. With two threads, the execution time increases significantly to 52.44s, making it even slower.
3. The poor performance is due to parallel overhead, inefficient memory access, and synchronization costs.

CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

```
[vazquezv@r001 PA3]$ gcc -fopenmp -o omp_jacobi omp_jacobi.c -lm
[vazquezv@r001 PA3]$ interact --ntasks-per-node=2 -t 10:00

A command prompt will appear when your session begins
"Ctrl+d" or "exit" will end your session

--ntasks-per-node=2 --time=10:00
salloc -J Interact --ntasks-per-node=2 --time=10:00
salloc: Pending job allocation 29413480
salloc: job 29413480 queued and waiting for resources
salloc: job 29413480 has been allocated resources
salloc: Granted job allocation 29413480
salloc: Waiting for resource configuration
salloc: Nodes r001 are ready for job
[vazquezv@r001 PA3]$ export OMP_NUM_THREADS=1
[vazquezv@r001 PA3]$ ./omp_jacobi

JACOBI_OPENMP:
  C/OpenMP version
  Jacobi iteration to solve A*x=b.

  Number of variables  N = 50000
  Number of iterations M = 10000

Time for jacobi iteration: 18.977523 seconds

  Part of final solution estimate:

       0             0
       1             0
       2             0
       3             0
       4             0
       5             0
       6             0
       7             0
       8             0
       9             0
...
    49989        45621.4
    49990        46018.4
    49991        46415.3
    49992          46813
    49993        47210.7
    49994        47608.9
    49995        48007.1
    49996        48405.7
    49997        48804.3
    49998        49203.2
    49999        49602.1

JACOBI_OPENMP:
  Normal end of execution.
```

```
[vazquezv@r001 PA3]$ export OMP_NUM_THREADS=2
[vazquezv@r001 PA3]$ ./omp_jacobi

JACOBI_OPENMP:
  C/OpenMP version
  Jacobi iteration to solve A*x=b.

  Number of variables  N = 50000
  Number of iterations M = 10000

Time for jacobi iteration: 52.436927 seconds

  Part of final solution estimate:

       0             0
       1             0
       2             0
       3             0
       4             0
       5             0
       6             0
       7             0
       8             0
       9             0
...
    49989        45621.4
    49990        46018.4
    49991        46415.3
    49992          46813
    49993        47210.7
    49994        47608.9
    49995        48007.1
    49996        48405.7
    49997        48804.3
    49998        49203.2
    49999        49602.1

JACOBI_OPENMP:
  Normal end of execution.
[vazquezv@r001 PA3]$
```

CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

Task 3. (20 pts) Fix the performance problem with omp_jacobi.c that you discovered in Task 2 and recompile the code. Then get 16 cores on a compute node by typing

interact --ntasks-per-node=16 -t 30:00

Set OMP_NUM_THREADS appropriately to run with 2, 4, 8, and 16 threads. Report your times in a table. Graph the results showing the parallel speedup and comparing with linear speedup. Turn in your modified **omp_jacobi.c** along with your writeup.

2 threads

```
[vazquezv@r004 PA3]$ gcc -fopenmp -o omp_jacobi omp_jacobi.c -lm
[vazquezv@r004 PA3]$ interact --ntasks-per-node=16 -t 30:00

A command prompt will appear when your session begins
"Ctrl+d" or "exit" will end your session

--ntasks-per-node=16 --time=30:00
salloc -J Interact --ntasks-per-node=16 --time=30:00
salloc: error: QOSMaxSubmitJobPerUserLimit
salloc: error: Job submit/allocate failed: Job violates accounting/QOS policy (job submit limit, user's size and/or time limits)
[vazquezv@r004 PA3]$ export OMP_NUM_THREADS=2
[vazquezv@r004 PA3]$ ./omp_jacobi

JACOBI_OPENMP_OPTIMIZED:
C/OpenMP optimized version
Jacobi iteration to solve A*x=b.

Number of variables  N = 50000
Number of iterations M = 0

Time for jacobi iteration: 8.512331 seconds

Part of final solution estimate:
        0              0
        1              0
        2              0
        3              0
        4              0
        5              0
        6              0
        7              0
        8              0
        9              0
...
    49989        45621.4
    49990        46018.4
    49991        46415.3
    49992          46813
    49993        47210.7
    49994        47608.9
    49995        48007.1
    49996        48405.7
    49997        48804.3
    49998        49203.2
    49999        49602.1

JACOBI_OPENMP_OPTIMIZED:
Execution finished successfully.
```

CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

4 threads

```
[vazquezv@r004 PA3]$ export OMP_NUM_THREADS=4
[vazquezv@r004 PA3]$ ./omp_jacobi

JACOBI_OPENMP_OPTIMIZED:
C/OpenMP optimized version
Jacobi iteration to solve A*x=b.

Number of variables  N = 50000
Number of iterations M = 0

Time for jacobi iteration: 4.550709 seconds

Part of final solution estimate:
        0              0
        1              0
        2              0
        3              0
        4              0
        5              0
        6              0
        7              0
        8              0
        9              0
...
    49989        45621.4
    49990        46018.4
    49991        46415.3
    49992          46813
    49993        47210.7
    49994        47608.9
    49995        48007.1
    49996        48405.7
    49997        48804.3
    49998        49203.2
    49999        49602.1

JACOBI_OPENMP_OPTIMIZED:
Execution finished successfully.
```

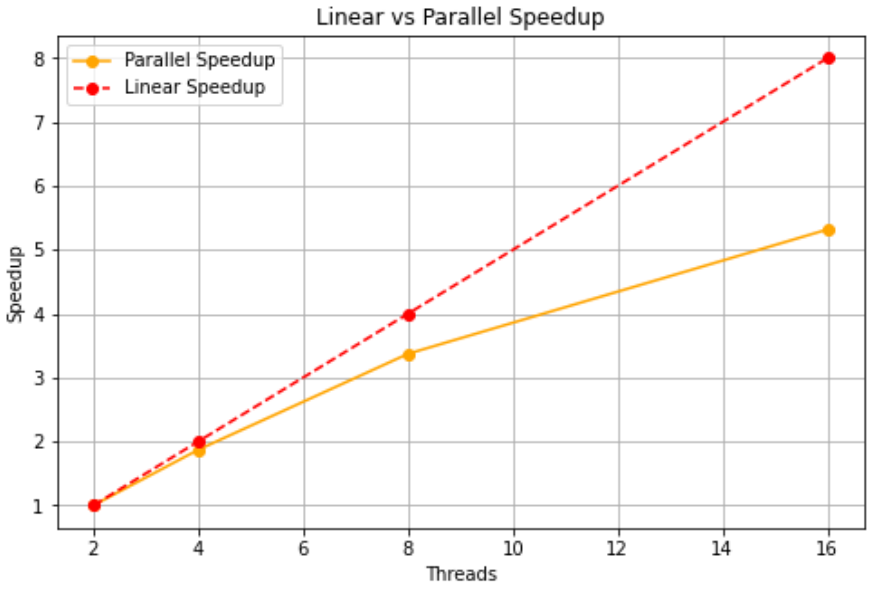
8 threads

```
[vazquezv@r004 PA3]$ export OMP_NUM_THREADS=8
[vazquezv@r004 PA3]$ ./omp_jacobi

JACOBI_OPENMP_OPTIMIZED:
C/OpenMP optimized version
Jacobi iteration to solve A*x=b.

Number of variables  N = 50000
Number of iterations M = 0

Time for jacobi iteration: 2.524449 seconds

Part of final solution estimate:
        0              0
        1              0
        2              0
        3              0
        4              0
        5              0
        6              0
        7              0
        8              0
        9              0
...
    49989        45621.4
    49990        46018.4
    49991        46415.3
    49992          46813
    49993        47210.7
    49994        47608.9
    49995        48007.1
    49996        48405.7
    49997        48804.3
    49998        49203.2
    49999        49602.1

JACOBI_OPENMP_OPTIMIZED:
Execution finished successfully.
```

CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

16 threads

```
[vazquezv@r004 PA3]$ export OMP_NUM_THREADS=16
[vazquezv@r004 PA3]$ ./omp_jacobi

JACOBI_OPENMP_OPTIMIZED:
C/OpenMP optimized version
Jacobi iteration to solve A*x=b.

Number of variables  N = 50000
Number of iterations M = 0

Time for jacobi iteration: 1.600584 seconds

Part of final solution estimate:
        0               0
        1               0
        2               0
        3               0
        4               0
        5               0
        6               0
        7               0
        8               0
        9               0
...
    49989         45621.4
    49990         46018.4
    49991         46415.3
    49992           46813
    49993         47210.7
    49994         47608.9
    49995         48007.1
    49996         48405.7
    49997         48804.3
    49998         49203.2
    49999         49602.1

JACOBI_OPENMP_OPTIMIZED:
Execution finished successfully.
[vazquezv@r004 PA3]$
```



Threads	Parallel Time Execution (s)	Parallel Speedup	Linear Speedup
2	8.512331	1.000000	1.0
4	4.550709	1.870551	2.0
8	2.524449	3.371956	4.0
16	1.600584	5.318266	8.0

CS 4175 Spring 2025
Shirley Moore, Instructor
Programming Assignment 3
50 points

One key optimization was replacing critical sections with OpenMP reductions. Previously, the use of #pragma omp critical introduced significant synchronization overhead, as threads had to wait for access to shared variables. By switching to reduction clauses, each thread accumulates results independently, and the final value is computed efficiently at the end, minimizing contention and improving parallel performance.

I also experimented with dynamic scheduling instead of static. This allows the workload to be distributed more flexibly among threads, preventing cases where some threads finish early while others remain overloaded. With schedule(dynamic, 500), work is assigned in smaller chunks, ensuring better load balancing, especially when dealing with irregular computation times across iterations.

Another significant enhancement involved optimizing memory access patterns. In the initial implementation, frequent accesses to neighboring elements in xnew caused cache contention, leading to performance degradation. By carefully restructuring the loop and ensuring better spatial locality, the updated version improves CPU cache utilization, reducing unnecessary memory operations and enhancing overall efficiency.

Task 4. (10 pts) Describe any obstacles you encountered in doing this assignment and if/how you overcame them. Summarize what you learned by doing this assignment.

One obstacle I encountered was job submission limits when trying to allocate 16 cores using interact --ntasks-per-node=16 -t 30:00. The system rejected the request due to policy constraints, so I had to adjust my resource allocation accordingly.

Another challenge was performance degradation with multiple threads. Initially, the program ran slower with more threads due to excessive synchronization from #pragma omp critical. I resolved this by using OpenMP reductions, which significantly improved efficiency.

Through this assignment, I learned how to apply data-parallel decomposition effectively and the importance of workload balancing using OpenMP’s scheduling strategies. I also gained insight into memory access optimization, which plays a crucial role in achieving better parallel speedup.