Reducing cache miss rate using thread overallocation to accelerate the MPI-OpenMP-based 2-D Hopmoc method

Frederico L. Cabral (fcabral@lncc.br)[1], Gabriel Costa (gcosta@lncc.br)[1], Carla Osthoff (osthoff@lncc.br)[1], Sanderson L. Gonzaga de Oliveira (sanderson@dcc.ufla.br)[2]

1 - National Laboratory of Scientific Computing (LNCC), 2 - Federal University of Lavras (UFLA)

Abstract

This paper applies the MPI-OpenMP-based two-dimensional Hopmoc method using the explicit work-sharing technique with a recently proposed mechanism to reduce implicit barriers in OpenMP. Specifically, this paper applies the numerical algorithm to provide approximate solutions to the advection-diffusion equation. Additionally, this article splits the mesh used by the numerical method and distributes them to over-allocated threads. The mesh partitions became so small that the approach reduced the cache miss rate. Consequently, the strategy accelerated the numerical method in multicore systems. This paper then evaluates the results of implementing the strategy under different metrics. As a result, the use of the set of techniques improved the performance of the parallel numerical method.

HOPMOC versions

```
begin
    Each MPI process computes its partition mesh
    while time < FinalTime do
        ...
        #pragma omp for
        for columns in the partition mesh do
            for all lines in the input mesh do
                MMOC or an explicit or implicit time semi-step
            end
        end
        ...
    end
end
```

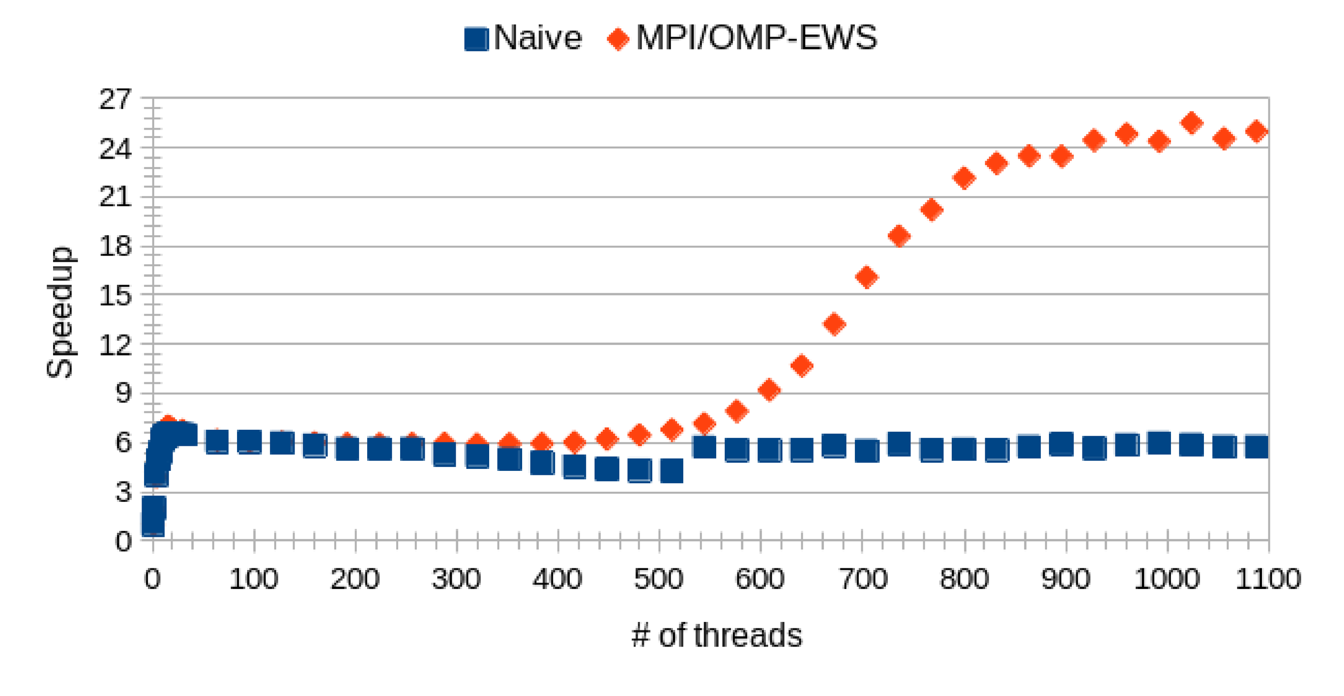
(a) Pseudocode excerpt of parallel Hopmoc method using naive MPI-OMP

```
begin
    Each MPI process computes its partition mesh
    Each OMP thread computes its local
      mesh inside the partition mesh allocated to a process
    while time < FinalTime do
        ...
        lock
        for columns in the thread local mesh do
            for all lines in the input mesh do
                MMOC or an explicit or implicit time semi-step
            end
        end
        unlock
        Each OMP thread waits for its neighbours
        ...
    end
end
```

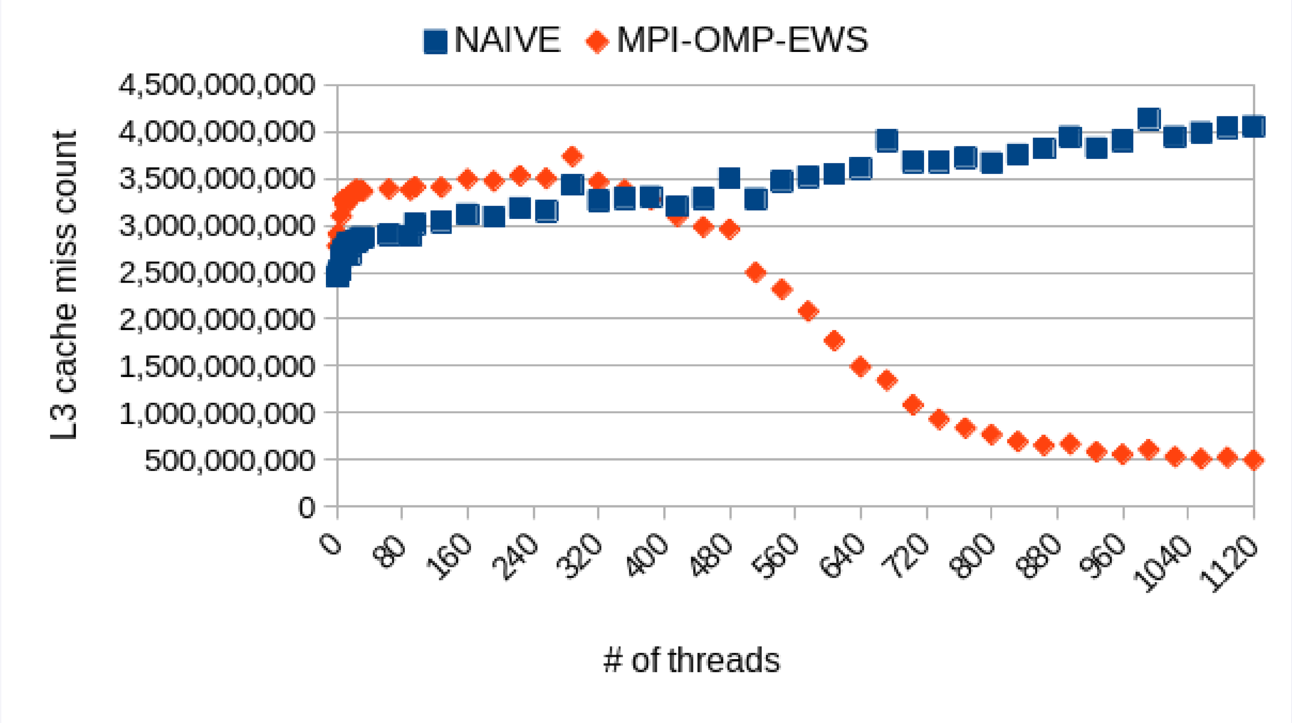
(b) Pseudocode excerpt of parallel Hopmoc method using MPI-OMP-EWS

Figure 1: Two distinct implementaions of th Hopmoc method

Experiment performed on a Intel® Xeon® processor E5-2698 machine (32 physical cores)

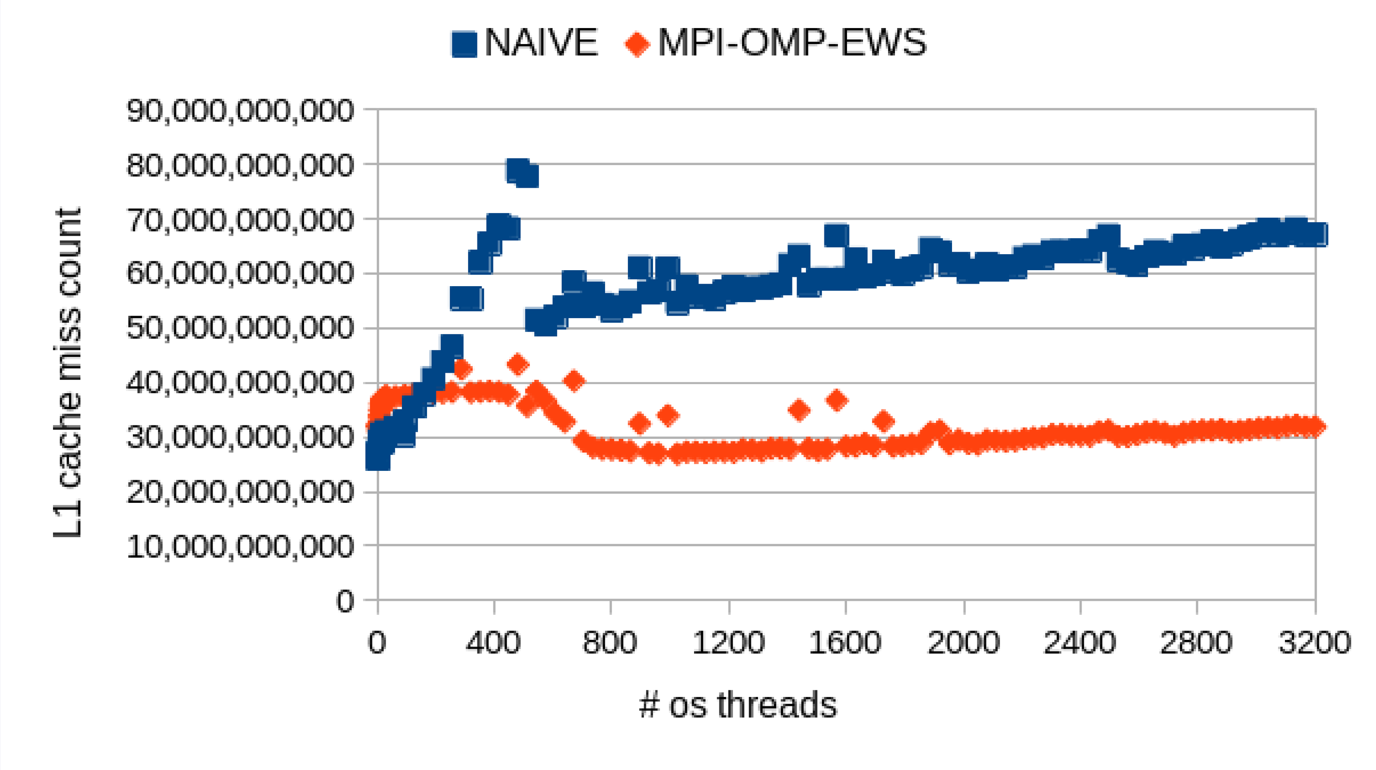


(a) Speedups from executions of naive and MPI-OMP-EWS implementations of the two-dimensional Hopmoc method varying the number of threads from 1 to 1,100 threads.

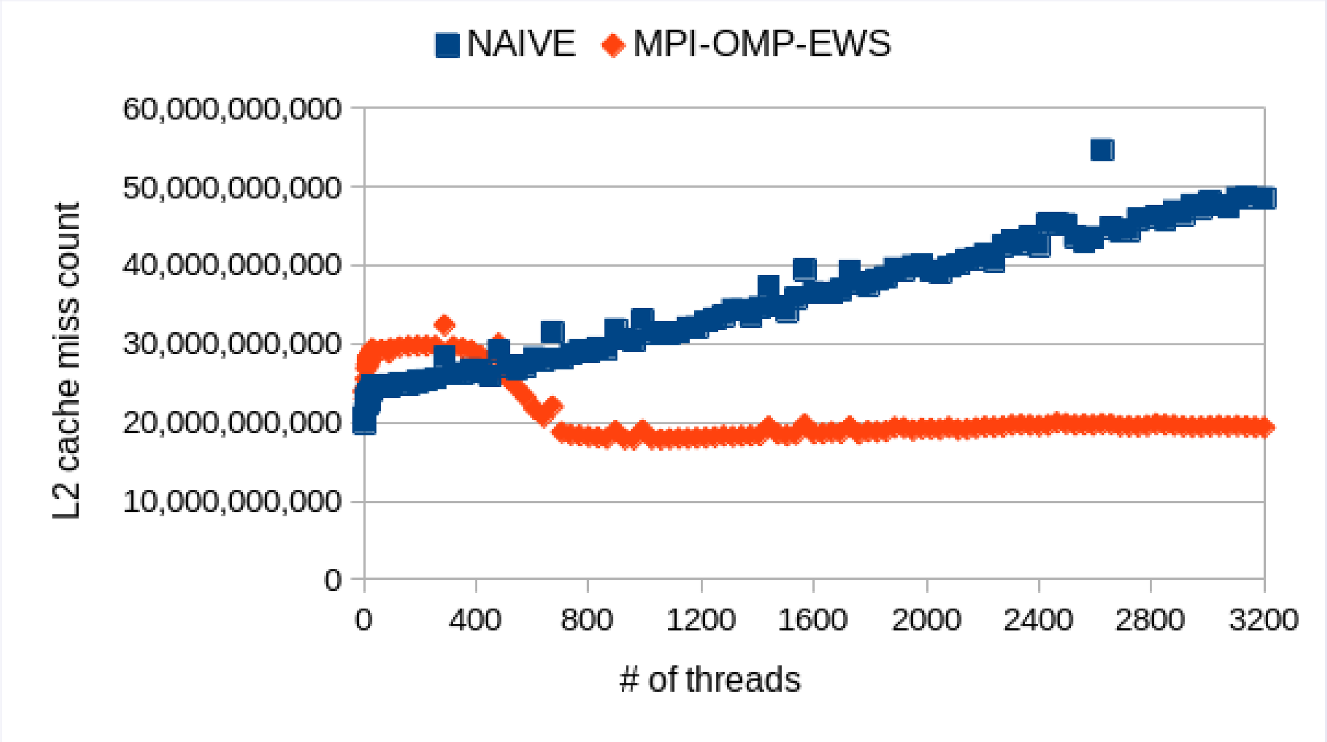


(b) LLC miss count from executions of naive and MPI-OMP-EWS implementations of the two-dimensional Hopmoc method varying the number of threads from 1 to 1,200 threads

Figure 2: Speedup and LLC miss count



(a) L1 cache miss count from executions of naive and MPI-OMP-EWS implementations of the two-dimensional Hopmoc method varying the number of threads from 1 to 3,200 threads



(b) L2 cache miss count from executions of naive and MPI-OMP-EWS implementations of the two-dimensional Hopmoc method varying the number of threads from 1 to 3,200 threads

Figure 3: L1 and L2 cache miss counts

Experiment performed on a Intel® Xeon® Gold 6230R (104 physical cores and 208 threads)

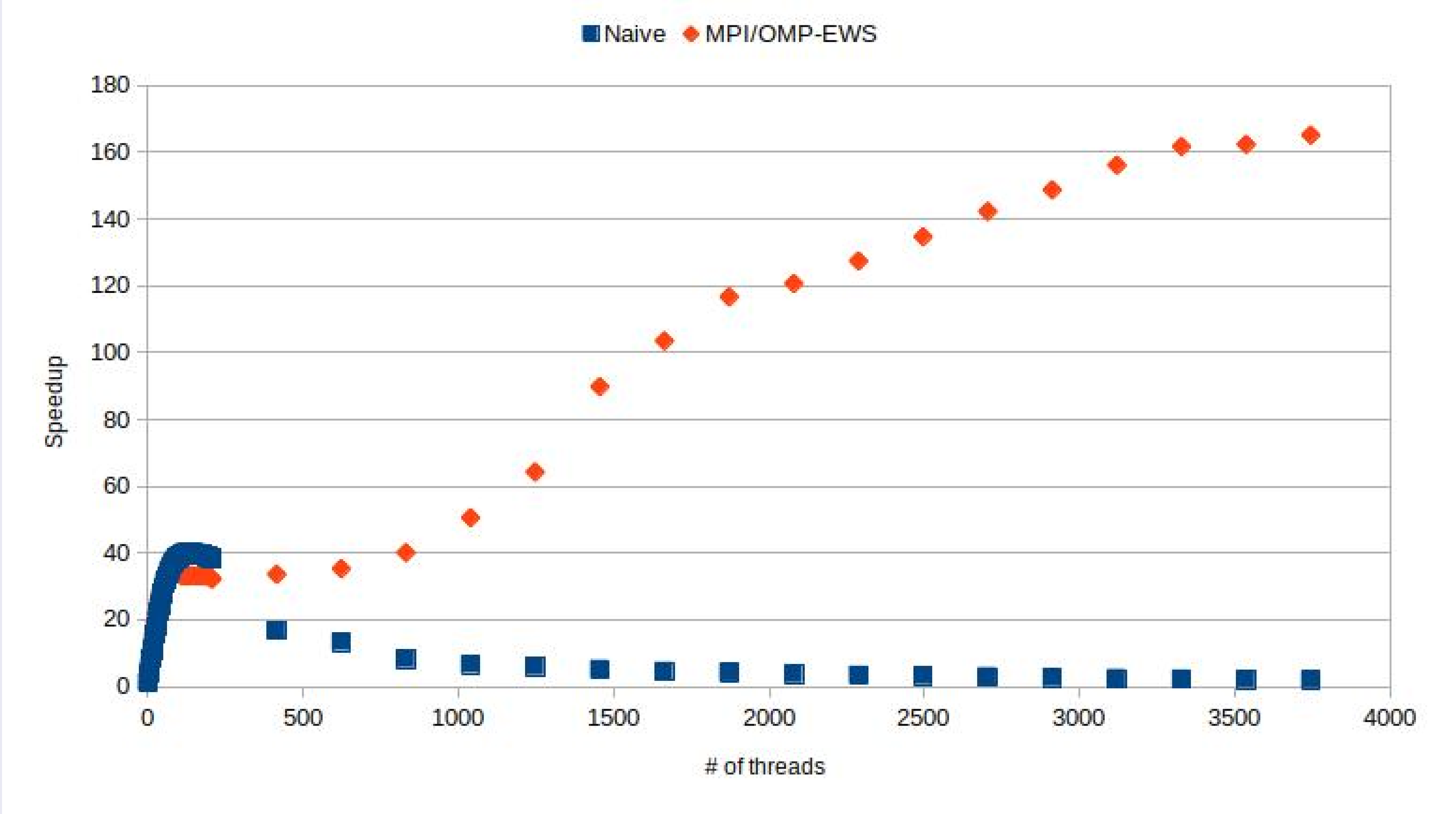


Figure 4: Speedups from executions of naive and MPI-OMP-EWS-based implementations of the two-dimensional Hopmoc method varying the number of threads from 1 to 3,744 threads.

Bibliography

1 Cabral, F.L., Gonzaga de Oliveira, S.L., Osthoff, C., Costa, G.P., Brandao,D.N., Kischinhevsky, M.; An evaluation of MPI and OpenMP paradigmsin finite-difference explicit methods for PDEs on shared-memory multi-and manycore systems. Concurrency and Computation: Practice and Experience, 32(20), e5642 (2020). https://doi.org/https://doi.org/10.1002/cpe.5642,https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5642, e5642 cpe.5642