Parallel A* Star

HPC project

Simone Bianchin
University of Trento

Jacopo Clocchiatti
University of Trento

1. Introduction

The A* algorithm stands as a fundamental tool in the domains of artificial intelligence and optimization, offering a reliable means of finding the shortest path between two points in a graph. It is a greedy algorithm, meaning that it always expands the node with the lowest estimated cost to the goal. This makes it efficient for finding the shortest path in most cases. Its ubiquity in applications like route planning and robotics is a testament to its effectiveness. However, as problems grow in complexity, so does the demand for computational efficiency.

1.1. Challenge & state of the art

There are several challenges in parallelizing the A* algorithm. One challenge is that the algorithm is inherently sequential. This is because it depends on the results of previous expansions to determine which nodes to expand next.

There are several state-of-the-art methods for parallel A*. One method is to use a divide-and-conquer approach. This involves dividing the graph into smaller subgraphs and then solving each subgraph in parallel. Another method is to use a work-stealing approach. This involves having a pool of threads that are constantly looking for work to do. When a thread finishes its current task, it steals a task from another thread.

2. Problem analysis

The A* algorithm is a graph search algorithm that finds the shortest path between a start node and a goal node. It is a greedy algorithm, meaning that it always expands the node with the lowest estimated cost to the goal. As an assumption we set that we can move only along the 4 basic directions, so we exclude diagonal movements.

The algorithm works as follows:

- Initialize a priority queue Q with the start node.
- While Q is not empty:
  - Remove the node with the lowest estimated cost from Q.
  - If the node is the goal node, then stop.
  - Otherwise, expand the node and add its neighbors to Q.

The estimated cost of a node is the sum of its actual cost from the start node and its estimated cost to the goal node. The estimated cost to the goal node is a heuristic function that estimates the distance from the node to the goal node. The A* algorithm is guaranteed to find the shortest path between the start node and the goal node, if the heuristic function is admissible, meaning that it never overestimates the distance to the goal node. The A* algorithm is a popular algorithm for finding the shortest path in a graph. It is efficient for many types of graphs, and it can be easily implemented.

Here are some additional details about the A* algorithm:

- The priority queue is used to keep track of the nodes that have not yet been expanded. The nodes are ordered in the queue according to their estimated cost to the goal node.
- The expansion of a node involves adding its neighbors to the priority queue. The neighbors are added with their estimated costs to the goal node.
- The heuristic function is used to estimate the distance from a node to the goal node. The heuristic function can be any function that estimates the distance to the goal node, but it is typically a function that is easy to compute.

2.1. Serial code

Algorithm 1 A* Algorithm

```
function ASTAR(startNode, goalNode)
    openSet ← priority queue containing startNode
    cameFrom ← empty map
    gScore[startNode] ← 0
    fScore[startNode] ← heuristic estimate of cost from startNode to goalNode
    while not openSet.isEmpty() do
        currentNode ← node in openSet with lowest fScore value
        if currentNode is the goalNode then
            return RECONSTRUCT-PATH(cameFrom, currentNode)
        end if
        openSet.remove(currentNode)
        for each neighbor neighbor of currentNode do
            tentativeGScore ← gScore[currentNode]+ actual cost from currentNode to neighbor
            if tentativeGScore < gScore[neighbor] then
                cameFrom[neighbor] ← currentNode
                gScore[neighbor] ← tentativeGScore
                fScore[neighbor] ← gScore[neighbor]+ heuristic estimate of cost from neighbor to goalNode
                if neighbor not in openSet then
                    openSet.add(neighbor)
                end if
            end if
        end for
    end while
    return Failure (no path found)
end function
```

2.2. Approaches to Parallelizing the A* Algorithm

As previously discussed in Section 1.1, the A* algorithm's sequential nature poses limits for parallelization, which make it advantageous only in specific scenarios where the standard implementation is less effective.

2.3. Parallel Solution Architecture

We explored various approaches to parallelizing the A* algorithm and finalized a single viable strategy, which was subsequently implemented. This approach involves the following steps:

1. Partitioning the maze into several segments.
2. These segments are connected through a set of *Exit Points* on each side, with N pairs of cells acting as transit gateways.
3. Assigning each maze segment to a separate processing entity (processor, core, or process).
4. Every processing entity computes paths among all exit points, including potential start and end points within its segment.
5. Aggregating these paths to formulate the overall optimal path from the start to the end point.

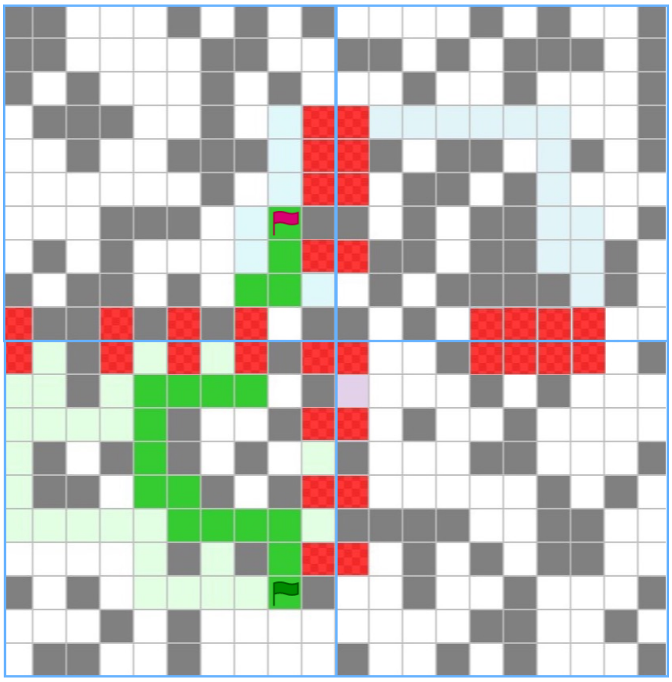


Figure 1: Illustration of Maze Resolution Using Our Parallel Implementation. Image generated with a python script given the output of a real execution as input.

Figure 1 demonstrates a solved maze to elucidate the fundamental concepts of our implementation. The four light blue squares represent the divided segments processed independently. The segments are interconnected by red exit points. The paths determined by each processing unit are shown in lighter shades, with the final optimal path marked in a more vivid green.

3. Implementation

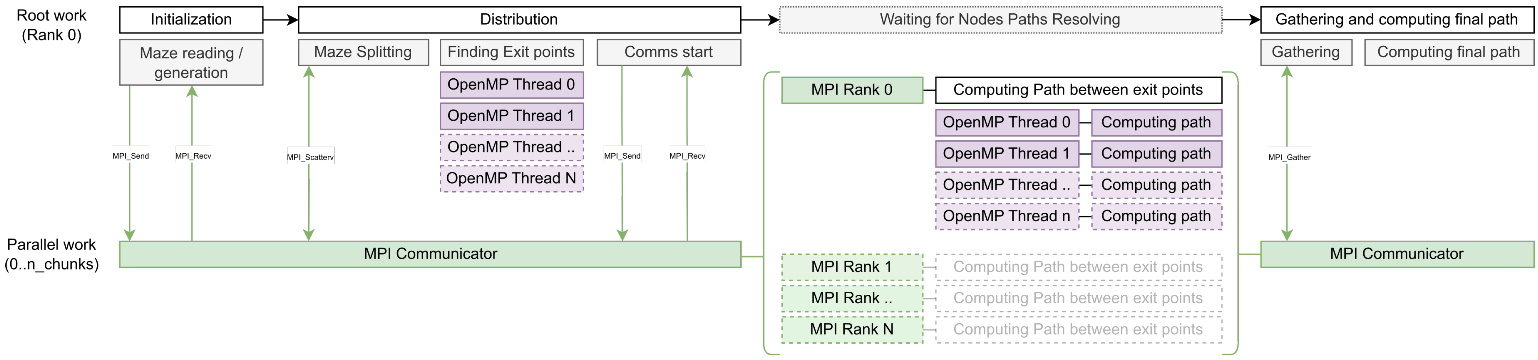
Figure 4 shows an overall view over our implementation, specified every parallel call and choice between a serial, MPI or OpenMP execution. The green color highlights the MPI ranks (Communicator) and purple the OpenMP Threads. The row on top contains the work handled by the root rank (rank 0), the bottom row instead holds all the parallel work handled by all the ranks (0..n_chunks)

3.1. Distribution

3.1.1. Maze Splitting

To maximise the code efficiency we worked to use as many native MPI calls as possible, as well

Figure 2: Overall MPI and OpenMPI implementations



as its Datatypes. Since the maze in the root rank is allocated as a 1D matrix, diving it in submazes has been very challenging. We opted to use `MPI_Type_vector`. It's very flexible and it allows to build very dynamic Datatypes from a collection of blocks of elements in an array.
Image 3 show how we define the `MPI_Datatype` thanks to `MPI_Type_vector`. `MPI_Type_vector(count, block_length, stride, node_type, &node_vector_type)`.
The square shows the matrix correctly represented, under it there is the flattened version, i.e how it's memory allocated.
This way to distribute the maze has however some limitation, we can't divided in chunks that are not squares. Also the number of chunks should be a perfect square. Anyway in a real context this is not a major problem since a non fit maze could be extended with only obstacles, at the cost of having a worst distribution since some ranks would receive full or partial black chunks.

3.1.2. Finding exit points

Exit points are identified by the root rank while parallelizing tasks using *OpenMP*. This approach is necessary because only the root is aware of what lies beyond each chunk's boundaries. Referring to picture 1, the root rank assigns each task of handling each side shared between two chunks different threads. Each thread begins by attempting to locate up to N exit points, starting from N equally spaces cells. If a cell is occupied the search moves to one extreme and then to the other. An exit point consists by two adiacent cells in the maze, both of which must be unoccupied.

Figure 3: `MPI_Type_Vector_matrix` in action visualized

3.2. A* worker

Once we have the set of exit points for each "cell," we assign each cell to a different worker (in this case, we use MPI for communication). Consequently, each worker possesses the cells within its assigned part of the maze, along with a set of exit points, including the starting and exit points if they exist. Utilizing *OpenMP*, we conduct path-finding between these sets of points, thereby identifying all possible paths (if they exist) between the exit points. Within each individual thread, we copy the cell content into a local variable and conduct the path search to avoid potential race conditions. The pre-processing phase involves creating combinations between the exit points to ensure there are no repetitions, and the local copy of the cell helps mitigate the risk of race conditions, given that only read operations are performed on shared variables. After finding a path, it is written to a shared array containing all discovered paths. This writing operation is encapsulated within a critical section of the *OpenMP* parallel operation. Subsequently, the list of found paths is passed back to the root rank, which aggregates them to form the final path.

3.3. Benchmark on the HPC@UniTrento cluster

To assess our solution, we developed a bash script to execute each test. We opted for mazes of increasing dimensions, specifically selecting dimensions of 64, 256, 1094, and 4096 units for each side of the maze. These dimensions were not arbitrary; our system necessitates that the input maze has **perfect square** dimensions and is **divisible by 4**. Hence, these dimensions were deliberately chosen.

For each maze, we tested various configurations. We conducted tests with 1 or 2 nodes, 1/2/4/8 CPUs (if testing with 2 nodes, we started with 2 CPUs), 1/4/16 processes with MPI (similarly to the choice of dimensions, they must be perfect square and divisible by 4), and 1/2/4/8 OpenMP threads. For each configuration, we ran 5 tests and averaged the results to mitigate fluctuations caused by external factors unrelated to the code. The obtained results are presented in the table in Section 5.

3.3.1. Analysis

Upon initial inspection of the overall data, inconsistencies became apparent due to anomalies resulting from the varied configurations tested, encompassing both optimal and suboptimal options. We began the analysis by searching for a "sweet spot" where our parallel solution could outperform the serial one, particularly for larger mazes.

Upon reviewing the images, it becomes evident that the speedup sweet spot is achieved with 1 CPU, 16 Processes (`comm_sz`), 8 threads, and 8 exit points. We can guess that adding more CPUs may introduce overhead, potentially increasing the total processing time. Additionally, having 8 exit points, the minimum in the possible configurations, reduces computation time but at the expense of a less optimized path.

Once identifying the sweet spot, we reanalyzed all the graphs with this specific configuration to accurately assess the performance of our parallel implementation.

Observing how the performance changes with different numbers of processes (1 serial, 4-16 parallel) as the maze size increases, the parallel implementation appears to become increasingly competitive.

Follows the exact values with the related Speedup and Efficiency, tables 1,2, and 3.

Table 1: Sum of Executions Time for Each `comm_sz` at Different Dimensions

PROCESS	64	256	1024	4096
1	0.0328	0.0499	0.3212	28.6608
4	0.0888	0.2431	0.9844	34.1893
16	0.1033	0.321	1.9241	29.3722

Table 2: Speedup for Each `comm_sz` at Different Dimensions

PROCESS	S(64)	S(256)	S(1024)	S(4096)
1.0	1.0	1.0	1.0	1.0
4	36.94	20.53	32.63	83.83
16	31.75	15.55	16.69	97.58

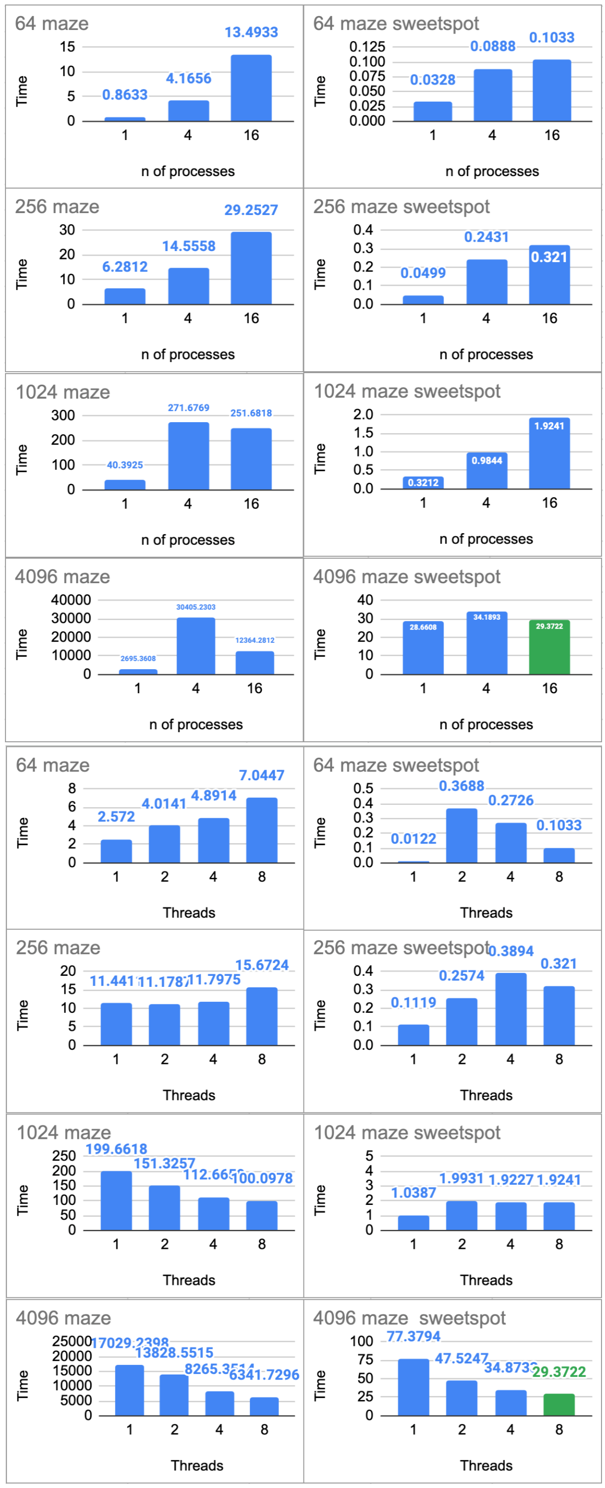
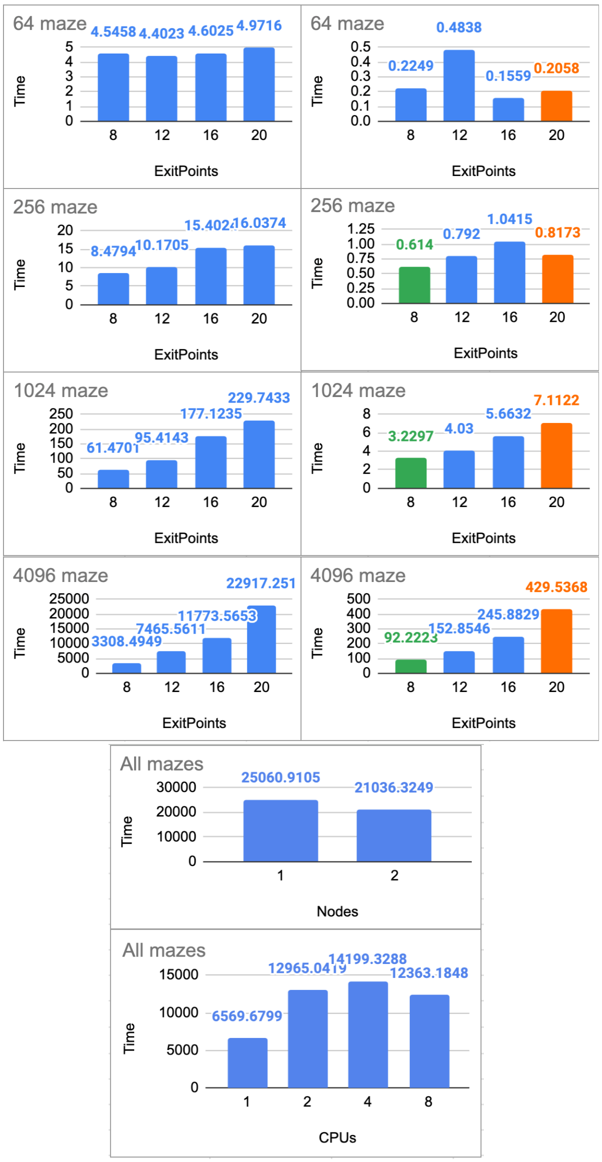


Table 3: Efficien for Each `comm_sz` at Different Dimensions

PROCESS	E(64)	E(256)	E(1024)	E(4096)
1	1.00	1.00	1.00	1.00
4	0.09	0.56	1.59	2.57
16	0.02	0.49	1.07	5.85

Considering the promising results, we conducted only a limited number of executions due to deadline constraints, using a 16K maze. What we obtained validates our success hypothesis, sur-



passing the turning point at which our implementation becomes more advantageous than the serial one.

Table 4: 16k maze performance

PROCESS	16384	S(16394)	E(16394)
1	15,717.67	1.00	1.00
4	13,672.61	1.15	0.29
16	5,549.74	2.83	0.71

4. Final discussion

4.1. Conclusion

We developed a parallel solution to the classic A* algorithm. Our approach is based on splitting the maze in different cells, find couples of border points between these cells (one point of the couple in one cell and the other point in the other cell) and find the optimal paths between these "exit-points". Lastly we use these paths we found to reconstruct the optimal path between the starting and ending points. We can assert that this parallel solution could solve a different need than the standard one. Our solution concept does not furnish the optimal path to reach the end cells, and it's implementation is constrained to a 2D context with both free cells and obstacles. By mapping numerous potential paths across different chunks throughout the map content, the performance is not exceptional with small mazes. However, this limitation is overcome by larger mazes (>16k), and furthermore, future path findings would be nearly instantaneous in an already mapped context.

4.2. Future Improvements

A primary improvement could involve a more efficient path computation within each cell, leading to an overall reduction in execution time. This can be achieved by concurrently exploring the three remaining directions, different from the one already explored, using new threads. This improvement resembles a shared queue approach, where potential points to explore are appended to a shared priority queue, prioritized based on the score of each point. Another improvement could focus on minimizing communication between processes, ideally eliminating the need to copy and pass entire cells. This optimization can be particularly beneficial when dealing with large maze dimensions. Additionally, enhancing the flexibility of maze splitting could be considered. Instead of restricting the maze to perfect squares, allowing for more flexible divisions could enable the application of our program to a broader range of mazes with an optimal distribution among the ranks.

4.3. Possible variation

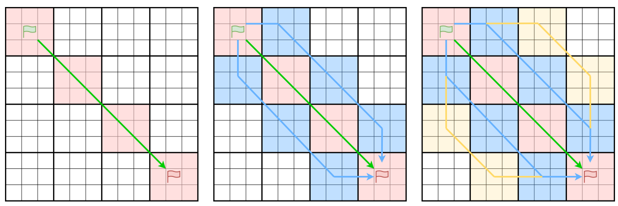


Figure 4: Concept of the variation visualized

Our implementation concept could be recursively applied to itself. To expedite the generation of a potential solution and eventually achieve a more efficient one instead of assigning, executing, and gathering all the chunks together simultaneously, prioritization could be introduced. For instance, utilizing a heuristic function could involve considering first the chunks that lie in a straight line towards the end cell. After gathering and calculating the initial, potentially sub-optimal, path, the adjacent chunks could then be taken into consideration, and this process could continue iteratively convering at the end all the maze. This concept could further highlight the strength of a parallel solution, introducing a slightly increased cost to obtain the optimal path more quickly.

Appendices

5. Results

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
8	1	1	1	1	0,001321	0,034644	0,314720	33,269840
8	1	1	1	2	0,005307	0,152577	0,680738	51,263056
8	1	1	1	4	0,016694	0,075199	0,369979	23,146298
8	1	1	1	8	0,032795	0,100540	0,321218	33,031391
8	1	1	4	1	0,002091	0,106929	2,613739	89,798034
8	1	1	4	2	0,002021	0,242363	1,177375	65,307930
8	1	1	4	4	0,222010	0,182293	1,216190	60,998748
8	1	1	4	8	0,088832	0,356661	0,984350	39,867881
8	1	1	16	1	0,012153	0,364289	1,038658	46,215234
8	1	1	16	2	0,368823	0,315222	1,993092	62,033577
8	1	1	16	4	0,272611	0,345015	1,922716	128,315835
8	1	1	16	8	0,103293	0,349340	1,924119	41,756096
8	1	2	1	1	0,001497	0,092208	0,849009	22,247095
8	1	2	1	2	0,001488	0,123782	0,392585	23,709617
8	1	2	1	4	0,003154	0,120399	0,288626	23,337916
8	1	2	1	8	0,025361	0,145415	0,319186	27,978405
8	1	2	4	1	0,058886	0,197893	1,492473	92,708286
8	1	2	4	2	0,045961	0,187036	0,677797	66,637652
8	1	2	4	4	0,051570	0,211106	0,543515	65,222702
8	1	2	4	8	0,041415	0,196184	0,621588	39,531487
8	1	2	16	1	0,130082	0,242866	1,897867	65,022749
8	1	2	16	2	0,018472	0,276391	0,527520	41,133056

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
8	1	2	16	4	0,153248	0,174425	0,755501	45,016829
8	1	2	16	8	0,256027	0,178165	0,871968	24,880703
8	1	4	1	1	0,002409	0,064889	0,358430	33,471247
8	1	4	1	2	0,003554	0,085064	0,377970	21,919416
8	1	4	1	4	0,002138	0,084307	0,316783	22,000789
8	1	4	1	8	0,017560	0,101621	0,310635	21,545824
8	1	4	4	1	0,002784	0,117017	1,127365	67,334158
8	1	4	4	2	0,009516	0,108294	0,673968	68,710893
8	1	4	4	4	0,004092	0,132113	0,523640	52,532456
8	1	4	4	8	0,088897	0,137379	0,775058	43,259162
8	1	4	16	1	0,053481	0,214731	0,745875	70,533811
8	1	4	16	2	0,030437	0,244313	0,701840	38,072456
8	1	4	16	4	0,066929	0,214751	0,557562	38,135182
8	1	4	16	8	0,204521	0,334677	0,381958	30,797092
8	1	8	1	1	0,002225	0,070149	0,368758	24,257200
8	1	8	1	2	0,001542	0,065997	0,322171	24,082742
8	1	8	1	4	0,001581	0,057390	0,286592	25,791116
8	1	8	1	8	0,005651	0,072388	0,400259	23,649748
8	1	8	4	1	0,024290	0,063901	1,134375	68,241332
8	1	8	4	2	0,017260	0,082124	0,712805	63,102455
8	1	8	4	4	0,027234	0,067153	0,609935	45,238107
8	1	8	4	8	0,043233	0,113881	0,702081	32,381354
8	1	8	16	1	0,102580	0,188987	0,592524	39,331227
8	1	8	16	2	0,066401	0,176610	0,615343	39,659085
8	1	8	16	4	0,092966	0,143302	0,564568	28,167345
8	1	8	16	8	0,156495	0,203291	0,382937	27,970197
8	2	2	1	1	0,001510	0,046492	0,383175	23,255257
8	2	2	1	2	0,001464	0,088777	0,375153	23,889465
8	2	2	1	4	0,008349	0,090620	0,292058	22,756483
8	2	2	1	8	0,051015	0,085998	0,448606	25,331121
8	2	2	4	1	0,004320	0,117646	1,314074	75,372662
8	2	2	4	2	0,064257	0,120422	0,826255	53,638256
8	2	2	4	4	0,033759	0,155327	0,633854	52,841720
8	2	2	4	8	0,004378	0,255529	1,179430	37,777477
8	2	2	16	1	0,093616	0,126536	1,241274	53,057191

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
8	2	2	16	2	0,109503	0,192646	0,969110	37,988494
8	2	2	16	4	0,113133	0,221281	0,933132	50,629275
8	2	2	16	8	0,059389	0,266968	0,401685	38,536289
8	2	4	1	1	0,003833	0,056323	0,338271	28,426995
8	2	4	1	2	0,011681	0,062197	0,314230	27,664757
8	2	4	1	4	0,008545	0,095556	0,436968	25,304543
8	2	4	1	8	0,005434	0,109579	0,343106	24,359315
8	2	4	4	1	0,002782	0,089258	1,130092	84,677683
8	2	4	4	2	0,054890	0,085722	0,752304	44,672264
8	2	4	4	4	0,017646	0,171761	0,617136	42,823234
8	2	4	4	8	0,050121	0,267453	1,644613	31,262579
8	2	4	16	1	0,002856	0,179042	0,835467	58,347382
8	2	4	16	2	0,129338	0,250298	0,613239	47,337461
8	2	4	16	4	0,215379	0,320527	0,541738	38,934125
8	2	4	16	8	0,057875	0,311617	0,843660	28,065138
8	2	8	1	1	0,001824	0,055250	0,311973	24,263268
8	2	8	1	2	0,004813	0,079800	0,306348	22,039186
8	2	8	1	4	0,001800	0,077092	0,313948	25,444254
8	2	8	1	8	0,003142	0,112937	0,349497	21,655390
8	2	8	4	1	0,018704	0,090638	1,211377	75,291420
8	2	8	4	2	0,008763	0,094837	0,722475	118,233325
8	2	8	4	4	0,021331	0,199640	0,535464	52,320846
8	2	8	4	8	0,064581	0,178735	0,564985	46,175060
8	2	8	16	1	0,064103	0,162038	0,588919	37,846765
8	2	8	16	2	0,099209	0,186904	0,510338	27,990636
8	2	8	16	4	0,071338	0,135399	0,620041	27,248134
8	2	8	16	8	0,138320	0,174534	0,687780	34,564556
12	1	1	1	1	0,001400	0,046863	0,326097	27,393640
12	1	1	1	2	0,006600	0,115765	0,544782	28,161538
12	1	1	1	4	0,020605	0,085517	0,432912	33,561844
12	1	1	1	8	0,001508	0,118750	0,360335	27,097983
12	1	1	4	1	0,003968	0,318162	2,667641	339,922688
12	1	1	4	2	0,031289	0,243790	2,937227	130,799683
12	1	1	4	4	0,034366	0,370860	1,493747	91,518756
12	1	1	4	8	0,063153	0,394395	2,256971	83,538388

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
12	1	1	16	1	0,084815	0,529250	1,866624	161,539134
12	1	1	16	2	0,025397	0,496777	2,860762	114,664141
12	1	1	16	4	0,063192	0,568232	1,976901	124,086765
12	1	1	16	8	0,419141	0,550101	1,412702	76,386857
12	1	2	1	1	0,002513	0,050410	0,417155	28,508722
12	1	2	1	2	0,007273	0,055869	0,464829	26,951199
12	1	2	1	4	0,019936	0,059191	0,337561	26,626900
12	1	2	1	8	0,001539	0,065201	0,302522	24,870257
12	1	2	4	1	0,006444	0,127776	2,703268	294,815037
12	1	2	4	2	0,027060	0,099585	1,727435	173,071231
12	1	2	4	4	0,072963	0,091627	1,024549	126,918419
12	1	2	4	8	0,048700	0,091689	0,856076	106,854656
12	1	2	16	1	0,068597	0,148436	2,044760	192,060411
12	1	2	16	2	0,210623	0,168919	1,682891	110,123260
12	1	2	16	4	0,024736	0,311089	0,829974	92,766827
12	1	2	16	8	0,233572	0,451466	0,894555	63,657202
12	1	4	1	1	0,001498	0,080223	0,274841	21,559081
12	1	4	1	2	0,002450	0,063141	0,301072	24,068424
12	1	4	1	4	0,001862	0,070634	0,417500	22,555913
12	1	4	1	8	0,007120	0,079559	0,312178	21,720178
12	1	4	4	1	0,003891	0,162052	2,342292	302,818018
12	1	4	4	2	0,019336	0,204392	1,438193	161,729769
12	1	4	4	4	0,050412	0,121192	1,037039	119,986345
12	1	4	4	8	0,058469	0,271110	0,846747	108,168061
12	1	4	16	1	0,099040	0,363374	2,070545	156,702735
12	1	4	16	2	0,101900	0,209048	1,395142	112,160497
12	1	4	16	4	0,045685	0,342544	1,395233	75,553204
12	1	4	16	8	0,160573	0,226777	1,009758	53,673585
12	1	8	1	1	0,001880	0,062251	0,314738	21,611957
12	1	8	1	2	0,005082	0,082288	0,313885	23,053111
12	1	8	1	4	0,002383	0,079938	0,346053	25,834794
12	1	8	1	8	0,001936	0,087987	0,298182	28,099554
12	1	8	4	1	0,007410	0,160008	2,257470	337,211178
12	1	8	4	2	0,013860	0,158058	1,679067	158,558566
12	1	8	4	4	0,026735	0,142188	1,111567	102,983923

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
12	1	8	4	8	0,050041	0,138976	0,798086	92,238953
12	1	8	16	1	0,073706	0,230625	1,752444	109,855017
12	1	8	16	2	0,075773	0,257971	0,938476	88,530616
12	1	8	16	4	0,077892	0,140641	0,636316	60,590328
12	1	8	16	8	0,197501	0,222167	0,769106	46,659898
12	2	2	1	1	0,001388	0,072946	0,492373	22,647416
12	2	2	1	2	0,001672	0,096799	0,628870	25,619438
12	2	2	1	4	0,014882	0,090089	0,316956	22,377418
12	2	2	1	8	0,015515	0,110670	0,303662	20,261361
12	2	2	4	1	0,007181	0,155282	2,227121	248,648168
12	2	2	4	2	0,052702	0,244579	1,875983	154,354344
12	2	2	4	4	0,040124	0,286280	1,833236	99,814280
12	2	2	4	8	0,058362	0,335195	0,889969	88,497235
12	2	2	16	1	0,020822	0,316801	1,949387	135,955875
12	2	2	16	2	0,162885	0,247966	2,070040	111,742861
12	2	2	16	4	0,124890	0,284797	1,107744	143,797882
12	2	2	16	8	0,199030	0,288377	1,591264	70,329628
12	2	4	1	1	0,001285	0,072813	0,286941	28,527450
12	2	4	1	2	0,001353	0,085035	0,290836	25,332136
12	2	4	1	4	0,009824	0,100548	0,334455	23,339432
12	2	4	1	8	0,012203	0,128032	0,393544	19,703402
12	2	4	4	1	0,018603	0,129451	2,350915	214,336740
12	2	4	4	2	0,004048	0,237757	1,689069	124,636677
12	2	4	4	4	0,099836	0,273485	1,088310	111,667428
12	2	4	4	8	0,052473	0,334034	0,869112	108,252353
12	2	4	16	1	0,039741	0,358833	1,731790	242,139566
12	2	4	16	2	0,145057	0,316765	1,629814	116,821855
12	2	4	16	4	0,056030	0,272915	1,164241	68,369648
12	2	4	16	8	0,255329	0,287718	0,869045	79,557819
12	2	8	1	1	0,001347	0,076418	0,289159	26,295083
12	2	8	1	2	0,007196	0,086590	0,347932	20,284579
12	2	8	1	4	0,002102	0,078070	0,356454	19,889997
12	2	8	1	8	0,002497	0,099085	0,293323	24,567020
12	2	8	4	1	0,023161	0,163396	2,064820	246,330478
12	2	8	4	2	0,031873	0,130265	1,580279	136,157547

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
12	2	8	4	4	0,025250	0,143780	1,099916	91,845104
12	2	8	4	8	0,025954	0,284922	1,141508	71,331598
12	2	8	16	1	0,055641	0,235175	1,547282	116,535562
12	2	8	16	2	0,021449	0,174242	0,931693	134,423722
12	2	8	16	4	0,101816	0,229166	0,670208	143,629838
12	2	8	16	8	0,142901	0,228367	0,628922	49,673684
16	1	1	1	1	0,001453	0,054770	0,369756	35,830626
16	1	1	1	2	0,002855	0,086698	0,328462	34,931165
16	1	1	1	4	0,014083	0,094126	0,554616	36,765834
16	1	1	1	8	0,005126	0,083802	0,373291	34,898659
16	1	1	4	1	0,006932	0,359944	4,265681	554,859801
16	1	1	4	2	0,016539	0,274307	5,791816	251,008551
16	1	1	4	4	0,003817	0,273999	2,276964	178,091800
16	1	1	4	8	0,013260	0,403899	3,036501	170,818636
16	1	1	16	1	0,021357	0,606945	8,585380	214,413390
16	1	1	16	2	0,146513	0,448879	2,870637	204,044076
16	1	1	16	4	0,134143	0,594741	2,676471	159,008694
16	1	1	16	8	0,137490	0,584669	2,253379	121,490361
16	1	2	1	1	0,001519	0,103337	0,312964	26,856722
16	1	2	1	2	0,015084	0,122666	0,319236	32,477034
16	1	2	1	4	0,013866	0,114651	0,465218	29,202600
16	1	2	1	8	0,009140	0,158262	0,305734	23,367655
16	1	2	4	1	0,015786	0,367133	4,117679	490,608807
16	1	2	4	2	0,004493	0,297704	5,691107	329,189996
16	1	2	4	4	0,052477	0,290800	2,786779	181,085785
16	1	2	4	8	0,109171	0,260232	1,334246	164,518233
16	1	2	16	1	0,061104	0,406920	4,473131	313,565324
16	1	2	16	2	0,097528	0,286290	2,519765	171,085699
16	1	2	16	4	0,151937	0,201727	2,525358	110,371913
16	1	2	16	8	0,046374	0,220501	1,722130	84,631842
16	1	4	1	1	0,001471	0,075439	0,298612	22,959574
16	1	4	1	2	0,004857	0,072622	0,326114	22,358017
16	1	4	1	4	0,019109	0,088890	0,293370	22,045875
16	1	4	1	8	0,009082	0,106173	0,447816	22,873269
16	1	4	4	1	0,028536	0,172715	4,675136	410,901616

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s) DIMENSION 64	256	1024	4096
16	1	4	4	2	0,008810	0,163534	3,694699	325,030583
16	1	4	4	4	0,083679	0,206686	2,513286	181,566346
16	1	4	4	8	0,011732	0,156521	1,387023	179,999481
16	1	4	16	1	0,164332	0,359492	3,054563	195,394720
16	1	4	16	2	0,022225	0,430862	2,593425	135,642691
16	1	4	16	4	0,100375	0,332319	2,282195	92,205023
16	1	4	16	8	0,049566	0,408369	1,963033	75,225761
16	1	8	1	1	0,001648	0,068291	0,337469	23,601609
16	1	8	1	2	0,001621	0,089382	0,294162	24,428319
16	1	8	1	4	0,005039	0,089080	0,327291	26,199890
16	1	8	1	8	0,003208	0,089522	0,353218	25,284480
16	1	8	4	1	0,006905	0,166607	4,213941	402,287496
16	1	8	4	2	0,021529	0,159352	3,124253	346,316042
16	1	8	4	4	0,043907	0,122871	2,553567	170,348337
16	1	8	4	8	0,060734	0,136167	1,447803	140,091647
16	1	8	16	1	0,039191	0,286663	3,068415	187,692629
16	1	8	16	2	0,103497	0,246986	2,029595	123,516272
16	1	8	16	4	0,108793	0,209136	1,244744	69,078809
16	1	8	16	8	0,129681	0,276730	1,022324	66,940825
16	2	2	1	1	0,001516	0,065692	0,320844	24,138414
16	2	2	1	2	0,010345	0,067573	0,332193	26,263888
16	2	2	1	4	0,014029	0,114417	0,358700	22,714608
16	2	2	1	8	0,011108	0,136471	0,307108	24,898287
16	2	2	4	1	0,009875	0,146169	4,701426	399,425259
16	2	2	4	2	0,008632	0,214271	4,014176	315,199357
16	2	2	4	4	0,121455	0,281865	3,923870	194,811430
16	2	2	4	8	0,064136	0,265803	1,885676	169,660464
16	2	2	16	1	0,175838	0,229123	4,122385	212,433414
16	2	2	16	2	0,127324	0,341662	4,236810	193,628508
16	2	2	16	4	0,191589	0,331987	2,857199	147,282235
16	2	2	16	8	0,342206	0,320532	2,588201	146,968480
16	2	4	1	1	0,001406	0,090089	0,321334	21,856242
16	2	4	1	2	0,012251	0,109046	0,316741	22,115861
16	2	4	1	4	0,002983	0,104907	0,300101	24,881529
16	2	4	1	8	0,023787	0,066857	0,463893	23,576052

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
16	2	4	4	1	0,021327	0,186414	4,728732	431,873688
16	2	4	4	2	0,065813	0,300005	3,637032	357,663535
16	2	4	4	4	0,048861	0,331186	2,231590	280,033240
16	2	4	4	8	0,053306	0,219370	1,365870	189,617062
16	2	4	16	1	0,131029	0,370866	3,801168	313,947989
16	2	4	16	2	0,014806	0,347831	1,937968	166,326110
16	2	4	16	4	0,203741	0,479397	1,844597	139,449990
16	2	4	16	8	0,147003	0,547351	2,184126	85,514991
16	2	8	1	1	0,001694	0,078129	0,367870	25,014368
16	2	8	1	2	0,007055	0,074693	0,318290	27,745103
16	2	8	1	4	0,012180	0,092014	0,292566	25,839462
16	2	8	1	8	0,022658	0,108332	0,341451	33,373877
16	2	8	4	1	0,010460	0,147533	4,475645	315,684140
16	2	8	4	2	0,028369	0,219818	3,621654	319,887090
16	2	8	4	4	0,010767	0,323593	2,355918	167,674872
16	2	8	4	8	0,060538	0,219944	1,332607	221,960272
16	2	8	16	1	0,125606	0,294042	2,971057	186,278091
16	2	8	16	2	0,132161	0,263541	1,907184	183,197032
16	2	8	16	4	0,143891	0,274278	1,347910	122,744075
16	2	8	16	8	0,131177	0,328656	1,507093	68,634118
20	1	1	1	1	0,001470	0,107758	0,370007	35,615244
20	1	1	1	2	0,007562	0,106528	0,348753	30,277604
20	1	1	1	4	0,005222	0,084444	0,486918	32,309115
20	1	1	1	8	0,029992	0,125189	0,534718	35,569349
20	1	1	4	1	0,008115	0,369344	5,922776	1094,610759
20	1	1	4	2	0,005598	0,459766	5,096209	852,511247
20	1	1	4	4	0,004996	0,438688	2,710017	424,011829
20	1	1	4	8	0,021747	0,490051	3,518213	266,256102
20	1	1	16	1	0,007189	1,044106	11,552586	452,425197
20	1	1	16	2	0,010338	0,971468	4,931056	260,385172
20	1	1	16	4	0,036239	0,926708	3,925745	238,376273
20	1	1	16	8	0,154110	0,827452	3,059257	127,121962
20	1	2	1	1	0,001525	0,055800	0,331805	27,330078
20	1	2	1	2	0,010567	0,062368	0,307175	23,665012
20	1	2	1	4	0,043363	0,063178	0,494340	22,045815

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s) DIMENSION 64	256	1024	4096
20	1	2	1	8	0,022596	0,063681	0,353801	27,303250
20	1	2	4	1	0,055922	0,196949	4,861760	1125,063425
20	1	2	4	2	0,029305	0,151606	4,229963	728,242880
20	1	2	4	4	0,010167	0,126006	2,909717	458,857034
20	1	2	4	8	0,052026	0,123170	2,878821	321,875324
20	1	2	16	1	0,134240	0,311890	6,140323	627,133479
20	1	2	16	2	0,173811	0,371517	6,617688	309,919999
20	1	2	16	4	0,114952	0,655420	3,414254	151,303364
20	1	2	16	8	0,374574	0,751736	3,141212	164,838284
20	1	4	1	1	0,001652	0,054023	0,319982	21,809645
20	1	4	1	2	0,008984	0,071367	0,329186	37,145651
20	1	4	1	4	0,001416	0,067948	0,321082	22,512517
20	1	4	1	8	0,012076	0,078509	0,302335	22,048042
20	1	4	4	1	0,008782	0,379770	6,480455	1052,873627
20	1	4	4	2	0,049744	0,330430	3,912650	944,896604
20	1	4	4	4	0,041555	0,381001	3,074542	473,616583
20	1	4	4	8	0,062450	0,316183	2,025494	324,822844
20	1	4	16	1	0,015728	0,417058	5,081837	302,190256
20	1	4	16	2	0,217387	0,566211	2,903072	305,424468
20	1	4	16	4	0,134400	0,500944	2,942666	167,598177
20	1	4	16	8	0,087080	0,415466	3,020161	114,916480
20	1	8	1	1	0,001499	0,061504	0,296928	22,957633
20	1	8	1	2	0,002131	0,085533	0,377828	25,101216
20	1	8	1	4	0,001911	0,086481	0,325953	27,518885
20	1	8	1	8	0,005660	0,103340	0,293500	22,765939
20	1	8	4	1	0,009029	0,264172	5,630594	827,657428
20	1	8	4	2	0,022782	0,244061	3,837211	787,379236
20	1	8	4	4	0,012280	0,193987	2,592198	400,910528
20	1	8	4	8	0,028870	0,189655	1,941289	266,550415
20	1	8	16	1	0,018264	0,460560	5,195378	281,080947
20	1	8	16	2	0,141465	0,350020	2,841786	238,290685
20	1	8	16	4	0,131024	0,316928	1,961175	140,628363
20	1	8	16	8	0,202606	0,293579	1,906129	99,762492
20	2	2	1	1	0,001550	0,078395	0,373247	23,732444
20	2	2	1	2	0,008647	0,119548	0,309613	22,952913

Table 5 continued from previous page

EXIT POINTS	NODE	CPU	PROCESS	THREAD	TIME (s)			
					DIMENSION			
					64	256	1024	4096
20	2	2	1	4	0,001805	0,093972	0,642245	24,514100
20	2	2	1	8	0,036541	0,084711	0,326021	20,623414
20	2	2	4	1	0,017783	0,266971	5,364088	986,183323
20	2	2	4	2	0,059064	0,244491	4,077059	688,701516
20	2	2	4	4	0,018141	0,225805	3,201043	390,181851
20	2	2	4	8	0,192192	0,779416	4,655615	275,655122
20	2	2	16	1	0,206019	0,330003	4,866629	496,070412
20	2	2	16	2	0,009934	0,404976	5,131302	339,184215
20	2	2	16	4	0,240102	0,503037	4,287526	194,012740
20	2	2	16	8	0,200917	0,593405	3,002821	123,446165
20	2	4	1	1	0,001507	0,096383	0,413823	24,709082
20	2	4	1	2	0,007047	0,060662	0,340912	20,155969
20	2	4	1	4	0,007893	0,072667	0,307191	19,539613
20	2	4	1	8	0,009869	0,109940	0,303062	19,337474
20	2	4	4	1	0,031707	0,308483	5,886841	1184,323721
20	2	4	4	2	0,064706	0,237091	4,079526	837,691104
20	2	4	4	4	0,018300	0,332507	3,006957	642,949228
20	2	4	4	8	0,115770	0,645016	2,284485	442,069694
20	2	4	16	1	0,046983	0,408327	4,764206	334,312953
20	2	4	16	2	0,162178	0,468340	3,188366	347,220887
20	2	4	16	4	0,095926	0,667400	3,265289	114,972133
20	2	4	16	8	0,146894	0,643768	3,384862	84,712396
20	2	8	1	1	0,002189	0,054050	0,288453	23,104279
20	2	8	1	2	0,002003	0,093958	0,338341	28,287311
20	2	8	1	4	0,008314	0,075459	0,321146	20,476734
20	2	8	1	8	0,003489	0,113031	0,295362	21,612974
20	2	8	4	1	0,012024	0,238989	5,105469	743,132928
20	2	8	4	2	0,048084	0,348570	3,788087	882,945446
20	2	8	4	4	0,070213	0,494670	2,684915	719,734921
20	2	8	4	8	0,068840	0,417249	2,130138	222,398906
20	2	8	16	1	0,048422	0,351427	4,965053	323,278793
20	2	8	16	2	0,110571	0,267668	2,863526	157,060705
20	2	8	16	4	0,091627	0,440788	1,887898	124,469416
20	2	8	16	8	0,269953	0,393511	1,561637	73,879697