

Operating Systems EE431L

Complex Engineering Problem - Report



Parallelized implementation of Gauss-Seidel method using OpenMP and POSIX Threads

Submitted by:

2016-EE-189 Muhammad Kamil

Submitted to:

Mr. M. Usama Zubair

CEP Report

Parallelized implementation of Gauss-Seidel
method using OpenMP and POSIX Threads

Muhammad Kamil 2016-EE-189

kamiljaved98@gmail.com

December 2020

*Department of Electrical Engineering
University of Engineering & Technology, Lahore*

Table of Contents

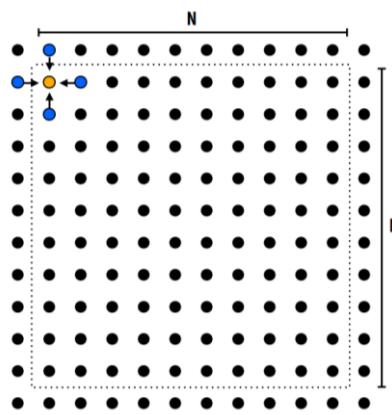
<i>Section</i>	<i>Part</i>	<i>Title</i>	<i>Page No.</i>
I		PROBLEM STATEMENT	13
II		RED-BLACK CELLS APPROACH	
	1	Sequential Implementation	13
	2	Parallel Implementation with OpenMP	16
		(a) using pragma omp critical	
		(b) using per-thread diff (no padding)	
		(c) using per-thread diff (with padding)	
	3	Parallel Implementation with Pthreads	16
		(a) using mutex lock	
		(b) using per-thread diff (no padding)	
		(c) using per-thread diff (with padding)	
III		ANTI-DIAGONALS APPROACH	
	1	Sequential Implementation	13
	2	Parallel Implementation with OpenMP	16
		(a) using pragma omp critical	
		(b) using per-thread diff (no padding)	
		(c) using per-thread diff (with padding)	
	3	Parallel Implementation with Pthreads	16
		(a) using mutex lock	
		(b) using per-thread diff (no padding)	
		(c) using per-thread diff (with padding)	
IV		SUMMARY OF RESULTS	13

Section I

Problem Statement

The task is to solve a partial differential equation on $(N+2) \times (N+2)$ grid (2D), in a parallel fashion, that is, to perform Gauss-Seidel sweeps over the grid until convergence. For a cell at point $[i, j]$, the new value is calculated using the formula:

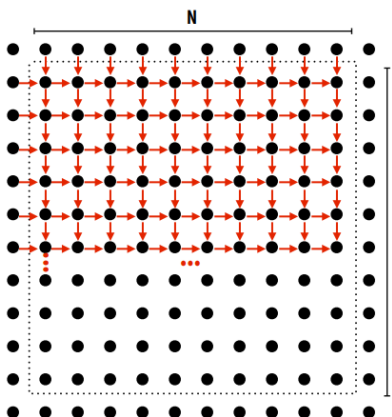
$$A[i, j] = 0.2 \times (A[i, j] + A[i, j - 1] + A[i - 1, j] + A[i, j + 1] + A[i + 1, j])$$



The simplest method to do a sweep would be to start at the first cell (indicated in yellow above), and move through the row, and then onto the next row. Repeat the process until convergence occurs.

To implement this in a parallelized method, some dependencies (per iteration over entire grid) must be taken care of, enumerated below.

1. Each row element depends on element to left.
2. Each row depends on previous row.



In this report, two approaches, namely (1) Red-Black Cells Approach, and (2) Anti-Diagonals Approach, are discussed, and the results of their implementation have been shared.

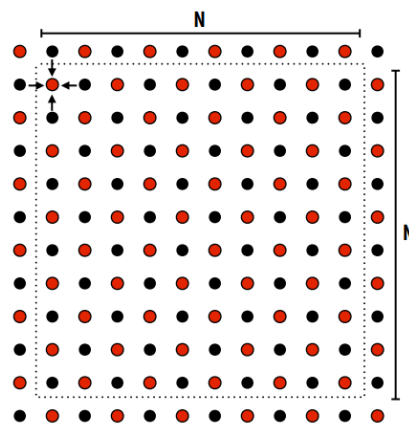
Section II.1

RED-BLACK CELLS APPROACH

Sequential Implementation

Methodology

In this approach, the alternate grid cells are bunched into red and black groups, as follows:



First, update all red cells. When done updating red cells, update all black cells (respect dependency on red cells). Repeat until convergence.

The following pseudocode sums up the process (for sequential approach):

```
convert image to grayscale

begin loop1

  diff = 0

  for each red cell 'redc'
    tmp = redc
    redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
    diff = diff + absolute(redc - tmp)
  endfor

  for each black cell 'blackc'
    tmp = blackc
    blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
    diff = diff + absolute(blackc - tmp)
  endfor

  if (diff/num_cells < threshold): end loop1
```

Results

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help

[kamiljaved@kj98-manjaro bin]$ ./main image01.jpg blurS.bmp
Execution time of gridSolver: 5.602798 seconds.
```

Execution Time = 5.6 seconds



Original Image



Resultant Image

Section II.1.a RED-BLACK CELLS APPROACH

Parallel Implementation with OpenMP using “pragma omp critical”

Parallelization of Red-Black Cells Approach

This approach can be parallelized by updating the red cells in parallel and the black cells in parallel. First, update all red cells in parallel. When done updating red cells, update all black cells in parallel (respect dependency on red cells). Repeat until convergence.

Methodology

The OpenMP library is used to update cells of one color in parallel in threads, using interleaved assignment (starting at thread_num for a thread, cells after step of thread_count assigned to that thread). Thread count was set to 4.

To avoid race condition on updating of the variable ‘diff’, the line(s) updating the ‘diff’ variable were wrapped inside a critical block.

The following pseudocode sums up the process:

```
convert image to grayscale

begin loop1

    diff = 0

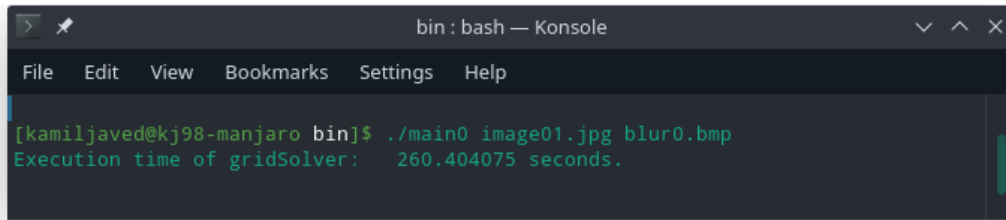
    #pragma omp parallel for
    for each red cell 'redc'
        tmp = redc
        redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
        #pragma omp critical { diff = diff + absolute(redc - tmp) }
    endfor

    #pragma omp barrier

    #pragma omp parallel for
    for each black cell 'blackc'
        tmp = blackc
        blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
        #pragma omp critical { diff = diff + absolute(blackc - tmp) }
    endfor

    if (diff/num_cells < threshold): end loop1
```

Results

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]'. The command './main0 image01.jpg blur0.bmp' has been executed, and the output is 'Execution time of gridSolver: 260.404075 seconds.'

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help

[kamiljaved@kj98-manjaro bin]$ ./main0 image01.jpg blur0.bmp
Execution time of gridSolver: 260.404075 seconds.
```

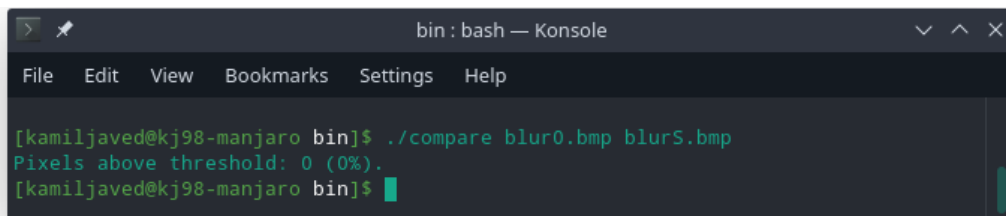
Execution Time = 260.4 seconds

Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{260.4} = 0.02$$

There is no speedup, in fact the method took longer time than sequential approach. This is because each thread must wait in a queue (in each iteration) to be able to update the 'diff' variable (as that is a critical section), i.e. the behavior effectively becomes sequential. And hence, the synchronization cost outweighs any benefit achieved by parallelizing here (using "pragma omp critical").

Correctness

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]'. The command './compare blur0.bmp blurS.bmp' has been executed, and the output is 'Pixels above threshold: 0 (0%).' followed by a new prompt.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help

[kamiljaved@kj98-manjaro bin]$ ./compare blur0.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```


Section II.1.b RED-BLACK CELLS APPROACH

Parallel Implementation with OpenMP using per-thread 'diff' (no padding)

Methodology

To avoid the blockage occurring at identified critical section i.e. updating 'diff', each thread is provided with a variable that it can use to store its personal diff-sum. Hence, an array `diffs[thread_count]` is kept, and the actual diff value is computed at the end of entire-grid iteration by simply summing all the per-thread diffs in the `diffs[]` array. The following pseudocode sums up the process:

```
begin loop1

    diffs[thread_count]  (initialize all elements of diffs[] to 0)

    #pragma omp parallel for
    for each red cell 'redc'
        tmp = redc
        redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num] = diffs[thread_num] + absolute(redc - tmp)
    endfor

    #pragma omp barrier

    #pragma omp parallel for
    for each black cell 'blackc'
        tmp = blackc
        blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num] = diffs[thread_num] + absolute(blackc - tmp)
    endfor

    diff = sum of all elements in diffs[]

    if (diff/num_cells < threshold): end loop1
```

Results

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main1 image01.jpg blur1.bmp
Execution time of gridSolver: 8.994484 seconds.
```

Execution Time = 8.99 seconds

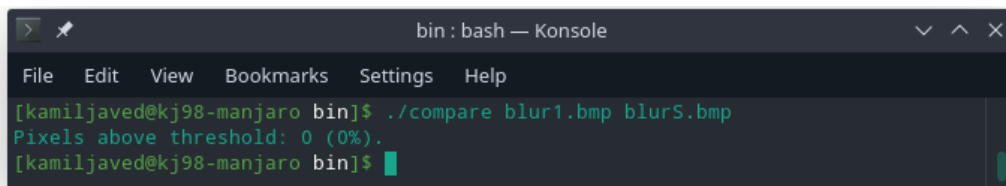
Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{8.99} = 0.62$$

There is no speedup, in fact the method still took longer time than sequential approach. The most probable cause of the delays is false-sharing here. False sharing is a term which applies when threads unwittingly impact the performance of each other while modifying independent variables sharing the same cache line.

False sharing occurs when another thread has written to an address within the cache line which causes the current thread's cache line to be flushed, and that other thread has written to a different address from the one that the current thread is referencing. Cache misses due to false sharing occur because the cache hardware operates at a cache-line granularity, not a data-word granularity.

Correctness

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]'. The command entered is './compare blur1.bmp blurS.bmp'. The output is 'Pixels above threshold: 0 (0%)'. The prompt is repeated on the next line.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur1.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```

Section II.1.c RED-BLACK CELLS APPROACH

Parallel Implementation with OpenMP using per-thread 'diff' (with padding)

Methodology

The false-sharing issue occurring due to using single array for per-thread diff without padding (would be contained within single cache line) can be avoided by changing the relevant data structures so that the different addresses referenced in each thread are on different cache lines.

The cache line-size for the CPU used was determined to be 64 Bytes. Hence, a padding of 60 Bytes (64 - sizeof(float) i.e. 4 = 60) was added along-with each element of the diffs array.

The following pseudocode sums up the process:

```
struct diffblk {
    float diff;
    char pad[60];
};

begin loop1

    struct diffblk diffs[thread_count];
    initialize all of diffs[].diff to 0

    #pragma omp parallel for
    for each red cell 'redc'
        tmp = redc
        redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num].diff = diffs[thread_num].diff + absolute(redc - tmp)
    endfor

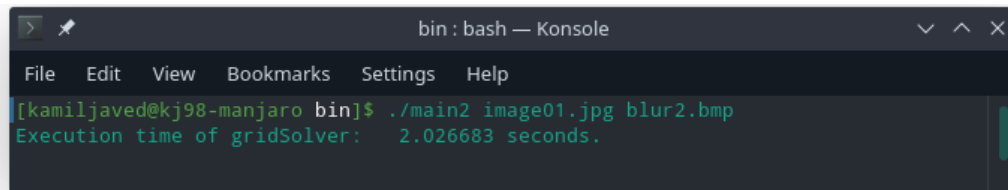
    #pragma omp barrier

    #pragma omp parallel for
    for each black cell 'blackc'
        tmp = blackc
        blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num].diff = diffs[thread_num].diff + absolute(blackc - tmp)
    endfor

    diff = sum of all diffs[].diff

    if (diff/num_cells < threshold): end loop1
```

Results

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]\$. The command './main2 image01.jpg blur2.bmp' has been executed, and the output is 'Execution time of gridSolver: 2.026683 seconds.'

Execution Time = 2.03 seconds

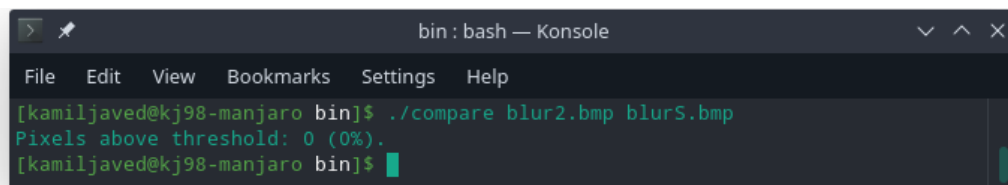
Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{2.03} = 2.76$$

$$\text{Speedup (compared to per_thread diff with no padding)} = \frac{8.99}{2.03} = 4.43$$

Speedup of 2.76 times is achieved compared to the sequential method. Considerable speedup is also achieved as compared to parallel approach using per-thread diff with no padding (false-sharing).

Correctness

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]\$. The command './compare blur2.bmp blurS.bmp' has been executed, and the output is 'Pixels above threshold: 0 (0%).'

Section II.2.a RED-BLACK CELLS APPROACH

Parallel Implementation with Pthreads using mutex lock

Methodology

The Pthreads library is used to update cells of one color in parallel in threads, using interleaved assignment (starting at thread_num for a thread, cells after step of thread_count assigned to that thread). Thread count was set to 4.

To avoid race condition on updating of the variable 'diff', the line(s) updating the 'diff' variable were wrapped inside a critical block.

The following pseudocode sums up the process (for a thread):

```
diff = 0

begin loop1 in thread_num=i

  for each row, starting at i, step by thread_count
    for each red cell 'redc' in row
      tmp = redc
      redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
      mutex_lock { diff = diff + absolute(redc - tmp) }
    endfor
  endfor

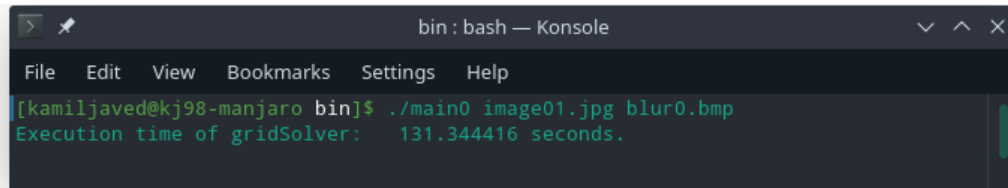
  barrier_wait()

  for each row, starting at i, step by thread_count
    for each black cell 'blackc' in row
      tmp = blackc
      blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
      mutex_lock { diff = diff + absolute(blackc - tmp) }
    endfor
  endfor

  barrier_wait()

if i==1:
  (test) if (diff/num_cells < threshold): end loop1 (for all threads)
  else diff = 0
  barrier_wait()
else:
  barrier_wait()
```

Results

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]\$'. The command entered is './main0 image01.jpg blur0.bmp'. The output is 'Execution time of gridSolver: 131.344416 seconds.'

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main0 image01.jpg blur0.bmp
Execution time of gridSolver: 131.344416 seconds.
```

Execution Time = 131.3 seconds

Speedups

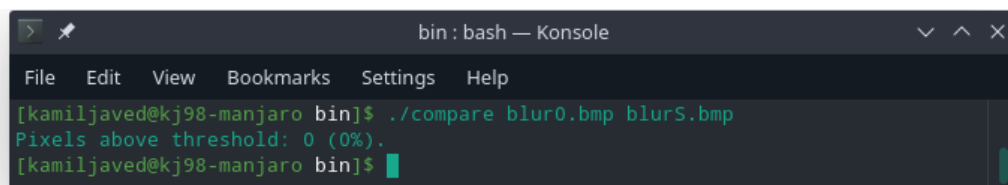
$$\text{Speedup (compared to sequential method)} = \frac{5.6}{131.3} = 0.04$$

$$\text{Speedup (compared to OpenMP pragma omp critical method)} = \frac{260.4}{131.3} = 1.98$$

There is no speedup compared to sequential method, in fact the method took longer time than sequential approach. This is because each thread must wait in a queue (in each iteration) to be able to update the 'diff' variable (as that is a critical section enclosed in mutex lock), i.e. the behavior effectively becomes sequential. And hence, the synchronization cost outweighs any benefit achieved by parallelizing here (using mutex lock).

The Pthread method (with mutex lock) has performed better than OpenMP approach using pragma omp critical, giving a speedup of 1.98 times compared to OpenMP.

Correctness

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]\$'. The command entered is './compare blur0.bmp blurS.bmp'. The output is 'Pixels above threshold: 0 (0%).' followed by a new prompt.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur0.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```

Section II.2.b

RED-BLACK CELLS APPROACH

Parallel Implementation with Pthreads using per-thread 'diff' (no padding)

Methodology

To avoid the blockage occurring at identified critical section i.e. updating 'diff', each thread is provided with a variable that it can use to store its personal diff-sum. Hence, an array `diffs[thread_count]` is kept, and the actual diff value is computed at the end of entire-grid iteration by simply summing all the per-thread diffs in the `diffs[]` array. The following pseudocode sums up the process:

```
diff = 0

diffs[thread_count]

begin loop1 in thread_num=i

    diffs[i-1] = 0

    for each row, starting at i, step by thread_count
        for each red cell 'redc' in row
            tmp = redc
            redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1] = diffs[i-1] + absolute(redc - tmp)
        endfor
    endfor

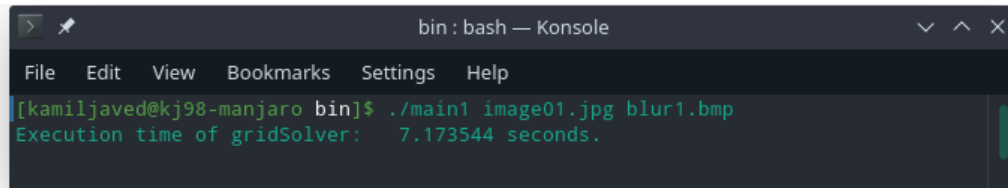
    barrier_wait()

    for each row, starting at i, step by thread_count
        for each black cell 'blackc' in row
            tmp = blackc
            blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1] = diffs[i-1] + absolute(blackc - tmp)
        endfor
    endfor

    barrier_wait()

if i==1:
    diff = sum of all elements in diffs[]
    (test) if (diff/num_cells < threshold): end loop1 (for all threads)
else diff = 0
    barrier_wait()
else:
    barrier_wait()
```

Results

A terminal window titled "bin : bash — Konsole" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is [kamiljaved@kj98-manjaro bin]\$. The command ./main1 image01.jpg blur1.bmp is entered. The output is Execution time of gridSolver: 7.173544 seconds.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main1 image01.jpg blur1.bmp
Execution time of gridSolver: 7.173544 seconds.
```

Execution Time = 7.17 seconds

Speedups

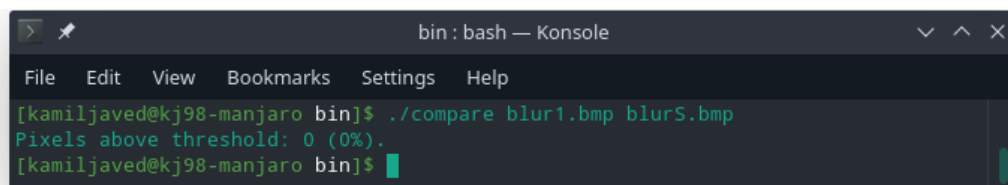
$$\text{Speedup (compared to sequential method)} = \frac{5.6}{7.17} = 0.78$$

$$\text{Speedup (compared to OpenMP per_thread diff no_padding method)} = \frac{8.99}{7.17} = 1.25$$

There is no speedup compared to sequential method, in fact the method still took longer time than sequential approach. The most probable cause of the delays is false-sharing here, as explained in Section II.1.b.

The Pthread method (with per-thread diff no-padding) has performed better than OpenMP approach using per-thread diff no-padding, giving a speedup of 1.25 times compared to OpenMP.

Correctness

A terminal window titled "bin : bash — Konsole" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is [kamiljaved@kj98-manjaro bin]\$. The command ./compare blur1.bmp blurS.bmp is entered. The output is Pixels above threshold: 0 (0%).

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur1.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```


Section II.2.c RED-BLACK CELLS APPROACH

Parallel Implementation with Pthreads using per-thread 'diff' (with padding)

Methodology

The false-sharing issue can be avoided by changing the relevant data structures so that the different addresses referenced in each thread are on different cache lines.

The cache line-size for the CPU used was determined to be 64 Bytes. Hence, a padding of 60 Bytes was used for each per-thread diff.

The following pseudocode sums up the process:

```
struct diffblk {
    float diff;
    char pad[60];
};

diff = 0

struct diffblk diffs[thread_count];

begin loop1 in thread_num=i

    diffs[i-1].diff = 0

    for each row, starting at i, step by thread_count
        for each red cell 'redc' in row
            tmp = redc
            redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1].diff = diffs[i-1].diff + absolute(redc - tmp)
        endfor
    endfor

    barrier_wait()

    for each row, starting at i, step by thread_count
        for each black cell 'blackc' in row
            tmp = blackc
            blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1].diff = diffs[i-1].diff + absolute(blackc - tmp)
        endfor
    endfor

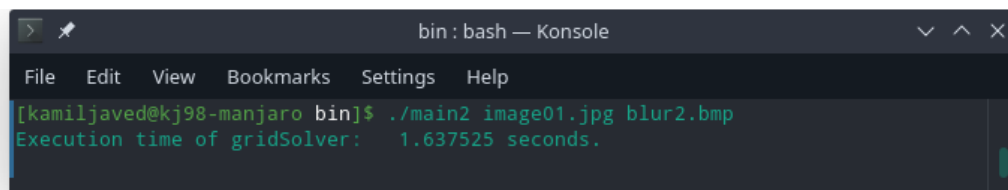
    barrier_wait()
```

```

if i==1:
    diff = sum of all diffs[].diff
    (test) if (diff/num_cells < threshold): end loop1 (for all threads)
    else diff = 0
    barrier_wait()
else:
    barrier_wait()

```

Results



```

bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main2 image01.jpg blur2.bmp
Execution time of gridSolver: 1.637525 seconds.

```

Execution Time = 1.63 seconds

Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{1.63} = 3.44$$

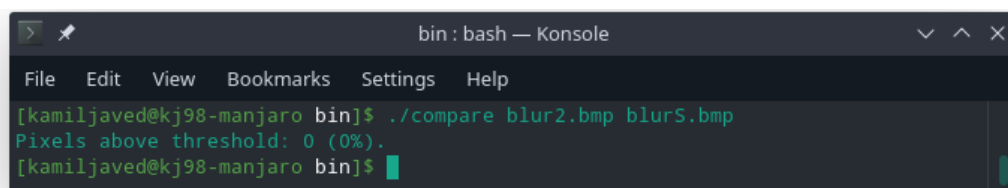
$$\text{Speedup (compared to per_thread diff with no padding)} = \frac{7.17}{1.63} = 4.4$$

$$\text{Speedup (compared to OpenMP per_thread diff with_padding method)} = \frac{2.03}{1.63} = 1.25$$

Speedup of 3.44 times is achieved compared to the sequential method. Considerable speedup is also achieved as compared to (Pthread) parallel approach using per-thread diff with no padding (false-sharing).

Also, the Pthread method (with per-thread diff with-padding) has performed better than OpenMP approach using per-thread diff with-padding, giving a speedup of 1.25 times compared to OpenMP.

Correctness



```

bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur2.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$

```

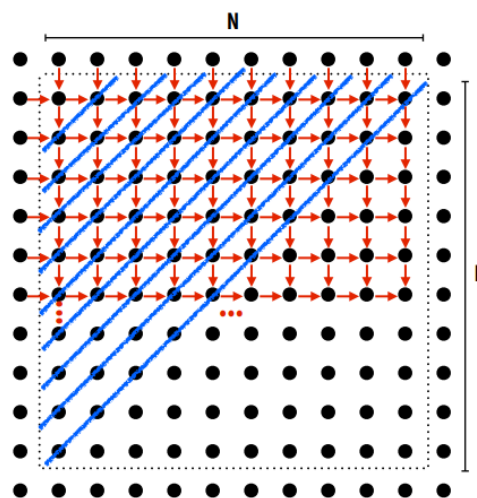
Section III.1

ANTI-DIAGONALS APPROACH

Sequential Implementation

Methodology

In this approach, the cells lying on an anti-diagonal are bunched into a group:



Update all cells in the first anti-diagonal, and move on to the next. Proceed until you reach the last anti-diagonal. Repeat until convergence.

The following pseudocode sums up the process (for sequential approach):

```
convert image to grayscale

begin loop1

    diff = 0

    for each anti-diagonal 'ad' in grid
        for each cell 'c' in 'ad'
            tmp = c
            c = 0.2 * (c + top_cell + bottom_cell + left_cell + right_cell)
            diff = diff + absolute(c - tmp)
        endfor
    endfor

    if (diff/num_cells < threshold): end loop1
```

Results

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main image01.jpg blurS.bmp
Execution time of gridSolver: 14.069267 seconds.
```

Execution Time = 14.1 seconds



Original Image



Resultant Image

Section III.1.a ANTI-DIAGONALS APPROACH

Parallel Implementation with OpenMP using “pragma omp critical”

Parallelization of Anti-Diagonals Approach

This approach can be parallelized by updating all the cells in a given anti-diagonal in parallel. After updating all cells in a given anti-diagonal in parallel, move on to the next anti-diagonal. Proceed until you reach the last anti-diagonal. Repeat until convergence.

Methodology

The OpenMP library is used to update cells on an anti-diagonal in parallel in threads, using interleaved assignment with an anti-diagonal (starting at thread_num for a thread, cells after step of thread_count assigned to that thread). Thread count was set to 4.

To avoid race condition on updating of the variable ‘diff’, the line(s) updating the ‘diff’ variable were wrapped inside a critical block.

The following pseudocode sums up the process:

```
convert image to grayscale

begin loop1

    diff = 0

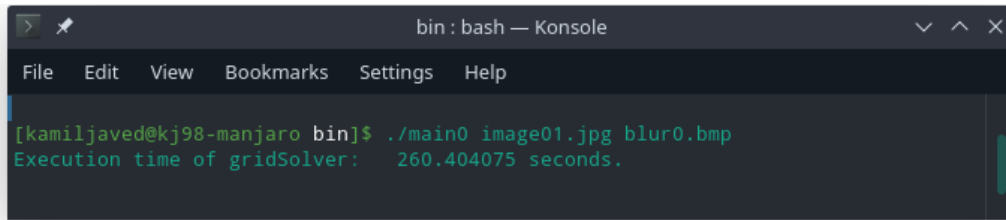
    #pragma omp parallel for
    for each red cell 'redc'
        tmp = redc
        redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
        #pragma omp critical { diff = diff + absolute(redc - tmp) }
    endfor

    #pragma omp barrier

    #pragma omp parallel for
    for each black cell 'blackc'
        tmp = blackc
        blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
        #pragma omp critical { diff = diff + absolute(blackc - tmp) }
    endfor

    if (diff/num_cells < threshold): end loop1
```

Results

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]'. The command './main0 image01.jpg blur0.bmp' has been executed, and the output is 'Execution time of gridSolver: 260.404075 seconds.'

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help

[kamiljaved@kj98-manjaro bin]$ ./main0 image01.jpg blur0.bmp
Execution time of gridSolver: 260.404075 seconds.
```

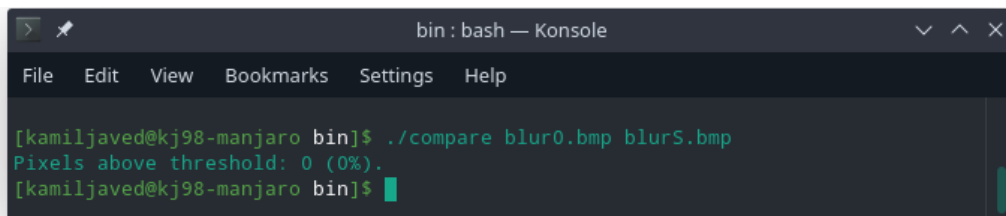
Execution Time = 260.4 seconds

Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{260.4} = 0.02$$

There is no speedup, in fact the method took longer time than sequential approach. This is because each thread must wait in a queue (in each iteration) to be able to update the 'diff' variable (as that is a critical section), i.e. the behavior effectively becomes sequential. And hence, the synchronization cost outweighs any benefit achieved by parallelizing here (using "pragma omp critical").

Correctness

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]'. The command './compare blur0.bmp blurS.bmp' has been executed, and the output is 'Pixels above threshold: 0 (0%).' followed by a new prompt.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help

[kamiljaved@kj98-manjaro bin]$ ./compare blur0.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```

Section III.1.b ANTI-DIAGONALS APPROACH

Parallel Implementation with OpenMP using per-thread 'diff' (no padding)

Methodology

To avoid the blockage occurring at identified critical section i.e. updating 'diff', each thread is provided with a variable that it can use to store its personal diff-sum. Hence, an array `diffs[thread_count]` is kept, and the actual diff value is computed at the end of entire-grid iteration by simply summing all the per-thread diffs in the `diffs[]` array. The following pseudocode sums up the process:

```
begin loop1

    diffs[thread_count]  (initialize all elements of diffs[] to 0)

    #pragma omp parallel for
    for each red cell 'redc'
        tmp = redc
        redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num] = diffs[thread_num] + absolute(redc - tmp)
    endfor

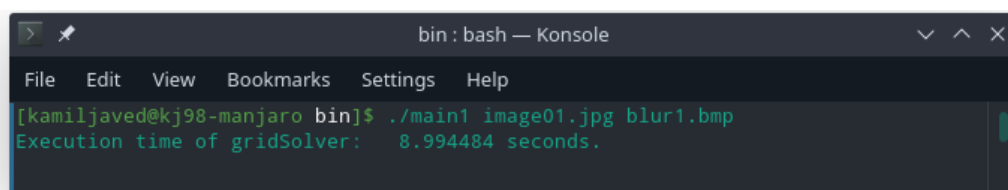
    #pragma omp barrier

    #pragma omp parallel for
    for each black cell 'blackc'
        tmp = blackc
        blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num] = diffs[thread_num] + absolute(blackc - tmp)
    endfor

    diff = sum of all elements in diffs[]

    if (diff/num_cells < threshold): end loop1
```

Results



```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main1 image01.jpg blur1.bmp
Execution time of gridSolver: 8.994484 seconds.
```

Execution Time = 8.99 seconds

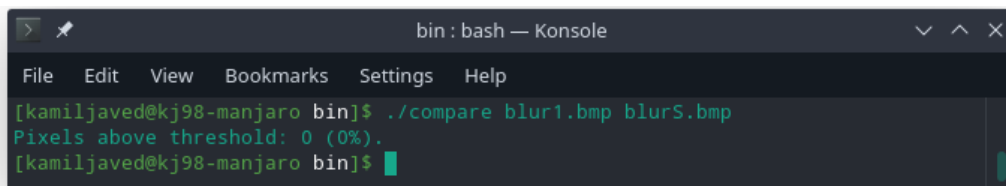
Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{8.99} = 0.62$$

There is no speedup, in fact the method still took longer time than sequential approach. The most probable cause of the delays is false-sharing here. False sharing is a term which applies when threads unwittingly impact the performance of each other while modifying independent variables sharing the same cache line.

False sharing occurs when another thread has written to an address within the cache line which causes the current thread's cache line to be flushed, and that other thread has written to a different address from the one that the current thread is referencing. Cache misses due to false sharing occur because the cache hardware operates at a cache-line granularity, not a data-word granularity.

Correctness

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The terminal shows a command prompt '[kamiljaved@kj98-manjaro bin]\$' followed by the command './compare blur1.bmp blurS.bmp'. The output is 'Pixels above threshold: 0 (0%).' followed by another prompt '[kamiljaved@kj98-manjaro bin]\$' with a green cursor.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur1.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```


Section III.1.c ANTI-DIAGONALS APPROACH

Parallel Implementation with OpenMP using per-thread 'diff' (with padding)

Methodology

The false-sharing issue occurring due to using single array for per-thread diff without padding (would be contained within single cache line) can be avoided by changing the relevant data structures so that the different addresses referenced in each thread are on different cache lines.

The cache line-size for the CPU used was determined to be 64 Bytes. Hence, a padding of 60 Bytes (64 - sizeof(float) i.e. 4 = 60) was added along-with each element of the diffs array.

The following pseudocode sums up the process:

```
struct diffblk {
    float diff;
    char pad[60];
};

begin loop1

    struct diffblk diffs[thread_count];
    initialize all of diffs[].diff to 0

    #pragma omp parallel for
    for each red cell 'redc'
        tmp = redc
        redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num].diff = diffs[thread_num].diff + absolute(redc - tmp)
    endfor

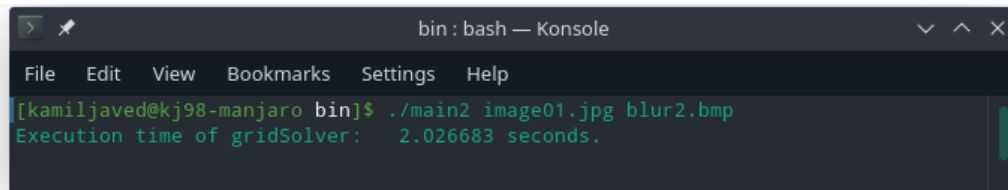
    #pragma omp barrier

    #pragma omp parallel for
    for each black cell 'blackc'
        tmp = blackc
        blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
        diffs[thread_num].diff = diffs[thread_num].diff + absolute(blackc - tmp)
    endfor

    diff = sum of all diffs[].diff

    if (diff/num_cells < threshold): end loop1
```

Results



```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main2 image01.jpg blur2.bmp
Execution time of gridSolver: 2.026683 seconds.
```

Execution Time = 2.03 seconds

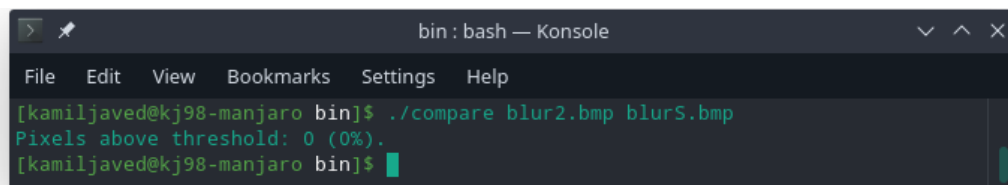
Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{2.03} = 2.76$$

$$\text{Speedup (compared to per_thread diff with no padding)} = \frac{8.99}{2.03} = 4.43$$

Speedup of 2.76 times is achieved compared to the sequential method. Considerable speedup is also achieved as compared to parallel approach using per-thread diff with no padding (false-sharing).

Correctness



```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur2.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```

Section III.2.a ANTI-DIAGONALS APPROACH

Parallel Implementation with Pthreads using mutex lock

Methodology

The Pthreads library is used to update cells of one color in parallel in threads, using interleaved assignment (starting at thread_num for a thread, cells after step of thread_count assigned to that thread). Thread count was set to 4.

To avoid race condition on updating of the variable 'diff', the line(s) updating the 'diff' variable were wrapped inside a critical block.

The following pseudocode sums up the process (for a thread):

```
diff = 0

begin loop1 in thread_num=i

  for each row, starting at i, step by thread_count
    for each red cell 'redc' in row
      tmp = redc
      redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
      mutex_lock { diff = diff + absolute(redc - tmp) }
    endfor
  endfor

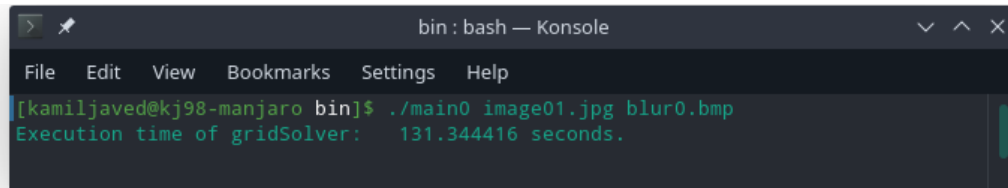
  barrier_wait()

  for each row, starting at i, step by thread_count
    for each black cell 'blackc' in row
      tmp = blackc
      blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
      mutex_lock { diff = diff + absolute(blackc - tmp) }
    endfor
  endfor

  barrier_wait()

if i==1:
  (test) if (diff/num_cells < threshold): end loop1 (for all threads)
else diff = 0
  barrier_wait()
else:
  barrier_wait()
```

Results

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]\$'. The command entered is './main0 image01.jpg blur0.bmp'. The output is 'Execution time of gridSolver: 131.344416 seconds.'

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main0 image01.jpg blur0.bmp
Execution time of gridSolver: 131.344416 seconds.
```

Execution Time = 131.3 seconds

Speedups

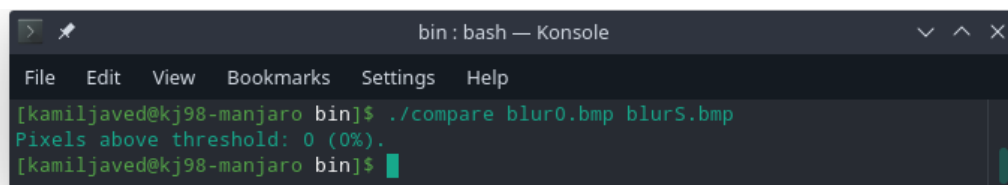
$$\text{Speedup (compared to sequential method)} = \frac{5.6}{131.3} = 0.04$$

$$\text{Speedup (compared to OpenMP pragma omp critical method)} = \frac{260.4}{131.3} = 1.98$$

There is no speedup compared to sequential method, in fact the method took longer time than sequential approach. This is because each thread must wait in a queue (in each iteration) to be able to update the 'diff' variable (as that is a critical section enclosed in mutex lock), i.e. the behavior effectively becomes sequential. And hence, the synchronization cost outweighs any benefit achieved by parallelizing here (using mutex lock).

The Pthread method (with mutex lock) has performed better than OpenMP approach using pragma omp critical, giving a speedup of 1.98 times compared to OpenMP.

Correctness

A terminal window titled 'bin : bash — Konsole' with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is '[kamiljaved@kj98-manjaro bin]\$'. The command entered is './compare blur0.bmp blurS.bmp'. The output is 'Pixels above threshold: 0 (0%).' followed by a new prompt.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur0.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```

Section III.2.b ANTI-DIAGONALS APPROACH

Parallel Implementation with Pthreads using per-thread 'diff' (no padding)

Methodology

To avoid the blockage occurring at identified critical section i.e. updating 'diff', each thread is provided with a variable that it can use to store its personal diff-sum. Hence, an array `diffs[thread_count]` is kept, and the actual diff value is computed at the end of entire-grid iteration by simply summing all the per-thread diffs in the `diffs[]` array. The following pseudocode sums up the process:

```
diff = 0

diffs[thread_count]

begin loop1 in thread_num=i

    diffs[i-1] = 0

    for each row, starting at i, step by thread_count
        for each red cell 'redc' in row
            tmp = redc
            redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1] = diffs[i-1] + absolute(redc - tmp)
        endfor
    endfor

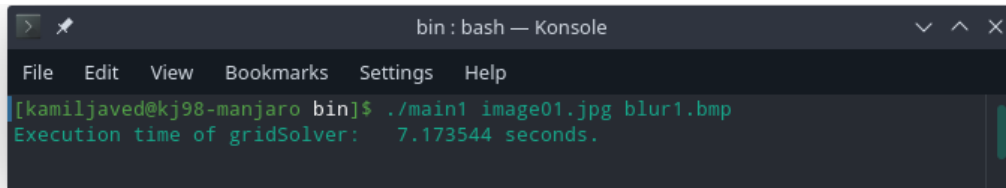
    barrier_wait()

    for each row, starting at i, step by thread_count
        for each black cell 'blackc' in row
            tmp = blackc
            blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1] = diffs[i-1] + absolute(blackc - tmp)
        endfor
    endfor

    barrier_wait()

if i==1:
    diff = sum of all elements in diffs[]
    (test) if (diff/num_cells < threshold): end loop1 (for all threads)
else diff = 0
    barrier_wait()
else:
    barrier_wait()
```

Results

A terminal window titled "bin : bash — Konsole" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is [kamiljaved@kj98-manjaro bin]\$. The command entered is ./main1 image01.jpg blur1.bmp. The output is Execution time of gridSolver: 7.173544 seconds.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main1 image01.jpg blur1.bmp
Execution time of gridSolver: 7.173544 seconds.
```

Execution Time = 7.17 seconds

Speedups

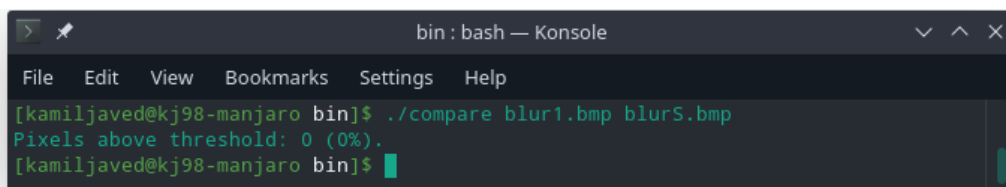
$$\text{Speedup (compared to sequential method)} = \frac{5.6}{7.17} = 0.78$$

$$\text{Speedup (compared to OpenMP per_thread diff no_padding method)} = \frac{8.99}{7.17} = 1.25$$

There is no speedup compared to sequential method, in fact the method still took longer time than sequential approach. The most probable cause of the delays is false-sharing here, as explained in Section II.1.b.

The Pthread method (with per-thread diff no-padding) has performed better than OpenMP approach using per-thread diff no-padding, giving a speedup of 1.25 times compared to OpenMP.

Correctness

A terminal window titled "bin : bash — Konsole" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The prompt is [kamiljaved@kj98-manjaro bin]\$. The command entered is ./compare blur1.bmp blurS.bmp. The output is Pixels above threshold: 0 (0%). The prompt is [kamiljaved@kj98-manjaro bin]\$.

```
bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur1.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$
```

Section III.2.c ANTI-DIAGONALS APPROACH

Parallel Implementation with Pthreads using per-thread 'diff' (with padding)

Methodology

The false-sharing issue can be avoided by changing the relevant data structures so that the different addresses referenced in each thread are on different cache lines.

The cache line-size for the CPU used was determined to be 64 Bytes. Hence, a padding of 60 Bytes was used for each per-thread diff.

The following pseudocode sums up the process:

```
struct diffblk {
    float diff;
    char pad[60];
};

diff = 0

struct diffblk diffs[thread_count];

begin loop1 in thread_num=i

    diffs[i-1].diff = 0

    for each row, starting at i, step by thread_count
        for each red cell 'redc' in row
            tmp = redc
            redc = 0.2 * (redc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1].diff = diffs[i-1].diff + absolute(redc - tmp)
        endfor
    endfor

    barrier_wait()

    for each row, starting at i, step by thread_count
        for each black cell 'blackc' in row
            tmp = blackc
            blackc = 0.2 * (blackc + top_cell + bottom_cell + left_cell + right_cell)
            diffs[i-1].diff = diffs[i-1].diff + absolute(blackc - tmp)
        endfor
    endfor

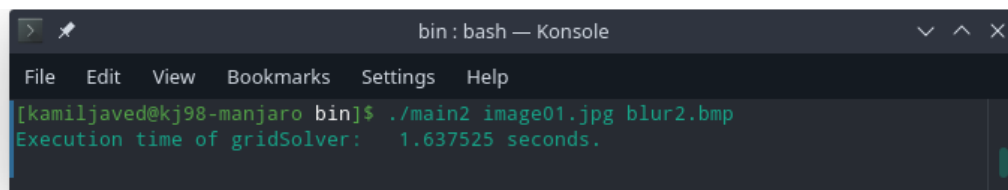
    barrier_wait()
```

```

if i==1:
    diff = sum of all diffs[].diff
    (test) if (diff/num_cells < threshold): end loop1 (for all threads)
    else diff = 0
    barrier_wait()
else:
    barrier_wait()

```

Results



```

bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./main2 image01.jpg blur2.bmp
Execution time of gridSolver: 1.637525 seconds.

```

Execution Time = 1.63 seconds

Speedups

$$\text{Speedup (compared to sequential method)} = \frac{5.6}{1.63} = 3.44$$

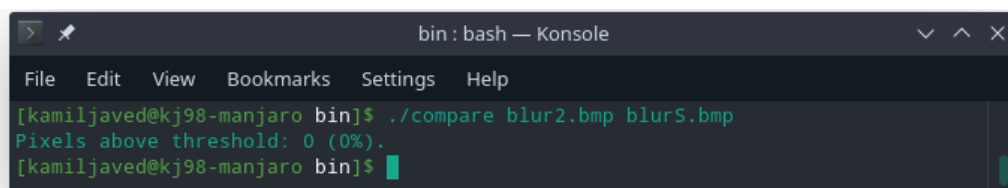
$$\text{Speedup (compared to per_thread diff with no padding)} = \frac{7.17}{1.63} = 4.4$$

$$\text{Speedup (compared to OpenMP per_thread diff with_padding method)} = \frac{2.03}{1.63} = 1.25$$

Speedup of 3.44 times is achieved compared to the sequential method. Considerable speedup is also achieved as compared to (Pthread) parallel approach using per-thread diff with no padding (false-sharing).

Also, the Pthread method (with per-thread diff with-padding) has performed better than OpenMP approach using per-thread diff with-padding, giving a speedup of 1.25 times compared to OpenMP.

Correctness



```

bin : bash — Konsole
File Edit View Bookmarks Settings Help
[kamiljaved@kj98-manjaro bin]$ ./compare blur2.bmp blurS.bmp
Pixels above threshold: 0 (0%).
[kamiljaved@kj98-manjaro bin]$

```