Jesse Coulson

CSCI 551 Spring 2023

Final Parallel Program

OpenMP vs MPI with Sobel Edge Detection

Numerical and Parallel Problem Description

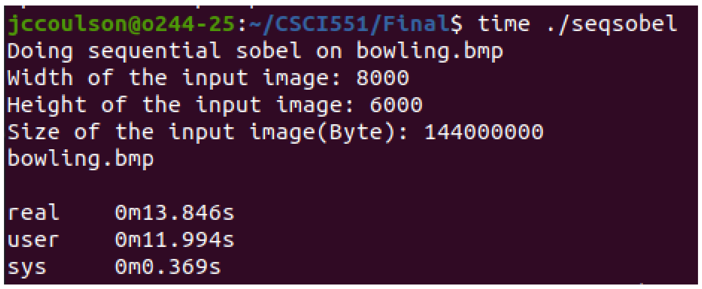
The program focuses on parallelizing the Sobel edge detection process on input images, and outputting an edge map. This optimization targets the vector operations that are required in Sobel transformations to change each pixel in the image.

Edge detection is important in many domains. In the realm of medical imaging, edge detection is a large part for identifying things such as tumors. Another application of edge detection is in autonomous vehicles. It is used for recognizing obstacles in real time which is necessary for it to operate.

The primary acceleration in this program is achieved by parallelizing tasks involving vector operations on an array of RGB values. So the main numerical methods applied center around vector operations and transformations. To facilitate this acceleration, parallel programming techniques using both MPI and OpenMP were used. A comprehensive build of all programs and verification of their outputs can be found in the appendix.

Sequential solution and computation time

Run of sequential sobel:



```
jccoulson@o244-25:~/CSCI551/Final$ time ./seqsobel
Doing sequential sobel on bowling.bmp
Width of the input image: 8000
Height of the input image: 6000
Size of the input image(Byte): 144000000
bowling.bmp

real    0m13.846s
user    0m11.994s
sys     0m0.369s
```

Times from 10 runs:

13.846, 13.895, 13.939, 13.786, 13.905, 13.931, 13.895, 13.890, 13.932, 13.905

Average time: **13.8924**

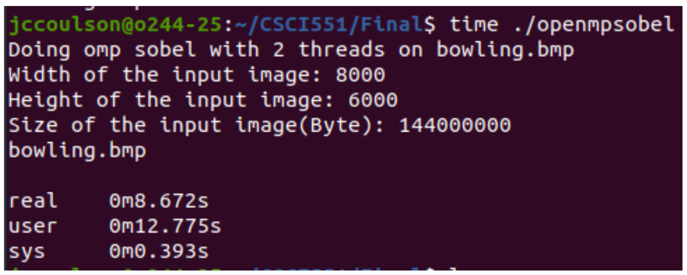
Parallel Design and solution with computation time

For parallelization I utilized both OpenMP and MPI. OpenMP offered a direct approach to parallelization. By just using an 'omp for' directive in the loop responsible for the Sobel transformation of each pixel, a significant speedup was achieved.

In contrast, MPI parallelization was much more complex. At a high level, RGB pixel values were sourced from the input image and allocated to rank 0. Subsequently, this rank broadcasted the data to all other ranks. Each rank was then assigned an evenly distributed segment of the RGB array. Each rank processed the Sobel transformation on its designated portion. Post-processing, MPI_Gather consolidated all the data segments into a singular array for rank 0, which then outputs the edge map image.

For performance metrics, I've presented the runtime for configurations with both 2 and 4 nodes/cores. Given the substantial speedup in the overall program, I used the './time' tool instead of relying on POSIX clock or MPI_Wtime, highlighting the overall acceleration achieved.

Run of OpenMP with 2 threads:



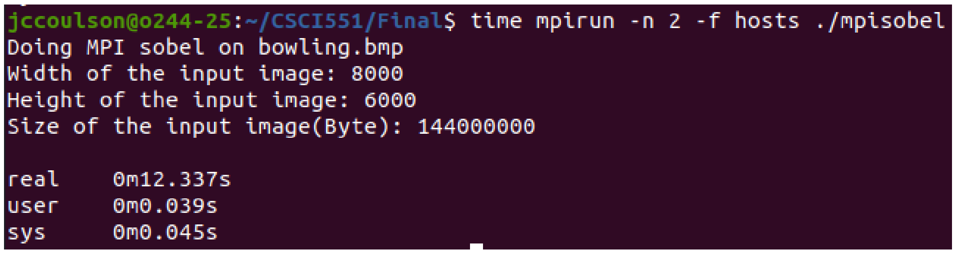
```
jccoulson@o244-25:~/CSCI551/Final$ time ./openmpsobel
Doing omp sobel with 2 threads on bowling.bmp
Width of the input image: 8000
Height of the input image: 6000
Size of the input image(Byte): 144000000
bowling.bmp

real    0m8.672s
user    0m12.775s
sys     0m0.393s
```

Time for 10 runs:

8.672, 8.974, 8.464, 8.531, 8.545, 8.487, 8.460, 8.418, 8.491, 8.456

Average time: **8.5498**

Run of MPI with 2 nodes:



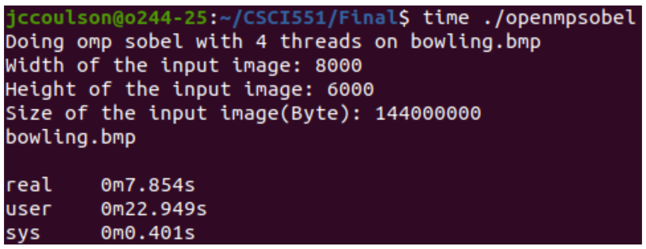
```
jccoulson@o244-25:~/CSCI551/Final$ time mpirun -n 2 -f hosts ./mpisobel
Doing MPI sobel on bowling.bmp
Width of the input image: 8000
Height of the input image: 6000
Size of the input image(Byte): 144000000

real    0m12.337s
user    0m0.039s
sys     0m0.045s
```

Time for 10 runs:

12.337, 11.101, 10.384, 10.992, 10.400, 10.980, 11.025, 11.030, 11.020, 10.942

Average time **11.0211**

Run of OpenMP with 4 threads:



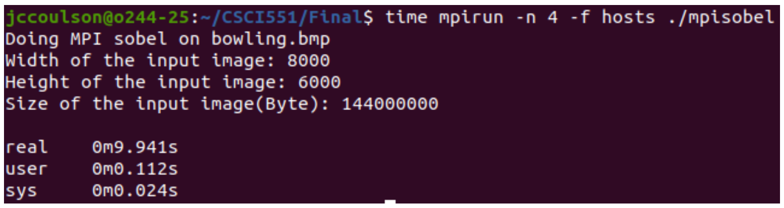
```
jccoulson@o244-25:~/CSCI551/Final$ time ./openmpsobel
Doing omp sobel with 4 threads on bowling.bmp
Width of the input image: 8000
Height of the input image: 6000
Size of the input image(Byte): 144000000
bowling.bmp

real    0m7.854s
user    0m22.949s
sys     0m0.401s
```

Times for 10 runs:

7.854, 7.896, 7.836, 7.839, 7.893, 8.020, 7.926, 7.906, 7.875, 7.774

Average time: **7.8819**

Run of MPI with 4 nodes:



```
jccoulson@o244-25:~/CSCI551/Final$ time mpirun -n 4 -f hosts ./mpisobel
Doing MPI sobel on bowling.bmp
Width of the input image: 8000
Height of the input image: 6000
Size of the input image(Byte): 144000000

real    0m9.941s
user    0m0.112s
sys     0m0.024s
```

Times for 10 runs:

9.941, 9.883, 10.400, 10.930, 9.883, 10.435, 10.046, 10.692, 10.997, 10.394

Average time: **10.3601**

The scaling factor for the following equations was obtained through the number of cores and nodes. For OpenMP I used the argument for number of threads at 2 and 4, on a 4 core system on an ecc-linux node. For MPI I used 4 separate nodes each using one core on the ecc-cluster.

Speedup and parallel Portion for 2 nodes/cores:

OpenMP:

Speedup = Tseq/Tpar = 13.8924/8.5498 = **1.62** times faster

P = (S(1-(1/SU)))/(S-1) = (2(1-(1/1.62)))/(2-1) = **.765** portion of code is parallel

MPI:

Speedup = Tseq/Tpar = 13.8924/11.0211 = **1.26** times faster

P = (S(1-(1/SU)))/(S-1) = (2(1-(1/1.26)))/(2-1) = **.413** portion of the code is parallel

Speedup and parallel portion for 4 nodes/cores:

OpenMP:

Speedup = Tseq/Tpar = 13.8924/7.8819 = **1.76** times faster

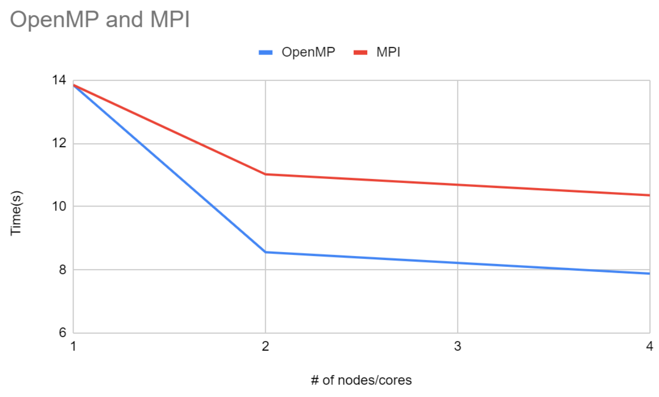
P = (S(1-(1/SU)))/(S-1) = (4(1-(1/1.76)))/(4-1) = **.576** portion of code is parallel

MPI:

Speedup = Tseq/Tpar = 13.8924/10.3601 = 1.34 times faster

P = (S(1-(1/SU)))/(S-1) = (4(1-(1/1.34)))/(4-1) = **.338** portion of the code is parallel

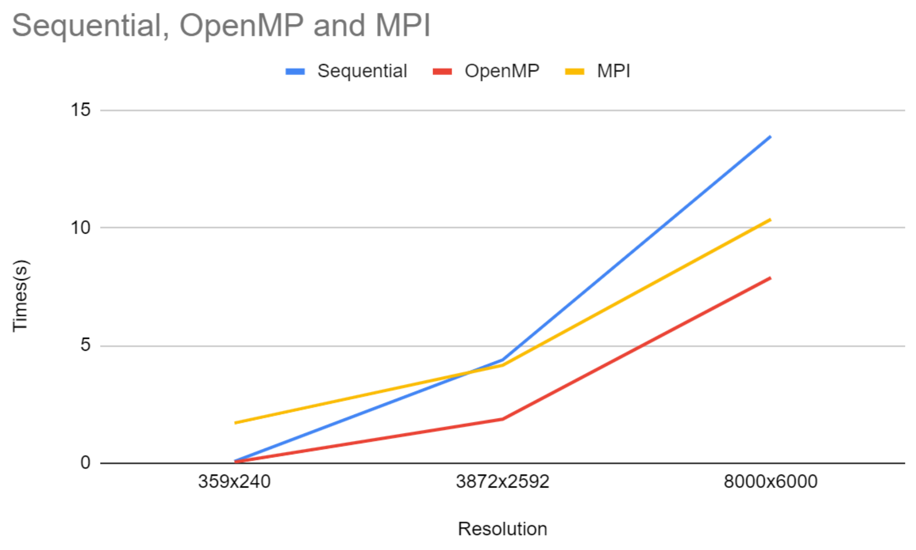
OpenMP vs MPI graph



Both OpenMP and MPI offer substantial speedup, but OpenMP is a bit better. The reason for this is the way the Sobel algorithm works, it's very easily parallelized since it's just using simple matrix operations. OpenMP efficiently handles this parallelism for the transformation, while MPI has to deal with the added overhead of message passing. Despite the overhead, MPI's parallelization still achieves a good speedup. The speedup realized doesn't hit the linear ideal of 2x and 4x, but it's still significant.

I will show problem scaling using four nodes/threads across three image resolutions: Bears.bmp(359,240), swan.bmp(3872, 2592), and bowling.bmp(8000,6000). The project directory contains all the referenced images. Detailed timed runs across these resolutions can be found in the appendix.

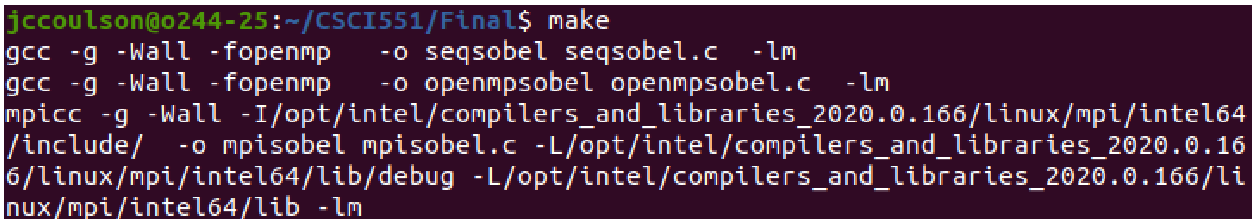
Graph of problem scaling:



As you can see from this graph OpenMP is always faster than MPI and sequential, and MPI does not start having speedup until the resolution of swan.bmp. After the problem scaling has overcome the overhead cost for MPI, MPI starts to speedup proportionally as much as OpenMP, while the sequential version continues to take more time as the resolution increases.

Appendix

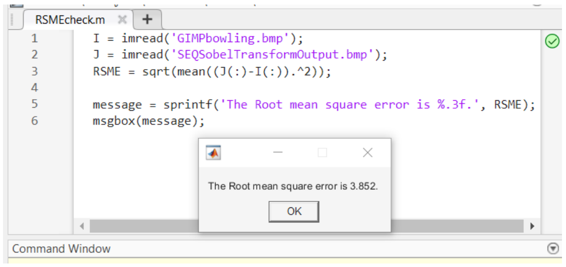
Build of programs:



```
jccoulson@o244-25:~/CSCI551/Final$ make
gcc -g -Wall -fopenmp   -o seqsobel seqsobel.c  -lm
gcc -g -Wall -fopenmp   -o openmpsobel openmpsobel.c  -lm
mpicc -g -Wall -I/opt/intel/compilers_and_libraries_2020.0.166/linux/mpi/intel64
/include/  -o mpisobel mpisobel.c -L/opt/intel/compilers_and_libraries_2020.0.16
6/linux/mpi/intel64/lib/debug -L/opt/intel/compilers_and_libraries_2020.0.166/li
nux/mpi/intel64/lib -lm
```

Verification of sequential:

I've included a matlab file to calculate RMSE between two images called RSMEcheck.m



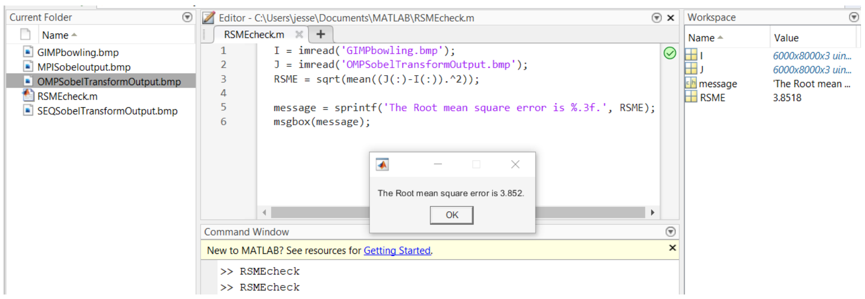
```
I = imread('GIMPbowling.bmp');
J = imread('SEQSobelTransformOutput.bmp');
RSME = sqrt(mean((J(:)-I(:)).^2));

message = sprintf('The Root mean square error is %.3f.', RSME);
msgbox(message);
```

The Root mean square error is 3.852.

The RSME being 3.852 means the images are very close to one another. The reason they are a little off is because there are probably slight differences in the implementation of the Sobel algorithm.

Verification of parallel:

OpenMP:



Current Folder
Name
GIMPbowling.bmp
MPISobeloutput.bmp
OMPSobelTransformOutput.bmp
RSMEcheck.m
SEQSobelTransformOutput.bmp
Editor - C:\Users\jesse\Documents\MATLAB\RSMEcheck.m
RSMEcheck.m

```
I = imread('GIMPbowling.bmp');
J = imread('OMPSobelTransformOutput.bmp');
RSME = sqrt(mean((J(:)-I(:)).^2));

message = sprintf('The Root mean square error is %.3f.', RSME);
msgbox(message);
```

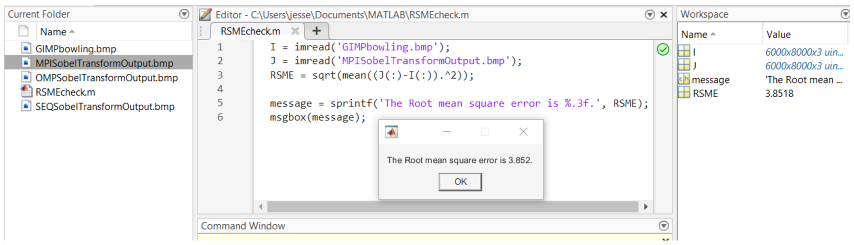
The Root mean square error is 3.852.

New to MATLAB? See resources for Getting Started.

```
>> RSMEcheck
>> RSMEcheck
```

Name	Value
I	6000x8000x3 uin...
J	6000x8000x3 uin...
message	'The Root mean ...
RSME	3.8518

MPI:


Current Folder
Name
GIMPbowling.bmp
MPISobelTransformOutput.bmp
OMPSobelTransformOutput.bmp
RSMEcheck.m
SEQSobelTransformOutput.bmp
Editor - C:\Users\jesse\Documents\MATLAB\RSMEcheck.m
RSMEcheck.m
Workspace
Name
Value
Command Window
OK

```
I = imread('GIMPbowling.bmp');
J = imread('MPISobelTransformOutput.bmp');
RSME = sqrt(mean((J(:)-I(:)).^2));

message = sprintf('The Root mean square error is %.3f.', RSME);
msgbox(message);
```

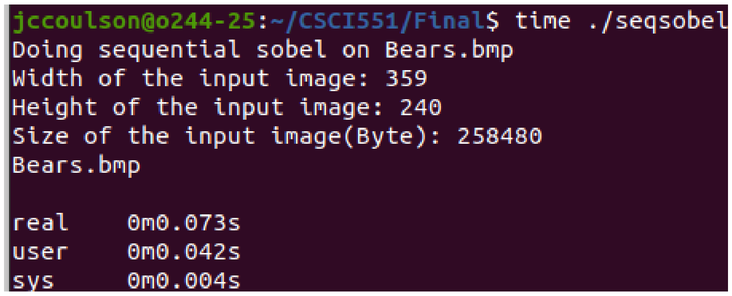
The Root mean square error is 3.852.

Name	Value
I	6000x8000x3 uin...
J	6000x8000x3 uin...
message	'The Root mean ...
RSME	3.8518

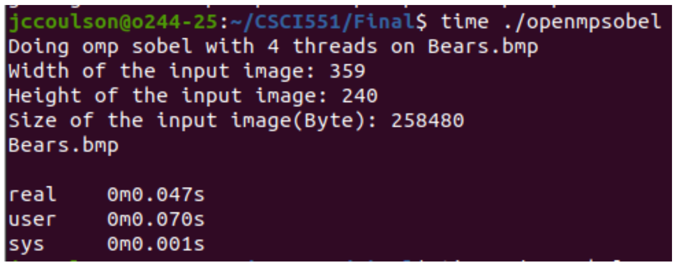
This verifies that all the parallel images have the same output as sequential and are fairly accurate to GIMP’s sobel.

Problem scaling times:

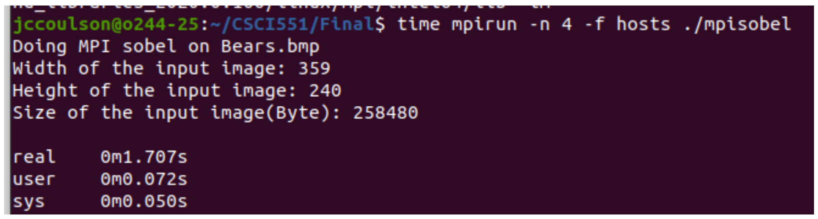
Times for Bears.bmp



```
jccoulson@o244-25:~/CSCI551/Final$ time ./seqsobel
Doing sequential sobel on Bears.bmp
Width of the input image: 359
Height of the input image: 240
Size of the input image(Byte): 258480
Bears.bmp

real    0m0.073s
user    0m0.042s
sys     0m0.004s
```

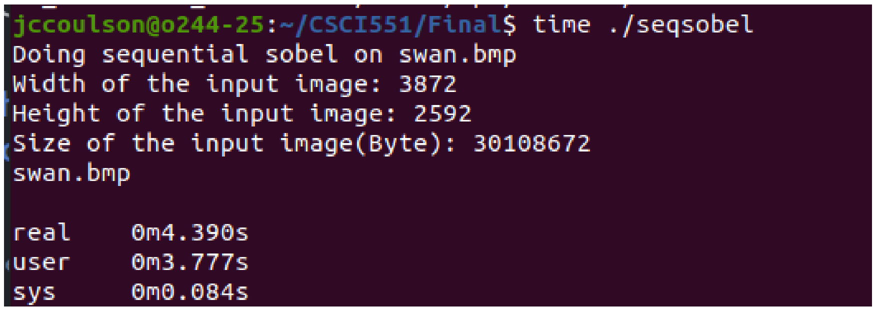


```
jccoulson@o244-25:~/CSCI551/Final$ time ./openmpsobel
Doing omp sobel with 4 threads on Bears.bmp
Width of the input image: 359
Height of the input image: 240
Size of the input image(Byte): 258480
Bears.bmp

real    0m0.047s
user    0m0.070s
sys     0m0.001s
```

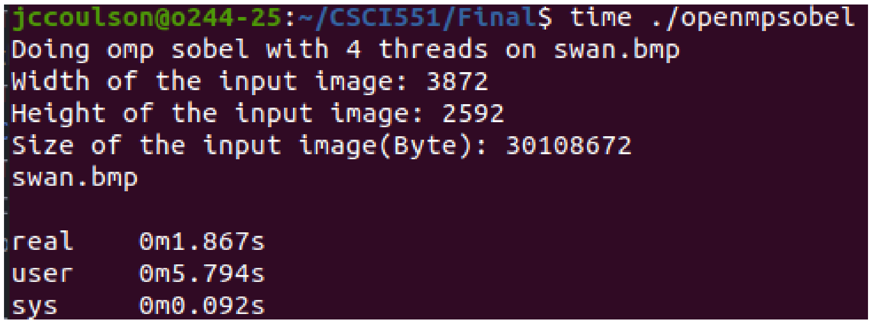


```
jccoulson@o244-25:~/CSCI551/Final$ time mpirun -n 4 -f hosts ./mpisobel
Doing MPI sobel on Bears.bmp
Width of the input image: 359
Height of the input image: 240
Size of the input image(Byte): 258480

real    0m1.707s
user    0m0.072s
sys     0m0.050s
```

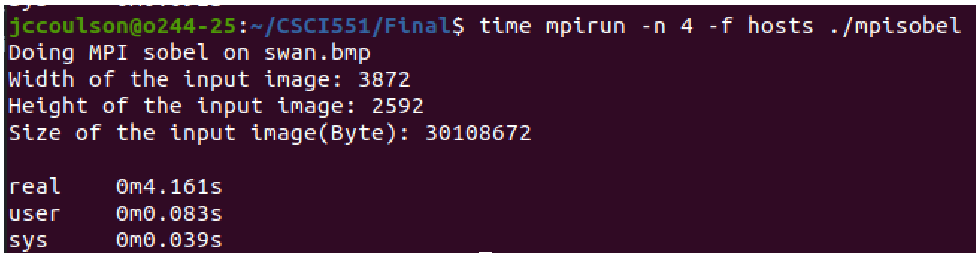
Times for swan.bmp:



```
jccoulson@o244-25:~/CSCI551/Final$ time ./seqsobel
Doing sequential sobel on swan.bmp
Width of the input image: 3872
Height of the input image: 2592
Size of the input image(Byte): 30108672
swan.bmp

real	0m4.390s
user	0m3.777s
sys	0m0.084s
```



```
jccoulson@o244-25:~/CSCI551/Final$ time ./openmpsobel
Doing omp sobel with 4 threads on swan.bmp
Width of the input image: 3872
Height of the input image: 2592
Size of the input image(Byte): 30108672
swan.bmp

real	0m1.867s
user	0m5.794s
sys	0m0.092s
```



```
jccoulson@o244-25:~/CSCI551/Final$ time mpirun -n 4 -f hosts ./mpisobel
Doing MPI sobel on swan.bmp
Width of the input image: 3872
Height of the input image: 2592
Size of the input image(Byte): 30108672

real	0m4.161s
user	0m0.083s
sys	0m0.039s
```

Times for bowling.bmp found in report.