Steven Comer

CS 1645

HW 6

2 April 2014

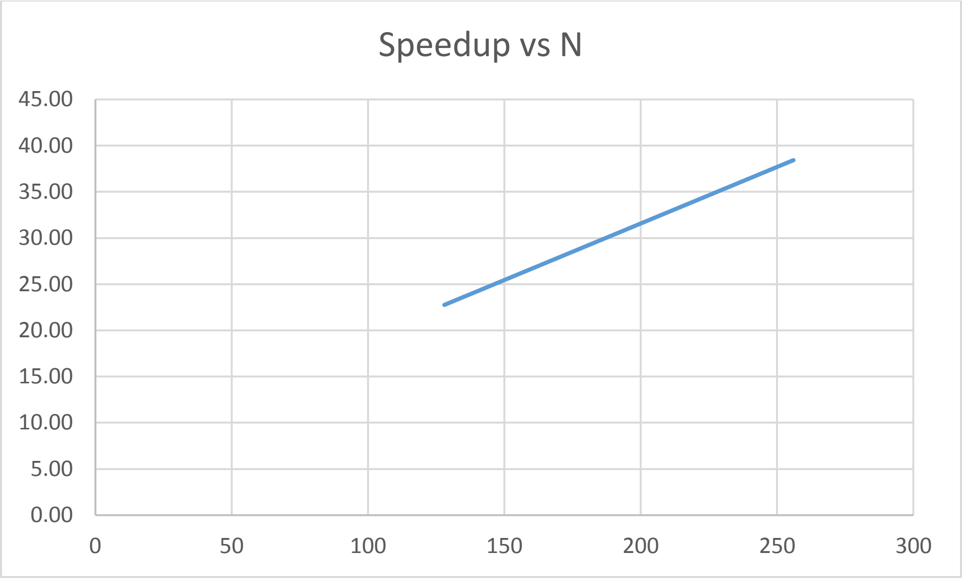
STENCIL

1. This is a general strategy of the parallelization effort. Why did you choose those directives to parallelize the program?

> This program contains six for loops, two instances of a triple nested for loop. For this reason, I chose to use the directive “#pragma acc kernels” in order to parallelize the code. There were two different possible place to insert this directive. I tested multiple combinations and placements of the directives, while keeping in mind the data dependencies inherent in the program. I also added the directive “#pragma acc data copy(A), create(Anew)” on the outermost while loop. This forces the accelerator to create Anew on its local memory and prevents unnecessary copying to and from the host node.

2. This is a speedup analysis. Use 2 different interesting values of N. Runtimes are measured in seconds.

N	***Seq Time***	***OpenAcc Time***	***Speedup***
128	25.26	1.11	22.76
256	259.14	6.74	38.45



3. This is a description of the performance bottlenecks. What is preventing the program from getting linear speedup?

The first set of nested for loops which actually performs the calculations is the bottleneck. This calculation includes dependencies on neighboring cells. These data dependencies prevent the program from getting linear speedup.

IMPLICIT

1. This is a general strategy of the parallelization effort. Why did you choose those directives to parallelize the program?

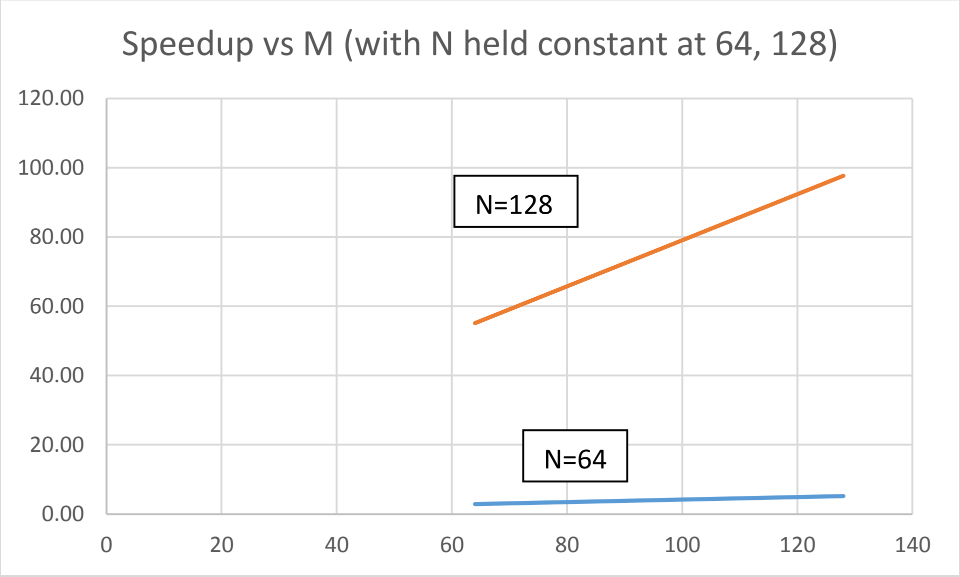
This program initially used a scattering algorithm to update the array “grid”. I changed the algorithm to a gathering algorithm by rearranging the nested for loops. This allows different threads to simultaneously and independently update different parts of the “grid” array. This greatly increases the amount of parallelization possible. I then added the directive “#pragma acc kernels” to the outermost of the 4 nested for loops. The compiler is able to parallelize the outer 3 loops using this process.

2. This is a speedup analysis. Use 2 different interesting values of N and M. The times are measured in seconds.

Sequential Time (N,M)	**64**	**128**
64	0.83	1.65
128	17.08	34.17

OpenAcc Time (N,M)	**64**	**128**
64	0.29	0.32
128	0.31	0.35

Speedup(N,M)	**64**	**128**
64	2.86	5.16
128	55.10	97.63



3. This is a description of the performance bottlenecks. What is preventing the program from getting linear speedup?

The performance bottleneck is the innermost for loop which updates “grid[i][j][k]”. This for loop must be executed sequentially because of dependencies and backward dependencies with “grid”. The outer three for loops are executed in parallel after the scatter to gather transformation and the application of the kernels pragma directive.