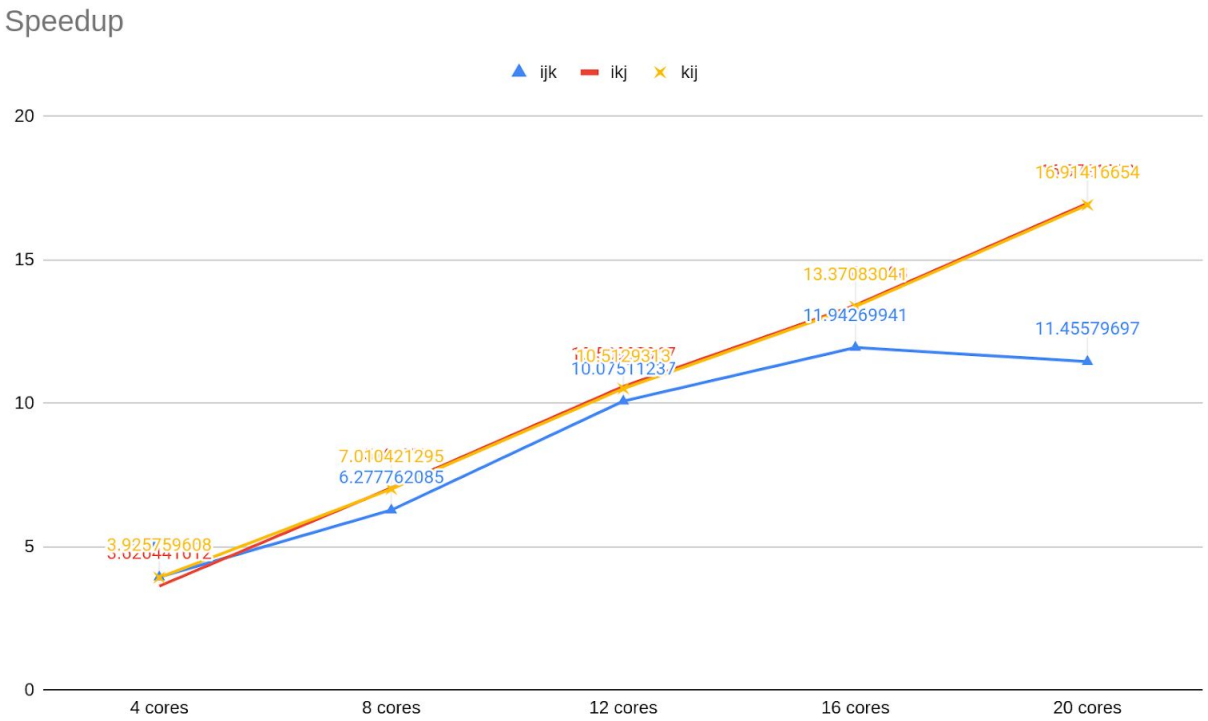
Jeff Anderson
Csci 551
4/19/2020

Matrix Multiplication IJK forms with MPI

Timing of 54 runs with the minimum speed in bold

	1 core	4 cores	8 cores	12 cores	16 cores	20 cores
ijk	**2355.793**	**597.31**	**375.26**	**233.823**	212.744	**205.642**
	2358.133	740.392	405.85	299.201	215.218	217.467
	2358.521	706.107	384.14	248.393	**197.258**	304.34
ikj	**1005.275**	315.648	143.884	**94.927**	**74.921**	116.895
	1006.704	277.18	**142.57**	95.111	75.401	119.14
	1006.601	**277.207**	143.156	99.628	76.431	**59.231**
kij	1009.869	257.21	**144.032**	96.865	76.136	60.569
	1009.725	278.458	144.539	97.826	76.413	59.716
	1010.156	**257.205**	144.446	**96.046**	**75.517**	**59.697**

Speedup Graph of IJK Forms

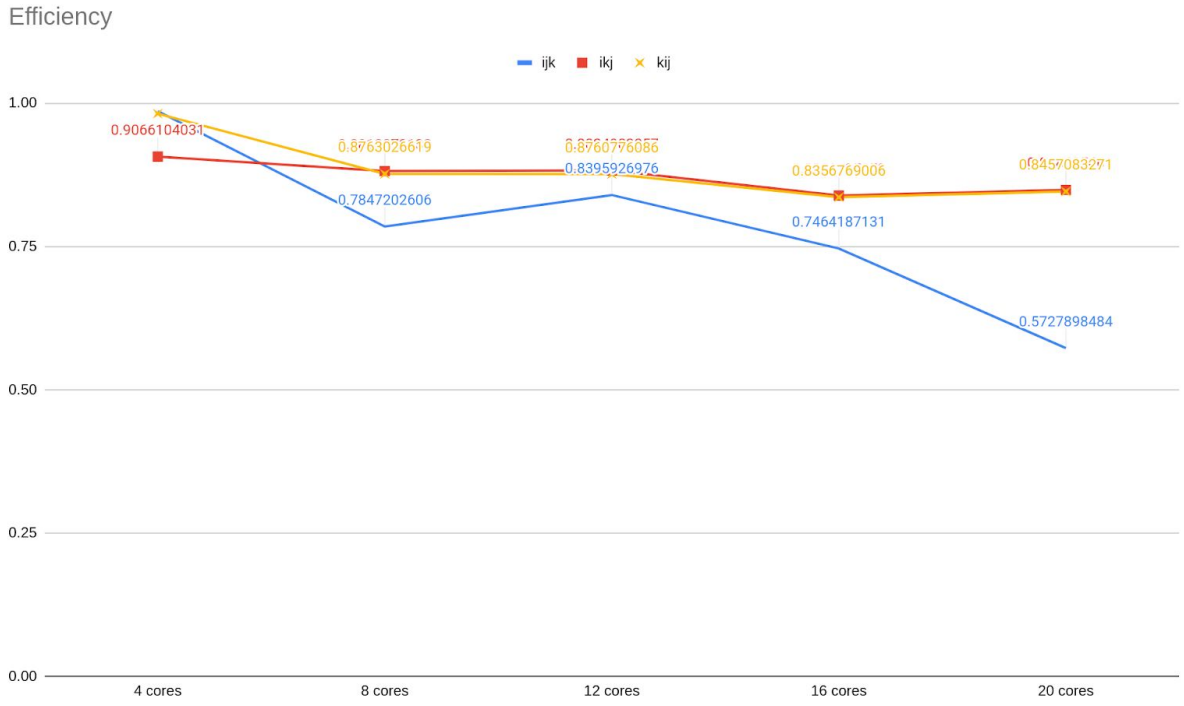


The ijk form seems to fall off quickly while the ikj and kij form remain much more linear.

Jeff Anderson
Csci 551
4/19/2020

Speedup					
	4 cores	8 cores	12 cores	16 cores	20 cores
ijk	**3.942676332**	**6.277762085**	**10.07511237**	**11.94269941**	**11.45579697**
ikj	**3.626441612**	**7.051097706**	**10.58997967**	**13.41780008**	**16.9721092**
kij	**3.925759608**	**7.010421295**	**10.5129313**	**13.37083041**	**16.91416654**

Efficiency of the IJK forms



The efficiency declines with all three forms but is more consistent in the ikj and kij forms than the ijk form.

Effciency					
	4 cores	8 cores	12 cores	16 cores	20 cores
ijk	**0.9856690831**	**0.7847202606**	**0.8395926976**	**0.7464187131**	**0.5727898484**
ikj	**0.9066104031**	**0.8813872133**	**0.8824983057**	**0.8386125052**	**0.84860546**
kij	**0.981439902**	**0.8763026619**	**0.8760776086**	**0.8356769006**	**0.8457083271**

Jeff Anderson
Csci 551
4/19/2020

This project presented me with more difficulty than I imagined. The MPI libraries can be difficult to understand and the documentation is very dense. It also seemed to be “picky” to me, and took me days to figure out why parallelization wasn’t working the way I expected to. It was due to out-of-order function calls to the MPI library. Once I figured that out, I was able to complete the project.

I used MPI_Broadcast in a previous assignment, but that wouldn’t totally cut it here. As instructed from a piazza post, the most crucial function for this project was using MPI_Scatterv, which breaks up an array evenly and allows for extra work to be done on a process you delegate to. Because of the fact that you might not always have clean numbers to use, you could have a remainder and have to designate which process should take on that extra work. MPI_Gatherv is the corresponding call to collect the data back from each individual process to get a global sum.

Sources:
https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_mm.c
https://stackoverflow.com/questions/27575912/mpi-debugging-segmentation-fault
https://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node70.html
https://gist.github.com/ehamberg/1263868/cae1d85dee821d45fb0cd58747aaf33370f3f1ed
An Introduction to Parallel Programming, Peter Pacheco
Code provided by Dr. Challinger in notes and linked to Pacheco’s code