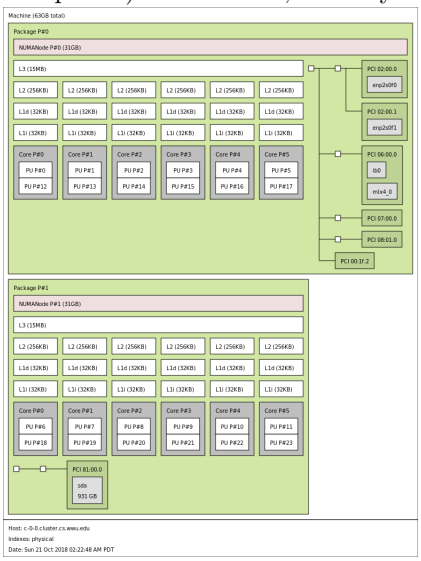
CSCI 515, Autumn 2018
Assignment # 1

Brian Barragan-Cruz

October 21, 2018

An architectural view of the c-0-0 node on the WWU cluster. This system has two sockets, twelve cores, four cache levels (2 of which are 32 KB and private, 1 of which is 256 KB and private, and the last is 15 MB and public). In addition, memory is 31 GB



The c-0-0 node has 12 CPUs. Hence, we use 12 threads when running affinity example. We can set the environment variable OMPPLACES and OMPPROCBIND to core or socket and close or spread, respectively. Therefore, there are four execution policies for this program. The execution policies OMPPLACES=core and OMPPROCBIND=close, OMPPLACES=core and OMPPROCBIND=spread, OMPPLACES=socket and OMPPROCBIND=spread, and OMPPLACES=socket and OMPPROCBIND=close correspond to policies 1, 2, 3, and 4. When OMPPLACES=core and OMPPROCBIND=close, the i^{th} thread will be assigned to the i^{th} core. This setup is the best choice since it makes use of the data locality between process i and process $i+1$ when referencing arrays *in* and *out*. When OMPPLACES=core and OMPPROCBIND=spread, the even process i will go to core i, while the odd process $i^{'}$ will go to core $i'+7$, which doesn't take advantage of the data locality of the workloads. Now when OMPLACES=sockets and OMPPROCBIND=spread, the even processes will be assigned to socket 0 and the odd processes will be assigned to socket 1. This runs the risk of having two processes competing for the same core even if there is an open core somewhere else. When the processes are close, this seems to increase the probability. In contrast when OMPLACES=sockets and OMPPROCBIND=spread, the probability seems to decrease

Since the best execution policy is OMPPLACES=core and OMPPROCBIND=close, we use this execution policy for Experiment 2: Parallelizing matmult.c using OpenMP. We simply parallelize the main loop of the matrix multiplication algorithm by distributing the rows of the first randomly generated matrix. On the third figure, you can see that the purple-blueish line has a somewhat constant increase in speedup as threads increase. However, at around 10 threads, the speedup starts to plateau. This is due to Amdahl's Law. On the other hand, there is an orange line that has much better performance. This line corresponds to the

modified matmult.c where we take the transpose of the matrix b and switch k and j within the third loop. This provides much better data locality and performance since the C programming language stores arrays in row-major order. Therefore, there is not as much thrashing, which was very much the case in the original matmult.c program. Furthermore, switching to dynamic scheduling seems to slow down the performance, since the program no longer optimally uses data locality.