Optimal Data Parallelization of the BLAST Bioinformatics Application on HPC clusters

Poster topic area: Unleashing the power of data

Lead author of poster: Trinity Cheng

Lead author e-mail: trinicheng@gmail.com

Lead author center/office/division: CDRH/OSEL/DIDSR

Mentor’s name: Mike Mikailov, PhD

Mentor’s email address: Mike.Mikailov@fda.hhs.gov

Poster synopsis: The Basic Local Alignment Search Tool (BLAST) is a widely used bioinformatics application for gene sequence search and alignment. By dividing the input query and database, BLAST can be run in parallel to greatly reduce runtime. In this study, we investigate the optimal partitioning of the query and database for the fastest runtime given p number of computing nodes.

Poster abstract:

The Basic Local Alignment Search Tool (BLAST) is an application widely used in the bioinformatics field for exploring biological sequences, such as RNA or amino acid sequences. To match a list of unknown sequences with known sequences archived in existing databases, the user provides two input files -- the database and the query, both usually having millions of lines of text, which makes BLAST computationally challenging. Researchers have applied high-performance computing techniques to develop parallel versions of BLAST such as mpiBLAST, ScalaBLAST, and DCBLAST. In 2017, Mikailov et al. developed the “dual segmentation” method to decrease the runtime of BLAST by dividing the database and query into *m* and *n* subsets, respectively, which were then processed in parallel using *m*\**n* nodes to gain significant speedup. Although the experiment data demonstrated the potential of the dual segmentation method, the optimal partitioning method for any given input data has yet to be found.

In this work, we consider the scenario of running BLAST with a database of *d* sequences and a query of *q* sequences on an HPC system with *p=m*\**n* nodes. By using the dual segmentation method, each of the *p* nodes runs BLAST with a partial database of *d/m* sequences and a partial query of *q/n* sequences. Our objective is to find the optimal integer *n*, which also determines *m=p/n*, such that the runtime is minimized when *p* nodes are available.

Our approach, based on the concept of strong scalability, estimates the overall runtime by measuring the runtime of a sub-job running on only one node, which performs 1/(*m*\**n*) of the total job. Among all 13 combinations of *m*\**n*=4096, we tested *n*=128, 256,… 4,096 to find the optimal value using the binary search approach. Then the optimal solution (*m,n*) was used to run the full dataset in parallel on *p* nodes.

The experiments, which used a 2.4 GB database of 523,449 sequences and a 15 GB query of 73,102,0235 sequences, were conducted on the CDRH HPC Betsy cluster. With 4,096 available nodes, the optimal solution was found to be (*m,n*)=(2,2048), in which the sub-job took 46.65 seconds to complete on a single node. In comparison, the previous solution of (*m,n*)=(27,152) took 171.97 seconds, 3.67x longer than ours. The (*m,n*)=(2,2048) solution was unexpected because it splits the database into two parts only and instead minimizes the query size.

To verify our findings, both solutions were tested by running in parallel on the CDRH HPC Betsy cluster. The (*m,n*)=(2,2048) configuration used 4,096 nodes and had an average array job task time of 39.45 seconds, with the fastest job running in 15.89 seconds and the slowest taking 245.77 seconds. The standard deviation was 17.81 seconds. In contrast, the (*m,n*)=(27,152) configuration used 4,104 computing nodes, but had an average array job task time of 179.96 seconds, with the fastest job running in 60.23 seconds and the slowest taking up to 470.25 seconds. The standard deviation was 73.062 seconds. By comparing the mean runtimes, the (*m,n*)=(2,2048) configuration runs 4.56x faster than the (*m,n*)=(27,152) configuration.

In this study, the optimal partitioning for a database of *d=*523,449 sequences and a query of *q=*73,102,023 sequences was found to be (*m,n*)=(2,2048). We learned that BLAST users may benefit from dividing the query rather than dividing the database. Although the effectiveness of this partitioning may vary based on different database and query sizes, we propose the method of estimating the overall runtime by measuring the runtime of a sub-job. For future work, we will test different database and query sizes to develop general guidelines for partitioning input data for the BLAST application.