HPC BLAST: Best Practices

The Application Acceleration Center of Excellence (AACE) and
Intel Parallel Computing Center at the
Joint Institute for Computational Sciences
University of Tennessee

May 26, 2016

1 Introduction

This document is intended to outline a series of best practices, or rules of thumb, for using HPC-BLAST to perform large scale sequence alignment searches on current and emerging architectures. The collected best practices are derived from scaling studies and other data gathering experiments and should provide a reasonable level of performance for many cases a user might encounter. However, please note that the given practices are based on results from execution of the **Beacon** cluster located at the Joint Institute for Computational Sciences and may not reflect the best parameters or guidelines for other compute resources.

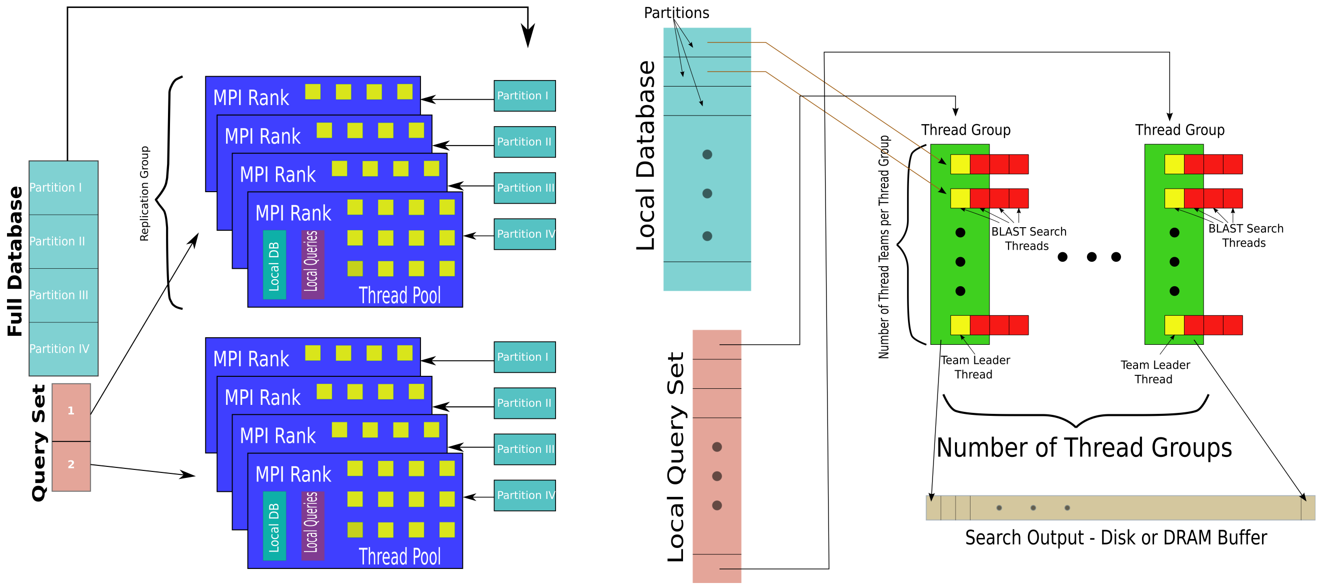
2 Beacon Architecture

The **Beacon** cluster is a Xeon Phi accelerated compute resource. The cluster features 46 compute nodes each with 2 8-core Intel Xeon E5-2670 processors and 4 Intel Xeon Phi 5110p coprocessors; an additional 2 compute nodes are available but are configured with different hardware. Compute nodes also have 256 GB of memory and are connected with FDR InfiniBand. Please see `www.nics.tennessee.edu/beacon` for further information regarding the **Beacon** cluster.

3 Best Practices

Best practices are given below for the two separate architectures found on the **Beacon** cluster, the Intel Xeon processor and the Intel Xeon Phi coprocessor, since the performance of HPC-BLAST is characterized differently on these two platforms. As explained in the HPC-BLAST User Manual, there are a number of different parameters that can be adjusted to specify the manner in which a parallel HPC-BLAST job can execute: the number of MPI ranks, number of replication groups, number of ranks that constitute a replication group, number of thread groups, number of database partitions (team leaders) per thread group, and number of search threads used inside the BLAST engine.

The different components of HPC-BLAST are summarized here so that the terminology used below is clear. HPC-BLAST employs a hierarchical approach for assigning search tasks. At the highest level, MPI ranks are utilized to distribute the pieces of a database partition among compute resources. A replication group is a collection of MPI ranks that aggregately posses the entire database. Each MPI rank in a replication group has the same query input since the database is distributed. Multiple replication groups can be used so that the query input can be distributed between different replication groups. It is also possible to use replication groups with only a single rank; i.e. the database is not distributed, though it may still be partitioned. Within each MPI rank, HPC-BLAST employs a novel threading model to distribute tasks in a similar manner as the MPI layer. A thread group plays the role of the replication group. Here, a thread group is a collection of threads that share the same query subset but have different pieces of the local database. Threads in a thread group are called team leaders since they can instantiate searcher



(a) The MPI layer.

(b) The thread layer.

Figure 1: The HPC-BLAST hierarchy.

threads. The searcher threads are the threads that the NCBI BLAST engine provides. The MPI and thread levels can be seen visually in Figure 1.

It is natural to ask how many MPI ranks should be launched and how many threads should each rank use. First, let us address the number of threads to use on a single target: the two Xeon processors on a compute node of **Beacon** or a single Xeon Phi 5110p. Scaling studies have demonstrated that it is best to use all available threads with HPC-BLAST. On the Xeons, this means we use all 32 logical cores (including the 16 hyper-threads). For the Xeon Phi, we launch with all 240 hardware threads. For now, suppose we are using only a single MPI rank with HPC-BLAST per target; we will address multiple ranks per target below. Given this constraint there are some rules of thumb regarding the allocation of the threads to the different categories: thread groups, team leaders, and search threads. One caveat of using database partitioning inside the BLAST process (the team leader threads) is that it does not guarantee 100% NCBI compliant results. Scaling studies have also demonstrated that thread groups (query input partitioning) provide better speedup relative to team leaders and search threads; thus, we will not include team leaders in the best practices. The scaling studies also demonstrate that the benefit of thread groups alone has a limit based on total thread count. For the Xeon processors, the best performance for cases of thread counts greater than 16 was seen when 2 search threads were used in the decomposition. For example, if 16 total threads were desired than the run should be configured with 16 thread groups each with 1 database partition and 1 searcher thread. However, if 28 total threads were desired then we should use 14 thread groups with 1 database partition and 2 searcher threads. For the Xeon Phi, the threshold for including searcher threads in the decomposition is generally at 60 threads. As on the Xeon, HPC-BLAST on the Xeon Phi typically performs well with 2 searcher threads when above the threshold, but some inputs might see best performance with 3 or 4 searcher threads.

The above guidelines are applicable to situations in which there is enough input to justify the number of thread groups; i.e. enough work to parallelize. At a minimum, there should be enough residues (or letters) so that each thread group has one batch's worth of queries. Here, a batch refers to the concatenated query that NCBI-BLAST uses; i.e. multiple queries from the input are combined and searched through the database as one entity. For *blastp* searches, the batch is 10,000 residues. For *blastn*, the NCBI-BLAST engine uses an adaptive batch size, but 100,000 residues represents a good minimum number to assign each thread group. If there is not enough work to give all thread groups at least one batch worth of queries, then it is probably best to consider a database partition at the MPI level and distribute the database subsets to the MPI ranks inside a single replication group.

Scaling experiments have also demonstrated that when doing *blastp* searches on the Xeon Phi with HPC-BLAST there is a potential for increased performance by using multiple MPI ranks as opposed to a single rank with 240 threads, even when there is enough work to justify 240 thread groups. A good rule of thumb is to use 4, 6, or 8 MPI ranks per Xeon Phi. Further, the best performance was observed when 2 replication groups were used. As an example, consider the following scenario. We decide to use 6 MPI ranks and 2 replication groups. Thus, we would partition the database into 3 subsets (3 ranks per replication group). Each HPC-BLAST MPI rank would then have 40 threads (from 240/6), at most. When using multiple replication groups on the Xeon Phi, keep in mind that the memory is limited to 8 GB. It is recommended to use no more than 4GB of memory for the database summed across all ranks on the device. In the previous example, the database should be no larger than 2 GB since 2 replication groups are employed. Results may vary depending on query and database composition, but runtimes of *blastn* searches on the Xeon Phi and *blastp* and *blastn* searches on the Xeon processors did not benefit significantly by using multiple MPI ranks per target.

The number of MPI ranks that should be launched during a parallel search is influenced by both the size of the query input and the size of the database. The size of the database will contribute to the number of ranks in a replication group. Good performance has been observed with large databases, such as **nr**, when the size of the formatted sequence files (the *.psq* or *.nsq* files) are around 2 GB. The number of MPI ranks per replication group is then determined by how many subsets are generated in the database partition by the *makeblastdb* tool (see the HPC-BLAST User Manual for instruction). As for the number of replication groups to employ, the above advice should be followed in that each replication group should have enough queries to give each thread group at least one full batch worth of residues; though, this is a lower bound and a more reasonable middle ground might be to aim for ten batches worth per replication group.

3.1 Summary

- Use all available logical cores on the desired architecture before adding more compute resources: 32 threads for the 2 Xeon E5-2670's and 240 threads for Intel Xeon Phi 5110p's.
- Avoid using team leaders unless there is a good reason for using them.
- Generally, it is better to use more thread groups (query partitioning) than searcher threads up to a threshold. Use 2 searcher threads per thread group thread above a count of 16 total threads on the Xeon processor. Use 2-4 searcher threads on the Xeon Phi coprocessor when the total thread count exceeds 60.
- If the number of thread groups is such that each thread would have fewer than one full concatenated query to search, run instead with more MPI ranks each with a reduced thread group count; i.e. use the MPI level to distribute the database work.
- For *blastp* searches on the Xeon Phi, consider using 4, 6, or 8 MPI ranks per device with 2 replication groups.
- On the Xeon Phi 5110p coprocessors, limit the total amount of database across all MPI ranks to 4 GB.
- The database should be partitioned such that sequence files are all around 2 GB in size; this also determines how many MPI ranks will form a replication group.
- For large runs with multiple replication groups, use a number of replication groups so that each group has around 10 batches worth of queries, if resources allow.