CEZARY KOSKO 337256

Communication-Avoiding Parallel Sparse-Dense Matrix Multiplication

The MPI Programming Assignment

Data format

Fields of A are stored within 1D arrays containing (row, column, value) tuples (i.e. COO).
B and C are stored within regular 2D arrays (the first index being the row index, so that replication/printing is easier).

Non-basic MPI usage (and basically the implementation)

2 custom communicators are introduced, one for shift operations, the other for replications.
A custom type for COO tuples is introduced as well.

At first chunks of A are scattered among all processes (first their sizes using MPI_Scatter, after that the chunks themselves, using MPI_Scatterv - the chunks may not be even sized). (The array containing A is sorted row-wise if innerABC is used, column-wise in the other case, so that the right chunks get to the right processes at a low cost (and without dirty code))

Basic info such as the number of A's rows are broadcasted using MPI_Bcast.

Then A's chunks are replicated (inside said replication communicator), first by sending their size via MPI_Allgather and then the chunks via MPI_Allgatherv.

Same with B if innerABC is used.

Then, if innerABC is used, we execute the initial shifts[1]. Note these only need to be executed once, as their purpose is to evenly distribute the output's rows between the replication group and that is preserved after counting A*B.

Then the actual multiplication + shifts are executed.

If innerABC, C's chunks are combined from pieces by calling MPI_Allreduce row by row, also taking care of broadcasting the new B. While that's one unnecessary Broadcast per computation (we don't have to Broadcast C after the last iteration), code's much cleaner that way.

As for the result, count is realized using MPI_Reduce (when innerABC, only processes whose rank is 0 in their replication groups weigh in, others give in 0s).

---

[1]Shifts are realized using 2 MPI_Isend and 2 MPI_Recv operations (sending the size/contents of our chunk and receiving new ones after that).

Printing, similarly (non-0s in innerABC do not contribute) is realized by first MPI_Gather-ing the chunks' sizes and MPI_Gatherv-ing them row by row. (single rows are not bigger than B, hence fit in the memory).

Optimization

In my opinion, the biggest optimization is avoiding initial shifting for every iteration of innerABC. Others are a result of using collective MPI operations on custom communicators, hence minimizing the overhead needed for broadcasts and such.

So as not to complicate the code, in innerABC I've used MPI_Allreduce instead of a Reduce + Bcast combination, where Bcast might be disabled for the last iteration, which would minimize the communication costs an additional little bit.