CMOR 421 Homework 3: MPI

Hubert King

April 9th, 2024

1 Simplified SUMMA Algorithm

For our simplified SUMMA, we use the following approach. Algorithm 1 provides a more detailed outline.

- Scatter partitions of A and B to the processes, stored as A^{local} and B^{local}.
- Set $C^{local} \leftarrow 0$.
- Set $blocksize \leftarrow n \div \sqrt{s}$.
- Allocate buffers A^{buffer} and B^{buffer}.
- Initialize row and column communicators.
- In the main computational loop, all processes perform p iterations of the following:
  - Broadcast A^{local} across its row.
  - Broadcast B^{local} across its column.
  - Receive row broadcast into A^{buffer}.
  - Receive column broadcast into B^{buffer}.
  - Accumulate $C^{local} \leftarrow A^{buffer} \times B^{buffer}$.
- Gather blocks of C from all processes onto root process.

2 Cannon's Algorithm

For Cannon's algorithm, we use the following approach, keeping the same *blocksize* from SUMMA and procedure for the initial scatter of the partitions of A, B, and C.

- For each row i of A, perform a left-rotating shift of the partitions by i position.
- For each row j of B, perform a upper-rotating shift of the partitions by j position.
- In the main computational loop, all processes perform p iterations of the following:
  - Accumulate $C^{local} \leftarrow A^{local} \times B^{local}$
  - Perform left-circular shift of A on all rows by 1 position.
  - Perform upward-circular shift of B on all columns by 1 position.
- Gather blocks of C from all processes onto the root process.

Algorithm 1 SUMMA Algorithm Pseudocode

```
Input: $A$, $B$, $rank$, $size$
Output: $C$
$blocksize \leftarrow n/\sqrt{p}$
if $rank = 0$ then
    for $k = p - 1$ to $0$ do
        for $i = 1$ to $blocksize$ do
            for $j = 1$ to $blocksize$ do
                $A_{ij}^{local} \leftarrow A_{ij}$
                $B_{ij}^{local} \leftarrow B_{ij}$
                $C_{ij}^{local} \leftarrow 0$
            end for
        end for
        if $k > 0$ then
            Send $A^{local}, B^{local}, C^{local}$ to process $k$.
        end if
    end for
end if
if $rank > 0$ then
    Receive $A^{local}, B^{local}, C^{local}$ from $rank = 0$.
end if
for $k = 0$ to $\sqrt{p} - 1$ do
    for $i = 1$ to $\sqrt{p}$ do
        Process holding block $A(i,k)$ broadcasts to its row.
    end for
    for $j = 1$ to $\sqrt{p}$ do
        Process holding block $B(k,j)$ broadcasts to its column.
    end for
    Receive block $A(i,k)$ into $A^{buffer}$.
    Receive block $B(k,j)$ into $B^{buffer}$.
    for $i = 1$ to $blocksize$ do
        for $j = 1$ to $blocksize$ do
            $C_{ij}^{local} \leftarrow C_{ij}^{local} + A_{ij}^{buffer} \cdot B_{ij}^{buffer}$
        end for
    end for
end for
if $rank > 0$ then
    Send $C^{local}$ to $rank = 0$.
end if
if $rank = 0$ then
    $C \leftarrow C^{local}$
    for $k = size$ to $0$ do
        Receive $C^{local}$ from $rank = k$
        $C \leftarrow C^{local}$
    end for
end if
return $C$
```

3 Miscellaneous Details

To generate a random matrices for testing, we utilize the random number generation engine provided by the C++ standard library. We perform a correctness check by comparing product C from the serial and parallel algorithms element-wise, with a tolerance of 1e-9. Timing results are measured in milliseconds.

4 Build and Run Instructions

Access NOTS via a login node and load the necessary modules:

```
module load GCCcore/13.2.0
module load OpenMPI
```

Verify that the module is loaded correctly and the correct version of GCC is being used:

```
gcc --version
mpic++ --version
```

Next, compile the drivers with the following command:

```
mpic++ -o summa -Iinclude summa.cpp src/functions.cpp
mpic++ -o cannon -Iinclude cannon.cpp src/functions.cpp
```

After successful compilation, the programs can be tested by running the following command in the login node, where dimension is replaced by the desired matrix size.

```
mpirun -n <processors> cannon <dimension>
mpirun -n <processors> summa <dimension>
```

To run, we use the following script, named job.slurm, which requests resources and runs the program on NOTS:

```
#!/bin/bash
#SBATCH --job-name=CMOR-421-521
#SBATCH --partition=scavenge
#SBATCH --reservation=CMOR421
#SBATCH --ntasks=<requested-processors>
#SBATCH --mem-per-cpu=1G
#SBATCH --time=00:30:00
echo "My job ran on:"
echo $SLURM_NODELIST
srun -n <processors> summa <dimension>
srun -n <processors> cannon <dimension>
```

Submit the job with the following command:

```
sbatch job.slurm
```

After job completion, view the output with the following command:

```
cat slurm-<job-number>.out
```

Sample output:

```
My job ran on:
bc8u27n1
Matrix size n = 1024
Serial elapsed time = 8132.66
Elapsed time = 2.37754
```

```
Serial product and Cannon's product are equal to machine precision.
Matrix size n = 1024
Serial elapsed time = 8123.66
Elapsed time = 2.37487
Serial product and SUMMA product are equal to machine precision.
```