Exercise for Course

Parallel High-Performance Computing

Dr. S. Lang

Return: 15. January 2015 at the beginning of the exercise or earlier

---

Task 21 MPI: Communication Times (5 points)

We want to measure the times, that are required for the MPI-send routines `MPI_Send`, `MPI_Ssend`, `MPI_Bsend` and `MPI_Rsend` (synchronous resp. asynchronous send, ...), until

1. the calling node has gained control over the program after finishing the related send instruction,
2. the receiver has gotten the message completely.

Subtask (a)

For this task write a program, that chooses one of these routines per command line parameter for sending a message to another process. The size of the message may be as well passed as parameter. The sender measures the time duration until the send command is finished, this means it has again control over the program and the receiver measures the duration, that it needs to completely receive the message. To gain reliable values, it should be possible, to execute this procedure several times in sequence and then to average the timings. The count of repetitions should be adapted to the duration of the message passing. Thus fast transmissions are repeated more often. Of course the whole process has not be fully automatized.

Measure for each send type the times with several message sizes. The message size shall therefore vary from few Bytes up to multiple Mbytes. Measure also the hop time, this means the transmision time of a single byte.

Subtask (b)

The routine `MPI_Send` can work in blocking or non-blocking manner. Develop a test to detect, at which message size is started to transmit the message synchronously. Implement the test and find the particular n value for the pool installation.

Free Willi Task

Repeat the measurements of Subtask (a), work only local on one machine (how that works, see exercise 20). How compares the time now?

Hints

- Since we also want to measure the overhead of the communication, we measure real time under usage of `MPI_Wtime()`. In the pool has `MPI_Wtime()` the resolution $1\mu s$ (query by `MPI_Wtick()`, which should be sufficent for our purposes.
- Synchronise the processes before starting the time measurement via a `MPI_Barrier()`.
- Profiling and time measurement for MPI programs is not trivial. Here are some links onto helpful old and new documents:
  - An example like the one we want to implement in Subtask (a): http://cpansearch.perl.org/src/JOSH/Parallel-MPI-0.03/contrib/perf/mpi_timing.c
  - Pitfalls in the time measurement: http://www.mcs.anl.gov/research/projects/mpi/mpptest/hownot.html

– MPI-profiling suite:
  http://www.mcs.anl.gov/research/projects/mpi/mpptest/
– A particular extensive paper:
  http://perlplexity.org/Publications/hoefler-collmea.pdf

Task 22 MPI: Matrix Multiplication (10 points)

In the tasks 4 and 9 we have multiplied two (quadratic) matrices $A, B \in \mathbb{R}^{n \times n}$ ($C = A \cdot B$) and measured the flop rates with tiling resp. OpenMP. Now, we want to repeat this using MPI and further investigate the scalability of the problem regarding the process count.

Task

Write a program, that executes matrix multiplication with MPI in parallel. On the homepage a sequential variant including tiling is provided (compile it with `g++ -fopenmp mm_vanilla.c`), that you may but of course do not have to use. You can use the Cannon algorithm, that is proposed in the lecture, but also may implement another meaningful strategy (explain shortly your idea and concept). The matrix shall initially be distributed onto the processors in such a way, that not each processor stores the complete starting matrices A and B (since then a large part of the communication overhead vanishes). Initialize the matrices such that they are filled with a dense pattern, e.g. $a_{ij} = b_{ij} = i + j,\ i, j = 0 \ldots n - 1$. Measure the computing times in seconds and the FLOP rates for $m = 1, 2, 4, 6, 8, 12, 16, 20, 25, 32$ involved processes and matrix sizes (row count of the matrix) $n = 256, 512, 1024, 2048$. For measurement with $m = 1$ you can use the provided sequential program. Take for time measurement as previous `MPI_Wtime()` and synchronise at the beginning of the time measurement via `MPI_Barrier()`. Plot diagrams with computing time over the problem sizes as well as the speedup against the processor count P for the different problem sizes. Discuss shortly the results. If still available you could also compare with your old results of tasks 4 and 9.

Optional Task

Use on the individual computing nodes additionally tiling for increasing the cache usage and compare the measurements with the results of the untilled variant.

Hints

You could think about using the Cannon algorithm to solve the task. But to use this algorithm is not possible without constraints: First for Cannon has to apply $N \mod \sqrt{P} = 0$, which renders some of the above required combinations impossible: The count of computing nodes has to be the square of a integer number. Furthermore the algorithm of Cannon is especially efficient, if one multiplies large matrices, that can only be stored in a distributed manner. In our task the matrices are maximally $2048 \cdot 2048 \cdot 8$ Bytes = 32 MBytes large. Such they fit in the local memory of each node without any problems. If you still want to realize the Cannon algorithm, adapt P and N to the requirements and use larger matrix sizes, that are preferably stored in a distributed way.