HW1 - High Performance Computing

Andrew Szymczak
February, 2015

GitHub Repository

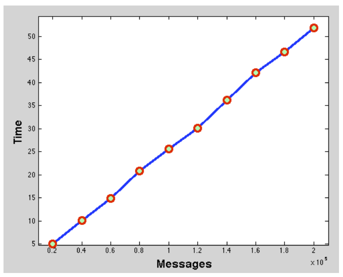
- `https://github.com/Turing-Apparatus/HPC-HW1.git`

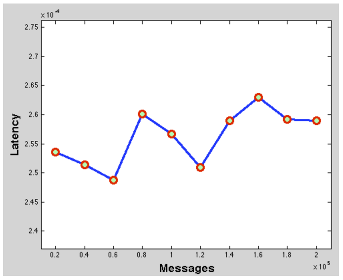
MPI ring communication. Write a distributed memory program that sends an integer in a ring starting from process 0 to 1 to 2 (and so on). The last process sends the message back to process 0

- I allow two command line parameters N and M. N specifies the number of loops and M specifies the number of integers sent in each message. Each message is an array of M integers.

  ```
  msg = (int *) malloc(sizeof(int) * M);
  ```

  The first element is the only one altered as `msg` is passed around; `msg[0] += rank`. If you don't input an argument or you input an illegal value, N, M default to 1. On my machine `ints` are 4 bytes, so to send 2MB messages I input $M = 524288$. If the number of processors is less than 1, the program aborts.
- After the last message is received at processor zero, I verify that `msg[0] = NP(P-1)/2` where P is the number of processors.
- I only tested my program locally. The latency for a message seems to be about .026 ms. 20 processors were used to make the following plots.



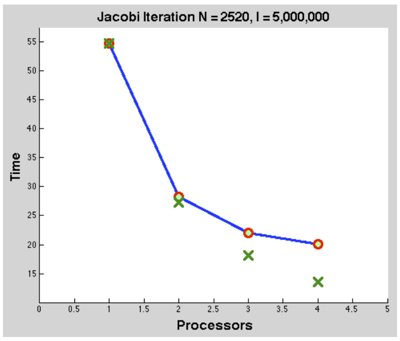


- With $P = 20$, $N = 10000$ and $M = 524288$, a total of 200000 messages were sent in 124.71 seconds. Since the limiting factor is bandwidth rather than latency, I can estimate a bandwidth of about 3.13 GB/s.

Distributed memory parallel Jacobi smoother. Use MPI to write a parallel version of the Jacobi smoother from Homework 0

- The first and last element of $\boldsymbol{u}^{(i)}$ are the shared indices. First we pass the previous values $u_{-2}^{(i)} \to u_0^{(i+1)}$ for all $i < p-1$. Then we perform the Jacobi iteration. Then we pass the new values $u_1^{(i)} \to u_{-1}^{(i-1)}$ for all $i > 0$.
- I use an `MPI_Allreduce` to calculate the residual on every processor. Since this is an expensive process, I only do it once every 100 iterations.
- Since my machine only utilizes four processors, my first scaling test was for $p = 1, 2, 3, 4$. Let I denote the number of iterations, and t the total time in seconds.

N	I	p	$\lvert Au - f\rvert$	t
2520	5000000	1	0.932	54.62
2520	5000000	2	0.932	28.26
2520	5000000	3	0.932	22.08
2520	5000000	4	0.932	20.12



Note how the residual is independent of p, as expected. Each processor handles 2 messages and N/p of the Jacobi work in each iteration. Assuming full connectivity (so that all messages are done in parallel), we can expect $\mathcal{O}(N/p)$ time (as marked by the `X`'s). The slowdown is due to the suboptimal parallelisation and overhead of message passing.

- My second scaling test only establishes the high overhead of message passing. Increasing $p > 4$ on my local machine effectively sequentializes crosstalk. Since there are $2p-2$ messages passed in every iteration, we can expect the computation time to grow $\sim p$.

N	I	p	$\lvert Au - f\rvert$	t
1200	100000	20	22.24	10.40
1200	100000	40	22.24	21.83
1200	100000	60	22.24	33.17
1200	100000	80	22.24	48.06
1200	100000	100	22.24	63.77

- Gauss-Seidel would be more dificult to program because of the implicit dependence in each iteration. We could try to do something analagous to the first bullet point: First we pass the old values $u_1^{(i)} \to u_{-1}^{(i-1)}$ for all $i < p-1$. Then we perform the Jacobi iteration. Then we pass the new values $u_{-2}^{(i)} \to u_0^{(i+1)}$ for all $i < p-1$. The problem is that you can't do this in parallel. Before $u^{(i)}$ can preform Jacobi, it must wait to recieve the new value $u_{-2}^{(i-1)}$, which isn't sent until all lower rank processes are complete. The process is essentially sequential, and even worse off due to message passing.