



F. Wermelinger
Office: Pierce Hall 211

Lab 4

Parallel reductions

Issued: March 6, 2023
Due: March 24, 2023 11:59pm

In this lab we are looking at the reduction operator¹. Reductions are important operations in parallel applications and can be performed in different ways that require us to study in a bit more depth.

Submission instructions: please see the lab submission section in the syllabus.^a

^a<https://harvard-iacs.github.io/2023-CS205/pages/syllabus.html#lab-submission>

Task 1: Parallel Reductions

A reduction is a class of parallel algorithms that transform $\mathcal{O}(n)$ inputs to $\mathcal{O}(1)$ output by means of an associative binary operator $\oplus$. The number n could be the number of MPI ranks for example. Some examples of reductions include:

- finding a global minimum or maximum
- summation of values or sum of squares
- product sums such as an inner product (dot-product) which are at the core of matrix-vector and matrix-matrix products
- logical operations
- reductions are the core of programming models such as map reduce² that are used by Apache Hadoop for example
- MPI has of course a reduction operation as well

Note that because of the associativity of $\oplus$, there are various ways a reduction can be performed. Consider the reduction of a vector with 8 elements

$$v_0 \oplus v_1 \oplus v_2 \oplus v_3 \oplus v_4 \oplus v_5 \oplus v_6 \oplus v_7,$$

¹https://en.wikipedia.org/wiki/Reduction_operator

²<https://en.wikipedia.org/wiki/MapReduce>

a serial reduction would look like this when we highlight the associativity

$$(v_0 \oplus (v_1 \oplus (v_2 \oplus (v_3 \oplus (v_4 \oplus (v_5 \oplus (v_6 \oplus v_7))))))))).$$

It involves 7 steps and does not exploit parallelism. Tree based reductions on the other hand are parallel and will result in better performance. The MPI reduction operations, e. g., `MPI_Reduce` do of course implement optimized tree based algorithms. Examples for such log-step reductions can be done pairwise or interleaved and take this form of associativity

$$(((v_0 \oplus v_1) \oplus (v_2 \oplus v_3)) \oplus ((v_4 \oplus v_5) \oplus (v_6 \oplus v_7))).$$

Figure 1 depicts these approaches graphically. A serial reduction in MPI would require n iterations of sending messages, while the tree based approaches only involve $\log(n)$ iterations of communication.

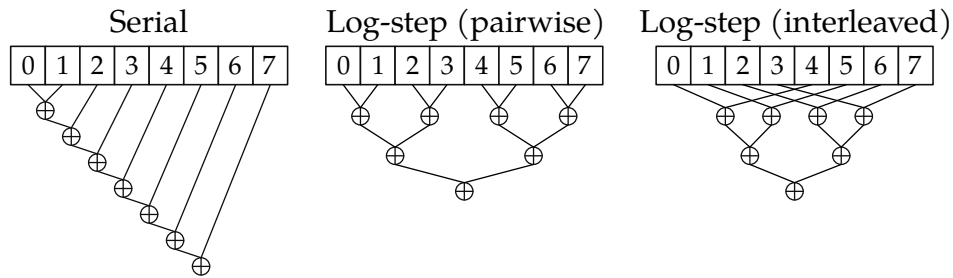


FIGURE 1: Different type of reductions possible due to the associativity of the $\oplus$ operator. The deeper the figure in the vertical direction, the less parallelism exploited.

In this lab we want to compute the following sum in parallel

$$\sum_{k=1}^n k = 1 + 2 + \dots + (n-1) + n = \frac{n(n+1)}{2}, \quad (1)$$

where n is an arbitrary number. We use m MPI ranks where each rank computes a partial sum which we then reduce among the ranks.

- Complete the TODO:A: labels in the file [code/t1/reduction.cpp](#) such that MPI is correctly initialized and finalized. Make sure you have a recent compiler and MPI library loaded before you compile your code

```
$ module load gcc/12.1.0-fasrc01 openmpi/4.1.3-fasrc01
```

You can then build your code by typing

```
$ make
```

- We assume that $n > m$, where m is the number of MPI ranks we are going to parallelize this problem with. To simplify the reduction a bit we assume that m is a *power of two*. In this task each rank i should compute a partial sum σ_i which consists of $\lfloor \frac{n}{m} \rfloor$ terms. Note that n may not necessarily divide evenly. Make sure the remainder is distributed among the ranks such that you minimize load imbalance. Follow the TODO:B: labels in the provided code to complete this part. In the next task we will reduce σ_i among the m ranks.

- c) In this task we implement a log-step algorithm to reduce equation 1 to the final result. For example, assume you have $m = 8$ ranks with indices $i = 0, 1, 2, \dots, 7$, then we aim to implement the following communication pattern for the reduction

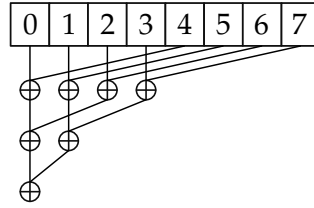


FIGURE 2: Communication pattern used for this task. The numbers in the boxes correspond to the different MPI ranks.

In the first iteration of communications the following partial sums are computed on ranks $i = 0, 1, 2, 3$

$$\sigma_0 = \sigma_0 + \sigma_4, \quad \sigma_1 = \sigma_1 + \sigma_5, \quad \sigma_2 = \sigma_2 + \sigma_6, \quad \sigma_3 = \sigma_3 + \sigma_7.$$

In the second iteration ranks 0 and 1 compute $\sigma_0 = \sigma_0 + \sigma_2$ and $\sigma_1 = \sigma_1 + \sigma_3$, respectively, and finally the root rank computes the result $\sigma_0 = \sigma_0 + \sigma_1 = \sum_{k=1}^n k$. Note that all of these steps involve communication with a specific rank to obtain the missing σ_i for the partial summation on the target rank. You may only assume that m is any power of two. Follow the TODO:C: labels and implement this reduction³ You can run your code on one compute node with $m = 32$ ranks using the provided `reduction.sh` job submission script. If the 32 core nodes are busy, you may adjust the requested resources in the script to target 16 MPI ranks which may run on a 28 core node instead (recall that we assume m is a power of two). Submit it with

```
$ sbatch reduction.sh
```

Please include the output file `*.out` generated by your job in your final lab submission as well and make sure it contains the expected output. If you implemented the reduction correctly and assume that m is a power of two, your result should match the exact relation in equation 1 which you can use to check the correctness of your implementation.

³Note that in practice you would use the `MPI_Reduce` routine to accomplish this. You typically would not attempt to implement a common algorithm like this yourself, unless you need something special. We do it here for exercise.