Butterfly allreduce

1 Butterfly allreduce

This exercise is about parallelizing the allreduce operation using a butterfly scheme.

Assume an `array` of integer values of size `n` and a complex and time consuming operator $\oplus$ which is **associative and commutative**. The allreduce operation is such that

$$array[i] = array[0] \oplus \cdots \oplus array[n-1] \quad \forall i$$

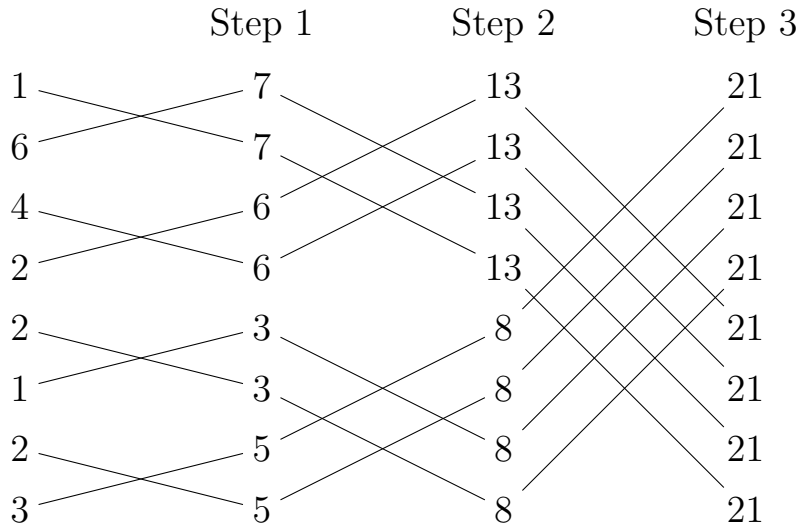
that is, after this operation is performed, all the elements of `array` contain the reduction of all the initial elements of `array` using the *oplus* operator.

In order to make the understanding and the debugging easier, we will assume, in this document and in the code, that the size of the array is a power of 2, i.e., $n = 2^l$ and that the $\oplus$ operator is just a (time consuming) sum of integers.

The allreduce operation is achieved in l steps where, at each step p, coefficients are reduced pairwise with a stride of $s = 2^p$. Here is a sequential code that achieves this

```
p = 0;
while(p<l){
  s = pow(2,p);
  for(i=0; i<n; i+=2*s){
    for(j=0; j<s; j++){
      int r = operator(array[i+j],array[i+j+s]);
      array[i+j]   = r;
      array[i+j+s] = r;
    }
  }
  p+=1;
}
```

For the case $l = 3$ the functioning of the method is illustrated below



2 Package content

In the `butterfly` directory you will find the following files:

- `main.c`: This file contains the main program. This reads from command line the parameter l such that the size of the array is $n = 2^l$. It generates the array, and calls the routine `butterly_seq` which achieves the sequential butterfly allreduce with the code presented above. Then it calls the `butterfly_par` which computes the butterfly allreduce in parallel (this has to be implemented as explained below). Finally it checks if the result of the parallel allreduce is correct. **Only this file must be modified**.
- `aux.c, aux.h`: these two files contain auxiliary routines and declarations and **must not be modified**.

The code can be compiled with the `make` command: just type `make` inside the `butterfly` directory; this will generate a `main` program that can be run like this:

```
$ ./main l
```

If $l \leq 5$ the original list as well as the result of the `butterfly_seq` and `butterfly_par` are printed upon execution of the main program.

3 Assignment

- ⌨ The objective of this exercise is to parallelize the butterfly allreduce code presented above. **This as to be done using the OpenMP**

`task` **directive**. At the beginning the `butterfly_par` is a copy of the `butterfly_seq` routine; it has to be modified to achieve the parallelization.

- ✎ Report the execution times for the implemented parallel version with 1, 2 and 4 threads and for different array sizes. Analyze and comment on your results: is the achieved speedup reasonable or not? Report your answer in the form of comments at the bottom of the `main.c` file.

Advice

- When developing your code, always work on arrays of small sizes (8, 16 or 32 elements, for example) but when you want to evaluate the performance use large size arrays (256, 512 or 1024 elements, for example);