2-norm computation without overflow

1 2-norm computation

The 2-norm of a vector x of size n is simply defined as

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$$

Unfortuntely computing and summing the squares may overflow the maximum value imposed by the chosen arithmetic even if the final value of the norm does not. Therefore in practice more sophisticated methods have to be used.

One way of handling this problem consists in scaling all the coefficients of the x vector by the greatest (in absolute value) among them. However, this scaling should not be done at once prior to computing the norm because this would be to costly rather, it can be done in the course of the norm computation. For this we will use two scalars; assuming we are at iteration `i` of the computation:

- `scale`: contains the maximum absolute value of all the coefficients from 1 to i;
- `ssq`: contains the sum of the squares of the coefficients from 1 to i normalized by the square of `square`.

If, at some iteration a coefficient which has greater absolute value than `scale` is met, `scale` is set to its value and `ssq` is updated accordingly. This leads to the following code:

```
scale = 0.0;
ssq   = 1.0;

for(i=0; i<n; i++){
  if(x[i]!=0.0){
    absxi = fabs(x[i]);
    if(scale < absxi){
      ssq = 1.0 + ssq*pow(scale/absxi,2);
      scale = absxi;
    } else {
      ssq = ssq + pow(absxi/scale,2);
    }
  }
}

norm = scale*sqrt(ssq);
```

The objective of this exercise is to parallelize this operation. For this we will rely on the following fact. Assume we have two vectors x_1 and x_2 and we execute the above algorithm on each one of them finding, respectively, `scale1`, `ssq1`, `scale2` and `ssq2`. The norm of the vector

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

can be easily found with the following code

```
if(scale1>scale2) {
  scale = scale1
  ssq   = ssq1 + ssq2*pow(scale2/scale1,2);
} else {
  scale = scale2
  ssq   = ssq2 + ssq1*pow(scale1/scale2,2);
}
norm = scale*sqrt(ssq);
```

2 Package content

In the `norm2` directory you will find the following files:

- `main.c`: this file contains the main program that creates a vector x of size n and then computes its 2-norm using first the sequential routine `dnorm2_seq` and then the two parallel routine `dnorm2_par` that has to be implemented as described below.

The code can be compiled with the `make` command: just type `make` inside the `norm2` directory; this will generate a `main` program that can be run like this:

```
$ ./main n
```

where n is the size of the vector whose norm has to be computed.

3 Assignment

- At the beginning, the `dnorm2_par` routine is a copy of `dnorm2_seq`. Modify this routine in order to parallelize it in such a way that each thread computes the partial norm of subset of the coefficients of x and then all these partial norms are combined through the reduction operation described above. The reduction operation has to be computed by hand has described above without relying on the OpenMP `reduction` clause. This can be achieved through the use of a shared array.

- ✎ Analyze and compare the performance of the parallel code using one, two and four threads. Report the execution times in the `responses.txt` file and comment on the observed results: did you observe any speedup (reduction of the execution time) using 2 and 4 threads instead of 1? How can you interpret these results?

4 Advice

- As an example, an integer array of size n can be allocated in the following way

```
int *array;
array = (int*)malloc(n*sizeof(int));
```