

Dense matrix multiplication

07/12/2016

1 Dense matrix multiplication

Assume three matrices A, B, C partitioned into $n \times n$ blocks of size nb as such (for A):

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{pmatrix}.$$

The operation $C = C + A \times B$ can be computed with these three nested loops

```
for(i=0; i<n; i++)
  for(j=0; j<n; j++)
    for(k=0; k<n; k++)
    {
      block_mult(a[i][k], b[k][j], c[i][j], nb);
    }
```

where `a[i][k]` contains A_{ik} and `block_mult(a[i][k], b[k][j], c[i][j])` computes $C_{ij} = C_{ij} + A_{ik} \times B_{kj}$.

This exercise is about parallelizing the above operation using the OpenMP `task` construct.

2 Package content

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

- `main.c`: this file contains the main program which first calls the `init_data` routine which initializes with random values the three matrices `a`, `b` and `c` and makes a copy `d` of `c`. These matrices are coded as 2D arrays of type `block`; each entry `a[i][j]` is simply a 2D array of size `nb * nb`. The main program then calls the `sequential_product` routine executes the three nested loops above computing the product `c = c+a*b` and the

`parallel_product` routine which contains the parallelized code to be implemented and computes the product $\mathbf{d} = \mathbf{d} + \mathbf{a} * \mathbf{b}$. Finally, the main program checks that `c` and `d` are the same. **Only this file has to be modified for this exercise.**



- `aux.c`, `aux.h`: these two files contain auxiliary routines and **must not be modified**.

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

```
$ ./main n
```

where `n` is the number of blocks in rows and columns of the matrices.

3 Assignment

-  At the beginning, the `parallel_product` is a copy of `sequential_product`. Modify this routine in order to parallelize it. The parallelization **must** be based on the OpenMP `task` construct. The `depend` clause can be used to define dependencies between tasks, where needed. Make sure that the difference between `c` and `d` after the product (i.e., the value printed on screen by the `compare_matrices` routine) is smaller than 10^{-14} .
-  Report the execution times for the implemented parallel version with 1, 2 and 4 threads and compare them with the sequential routine. What speedup could you achieve? analyze and comment on your results. Report your answer in the `responses.txt` file.

Advice

- When using the OpenMP `task` construct, always think about data scoping to make sure input data has the correct value upon execution of the task and returned results do not go out of scope when the task is finished.
- Note that the `depend` clause allows you to specify dependencies due to access to single coefficients of an array. For example:

```
#pragma omp task depend(inout:x[i])
    x[i] += 1;
```