Programming Exercise

NumScale

17/06/2016

The goal of this exercise is to showcase your programming skills as well as your parallel programming knowledge. You may use any resource at your disposal to complete the exercise, but please make a reasonable attempt to provide your own work and not someone else's solution.

1 Partial sum

Given a matrix M of size $n \times m$, consider a program that computes the sum of each column so that the result vector V of size m is defined like so:

$$\forall j \in [\![0, m[\![, V(j) = \sum_{i=0}^{n-1} M(i, j) \tag{1}$$

1.1 Sequential and scalar implementation

Write a sequential and scalar implementation in C or C++ for single-precision floating point data. The data structures to use and the memory layout of the object are left at your discretion; in particular you can use a row-major or column-major layout.

Try to make this implementation as efficient as possible without using explicit parallelization mechanisms. The target for running the program should be an AMD64 processor of the Nehalem micro-architecture or similar. Please outline why you think your implementation makes efficient use of the processor.

We are interested in the computation part of the program, there is no need to provide good mechanisms to input the matrix and output the resulting vector.

1.2 Parallel implementation

Please provide one parallel implementation using the parallel technology of your choosing, or a combination of several technologies, including computing on the CPU, the GPU or both. You may use for example tools like SSE, pthread, Boost.Thread, OpenMP, OpenCL, CUDA, MPI or HPX.

For the purpose of parallelization, we will assume that + on floating-point operands is associative.

There is no need to provide a perfect solution, just make a reasonable effort at showing you understand at least one parallel technology, and be aware of its advantages and limitations.

How does your implementation behave when n is small? When m is small? When both n and m are small? When both n and m are big?

1.3 Generalization

The program could operate on any arithmetic type with any associative binary operation and a neutral value. In our case we computed the sum, but the same algorithm could also have computed the product for example.

Use C++ templates to generalize your scalar implementation to any type and any folding operation. If the technology allows to do it reasonably easily, also generalize your parallel implementation.

Ensure that your usage of C++ templates does not slow down the computation at runtime.