Projecto Integrado (PI)

A study on scalable numerical algorithms

Tasks

Develop the following tasks for each proposed algorithm (list below):

1. Design and code a sequential version; analyse qualitative and quantitatively the performance of this version; identify those parts of the algorithm/code that are better suited for optimization and/or parallel computing. *(AMN/SCD/PCP)*
2. Optimize the sequential code through the use of specific techniques and numerical libraries, with an evaluation of the overall impact on performance. *(AMN)*
3. Design and implement a shared-memory parallel version of the algorithm with OpenMP; test and evaluate performance on one node in the SeARCH cluster. *(PAC/PCP)*
4. Design and implement a multi-threaded hybrid code that combines shared-memory (on multi-cores and dual CPU device on a single node) with distributed-memory on CUDA-enabled GPU devices; test and evaluate performance on one node in the SeARCH cluster. *(SCD/AMN)*
5. Design and implement a distributed-memory parallel version of the algorithm with MPI; test and evaluate performance on one node in the SeARCH cluster. *(PCP/PAC/AMN)*

Methodologies, reports and presentations

1. Teams: 2 teams, 2 students each; one project per team.
2. Deliverables: reports and presentations in English; when and what:
   a. At the end of each phase (see the chronogram below), each team delivers a report (4 or 5 pages) and makes an oral presentation to all students in the UCE (everyone is requested to participate with questions);
   b. At the end of the semester, each team delivers a final report (8 pages, plus annexes), which replaces the previous partial reports; each student gives a final oral presentation and discussion (same target audience and requirements as before).

Chronogram

- Phase 1 (tasks 1 and 2): oral presentation 17th May, report 21st May
- Phase 2 (tasks 3 and 4): oral presentation 7th June, report 12th June
- Phase 3 (task 5): oral presentation and a draft of final report 28th June, final report 9th July.

List of projects

1. Choleski factorization

Many scientific problems require the solution of systems of linear equations **Ax** = **b**. An important class of methods for solving the system are the so-called direct methods. Two major tools in this context are **(i)** the LU factorization for general matrices (almost always used with partial pivoting to make it numerically stable) and **(ii)** the Choleski factorization for symmetric positive definite matrices.

Given **A**, the Choleski factorization computes the lower triangular matrix **L** such that $\mathbf{A} = \mathbf{LL}^T$. To solve **Ax** = **b** we solve two triangular systems, **Ly** = **b** for **y**, and finally LTx = **y** for **x**.

The goal of this project is the development, test and analysis of sequential and parallel implementations of the $\mathbf{LL}^T$ factorization and the solution of the two triangular systems.

The following aspects should be addressed:
- stability of the Choleski decomposition;
- locality of data: re-use cached data as much as possible (blocked algorithm).
- potential parallelism for shared memory, for distributed memory and also for co-processing on an accelerator device (a GPU).

2. Numerical solution of Poisson's equation with finite differences

Finite differences are frequently used for the numerical solution of partial differential equations. One of the best known is Poisson's equation that occurs in the context of steady-state heat distribution. Here, the problem will be considered over a thin square plate and the five-point formula will be used for the discretization mesh.

From the numerical point of view, the result is a very large sparse linear system and iterative methods are usually more adequate. We will consider Jacobi and Gauss-Seidel methods. Even if there is no need to store a matrix, the matrix formulation of the method is important to discuss the convergence properties of the iterative methods.

The following aspects should be addressed:
- estimation of the truncation errors;
- convergence properties of the Jacobi and Gauss-Seidel methods in this problem;
- locality of data: re-use cached data as much as possible;
- potential parallelism, for shared memory, for distributed memory and also for co-processing on an accelerator device (a GPU).

Bibliography

[1] Matrix Computations, G. Golub and C. Van Loan, 3rd ed. The Johns Hopkins Univ. Press, 1996.

[2] Parallel Programming in C with MPI and OpenMP, M. J. Quinn, McGraw-Hill, 2003.