Internship (5/6 months) at the POEMS laboratory

M2 level in applied mathematics, scientific computing, or mechanics

Lazy $\mathscr{H}$-matrix factorisation: shared memory parallelism based on a dataflow approach

Summary

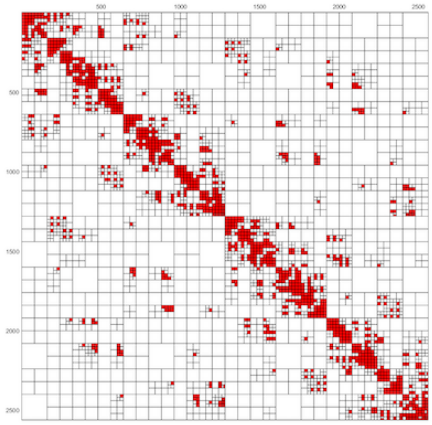
In this project we will focus on the efficient parallelisation of hierarchical matrix factorisation based on a dataflow approach. Starting from a *lazy* and serial algorithm which attempts to minimise the number of allocations, we will explore how to parallelise it based on a dataflow approach. The main idea is to describe how each basic operation in the serial algorithm access the underlying data, and create a directed acyclic graph (DAG) to represent the computation. A simple scheduler can then be run on the DAG to parallelise the algorithm while ensuring sequential consistency.

This project will be carried out in the context of two existing libraries: HMatrices.jl and DataFlowTasks.jl. The internship will take place at the POEMS laboratory, and will be supervised by Luiz M. Faria (CR INRIA) and Stephanie Chaillat (DR CNRS).

Motivation and overview

Solving linear systems of the form $A\,x = b$ is a cornerstone of not only numerical linear algebra, but of applied mathematics in general as most methods rely, on way or another, on finding solutions to linear systems. When A is an $n \times n$ dense matrix, a direct factorisation such as LU is know to cost $\mathcal{O}(n^3)$ operations, making such a procedure impractical for large problems (say $n \sim 50000$). There are, however, several applications whereas despite the denseness of the underlying matrix, its particular structure is such that lower complexity algorithms are possibly. In this project, we will focus on the matrices A for which a hierarchical low-rank approximation is indeed feasible.

The main idea behind hierarchical matrices is that, after a certain permutation of rows and columns, the underlying matrix possesses large blocks which are of low rank. If we let $R \in \mathbb{R}^{p\times q}$ be a rank k matrix over the reals, then it is possible to find a *data sparse* representation of it such as $R = A\,B^t$, where $A \in \mathbb{R}^{p\times r}$ and $B \in \mathbb{R}^{q\times r}$, and B^t is the transpose of B. Assuming that $r \ll \min(p, q)$, storing the matrix R in the aforementioned outer product format saves storage. Furthermore, the outer product format also allows for a lower complexity product. The entire matrix is then represented as a hierarchical data structure, with certain (small) blocks being simply dense matrices, while other (large) blocks begin low-rank matrices. Provided enough of such low rank blocks exist in the underlying matrix, it can then be shown that a large set of linear algebra operations can be performed in dense matrices of size $n \times n$ with a complexity (and storage) that scales quasi-linearly with n [1].



Among the possibly algebraic operations that one can perform with a hierarchical matrix, the direct LU factorisation is a particularly important one in cases where: (i) one is interested in solving $A\,x = b$ for several right hand sides b, and/or the matrix A is not well-conditioned. The complex data-structure of hierarchical matrices, together with the need to preserve the low-rank structure of blocks during the factorisation, makes the parallelisation as well as efficient serial implementation of the hierarchical LU algorithm a challenging problem [2]. Recently, some effort has been paid to improve the serial algorithm by grouping certain hierarchical matrix products (see e.g. [3]), but the parallelisation of such techniques remains unexplored. The principal research direction of this project is to combine a data flow approach to parallelism [2] with the newer algorithms such as those discussed in [3].

Expected outcome

At the end of the internship the student is expected to have fully grasped the main ideas behind hierarchical matrix factorisation, as well as to develop a better understanding of the various challenges associated with shared memory parallelism (e.g. race conditions and how to avoid them). In particular, the student should be comfortable with the classical hierarchical LU factorisation procedure, and understand the main difficulties related to its parallelisation. A working, even if inefficient, parallel version of the hierarchical lu factorisation is also expected. Finally, the student should attempt to answer some of the following questions regarding the proposed technique:

(a) Is the *lazy* approach interesting from a parallelisation point of view?

(b) What are the main bottlenecks in the parallelisation, and how can one improve on them?

(c) Can one theoretically analyse the number of allocations in the classical vs. *lazy* factorization algorithms for a constant rank approximation?

The project will (roughly) follow the outline below:

- First, a literature review will be performed in order for the student to become familiar with the basic equations and algorithms we will study (≈ 1.5 month)
- If the student is not familiar with the Julia programming language, he/she will follow some online tutorials, as well as perform some basic tasks to better grasp the details of the language (≈ 1 moth, in parallel with the literature review)
- After becoming familiar with the basic idea of the algorithms and the programming language, the student will dive into the implementation details of HMatrices.jl and DataFlowTasks.jl. In particular, the student should understand the main design principles and where the modifications are needed in order to parallalelize the algorithm (≈ 1.5 months)
- The second half (3 to 6 months) will be devoted to the novel work. In particular:
    - Implement a parallel version of the *lazy hierarchical matrix* factorization
    - Analyse the algorithm and characterise both the operations count (flops) as well as the number of memory allocations (bytes).
    - Detect the possible bottlenecks, study the scaling properties, and if time permits compare to alternative approaches.

Student profile

This project requires familiarity with numerical linear algebra concepts as well as some understanding of how to analyse the computational complexity of an algorithm (e.g. counting the number of flops) and number of allocations. Some knowledge of boundary integral equation would be helpful to better understand the main application case, but is not necessary. Familiarity with a compiled (e.g. C++) as well as an interpreted (e.g. Python) programming language is necessary. All programming tasks will be performed in Julia, so basic familiarity with the language is a plus, but not required. Most importantly, the student should be interested and curious about programming, scientific computing, algorithms, and their efficient implementation.

References:

[1] Bebendorf, M. (2008). *Hierarchical matrices*.
[2] Lizé B. (2014) Résolution directe rapide pour les éléments finis de frontière en électromagnétisme et acoustique: $\mathcal{H}$-Matrices. Parallélisme et applications industrielles.
[3] Dölz J, Harbrecht H, Multerer MD (2019). On the best approximation of the hierarchical matrix product.

Contact: luiz.maltez-faria@inria.fr and stephanie.chaillat@ensta-paris.fr