a multifrontal threaded solver for sparse least-squares problems

The advent of multicore processors represents a disruptive event in the history of computer science as conventional parallel programming paradigms are proven incapable of fully exploiting their potential for concurrent computations. The objective of the qr.mumps project is to develop a novel approach to the parallelization of sparse matrix factorizations aiming at obtaining high efficiency and scalability on new generation multicore architectures. The result of this effort is the qr.mumps software, a package for the efficient computation of the QR factorization of sparse matrices on modern computing environments.

The multifrontal method for QR factorizations

The multifrontal QR method relies on the concept of elimination tree, a graph that captures the dependencies among computational tasks in the factorization process. Every node of the tree is associated with a frontal matrix that contains all the coefficients affected by the elimination of the corresponding pivot. The QR factorization consists in a bottom-up traversal of the tree where, at each node, two operations are performed:

- assembly: a set of rows from the original matrix is assembled together with data produced by the processing of child nodes to form the frontal matrix of the parent
- factorization: the frontal matrix is factorized. This step produces a few rows of the global QR factor, a complement which corresponds to the data that will be later assembled into the parent node and a set of Householder vectors which are used to implicitly represent the global QR factor.

A multifrontal QR method

- once a frontal matrix is assembled, its rows are sorted in order of increasing index of the leftmost nonzero; the zeros in the bottom-right corner can thus be ignored
- the frontal matrix is completely factorized; the contribution blocks are thus smaller and the overall number of flops reduced

Classic multithreading

Two sources of parallelism are available in the multifrontal method:

- tree parallelism: nodes that belong to separate subtrees are independent and can thus be factorized in parallel
- node parallelism: if a frontal matrix is big enough, it can be factorized in parallel by multiple threads

In classic multithreading the two sources are exploited sequentially, which limits the degree of parallelism and introduces strict parallelizing synchronizations

Fine-grained multithreading

In order to handle both tree and node parallelization in the same framework, a block-column partitioning of the fronts is applied and five elementary operations are defined:

- activate: the activation of a front consists in computing its structure (row/column indices, staircase set), and allocating the related memory
- panel: this operation amounts to computing the QR factorization of a block-column
- update: updating a block-column with respect to a panel corresponds to applying to the block-column the Householder reflections resulting from the panel reduction
- assemble: assembles all columns into the parent node (if existing)
- clean: cleaning a front means storing the R and H coefficients aside and deallocating the memory

The multifrontal factorization of a sparse matrix can thus be defined as a sequence of tasks, each task corresponding to the execution of an elementary operation of the type described above. The tasks are assigned in a Directed Acyclic Graph (DAG); the edges of the DAG define the dependencies among tasks and the order in which they have to be executed.

Asynchronous task scheduling

The execution of tasks in the DAG is controlled by a data-flow model: a task is dynamically scheduled for execution as soon as all the tasks on which it depends have finished. The scheduling of tasks can be guided by a set of rules that prioritize the execution of tasks based on, for example, memory locality or task size.

- The performance of the factorization may be severely limited by the scalability of the task scheduling when the elimination tree is too wide and many nodes are allocated at the same time. Two techniques may be used to overcome this problem:
  - logical amalgamation: small nodes are treated sequentially at the moment when their father is activated
  - logical tree pruning: a level in the tree is identified such that all the subtrees below it can be assigned to threads with a good load balance and treated sequentially

Dynamic memory

Multifrontal solvers are traditionally based on the usage of a stack; a large memory area allocated at once and where all the frontal matrices are pushed or popped as they are treated. Although the stack has many favorable properties in sequential execution if the tree is traversed in postorder, in parallel, the frontal matrices cannot be pushed in a LIFO order because the postorder cannot be guaranteed. This considerably complicates the memory management. Moreover, a single big memory area allocation may limit the scalability of the factorization on NUMA systems.

qr.mumps

qr.mumps is a software package for the QR factorization of sparse matrices specifically designed for achieving high performance and scalability on modern multicore architectures. Main features:

- >20000 lines of Fortran 95/2003 code
- OpenMP based multithreading
- fully parallel factorization; backarrow/forward substitution by RIF and QIF2 matrix application
- singlenters detection
- COAM, METIS, SCOTCH or user-provided column permutation for fill-in minimization
- supports real and complex problems in single or double precision
- C interface

References:

- A. Buttari, “Fine-grained sparse QR factorization for multi-threaded architectures,” in poster at the IEEE 2007 conference proceedings