How two-sided matrix transformation algorithms can benefit from task parallelism

Myllykoski, Mirko

Umeå university

This talk focuses on the two-sided matrix transformation algorithms that are used in the solution of dense and non-symmetric eigenvalue problems

\[ Ax_i = \lambda_i x_i, \]

where \( A \in \mathbb{R}^{n \times n}, \ 0 \neq x_i \in \mathbb{C}^n \) and \( \lambda_i \in \mathbb{C} \). The route of acquiring the eigenvalues \( \lambda_i \) and the eigenvectors \( x_i \) of the matrix \( A \) usually includes the following three steps:

**Hessenberg reduction:** The matrix \( A \) is reduced to upper Hessenberg form by an orthogonal similarity transformation \( A = Q_1 H Q_1^T \), where \( H \) is upper Hessenberg and \( Q_1 \) is orthogonal.

**Schur reduction:** The Hessenberg matrix \( H \) is reduced to Schur form by an orthogonal similarity transformation \( H = Q_2 S Q_2^T \), where \( S \) is upper quasi-triangular with 1 × 1 and 2 × 2 blocks on the diagonal and \( Q_2 \) is orthogonal. The eigenvalues can be determined from the diagonal blocks of \( S \).

**Eigenvectors:** Finally, we compute vectors \( y_i \in \mathbb{C}^n \) from \( (S - \lambda_i I) y_i = 0 \) and backtransform to the original basis by \( x_i = Q_1 Q_2 y_i \).

Optionally, a fourth step can be performed to acquire a desired invariant subspace of \( A \):

**Eigenvalue reordering:** The Schur form \( S \) is reordered, such that a selected set of eigenvalues appears in the leading diagonal blocks of an updated Schur form \( \hat{S} \), by an orthogonal similarity transformation \( S = Q_3 \hat{S} Q_3^T \), where \( Q_3 \) is orthogonal.

The Schur reduction and eigenvalue reordering steps apply a series of overlapping localized orthogonal transformations to the matrices \( H \) and \( S \), respectively. A modern algorithm groups a set of localized orthogonal transformations together and initially applies them only within a small diagonal window in order to achieve higher arithmetic intensity. Existing ScaLAPACK-style algorithms can leverage the parallelism arising from several concurrent diagonal windows and can theoretically reach full core utilization during certain stages of the algorithms. However, although these algorithms are very carefully crafted, all necessary assumptions are not always satisfied and some stages will always have lower core utilization.

A new library called *StarNEig* relies on an alternative approach of describing the algorithms as acyclic directed graphs known as task graphs, where the vertices represent the various computational operations (a.k.a tasks) and the edges describe the data dependencies between the tasks. A runtime system (StarPU) is responsible for scheduling the tasks to the various computational resources, such as CPU cores and GPUs, in a sequentially consistent order as dictated by the data dependencies. This leads to more flexible algorithms that are able to reach a much higher core utilization compared with the ScaLAPACK-style algorithms. Central to this improvement is the fact that carefully selected and constructed task graphs enclose many opportunities for increased concurrency which can be automatically detected and exploited by the runtime system. This talk will describe the process of constructing these task graphs and present computational results documenting the increased performance compared to ScaLAPACK-style algorithms.

The work has been done in collaboration with Carl Christian Kjelgaard Mikkelsen, Angelika Schwarz, Lars Karlsson, Bo Kågström, Mahmoud Eljammaly and Björn Adlerborn.