Scalable and Portable LU Factorization with Partial Pivoting on top of Runtime Systems

Alycia Lisito, Mathieu Faverge, David Goudin, Matthieu Kuhn, Pierre Ramet

▶ **To cite this version:**

Alycia Lisito, Mathieu Faverge, David Goudin, Matthieu Kuhn, Pierre Ramet. Scalable and Portable LU Factorization with Partial Pivoting on top of Runtime Systems. journée calculs et données 2024, Nov 2024, Bordeaux, France. hal-04867213

HAL Id: hal-04867213

https://inria.hal.science/hal-04867213v1

Submitted on 6 Jan 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Scalable and Portable LU Factorization with Partial Pivoting on top of Runtime Systems

Alycia Lisito, Mathieu Faverge, David Goudin, Matthieu Kuhn, Pierre Ramet

Eviden, Inria, LaBRI, Université de Bordeaux, Bordeaux INP

Context

We explore solutions to implement efficient **LU factorization with partial pivoting** using the **sequential task flow programming model**. Due to the pivoting strategy, the algorithm generates a **large number of very small tasks**, which usually overload the runtime system and make it inefficient. We propose two solutions to improve the efficiency and reduce the number of tasks. First, we apply well-known **blocking strategies** in the context of task-based algorithms. Secondly, we explore **batching** techniques to reduce the number of tasks submitted to the runtime system. Moreover, in distributed architectures, partial pivoting generates **many reductions on the critical path** throughout the factorization which needs to be carefully handled to reach high performance. Two task-based reduction algorithms are proposed to express these operations and improve the runtime reactivity on the critical path. These proposals have been implemented in the dense linear algebra library **CHAMELEON** on top of the **STARPU** runtime system.

Objective

- Implement an LU factorization for large heterogeneous architectures with performance portability
- Being able to schedule efficiently large and fine grains computations

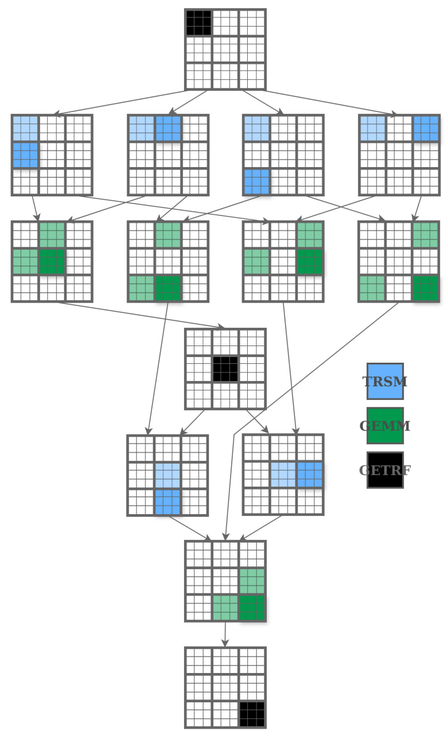
Proposed solution

- Use task-based programming models
  → STARPU runtime system
- Extend existing task-based dense linear algebra library
  → CHAMELEON

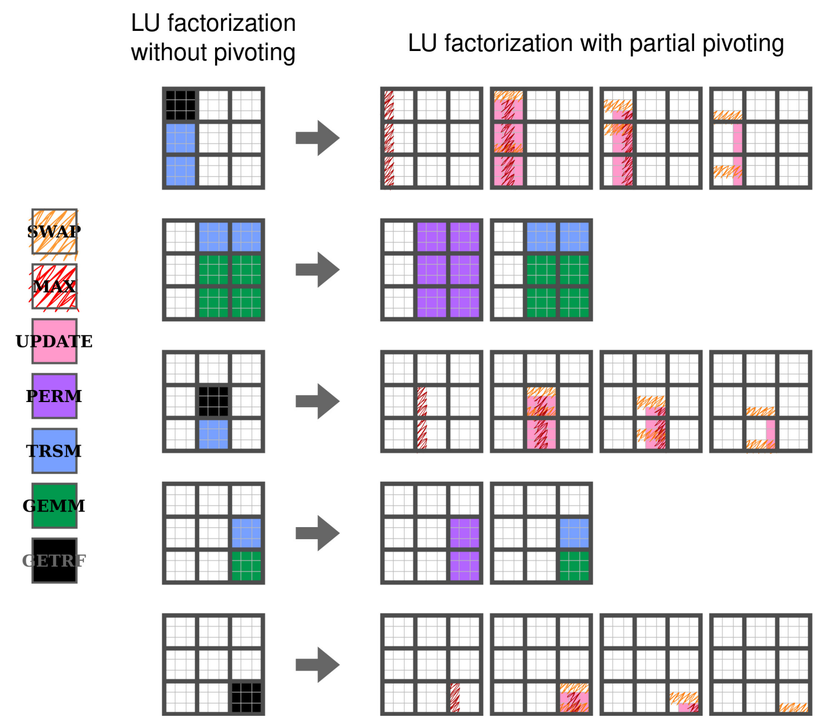
Problems

- Need to look for the maximum value at each column
  - Many small grain reduction tasks
  - Add synchronizations and communications
- Need to apply efficiently the permutation
  - Require to collect data from many tiles of the matrix, on different devices
  - Another set of reduction/broadcast operations that need to be implmented efficiently on heterogeneous distribued context
- Due to the last two points, the set of tasks to manage becomes extremely larger that w/o pivoting

Task-based LU factorization w/o partial pivoting

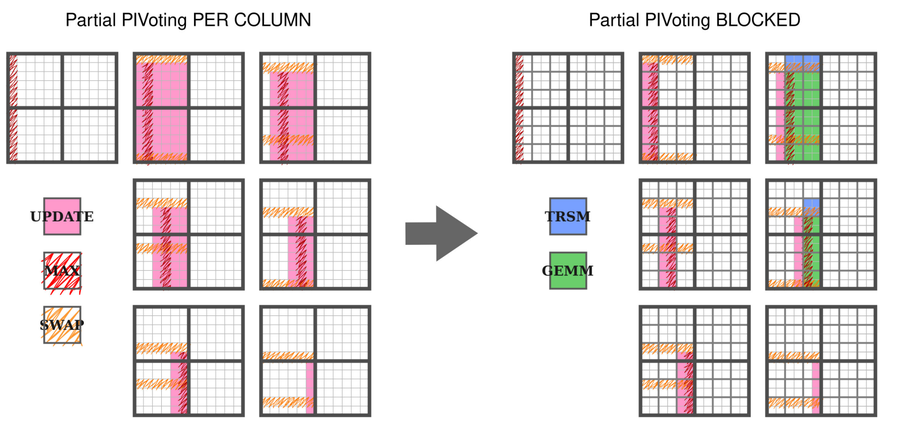


Extension to LU w/ partial pivoting

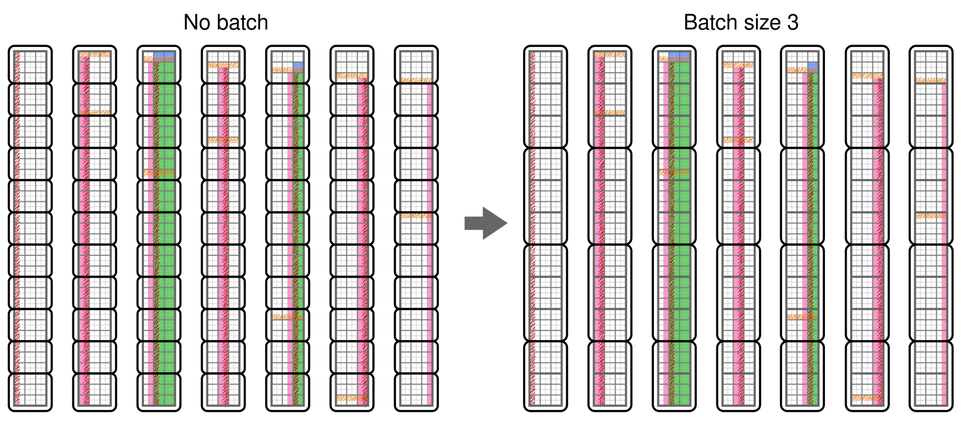


Task number optimizations

Inner blocking



Task batching



Inner blocking

- Increases tasks granularity
- Exploits runtime local reduction

Task batching

- Reduces the runtime overhead
- Reduces the number of tasks
- Increases tasks size

Reduction optimizations

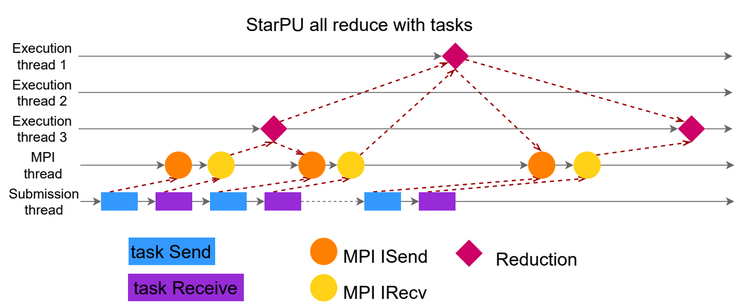
All reduce using the STARPU tasks

- STARPU handles the communications
- Different workers involved in the reduction
- Does not block the workers while waiting for the communications
- Generates many small tasks

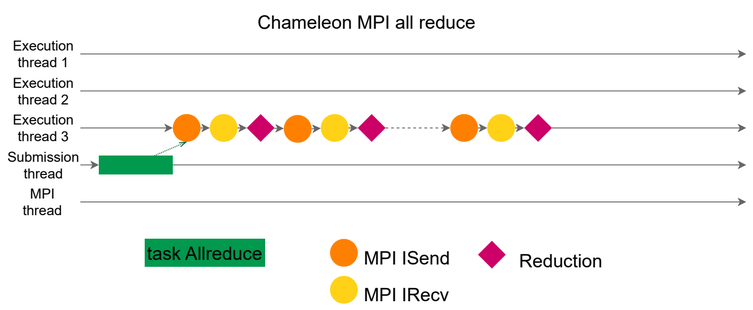
All reduce with MPI communication in one STARPU task

- One worker per reduction
- Blocks the worker while waiting for the communications
- Reduces the runtime overhead
- Reduces the number of tasks

All reduce with STARPU tasks

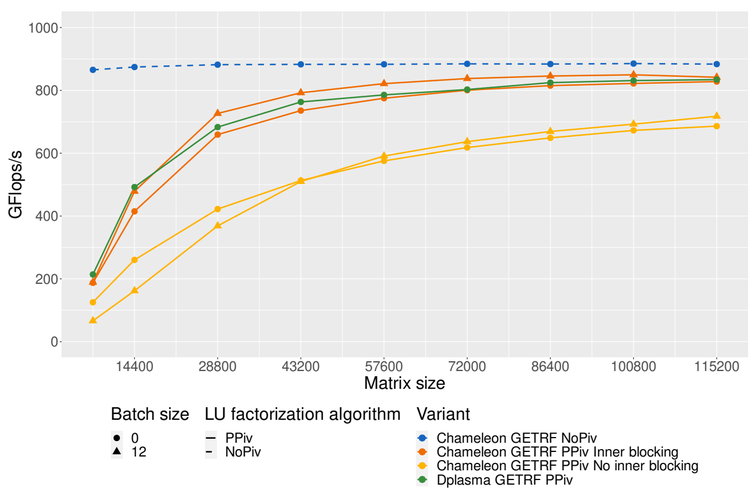


All reduce in one STARPU task

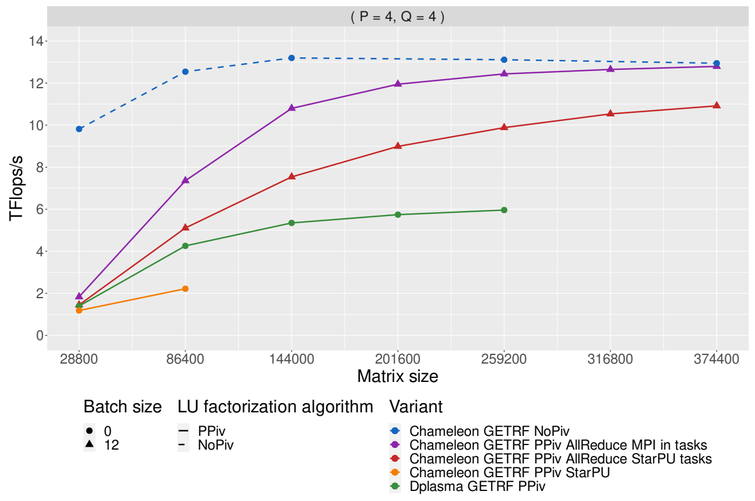


Experiments

Results in shared memory



Results on distributed architecture



Results

- Inner bocking with tasks batching solution reaches 95% of the NoPiv performance in shared memory for matrices larger than 70k
- On distributed architecture, the best solution reaches the NoPiv performance (13TFlops/s)
- Both solutions outperform the OpenSource task-based existing solution from DPLASMA
- The scalability of the best solution reaches 97% on 16 nodes

Future work

- Use of hierchical tasks to filter the permutation
- Choose the batching size dynamically

CEPP team, Eviden Pessac and TOPAL team, Inria Talence

alycia.lisito@inria.fr