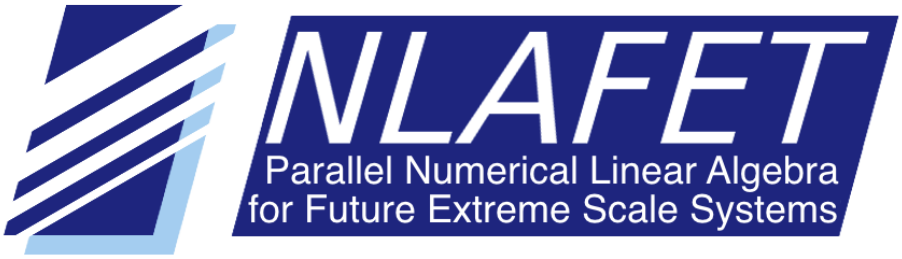


H2020–FETHPC–2014: GA 671633

D2.2

Scalability and Tunability of Factorization Algorithms

April 2018

Document information

Scheduled delivery	2018-04-31
Actual delivery	2018-04-27
Version	1.1
Responsible partner	UMU

Dissemination level

PU — Public

Revision history

Date	Editor	Status	Ver.	Changes
2018-02-06	Bo Kågström	Draft	0.1	Initial structure of the document.
2018-04-06	Lars Karlsson	Draft	1.0	Draft for internal review.
2018-04-25	Lars Karlsson	Draft	1.1	Revised after internal review.

Author(s)

Lars Karlsson (UMU)
Mahmoud Eljammaly (UMU)
Bo Kågström (UMU)

Internal reviewers

Jan Papež (INRIA)
Sébastien Cayrols (STFC)
Florent Lopez (STFC)

Contributors

Mawussi Zounon (UNIMAN)
Mirko Myllykoski (UMU)

Copyright

Acknowledgements

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under the grant agreement number 671633.

Table of Contents

1 Introduction **3**

2 Aims and Scope of the Evaluation **3**
2.1 Tunability 3
2.2 Scalability 3

3 Experiments **4**
3.1 Tunability experiments 4
3.2 Scalability experiments 5
3.3 Machines 5

4 Results for the Cholesky Factorization **6**

5 Results for the Symmetric Indefinite Factorization **7**

6 Results for the LU Factorization **8**

7 Results for the QR Factorization **9**

8 Conclusions **9**

List of Figures

1 Illustration of the tunability experiments. 5
2 Tunability of PLASMA `dpotrf` for $n = 20000$. 6
3 Scalability of PLASMA `dpotrf`. 6
4 Tunability of PLASMA `dsytrf` for $n = 20000$. 7
5 Scalability of PLASMA `dsytrf`. 8
6 Tunability of PLASMA `dgetrf` for $n = 20000$. 8
7 Scalability of PLASMA `dgetrf`. 9
8 Tunability of PLASMA `dgeqrf` for $n = 20000$. 10
9 Scalability of PLASMA `dgeqrf`. 10

1 Introduction

The *Description of Action* (DoA) states for deliverable D2.2:

> "*D2.2: Scalability and Tunability of Factorization Algorithms*
>
> Report on scalability and tunability of the software implementing novel factorization algorithms."

The deliverable is in the context of Task 2.1 ("Linear System Solvers") and is based on work reported in the M18 deliverable D2.1 and extensions thereof. More precisely, we characterize both the *tunability* and the *scalability* of the Cholesky, symmetric indefinite, LU, and QR factorization functions implemented in the PLASMA[1] library. This library is built on top of OpenMP and supports homogeneous shared-memory systems. An evaluation of a StarPU version of PLASMA called PLASTAR, with support for distributed memory, was originally planned but the software was not ready for large-scale testing at the time of writing. This is partly due to limitations and bugs in 3rd party software beyond our control (see also Remark 1 on page 7).

The rest of this report is organized as follows. We clarify in Section 2 the aims and scope of the evaluations. We describe the experiments we performed to gather data in Section 3. The results are grouped per factorization in Sections 4– 7. We end by some concluding remarks in Section 8.

2 Aims and Scope of the Evaluation

2.1 Tunability

With the tunability part of the evaluation we aim to answer the following three questions:

1. How large performance gains can be expected from auto-tuning?
2. How does each parameter separately affect the performance?
3. How good are randomly selected parameter settings?

Note that the aim of the evaluation is to characterize how the performance is affected by the tunable parameters. In particular, we are *not* attempting in this report to evaluate appropriate ways to *automatically tune* the various implementations.

2.2 Scalability

With the scalability part of the evaluation we aim to characterize the weak and strong scalability in terms of core utilization, i.e., a kind of parallel efficiency measured relative to the peak rate of the machine rather than the best sequential implementation. There are several reasons for this choice. First, it avoids possible criticisms for choosing (and tuning) a specific sequential reference implementation over another. Second, it reduces the amount of data to report since it avoids the need to produce a comparison *both* against a reference implementation *and* against the machine. Third, it avoids the issue of fairness when comparing a single-core run with a multi-core run in the presence of dynamic voltage

[1] `https://bitbucket.org/icl/plasma`, changeset `3304b1b119c9cad3aaf35312da55f3c4d92488b9`

and frequency scaling (which, arguably unfairly, benefits the former but not the latter). Fourth, it makes it practical (and meaningful) to run tests too large to fit in the memory of one core or too large to complete within a reasonable amount of time on a single core.

The *core utilization* metric of parallel efficiency[2] is defined as

$$C(n,p) = \frac{F(n)}{pRT(n,p)} \in (0,1], \tag{1}$$

where n is the matrix dimension, p is the core count, R is the peak rate of one core, $F(n)$ is the flop count, and $T(n,p)$ is the parallel execution time. The flop count $F(n)$ is taken to be the flop count of a sequential unblocked algorithm in order to avoid potential inflation due to redundant/extra computations in the parallel cache-blocked algorithms.

A plot of the core utilization versus the core count conveys two important pieces of information at once. The *level* of the graph (i.e., where the utilization lies in the range from 0% to 100%) reveals the *performance* (high utilization translates into high performance). The *slope* of the graph (e.g., increasing, constant, decreasing) reveals the *scalability* (linear speedup translates into a horizontal line).

Our aim is to find the problem size threshold above for which the scaling is linear. Below this threshold the problem size can thus be considered "too small" to use the software/hardware combination efficiently.

3 Experiments

In this section we describe the experiments conducted to gather the data that underlies our tunability and scalability evaluations.

3.1 Tunability experiments

We carried out three different experiments to evaluate the tunability. Figure 1 illustrates (conceptually) how the various experiments explore the parameter space.

The Auto-Tune experiment. To answer the first question listed in Section 2.1, we ran a hill-climbing auto-tuner to find optimized (not optim*al*) parameters. By measuring the performance using both *default* and *optimized* parameters we can measure the performance gains that are achievable through (auto-)tuning. However, these numbers should not be taken out of context since they say more about how appropriate the default configuration happens to be for the particular problem size, machine, and core count.

The Sweep experiment. To answer the second question, we performed one-dimensional *sweeps* over each parameter. The other parameters (if any) were set to (a) their default values and (b) their optimized values.

The Random experiment. To answer the third question, we repeatedly sampled the set of reasonable parameters uniformly at random.

[2]Note that this definition of efficiency is not equivalent to the widely used definition $E = T_s/(pT_p)$.

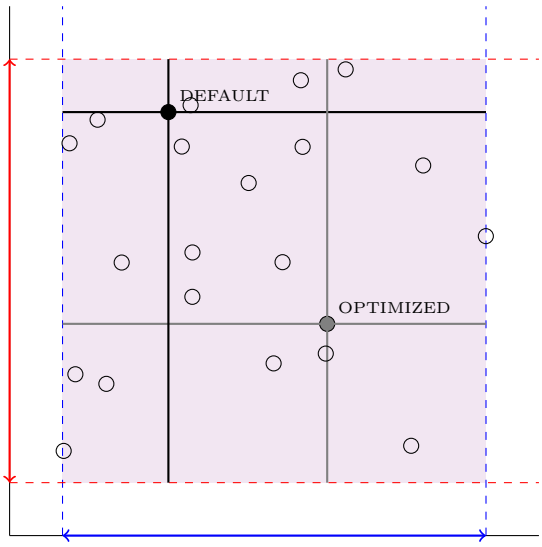


Figure 1: Conceptual illustration of how the tunability experiments probe the parameter space. There are two parameters (one on each axis). Their reasonable ranges (delineated on the axes) define the highlighted set of reasonable parameter values. The black and gray discs show the default and optimized parameters, respectively. The solid vertical and horizontal lines show the probes made in a "Sweep" experiment. The circles show the random samples probed in a "Random" experiment.

3.2 Scalability experiments

We carried out two different experiments to evaluate the scalability.

The Strong Scalability experiment. To evaluate the strong scalability we fixed the problem size to a few different values and varied the core count.

The Weak Scalability experiment. To evaluate the weak scalability we scaled the dimension n to keep the memory-load-per-core ratio n^2/p constant when varying the core count p. We repeated the experiment for a few different memory loads.

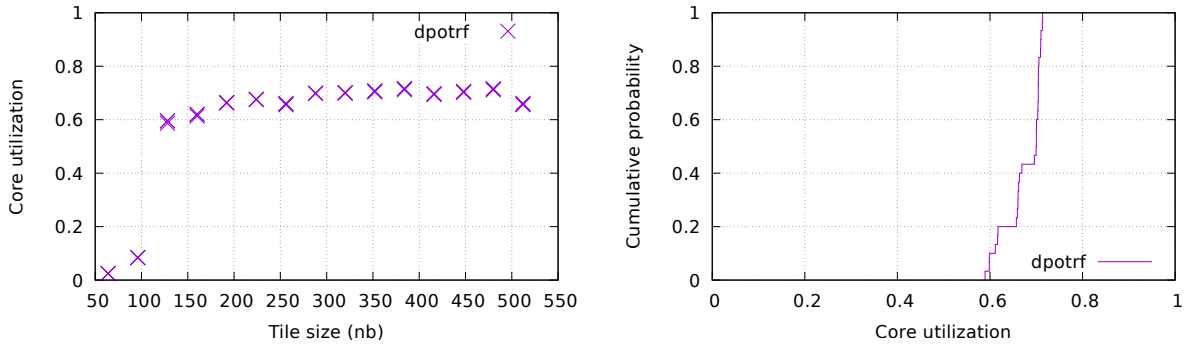
3.3 Machines

All computational experiments were performed on the Kebnekaise system at High Performance Computing Center North (HPC2N), Umeå University. One compute node of Kebnekaise consists of 28 Intel Xeon E5-2690v4 cores, grouped into 2 NUMA islands, each with 14 cores. The cores run with a base frequency of 2.6 GHz and a peak performance of 46.4 GFlops/s (with Turbo Mode frequency). The node has a total memory of 128 GB. Tests for the PLASMAlibrary were performed on one full compute node (28 cores). The node was used exclusively during the experiments. All tests were performed three times and all data points are reported in the results to give an indication of the amount of variance.

4 Results for the Cholesky Factorization

A symmetric and positive definite matrix $A \in \mathbb{R}^{n \times n}$ admits a *Cholesky factorization* $A = LL^T$, where L is lower triangular. A Cholesky factorization can be stably computed by a sequence of elementary row operations (without row interchanges/pivoting). By exploiting symmetry, the flop count is $F(n) = n^3/3 + \mathcal{O}(n^2)$.

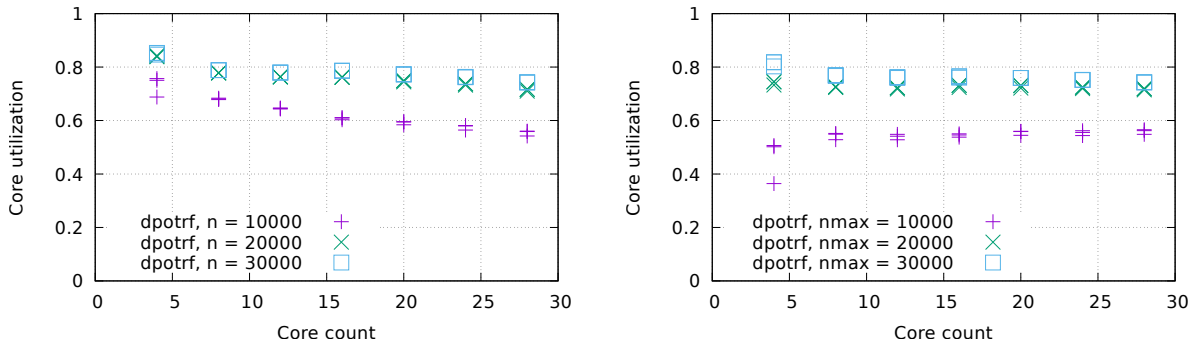
The PLASMA version of `dpotrf` has one tunable parameter, namely, the *tile size* `nb`. The tile size may range from 1 to n, but in practice one would consider range 64–512.



(a) Tile size sweep. (b) Empirical performance distribution.

Figure 2: Tunability of PLASMA `dpotrf` for $n = 20000$.

The results of the tile size sweep are reported in Figure 2a. The results of the Random experiment are summarized in the form of *empirical distributions* (with performance normalized as core utilization). In other words, a point (x, y) on the curve says that the probability that randomly chosen parameters will lead to a core utilization $C(n, p) \leq x$ is $y \in [0, 1]$. Ideally, the curve should be a vertical line at a core utilization of 100%. This means that the performance is optimal no matter how the parameters are set. When the curve is close to vertical it signals that the performance is not very sensitive to the choice of parameters. On the other hand, the performance is sensitive if the curve jumps horizontally near the top of the graph. This implies that it is possible to achieve (relatively) high performance but it is very unlikely to randomly stumble upon the right parameter values. The empirical performance distribution is shown in Figure 2b. The results from the strong and weak scaling experiments are shown in Figures 3a and 3b, respectively.



(a) Strong scalability. (b) Weak scalability.

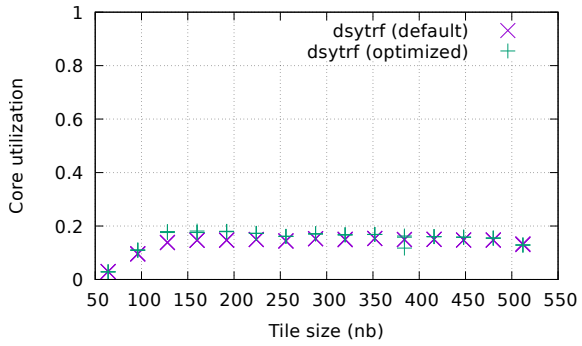
Figure 3: Scalability of PLASMA `dpotrf`.

5 Results for the Symmetric Indefinite Factorization

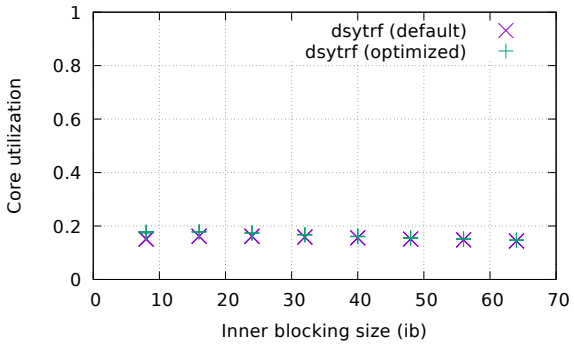
For symmetric but *indefinite* matrices $A \in \mathbb{R}^{n \times n}$, Aasen's method and its derivatives compute symmetric factorizations of the form $PAP^T = LTL^T$, where L is lower triangular, T is banded, and P is a permutation matrix. The flop count is $F(n) = n^3/3 + \mathcal{O}(n^2)$, i.e., the same as for Cholesky.

The PLASMA version of `dsytrf` has two tunable parameters: the *tile size* `nb` and the *inner blocking size* `ib`. The panel factorization (of `nb` columns) is performed as a sequence of smaller panel factorizations of `ib` columns. The inner blocking size could range from 1 to `nb`, but in practice one wold consider a range 8–64.

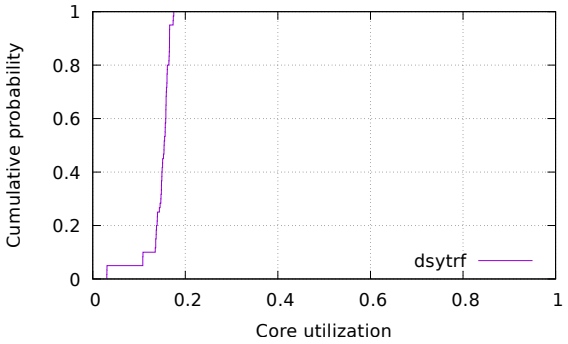
Remark 1. *The* ***`dsytrf`*** *and* ***`dgetrf`*** *functions also have an important parameter called* ***`mtpf`*** *that controls the number of threads to use in parallel panel factorizations. Unfortunately, there is an open bug in the GNU OpenMP library that can trigger deadlocks when parallel panel factorizations are enabled. The other alternative at our disposal is to use the Intel compiler (and associated OpenMP library) but that compiler does not yet support OpenMP task priorities and the code does not compile. Since this parameter can have a large impact on performance, the results presented in this report are overly pessimistic. In other words, higher performance than what is reported here can be expected by setting* ***`mtpf`*** *to a value* > 1.



(a) Tile size sweep.



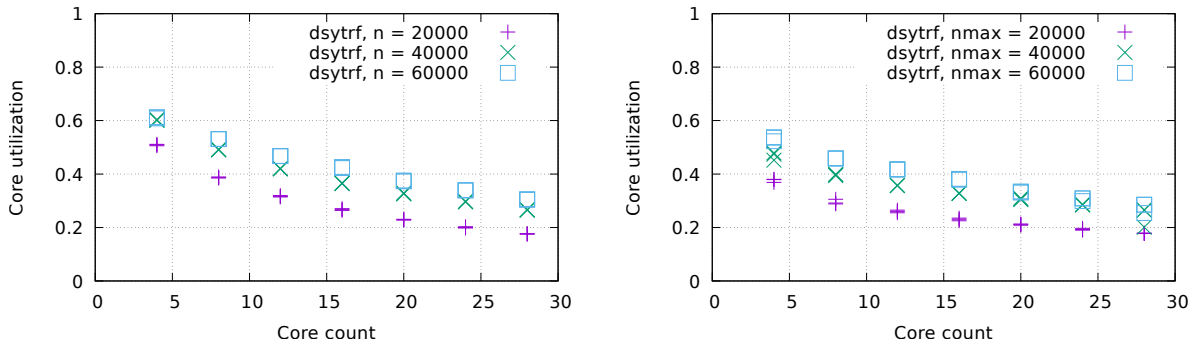
(b) Inner blocking size sweep.



(c) Empirical performance distribution.

Figure 4: Tunability of PLASMA `dsytrf` for $n = 20000$.

The results of the tile size sweep are reported in Figure 4a, and the results of the inner blocking size sweep are reported in Figure 4b. The empirical performance distribution is shown in Figure 4c. The results from the strong and weak scaling experiments are shown in Figures 5a and 5b, respectively.



(a) Strong scalability.

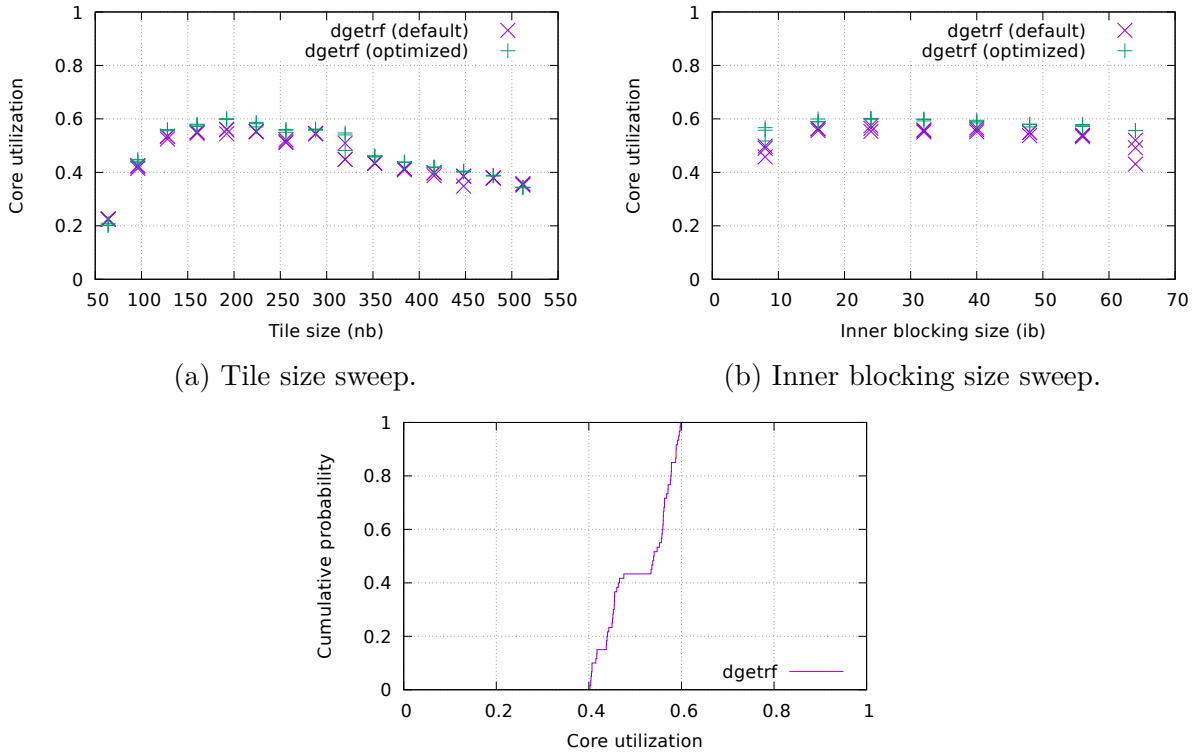
(b) Weak scalability.

Figure 5: Scalability of PLASMA `dsytrf`.

6 Results for the LU Factorization

A square and invertible matrix $A \in \mathbb{R}^{n \times n}$ admits an *LU factorization* with partial pivoting of the form $PA = LU$, where L is lower triangular, U is upper triangular, and P is a permutation matrix. The factorization can be stably computed by elementary row operations with a flop count of $F(n) = 2n^3/3 + \mathcal{O}(n^2)$, i.e., twice that of the symmetric factorizations.

The PLASMA version of `dgetrf` has two tunable parameters: the *tile size* `nb` and the *inner blocking size* `ib`. Like `dsytrf`, each panel factorization is performed as a sequence of smaller panel factorizations of width `ib`.



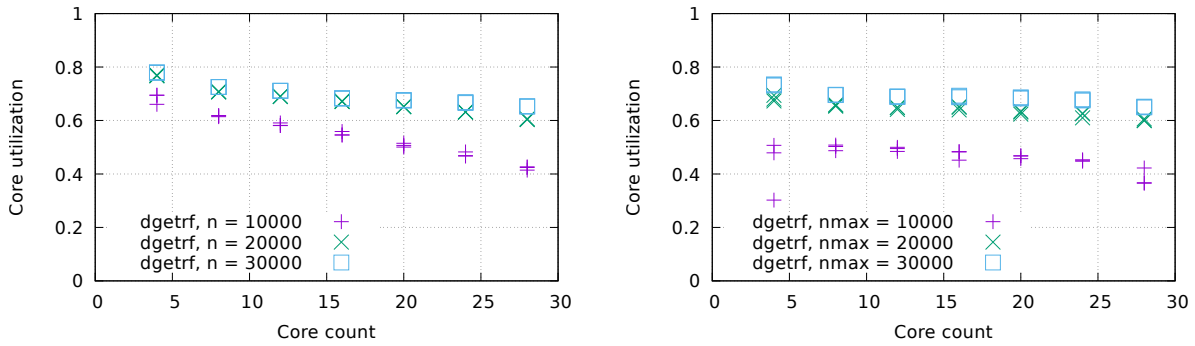
(a) Tile size sweep.

(b) Inner blocking size sweep.

(c) Empirical performance distribution.

Figure 6: Tunability of PLASMA `dgetrf` for $n = 20000$.

The results of the tile size sweep are reported in Figure 6a, and the results of the inner blocking size sweep are reported in Figure 6b. The empirical performance distribution is shown in Figure 6c. The results from the strong and weak scalability experiments are shown in Figures 7a and 7b, respectively.



(a) Strong scalability.

(b) Weak scalability.

Figure 7: Scalability of PLASMA `dgetrf`.

7 Results for the QR Factorization

A general matrix $A \in \mathbb{R}^{n\times n}$ admits a *QR factorization* $A = QR$ where $R \in \mathbb{R}^{n\times n}$ is upper triangular and $Q \in \mathbb{R}^{n\times n}$ is orthogonal. A QR factorization can be computed using Householder reflectors with flop count $F(n) = 4n^3/3 + \mathcal{O}(n^2)$.

The PLASMA version of `dgeqrf` has two tunable parameters: the *tile size* `nb` and the *inner blocking size* `ib`.

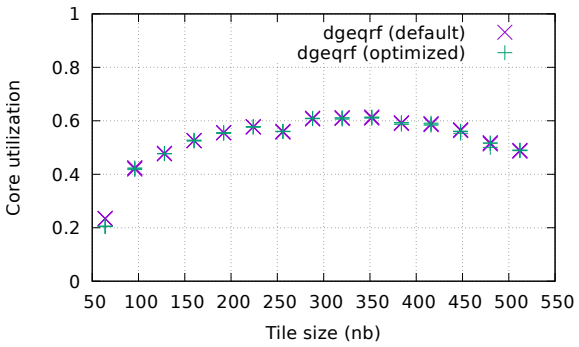
The results of the tile size sweep are reported in Figure 8a, and the results of the inner blocking size sweep are reported in Figure 8b. The empirical performance distribution is shown in Figure 8c. The results from the strong and weak scalability experiments are shown in Figures 9a and 9b, respectively.

8 Conclusions

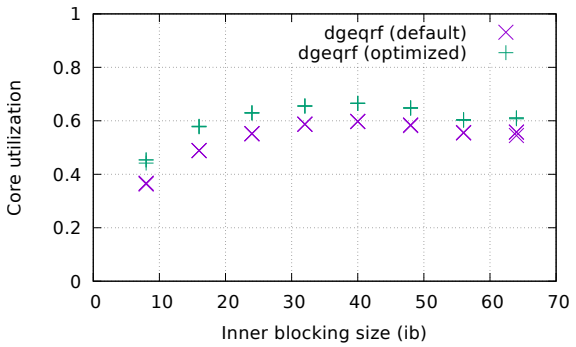
According to Figures 2a, 4a, 6a, and 8a it is important to not set the tile size parameter too small. For Cholesky and symmetric indefinite factorizations, the performance is insensitive to the tile size for a large range of values. In contrast, for both LU and QR there is a rather small range of appropriate values and the tile size can easily be set too large.

The inner blocking size parameter `ib` has, according to Figures 4b and 6b, a very small impact on the performance of both symmetric indefinite and LU. In contrast, the impact of the same parameter on the QR factorization is more substantial (see Figure 8b).

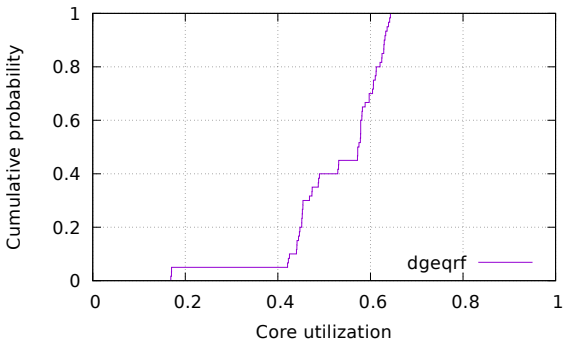
When it is likely that a parameter setting chosen at random results in a performance (or, equivalently, a core utilization) that is close to the best performance the implementation is capable of, then it is not particularly difficult to find a good parameter setting. We may therefore claim that such implementations are “easy to tune”. Conversely, an implementation that is “difficult to tune” makes it unlikely that a randomly selected setting will be good. The empirical performance distributions in Figures 2b, 4c, 6c, and 8c



(a) Tile size sweep.

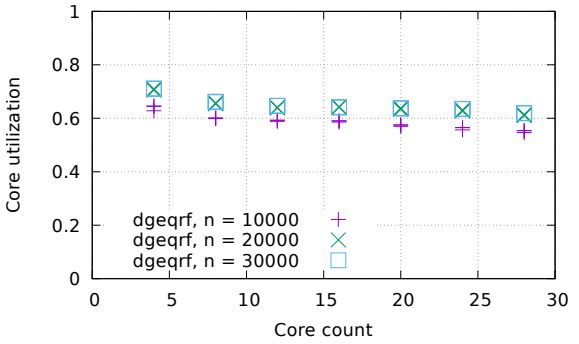


(b) Inner blocking size sweep.

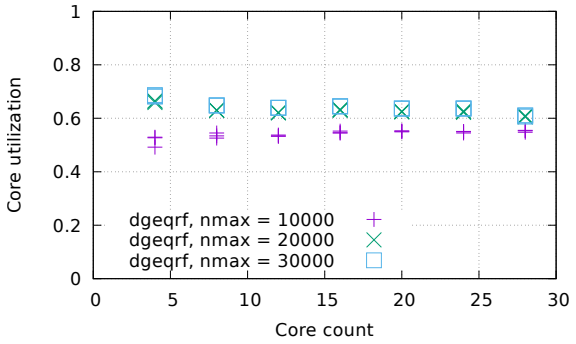


(c) Empirical performance distribution.

Figure 8: Tunability of PLASMA `dgeqrf` for $n = 20000$.



(a) Strong scalability.



(b) Weak scalability.

Figure 9: Scalability of PLASMA `dgeqrf`.

suggest that the easiest function to tune is `dpotrf` followed closely by `dsytrf` before a definite gap in difficulty to `dgetrf` and `dgeqrf` (which are comparable in difficulty).

The scalability experiments reported in Figures 3, 5, 7, and 9 show that all functions except `dsytrf` scale well and saturate the machine for matrices of dimension $n \gtrsim 20000$.

Acknowledgements

We thank the High Performance Computing Center North (HPC2N) at Umeå University for providing computational resources and valuable support during test and performance runs.