Efficient Performance of Parallel Cholesky Factorization Using Multi-Core System

Mohammed W. Al-Neama

Education College for Girls, Mosul University, Iraq

ABSTRACT

Modern processors have multiple cores, making multiprocessing essential for competitive linear algebra. Cholesky factorization considers one of the most essential linear algebra systems. This paper focused on a Cholesky factorization, as well as, apply the proposed technique to other problems in linear algebra. Theuse of the SCHEDULE package helps to introduce parallelism in a transportable way. High performances are obtained when the program is running by C++ with Open MP library on an Intel Core-i7-3770 quad-core processor of 3.40GHz and main memory of 8GB.

Keywords: Cholesky factorization; BLAS; symmetric linear systems; parallel computing.

HOW TO CITE THIS ARTICLE

Mohammed W. Al-Neama, “Efficient Performance of Parallel Cholesky Factorization Using Multi-Core System”, International Journal of Enhanced Research in Science, Technology & Engineering, ISSN: 2319-7463, Vol. 7 Issue 6, June -2018.

1 INTRODUCTION

The Cholesky factorization is considered one of the important analyzed algorithms in linear algebra[1]. It is a straightforward algorithm.
Let

$$A=LL^T$$

Where *:* a is a positive definite symmetric matrix-pivoting is not necessary-.
L is a lower triangular matrix.
The main step for a Cholesky factorization looks like:

```
For…
    For…
        For…
            $a_{ij} = a_{ij} - l_{ik} . a_{kj}$ ($l_{ik} = a_{ik}/a_{kk}$)
```

and it depends on the position of the loop parameters*i*, *j*, and*k*, respectively, whether it has been dealt with row, column or sub-matrix Cholesky factorization[2]. All forms have the same number of floating-point operations, indicating that the amount of arithmetic is exactly the same for all variants, although the access of data and the updating are different.

Column Choleskyappeared to be preferred for platform with vector-processing capabilities. For parallelism, shared memory systems with column or row draped interleaved storage are considered. We discuss on block-column Cholesky[3]. For convenience, letthe block-size NB is a divisor of matrix order*n*and

$$K = \frac{n}{NB} \quad (1)$$

The block factorization can be visualized as in (Figure 1). All elements of a are block-matrices: $A_{l:j-l,l:j-1}$, A_{jj}, and$A_{j+l:k,j+l:k}$ are symmetric block-matrices containing $(j-l)\times(j-1)$ blocks, a single block, and $(k-j)\times(k-j)$ blocks, respectively.

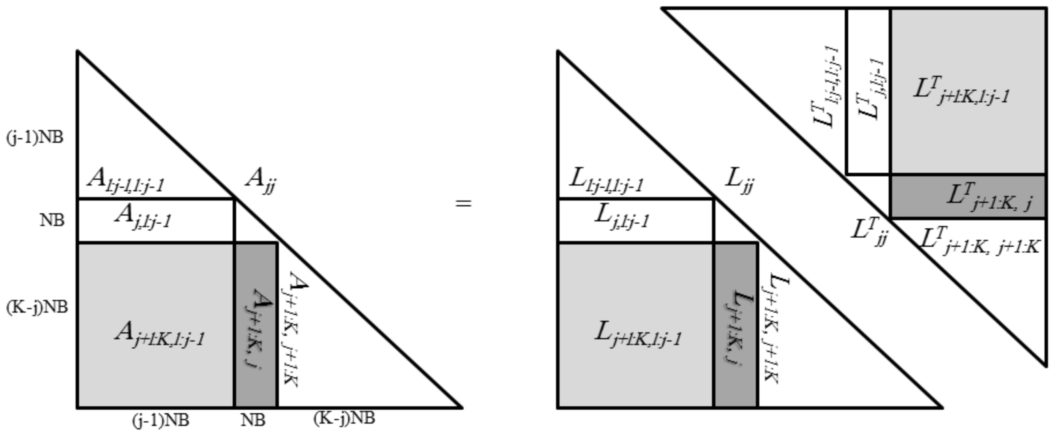


Figure 1: The block factorization can be visualized[1]

Assume the first *(j-l)×NB* columns, or the matrices $L_{1:j-1,1:j-1}$, $L_{j,1:j-1}$, *and* $L_{j+1:k,1:j-1}$ to be known. In step*j*, the next block-column (a set of NB single columns) of *L*will be computed. By equating compute obtain LL^T and A, then the following two relations will be obtained:

$$A_{jj} = L_{j,1:j-1}.L^T_{j,1:j-1} - L_{j,j}.L^T_{j,j} \quad (2)$$

$$A_{j+1:k,j} = L_{j+1:k,1:j-1}.L^T_{j,1:j-1} - L_{j+1:k,j}.L^T_{j,j} \quad (3)$$

From eq. (2) L_{jj} can be calculated. The operations involved are:

(l) a symmetric rank-k update:

$$A^{(1)}_{jj} \leftarrow A_{jj} - L_{j,1:j-1}.L^T_{j,1:j-1} \quad (4a)$$

(2)Asingle block of aCholesky factorization:

$$\mathrm{L_{jj}} \leftarrow \mathrm{Cholesky}(\mathrm{A^{(1)}_{jj}}) \quad (4b)$$

The eq. (3) delivers $L_{j+1:k,j}$.

(3)Amatrix-matrix product

$$A^{(1)}_{j+1:k,j} \leftarrow A_{j+1:k,j} - L_{j+1:k,1:j-1}.L^T_{j,1:j-1}. \quad (4c)$$

(4) Finally, a triangular system have to be solved

$$L_{j+1:k,j} \leftarrow A^{(1)}_{j+1:k,j}.L^{-T}_{jj} \quad (4d)$$

If all operations are performed on single blocks and if all components are single blocks as well, then (4a) will be as:

$$A^{(1)}_{jj} \leftarrow A_{jj} - \sum_{i=1}^{j-1} L_{ji}.L^T_{ji} \quad (5a)$$

where $L_{j,1:j-1}$will be divided into the single blocks L_{ji}, (i = *l*,...,*j* - *l*). Analogously, operation (4c) can be translated into

$$A^{(1)}_{jj} \leftarrow A_{lj} - \sum_{i=1}^{j-1} L_{li}.L^T_{li} \quad , l = j+1, \dots, K \quad (5b)$$

The multiplication of matrix-matrix in (5a) and (5b) are data-independent and can be carried out in parallel. This approach, however, requires additional memory, since the temporary result matrices

$$B^i_j = L_{ji}.L^T_{ji} \text{ and } C^i_{lj} = L_{li}.L^T_{ji},\ i = 1, \dots, j-1,\ l = j+1, \dots, k$$

are generated. In the end, the matrices B_j^i and C_{lj}^i, have to be subtracted from A_{jj} and A_{lj}, respectively.
An alternative way to perform the j^{th} step of the factorization is to translate (4a)-(4d) into:

$$A_{jj}^i \leftarrow A_{jj}^{i-1} - L_{ji} . L_{ji}^T \quad , i = 1, \dots, j-1, \qquad (6a)$$

$$\mathrm{L_{jj}} \leftarrow \mathrm{Cholesky}\left(\mathrm{A_{jj}^{j-1}}\right), \qquad (6b)$$

$$A_{lj}^i \leftarrow A_{lj}^{i-1} - L_{li} . L_{ji}^T \quad , i = 1, \dots, j-1, \; l = j+1 \dots, k, \qquad (6c)$$

$$L_{lj} \leftarrow A_{lj}^{j-1} . L_{jj}^{-T}, \; l = j+1 \dots, k, \qquad (6d)$$

where A_{ij}^0 denotes the original sub-matrix A_{ij}. In this case, the update of symmetric rank-*k* (6a) and the matrix-matrix products of the i-loop of (6c) can't be executed simultaneously, since each update requires data of the update of previously computed.

However, for different values of *l* the products of matrix-matrix, possibly followed by the solution of the triangular system (6d), are data-independent. Summarizing, the calculationsof (4a)-(4d) has been fine-grained calculations. In the next section, the execution dependencies between them in order to specify a parallel computation will be presented.

2 PROPOSED METHOD

Throughout this paper, computational kernels for basic operations in linear algebrawill be used. These kernels are termed the BLAS, for Basic Linear Algebra Subprograms. The Level 2 BLAS [1] incorporates matrix-vector operations, and the Level 3BLAS comprises matrix-matrix kernels[4]. In (6a)-(6d), the original computations were decomposed into fine-grained. Each of these will be associated with its BLAS function name. The Level 3 BLAS used are:

_ SYRK for performing a symmetric rank-*k* update on the diagonal blocks,
_ TRSM for solving a number of systems with the same triangular coefficient matrix.
_ GEMM for multiplying two matrices.

The fourth operation to perform is the Cholesky factorization, referred to as

_ LLT for factorizing a diagonal block.

In this paper**,** the term processor task rather than unit of computation will be used. A is partitioned into K×K blocks. Both A and the diagonal blocks are symmetric. From (6a)-(6d) the number of processes needed to compute the complete factorization of *A* will be derived:

$$\left.\begin{array}{r} \_LLT: K, \\ \_SYRK: \frac{1}{2}K(K-1), \\ \_TRSM: \frac{1}{2}K(K-1), \\ \_GEMM: \frac{1}{6}K(K-1)(K-2). \end{array}\right\} \qquad (7)$$

This implies that the total number of tasks will be

$$\mathrm{M}=\frac{1}{6}K(K-1)(K-2). \qquad (8)$$

For the description of our algorithm. It is convenient to number the tasks. A list schedule $\mathrm{L_M}$ of M tasks denoted by:

$$L_M = \{T_1, T_2, \dots, T_M\} \qquad (9)$$

Represents a certain order of the *M* tasks. The scheduling will be determined according to the choice of ordering. If the tasks on the matrices of the first column from 1 to k will be numbered and those of the second column (i.e., 2 operations/block) from *K+1* to *K+2(K-1)* and soon then the ordering that corresponds to the column Cholesky will be obtained. Analogously, a numbering along the rows will result in a row Cholesky, and a sub-matrix Cholesky corresponds with a numbering starting with the first updates succeeded by the second updates etcetera.

The aim of this study is to apply a simple scheduling strategy and to find an optimal value for *K,* the partitioning parameter (1).To obtain a good speed up with a multi tasked code, the cores concurrently have to keep active as much as possible and have to minimize memory conflicts between cores. An execution of the tasks strictly in conformity with one of our proposed numberings will not generate an optimal code. Many processes are data-dependent and cores may be idle while several tasks ready to be executed are waiting to be activated. Our algorithm for a fixed number of parallel cores say *p,* will execute tasks, even when their results are not needed at that time.

The scheduling strategy:

(A) The only schedulable task to start with is the factorization of the first diagonal block. As soon as this process has been completed, k-l tasks become schedulable, namely the calls to _TRSM on the first column matrices A_{j1}, $j = 2,...,k$. Assume that $p < K-1$, then the next *p* schedulable tasks will be continued. For $p>K-1$ only K-l cores can be active and $p-(K-1)$ cores will still be idle.

(B) When a process has finished on core P_α then other tasks may become schedulable. The next task on core P_α will be the first ready task in list L_M. The ordering of the tasks is determined by the selected numbering. If no schedulable tasks are available then core P_α has to wait until schedulable tasks are generated by tasks on other cores.

(C) Repeat (B) until all tasks have been completed. It is easy to determine for each process which dependencies have to be satisfied and to determine which processes depend on that specific process. The data dependency graph for a matrix partitioned into 5×5 blocks can be shown in (Figure 2). a node is specified by either an a or an L denoting the computation of

A_{ij}^k : k^{th} temporary update of sub-matrix A_{ij} (k < j-l),

L_{ij} : the final update of sub-matrix A_{ij} .

Note that the dependency graph of a matrix partitioned into *4×4* blocks is a part of the graph of (Figure 2), namely, that graph spanned by the nodes A_{ij}^k and L_{ij} with $1<j<i<4$. The amount of parallelism decreases during the course of the factorization will be remarked. At the end, the calculations of L_{54}, A_{55}^4, and L_{55} cannot be done parallel. Assume that the computation costs only depend on the number of floating-point operations. Let, 1 CU = the number of floating-point operations for a Cholesky factorization of a block of order NB.

So, the computational costs as listed in (Table 1) for the different operations on equally sized blocks of order NB. The value at the top right of a node in (Figure 2) stands for the minimal execution time expressed in CUs on an unbounded number of cores to perform the associated process and its preceding tasks. The solid line in (Figure 2) shows the critical (minimal) path of 35 CU.

The speed-up is defined by:

$$\boldsymbol{S} = \frac{\text{Time used by } l \text{core}}{\text{Time used by } p \text{cores}} \qquad (10)$$

and the efficiency by:

$$\text{Efficiency} = \frac{\mathrm{S_p}}{p} \times 100\%.$$

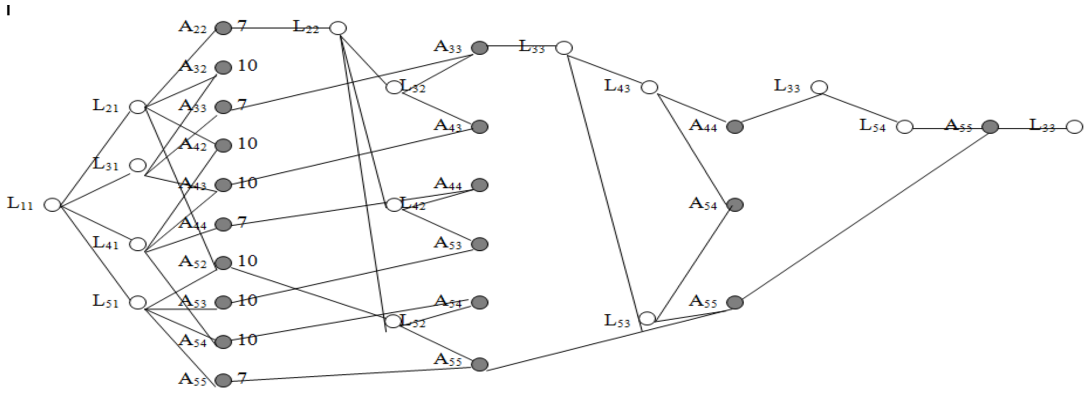


Figure 2: Shows the graph data dependency for the Cholesky factorization of*A* partitioned into 5×5 blocks on 4 cores

From (Figure 2) and the values of (Table 1), for this example with K = 5, the maximum speedup is:

$$\boldsymbol{S} = \frac{\text{Time required on1core}}{\text{Time required for critical path}} = \frac{125\ \text{CU}}{35\ \text{CU}} = 3.56$$

Table 1: Theoretical execution times expressed in CU's

Operation	FLOPs	CU
-LLT	1/3NB^3+…	1
-SYRK	NB^3+…	3
-TRSM	NB^3+…	3
-GEMM	3NB^3+…	6

assuming enough cores to be available. For the graph of (Figure 2) the maximum number of processes which can be computed in parallel is 10, namely the first updates $A_{22}^1, A_{32}^1, \ldots, A_{55}^1$ which cote: the whole matrix except for the first column. Hence. on ten or more cores. the efficiency cannot exceed 35.7%.

Suppose four cores are available, and the tasks have been numbered corresponding to the column Cholesky. Figure 3 shows the scheduling based on the CU distribution of(Table 1). In this case, 42 CU is required, and the speed-up is

$$\boldsymbol{S_4} = \frac{125\ \text{CU}}{42\ \text{CU}} = 2.98$$

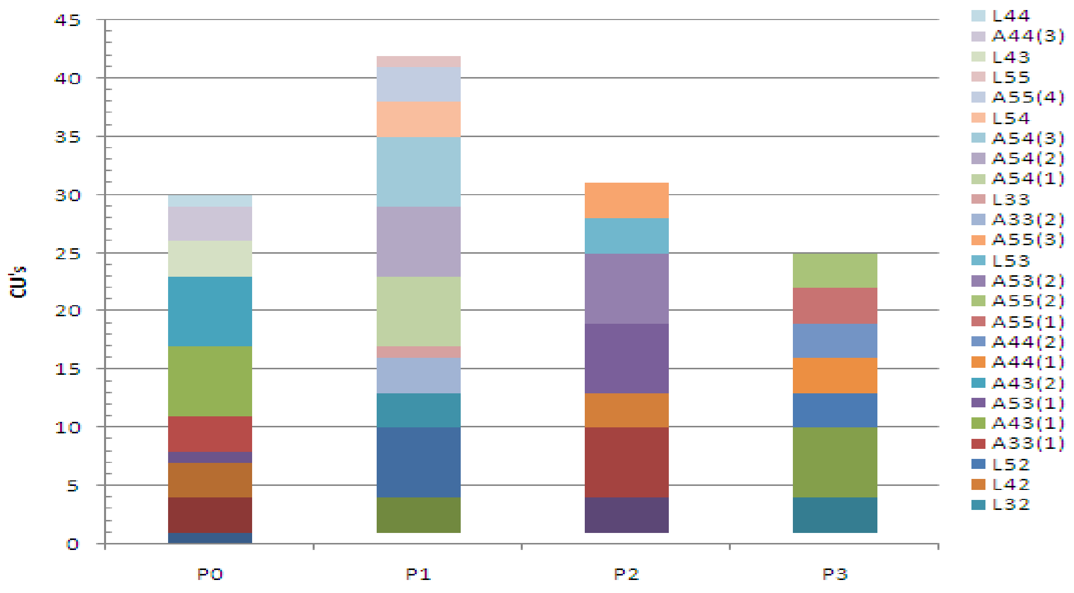


Figure 3. Scheduling of the Cholesky factorization of a **matrix partitioned into 5×5 blocks on four cores**

The efficiency of 74% is twice the efficiency obtained on 8 cores. Another numbering of the tasks might result in another speedup, and the speed-up for this example is not optimal. The critical path method (CP) considered as the most efficient heuristic method for solving the scheduling problem in hand needs 39 CU for our 5×5 problem. The CP method is based on initial execution time values T_i. The CP method will give rise to an inefficient execution when the estimated initial Ti values vary little from the values obtained during execution [5].

At the beginning, as well as, at the end of the factorization process, operations cannot be performed in parallel. To minimize the execution time of the initial and final phase, a smaller value of the block-size NH might be considered. Theoretically, on a fixed number of cores, the performance increases with the number of blocks and the maximum speedup will be reached for a block-size of l. In the next section,the characteristics of the platform will be shown how play an important role in the performance in the choice of the block-size.

3APORTABLE IMPLEMENTATION BASED ON SCHEDULE AND BLAS

When implementing parallel block algorithm efficient of that has be wide variety to a parallel machines needs portable implementation define dependencies data and to one a parallel and coordinate the parallel execution. For this purpose the SCHEDULE[6] package will be used. In addition, the implemented calls Level 3 BLAS. For the single diagonal blocks of order NB an unblocked Level 2 BLAS implementation of Cholesky factorization the used.

Machine dependencies

In the previous section, the performance of the parallel block algorithm depends on will be explained:

K	: the partitioning parameter,
P	: the number of cores,
$\boldsymbol{L}_M$	: the scheduling,
CU	: the ratio of the execution times for the different tasks.

Theoretically, the speedup for fixed values of K, p, L, and some well-defined CU-distribution could be calculated. In practice, however, machine-dependent aspects influence the CU-distribution. It is closely related to the BLAS implementation. The BLAS performance in turn strongly depends on the data structure and the block-size. Moreover, the influence of possible reuse of the cache contents can hardly be expressed in terms of the above mentioned variables[7]. In the following section, the scheduling by SCHEDULE will be focused on, which rather differs from the scheduling that proposed in previous Section.

The scheduling by SCHEDULE

The SCHEDULE package doesn't allow assigning priorities to tasks, which could be desirable, for example, to reuse cache (if possible) or to rank time-consuming tasks above less time-consuming tasks. This implies that the influence of a particular ordering cannot be measured. In practice, even for small values of K, the scheduling on a fixed number of cores turns out to be unique for each run. This can be explained by the execution times of the tasks. For this problem only four different execution times $T_{\text{-GEMM}}$, $T_{\text{-TRSM}}$, $T_{\text{-SYRK}}$ and $T_{\text{-DTT}}$will be distinguished. Most scheduling problems are dealing with a larger variation in execution times which makes it easier to predict the flow of execution. In Section 4.2 a few examples of the scheduling by SCHEDULE will be presented.

4 EXPERIMENTS

Three different implementations of the Cholesky factorization will be explained:

DLLTB
The algorithm as described in the previous sections will be referred to as urns. The way the data is stored influences the performance. Our algorithm operates on single blocks. For that reason, the matrix a fine-grained into blocks. This means that the matrix is stored block-wise by means of a four-dimensional array:

$$A[1{:}NB, 1{:}NB, 1{:}K, 1{:}K].$$

The clement $A[i, j, k, l]$ refers to element (i, j) of block (k, l).

DLLT3
In this paper, the performance of an “ordinary" Level 3 BLAS implementation to perform the Cholesky Factorization will be considered. It can be compared with DPOTRF from LAPACK[6]. The function DLLT3 exploits parallelism within the BLAS kernels. Operations are not performed on single blocks but on much larger block-matrices. Figure(4)displays how such blocks were composed.

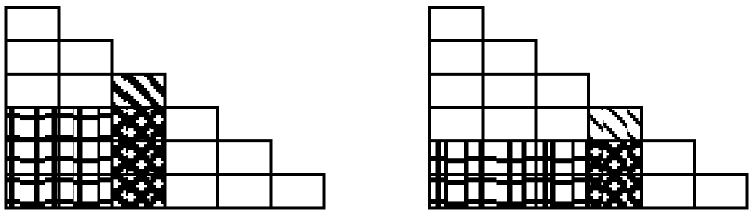


Figure 4: The combinations of hooks per step

DLLT

Both DLLTB and DLLT3 need a function to compute the Cholesky factorization of a sub-matrix of order NB. The unblocked implementation we used for this job is the function DLLT based on Level 2 BLAS. our performs a column Cholesky factorization which is well suited for vector machines[6].

The machines used in the numerical experiments are an Intel core-i7 with eight cores. In Section 3, we proposed to use BLAS to obtain high performances. All levels of BLAS are vendor-supported and these codes are more powerful than model implementations written in C++. For the SIMD vectorized codes are available for DGEMM and DTRSM, but neither of them has been parallelized. All experiments are carried out in double precision.

Performances of BLAS

Both the single and multi-core BLAS performancewill be considered[8]. We have experimented with several block-sizes, among which 64, 80, 96. The results for single NB×NBblocks (Table 2) can be used to analyze the performance of ours, since the SCHEDULE tasks are single core tasks each.

Table 2: Performance of BLAS 3 kernels on a single andeight cores with the speedup

Function	NB=64			NB=80			NB=96		
	Single core (sec.)	Eight cores (sec.)	Speed-up	Single core (sec.)	Eight cores (sec.)	Speed-up	Single core (sec.)	Eight cores (sec.)	Speed-up
DGEMM	19.23	5.77	3.33	37.50	8.11	4.63	73.12	10.50	6.97
DTRSM	10.48	3.11	3.37	20.44	4.37	4.68	39.85	5.66	7.04
DSYRK	3.50	0.98	3.57	6.83	1.38	4.96	13.31	1.78	7.46

Figure 5 shows the BLAS performance on eight cores. For each step *j*, the performance of the operations (4.a), (4.e). and (4d) was measured, denoted by DSYRK, DGEMM, and DTRSM, respectively. Figure (5.b) Illustrates that the BLAS performance on the single and eight cores strongly depends on (a) NB=64 (b) NB=80 (c) NB=96.

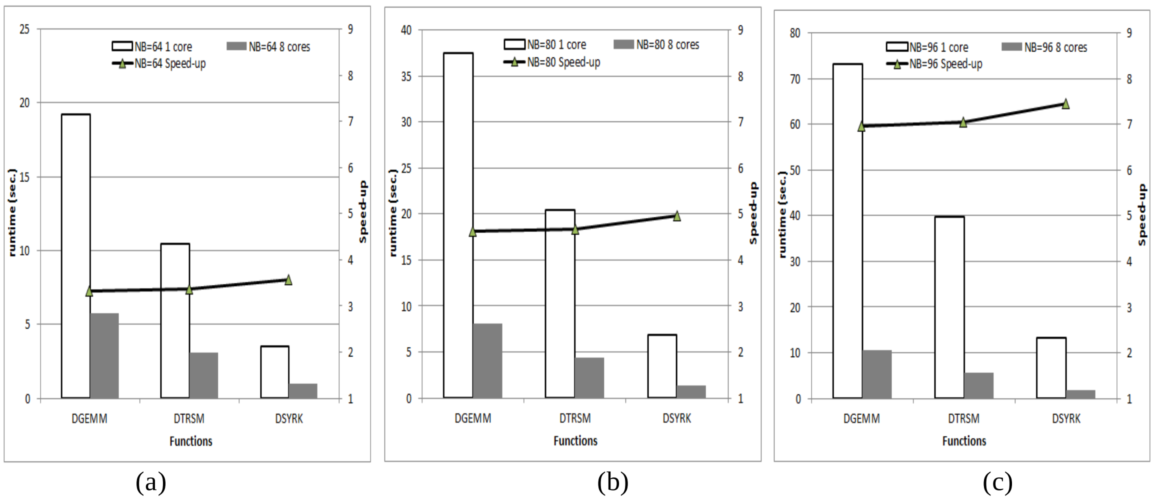


Figure 5: Illustrates that the BLAS performance on the single and eight cores strongly depends on (a) NB = 64 (b) NB = 80 (c) NB = 96

For the single core case. an improved version of DSYRK is used based on shared memory intrinsic function DOT PRODUCT[9-11]. This alternative code is not well suited for DLLT3, since *n* >> NB. For that case, it is better to use the original vendor-provided DSYRK implementation.

Performances of Cholesky factorizations

Unfortunately, the number of active tasks, that can be handled by SCHEDULE[5], is restricted to 1000. From the number of tasks *M* given by formula (8), we derive that *K*, presenting a partitioning into K×K blocks, may not exceed 17. The number of jobs is of order K3, which implies that an extension of the array lengths of SCHEDULE hardly conduces to a larger*K*value.

Figure (6.a) displays the speedup (cf. formula (10)) obtained for four cores. The block-size for this experiment is 64, and ***K***varies from 1 up to 17. In Figures (6.b-c), the speed-up for the eight-core was shown for NB=32 and NB=96, respectively. For the theoretical case, the speed-up was independent of the block-size. In practice, however, the speed-up will increase when the block-size increases, since the overhead of scheduling, such as the creation of the data dependency graph, will proportionally decline to the total computation time. Recall that the overhead of scheduling is minimal for one core, since processes never become data-dependent.

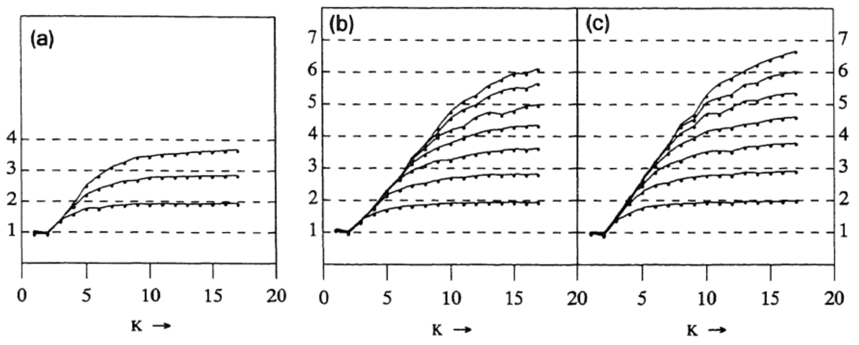
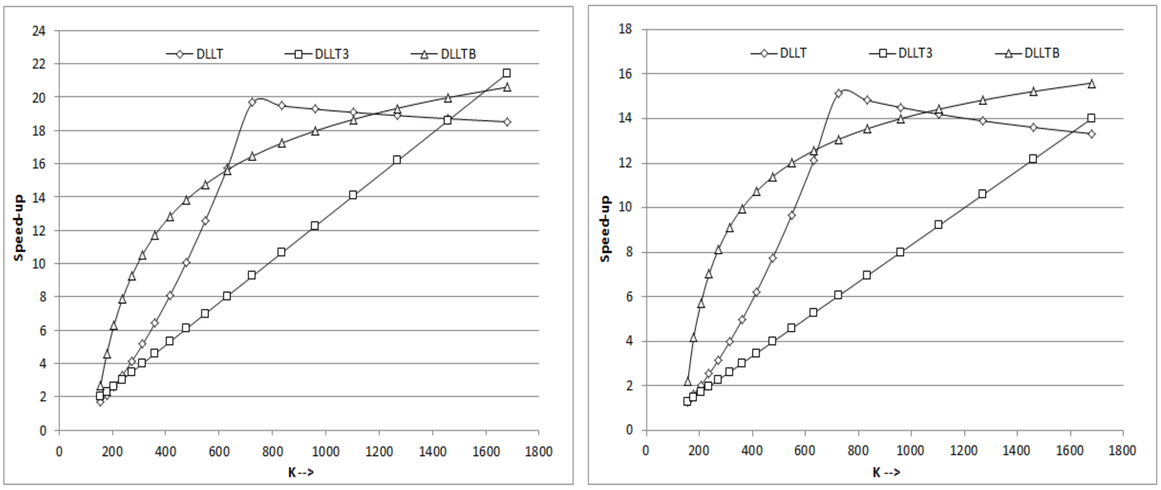


Figure 6: Speed-up for 2-4 processors on (a) NB = 64 (b) NB = 80 (c) NB = 96

In (Figure 7) the Mflops obtained for the four cores for DLLT (Level 2 BLAS), DLLT3 (Level 3 BLAS) and DLLTB (SCHEDULE combined with Level 3 BLAS) are listed. It is clear that the speed of the unblocked DLLT is independent of the number of blocks and does depend on the matrix order. This declares why its shape differs in each picture. The maximum performance of DLLT is reached for *n* = 256. TheDLLT3 with *K*=1 corresponds to a single is called DLLT. The same is true for DLLTB. However, in that case, DLLT will be performed on a single core.



(a) NB=64 **(b) NB=96**

Figure 7: Performance of DLLT, DLLT3 and DLLTB using 8 cores

The speed of DLLT3 on the eight cores is very disappointing, probably due to the low performance of the BLAS function DTRSM. the performance of DLLT3 on the four cores based on exactly the same BLAS implementation as used for DLLT3. The large difference in performance between DLLT3 and DLLTB is also caused by the different data structure. To illustrate this we have experimented with DLLTB, where the matrix to be factorized was stored block-wise (four-dimensional). The results for the four cores, and the elapsed time, are listed in (Table 3).

Table 3: Influence of data structure

Function	Number of blocks	NB	Mflops	No. of page faults & swaps	Elapsed time (sec.)
DLLTB (4-dim)	17	96	22.5	31pf+0w	15:86
DLLTB (2-dim)	17	96	20.3	1707 pf+2w	43:02
DLLT3 (2-dim)	17	96	18.3	553 pf+1w	50:25

For all experiments discussed up till now, the matrix order *n* was a multiple of the block-size NB. This yields that all blocks are of order NB. If not, then the sub-matrices of the last row *K* areofdimensionn-(K-1)×NB byNB,whereasthediagonalblockisoforder n-(K-l)×NB.As a consequence,theexecution time ofoperations on suchblocks will differ fromthe time requiredforsquareblocksoforderNB.Figure(8)displaystheperformancefor matricesoforder*n*=300,304,..., 800andforNB=32andNB=64.Thedottedlinesconnect thepointswithn=K×NBand *K* a positiveinteger.WeseethatforNB=32thehighest performance is obtainedfor such points.For NB=64 the opposite is true; function DLLTB performs even better in case of unequally sized blocks.

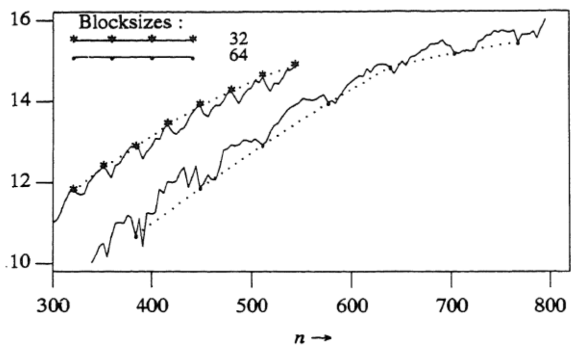


Figure 8: Performance of DLLTB with unequally sized block

CONCLUSIONS

The conclude of this study that parallelism over the blocks is a useful way to achieve high efficiency. This paper focused on the Cholesky factorization, but our technique can also be applied to other problems in linear algebra. The use of the SCHEDULE package helps to introduce parallelism in a transportable way. The platform used to run the program is Intel coreI7, high performances were obtained.

On the corei7 higher speed-up rates are achieved for program which applies parallelism over the kernels than for codes exploiting parallelism within the BLAS kernels. Moreover, the amount of data traffic has been reduced; for each core at most three sub-matrices of order NB are needed at a time.

This happens in case of a_GEMM operation, other operations need less data. If these sub-matrices are explicitly stored block-wise, then for a suitable block-size data can be kept in cache. For DLLT2 and DLLT3 situation is different. Here, the data needed per operation is not bounded by the block-size.

The data management can only be organized within a BLAS kernel and it is hoped that this is well done by the manufacturer. One important side effect on the different data access pattern which wouldbe mentioned. Not only was the CPU time for DLLTB less than for DLLT (Level 2 BLAS) and DLLT3 (bevel 3 BLAS).but also the wall clock time was much shorter.

Nevertheless. the performance of DLLTB based on SCHEDULE in combination with tuned BLAS can still be increased: firstly, when a more efficient BLAS particularly tuned for a single core can be used, and secondly, when a better scheduling of the tasks can be applied. a possible. The amount of partitioning into more than 1700×l700 blocks must possible,and the amount of parallelism is potentially high enough to experiment with other computation orderings which may result in a higher performance.

REFERENCES

[1]. Duff, I. S., Erisman, A. M., and Reid, J. K. (2017). *Direct methods for sparse matrices*. Oxford University Press.
[2]. Netzer, G. (2015). Efficient LU Factorization for Texas Instruments Keystone Architecture Digital Signal Cores.
[3]. Ortega, J. M. (2013). *Introduction to parallel and vector solution of linear systems*. Springer Science and Business Media.

[4]. Dongarra, J., Hammarling, S., Higham, N. J., Relton, S. D., Valero-Lara, P., andZounon, M. (2017). The Design and Performance of Batched BLAS on Modern High-Performance Computing Systems. *Procedia Computer Science*, *108*, 495-504.

[5]. Aronsson, P. (2006). *Automatic Parallelization of Equation-Based Simulation Programs* (Doctoral dissertation, Institutionenfördatavetenskap).

[6]. Hanson, F. B. (2003). Local supercomputing training in the computational sciences using remote national centers. *Future Generation Computer Systems*, *19*(8), 1335-1347.

[7]. Ortega, J. M. (2013). *Introduction to parallel and vector solution of linear systems*. Springer Science and Business Media.

[8]. Meng, J., Morozov, V. A., Kumaran, K., Vishwanath, V., andUram, T. D. (2011, November). GROPHECY: GPU performance projection from CPU code skeletons. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*(p. 14). ACM.

[9]. Amestoy, P. R., Duff, I. S., andL'excellent, J. Y. (2000). Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer methods in applied mechanics and engineering*, *184*(2), 501-520.

[10]. Dong, T., Haidar, A., Luszczek, P., Harris, J. A., Tomov, S., andDongarra, J. (2014, August). LU factorization of small matrices: Accelerating batched DGETRF on the GPU. In *High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), 2014 IEEE Intl Conf on* (pp. 157-160). IEEE.

[11]. Amoozegar, M., andNezamabadi-pour, H. (2015). a multi-objective approach to model-driven performance bottlenecks mitigation. *ScientiaIranica. Transaction D, Computer Science and Engineering, Electrical*, *22*(3), 1018.