High Performance Matrix Computations
Homework 2 — Reference Solution

Elmar Peise

June 30, 2015

1 Step 1: Installation

1.1 System description

We use ELAPS on a cluster node with two Intel Sandy Bridge-EP E5-2670 CPU running at 2.6 GHz. Each CPU has 8 cores and up to 2 hardware threads per core (hyper-threading). Each core can perform 8 double precision floating point operations per cycle: 1 ADD + 1 MUL on vectors of 4 doubles (AVX). Hence the theoretical double precision peak performance is $2.6 \cdot 8 = 20.8$ GFLOPS/s per core. The single precision performance is twice is high.

The node is used interactively and exclusively.

1.2 Sampler Setup

Our Sampler uses OpenBLAS version 0.2.14 and is generated with the following configuration file (`step1/SandyBridge_OpenBLAS.cfg`):

```
 1  BLAS_NAME=OpenBLAS
 2  SYSTEM_NAME=SandyBridge
 3  CFLAGS="-fopenmp"
 4  CXXFLAGS="-fopenmp"
 5  LINK_FLAGS="-L${HOME}/software/openblas/lib/ \
 6  -lopenblas_sandybridgep-r0.2.14 -lgfortran -fopenmp \
 7  -L${PAPI_ROOT}/lib64 -lpapi"
 8  INCLUDE_FLAGS="-I${PAPI_ROOT}/include"
 9  BACKEND_PREFIX="OPENBLAS_NUM_THREADS={nt}"
10  . ./gathercfg.sh
11  DFLOPS_PER_CYCLE=8
```

The Sampler is set up to be used interactively. If we were to use it through the LSF batch job system, we would add the similar to the following:

```
BACKEND="lsf"
BACKEND_HEADER="#BSUB -x -M 10000 -W 00:15 -R model==SandyBridge_EP"
```

Note that this script configures how the number of threads for OpenBLAS is controlled in line 9.

1.3 `dgemm` performance

We use the following bash script as an input for the generated Sampler to test the performance of `dgemm` on square matrices of increasing size (`step1/input.sh`):

```
#!/bin/bash
for n in {100..4000..100}; do
    echo dgemm N N $n $n $n 1 [$((n * n))] $n [$((n * n))] $n 1 [$((n * n))] $n
done
```

(The produce output can be found in `step1/output.txt`.)

The GFLOPS/s plot for this experiment in Figure 1 is generated using the following gnuplot script (`step1/gflops.plt`):

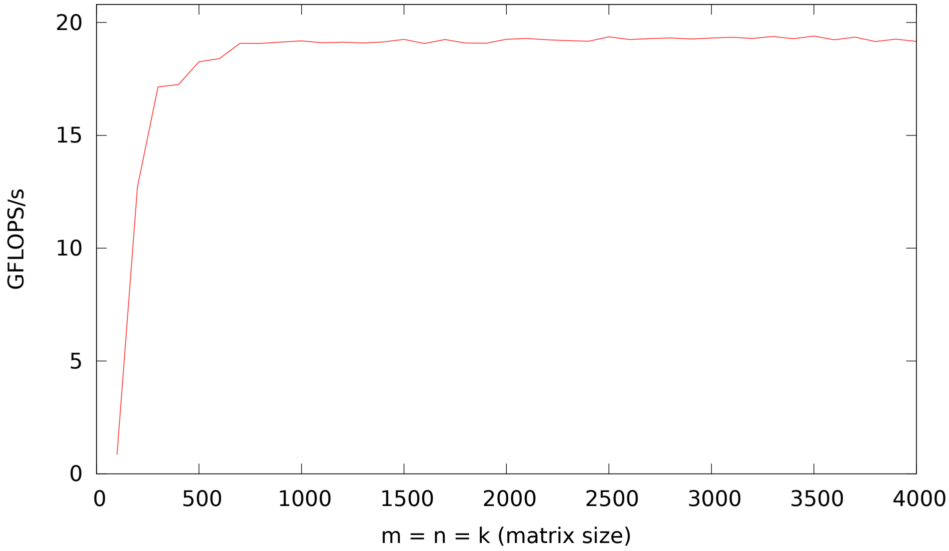


Figure 1: Performance of `dgemm`

```
set term pdfcairo
set output "gflops.pdf"
set xlabel "m = n = k (matrix size)"
set ylabel "GFLOPS/s"
set nokey
set yrange [0:20.8]
plot "output.txt" u (100 * ($0 + 1)):(2.6 * 2 * (100 * ($0 + 1)) ** 3 / $1) w l t "dgemm"
```

The plot shows the expected performance profile for `dgemm`, reaching ≈ 19 GFLOPS/s, which corresponds an to efficiency of $\approx 90\%$.

2 Step 2: BLAS-3

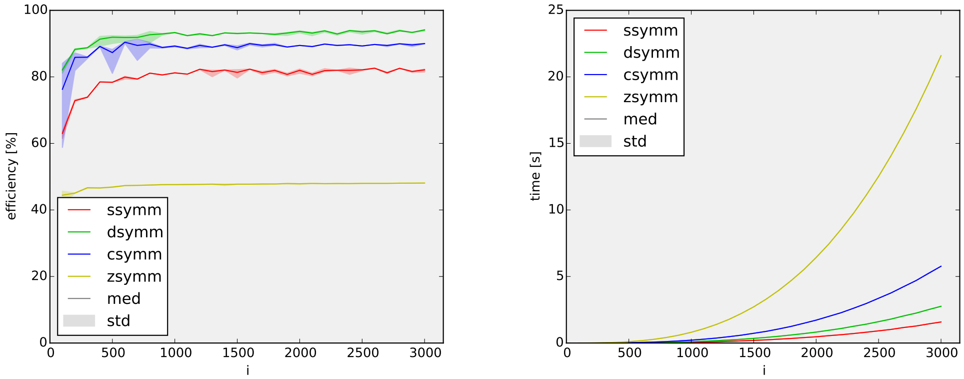
2.1 Performance of `symm`

We consider the kernel `symm` ($C = C + AB$, where A is symmetric). For `ssymm` we use the experiment setup `step2/{s,d,c,z}symm.elr` (Figure 8). The experiments for the other datatypes are analogous.

The resulting execution time and efficiencies of these experiments shown in Figure 2. For single real, single complex, and double real, OpenBLAS reaches an efficiencies between 80 and 95%; for double precision complex on the other hand the efficiency doesn't even reach 50%, which indicates that the kernel is not optimized for AVX 2. Considering the execution times, the double precision kernels take about twice as long as the corresponding single precision counter parts since the CPU can perform twice as many single precision operations per cycles as double precision operations. The execution times for the complex kernels is about four times higher than for the real kernels since these kernels involve four times more mathematical operations.

2.2 Multithreaded OpenBLAS vs. OpenMP

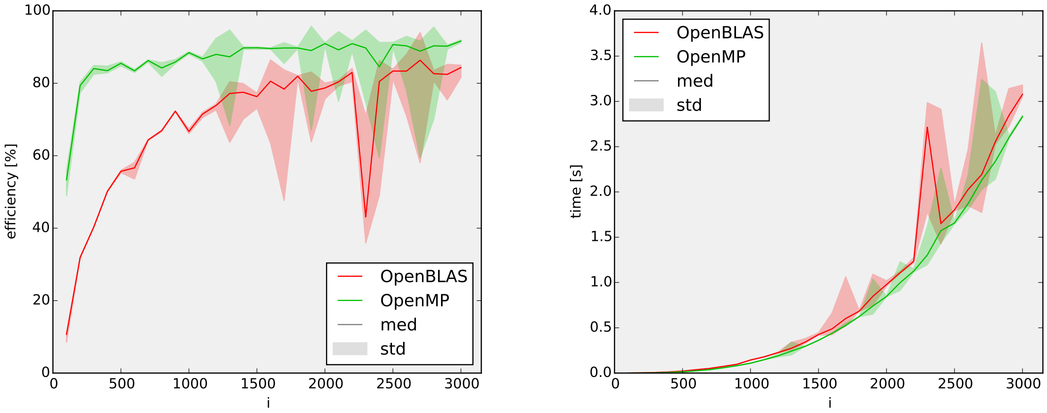
With `step2/{mt,omp}.elr` (Figure 9),w e compare the performance of multithreaded OpenBLAS with running single-threaded kernels in parallel through OpenMP for a sequnce of 16 `dsymm` kernels. Here, the symmetric matrix A is the same in each kernel invocation, while dfferent matrices B and C are used. The efficiency and execution time of their results are shown in Figure 3; across the board running single-threaded kernels in parallel through OpenMP is faster



(a) Efficiency

(b) Execution time

Figure 2: Perfromance of `symm` for all data types



(a) Efficiency

(b) Execution time

Figure 3: Multithreaded OpenBLAS vs. OpenMP-parallel kernels

than running the kernels one after the other with multithreaded OpenBLAS. Note that in these experiments an efficiency of 100% corresponds to 128 flops/cycle.

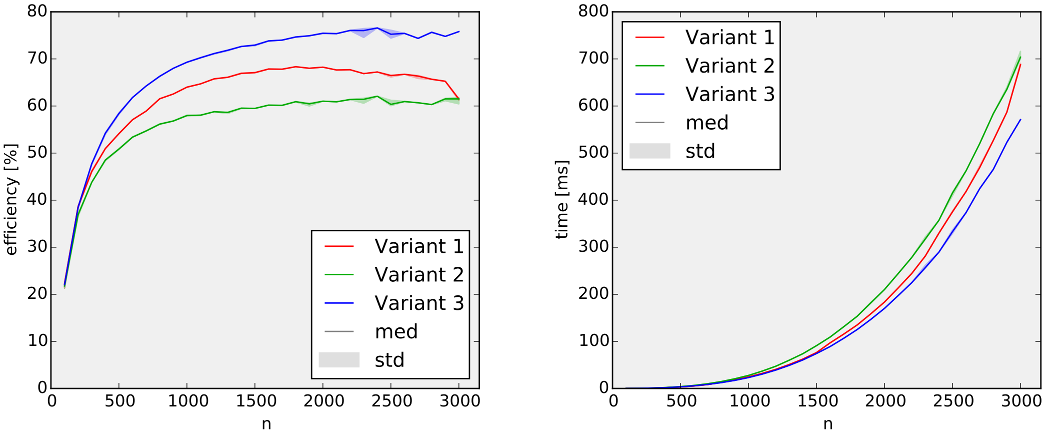
3 Step 3: Cholesky

3.1 Cholesky Variants for Different Matrix Sizes

The three blocked algorithms for the Cholesky decomposition with increasing matrix size n and the block-size fixed to 32 is given by `step3/chol{1,2,3}.elr` (Figure 10). Here, p represents the traversal along the diagonal. Note that the SPD randomization kernel `dporand` is only invoked on A_{11}. The performance of the experiments are shown in Figure 4; all variants present a similar behaviour, while variant 3 is the fastest.

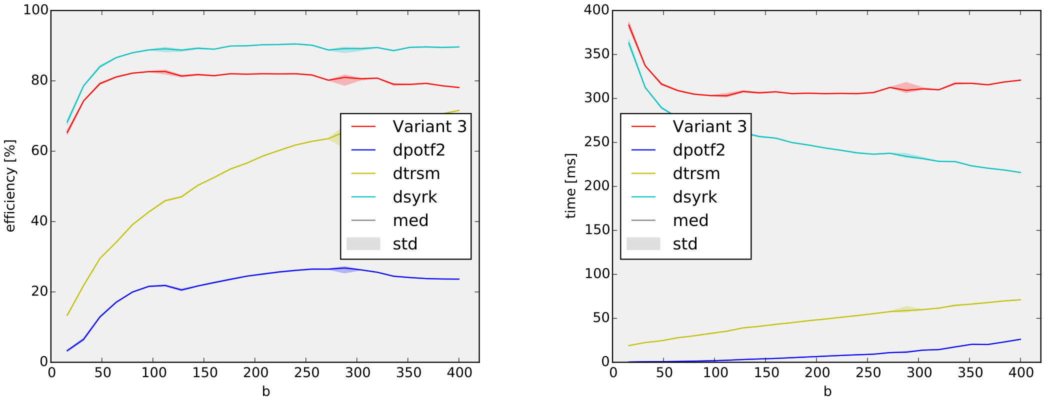
3.2 Block-Size Optimization for Cholesky Variant 3

We use the experiment `step3/chol3_nb.elr` (Figure 11) to determine the optimal block-size for Cholesky variant 3 and matrix-size 2500. The resulting performance is shown in Figure 5. The



(a) Efficiency (b) Execution time

Figure 4: Performance of the Cholesky variants for different matrix sizes



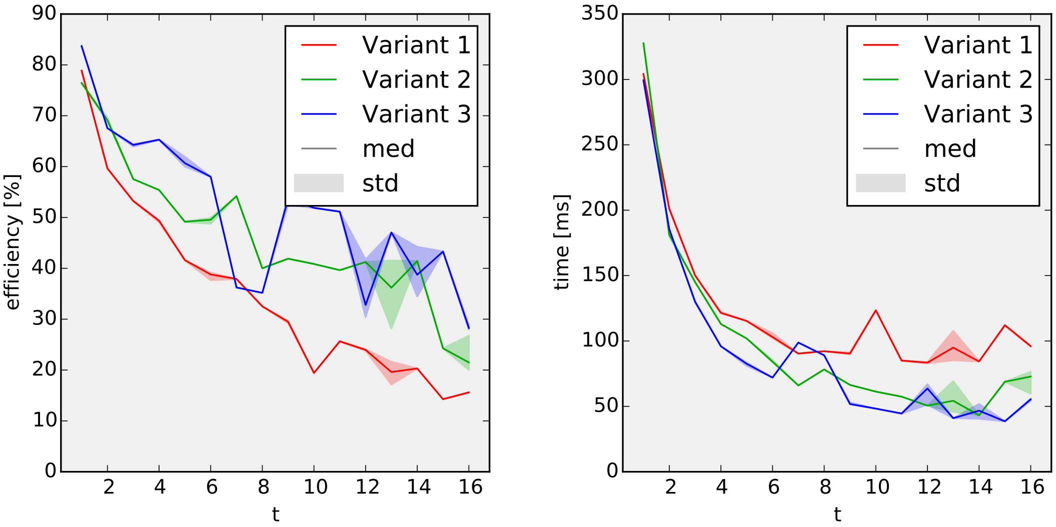
(a) Efficiency (b) Execution time

Figure 5: Performance of Cholesky variant 3 for varying block-sizes at matrix size 2500.

best efficiency is obtained for a block-size of 112. Furthermore, we can see how the block-size influences performance: For very small block-sizes, `dsyrk` is invoked very often with a very thin A_{21}, for which is not very efficient; as the block-size increases, the efficiency of `dsyrk` increases quickly, while the computational load is shifted towards `dtrsm` and eventually `dpotf2`; since the latter is very inefficient for larger matrices, increasing the block-size too far, will also yield poor performance.

3.3 Parallel Scalability

We analyze the scalability of the three Cholesky variants for matrix size 2500 and block-size 112 with the experiments `step3/chol{1,2,3}_mt.elr` (Figure 12). The resulting performance is shown in Figure 6. For all three variants, the eficiency quickly drops from ≈ 80% to 15-40, while variant 3 remains the fastest.



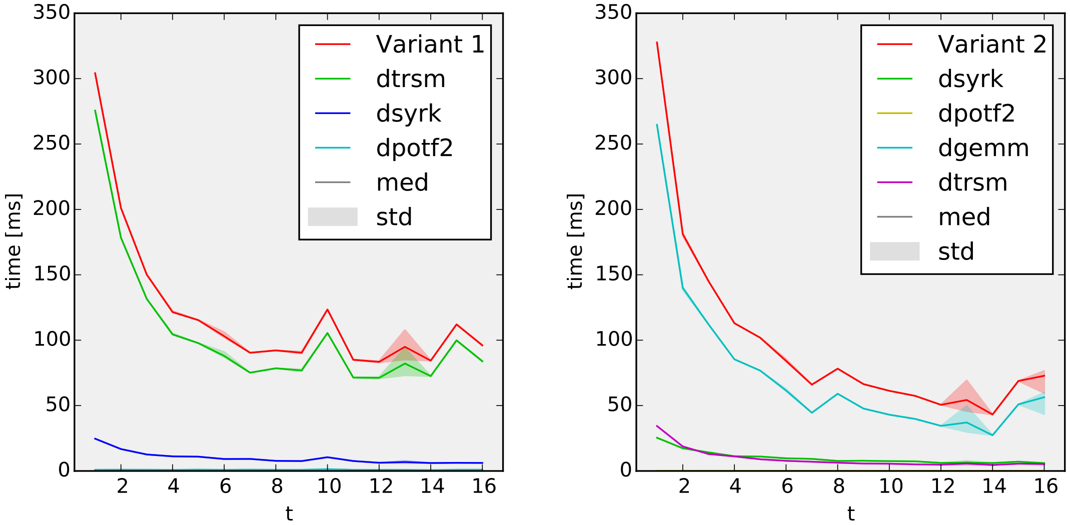
(a) Efficiency

(b) Execution time

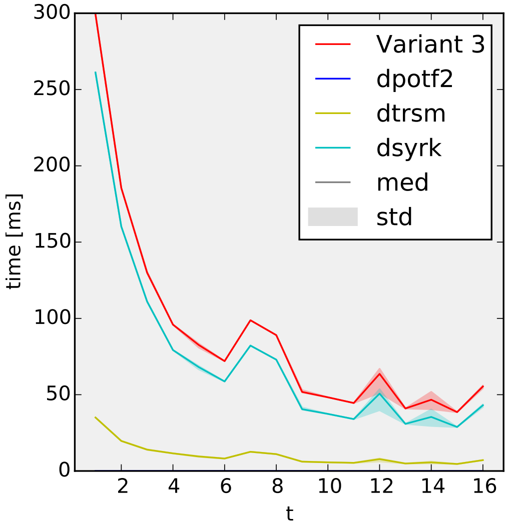
Figure 6: Scalability of the three Cholesky variants for matrix size 2500 and block-size 112

3.4 Bottleneck analysis

For the bottleneck analysis we consider the parallel scaling experiments `step3/chol{1,2,3}_mt.elr` from the previous section. A breakdown of the time spent in each kernel is given in Figure 7. The results show that for each kernel the scaling behaviour is strongly dominated by the kernel that covers most of the algorithm's computation. These are `dtrsm` involving A_{00} (variant 1), `dgemm` involving A_{21} (variant 2), and `dsyrk` involving A_{22} (variant 3).



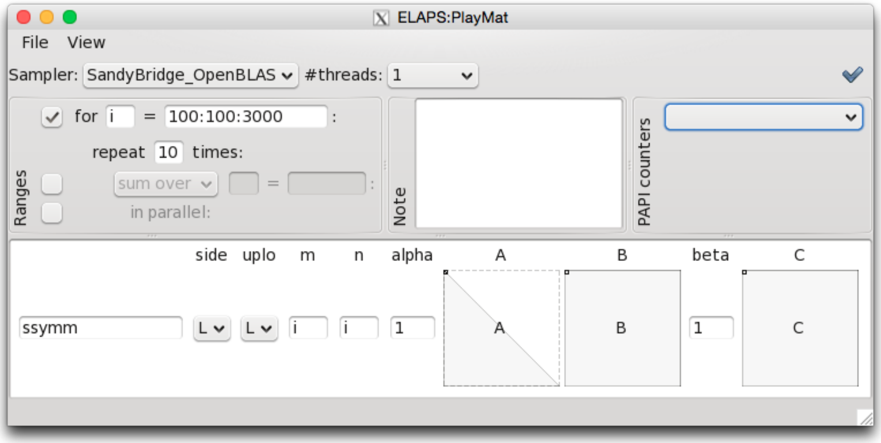
(a) Variant 1 (b) Variant 2



(c) Variant 3

Figure 7: Breakdown of the time spent in each kernel of the scalability experiments.

A Experiment Setups



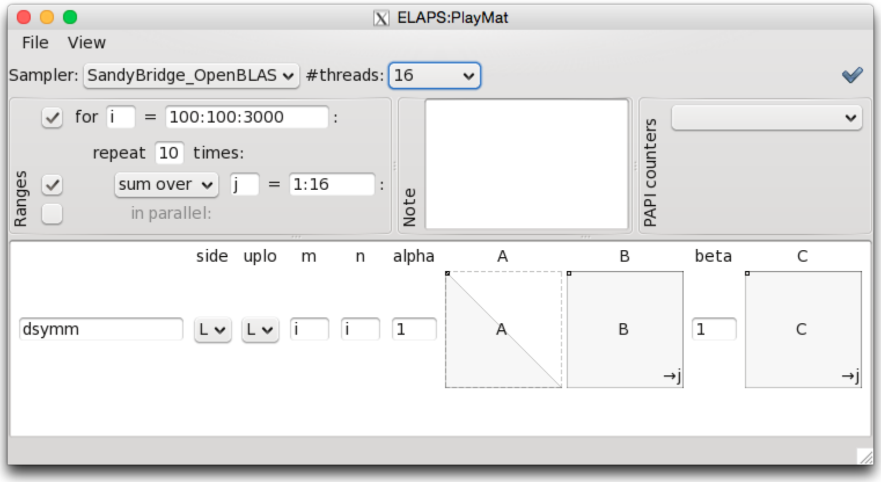
Sampler: SandyBridge_OpenBLAS #threads: 1

☑ for i = 100:100:3000 :

repeat 10 times:

	side	uplo	m	n	alpha	A	B	beta	C
ssymm	L	L	i	i	1	A	B	1	C

Figure 8: Setup for `step2/ssymm.elr`.



Sampler: SandyBridge_OpenBLAS #threads: 16

☑ for i = 100:100:3000 :

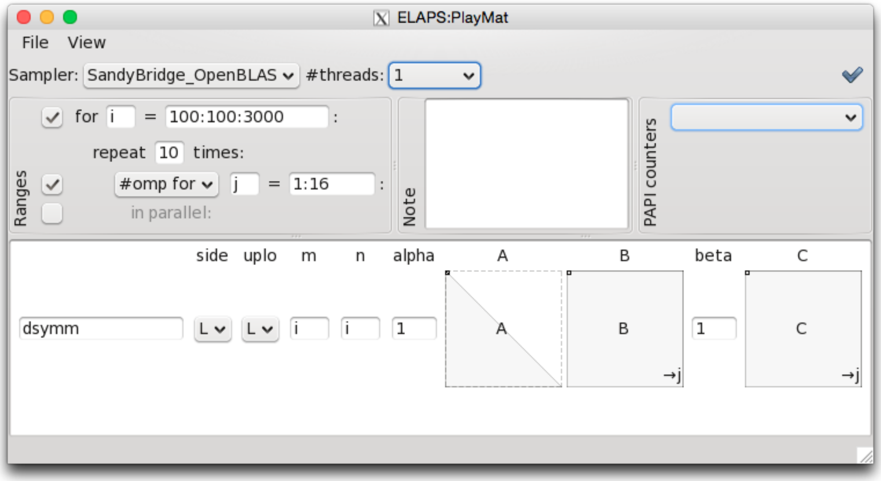
repeat 10 times:

☑ sum over j = 1:16 :

☐ in parallel:

	side	uplo	m	n	alpha	A	B	beta	C
dsymm	L	L	i	i	1	A	B	1	C

(a) Setup for multithreaded OpenBLAS (`step2/mt.elr`)



Sampler: SandyBridge_OpenBLAS #threads: 1

☑ for i = 100:100:3000 :

repeat 10 times:

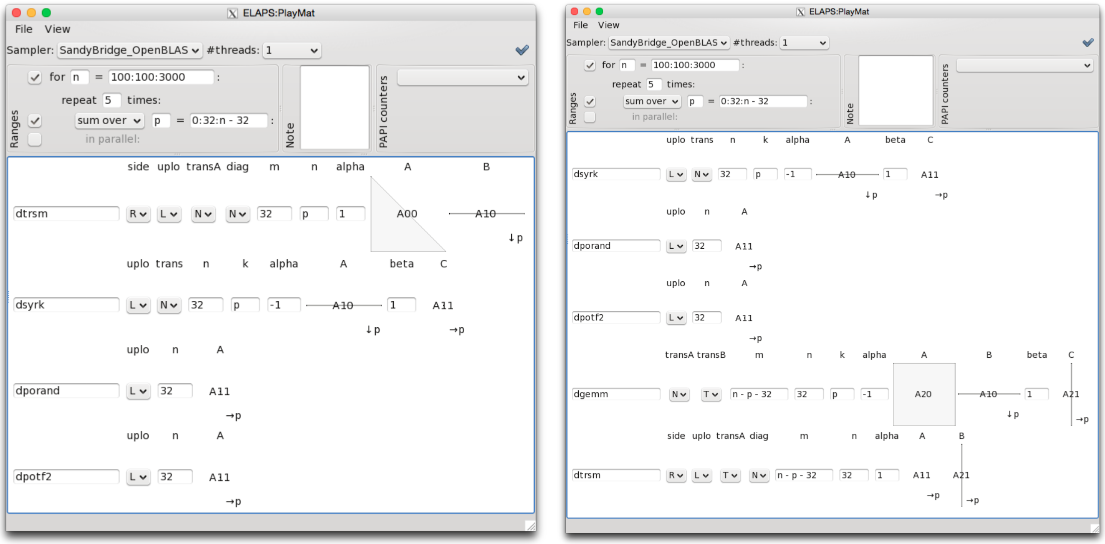
☑ #omp for j = 1:16 :

☐ in parallel:

	side	uplo	m	n	alpha	A	B	beta	C
dsymm	L	L	i	i	1	A	B	1	C

(b) Setup for OpenMP parallel kernels (`step2/omp.elr`)

Figure 9: Setups for the parallelism comparison OpenBLAS vs. OpenMP



Sampler: SandyBridge_OpenBLAS #threads: 1

☑ for n = 100:100:3000 :

repeat 5 times:

☑ sum over p = 0:32:n - 32 :

☐ in parallel:

	side	uplo	transA	diag	m	n	alpha	A	B
dtrsm	R	L	N	N	32	p	1	A00	A10 ↓p

	uplo	trans	n	k	alpha	A	beta	C
dsyrk	L	N	32	p	-1	A10 ↓p	1	A11 →p

	uplo	n	A
dporand	L	32	A11 →p

	uplo	n	A
dpotf2	L	32	A11 →p

(a) Variant 1 (`step3/chol1.elr`)

Sampler: SandyBridge_OpenBLAS #threads: 1

☑ for n = 100:100:3000 :

repeat 5 times:

☑ sum over p = 0:32:n - 32 :

☐ in parallel:

	uplo	trans	n	k	alpha	A	beta	C
dsyrk	L	N	32	p	-1	A10 ↓p	1	A11 →p

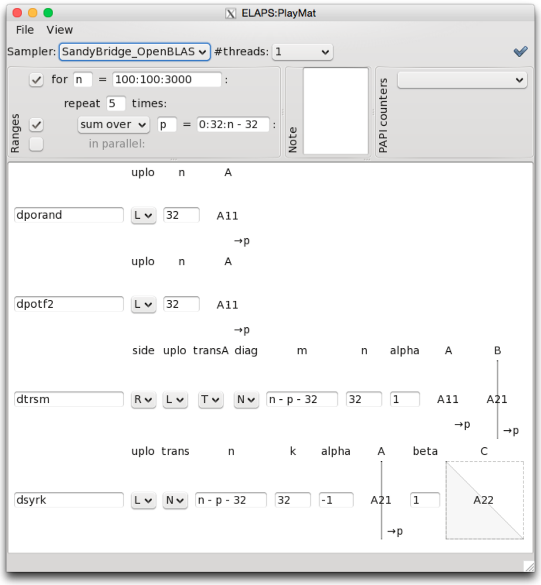
	uplo	n	A
dporand	L	32	A11 →p

	uplo	n	A
dpotf2	L	32	A11 →p

	transA	transB	m	n	k	alpha	A	B	beta	C
dgemm	N	T	n - p - 32	32	p	-1	A20	A10 ↓p	1	A21 →p

	side	uplo	transA	diag	m	n	alpha	A	B
dtrsm	R	L	T	N	n - p - 32	32	1	A11 →p	A21 →p

(b) Variant 2 (`step3/chol2.elr`)



Sampler: SandyBridge_OpenBLAS #threads: 1

☑ for n = 100:100:3000 :

repeat 5 times:

☑ sum over p = 0:32:n - 32 :

☐ in parallel:

	uplo	n	A
dporand	L	32	A11 →p

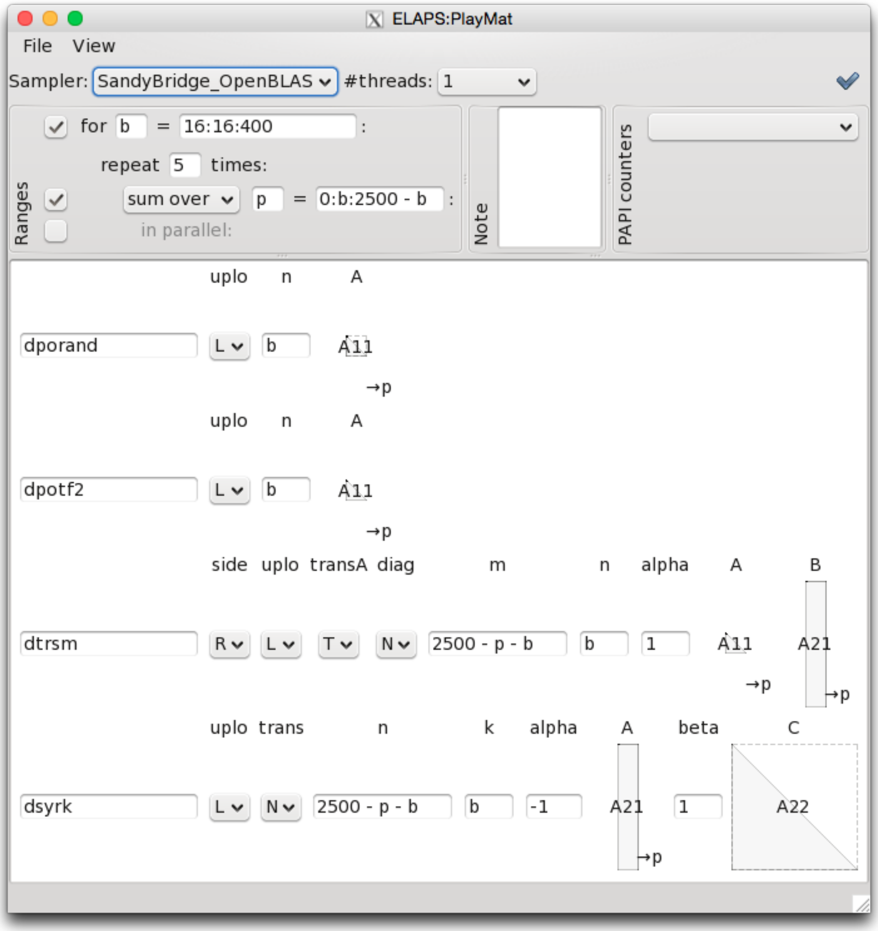
	uplo	n	A
dpotf2	L	32	A11 →p

	side	uplo	transA	diag	m	n	alpha	A	B
dtrsm	R	L	T	N	n - p - 32	32	1	A11 →p	A21 →p

	uplo	trans	n	k	alpha	A	beta	C
dsyrk	L	N	n - p - 32	32	-1	A21 →p	1	A22

(c) Variant 3 (`step3/chol3.elr`)

Figure 10: Setups for the blocked Cholesky algorithms



Sampler: SandyBridge_OpenBLAS #threads: 1

☑ for b = 16:16:400 :

repeat 5 times:

☑ sum over p = 0:b:2500 - b :

☐ in parallel:

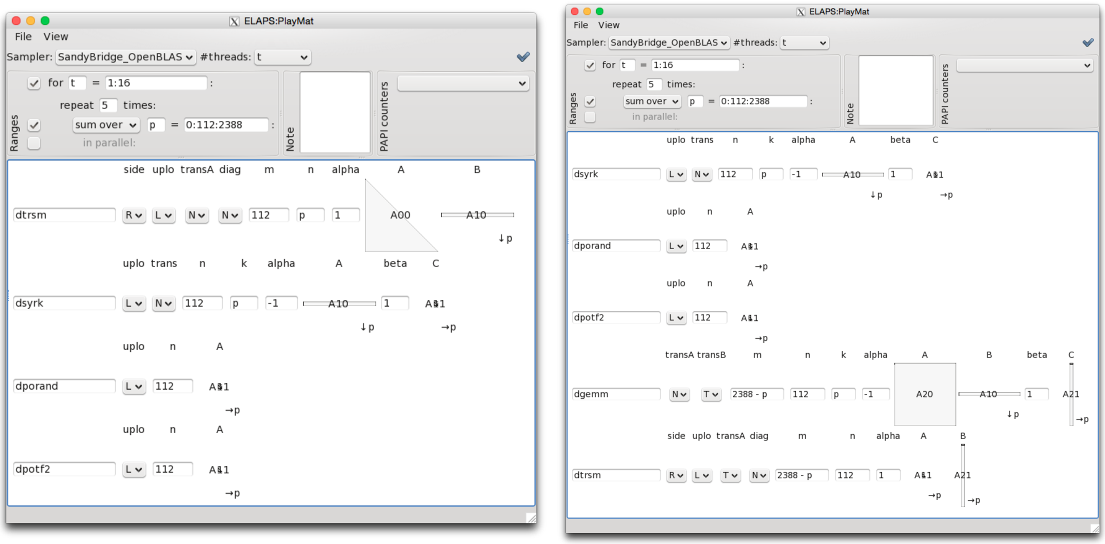
dporand: uplo L, n b, A A11 →p

dpotf2: uplo L, n b, A A11 →p

dtrsm: side R, uplo L, transA T, diag N, m 2500 - p - b, n b, alpha 1, A A11 →p, B A21 →p

dsyrk: uplo L, trans N, n 2500 - p - b, k b, alpha -1, A A21 →p, beta 1, C A22

Figure 11: Block-size optimization for Cholesky variant 3 (`step3/chol3_nb.elr`)



Sampler: SandyBridge_OpenBLAS #threads: t

☑ for t = 1:16 :

repeat 5 times:

☑ sum over p = 0:112:2388 :

☐ in parallel:

	side	uplo	transA	diag	m	n	alpha	A	B
dtrsm	R	L	N	N	112	p	1	A00	A10 ↓p

	uplo	trans	n	k	alpha	A	beta	C
dsyrk	L	N	112	p	-1	A10 ↓p	1	A11 →p

	uplo	n	A
dporand	L	112	A11 →p

	uplo	n	A
dpotf2	L	112	A11 →p

(a) Variant 1 (`step3/chol1_mt.elr`)

Sampler: SandyBridge_OpenBLAS #threads: t

☑ for t = 1:16 :

repeat 5 times:

☑ sum over p = 0:112:2388 :

☐ in parallel:

	uplo	trans	n	k	alpha	A	beta	C
dsyrk	L	N	112	p	-1	A10 ↓p	1	A11 →p

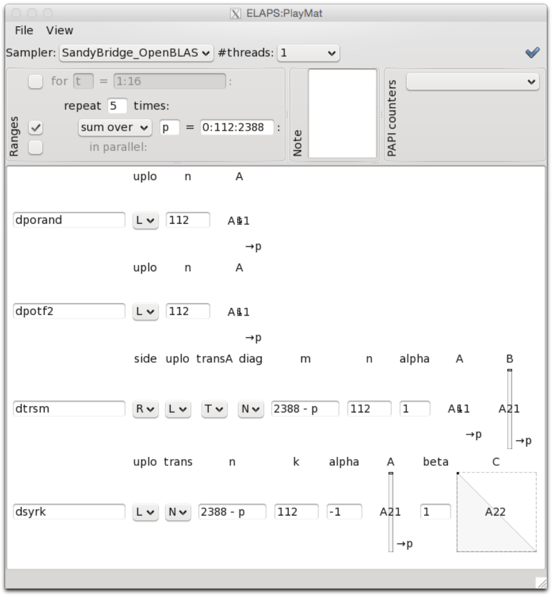
	uplo	n	A
dporand	L	112	A11 →p

	uplo	n	A
dpotf2	L	112	A11 →p

	transA	transB	m	n	k	alpha	A	B	beta	C
dgemm	N	T	2388 - p	112	p	-1	A20	A10 ↓p	1	A21 →p

	side	uplo	transA	diag	m	n	alpha	A	B
dtrsm	R	L	T	N	2388 - p	112	1	A11 →p	A21 →p

(b) Variant 2 (`step3/chol2_mt.elr`)



Sampler: SandyBridge_OpenBLAS #threads: 1

☐ for t = 1:16 :

repeat 5 times:

☑ sum over p = 0:112:2388 :

☐ in parallel:

	uplo	n	A
dporand	L	112	A11 →p

	uplo	n	A
dpotf2	L	112	A11 →p

	side	uplo	transA	diag	m	n	alpha	A	B
dtrsm	R	L	T	N	2388 - p	112	1	A11 →p	A21 →p

	uplo	trans	n	k	alpha	A	beta	C
dsyrk	L	N	2388 - p	112	-1	A21 →p	1	A22

(c) Variant 3 (`step3/chol3_mt.elr`)

Figure 12: Setups for the blocked Cholesky algorithms with increasing number of threads