Performance Analysis Using PaRSEC Profiling Tools: Case Study with TLR Cholesky

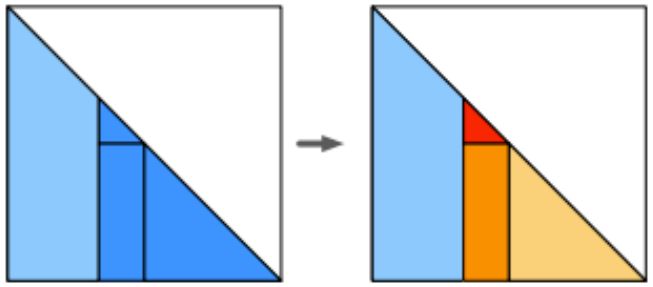
Yu Pei, Qinglei Cao, Thomas Herault, George Bosilca

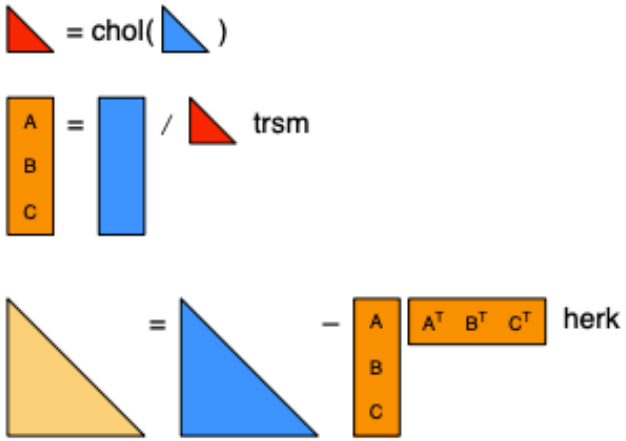
May 7th, 2020

PaRSEC User Group Meeting

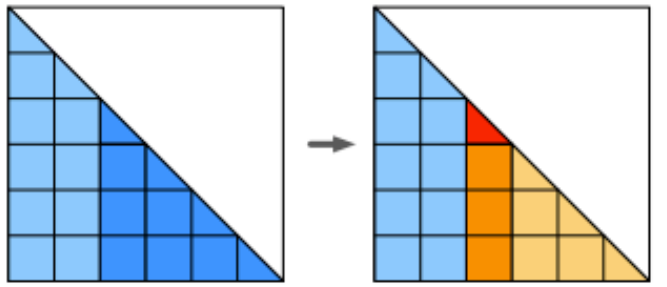
Cholesky Factorization

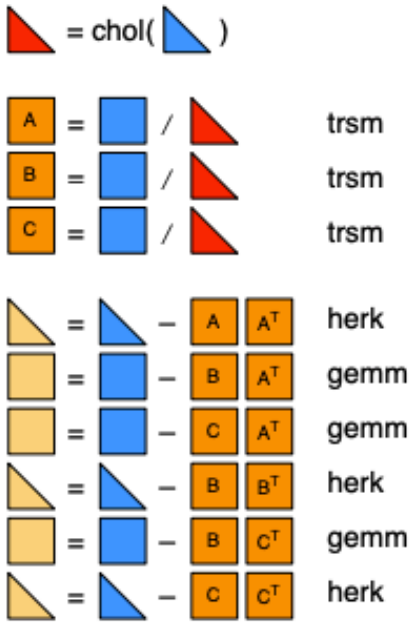
LAPACK Algorithm (right looking)





Tile Algorithm



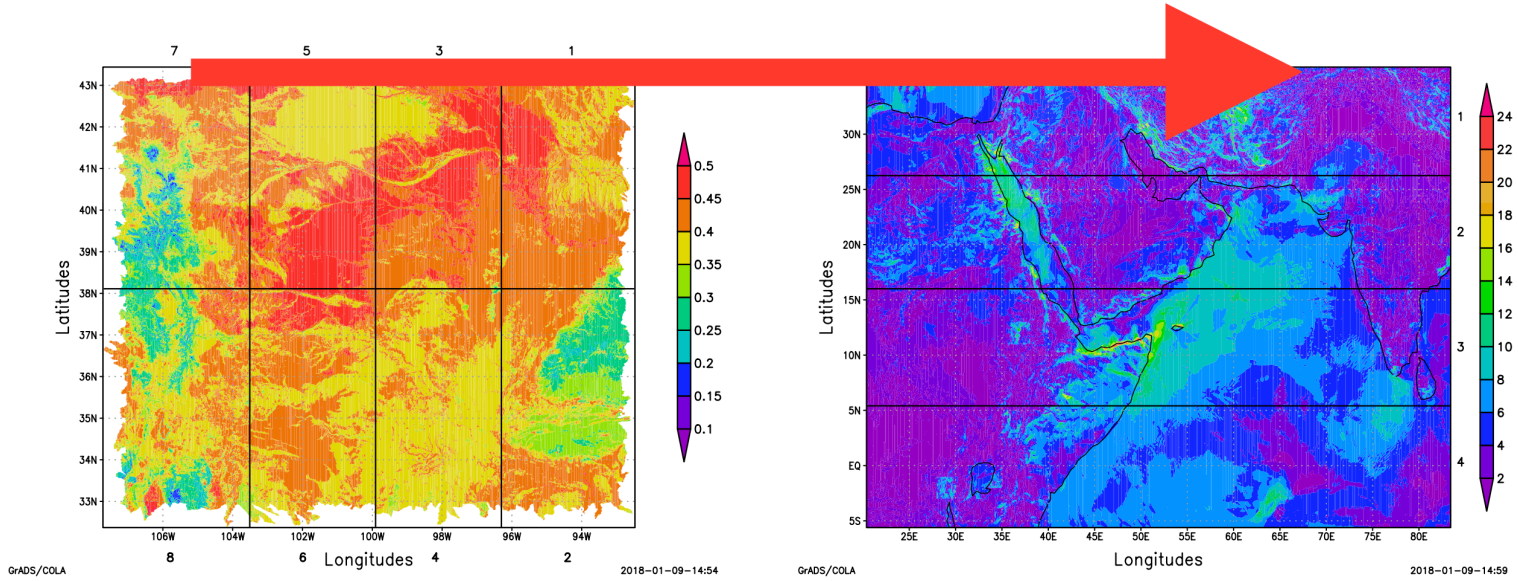


Original slide from Mark Gates Presentation

TLR Cholesky Factorization

- Climate and weather can be predicted statistically via geospatial Maximum Likelihood Estimates (MLE), as an alternative to running large ensembles of forward models:

$$\ell(\boldsymbol{\theta}) = -\frac{1}{2}\mathbf{Z}^T\Sigma^{-1}(\boldsymbol{\theta})\mathbf{Z} - \frac{1}{2}\log|\Sigma(\boldsymbol{\theta})|$$

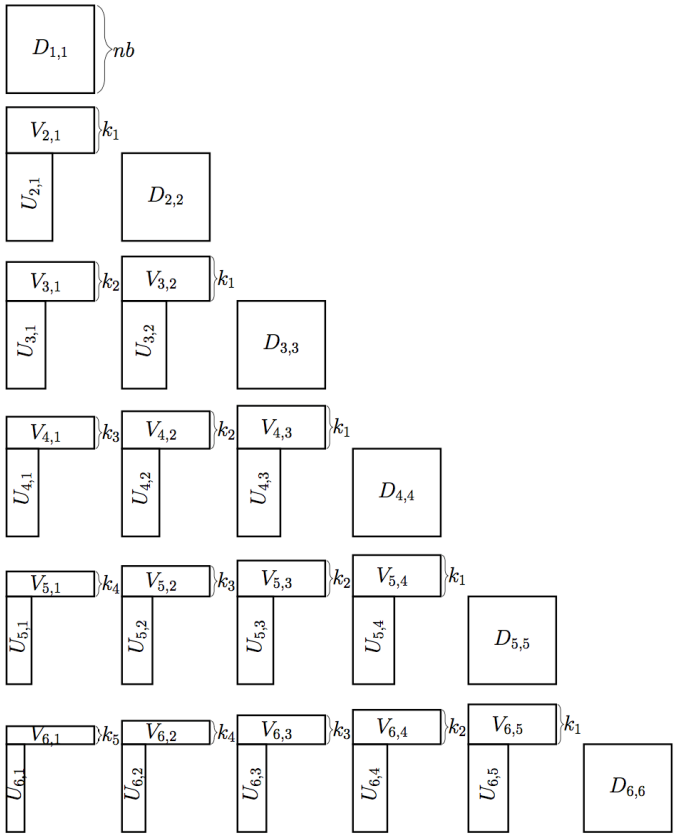


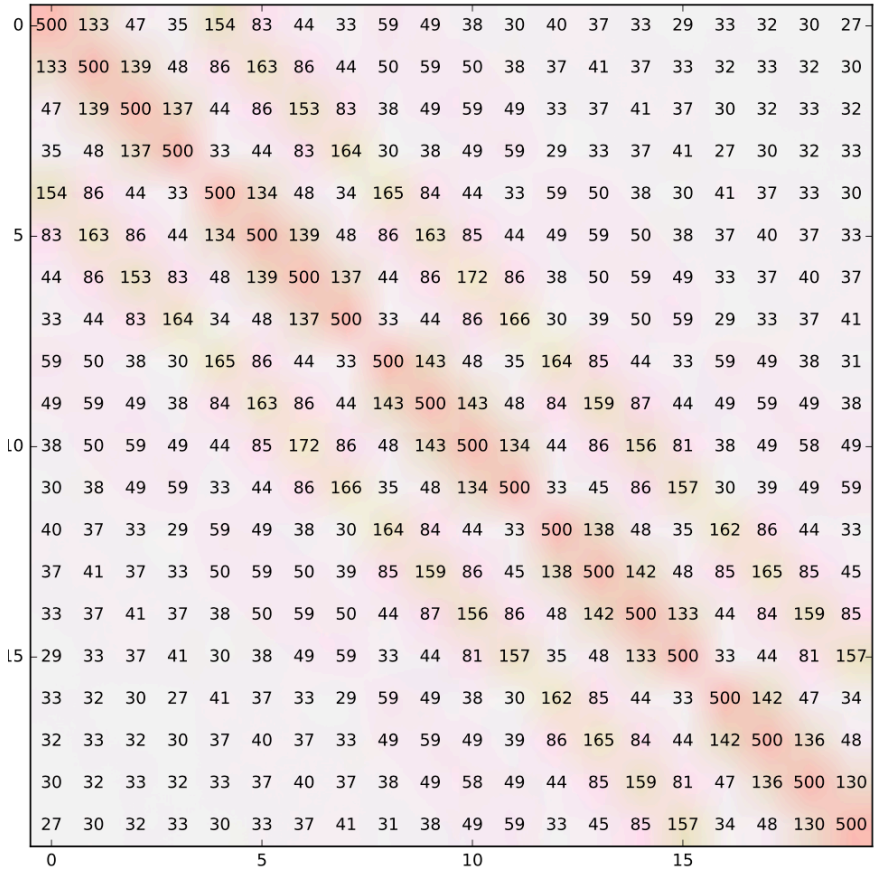
Soil moisture

Wind speed

TLR Format

- TLR format with varying ranks
- Geospatial statistics, matrix size: 20000, tile size: 500, accuracy threshold: 10^{-9}, 2D problem





TLR Cholesky Factorization

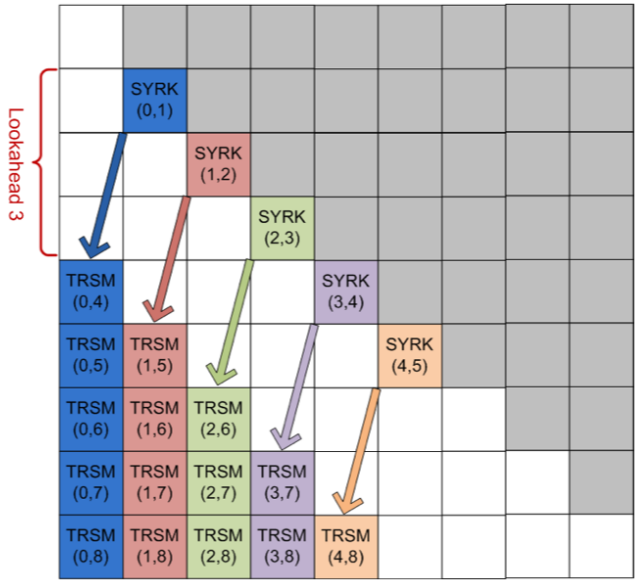
Differences with dense Cholesky

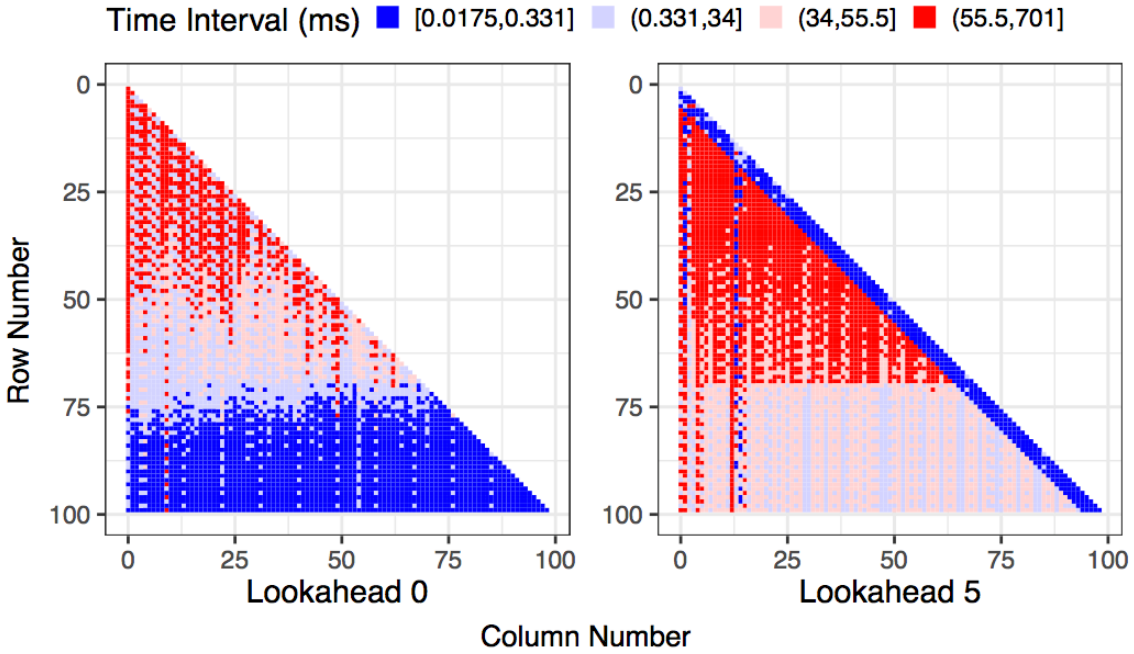
- Data format: only tiles on-diagonal are dense
- tiles off-diagonal are approximated by using a variant of the singular value decomposition (SVD).
- Computational kernels:
  - LR_GEMM kernels with arithmetic complexity = $O(nb * rank^2)$ instead of $O(nb^3)$
- We need to have the work on the diagonal (critical path) to start as soon as possible, since they enables the next round of update, and they are magnitudes more expensive

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Profiling To Aid Performance Analysis

- PaRSEC enables tasks as soon as all their dependencies are available, therefore allows maximum parallelization without the constraint of sequential code visibility, or window size, for task insertion.
- To prioritize tasks on the critical path, a control dependency between SYRK and TRSM of the same panel factorization is used, delaying the discovery of parallelism outside the critical path (corresponding to the update operation).





Time between data is ready and TRSM starts.
Left, without lookahead; right, with lookahead of 5
Each point represents one TRSM; matrix has 100 × 100 tiles

Experiments Settings

1. Enable PaRSEC Tracing and Dot file generation, run TLR Cholesky
2. Write Python/R scripts to extract the data and plotting
3. Use Pandas/NetworkX from Python, and ggplot2 etc from R for plotting (personal preference, matplotlib will work just as well)

Our Example

- We run a test case with 100 by 100 tiles on 9 compute nodes (3 by 3 compute grid)
- Combine the generated dot files and binary traces into one dot file and HDF5 file

```
ypei2@saturn:~/lorapo/profiling$ ls *.dot
l0_270000_2700_9_10-0.dot  l0_270000_2700_9_10-5.dot  l5_270000_2700_9_10-0.dot  l5_270000_2700_9_10-5.dot
l0_270000_2700_9_10-1.dot  l0_270000_2700_9_10-6.dot  l5_270000_2700_9_10-1.dot  l5_270000_2700_9_10-6.dot
l0_270000_2700_9_10-2.dot  l0_270000_2700_9_10-7.dot  l5_270000_2700_9_10-2.dot  l5_270000_2700_9_10-7.dot
l0_270000_2700_9_10-3.dot  l0_270000_2700_9_10-8.dot  l5_270000_2700_9_10-3.dot  l5_270000_2700_9_10-8.dot
l0_270000_2700_9_10-4.dot  l0_270000_2700_9_10.dot    l5_270000_2700_9_10-4.dot  l5_270000_2700_9_10.dot
ypei2@saturn:~/lorapo/profiling$ ls l*_270000_2700_9_*.h5
l0_270000_2700_9_10-lfq-fLkJRC.h5  l5_270000_2700_9_10-lfq-plqJpP.h5
ypei2@saturn:~/lorapo/profiling$
```

Our Example

- A helper Python script then reads in the dot file and store the task graph into a networkX object
- Now we can read in the information in HDF5 and link them both

```
import pandas as pd
import re
import parsec_dag as pdag
import argparse
import sys
```

Load in the H5 and preprocess a little bit

```
t = pd.HDFStore('10_270000_2700_9_10-lfq-fLkJRC.h5')
#t = pd.HDFStore('potrf_prof-20-100000-1000-lfq-KPZa7A.h5')
```

```
t.events = t.events.fillna(-1)
t.events[['tid', 'did', 'tpid', 'node_id', 'id', 'src', 'dst']] = \
    t.events[['tid', 'did', 'tpid', 'node_id', 'id', 'src', 'dst']].astype(int)
```

```
tasks = t.events[t.events.type >= t.event_types['PUT_CB']+1]
tasks.set_index(['taskpool_id', 'type', 'id'], inplace=True) # for normal task, use id field for ID
```

Create the networkx DAG from the generated dot file

```
dag = pdag.ParsecDAG()
dag.load_parsec_dot_files(["10_270000_2700_9_10.dot"])
```

- Based on the tpid, tid and did to link the DAG with Profile Info, so that we can link the network event time with task execution time

```
res = []
get_name = re.compile('([0-9]+),\s*([0-9]+)')
for i in range(100): # All the potrf_dpotrf
    cur_name = "potrf_dpotrf_3_" + str(i)
    an = dag.node_from_name(cur_name)
    cur_task = tasks.loc[(an['tpid'], an['did'], an['tid'])]
    slist = dag.successors_from_id(an['tpid'], an['did'], an['tid']) # successors of the task
    network_events = t.events[(t.events.tid == an['tid'])& \
                              (t.events.type == t.event_types['MPI_DATA_PLD_RCV'])& \
                              (t.events.did == an['did'])]
    for j in range(network_events.shape[0]): # each transfer in the broadcast
        network_event = network_events.iloc[j]
        for n in slist.keys():
            s = dag.node_from_name(n)
            succ_event = tasks.loc[(s['tpid'], s['did'], s['tid'])]
            if int(network_event.node_id) == succ_event.node_id:
                cur_res = [] # a new entry for the task
                cur_res.extend([int(cur_task.node_id), cur_task.end]) # task itself's info
                #print s['param']
                gemm_index = get_name.match(s['param']).groups()
                if gemm_index[1] != '0':
                    gemm_name = "potrf_dgemm_3_" + \
                                gemm_index[0] + "_" + gemm_index[1] + \
                                "_" + str(int(gemm_index[1]) -1)
                    gemm = dag.node_from_name(gemm_name)
                    gemm_event = tasks.loc[(gemm['tpid'], gemm['did'], gemm['tid'])]
                    cur_res.extend([gemm_event.begin, gemm_event.end])
                else:
                    cur_res.extend([9999., 9999.])
                task_node = dag.node_from_name(n)
                cur_res.extend([int(network_event.node_id), network_event.end, task_node['label'], \
                                task_node['param'], int(succ_event.node_id), succ_event.begin, succ_event.end])
                res.append(cur_res)
df = pd.DataFrame(res)
df.to_csv("l5_270000_2700_9_10.csv")
```

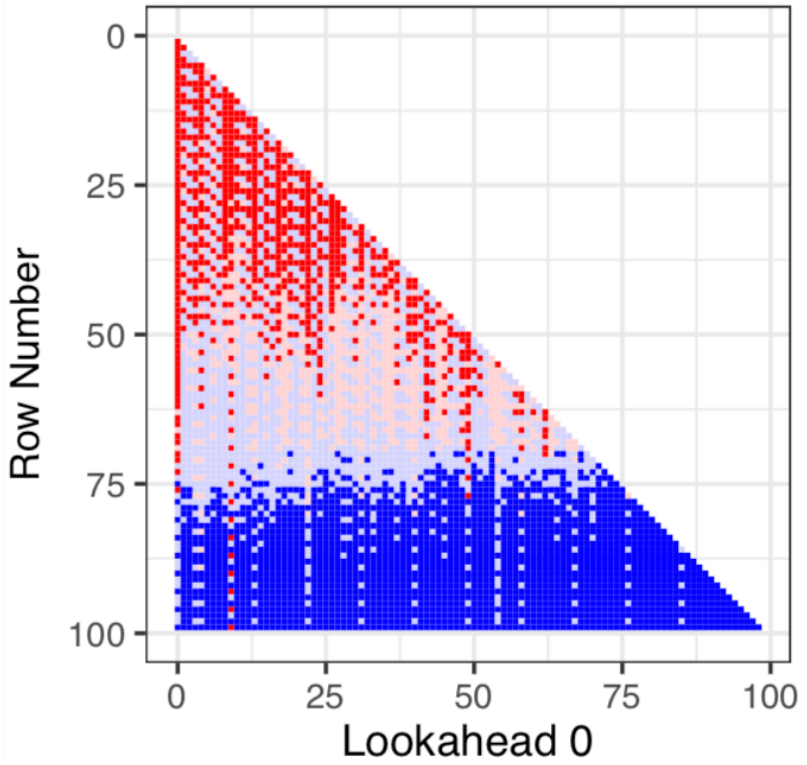
Our Example

- The combined data looks like the following, due to my personal preference, I imported the CSV into R for plotting with ggplot2
- Now it has the node ID of the POTRF, Network Recv and TRSM, timestamps as well. As well as the TRSM Index to identify the tasks

```
> head(dat)
  potrf_node   potrf_end gemm_start gemm_end recv_node    recv_end        trsm trsm_index trsm_node  trsm_start    trsm_end execution
1          0 26651610436       9999     9999         6 26711107701 potrf_dtrsm      95, 0         6 26732960669 26743855236  10894567
2          0 26651610436       9999     9999         6 26711107701 potrf_dtrsm      89, 0         6 26732991019 26743854775  10863756
3          0 26651610436       9999     9999         6 26711107701 potrf_dtrsm      41, 0         6 26842474514 26853329763  10855249
4          0 26651610436       9999     9999         6 26711107701 potrf_dtrsm      68, 0         6 26788312613 26798721599  10408986
5          0 26651610436       9999     9999         6 26711107701 potrf_dtrsm      14, 0         6 26864736731 26877097787  12361056
6          0 26651610436       9999     9999         6 26711107701 potrf_dtrsm      71, 0         6 26787988438 26798721670  10733232
```

- Finally we can generate the profiling plot we shown before

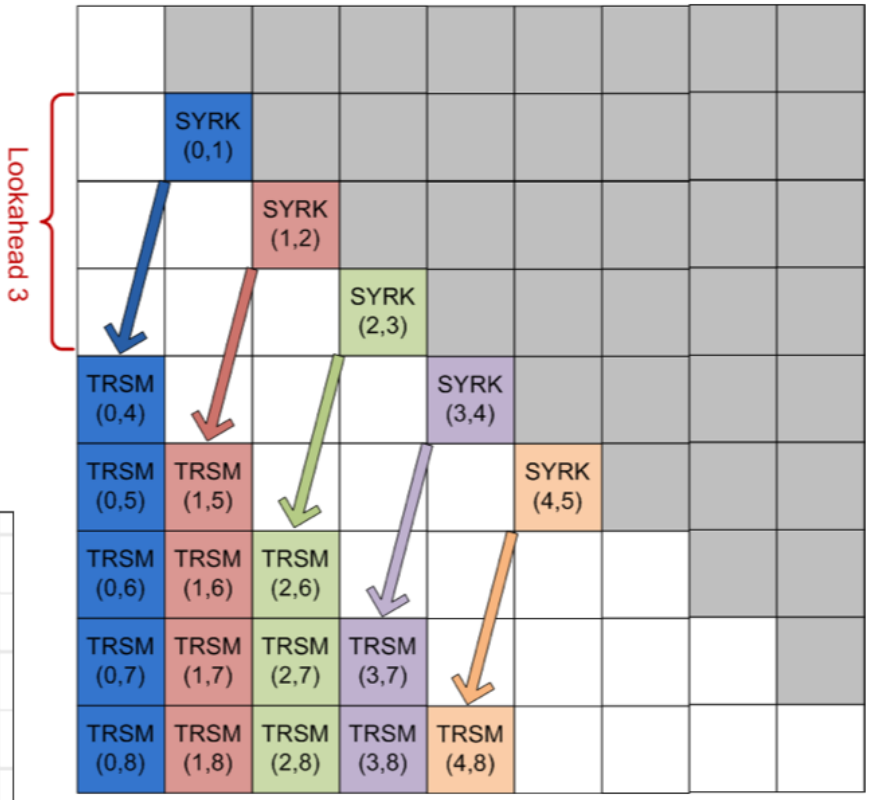
```
library('ggplot2')
library('reshape2')
library('ggpubr')

dat = read.csv('../data/l0_270000_2700_9_10.csv')
dat$X = NULL
names(dat) = c('potrf_node', 'potrf_end', 'gemm_start', 'gemm_end',
               'recv_node', 'recv_end', 'trsm', 'trsm_index', 'trsm_node', 'trsm_start', 'trsm_end')

dat$execution = dat$trsm_end - dat$trsm_start
# If network event, interval of network event to TRSM begin
network = dat[dat$recv_node == dat$trsm_node, ]
# min of GEMM finish to TRSM start and network recv to TRSM start
network$interval = pmin(network$trsm_start - network$gemm_end, network$trsm_start - network$recv_end)
#network$interval = network$trsm_start - network$recv_end

# same node
same = dat[dat$potrf_node == dat$trsm_node, ]
# min of GEMM finish to TRSM start and potrf finish to TRSM start
same$interval = pmin(same$trsm_start - same$gemm_end, same$trsm_start - same$potrf_end)
#same$interval =  same$trsm_start - same$gemm_end

res = rbind(same[, c('trsm_index', 'execution', 'interval')],
            network[, c('trsm_index', 'execution', 'interval')])
res$m = as.integer(gsub("([0-9]+),\\s*([0-9]+)", "\\1", res[,1]))
res$n = as.integer(gsub("([0-9]+),\\s*([0-9]+)", "\\2", res[,1]))
res$interval = res$interval/1e6 # convert to ms
l0 = res
l0$cut = cut(l0$interval, breaks = as.vector(temp),
             include.lowest = TRUE)
ggplot(l0, aes(n, m)) + geom_tile(aes(fill = cut)) +
  scale_y_reverse( lim=c(100,0)) +
  theme_bw(base_size = 24) +
  labs(x = 'Lookahead 0', y = NULL) +
  scale_fill_manual(name="Time Interval (ms)", values = c("#0000FF", "#D5D5FF", "#FFD5D5", "#FF0000"))
```

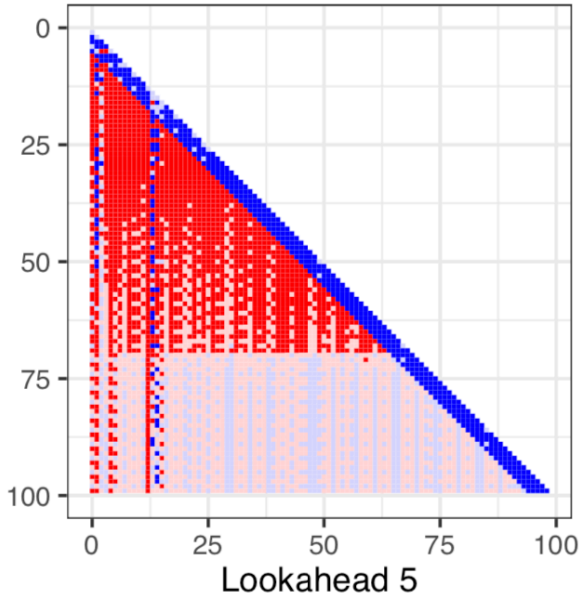


https://colab.research.google.com/drive/1uUlYyV6iBLCtRvMB1Hww_nWrOJMfXszd

New Lookahead

- PaRSEC enables tasks as soon as all their dependencies are available, therefore allows maximum parallelization without the constraint of sequential code visibility, or window size, for task insertion.
- To prioritize tasks on the critical path, a control dependency between SYRK and TRSM of the same panel factorization is used, delaying the discovery of parallelism outside the critical path (corresponding to the update operation).





Conclusion

- Present a use case for the profiling system of PaRSEC: the mechanisms embedded in the runtime system to extract critical information and produce a trace of the execution, and the tools allowing users to manage this collection of events
- Demonstrate the use of Python/R scripts for performance analysis to show optimization of TLR Cholesky

- Take away message: Can easily generate the profiling info from the task execution, enable flexible performance analysis