BBLAS APIs and Memory Layouts

Samuel Relton

 samuel.relton@manchester.ac.uk

 @sdrelton

 samrelton.com

 blog.samrelton.com

Joint work with Mawussi Zounon

 mawussi.zounon@manchester.ac.uk

24th February 2017

Manchester Numerical Analysis

Outline

Aim: Generate discussion on standard API and memory layout for BBLAS.

- Batched linear algebra
- APIs for batched BLAS
  - Available options
  - Comparison
  - Talking points
- Memory layouts
  - Available options
  - Experiments using interleaved layout
  - Talking points

M\crNA
Manchester Numerical Analysis

Batched linear algebra

We are interested in solving thousands of small matrix problems simultaneously. E.g. for GEMM

$$C_i \leftarrow \alpha_i A_i B_i + \beta_i C_i, \qquad i = 1 : \mathit{batch\_count}.$$

There are two main types of batch:

- fixed batch – A_i, B_i, C_i have constant sizes, α_i, β_i constant.
- variable batch – A_i, B_i, C_i have varying sizes, α_i, β_i vary.

Manchester Numerical Analysis

APIs for Batched BLAS

There are 3 main APIs that we will discuss in more detail:

- Flag-based API
- Separate fixed and variable functions
- Group-based API

We use GEMM to show the different APIs in C-code.

Manchester Numerical Analysis

Flag-based API

```
dgemm_batch(
*transA, *transB,
*m, *n, *k,
*alpha, **arrayA, **ldA,
**arrayB, *ldB,
*beta, **arrayC, *ldC,
batch_count, batch_type)
```

- batch_type – enum with value BATCH_FIXED or BATCH_VARIABLE.
- Treating both fixed and variable in one function means m, n, etc. must all be pointers.

Manchester Numerical Analysis

Separate fixed and variable API

```
dgemm_batch_fixed(
transA, transB,
m, n, k,
alpha, **arrayA, ldA,
**arrayB, ldB,
beta, **arrayC, ldC,
batch_count)
```

```
dgemm_batch_variable(
*transA, *transB,
*m, *n, *k,
*alpha, **arrayA, **ldA,
**arrayB, *ldB,
*beta, **arrayC, *ldC,
batch_count)
```

- Two functions for each BLAS operation.
- Fixed batch operations become simpler for user.

M\crNA
Manchester Numerical Analysis

Group-based API

```
dgemm_batch(
*transA, *transB,
*m, *n, *k,
*alpha, **arrayA, **ldA,
**arrayB, *ldB,
*beta, **arrayC, *ldC,
group_count, *group_size)
```

- Group together multiple fixed batches. E.g. 2 groups with $m = \{3, 5\}$, $n = \{3, 2\}$, $group\_size = \{100, 200\}$ etc.
- Complicates simple fixed and variable batches.
- Potential optimizations e.g. combining multiple small matrices of different sizes to fill vector units.
- Currently used in MKL.

M\crNA
Manchester Numerical Analysis

Group-based API experiments

Is the extra complication of the group-based API worthwhile?

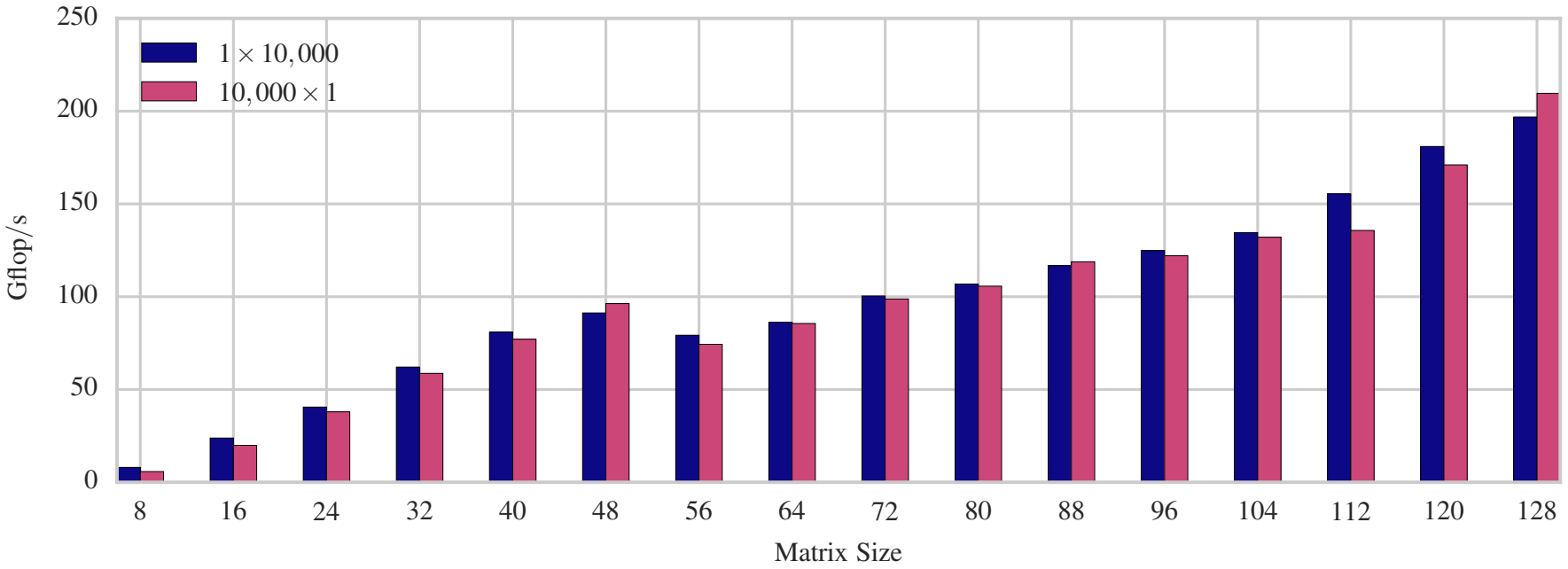
Two experiments:

1. 1 group of 10,000 DGEMM vs 10,000 groups of 1 DGEMM
2. Group-based API vs multiple calls to fixed batch API

Setup:

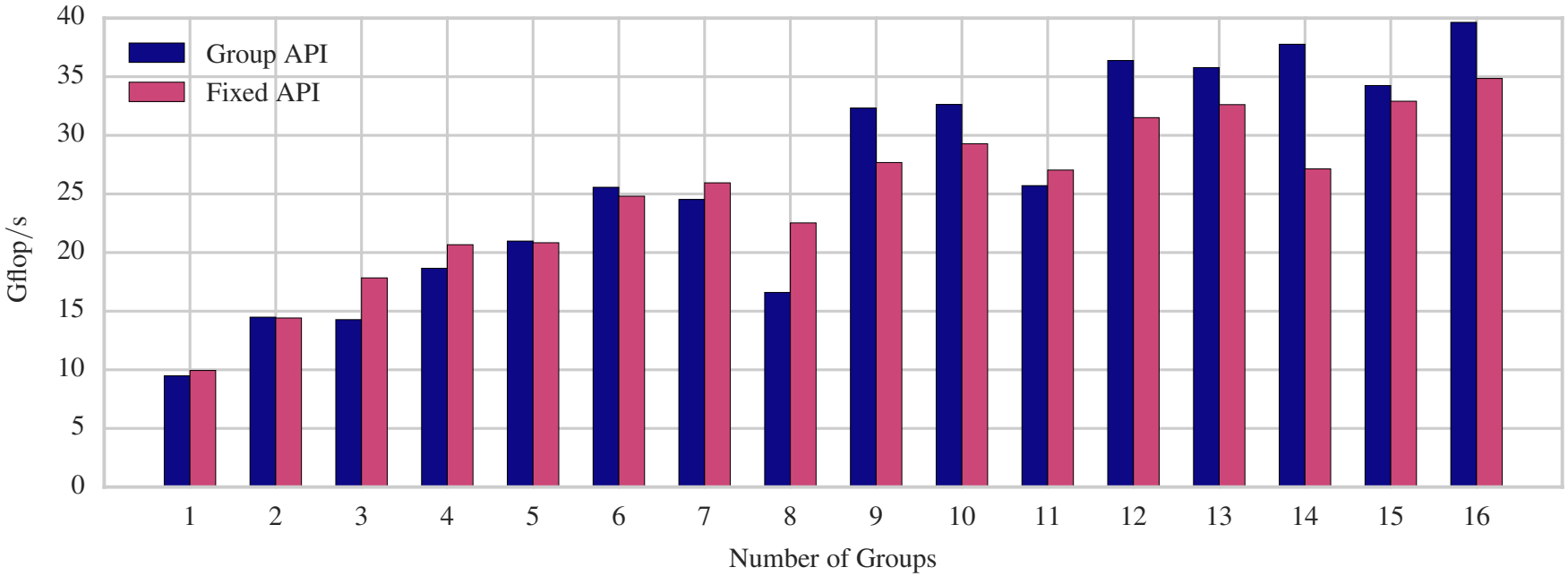
- 20 core NUMA node
- Intel MKL 11.3.3 (not latest)

Manchester Numerical Analysis



- Not a big difference between many or few groups.

M\crNA
Manchester Numerical Analysis



- Each group has different number of matrices with different sizes
- Mixed results, groups better for larger numbers of groups

M\crNA
Manchester Numerical Analysis

Talking points

Here a few ideas for discussion on which API to take as the standard.

- Separate functions for fixed/variable makes both cases simple as possible.
- Need to consider ease of using each API for non-expert BLAS users.
- Group-based API not immediately intuitive but some performance boost. Makes variable batches a little awkward.
- Other potential optimizations of group-based API?
- Addition of info parameter similar to LAPACK?

M\crNA
Manchester Numerical Analysis

Memory layouts

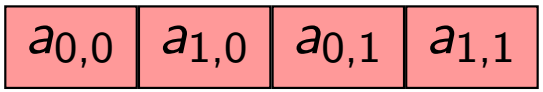
Once an API is chosen we also need to standardize the memory layout.

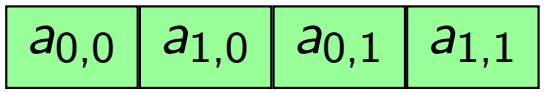
Three options:

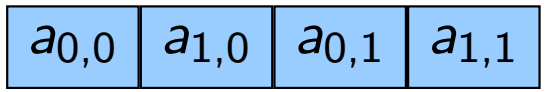
- Pointer-to-pointer (P2P) layout
- Strided layout
- Interleaved layout (fixed batch only)

M\crNA
Manchester Numerical Analysis

P2P layout



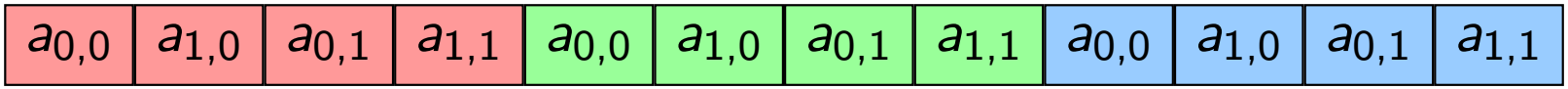




- Matrices spread in RAM
- Very easy for users to understand/create
- Flexibility to add more matrices into the batch
- Not cache friendly when loading matrices into memory

M\crNA
Manchester Numerical Analysis

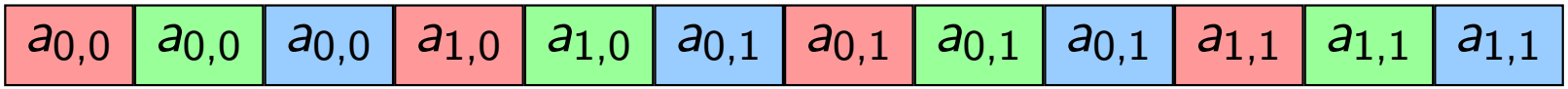
Strided layout



- Matrices grouped in RAM
- Adding more matrices requires reallocating large chunk of memory
- Less prone to cache misses

M\crNA
Manchester Numerical Analysis

Interleaved layout



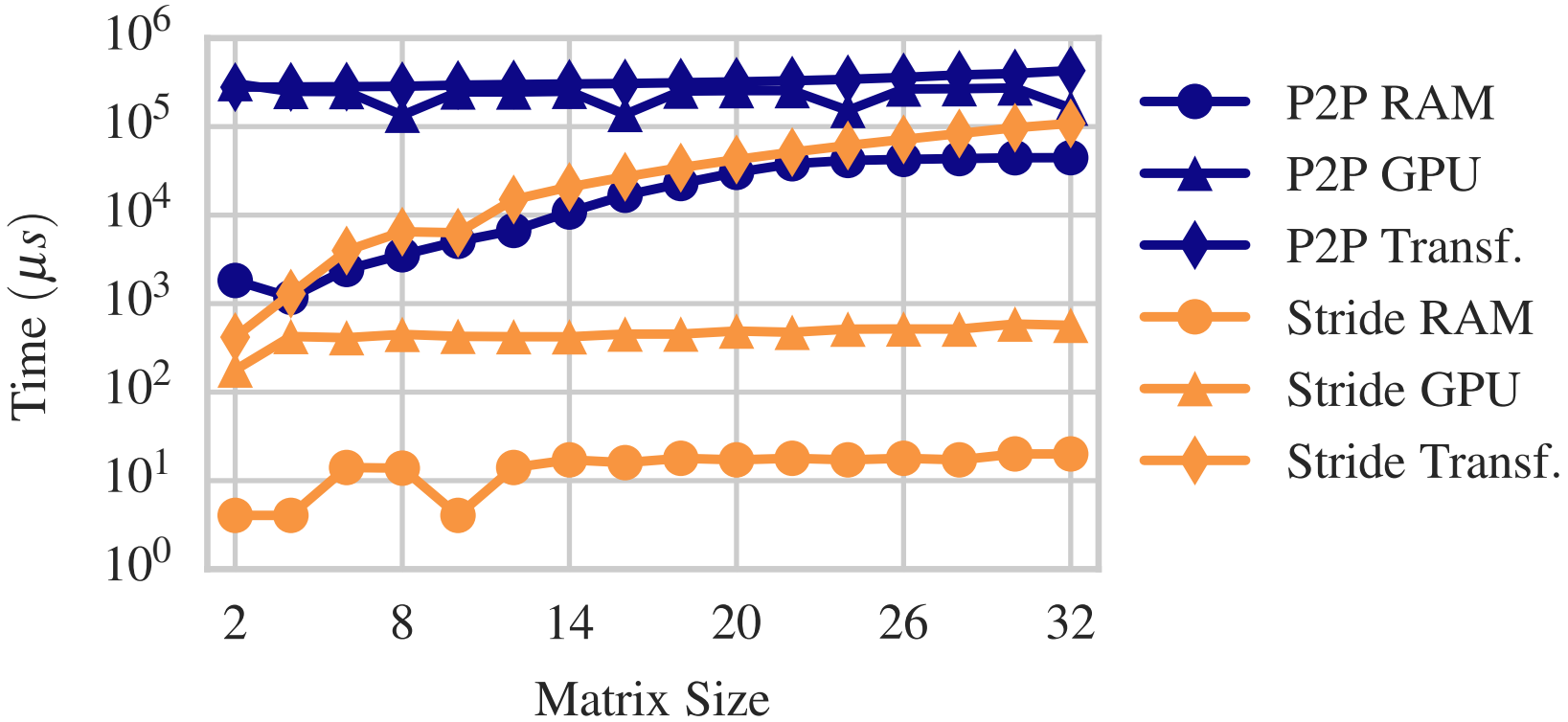
- Matrices grouped in RAM (permutation of strided)
- Difficult for users to create
- Cannot be used for variable batches
- Less prone to cache misses
- Maximizes use of vector units
- Avoids synchronization points in e.g. TRSM
- Block interleaved: Interleave first k matrices then next k etc.
- Intel has proposed similar "Compact BLAS" (Tim Costa).

M\crNA
Manchester Numerical Analysis

Memory transfer experiment

Many users may want to offload computation to a GPU or similar device.

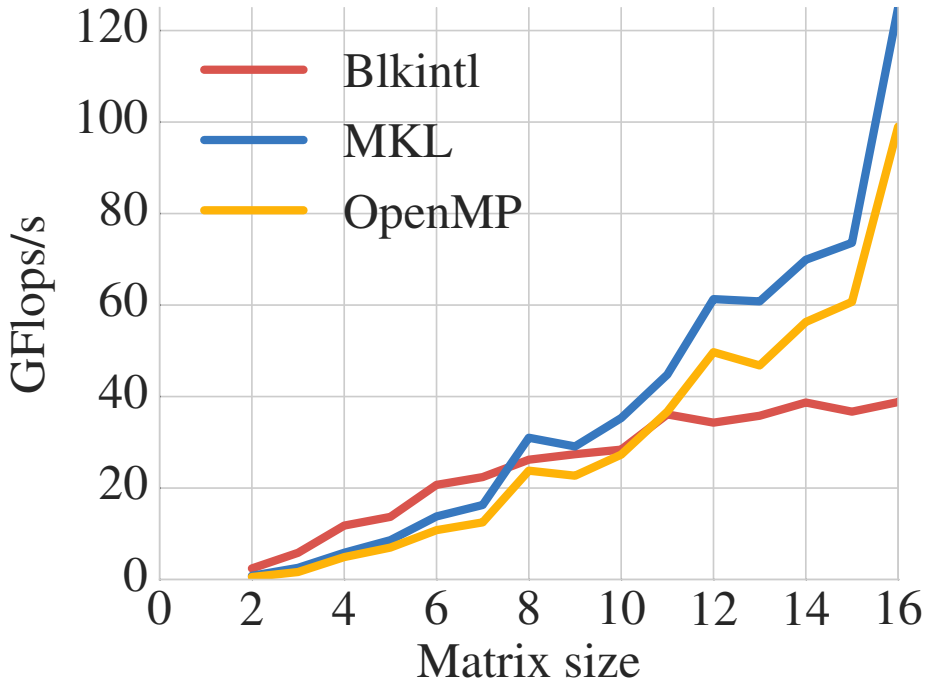
NVIDIA K40c GPU connected to 20 core NUMA node.



- Time to allocate in RAM/GPU and transfer time (batch of 10k).
- Interleaved is simply a permutation of stride.

M\crNA
Manchester Numerical Analysis

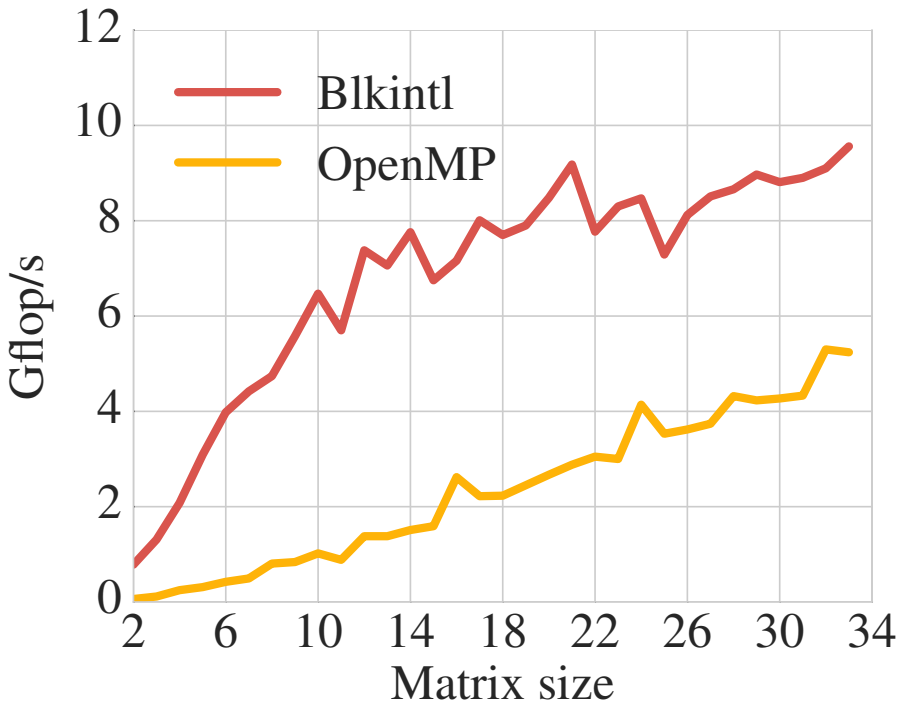
Interleaved memory layout - GEMM (Intel KNL, 10k batch)



- Includes time to convert memory (P2P to interleaved and back)
- Difficult to beat MKL optimized GEMM

Manchester Numerical Analysis

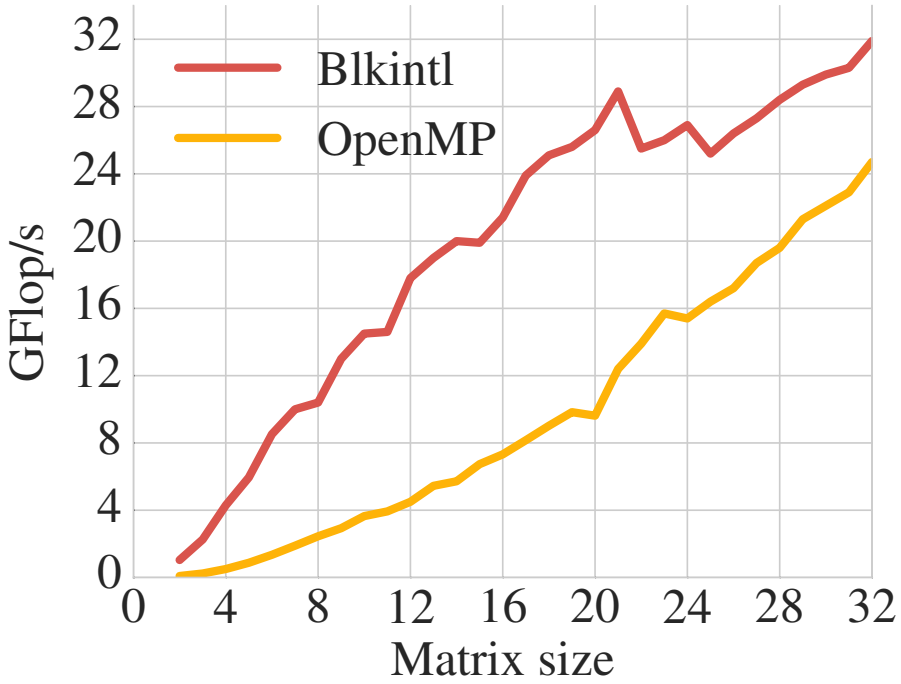
Interleaved memory layout - TRSM (Intel KNL, 10k batch)



- Block interleaved avoids waiting for division in each column
- Using 4 right-hand sides here.

Manchester Numerical Analysis

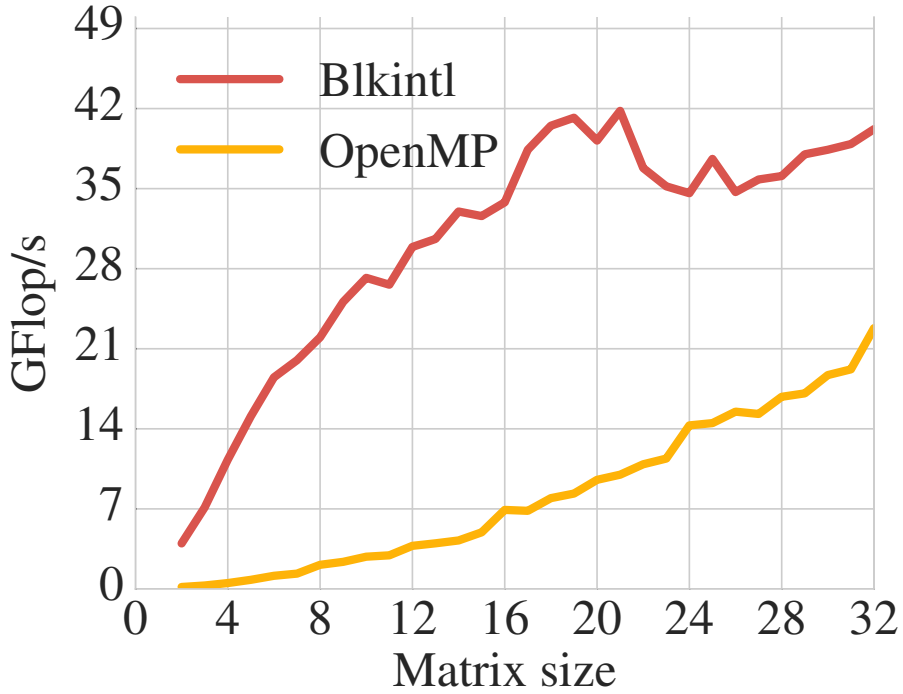
Interleaved memory layout - DPOTRF (Intel KNL, 10k batch)



- As matrix size continues to grow OpenMP kernel eventually overtakes: our custom dpotrf kernel is less efficient than MKLs for large problems.

Manchester Numerical Analysis

Interleaved memory layout - DPOTRS (Intel KNL, 10k batch)



- 4 right-hand sides used here

Talking points

- User facing and internal data layouts can be different!
- P2P layout very bad for offloading computation.
- Interleaved formats are very complicated for users.
- P2P layout is most the flexible (easy to add new matrices etc).

M\crNA
Manchester Numerical Analysis