Science & Technology Facilities Council

Do we need to support alternative data formats for Batched BLAS?

Jonathan Hogg

STFC Rutherford Appleton Laboratory

May 18, 2016

WARNING

- I've not implemented this stuff personally
- Slightly outside my area of expertise
- Please shout up if you think I'm wrong!

Starting Point #1

What is the point of B-BLAS if I can do this?

```
#pragma omp parallel do
for(int i=0; i<n; ++i)
   dgemm(...);
```

Starting Point #1

What is the point of B-BLAS if I can do this?

```
#pragma omp parallel do
for(int i=0; i<n; ++i)
   dgemm(...);
```

Get rid of overheads?

- Only important for small matrices!

Starting Point #2

Sparse Cholesky

- Not uncommon to have 100s-10,000s small matrices
- Can be treated independently
- Sizes often in range 2×1 to 250×8.
- Perform partial Cholesky or similar

Small matrices

At these sizes keeping FMA port fed is the hard part.

Four enemies:

- Special op throughput (sqrt, div)
  - Haswell VDIVPD, VSQRTPD 16+ clocks/op
  - Steamroller VDIVPD, VSQRTPD 9+ clocks/op
- Instruction latency / number of avx registers
  - Haswell VFMADDPD 5 clocks; 16 registers
  - Steamroller VFMADDPD 5-6 clocks; 16 registers
- Memory latency
  - Haswell L1=4 cycles, L2=12 cycles, L3=36 cycles, RAM=L3+57ns $\approx$ 170 cycles
- Memory bandwidth
  - Haswell per cache line L1=0.5 cycles, L2=2.2 cycles, L3=4.7 cycles, RAM $\approx$ 45 cycles

Science & Technology Facilities Council

Focus on: memory

Only 16 AVX registers

- Can't have that many operations in flight
- Need most of these registers just to hide instruction latency?
- But L1 cache can (almost) keep up
  - Bandwidth $\Rightarrow$ 2 loads/cycle
  - 2 FMA ports/cycle
- L2, L3 and main memory can't keep up.
- Need to work with L1-size batches for implement batched Cholesky etc. on top?

Avoid wasted memory loads

Padding is a poor solution

We could just pad with zeroes to multiple of vector size.

AVX vector length 8 SP or 4 DP

AVX512 vector length 16 SP or 8 DP

Cuda warpSize 32 SP or 32 DP (but loads more flexible?)

DP (useful loads) / (total loads):

m	1	2	3	4	5	6	7	8	129	257
AVX	0.25	0.50	0.75	1.00	0.63	0.75	0.88	1.00	0.98	0.99
AVX512	0.13	0.25	0.38	0.50	0.63	0.75	0.88	1.00	0.95	0.97
CUDA	0.03	0.06	0.09	0.13	0.16	0.19	0.22	0.25	0.81	0.89

SP (useful loads) / (total loads):

m	1	2	3	4	5	6	7	8	129	257
AVX	0.13	0.25	0.38	0.50	0.63	0.75	0.88	1.00	0.95	0.97
AVX512	0.07	0.13	0.19	0.25	0.31	0.38	0.44	0.50	0.90	0.94
CUDA	0.03	0.06	0.09	0.13	0.16	0.19	0.22	0.25	0.81	0.89

FP16 even worse!

Padding is a poor solution

We could just pad with zeroes to multiple of vector size.

AVX vector length 8 SP or 4 DP

AVX512 vector length 16 SP or 8 DP

Cuda warpSize 32 SP or 32 DP (but loads more flexible?)

DP (useful loads) / (total loads):

m	1	2	3	4	5	6	7	8	129	257
AVX	0.25	0.50	0.75	1.00	0.63	0.75	0.88	1.00	0.98	0.99
AVX512	0.13	0.25	0.38	0.50	0.63	0.75	0.88	1.00	0.95	0.97
CUDA	0.03	0.06	0.09	0.13	0.16	0.19	0.22	0.25	0.81	0.89

SP (useful loads) / (total loads):

m	1	2	3	4	5	6	7	8	129	257
AVX	0.13	0.25	0.38	0.50	0.63	0.75	0.88	1.00	0.95	0.97
AVX512	0.07	0.13	0.19	0.25	0.31	0.38	0.44	0.50	0.90	0.94
CUDA	0.03	0.06	0.09	0.13	0.16	0.19	0.22	0.25	0.81	0.89

FP16 even worse!

Generally want m $>$ 2.5 $\times$ vector length

The interleaved solution

- If we have vector length operations of same type/size

4 matrices of size 2×2:

$$a_{00}^{(0)} a_{00}^{(1)} a_{00}^{(2)} a_{00}^{(3)} a_{10}^{(0)} a_{10}^{(1)} a_{10}^{(2)} a_{10}^{(3)} a_{01}^{(0)} a_{01}^{(1)} a_{01}^{(2)} a_{01}^{(3)} a_{11}^{(0)} a_{11}^{(1)} a_{11}^{(2)} a_{11}^{(3)}$$

The interleaved solution

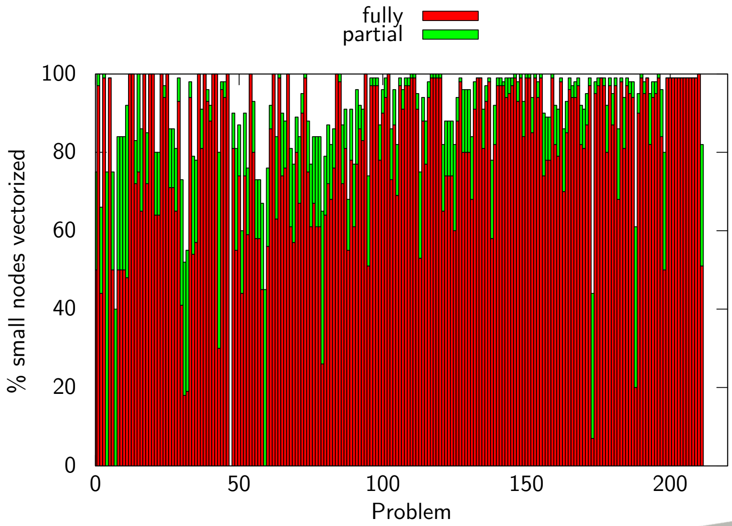
- If we have vector length operations of same type/size

4 matrices of size 2×2:

$$a_{00}^{(0)} a_{00}^{(1)} a_{00}^{(2)} a_{00}^{(3)} a_{10}^{(0)} a_{10}^{(1)} a_{10}^{(2)} a_{10}^{(3)} a_{01}^{(0)} a_{01}^{(1)} a_{01}^{(2)} a_{01}^{(3)} a_{11}^{(0)} a_{11}^{(1)} a_{11}^{(2)} a_{11}^{(3)}$$

- Easy to load the a_{ij}-th entry of vec-len matrices in one go
- All oeprations with different vector operations are independent
- No need for horizontal reductions
- No wasted loads (if multiple of vector size in batch)

Sparse Cholesky: can we use it?



Transform to interleaved?

Just use a more complicated load...

Data set size	VMOVAP	_mm256_set_pd()	VGATHER
234 KB	[2.874, 3.271]	[4.143, 4.869]	[10.333, 11.663]
7.64 MB	[3.732, 3.954]	[4.243, 4.934]	[10.326, 10.995]
764 MB	[7.607, 9.107]	[11.280, 14.217]	[10.620, 10.644]

- Overhead $>$40% on L2, $>$10% on L3
- New VGATHER only worthwhile from Main memory?

Special function units?

Do we need to consolidate DIV, SQRT to be vectorized?

Single TRSM solve $XA = B$, A is $n \times n$ and X, B are $m \times n$:

- At least n DIVs
- At least mn FMAs

$\Rightarrow$ DTRSM bound on scalar DIV throughput if $m \leq 64$.
(64 = 16 clocks/DIV * veclen 4 for FMA)

$\Rightarrow$ Reduced to $m \leq 16$ if we can use vector DIV.

Special function units?

Do we need to consolidate DIV, SQRT to be vectorized?

Single TRSM solve $XA = B$, A is $n \times n$ and X, B are $m \times n$:

- At least n DIVs
- At least mn FMAs

$\Rightarrow$ DTRSM bound on scalar DIV throughput if $m \leq 64$.
(64 = 16 clocks/DIV * veclen 4 for FMA)

$\Rightarrow$ Reduced to $m \leq 16$ if we can use vector DIV.

- For TRSM can get this within single matrix - but not for Cholesky (RSQRT is on critical path + also need a DIV per column).
- Can do this without interleaving - but fiddly!
- Interleaved data makes this trivial.
- But we might be bound on memory loads anyway?

Science & Technology Facilities Council

Why not a separate “microblas”?

- This might be a good idea implementation-wise
- But we’re discussing an API specification
- Balance: extra implementation cost vs benefit

If we don’t support very small matrices what’s the point?

- Clearly batched BLAS are useful on GPUs:
  - Can’t write own fast code - need access to physical register allocation that isn’t available without going below PTX.
  - Kernel launch overheads.
- Less useful on CPUs if we do the trivial:
  - Running multiple matrices with OpenMP not hard.
  - DIY is better with existing parallel schemes (eg tasks).
- On CPUs need to deliver a benefit on a single core level
  - Only run argument checking overhead once (or not at all!)
  - Use multiple matrices to hide memory and instruction latencies
  - Above advantages only really significant for very small matrices?

Science & Technology Facilities Council

Support for memory alignment

Without memory alignment:

- Need “top” and “tail” loops (handle unaligned parts)
- These can be large overhead on small matrices

With memory alignment:

- Users would need to promise aligned vectors
- Leading dimension multiple of vector size
- To avoid tail loop, need to zero out unwanted part
- Users can often meet these conditions cheaply!
- Probably faster even without explicit exploitation

Notes:

- No native support for memory alignment in Fortran :(

Science & Technology Facilities Council

Suggested Changes

Change: support memory alignment

- Add “aligned” flag. User promises:
  1. First element of any matrix is aligned
  2. `lda` is multiple of vector length
- Easily ignored by vendors if no desire to implement
- Easy win?

Suggested Changes

Change: support memory alignment

- Add “aligned” flag. User promises:
  1. First element of any matrix is aligned
  2. `lda` is multiple of vector length
- Easily ignored by vendors if no desire to implement
- Easy win?

Change: support interleaved data format

- Either “interleaved” flag; or
- Add an “`lda`-like” variable for the interleaving.
  - Easy to detect when this is 1 and run traditional code.
- Extra work for implementors
- Lack may drive people to implement own code instead
- Need to assess cost/benefit?

Science & Technology Facilities Council

One more thing...

Can we tack a 2d sparse scatter on the end of _gemm?

- i.e. provide arrays rlist, clist and do $c(rlist(i), clist(j)) += \sum_k a(i,k) * b(k,j)$
- Avoid C falling out of cache
- Hide indirection latency behind arithmetic
- Essential to sparse direct solvers

Discuss.

Aligned vs Unaligned Loads

Data set size	VMOVAP	VMOVUP	
		aligned	unaligned
234 KB	[2.897, 3.462]	[2.886, 3.427]	[3.208, 3.827]
7.64 MB	[3.707, 3.968]	[3.659, 3.946]	[3.991, 4.288]
764 MB	[7.967, 7.979]	[7.956, 7.969]	[8.078, 8.096]

[mean-1sd, mean+1sd]$\times 10^{-10}$ per load

$\Rightarrow$ Alignment of data important, not instruction?

Aligned vs Unaligned Loads

Data set size	VMOVAP	VMOVUP	
		aligned	unaligned
234 KB	[2.897, 3.462]	[2.886, 3.427]	[3.208, 3.827]
7.64 MB	[3.707, 3.968]	[3.659, 3.946]	[3.991, 4.288]
764 MB	[7.967, 7.979]	[7.956, 7.969]	[8.078, 8.096]

[mean-1sd, mean+1sd]$\times 10^{-10}$ per load

$\Rightarrow$ Alignment of data important, not instruction?

BUT: Alignment allows direct load in eg VFMADDPD.

	VADDPD
	w mem operand
234 KB	[6.607, 7.788]
7.64 MB	[6.849, 7.349]
764 MB	[7.019, 7.153]

Summary of results

- Aligned vs unaligned load instructions **does not** matter too much.
- Alignment of memory **does**: 5–15% penalty.
- Transforming to interleaved in registers: 40% overhead.
- Padding bad solution for small matrices: Need $m \geq 2.5v_{len}$ to be at least 75% efficient. Length 32 vector $\Rightarrow m \geq 72$

Science & Technology Facilities Council