

NVPL BLAS Overview Evarist Fomenko | BLIS Retreat. Sep 26-27, 2024





Agenda

- Overview
- Library Architecture
- Optimizations

GEMM u-kernel

Synchronization Overheads

Framework Overheads

Conclusions



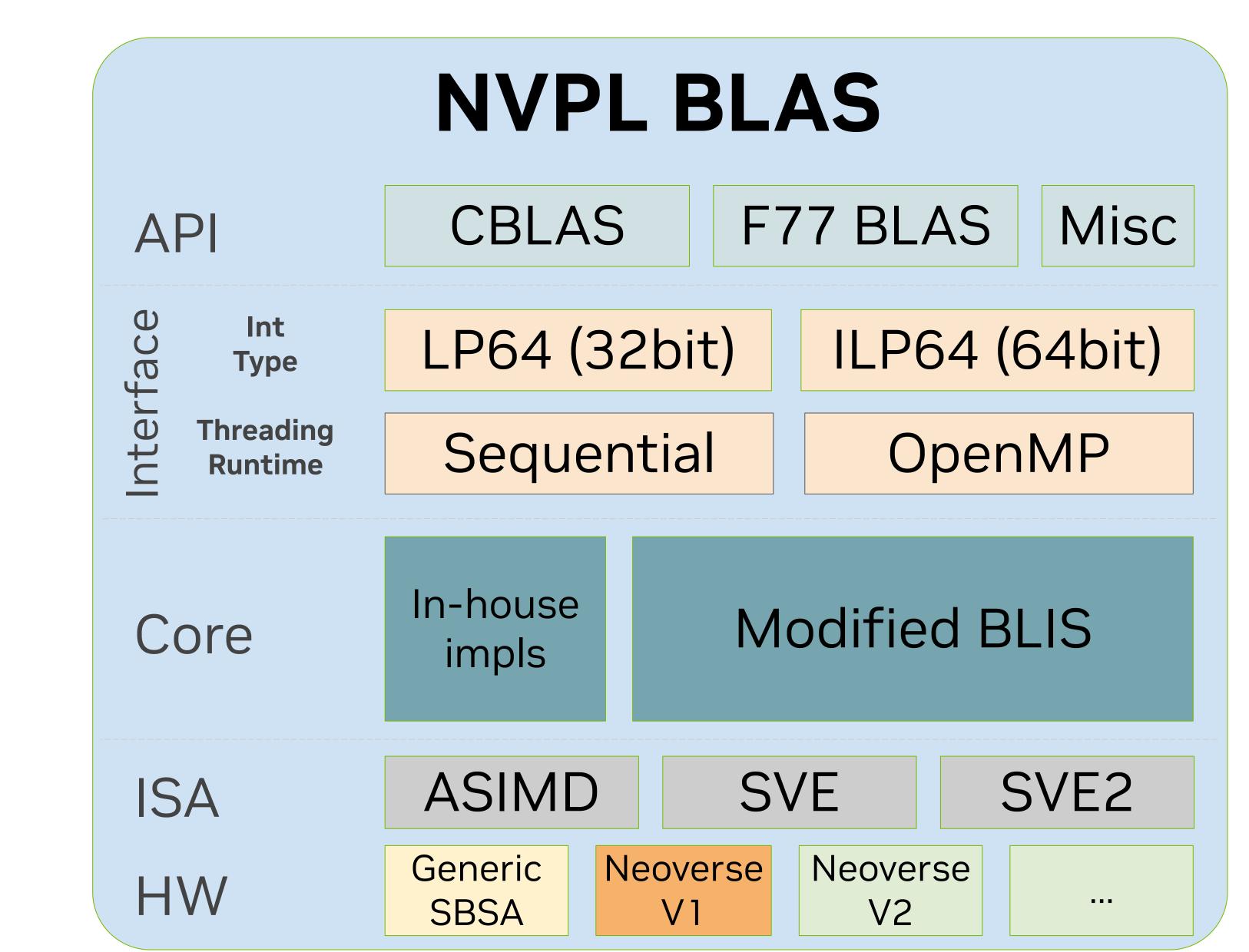


• Part of Nvidia Performance Libraries

- Initial release NVPL 23.11 (Nov'23)
- <u>Latest</u> release NVPL 24.7 (Jul'24)
- API
 - Standard Netlib API: F77 and C
 - Extra: batch GEMM, threading control, verbosity, ...
- Interface
 - LP64 and ILP64 interfaces
 - Switched with -DNVPL_ILP64 compiler flag
 - Different thread runtime: sequential and OpenMP (any vendor)
- Implementation
 - Runtime dispatching based on CPU architecture
- Inspiration: BLIS, Intel MKL, ArmPL

```
NVPL_BLAS_VERBOSE: NVPL BLAS version 0.1.0
```

Overview



NVPL_BLAS_VERBOSE: Platform: Neoverse V2, cores:72 sve_width:128. Cache: L1:64 KB (cl:64 ways:4 sets:256) L2:1024 KB NVPL_BLAS_VERBOSE: dgemm_(N,N,128,256,123,2,0xfffdddb62000,128,0xfffdddb12000,123,-1,0xfffdddac2000,128) time_us:4742 NVPL_BLAS_VERBOSE: cscal_(1024,0xaaab41ed69e0,0xaaab41ee5000,1) time_us:5.76 int:lp64 max_nthr:72 tid:fffdddeb0020



Name: libnvpl_blas_{,i}lp64_{seq,gomp}.so Symbol mangling: nvpl_blas_* (except for BLAS) Responsibilities:

- Int32 / Int64 APIs
 - Switched with -DNVPL_ILP64
- Threading RT
- Verbose

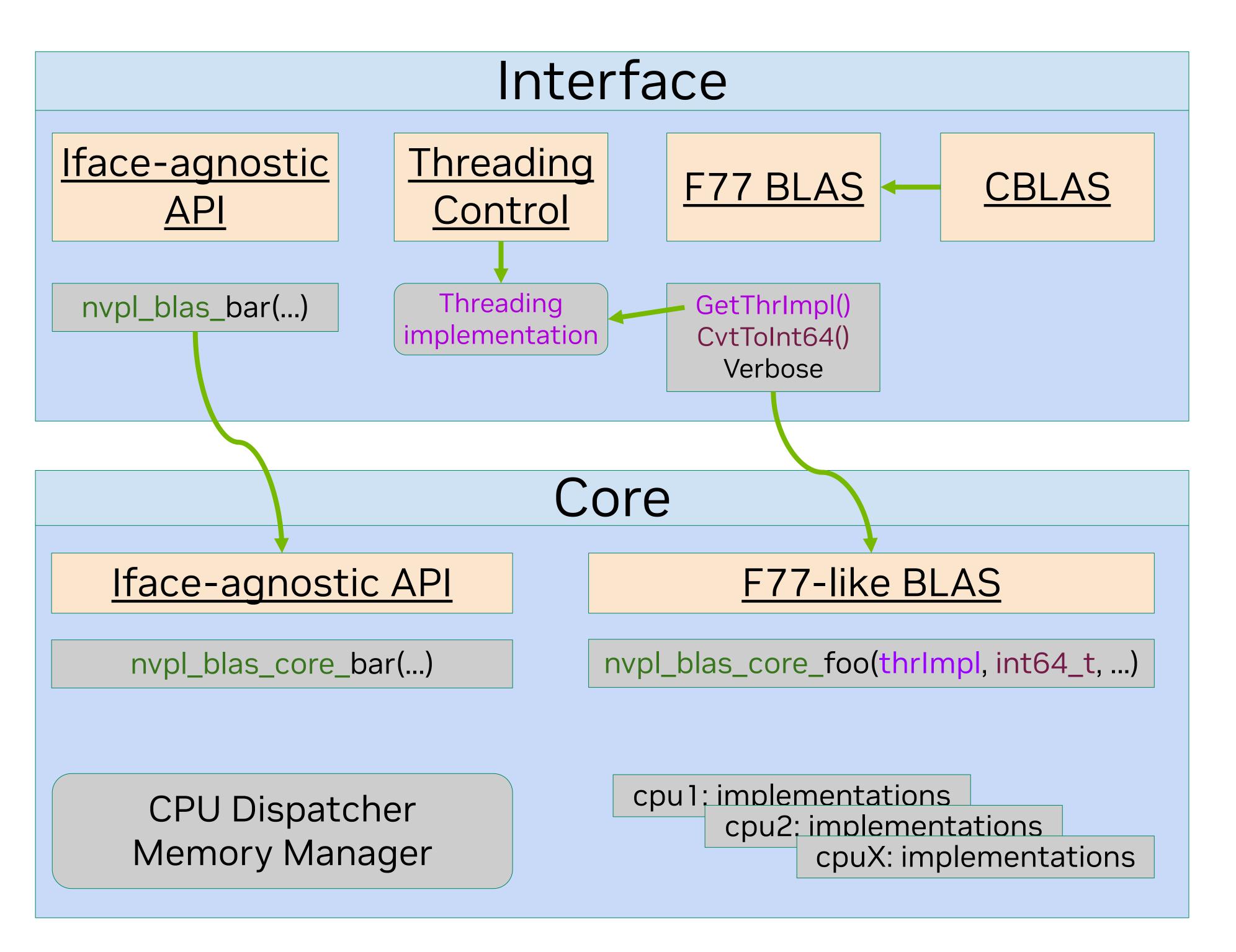
Depends: core, threading rt

Name: libnvpl_blas_core.so Symbol mangling: nvpl_blas_core_* Responsibilities:

- Actual BLAS implementations (all int64)
- CPU dispatcher
- Memory manager
- Other service functionality

Depends: libc, lpthread, lm

Library Architecture



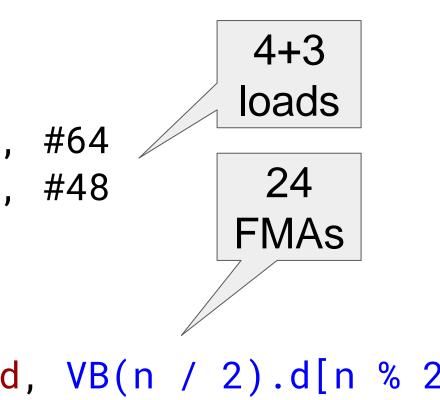


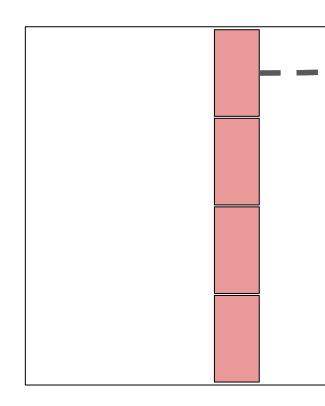
- Very similar to a typical Armv8-A kernel, e.g. Cortex A57
- μ-kernels are written in assembler
- Most gains come from proper memory prefetches
 - Upside is around 5-10%

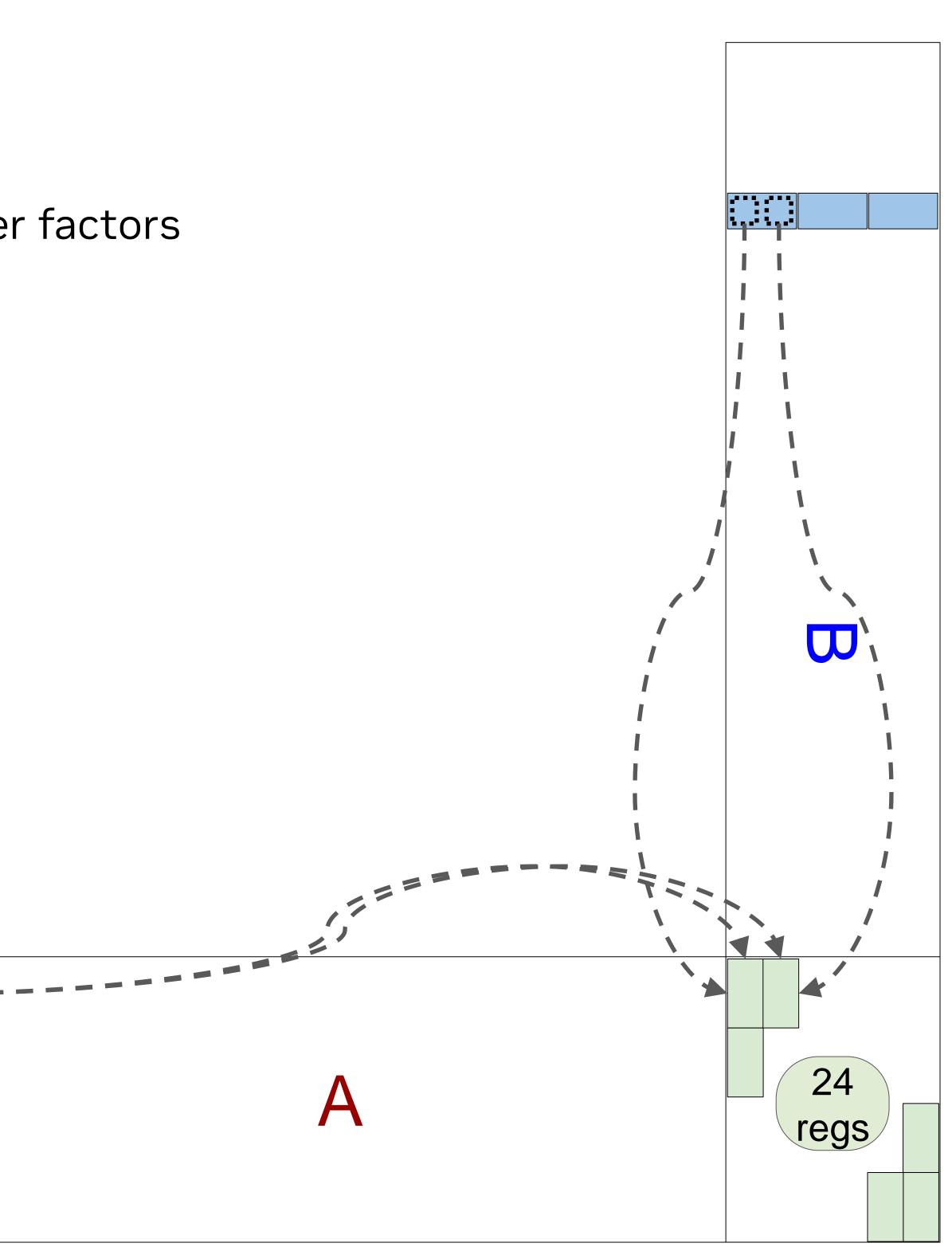
```
# MR = 8
            (alternative: 6)
            (alternative: 8)
# NR = 6
# KC = 512
µdgemm_MRxNRxKC():
    VC(0:6, 0:4) = 0
    .L_loop_k
    for uk = 0 .. 4:
        LD1 \{VA(0).2d - VA(3).2d\}, [ptrA], #64
        LD1 {VB(0).2d - VB(2).2d}, [ptrB], #48
        for n = 0 .. 6: # NR
            for m = 0 ... 4: \# MR/2
                FMLA VC(n, m).2d, VA(m).2d, VB(n / 2).d[n % 2]
    SUB xk, xk, 4
   JNZ xk, .L_loop_k
   C_{r}[:, :] = alpha \times VC(:, :) + beta \times C_{r}[:, :]
```

Optimizations Grace µ-kernel

• Statically generated using Python to easily adjust blocking, data types, and other factors









GE GE	MM_s -	1.04	1.06	1.00	1.04	1.05	- 1.4
	MM_d -	1.05	1.06	1.05	1.10	1.07	
GE GE	MM_c -	1.02	1.06	1.01	1.03	1.02	- 1.2
GE GE GE NON-GE	MM_z -	1.04	1.05	1.06	1.06	1.04	1.0
	MM_s -	1.02	1.06	1.02	1.06	1.07	- 1.0
Per Non-Ge	MM_d -	1.04	1.07	1.04	1.12	1.12	- 0.8
ମ୍ମ NON-GE ଦ୍ୟୁ	MM_c -	1.02	1.07	1.05	1.06	1.06	
ם NON-GE	EMM_z -	1.05	1.09	1.09	1.11	1.10	- 0.6
		1	8 Nu	16 mber of threa	64 ads	72	

Optimizations Grace µ-kernel

Geomean speed-up NVPL BLAS 0.0.4 + Grace uker vs NVPL BLAS 0.0.4 NVIDIA Grace CPU Superchip

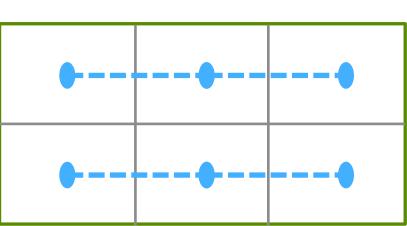


- Asynchronous creation of thread info (thrinfo_t) tree
- Faster broadcast
 - Avoid 1 of 2 barriers by altering the shared space for sent object
- Replace 1 barrier for N*M threads with N independent barriers for M threads • At matrix B packing, as there will be M barriers for N threads at matrix A packing
- Replace BLIS allocator with glibc malloc
 - Consider improving/reimplementing the allocator or using something like tcmalloc/jemalloc/etc.
- Some implementation rework to avoid redundant barriers

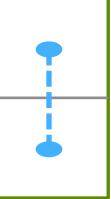
```
Execution
13_int() \rightarrow gemm_blk_var2() \rightarrow
13_int() \rightarrow gemm_blk_var3() \rightarrow
13_int() \rightarrow [barrier()] \rightarrow 13_pack_b() \rightarrow [barrier()] \rightarrow 
13_int() \rightarrow gemm_blk_var1() \rightarrow
13_int() \rightarrow [barrier()] \rightarrow 13_pack_a() \rightarrow [barrier()] \rightarrow 
13_int() \rightarrow gemm_ker_var2() \rightarrow
\rightarrow
µgemm()
```

Optimizations **Reducing Synchronization Overheads**











type)

data

ding

clu

. Ú

0

m

S

<1

- GEMM s -
- GEMM d -
- GEMM c -
- GEMM_z -
- NON-GEMM s -
- Flav NON-GEMM d -

 - NON-GEMM c -
- В NON-GEMM z -

Optimizations Reducing Synchronization Overheads

Geomean speed-up NVPL BLAS 0.0.4 + Grace uker + less barrier vs NVPL BLAS 0.0.4 + Grace uker NVIDIA Grace CPU Superchip

0.99	1.12	1.17	1.13	1.18	- 1.4		
1.00	1.09	1.13	1.11	1.13			
1.00	1.07	1.13	1.07	1.09	- 1.2		
1.00	1.05	1.09	1.05	1.04	1.0		
1.00	1.09	1.15	1.14	1.16	- 1.0		
1.00	1.07	1.12	1.10	1.12	- 0.8		
1.00	1.06	1.11	1.07	1.09			
1.00	1.04	1.08	1.05	1.05	- 0.6		
່ 8 16 64 72 Number of threads							

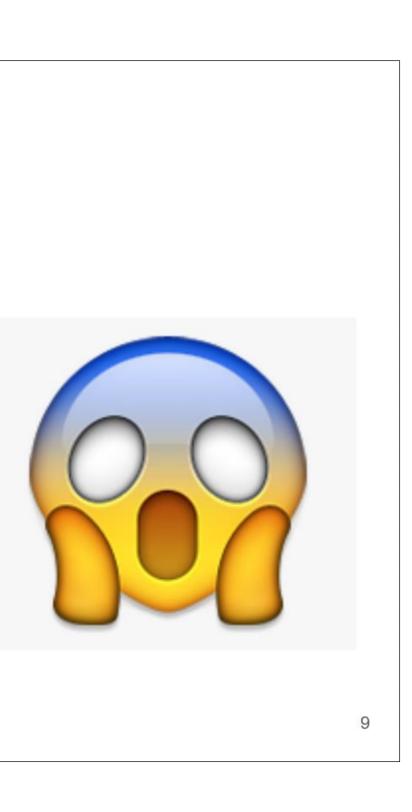


Replace BLIS abstraction for computing BLAS Level 3 operations for GEMM with a direct implementation

- This gives the best performance.
- Driver is only 500 lines of code.
- Tried to optimize the abstractions.
 - No luck up until now.
 - Might need to do the same for the remaining BLAS Level 3 ops.

GEMM Implementation. Practice

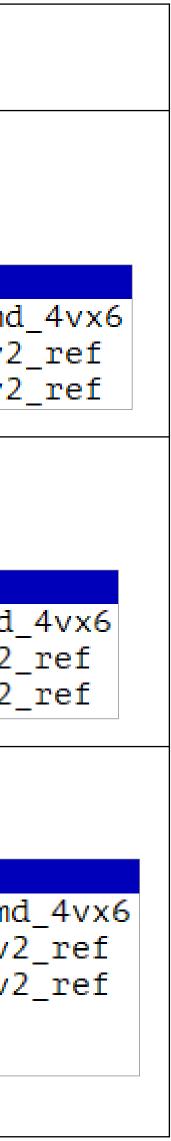
(gdł) bt		
#0	0x0000fffff782c764	in	ugemm_asimd_d3vx8 () from /home/1
#1	0x0000fffff78c2b2c	in	bli_gemm_ker_var2 () from /home/1
#2	0x0000fffff78add98	in	bli_l3_int () from /home/nvidia/e
#3	0x0000fffff78b0328	in	bli_l3_packa () from /home/nvidia
#4	0x0000fffff78add98	in	bli_l3_int () from /home/nvidia/e
#5	0x0000fffff78bd74c	in	bli_gemm_blk_var1 () from /home/1
#6	0x0000fffff78add98	in	bli_l3_int () from /home/nvidia/e
#7	0x0000fffff78b0534	in	bli_l3_packb () from /home/nvidia
#8	0x0000fffff78add98	in	bli_l3_int () from /home/nvidia/e
#9	0x0000fffff78bdd24	in	bli_gemm_blk_var3 () from /home/1
#10	0x0000fffff78add98	in	bli_l3_int () from /home/nvidia/e
#11	0x0000fffff78bd9fc	in	<pre>bli_gemm_blk_var2 () from /home/1</pre>
#12			bli_l3_int () from /home/nvidia/e
#13	0x0000fffff78ace6c	in	<pre>bli_l3_thread_decorator_entry ()</pre>
#14	0x0000fffff78ad034	in	bli_l3_thread_decorator () from ,
#15	0x0000fffff78be5d0	in	bli_gemm_front () from /home/nvic
#16	0x0000fffff78ae104	in	<pre>bli_gemm_ex () from /home/nvidia,</pre>
#17	0x0000fffff78e65f8	in	<pre>nvcpublas_core_dgemm () from /hor</pre>
#18	0x0000fffff7a89b04	in	dgemm_ () from /home/nvidia/efome
	0 0000 1 5		



Optimizations Reducing Framework Overheads

	Before	After
64 ³	86.8%, 33.6 Gflop	97.4%, 38.5 GFlops
1 thr	Overhead Symbol 77.00% [.] bli_dgemm_neoverse_v2_asimd_4vx6 5.82% [.] bli_dpackm_nrxk_neoverse_v2_ref 3.96% [.] bli_dpackm_mrxk_neoverse_v2_ref	Overhead Symbol 85.88% [.] bli_dgemm_neoverse_v2_asimd 6.91% [.] bli_dpackm_nrxk_neoverse_v2 4.61% [.] bli_dpackm_mrxk_neoverse_v2
512 ³	99.4%, 46.6 GFlop	99.8%, 47.0 GFlops
1 thr	Overhead Symbol 96.05% [.] bli_dgemm_neoverse_v2_asimd_4vx6 2.15% [.] bli_dpackm_mrxk_neoverse_v2_ref 1.21% [.] bli_dpackm_nrxk_neoverse_v2_ref	Overhead Symbol 95.81% [.] bli_dgemm_neoverse_v2_asimd_ 2.73% [.] bli_dpackm_mrxk_neoverse_v2_ 1.23% [.] bli_dpackm_nrxk_neoverse_v2_
	92.9%, 689 GFlop	95.6%, 713 GFlops
512³ 16 thr	<pre>Overhead Symbol 88.98% [.] bli_dgemm_neoverse_v2_asimd_4vx6 2.43% [.] bli_dpackm_mrxk_neoverse_v2_ref 1.82% [k] 0xffffc0e00858e3b8 1.47% [.] bli_dpackm_nrxk_neoverse_v2_ref 0.80% [k] 0xffffc0e0084d0ca4 0.62% [.] bli thrcomm barrier atomic</pre>	Overhead Symbol 92.08% [.] bli_dgemm_neoverse_v2_asimd 2.09% [.] bli_dpackm_mrxk_neoverse_v2 1.40% [.] bli_dpackm_nrxk_neoverse_v2 1.23% [k] 0xffffc0e00858e3b8 0.57% [k] 0xffffc0e0084d0ca4

9





Nvidia Grace CPU Superchip

NVPL BLAS 0.0.7-rc1 + New GEMM Driver vs NVPL BLAS 0.0.7-rc1 performance ratio Precision: d, Number of threads: 1

	64	128	256	512	1024	-
GEMM_d_NN -	1.15	1.02	1.01	1.01	1.01	- 1.
GEMM_d_NT	1.16	1.04	1.01	1.01	1.01	- 1.
GEMM_d_TN -	1.15	1.03	1.01	1.02	1.01	- 1. - 0.
GEMM_d_TT -	1.14	1.03	1.02	1.01	1.01	- 0.
			m = n = k			-

NVPL BLAS 0.0.7-rc1 + New GEMM Driver vs NVPL BLAS 0.0.7-rc1 performance ratio Precision: d, Number of threads: 8

_	64	128	256	512	1024	
GEMM_d_NN -	1.58	1.22	1.03	1.02	1.01	- 1.4
GEMM_d_NT	1.57	1.20	1.03	1.01	1.01	-1.
GEMM_d_TN -	1.53	1.20	1.02	1.01	1.02	- 1.
GEMM_d_TT	1.58	1.17	1.05	1.01	1.01	- 0.
80-			m = n = k			100

NVPL BLAS 0.0.7-rc1 + New GEMM Driver vs NVPL BLAS 0.0.7-rc1 performance ratio Precision: d, Number of threads: 72

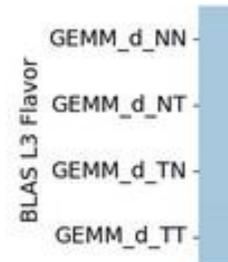
	256	512	1024	2048	4096	-
GEMM_d_NN -	1.49	1.12	1.03	1.01	1.00	- 1.4
GEMM_d_NT	1.50	1.12	1.02	1.01	1.00	- 1.2
GEMM_d_TN -	1.48	1.14	1.02	1.01	1.01	- 1.0
GEMM_d_TT	1.47	1.13	1.03	1.01	1.00	- 0.6
1			m = n = k			

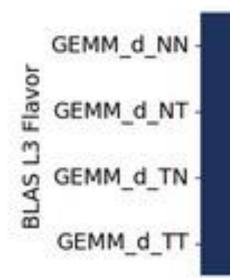


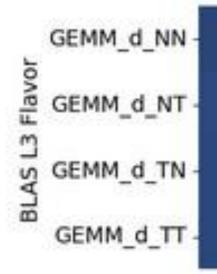


eads (64)72 thr

Optimizations Reducing Framework Overheads







AWS Graviton 3

NVPL BLAS 0.0.7-rc1 + New GEMM Driver vs NVPL BLAS 0.0.7-rc1 performance ratio Precision: d, Number of threads: 1

64	128	256	512	1024	-
1.18	1.04	1.01	1.01	1.01	- 1.4
1.18	1.05	1.01	1.01	1.01	- 1.2
1.18	1.05	1.01	1.01	1.00	- 1.0
1.19	1.05	1.01	1.01	1.00	- 0.6
		m = n = k			-

NVPL BLAS 0.0.7-rc1 + New GEMM Driver vs NVPL BLAS 0.0.7-rc1 performance ratio Precision: d, Number of threads: 8

64	128	256	512	1024	-
1.72	1.24	1.04	1.01	1.02	- 1.4
1.77	1.21	1.06	1.01	1.01	-1.2
1.74	1.24	1.06	1.01	1.01	- 1.0
1.76	1.21	1.06	1.01	1.02	- 0.6
		m = n = k			

NVPL BLAS 0.0.7-rc1 + New GEMM Driver vs NVPL BLAS 0.0.7-rc1 performance ratio Precision: d, Number of threads: 64

256	512	1024	2048	4096	-
1.46	1.10	1.03	1.01	1.01	- 1.4
1.45	1.10	1.03	1.01	1.01	- 1.2
1.45	1.10	1.03	1.01	1.01	- 1.0
1.46	1.11	1.03	1.01	1.00	- 0.6
		m = n = k			



- NVPL BLAS is based on BLIS with few library architecture extensions to suite binary distribution model • BLIS is a very powerful and flexible framework with amazing code reuse Abstractions and flexibility don't come for free

- Nvidia Grace CPU doesn't require complicated low-level programming

NVPL BLAS next steps

- Extend the functionality
- Continue performance optimizations
 - Small shaped GEMMs
 - Non-GEMM BLAS Level 3
 - Improving thread decomposition
 - Direct complex GEMM implementation
 - BLAS Level 1 and 2

Conclusion and Next Steps





