



# SYCL and oneMKL

Hugh Bird

Rafal Bielski

Duncan McBain

Pablo Lopez Ramos

Argonne – 20<sup>th</sup> June 2024



Established 2002 in  
**Edinburgh, Scotland.**

Grown successfully to around  
100 employees.

In 2022, we became a **wholly  
owned subsidiary** of Intel.



Committed to expanding the  
**open ecosystem** for  
heterogeneous computing.

Through our involvement in  
oneAPI and SYCL  
governance, we help to  
**maintain and develop** open  
standards.



Developing at the forefront  
of **cutting-edge research.**

Currently involved in two  
research projects - **SYCLOPS**  
and **AERO**, both funded by  
the Horizon Europe Project.

# Today's event

We will show how to achieve **portability** of mathematical computations **across GPU vendors** using **oneMKL**

60 min presentation + 30 min hands-on session

Please ask questions at the end of each section (agenda in the next slide)



**Hugh Bird**

Staff Software Engineer @ Codeplay  
*member of Performance Libraries Team and oneMKL contributor*



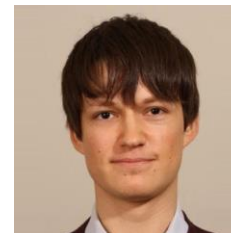
**Duncan McBain**

Senior Software Engineer @ Codeplay  
*product owner of the oneAPI Support Team*



**Pablo Lopez Ramos**

Software Engineering Contractor @ Codeplay  
*member of the oneAPI Support Team*



**Rafal Bielski**

Senior Software Engineer @ Codeplay  
*supports SYCL users in achieving the best performance*

# Agenda

- A quick introduction to SYCL
- What is oneMKL
- oneMKL Interface Library
  - What can it do?
  - How do you use it?
  - How does it work?
  - Building it
  - Gotchas
  - Performance
- Workshop
  - Build and run an example with oneMKL

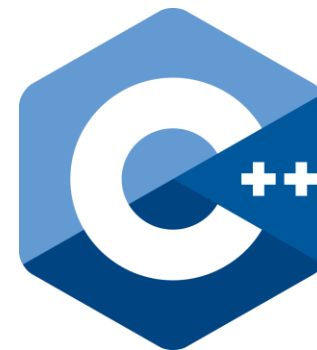
# SYCL

A really quick introduction

SYCL is a **single-source**, **high-level**, **standard C++** programming model, that can **target a range of heterogeneous platforms**

Open standard provided by the non-profit cross-industry **Khronos Group**

Well-defined **concurrency and memory models** enable more optimisation and performance opportunities



# Single C++ source for all architectures

```
1 #include <sycl/sycl.hpp>
2 #include <vector>
3
4 int main() {
5     constexpr static size_t N{10000};
6     std::vector<float> a(N, 1.0f);
7     std::vector<float> b(N, 2.0f);
8     std::vector<float> c(N, 0.0f);
9
10    sycl::queue q{}; Device management with queues
11    {
12        sycl::buffer buf_a{a};
13        sycl::buffer buf_b{b}; Memory management with buffers
14        sycl::buffer buf_c{c};
15        q.submit([&](sycl::handler& h){
16            sycl::accessor acc_a{buf_a, h, sycl::read_write};
17            sycl::accessor acc_b{buf_b, h, sycl::read_only};
18            sycl::accessor acc_c{buf_c, h, sycl::write_only, sycl::no_init};
19            h.parallel_for<class my_kernel>(N, [=](sycl::id<1> id){
20                acc_a[id] += acc_b[id];
21                acc_c[id] = 2.0f * acc_a[id];
22            });
23        }).wait();
24    }
25
26    for (float x : c) {std::cout << x << " ";}
27    std::cout << std::endl;
28
29    return 0;
30 }
```

Submit a work unit to a queue

Execute device code



GPU



CPU



FPGA

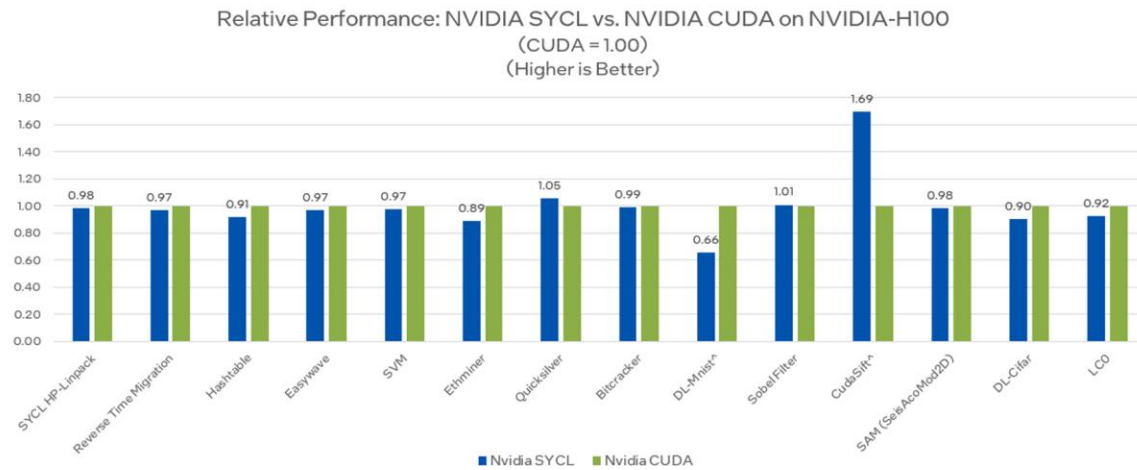


Specialised processors

- Standard C++
  - SYCL 2020 based on ISO C++17
- Unlike in other parallel programming APIs, there are:
  - No pragmas or macros
  - No special attributes
  - No language extensions

# SYCL performance is comparable to native CUDA/HIP

## On NVIDIA GPU – SYCL Provides Comparable Performance to CUDA



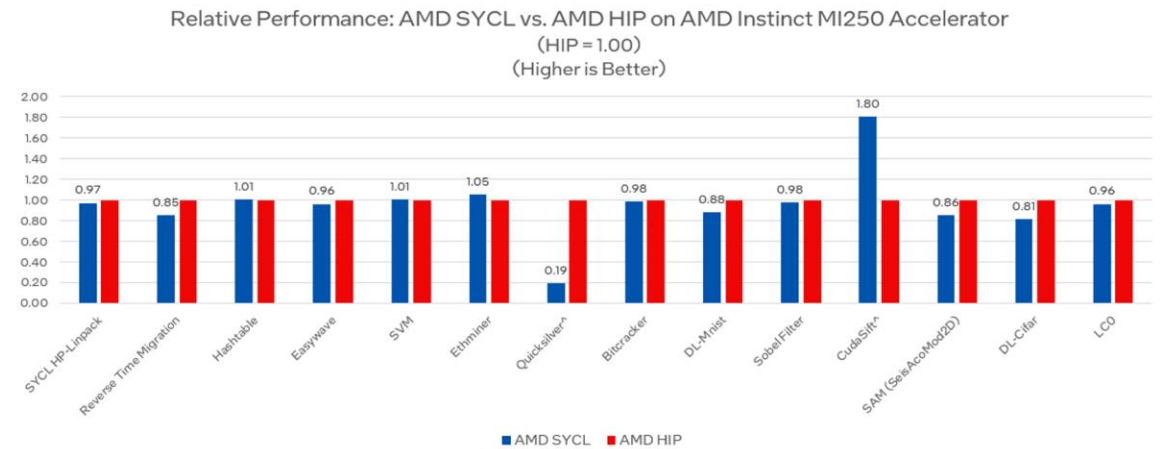
**Testing Date:** Performance results are based on testing by Intel as of August 1, 2023 and may not reflect all publicly available updates.

**Configuration Details and Workload Setup:** Intel® Xeon® Platinum 8360Y CPU @ 2.4GHz, 2 socket, Hyper Thread On, Turbo On, 256GB Hynix DDR4-3200, ucode 0xd000389, GPU: Nvidia H100 PCIe 80GB GPU memory, Software: Velocity Bench benchmark suite branch from 8/1/23, SYCL open source/CLANG 17.0.0, CUDA SDK 12.0 with NVIDIA-NVCC 12.0.76, cuMath 12.0, cuDNN 12.0, Ubuntu 22.04.1, SYCL open source/CLANG compiler switches: -fsycl-targets=nvptx64-nvidia-cuda -Xsycl-target-backend=cuda-gpu-arch=sm\_90, NVIDIA NVCC compiler switches: -O3 -gencode arch=compute\_90,code=sm\_90. Represented workloads with Intel optimizations.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at [www.intel.com/PerformanceIndex](https://www.intel.com/PerformanceIndex). Your costs and results may vary.

## On AMD GPU – SYCL Provides Comparable Performance to HIP



**Testing Date:** Performance results are based on testing by Intel as of August 1, 2023 and may not reflect all publicly available updates.

**Configuration Details and Workload Setup:** AMD EPYC 7313 CPU @ 3.0GHz, 2 socket, AMD Simultaneous Multi-Threading Off, AMD Precision Boost Enabled, 512GB DDR4, ucode 0xa00144, GPU: AMD Instinct MI250 OAM, 128GB GPU memory, Software: Velocity Bench benchmark suite branch from 8/1/23, SYCL open source/CLANG 17.0.0, AMD ROCm 5.6.0 with roc-SOLVER 5.6.0, rocBLAS 5.6.0, Ubuntu 20.04.4, SYCL open source/CLANG compiler switches: -O3 -fsycl -fsycl-targets=amdgn-amd-amdhsa -Xsycl-target-backend=offload-arch=gfx90a, AMD-ROCm compiler switches: -O3. Represented workloads with Intel optimizations.

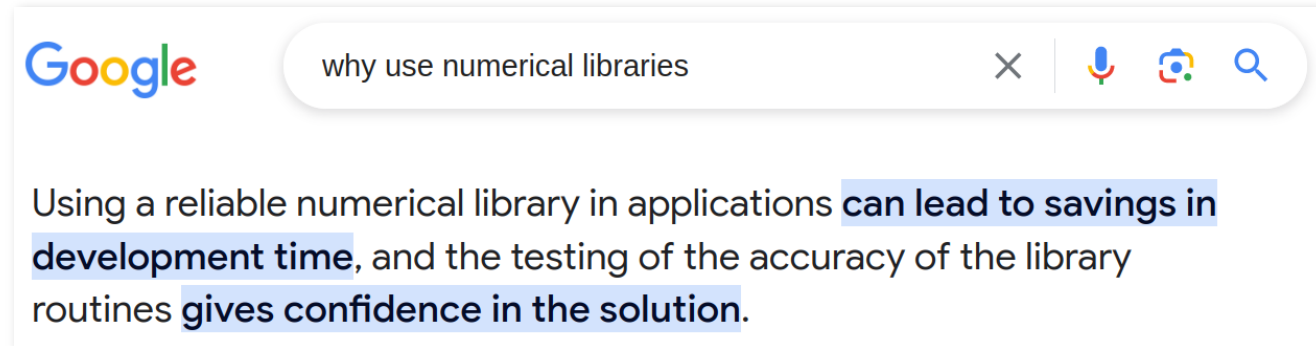
Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at [www.intel.com/PerformanceIndex](https://www.intel.com/PerformanceIndex). Your costs and results may vary.

See [our blog post](#) for more details on these benchmark results



# But should you even write your own kernels?



The open-standard **oneAPI ecosystem** centred around SYCL comes to help!

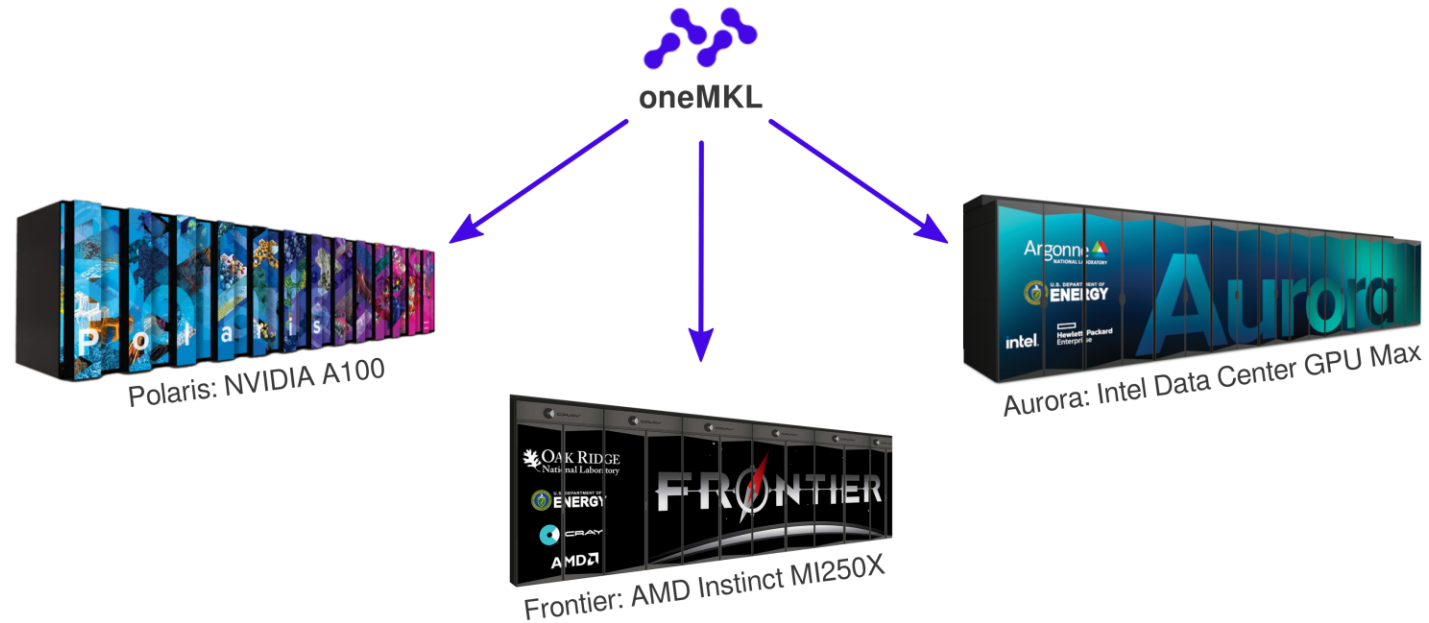
You might be familiar with some of the vendor-specific GPU numerical libraries

- Intel: *Math Kernels Library*
- NVIDIA: *cuBLAS, cuSOLVER, cuRAND, cuFFT*
- AMD: *rocBLAS, rocSOLVER, rocRAND, rocFFT*

Imagine being able to use all of them with single source code → **oneMKL**

# oneMKL provides performance and portability

write single source code



run everywhere

# The oneMKLs

one API, two implementations, and three things

# Pieces of the puzzle

- oneMKL consists of three parts:
- The oneMKL specification - part of the oneAPI specification
- An open-source library implementing the MKL API - oneMKL Interfaces
- The original Intel optimised maths routines - for clarity, *Intel® MKL*

# oneAPI and oneMKL

- oneAPI has a *specification* describing how its components should behave
- oneMKL is a *component* of oneAPI covering fundamental mathematical routines for HPC, engineering, science etc.
- The UXL (Unified Acceleration) Foundation develops these specifications
- The specification is open-source, available on GitHub

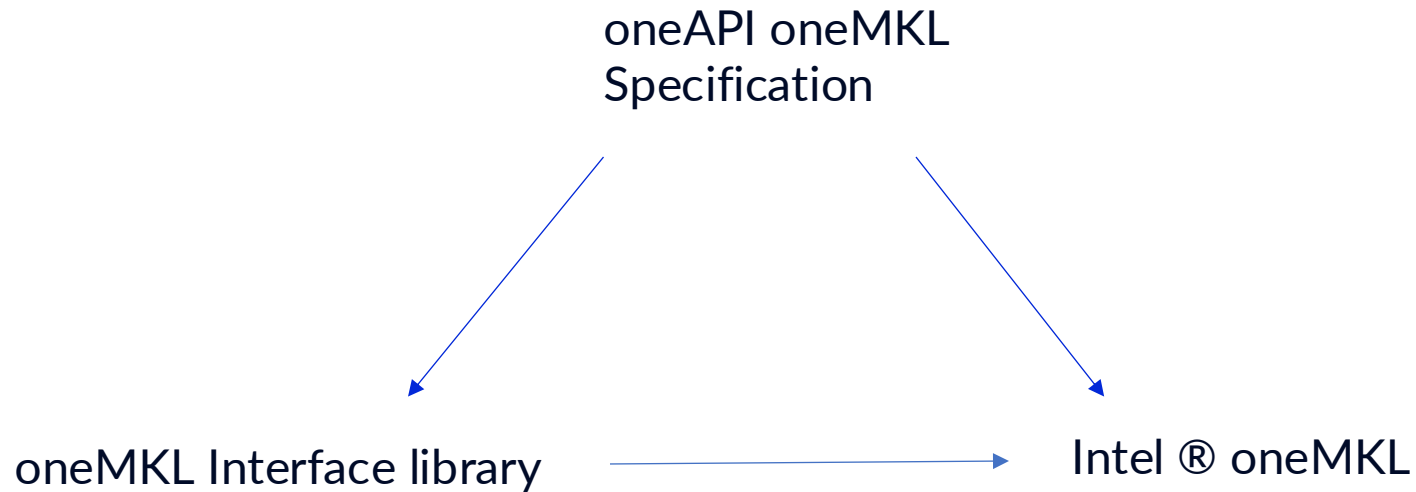
# oneMKL Interface Library

- The topic of this presentation!
- Implements the oneMKL specification, dispatching to other libraries under-the-hood
  - Intel (Intel's MKL)
  - Nvidia (cuBLAS, cuRAND, cuFFT etc.)
  - AMD (rocBLAS, rocFFT etc.)
  - And SYCL-supported devices ("generic" SYCL code)
- DPC++ and AdaptiveCpp
- (AdaptiveCpp support varies by backend but is being worked on)

# Intel® oneMKL

- We'll refer to this as just “MKL” to reduce the confusion
- Intel CPU and Intel GPU
- Mostly conforms to the oneMKL spec except for some legacy reasons
- Available as part of the Intel oneAPI base toolkit

# The oneMKLs



- In short, oneMKL interfaces and Intel® MKL both implement the oneMKL specification
- oneMKL interfaces can dispatch to Intel® MKL as well as other vendor libraries
- Intel® MKL is available via the Intel website as part of the oneAPI base toolkit
- oneMKL interfaces are available on GitHub

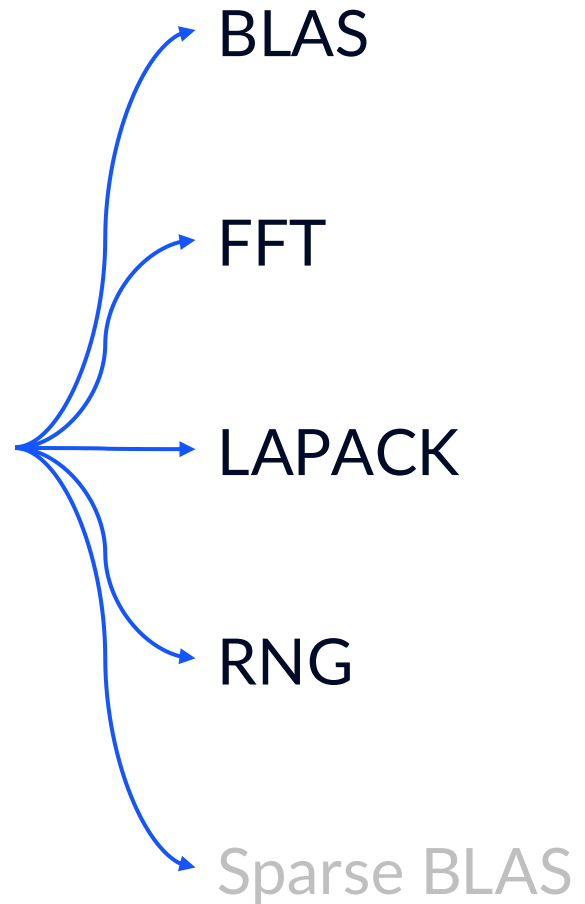


oneMKL: capabilities

# Domains

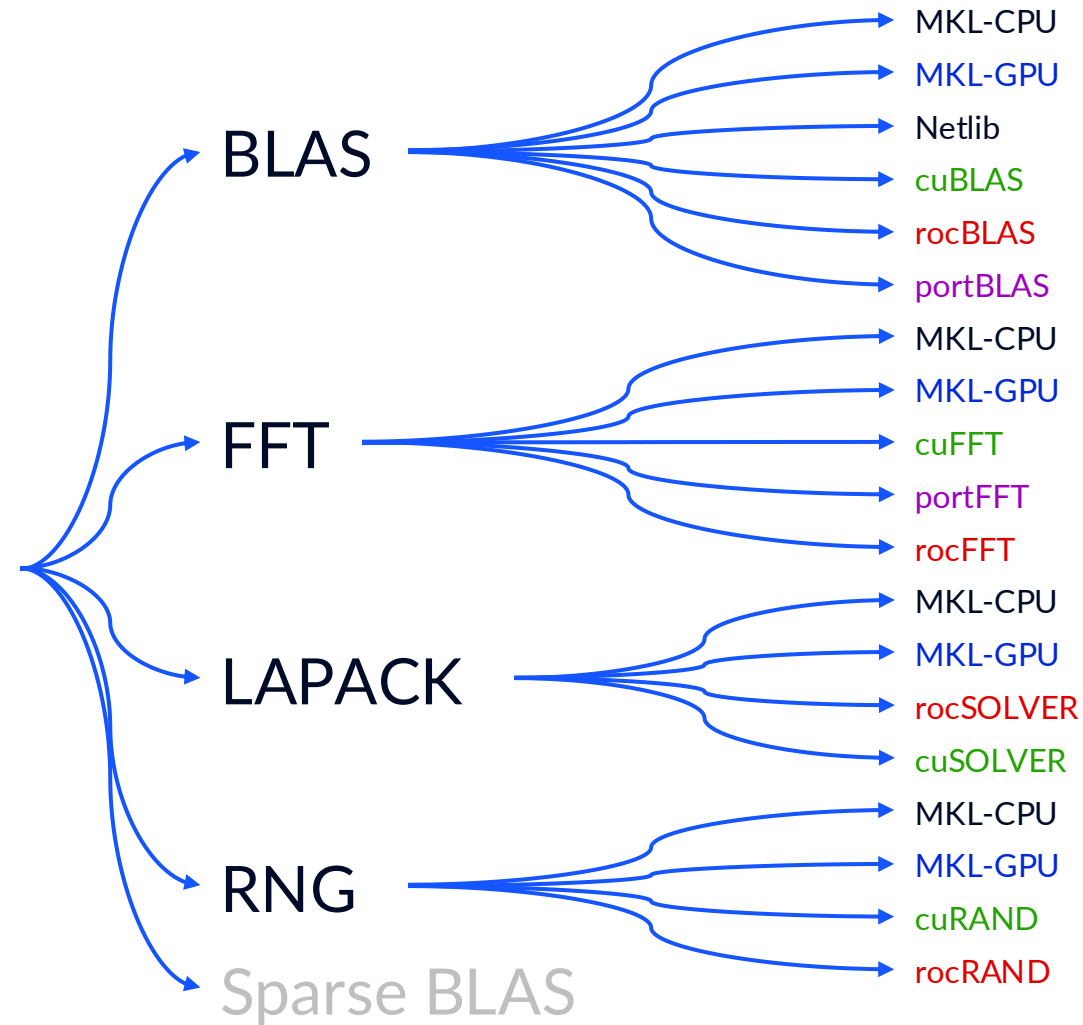
- BLAS
- LAPACK
- DFT
- RNG
- Sparse BLAS

**oneMKL  
Interface Library**



# Backends

## oneMKL Interface Library



# Runtime dispatch

```
// Get a sycl::queue on any vendor's device.  
sycl::queue myQueue;  
  
// oneMKL handles the dispatch to the  
// correct backend library.  
oneapi::mkl::<fn>(myQueue, ...);
```

- oneMKL can build with support for multiple vendors at once.
- oneMKL can automatically dispatch to the correct backend library.
- Backends are lazily dlopened

# Static dispatch

```
using oneapi::mkl;

// Choose a particular device
sycl::queue intelQueue(myIntelGPUSelector);

// Use a selector that uses a particular /
// oneMKL backend.
backend_selector<backend::mklgpu>
    mklgpuSelector{intelGpuQueue};

// Call a backend function directly.
oneapi::mkl::<fn>(mklgpuSelector, ...);
```

- Avoid overhead of dispatch tables by linking directly against backend libraries.

# oneMKL demo

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

oneMKL: using it

# BLAS

```
sycl::queue syclQueue;  
  
// Your data needs to be accessible on the GPU.  
auto dev_A = sycl::malloc_device<float>(sizeA, syclQueue);  
// ... allocate memory, give it relevant values.  
  
// Its like the BLAS API, but taking a queue argument. The USM API returns an event.  
gemm_done = oneapi::mkl::blas::column_major::gemm(syclQueue, transA, transB, m, n, k, alpha,  
                                                  dev_A, ldA, dev_B, ldB, beta, dev_C, ldC);  
  
// Wait for the work to finish.  
gemm_done.wait_and_throw();
```

[https://github.com/oneapi-src/oneMKL/blob/develop/examples/blas/run\\_time\\_dispatching/level3/gemm\\_usm.cpp](https://github.com/oneapi-src/oneMKL/blob/develop/examples/blas/run_time_dispatching/level3/gemm_usm.cpp)



# Random number generation

```
using oneapi::mkl;

// A random number generator is linked to a sycl::queue
rng::default_engine engine(syclQueue, seed);
rng::uniform<float> distribution(low, high);

// Use the state we generated earlier.
auto eventOut = rng::generate(distribution, engine, n, deviceMem);

// Wait for the work to finish.
eventOut.wait_and_throw()
```

[https://github.com/oneapi-src/oneMKL/blob/develop/examples/rng/run\\_time\\_dispatching/uniform\\_usm.cpp](https://github.com/oneapi-src/oneMKL/blob/develop/examples/rng/run_time_dispatching/uniform_usm.cpp)

# DFT

```
using oneapi::mkl;

// A descriptor describes the DFT you want..
dft::descriptor<dft::precision::SINGLE, dft::domain::REAL> desc(N);
desc.set_value(dft::config_param::PLACEMENT, dft::config_value::INPLACE);

// Once set, it is committed on for the chosen queue.
desc.commit(syclQueue);

// Compute the DFTs...
auto computeEvent = dft::compute_forward(desc, x_usm);

// Wait for the result.
computeEvent.wait_and_throw();
```

[https://github.com/oneapi-src/oneMKL/blob/develop/examples/dft/run\\_time\\_dispatching/real\\_fwd\\_usm.cpp](https://github.com/oneapi-src/oneMKL/blob/develop/examples/dft/run_time_dispatching/real_fwd_usm.cpp)

# LAPACK

```
using oneapi::mkl;
// Some APIs need scratch memory to be pre-allocated.
std::int64_t getrf_scratchpad_size = lapack::getrf_scratchpad_size<float>(syclQueue, m, n, lda);
float* getrf_scratchpad = sycl::malloc_shared<float>(getrf_scratchpad_size, syclQueue);

// ... More allocs, etc.
// LU factorization on device
auto getrfDone = lapack::getrf(syclQueue, m, n, devA, lda, dev_ipiv, getrf_scratchpad, getrf_scratchpad_size);
// Use LU factorization to solve system on device. Needs LU factorization to be complete.
auto getsrDone = lapack::getrs(syclQueue, trans, n, nrhs, devA, lda, dev_ipiv,
                              devB, ldb, getsr_scratchpad, getsr_scratchpad_size, {getrfDone});

// Wait until calculations are done
syclQueue.wait_and_throw();
```

[https://github.com/oneapi-src/oneMKL/blob/develop/examples/lapack/run\\_time\\_dispatching/getrs\\_usm.cpp](https://github.com/oneapi-src/oneMKL/blob/develop/examples/lapack/run_time_dispatching/getrs_usm.cpp)

# CMake

## oneMKL is installed

```
find_package(oneMKL REQUIRED)

// Link everything, runtime dispatch
target_link_library(mytarget PRIVATE MKL::onemkl)

// Link against specific backend
target_link_library(mytarget PRIVATE
    MKL::onemkl_<domain>_<backend>)
```

... And add <install\_dir>/lib to your LD\_LIBRARY\_PATH if its installed in a non-standard location, otherwise dlopen doesn't work.

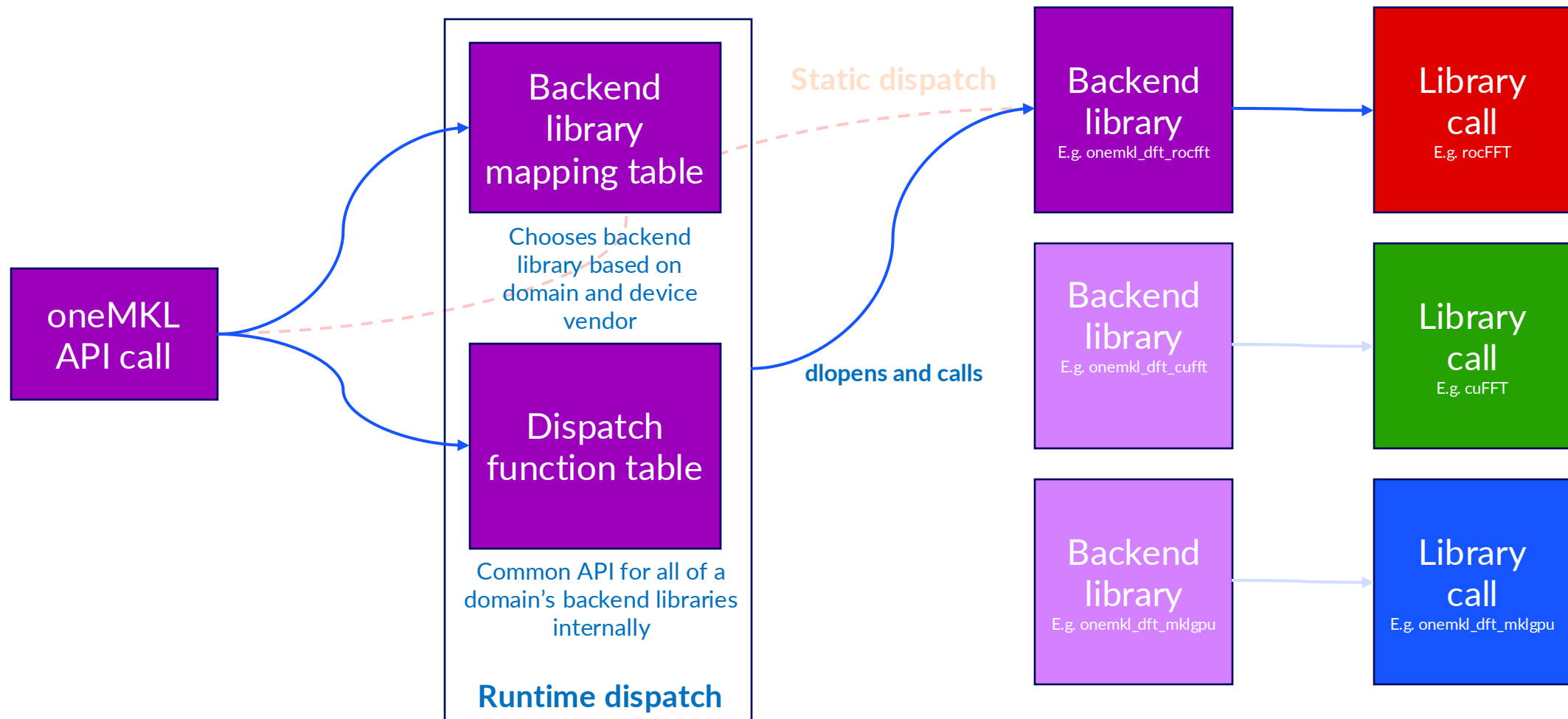
## Using FetchContent

```
include(FetchContent)
set(BUILD_FUNCTIONAL_TESTS OFF)
set(BUILD_EXAMPLES OFF)
set(ENABLE_<BACKEND_NAME>_BACKEND ON)
FetchContent_Declare(
    onemkl_interface_library
    GIT_REPOSITORY https://github.com/oneapi-src/oneMKL.git
    GIT_TAG develop
)
FetchContent_MakeAvailable(onemkl_interface_library)

target_link_libraries(myTarget PRIVATE onemkl)
// or for a specific backend
target_link_libraries(myTarget PRIVATE onemkl_<domain>_<backend>)
```

oneMKL: on the inside

# The runtime dispatch mechanism



# oneMKL: building it

The documentation makes it look harder than it is

# With DPC++

```
cmake $ONEMKL_DIR \  
-GNinja \  
-DCMAKE_CXX_COMPILER=icpx \  
-DCMAKE_C_COMPILER=icx \  
-DENABLE_MKLGPU_BACKEND=ON \  
-DENABLE_MKLCPU_BACKEND=ON \  
-DENABLE_CUFFT_BACKEND=ON \  
-DENABLE_CUBLAS_BACKEND=ON \  
-DENABLE_ROCRAND_BACKEND=ON \  
-DENABLE_FUNCTIONAL_TESTS=OFF \  
-DHIP_TARGETS=gfx90a
```

- Building isn't that complicated.
  - Enable the backends you want
  - Set HIP\_TARGETS on AMD
  - Disable functional tests in most cases
- What does supported vs unsupported mean?
  - Supported is what we actually test with
  - But using icpx + Codeplay plugins does work, and it's probably what you should do.
- On AMD, you can only have a single arch right now.



oneMKL: gotchas

# oneMKL: gotchas

- Backend libraries don't all support every feature
  - Eg. The cuFFT backend doesn't support scaling.
- Backend libraries make different guarantees
  - Eg. The rocFFT backend can modify input.

# oneMKL: gotchas

- Variadic functions like
  - `desc.set_value(dft::config_param::INPUT_STRIDES, myStrides);`
  - The spec uses `int64_t`
  - Variadic arguments means that the compiler won't tell you you're wrong.
  
- `LD_LIBRARY_PATH`

# Coming from Intel® MKL

## Intel® MKL

```
#include <oneapi/mkl/dfti.hpp>
```

```
DFTI_INPLACE
```

## oneMKL Interface Library

```
#include <oneapi/mkl/dft.hpp>
```

```
oneapi::mkl::dft::config_value::INPLACE
```

oneMKL: performance

# Performance

oneMKL is a thin wrapper calling native backend libraries

- very little overhead, negligible in typical HPC use cases
  - we are working on improving the overhead for small workloads where it may be more visible
- you get **comparable performance + portability**

Let's test this with a simple GEMM example!

$$C \leftarrow \alpha * op(A) * op(B) + \beta * C$$

$op(X)$  is one of  $op(X) = X$  or  $op(X) = X^T$  or  $op(X) = X^H$

$\alpha$  and  $\beta$  are scalars

$A$ ,  $B$  and  $C$  are matrices

$op(A)$  is an  $m$ -by- $k$  matrix

$op(B)$  is a  $k$ -by- $n$  matrix

$C$  is an  $m$ -by- $n$  matrix

# Performance

- Code from [VelocityBench hplinpack DPC++ example](#)
- We call it with double-precision matrices with  $\{m,n,k\} = \{16384, 2048, 2048\}$  filled with random values in the range 0.0–1.0
- Three code versions compiled into four executables:
  - CUDA: `cublasDgemm`
  - HIP: `hipblasDgemm`
  - IntelMKL / oneMKL (same API): `oneapi::mkl::blas::column_major::gemm`

# Performance

Same code runs on 7 different devices from 3 different vendors (6 GPUs and 1 CPU)

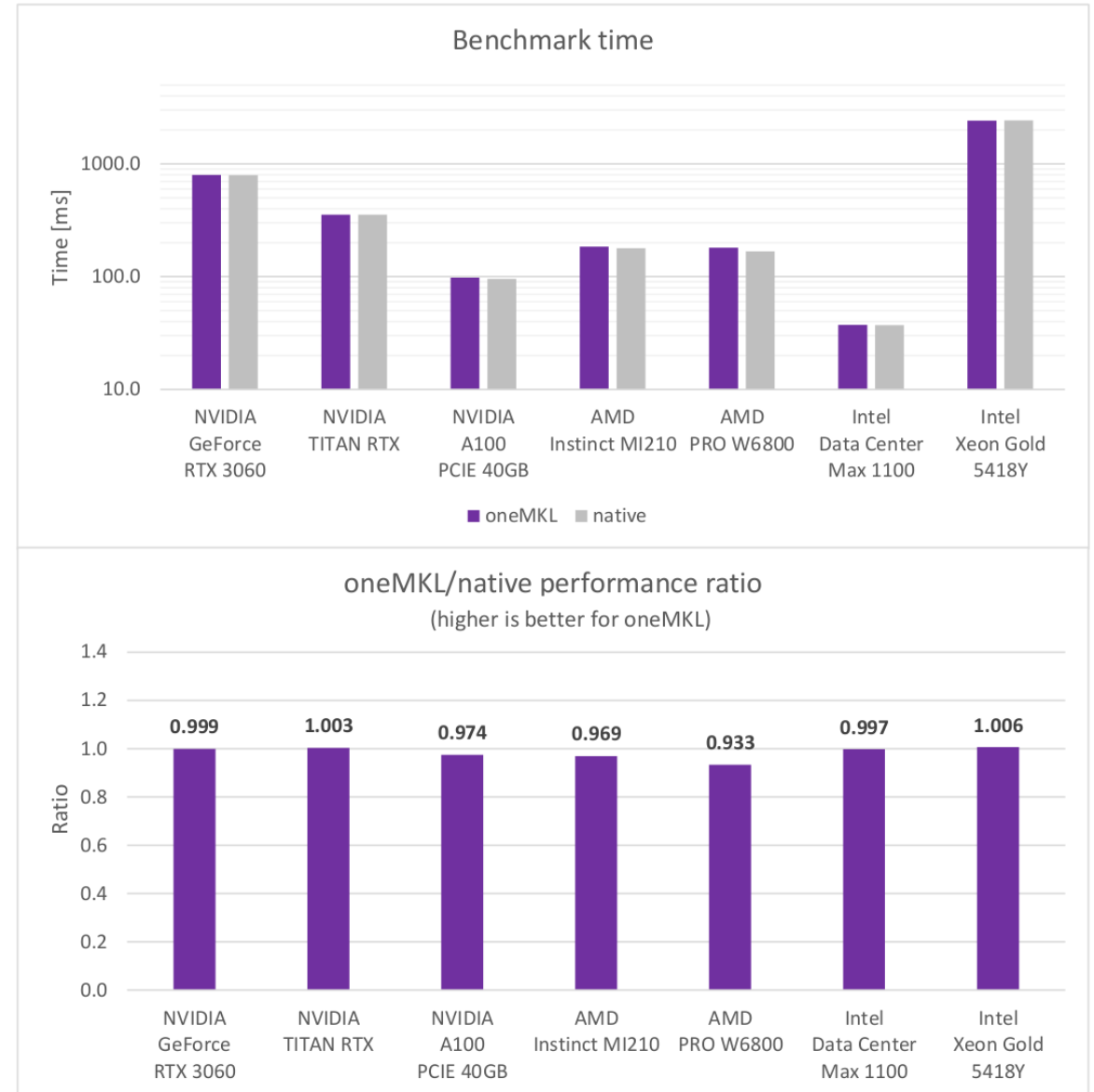
Comparable results to the native library in all cases

No need to maintain three versions of the code if just one does it!

Performance varies by use, configuration and other factors. Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.

No product or component can be absolutely secure. Your costs and results may vary. Intel technologies may require enabled hardware, software or service activation.

Details of the software and hardware used to produce these results are available in the backup slides.



"native" means cuBLAS on NVIDIA GPU, hipBLAS on AMD GPU and Intel MKL on Intel GPU/CPU



Time to give it a go

## ... But first!

- Want something that oneMKL doesn't have / support?
  - Make an issue!
- Find a bug?
  - Make an issue!
- Finding something confusing?
  - Make an issue!

Issues let us justify spending time on improving oneMKL.

# Hands-on with oneMKL

- We have a pre-prepared single-page application that needs the oneMKL section added on GitHub:
- [https://github.com/codeplaysoftware/syclacademy/tree/main/Code\\_Exercises/OneMKL\\_gemm](https://github.com/codeplaysoftware/syclacademy/tree/main/Code_Exercises/OneMKL_gemm)
- Instructions are on the page, but the starting skeleton is in the "source" file, with the answer in the "solution" file, but we'd encourage you to give it a go before checking!

# Hints

- The sample is performing a GEMM, so add this function
- If you are using the compiler standalone, i.e. without CMake or similar, flags are required:
- `icpx -fsycl -L$ENV{MKLR00T} -lonemkl solution_onemkl_usm_gemm.cpp`
- If using USM, copies to the device and back will be required



# oneAPI Plugins for NVIDIA/AMD

Scan QR code or visit [developer.codeplay.com](https://developer.codeplay.com)





# Disclaimers

A wee bit of legal

Performance varies by use, configuration and other factors.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

© Codeplay Software Ltd.. Codeplay, Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

# Performance benchmark details

Main function: <https://gist.github.com/rafbiels/e93b70098d46e947ce825eb1cc95f6b5>

VelocityBench dpcpp\_dgemm.cpp: [https://github.com/oneapi-src/Velocity-Bench/blob/50343b438e838ceae1eb11a10196d3ae90aebb67/hplinpac/dpcpp/hpl-2.3/src/dpcpp/dpcpp\\_dgemm.cpp](https://github.com/oneapi-src/Velocity-Bench/blob/50343b438e838ceae1eb11a10196d3ae90aebb67/hplinpac/dpcpp/hpl-2.3/src/dpcpp/dpcpp_dgemm.cpp)

Base compilation command: `icpx -fsycl -fsycl-targets=${SYCL_TARGET} ${OFFLOAD_ARCH_FLAGS} -o onemkl main.cpp dpcpp_dgemm.cpp`

Extra flags:

oneMKL: `-lonemkl`

Intel MKL: `-DMKL_ILP64 -qmkl=parallel -qtbb`

cuBLAS: `-DUSE_CUBLAS -lcublas -lcuda -lcudart -L$(dirname $(which nvcc))/../lib64`

hipBLAS: `-DUSE_HIPBLAS -D__HIP_PLATFORM_AMD__=${HIP_TARGET} -L${ROCM_PATH}/hipblas/lib/ -L${ROCM_PATH}/hip/lib -lhipblas -lamdhip64`

Software stack: Ubuntu 22.04.4 LTS, oneAPI Base Toolkit 2024.1, CUDA 12.4, ROCm 5.4.3, oneMKL interfaces commit 6d6a7b711dbc55c49370b8ddb9db6e81a6ac27 + PR #490

Hardware (6 machines):

1. Intel i9-12900K CPU + NVIDIA GeForce RTX 3060 GPU
2. Intel Xeon Platinum 8268 CPU + NVIDIA TITAN RTX GPU
3. Intel Xeon Gold 6326 CPU + NVIDIA A100 PCIE 40GB GPU
4. 2x AMD EPYC 7402 CPU + AMD Instinct MI210 GPU
5. Intel i9-12900K CPU + AMD Radeon PRO W6800 GPU
6. 2x Intel Xeon Gold 5418Y CPU + Intel Data Center Max 1100 GPU

Tested on 31 May 2024

Performance varies by use, configuration and other factors.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details.

No product or component can be absolutely secure.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.