# The file split\_test\_helper.h was generated at first run,

# it is now included in test/

if(EXISTS ${CMAKE\_CURRENT\_BINARY\_DIR}/split\_test\_helper.h)

file(REMOVE ${CMAKE\_CURRENT\_BINARY\_DIR}/split\_test\_helper.h)

endif()

set\_property(GLOBAL PROPERTY EIGEN\_CURRENT\_SUBPROJECT "Unsupported")

add\_custom\_target(BuildUnsupported)

include\_directories(../../test ../../unsupported ../../Eigen

${CMAKE\_CURRENT\_BINARY\_DIR}/../../test)

find\_package (Threads)

find\_package(GoogleHash)

if(GoogleHash\_FOUND)

add\_definitions("-DEIGEN\_GOOGLEHASH\_SUPPORT")

include\_directories(${GOOGLEHASH\_INCLUDES})

ei\_add\_property(EIGEN\_TESTED\_BACKENDS "GoogleHash, ")

else()

ei\_add\_property(EIGEN\_MISSING\_BACKENDS "GoogleHash, ")

endif()

find\_package(Adolc)

if(Adolc\_FOUND)

include\_directories(${ADOLC\_INCLUDES})

ei\_add\_property(EIGEN\_TESTED\_BACKENDS "Adolc, ")

if(EIGEN\_TEST\_CXX11)

ei\_add\_test(forward\_adolc "" ${ADOLC\_LIBRARIES})

else()

message(STATUS "Adolc found, but tests require C++11 mode")

endif()

else()

ei\_add\_property(EIGEN\_MISSING\_BACKENDS "Adolc, ")

endif()

# this test seems to never have been successful on x87, so is considered to contain a FP-related bug.

# see thread: "non-linear optimization test summary"

ei\_add\_test(NonLinearOptimization)

ei\_add\_test(NumericalDiff)

ei\_add\_test(autodiff\_scalar)

ei\_add\_test(autodiff)

ei\_add\_test(BVH)

ei\_add\_test(matrix\_exponential)

ei\_add\_test(matrix\_function)

ei\_add\_test(matrix\_power)

ei\_add\_test(matrix\_square\_root)

ei\_add\_test(alignedvector3)

ei\_add\_test(FFT)

ei\_add\_test(EulerAngles)

find\_package(MPREAL)

if(MPREAL\_FOUND AND EIGEN\_COMPILER\_SUPPORT\_CPP11)

ei\_add\_property(EIGEN\_TESTED\_BACKENDS "MPFR C++, ")

include\_directories(${MPREAL\_INCLUDES})

ei\_add\_test(mpreal\_support "-std=c++11" "${MPREAL\_LIBRARIES}" )

else()

ei\_add\_property(EIGEN\_MISSING\_BACKENDS "MPFR C++, ")

endif()

ei\_add\_test(sparse\_extra "" "")

find\_package(FFTW)

if(FFTW\_FOUND)

ei\_add\_property(EIGEN\_TESTED\_BACKENDS "fftw, ")

include\_directories( ${FFTW\_INCLUDES} )

if(FFTWL\_LIB)

ei\_add\_test(FFTW "-DEIGEN\_FFTW\_DEFAULT -DEIGEN\_HAS\_FFTWL" "${FFTW\_LIBRARIES}" )

else()

ei\_add\_test(FFTW "-DEIGEN\_FFTW\_DEFAULT" "${FFTW\_LIBRARIES}" )

endif()

else()

ei\_add\_property(EIGEN\_MISSING\_BACKENDS "fftw, ")

endif()

option(EIGEN\_TEST\_OPENGL "Enable OpenGL support in unit tests" OFF)

if(EIGEN\_TEST\_OPENGL)

find\_package(OpenGL)

find\_package(GLUT)

find\_package(GLEW)

if(OPENGL\_FOUND AND GLUT\_FOUND AND GLEW\_FOUND)

include\_directories(${OPENGL\_INCLUDE\_DIR} ${GLUT\_INCLUDE\_DIR} ${GLEW\_INCLUDE\_DIRS})

ei\_add\_property(EIGEN\_TESTED\_BACKENDS "OpenGL, ")

set(EIGEN\_GL\_LIB ${GLUT\_LIBRARIES} ${GLEW\_LIBRARIES} ${OPENGL\_LIBRARIES})

ei\_add\_test(openglsupport "" "${EIGEN\_GL\_LIB}" )

else()

ei\_add\_property(EIGEN\_MISSING\_BACKENDS "OpenGL, ")

endif()

else()

ei\_add\_property(EIGEN\_MISSING\_BACKENDS "OpenGL, ")

endif()

ei\_add\_test(polynomialsolver)

ei\_add\_test(polynomialutils)

ei\_add\_test(splines)

ei\_add\_test(gmres)

ei\_add\_test(dgmres)

ei\_add\_test(minres)

ei\_add\_test(idrs)

ei\_add\_test(levenberg\_marquardt)

ei\_add\_test(kronecker\_product)

ei\_add\_test(bessel\_functions)

ei\_add\_test(special\_functions)

ei\_add\_test(special\_packetmath "-DEIGEN\_FAST\_MATH=1")

if(EIGEN\_TEST\_CXX11)

if(EIGEN\_TEST\_SYCL)

set(EIGEN\_SYCL ON)

# Forward CMake options as preprocessor definitions

if(EIGEN\_SYCL\_USE\_DEFAULT\_SELECTOR)

add\_definitions(-DEIGEN\_SYCL\_USE\_DEFAULT\_SELECTOR=${EIGEN\_SYCL\_USE\_DEFAULT\_SELECTOR})

endif()

if(EIGEN\_SYCL\_NO\_LOCAL\_MEM)

add\_definitions(-DEIGEN\_SYCL\_NO\_LOCAL\_MEM=${EIGEN\_SYCL\_NO\_LOCAL\_MEM})

endif()

if(EIGEN\_SYCL\_LOCAL\_MEM)

add\_definitions(-DEIGEN\_SYCL\_LOCAL\_MEM=${EIGEN\_SYCL\_LOCAL\_MEM})

endif()

if(EIGEN\_SYCL\_MAX\_GLOBAL\_RANGE)

add\_definitions(-DEIGEN\_SYCL\_MAX\_GLOBAL\_RANGE=${EIGEN\_SYCL\_MAX\_GLOBAL\_RANGE})

endif()

if(EIGEN\_SYCL\_LOCAL\_THREAD\_DIM0)

add\_definitions(-DEIGEN\_SYCL\_LOCAL\_THREAD\_DIM0=${EIGEN\_SYCL\_LOCAL\_THREAD\_DIM0})

endif()

if(EIGEN\_SYCL\_LOCAL\_THREAD\_DIM1)

add\_definitions(-DEIGEN\_SYCL\_LOCAL\_THREAD\_DIM1=${EIGEN\_SYCL\_LOCAL\_THREAD\_DIM1})

endif()

if(EIGEN\_SYCL\_REG\_M)

add\_definitions(-DEIGEN\_SYCL\_REG\_M=${EIGEN\_SYCL\_REG\_M})

endif()

if(EIGEN\_SYCL\_REG\_N)

add\_definitions(-DEIGEN\_SYCL\_REG\_N=${EIGEN\_SYCL\_REG\_N})

endif()

if(EIGEN\_SYCL\_USE\_PROGRAM\_CLASS)

add\_definitions(-DEIGEN\_SYCL\_USE\_PROGRAM\_CLASS=${EIGEN\_SYCL\_USE\_PROGRAM\_CLASS})

endif()

if(EIGEN\_SYCL\_ASYNC\_EXECUTION)

add\_definitions(-DEIGEN\_SYCL\_ASYNC\_EXECUTION=${EIGEN\_SYCL\_ASYNC\_EXECUTION})

endif()

if(EIGEN\_SYCL\_DISABLE\_SKINNY)

add\_definitions(-DEIGEN\_SYCL\_DISABLE\_SKINNY=${EIGEN\_SYCL\_DISABLE\_SKINNY})

endif()

if(EIGEN\_SYCL\_DISABLE\_DOUBLE\_BUFFER)

add\_definitions(-DEIGEN\_SYCL\_DISABLE\_DOUBLE\_BUFFER=${EIGEN\_SYCL\_DISABLE\_DOUBLE\_BUFFER})

endif()

if(EIGEN\_SYCL\_DISABLE\_RANK1)

add\_definitions(-DEIGEN\_SYCL\_DISABLE\_RANK1=${EIGEN\_SYCL\_DISABLE\_RANK1})

endif()

if(EIGEN\_SYCL\_DISABLE\_SCALAR)

add\_definitions(-DEIGEN\_SYCL\_DISABLE\_SCALAR=${EIGEN\_SYCL\_DISABLE\_SCALAR})

endif()

if(EIGEN\_SYCL\_DISABLE\_GEMV)

add\_definitions(-DEIGEN\_SYCL\_DISABLE\_GEMV=${EIGEN\_SYCL\_DISABLE\_GEMV})

endif()

if(EIGEN\_SYCL\_DISABLE\_ARM\_GPU\_CACHE\_OPTIMISATION)

add\_definitions(-DEIGEN\_SYCL\_DISABLE\_ARM\_GPU\_CACHE\_OPTIMISATION=${EIGEN\_SYCL\_DISABLE\_ARM\_GPU\_CACHE\_OPTIMISATION})

endif()

if(EIGEN\_SYCL\_TRISYCL)

# triSYCL now requires c++17.

set(CMAKE\_CXX\_STANDARD 17)

else()

if(MSVC)

# Set the host and device compilers C++ standard to C++14. On Windows setting this to C++11

# can cause issues with the ComputeCpp device compiler parsing Visual Studio Headers.

set(CMAKE\_CXX\_STANDARD 14)

list(APPEND COMPUTECPP\_USER\_FLAGS -DWIN32)

else()

set(CMAKE\_CXX\_STANDARD 11)

list(APPEND COMPUTECPP\_USER\_FLAGS -Wall)

endif()

# The following flags are not supported by Clang and can cause warnings

# if used with -Werror so they are removed here.

if(COMPUTECPP\_USE\_COMPILER\_DRIVER)

set(CMAKE\_CXX\_COMPILER ${ComputeCpp\_DEVICE\_COMPILER\_EXECUTABLE})

string(REPLACE "-Wlogical-op" "" CMAKE\_CXX\_FLAGS ${CMAKE\_CXX\_FLAGS})

string(REPLACE "-Wno-psabi" "" CMAKE\_CXX\_FLAGS ${CMAKE\_CXX\_FLAGS})

string(REPLACE "-ansi" "" CMAKE\_CXX\_FLAGS ${CMAKE\_CXX\_FLAGS})

endif()

list(APPEND COMPUTECPP\_USER\_FLAGS

-DEIGEN\_NO\_ASSERTION\_CHECKING=1

-no-serial-memop

-Xclang

-cl-mad-enable)

endif()

ei\_add\_test(cxx11\_tensor\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_image\_op\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_math\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_forced\_eval\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_broadcast\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_device\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_reduction\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_morphing\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_shuffling\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_padding\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_builtins\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_contract\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_concatenation\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_reverse\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_convolution\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_striding\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_chipping\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_layout\_swap\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_inflation\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_random\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_generator\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_patch\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_image\_patch\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_volume\_patch\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_argmax\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_custom\_op\_sycl ${STD\_CXX\_FLAG})

ei\_add\_test(cxx11\_tensor\_scan\_sycl ${STD\_CXX\_FLAG})

set(EIGEN\_SYCL OFF)

endif()

ei\_add\_test(cxx11\_eventcount "-pthread" "${CMAKE\_THREAD\_LIBS\_INIT}")

ei\_add\_test(cxx11\_runqueue "-pthread" "${CMAKE\_THREAD\_LIBS\_INIT}")

ei\_add\_test(cxx11\_non\_blocking\_thread\_pool "-pthread" "${CMAKE\_THREAD\_LIBS\_INIT}")

ei\_add\_test(cxx11\_meta)

ei\_add\_test(cxx11\_maxsizevector)

ei\_add\_test(cxx11\_tensor\_argmax)

ei\_add\_test(cxx11\_tensor\_assign)

ei\_add\_test(cxx11\_tensor\_block\_access)

ei\_add\_test(cxx11\_tensor\_block\_eval)

ei\_add\_test(cxx11\_tensor\_block\_io)

ei\_add\_test(cxx11\_tensor\_broadcasting)

ei\_add\_test(cxx11\_tensor\_casts)

ei\_add\_test(cxx11\_tensor\_chipping)

ei\_add\_test(cxx11\_tensor\_comparisons)

ei\_add\_test(cxx11\_tensor\_concatenation)

ei\_add\_test(cxx11\_tensor\_const)

ei\_add\_test(cxx11\_tensor\_contraction)

ei\_add\_test(cxx11\_tensor\_convolution)

ei\_add\_test(cxx11\_tensor\_custom\_index)

ei\_add\_test(cxx11\_tensor\_custom\_op)

ei\_add\_test(cxx11\_tensor\_dimension)

ei\_add\_test(cxx11\_tensor\_empty)

ei\_add\_test(cxx11\_tensor\_executor "-pthread" "${CMAKE\_THREAD\_LIBS\_INIT}")

ei\_add\_test(cxx11\_tensor\_expr)

ei\_add\_test(cxx11\_tensor\_fft)

ei\_add\_test(cxx11\_tensor\_fixed\_size)

ei\_add\_test(cxx11\_tensor\_forced\_eval)

ei\_add\_test(cxx11\_tensor\_generator)

ei\_add\_test(cxx11\_tensor\_ifft)

ei\_add\_test(cxx11\_tensor\_image\_patch)

ei\_add\_test(cxx11\_tensor\_index\_list)

ei\_add\_test(cxx11\_tensor\_inflation)

ei\_add\_test(cxx11\_tensor\_intdiv)

ei\_add\_test(cxx11\_tensor\_io)

ei\_add\_test(cxx11\_tensor\_layout\_swap)

ei\_add\_test(cxx11\_tensor\_lvalue)

ei\_add\_test(cxx11\_tensor\_map)

ei\_add\_test(cxx11\_tensor\_math)

ei\_add\_test(cxx11\_tensor\_mixed\_indices)

ei\_add\_test(cxx11\_tensor\_morphing)

ei\_add\_test(cxx11\_tensor\_move)

ei\_add\_test(cxx11\_tensor\_notification "-pthread" "${CMAKE\_THREAD\_LIBS\_INIT}")

ei\_add\_test(cxx11\_tensor\_of\_complex)

ei\_add\_test(cxx11\_tensor\_of\_const\_values)

ei\_add\_test(cxx11\_tensor\_of\_strings)

ei\_add\_test(cxx11\_tensor\_padding)

ei\_add\_test(cxx11\_tensor\_patch)

ei\_add\_test(cxx11\_tensor\_random)

ei\_add\_test(cxx11\_tensor\_reduction)

ei\_add\_test(cxx11\_tensor\_ref)

ei\_add\_test(cxx11\_tensor\_roundings)

ei\_add\_test(cxx11\_tensor\_scan)

ei\_add\_test(cxx11\_tensor\_shuffling)

ei\_add\_test(cxx11\_tensor\_simple)

ei\_add\_test(cxx11\_tensor\_striding)

ei\_add\_test(cxx11\_tensor\_sugar)

ei\_add\_test(cxx11\_tensor\_thread\_local "-pthread" "${CMAKE\_THREAD\_LIBS\_INIT}")

ei\_add\_test(cxx11\_tensor\_thread\_pool "-pthread" "${CMAKE\_THREAD\_LIBS\_INIT}")

ei\_add\_test(cxx11\_tensor\_trace)

ei\_add\_test(cxx11\_tensor\_volume\_patch)

# ei\_add\_test(cxx11\_tensor\_symmetry)

if("${CMAKE\_SIZEOF\_VOID\_P}" EQUAL "8" AND NOT CMAKE\_CXX\_COMPILER\_ID STREQUAL "MSVC")

# This test requires \_\_uint128\_t which is only available on 64bit systems

ei\_add\_test(cxx11\_tensor\_uint128)

endif()

endif()

# These tests needs nvcc

find\_package(CUDA 7.0)

if(CUDA\_FOUND AND EIGEN\_TEST\_CUDA)

# Make sure to compile without the -pedantic, -Wundef, -Wnon-virtual-dtor

# and -fno-check-new flags since they trigger thousands of compilation warnings

# in the CUDA runtime

# Also remove -ansi that is incompatible with std=c++11.

string(REPLACE "-pedantic" "" CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS}")

string(REPLACE "-Wundef" "" CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS}")

string(REPLACE "-Wnon-virtual-dtor" "" CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS}")

string(REPLACE "-fno-check-new" "" CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS}")

string(REPLACE "-ansi" "" CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS}")

message(STATUS "Flags used to compile cuda code: " ${CMAKE\_CXX\_FLAGS})

if("${CMAKE\_CXX\_COMPILER\_ID}" STREQUAL "Clang")

set(CUDA\_NVCC\_FLAGS "-ccbin ${CMAKE\_C\_COMPILER}" CACHE STRING "nvcc flags" FORCE)

endif()

if(EIGEN\_TEST\_CUDA\_CLANG)

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -std=c++11")

string(APPEND CMAKE\_CXX\_FLAGS " --cuda-path=${CUDA\_TOOLKIT\_ROOT\_DIR}")

foreach(ARCH IN LISTS EIGEN\_CUDA\_COMPUTE\_ARCH)

string(APPEND CMAKE\_CXX\_FLAGS " --cuda-gpu-arch=sm\_${ARCH}")

endforeach()

endif()

set(EIGEN\_CUDA\_RELAXED\_CONSTEXPR "--expt-relaxed-constexpr")

if (${CUDA\_VERSION} STREQUAL "7.0")

set(EIGEN\_CUDA\_RELAXED\_CONSTEXPR "--relaxed-constexpr")

endif()

set(NVCC\_ARCH\_FLAGS)

foreach(ARCH IN LISTS EIGEN\_CUDA\_COMPUTE\_ARCH)

string(APPEND NVCC\_ARCH\_FLAGS " -gencode arch=compute\_${ARCH},code=sm\_${ARCH}")

endforeach()

set(CUDA\_NVCC\_FLAGS "${EIGEN\_CUDA\_RELAXED\_CONSTEXPR} -Xcudafe \"--display\_error\_number\" ${NVCC\_ARCH\_FLAGS} ${CUDA\_NVCC\_FLAGS}")

cuda\_include\_directories("${CMAKE\_CURRENT\_BINARY\_DIR}" "${CUDA\_TOOLKIT\_ROOT\_DIR}/include")

set(EIGEN\_ADD\_TEST\_FILENAME\_EXTENSION "cu")

ei\_add\_test(cxx11\_tensor\_complex\_gpu)

ei\_add\_test(cxx11\_tensor\_complex\_cwise\_ops\_gpu)

ei\_add\_test(cxx11\_tensor\_reduction\_gpu)

ei\_add\_test(cxx11\_tensor\_argmax\_gpu)

ei\_add\_test(cxx11\_tensor\_cast\_float16\_gpu)

ei\_add\_test(cxx11\_tensor\_scan\_gpu)

set(EIGEN\_CUDA\_OLDEST\_COMPUTE\_ARCH 9999)

foreach(ARCH IN LISTS EIGEN\_CUDA\_COMPUTE\_ARCH)

if(${ARCH} LESS ${EIGEN\_CUDA\_OLDEST\_COMPUTE\_ARCH})

set(EIGEN\_CUDA\_OLDEST\_COMPUTE\_ARCH ${ARCH})

endif()

endforeach()

# Contractions require arch 3.0 or higher

if (${EIGEN\_CUDA\_OLDEST\_COMPUTE\_ARCH} GREATER 29)

ei\_add\_test(cxx11\_tensor\_device)

ei\_add\_test(cxx11\_tensor\_gpu)

ei\_add\_test(cxx11\_tensor\_contract\_gpu)

ei\_add\_test(cxx11\_tensor\_of\_float16\_gpu)

endif()

# The random number generation code requires arch 3.5 or greater.

if (${EIGEN\_CUDA\_OLDEST\_COMPUTE\_ARCH} GREATER 34)

ei\_add\_test(cxx11\_tensor\_random\_gpu)

endif()

unset(EIGEN\_ADD\_TEST\_FILENAME\_EXTENSION)

endif()

# Add HIP specific tests

if (EIGEN\_TEST\_HIP)

set(HIP\_PATH "/opt/rocm/hip" CACHE STRING "Path to the HIP installation.")

if (EXISTS ${HIP\_PATH})

list(APPEND CMAKE\_MODULE\_PATH ${HIP\_PATH}/cmake)

find\_package(HIP REQUIRED)

if (HIP\_FOUND)

execute\_process(COMMAND ${HIP\_PATH}/bin/hipconfig --platform OUTPUT\_VARIABLE HIP\_PLATFORM)

if ((${HIP\_PLATFORM} STREQUAL "hcc") OR (${HIP\_PLATFORM} STREQUAL "amd"))

include\_directories(${CMAKE\_CURRENT\_BINARY\_DIR})

include\_directories(${HIP\_PATH}/include)

set(EIGEN\_ADD\_TEST\_FILENAME\_EXTENSION "cu")

#

# complex datatype is not yet supported by HIP

# so leaving out those tests for now

#

# ei\_add\_test(cxx11\_tensor\_complex\_gpu)

# ei\_add\_test(cxx11\_tensor\_complex\_cwise\_ops\_gpu)

#

ei\_add\_test(cxx11\_tensor\_reduction\_gpu)

ei\_add\_test(cxx11\_tensor\_argmax\_gpu)

ei\_add\_test(cxx11\_tensor\_cast\_float16\_gpu)

ei\_add\_test(cxx11\_tensor\_scan\_gpu)

ei\_add\_test(cxx11\_tensor\_device)

ei\_add\_test(cxx11\_tensor\_gpu)

ei\_add\_test(cxx11\_tensor\_contract\_gpu)

ei\_add\_test(cxx11\_tensor\_of\_float16\_gpu)

ei\_add\_test(cxx11\_tensor\_random\_gpu)

unset(EIGEN\_ADD\_TEST\_FILENAME\_EXTENSION)

elseif ((${HIP\_PLATFORM} STREQUAL "nvcc") OR (${HIP\_PLATFORM} STREQUAL "nvidia"))

message(FATAL\_ERROR "HIP\_PLATFORM = nvcc is not supported within Eigen")

else ()

message(FATAL\_ERROR "Unknown HIP\_PLATFORM = ${HIP\_PLATFORM}")

endif()

endif()

else ()

message(FATAL\_ERROR "EIGEN\_TEST\_HIP is ON, but the specified HIP\_PATH (${HIP\_PATH}) does not exist")

endif()

endif()