```
# Default implmenentation for CuArrays
using CUDA: @workspace, @argout

memsize(a::CuArray) = sizeof(a)

function add!(α, A::CuArray{<:Any, N}, CA::Symbol,
                β, C::CuArray{<:Any, N}, indCinA) where {N}

    T = eltype(C)
    N == length(indCinA) || throw(IndexError("Invalid permutation of length $N:
$indCinA"))
    CA == :N || CA == :C ||
        throw(ArgumentError("Value of conjA should be :N or :C instead of $CA"))
    opA = (T <: Real || CA == :N) ? CUTENSOR_OP_IDENTITY : CUTENSOR_OP_CONJ
    opAC = CUTENSOR_OP_ADD

    descA = CuTensorDescriptor(A; op = opA)
    descC = CuTensorDescriptor(C; op = CUTENSOR_OP_IDENTITY)
    descD = descC
    typeCompute = convert(cudaDataType, T)
    modeA = collect(Cint, 1:N)
    modeC = collect(Cint, indCinA)
    stream = CuDefaultStream()
    if β == zero(β)
        cutensorPermutation(handle(), T[α], A, descA, modeA, C, descC, modeC,
                            typeCompute, stream)
    else
        cutensorElementwiseBinary(handle(), T[α], A, descA, modeA, T[β], C, descC,
                                  modeC, C, descC, modeC, opAC, typeCompute, stream)
    end

    return C
end

# TODO: the following seems broken due to bug in CUTENSOR?#
# function add!(α, A::CuArray{T, N}, CA::Symbol,
#                 β, C::CuArray{Complex{T}, N}, indCinA) where {T<:CublasReal,N}
#
#     N == length(indCinA) || throw(IndexError("Invalid permutation of length $N:
    $indCinA"))
#     CA == :N || CA == :C ||
#         throw(ArgumentError("Value of conjA should be :N or :C instead of $CA"))
#     opid = CUTENSOR_OP_IDENTITY
#     opAC = CUTENSOR_OP_ADD
#
#     if β == zero(β)
#         fill!(C, zero(Complex{T}))
#     elseif β != one(β)
#         rmul!(C, β)
#     end
#     sizeC = size(C)
#     stridesC = 2 .* strides(C)
#     Cr = reinterpret(T, C, (2*length(C),))
#
```

```
#     @show stridesC, sizeC
#
#     descA = CuTensorDescriptor(A; op = opid)
#     descCr = CuTensorDescriptor(Cr; size = sizeC, strides = stridesC, op = opid)
#     typeCompute = cudaDataType(T)
#     modeA = collect(Cint, 1:N)
#     modeC = collect(Cint, indCinA)
#     stream = CuDefaultStream()
#     cutensorElementwiseBinary(handle(), T[real(α)], A, descA, modeA, T[1], Cr,
    descCr,
#                               modeC, Cr, descCr, modeC, opAC, typeCompute, stream)
#     if imag(α) != 0
#         Ci = view(Cr, 2:length(Cr))
#         descCi = CuTensorDescriptor(Ci; size = sizeC, strides = stridesC, op =
    opid)
#         cutensorElementwiseBinary(handle(), T[imag(α)], A, descA, modeA, T[1],
    Ci, descCi,
#                                   modeC, Ci, descCi, modeC, opAC, typeCompute,
    stream)
#     end
#     return C
# end

function trace!(α, A::CuArray, CA::Symbol, β, C::CuArray,
                indCinA, cindA1, cindA2)

    T = eltype(C)
    NA, NC = ndims(A), ndims(C)
    NC == length(indCinA) ||
        throw(IndexError("Invalid selection of $NC out of $NA: $indCinA"))
    NA-NC == 2*length(cindA1) == 2*length(cindA2) ||
        throw(IndexError("invalid number of trace dimension"))

    opA = (T <: Real || CA == :N) ? CUTENSOR_OP_IDENTITY : CUTENSOR_OP_CONJ
    opReduce = CUTENSOR_OP_ADD

    sizeA = i->size(A, i)
    strideA = i->stride(A, i)
    tracesize = sizeA.(cindA1)
    tracesize == sizeA.(cindA2) || throw(DimensionMismatch("non-matching trace
sizes"))
    size(C) == sizeA.(indCinA) || throw(DimensionMismatch("non-matching sizes"))

    newstrides = (strideA.(indCinA)..., (strideA.(cindA1) .+ strideA.(cindA2))...)
    newsize = (size(C)..., tracesize...)
    descA = CuTensorDescriptor(A; op = opA, size = newsize, strides = newstrides)
    descC = CuTensorDescriptor(C; op = CUTENSOR_OP_IDENTITY)
    descD = descC
    typeCompute = cutensorComputeType(T)
    modeA = collect(Cint, 1:NA)
    modeC = collect(Cint, 1:NC)
    stream = CuDefaultStream()
    @workspace fallback=1<<13 size=@argout(
            cutensorReductionGetWorkspace(handle(),
                A, descA, modeA,
```

```
                C, descC, modeC,
                C, descC, modeC,
                opReduce, typeCompute,
                out(Ref{UInt64}(C_NULL)))
        )[] workspace->begin
            cutensorReduction(handle(),
                T[α], A, descA, modeA,
                T[β], C, descC, modeC,
                        C, descC, modeC,
                opReduce, typeCompute,
                workspace, sizeof(workspace), stream)
        end
    return C
end

function contract!(α, A::CuArray, CA::Symbol,
                   B::CuArray, CB::Symbol,
                   β, C::CuArray,
                   oindA::IndexTuple, cindA::IndexTuple,
                   oindB::IndexTuple, cindB::IndexTuple,
                   indCinoAB::IndexTuple, syms::Union{Nothing, NTuple{3,Symbol}} =
nothing)

    pA = (oindA...,cindA...)
    (length(pA) == ndims(A) && isperm(pA)) ||
        throw(IndexError("invalid permutation of length $(ndims(A)): $pA"))
    pB = (oindB...,cindB...)
    (length(pB) == ndims(B) && isperm(pB)) ||
        throw(IndexError("invalid permutation of length $(ndims(B)): $pB"))
    (length(oindA) + length(oindB) == ndims(C)) ||
        throw(IndexError("non-matching output indices in contraction"))
    (ndims(C) == length(indCinoAB) && isperm(indCinoAB)) ||
        throw(IndexError("invalid permutation of length $(ndims(C)): $indCinoAB"))

    sizeA = i->size(A, i)
    sizeB = i->size(B, i)
    sizeC = i->size(C, i)

    csizeA = sizeA.(cindA)
    csizeB = sizeB.(cindB)
    osizeA = sizeA.(oindA)
    osizeB = sizeB.(oindB)

    csizeA == csizeB ||
        throw(DimensionMismatch("non-matching sizes in contracted dimensions"))
    sizeAB = let osize = (osizeA..., osizeB...)
        i->osize[i]
    end
    sizeAB.(indCinoAB) == size(C) ||
        throw(DimensionMismatch("non-matching sizes in uncontracted dimensions"))

    TC = eltype(C)

    CA == :N || CA == :C ||
        throw(ArgumentError("Value of conjA should be :N or :C instead of $CA"))
```

```
CB == :N || CB == :C ||
    throw(ArgumentError("Value of conjB should be :N or :C instead of $CB"))

opA = (TC <: Real || CA == :N) ? CUTENSOR_OP_IDENTITY : CUTENSOR_OP_CONJ
opB = (TC <: Real || CB == :N) ? CUTENSOR_OP_IDENTITY : CUTENSOR_OP_CONJ
opC = CUTENSOR_OP_IDENTITY

strideA = i->stride(A, i)
strideB = i->stride(B, i)

cstrideA = strideA.(cindA)
cstrideB = strideB.(cindB)
ostrideA = strideA.(oindA)
ostrideB = strideB.(oindB)

descA = CuTensorDescriptor(A; op = opA, size = (osizeA..., csizeA...),
                                  strides = (ostrideA..., cstrideA...))
descB = CuTensorDescriptor(B; op = opB, size = (osizeB..., csizeB...),
                                  strides = (ostrideB..., cstrideB...))
descC = CuTensorDescriptor(C)
T = eltype(C)
typeCompute = cutensorComputeType(T)
opOut = CUTENSOR_OP_IDENTITY

NoA = length(osizeA)
NoB = length(osizeB)
Nc = length(csizeA)
modeoA = ntuple(n->n, NoA)
modeoB = ntuple(n->NoA+n, NoB)
modec = ntuple(n->NoA+NoB+n, Nc)

modeA = collect(Cint, (modeoA..., modec...))
modeB = collect(Cint, (modeoB..., modec...))
modeC = collect(Cint, indCinoAB)

algo = CUTENSOR_ALGO_DEFAULT
stream = CuDefaultStream()
pref = CUTENSOR_WORKSPACE_RECOMMENDED

alignmentRequirementA = Ref{UInt32}(C_NULL)
cutensorGetAlignmentRequirement(handle(), A, descA, alignmentRequirementA)
alignmentRequirementB = Ref{UInt32}(C_NULL)
cutensorGetAlignmentRequirement(handle(), B, descB, alignmentRequirementB)
alignmentRequirementC = Ref{UInt32}(C_NULL)
cutensorGetAlignmentRequirement(handle(), C, descC, alignmentRequirementC)
desc = Ref(cutensorContractionDescriptor_t(ntuple(i->0, Val(256))))
cutensorInitContractionDescriptor(handle(),
                                  desc,
                descA, modeA, alignmentRequirementA[],
                descB, modeB, alignmentRequirementB[],
                descC, modeC, alignmentRequirementC[],
                descC, modeC, alignmentRequirementC[],
                typeCompute)

find = Ref(cutensorContractionFind_t(ntuple(i->0, Val(64))))
```

```
    cutensorInitContractionFind(handle(), find, algo)

    @workspace fallback=1<<27 size=@argout(
            cutensorContractionGetWorkspace(handle(), desc, find, pref,
                                            out(Ref{UInt64}(C_NULL)))
        )[] workspace->begin
            plan = Ref(cutensorContractionPlan_t(ntuple(i->0, Val(640))))
            cutensorInitContractionPlan(handle(), plan, desc, find,
sizeof(workspace))

            cutensorContraction(handle(), plan, T[α], A, B, T[β], C, C,
                                workspace, sizeof(workspace), stream)
        end

    return C
end

# overwrite similar_from_indices to return zero initialized arrays
function similar_from_indices(T::Type, ind::IndexTuple, A::CuArray, CA::Symbol)
    sz = similarstructure_from_indices(T, ind, A, CA)
    return fill!(similar(A, T, sz), zero(T))
end
function similar_from_indices(T::Type, poA::IndexTuple, poB::IndexTuple,
                              p1::IndexTuple, p2::IndexTuple,
                              A::CuArray, B::CuArray, CA::Symbol, CB::Symbol)
    sz = similarstructure_from_indices(T, poA, poB, p1, p2, A, B, CA, CB)
    return fill!(similar(A, T, sz), zero(T))
end

# overwrite cached_similar_from_indices in order not to use cache for CuArray
  objects
function cached_similar_from_indices(sym::Symbol, T::Type, p1::IndexTuple,
p2::IndexTuple, A::CuArray, CA::Symbol)
    # also zero fill to avoid problems with cutensorElementwiseBinary
    return similar_from_indices(T, p1, p2, A, CA)
end

function cached_similar_from_indices(sym::Symbol, T::Type, poA::IndexTuple,
poB::IndexTuple,
    p1::IndexTuple, p2::IndexTuple, A::CuArray, B::CuArray, CA::Symbol, CB::Symbol)
    # also zero fill to avoid problems with cutensorElementwiseBinary
    return similar_from_indices(T, poA, poB, p1, p2, A, B, CA, CB)
end
```