miCore Guide

v1.0-beta

03/25/2020 by XiuYuLi

Description:

miCore is a high performance computing library used assembly to deep-optimizated for machine-learning on gfx9xx and later architectures; miCore API is thread-safe.

micore_status_t micore_create_handle(micore_handle_t* p_result)

: create micore context handle, must be called beforce any other micore API.

micore_tensorshape_t micore_create_tensorshape()

: create tensorshape, can be reused.

micore_parambox_t micore_create_parambox()

: create parameter box, only need create once and be reused.

void micore_set_tensorshape4d(

micore_tensorshape_t shape,

uint32_t nx,

uint32_t ny,

uint32_t nc,

uint32_t bs)

: set values of 'shape' which be created by 'micore_create_tensorshape', only used for data tensor.

void micore_set_tensorshape4d_filter(

uint32_t nx,

uint32_t ny,

uint32_t pnc,

uint32_t qnc)

: set values of 'shape' which be created by 'micore_create_tensorshape_filter', only used for filter tensor.

micore_padding_t micore_make_padding2d(

uint32_t pl,

uint32_t pr,

uint32_t pt,

uint32_t pb)

: Make paddings, 'pl' means left-padding size, 'pr' means right-padding size, 'pt' means top-padding size, 'pb' means bottom-padding-size, padding size must be <= filter size-1.

micore_stride_t micore_make_stride2d(uint32_t su, uint32_t sv)

: Make strides, start form zero, this means when stride is UxV, you must call 'micore_make_stride2d(U-1, V-1).

micore_dilation_t micore_make_dilation2d(uint32_t du, uint32_t dv)

: Make dilations, same policy as stride.

micore_status_t micore_build_parambox_fconv(

micore_parambox_t	param,
micore_conv_algo_t	algo,
uint32_t	mask,
micore_tensorshape_t	pshape,
micore_tensorshape_t	fshape,
micore_tensorshape_t	qshape,

micore_padding_t pad,

micore_stride_t str,

micore_dilation_t dla,

uint32_t ng)

:Build parambox named 'param' of FWD, 'pshape' is input-data shape of forward, 'fshape' is shape of filter, 'qshape' is output-data shape of forward, 'ng' is number groups, 'mask must be 'miCoreMaskPrecisionF32' now;

'algo' is micore_conv_algo_flexgemm' or 'micore_conv_algo_cell'(winograd) or 'micore_conv_algo_cellfft';

'micore_conv_algo_flexgemm' support group-conv, unit-strides, non-unit-strides, dilations and padding;

'micore_conv_algo_cell' only support for '3x3' filter with unit-strides and non-dilation,support group and padding, padding must be <=2;

'micore_conv_algo_cellfft' support padding, unit-strides;

'mask' must be 'micoreMaskPrecisionF32'.

micore_status_t micore_build_parambox_bconv(

micore_parambox_t param,

micore_conv_algo_t algo,

uint32_t mask,

micore_tensorshape_t pshape,

micore_tensorshape_t fshape,

micore_tensorshape_t qshape,

micore_padding_t pad,

micore_stride_t str,

micore_dilation_t dla,

uint32_t ng)

:Build parambox named 'param' of BWD, 'pshape' is input-data shape of forward, 'fshape' is shape of filter, 'qshape' is output-data shape of forward, 'ng' is number groups, 'mask must be 'miCoreMaskPrecisionF32' now;

'algo' is micore_conv_algo_flexgemm' or 'micore_conv_algo_cell'(winograd) or 'micore_conv_algo_cellfft';

'micore_conv_algo_flexgemm' support group, unit-strides and padding;

'micore_conv_algo_cell' only support for '3x3' filter with unit-strides and non-dilation, support group and padding, padding must be <=2;

'micore_conv_algo_cellfft' support padding, unit-strides;

'mask' must be 'micoreMaskPrecisionF32'.

```
micore_status_t micore_build_parambox_gconv(
micore_parambox_t       param,
micore_conv_algo_t      algo,
uint32_t                mask,
micore_tensorshape_t    pshape,
micore_tensorshape_t    fshape,
micore_tensorshape_t    qshape,
micore_padding_t        pad,
micore_stride_t         str,
micore_dilation_t       dla,
uint32_t                ng )
```

:Build parambox named 'param' of BWW, 'pshape' is input-data shape of forward, 'fshape' is shape of filter, 'qshape' is output-data shape of forward, 'ng' is number groups, 'mask must be 'miCoreMaskPrecisionF32' now;

'algo' is 'micore_conv_algo_cellfft' only now;

'micore_conv_algo_cellfft' support padding, unit-strides;

'mask' must be 'micoreMaskPrecisionF32'.

```
size_t micore_get_auxsize( micore_parambox_t param )
```

: Get aux-buffer size of 'param' (same means with 'worksapce').

```
void micore_fconv(
micore_handle_t     h,
micore_parambox_t   param,
```

```
void*           aux,
void*           dst,
const void*     src,
const void*     fil,
const void*     bias,
float           alpha,
uint32_t        mask,
hipStream_t     s
)
```

: FWD-conv routine, 'bias' not support and must be set 'NULL' now, 'mask' must be 'miCoreMaskActivationRelu' or '0'.

```
void micore_bconv(
micore_handle_t     h,
micore_parambox_t   param,
void*               aux,
void*               dst,
const void*         src,
const void*         fil,
float               alpha,
hipStream_t         s
)
```

: BWD-conv routine.

```
void micore_gconv(
micore_handle_t     h,
micore_parambox_t   param,
void*               aux,
void*               output_grad,
```

const void* pdata,

const void* qdata,

float alpha,

hipStream_t s

)

: BWW-conv, now only supported with FFT algorithm.

void micore_release_tensorshape(micore_tensorshape_t)

: Release tensorshape.

void micore_release_parambox(micore_parambox_t)

: Release parambox.

void micore_release_handle(micore_handle_t)

:Release micore handle.

Convention:

miCore only checks the most basic errors. Other errors need to be used strictly according to the limits of micore. Otherwise, the result is unknown, the following are the situations that need to be noted that miCore does not support yet:

0 : each single buffer size must be <=4GB with 'flexgemm' algorithm

1 : 'flexgemm'&'cell' not support non-unit-strides&dilation BWD

2 : 'flexgemm' not support BWW

3 : 'cell' not support non-unit-strides&dilation FWD&BWD

4 : 'cell' not support BWW-conv

5 : 'cellfft' not support group, non-unit-strides and dilations

Examples:

```
.....
micore_handle_t handle;
```

```
if(micore_create_handle( &handle )!=micore_success){
        printf( "error : micore init failed!\n" );
        exit(0);
 }

void *d_a, *d_b, *d_c, *d_aux;

uint32_t ng     =1;
uint32_t anx    =54;
uint32_t any    =54;
uint32_t bnx    =3 ;
uint32_t bny    =3 ;
uint32_t cnx    =p*2+anx-bnx+1;
uint32_t cny    =p*2+any-bny+1;
uint32_t pnc    =128;
uint32_t qnc    =256;
uint32_t bs     =64;

micore_tensorshape_t        Sa      =micore_create_tensorshape();
micore_tensorshape_t        Sb      =micore_create_tensorshape();
micore_tensorshape_t        Sc      =micore_create_tensorshape();
micore_parambox_t           param   =micore_create_parambox();
micore_padding_t            pad     =micore_make_padding2d(1,1,1,1);
micore_set_tensorshape4d( Sa, anx, any, pnc, bs );
micore_set_tensorshape4d( Sc, cnx, cny, qnc, bs );
micore_set_tensorshape4d_filter( Sb, bnx, bny, pnc, qnc );
hipMalloc((void**)&d_a, anx*any*pnc*bs *ng*sizeof(float));
hipMalloc((void**)&d_b, bnx*bny*pnc*qnc*ng*sizeof(float));
hipMalloc((void**)&d_c, cnx*cny*qnc*bs *ng*sizeof(float));
```

```
size_t auxsize=micore_get_auxsize( param );
if(auxsize>0){
        hipMalloc((void**)&d_aux, auxsize);
 }
......
if(micore_build_parambox_fconv( param, micore_conv_algo_cell,
miCoreMaskPrecisionF32, Sa, Sb, Sc, pad, str, dla, ng )!=micore_success){
        printf( "error : param of fwd build failed!\n" );
        micore_release_tensorshape( Sa );
        micore_release_tensorshape( Sb );
        micore_release_tensorshape( Sc );
        micore_release_parambox( param );
        micore_release_handle( handle );
        exit(0);
 }
micore_fconv( handle, param, d_aux, d_c, d_a, d_b, NULL, 1.f,
miCoreMaskActivationRelu, 0 );
......
if(micore_build_parambox_bconv( param, micore_conv_algo_cell,
miCoreMaskPrecisionF32, Sa, Sb, Sc, pad, 0, 0, ng )!=micore_success){
        printf( "error : param of bwd build failed!\n" );
        hipFree(d_a);
        hipFree(d_b);
        hipFree(d_c);
        micore_release_tensorshape( Sa );
        micore_release_tensorshape( Sb );
        micore_release_tensorshape( Sc );
        micore_release_parambox( param );
        micore_release_handle( handle );
        exit(0);
```

```
 }
micore_bconv( handle, param, d_aux, d_a, d_c, d_b, 1.f, 0 );
......
hipFree(d_a);
hipFree(d_b);
hipFree(d_c);
if(auxsize>0){ hipFree(d_aux); }
micore_release_tensorshape( Sa );
micore_release_tensorshape( Sb );
micore_release_tensorshape( Sc );
micore_release_parambox( param );
micore_release_handle( handle );
......
```