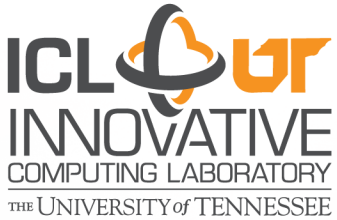
Creating a new operation with DPLASMA: a step by step guide

Anthony Danalis

ICL Lunch Talk

July 19th, 2013



PaRSEC & DPLASMA

- PaRSEC is the runtime engine
  - It can be used to implement affine DLA operations
  - Or any operation that can be expressed as a static PTG
  - Or any operation that has a fixed DAG upon startup
  - Or anything else!

- DPLASMA is PLASMA implemented on top of PaRSEC
  - It's just sweet.

PaRSEC team and (re)sources

- ***George Bosilca***
- ***Aurelien Bouteiller***
- ***Anthony Danalis***
- ***Jack Dongarra***
- Mathieu Faverge
- ***Thomas Herault***
- *Pierre Lemarinier*
- **Stephanie Moreaud**

ICL project site:
icl.cs.utk.edu/projectsdev/parsec/

Mercurial repository:
bitbucket.org/bosilca/dplasma

ICL UT

Getting started

- Download icl.cs.utk.edu/projectsfiles/dague/dplasma-1.0.0.tar.bz2
- Top level view:
  - CMakeLists.txt
  - CTestConfig.cmake
  - Doxyfile
  - License.txt
  - cmake_modules
  - contrib
  - data_dist
  - **dplasma**
  - include
  - src
  - tests
  - tools

- CMakeLists.txt
- cores
- include
- lib
- testing

Example serial code

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    for (k=0; k<N; k++)
      c[i][j] += a[i][k] * b[k][j];
```

Code after tiling (via pluto)

```
for ( t2=0; t2 <= floord(N-1,32); t2++ )
    for ( t3=0; t3<=floord(N-1,32); t3++ )
        for ( t4=0; t4<=floord(N-1,32); t4++ )
            for ( t5=32*t2; t5<=min(N-1,32*t2+31); t5++ )
                for ( t6=32*t3; t6<=min(N-1,32*t3+31); t6++ )
                    for ( t7=32*t4; t7<=min(N-1,32*t4+31); t7++ )
                        c[t5][t6] += a[t5][t7] * b[t7][t6];
```

ICL UT

Code after tiling (via pluto)

```
for ( t2=0; t2 <= floord(N-1,32); t2++ )
    for ( t3=0; t3<=floord(N-1,32); t3++ )
        for ( t4=0; t4<=floord(N-1,32); t4++ )
            for ( t5=32*t2; t5<=min(N-1,32*t2+31); t5++ )
                for ( t6=32*t3; t6<=min(N-1,32*t3+31); t6++ )
                    for ( t7=32*t4; t7<=min(N-1,32*t4+31); t7++ )
                        c[t5][t6] += a[t5][t7] * b[t7][t6];
```

Tile

Tiled based kernel

```
for ( t2=0; t2 <= floord(N-1,32); t2++ )
    for ( t3=0; t3<=floord(N-1,32); t3++ )
        for ( t4=0; t4<=floord(N-1,32); t4++ )
            zmm(a, b, c, t2, t3, t4, N);
```

Updating files

- Code of zmm() goes in dplasma/cores/core_zmm.c
- Prototype in dplasma/cores/dplasma_zcores.h
- dplasma/cores/CMakeLists.txt needs updating:

```
set(SOURCES
  core_ztrdv.c
  core_zmm.c
  core_zhetrf2_nopiv.c
  core_zhebut.c
  ${GPU_KERNEL_SOURCES}
)
```

Calling function: zgemm

```
int zgemm(PLASMA_desc a, PLASMA_desc b,
                  PLASMA_desc c, int N_sz, int ub )
{
    for (t2=0;t2<=ub;t2++)
        for (t3=0;t3<=ub;t3++)
            for (t4=0;t4<=ub;t4++)
                Insert_Task(zmm);
}
```

Calling function: zgemm

```
int zgemm(PLASMA_desc a, PLASMA_desc b,
                PLASMA_desc c, int N_sz, int ub )
{
    for (t2=0;t2<=ub;t2++)
        for (t3=0;t3<=ub;t3++)
            for (t4=0;t4<=ub;t4++)
                Insert_Task(zmm);
}
```

This code resides in a temporary file, say zgemm.c

Calling function: zgemm

```
int zgemm(PLASMA_desc a, PLASMA_desc b,
                PLASMA_desc c, int N_sz, int ub )
{
    for (t2=0;t2<=ub;t2++)
        for (t3=0;t3<=ub;t3++)
            for (t4=0;t4<=ub;t4++)
                Insert_Task(zmm);
}
```

This code resides in a temporary file, say zgemm.c

Insert_Task() Quark Interface

```
QUARK_Insert_Task(junk, CORE_zmm_quark, junk,
                  junk, a[t2][t4], INPUT,
                  junk, b[t4][t3], INPUT,
                  junk, c[t2][t3], INOUT,
                  junk, &(t2),     VALUE,
                  junk, &(t3),     VALUE,
                  junk, &(t4),     VALUE,
                  junk, &(N_sz),   VALUE,
                  0);
```

Insert_Task() Generic Interface

```
Insert_Task(zmm,
            a[t2][t4], INPUT,
            b[t4][t3], INPUT,
            c[t2][t3], INOUT,
            t2,        VALUE,
            t3,        VALUE,
            t4,        VALUE,
            N_sz,      VALUE);
```

New Quark to Old Quark

For Dyn. Sc. Quark codes, use Mathieu's "plasma2parsec.sh" & "core.h" (on zgemm.c)

Example from core.h:

```
#define QUARK_CORE_zssssm(quark, task_flags, m1, n1, m2, n2, k, ib, nb, A1, lda1, A2, lda2, L1, ldl1, L2, ldl2, IPIV) {\
    QUARK_Insert_Task((quark), CORE_zssssm_quark, (task_flags),\
        sizeof(int),                        &(m1),    VALUE,\
        sizeof(int),                        &(n1),    VALUE,\
        sizeof(int),                        &(m2),    VALUE,\
        sizeof(int),                        &(n2),    VALUE,\
        sizeof(int),                        &(k),     VALUE,\
        sizeof(int),                        &(ib),    VALUE,\
        sizeof(PLASMA_Complex64_t)*nb*nb,    (A1),             INOUT,\
        sizeof(int),                        &(lda1),  VALUE,\
        sizeof(PLASMA_Complex64_t)*nb*nb,    (A2),             INOUT | LOCALITY,\
        sizeof(int),                        &(lda2),  VALUE,\
        sizeof(PLASMA_Complex64_t)*ib*nb,    (L1),             INPUT,\
        sizeof(int),                        &(ldl1),  VALUE,\
        sizeof(PLASMA_Complex64_t)*ib*nb,    (L2),             INPUT,\
        sizeof(int),                        &(ldl2),  VALUE,\
        sizeof(int)*nb,                      (IPIV),           INPUT,\
        0);}
```

Customization

- You can name the matrix variables

  #pragma zssssm A1 A2 L1 L2 IPIV

- You can provide additional conditions

  #pragma PARSEC_INVARIANT B.mt==minMT

- Invariant expression may contain:

  `||, &&, ==, !=, <=, <, (, ), +, -, vars, ints`

JDF Generation

- tools/q2j/q2j –anti zgemm.c > zgemm.jdf
- q2j parameters:
  - -shmem
  - -phony_tasks
  - -line_numbers
  - -anti
  - -v
  - file_1.c [file_2.c ... file_N.c]

JDF Generation

- tools/q2j/q2j –anti zgemm.c > zgemm.jdf
- q2j parameters:
  - -**shmem** (controls read/write pseudotasks)
  - -**phony_tasks** (controls phony IN/OUT tasks)
  - -**line_numbers** (adds #line in body)
  - -**anti** (finalizes antidependencies – can be very slow)
  - -**v** (more verbose warnings, slightly more)
  - **file_1.c [file_2.c ... file_N.c]** (input file(s))

ICL UT

.q2jrc

- generate_line_numbers = 0
- produce_shmem_jdf = 0
- finalize_antideps = 0
- verbose_warnings = 0
- add_phony_tasks= 0
- direct_output = 0

Q2J limitations

- Codes must be affine*
- Loops must be somewhat canonical
  - (i++, not i+=7)
  - Induction variables only in loop headers
    - No for(i=0…){ j++; A[i][j] =… }
- General format must look like PLASMA/Quark codes
- Codes must be self-contained*

Not self-contained codes

```
PLASMA_zgelqs_Tile_Async(){
   plasma_parallel_call_3(plasma_pztile_zero,
      PLASMA_desc, plasma_desc_submatrix(descB, descA.m, 0, descA.n-descA.m, descB.n),
      PLASMA_sequence*, sequence,
      PLASMA_request*, request);

   plasma_parallel_call_9(plasma_pztrsm,
      PLASMA_enum, PlasmaLeft,
      PLASMA_enum, PlasmaLower,
      PLASMA_enum, PlasmaNoTrans,
      PLASMA_enum, PlasmaNonUnit,
      PLASMA_Complex64_t, 1.0,
      PLASMA_desc, plasma_desc_submatrix(descA, 0, 0, descA.m, descA.m),
      PLASMA_desc, plasma_desc_submatrix(descB, 0, 0, descA.m, descB.n),
      PLASMA_sequence*, sequence,
      PLASMA_request*, request);
}
```

plasma_desc_submatrix()

```
PLASMA_desc plasma_desc_submatrix(PLASMA_desc descA, int i, int j, int m, int n)
{
    PLASMA_desc descB;
    int mb, nb;

    descB = descA;
    mb = descA.mb;
    nb = descA.nb;
    // Submatrix parameters
    descB.i = i;
    descB.j = j;
    descB.m = m;
    descB.n = n;
    // Submatrix derived parameters
    descB.mt = (i+m-1)/mb - i/mb + 1;
    descB.nt = (j+n-1)/nb - j/nb + 1;
    return descB;
}
```

plasma_desc_submatrix()

plasma_desc_submatrix(descB, descA.m, 0, descA.n-descA.m, descB.n)

```
PLASMA_desc plasma_desc_submatrix(PLASMA_desc descA, int i, int j, int m, int n)
{
    PLASMA_desc descB;
    int mb, nb;

    descB = descA;
    mb = descA.mb;
    nb = descA.nb;
    // Submatrix parameters
    descB.i = i;
    descB.j = j;
    descB.m = m;
    descB.n = n;
    // Submatrix derived parameters
    descB.mt = (i+m-1)/mb - i/mb + 1;
    descB.nt = (j+n-1)/nb - j/nb + 1;
    return descB;
}
```

plasma_desc_submatrix()

plasma_desc_submatrix(descB, descA.m, 0, descA.n-descA.m, descB.n)

PLASMA_desc plasma_desc_submatrix(PLASMA_desc descA, int i, int j, int m, int n)
{
    descB = descA;
    mb = descA.mb;
    nb = descA.nb;
    descB.i = descA.m;
    descB.j = 0;
    descB.m = descA.n-descA.m;
    descB.n = descB.n;
    descB.mt = (descA.m+descA.n-descA.m-1)/descA.mb – descA.m/descA.mb + 1;
    descB.nt = (0+descB.n-1)/descA.nb - 0/descA.nb + 1;
}

plasma_desc_submatrix()

plasma_desc_submatrix(descB, descA.m, 0, descA.n-descA.m, descB.n)

PLASMA_desc plasma_desc_submatrix(PLASMA_desc descA, int i, int j, int m, int n)
{
    descB = descA;
    mb = descA.mb;
    nb = descA.nb;
    descB.i = descA.m;
    descB.j = 0;
    descB.m = descA.n-descA.m;
    descB.n = descB.n;
    descB.mt = (descA.n-1)/descA.mb – descA.m/descA.mb + 1;
    descB.nt = (descB.n-1)/descA.nb + 1;
}

Proposed extension

```
plasma_desc_submatrix_affine(PLASMA_desc descA,
                             int i,
                             int j,
                             int m,
                             int n,
                             int mt
                             int nt
                            );
```

Backend compilation daguepp

- JDF compiler: tools/dague-compiler/daguepp
- daguepp -o zgemm < zgemm.jdf
- daguepp will generate zgemm.c and zgemm.h
- .jdf, .c, and .h files should go in dplasma/lib
- We will also place the wrapper in dplasma/lib

Generated zgemm.h file

```
#define DAGUE_zgemm_DEFAULT_ARENA   0
#define DAGUE_zgemm_ARENA_INDEX_MIN 1
typedef struct dague_zgemm_object {
    dague_object_t super;
    dague_ddesc_t *dataa
    tiled_matrix_desc_t desca;
    dague_ddesc_t *datab
    tiled_matrix_desc_t descb;
    dague_ddesc_t *datac
    tiled_matrix_desc_t descc;
    int N_sz;
    int ub;
    dague_arena_t** arenas;
    int arenas_size;
} dague_zgemm_object_t;
extern dague_zgemm_object_t *dague_zgemm_new(dague_ddesc_t *dataa,
                                             dague_ddesc_t *datab,
                                             dague_ddesc_t *datac,
                                             int N_sz, int ub);
```

zgemm_wrapper.c NB interface

```
#include "zgemm.h"
dague_object_t *dplasma_zgemm_New( tiled_matrix_desc_t * desca,
                                   tiled_matrix_desc_t * descb,
                                   tiled_matrix_desc_t * descc,
                                   int N, int ub )
{
    dague_zgemm_object_t *o = dague_zgemm_new((dague_ddesc_t*)desca,
                                              (dague_ddesc_t*)descb,
                                              (dague_ddesc_t*)descc,
                                              N, ub);

    dplasma_add2arena_tile( o->arenas[DAGUE_zgemm_DEFAULT_ARENA],
                            desca->mb*desca->nb*sizeof(double),
                            DAGUE_ARENA_ALIGNMENT_SSE,
                            MPI_DOUBLE_COMPLEX, desca->mb );
    return (dague_object_t*)o;
}
```

zgemm_wrapper.c NB interface

```
void dplasma_zgemm_Destruct(dague_object_t *object)
{
    dague_zgemm_object_t *o = (dague_zgemm_object_t *)object;
    dplasma_datatype_undefine_type(
                &(o->arenas[DAGUE_zgemm_DEFAULT_ARENA]->opaque_dtt) );
    DAGUE_INTERNAL_OBJECT_DESTRUCT(o);
}
```

ICL UT

zgemm_wrapper.c B interface

```
int dplasma_zgemm(dague_context_t *dague,
                  tiled_matrix_desc_t * desca,
                  tiled_matrix_desc_t * descb,
                  tiled_matrix_desc_t * descc,
                  int N, int ub {

    dague_handle_t *dague_zgemm = NULL;
    dague_zgemm = dplasma_zgemm_New(ddesca, descb, descc, N, ub);

    if ( dague_zgemm != NULL ) {
        dague_enqueue( dague, (dague_handle_t*)dague_zgemm);
        dplasma_progress(dague);
        dplasma_zgemm_Destruct( dague_zgemm );
    }
```

Updating files

dplasma/lib/CMakeLists.txt

```
set(JDF
  zprint.jdf
  zgemm.jdf
  zlansy.jdf
)

set(SOURCES
 zprint_wrapper.c
 zgemm_wrapper.c
 zunmqr_wrapper.c
)
```

Updating files

Add wrapper prototypes in dplasma/include/dplasma_z.h

```
dague_object_t *dplasma_zgemm_New( tiled_matrix_desc_t * desca,
                                   tiled_matrix_desc_t * descb,
                                   tiled_matrix_desc_t * descc,
                                   int N, int ub )
void dplasma_zgemm_Destruct(dague_object_t *object)

int dplasma_zgemm(dague_context_t *dague,
                  tiled_matrix_desc_t * desca,
                  tiled_matrix_desc_t * descb,
                  tiled_matrix_desc_t * descc,
                  int N, int ub );
```

Writing testing_zgemm.c

```
#include "common.h"
#include "data_dist/matrix/two_dim_rectangle_cyclic.h”
int main(int argc, char ** argv)
{
    dague_context_t* dague;
    int iparam[IPARAM_SIZEOF];
    int ub;

    iparam_default_gemm(iparam);
    iparam_default_ibnbmb(iparam, 0, 32, 32);

    /* Initialize DAGuE */
    dague = setup_dague(argc, argv, iparam);
    PASTE_CODE_IPARAM_LOCALS(iparam);
    ub = floord(N-1,32);
```

testing_zgemm.c cont.

```
PASTE_CODE_ALLOCATE_MATRIX(ddescA, 1,
      two_dim_block_cyclic, (&ddescA, matrix_ComplexDouble, matrix_Tile,
                             nodes, cores, rank, MB, NB, M, N, 0, 0,
                             M, N, SMB, SNB, P));

dplasma_gemm(dague,
             (tiled_matrix_desc_t*)&ddescA,
             (tiled_matrix_desc_t*)&ddescB,
             (tiled_matrix_desc_t*)&ddescC,
             N,
             ub );

   dague_data_free(ddescA.mat);
   dague_ddesc_destroy((dague_ddesc_t*)&ddescA);
   cleanup_dague(dague, iparam);
```

Updating files

- Main (testing) program goes into dplasma/testing
- dplasma/testing/CMakeLists.txt needs updating:

  set(TESTS

  testing_zgemm.c
- tools/precision_generator/subs.py needs updating:

  ('sgemm','dgemm','cgemm','zgemm')

Obligatory voodoo

- Precision generator looks for following header:

```
/*
 * @precisions normal z -> s d c
 */
```

Compile & run

- cd dplasma/testing; make -j testing_dgemm
- mpirun -np 4 ./testing_dgemm –N 128 -t 32 -c 1 -p 4 -q 1

- For help on default parameters:

  ./testing_dgemm –h

Summary

- Source code: affine loops with “tiled” kernels
  - Quark or Generic interface for declaring tasks
- tools/q2j/q2j –anti zmyop.c > dplasma/lib/zmyop.jdf
- tools/dague-compiler/daguepp –o zmyop < zmyop.jdf
- Write wrapper and main()
  - dplasma/lib/zmyop_wrapper.c
  - dplasma/testing/tesing_zmyop.c
- Update */CMakeLists.txt, dplasma_z.h, subs.py