PLASMA Runtime Interface

Sicong Zhuang
BSC

March 2, 2015

1 Overview

Part of the PLASMA functionality has been ported to OmpSs. An intermediate layer (src/runtime) is added so that users can select a runtime mode (currently, QUARK or OmpSs) prior to the execution.

The functions ported so far are (available in the SVN repo: plasma-runtimes): **dgecon, dlacpy, dlaset, dorgqr, dgetrf, dasum, dlange, dgeqrf, dgemm, dsyrk, dtrsm, dpotrf, dooplap2tile, dooptile2lap**.

2 Usage

To toggle between QUARK and OmpSs:

First, select the dynamic runtime:

```
PLASMA_Set(PLASMA_SCHEDULING_MODE, PLASMA_DYNAMIC_SCHEDULING);
```

Subsequently, select the OmpSs runtime:

```
PLASMA_Set(PLASMA_RUNTIME_MODE, PLASMA_OMPSS);
```

or the QUARK runtime

```
PLASMA_Set(PLASMA_RUNTIME_MODE, PLASMA_QUARK);
```

The call semantic to the computation functions are exactly the same.

3 Porting new functions

- A new member is added to plasma_context_t struct - PLASMA_enum runtime - to indicate the runtime mode being used (control/context.h:25).
- Because the synchronization mechanism under OmpSs is different from that of QUARK, in every function to be ported plasma_dynamic_sync() should be replaced by RT_dynamic_sync(); e.g. in PLASMA_dgemm (compute/dgemm.c)

```
198     plasma_dooptile2lap( descC, C, NB, NB, LDC, N,  sequence, &request);
199     RT_dynamic_sync();
```

- In the actual computation function replace the call to QUARK_CORE_? with RT_CORE_?. Continue with the example of PLASMA_dgemm, in the function plasma_pdgemm_quark, RT_CORE_dgemm substitutes the previous QUARK_CORE_dgemm (compute/pdgemm.c).

- Implement the RT_CORE_? in the src/runtime directory. A partial code in runtime/rt_dgemm.c is given as a template

```
3   void RT_CORE_dgemm(Quark *quark, Quark_Task_Flags *task_flags,
4     PLASMA_enum transA, int transB,
5     int m, int n, int k, int nb,
6     double alpha, const double *A, int lda,
7     const double *B, int ldb,
8     double beta, double *C, int ldc) {
9     plasma_context_t *plasma;
10    plasma = plasma_context_self();
11    if (plasma->runtime == PLASMA_QUARK) {
12      QUARK_CORE_dgemm(quark, task_flags, transA, transB,
13        m, n, k, nb, alpha, A, lda, B, ldb, beta, C, ldc);
14    } else if (plasma->runtime == PLASMA_OMPSS) {
15      #pragma omp task in([nb*nb]A, [nb*nb]B) inout([nb*nb]C)
16      CORE_dgemm( transA, transB,
17        m, n, k,
18        (alpha), A, lda,
19        B, ldb, (beta),
20        C, ldc);
21    }
22  }
```

The code fragment below is an SPD solver (testing/testing_dspdsolv.c) which serves as an example to illustrate the usage of PLASMA-Runtime in a user program. It solves a linear system A x = B by first factorizing A into LL' and consequently using forward-substitute and backward-substitute to solve x.

```
62  PLASMA_Set( PLASMA_RUNTIME_MODE, PLASMA_OMPSS); //Toggle runtime mdoe to OmpSs
63  PLASMA_Set( PLASMA_TILE_SIZE, 384); //Set up tile size
64
65  GENMAT_SYM_FULL(N, A); //Initialize A with a SPD matrix
66
67  LAPACKE_dlarnv(IONE, ISEED, LDA*N, X); //Generate X for verification
68
69  cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, N, N, N,
70          1.0, A, LDA, X, LDA, 0.0, B, LDA);
71
72  PLASMA_dpotrf(PlasmaLower, N, A, LDA);
73  PLASMA_dtrsm(PlasmaLeft, PlasmaLower, PlasmaNoTrans, PlasmaNonUnit, N, N, 1.0, A, LDA, B, LDA);
74  PLASMA_dtrsm(PlasmaLeft, PlasmaLower, PlasmaTrans, PlasmaNonUnit, N, N, 1.0, A, LDA, B, LDA);
```