Pacific Northwest
NATIONAL LABORATORY

Programming FPGA

Roberto Gioiosa

U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Programming Extremely Heterogeneous Systems

- What MCL is not for:
  - Programming single-device systems
    - ✓ Can still make advantage of asynchronous task execution
    - ✓ Simplified programming model
    - ✓ Incur in scheduling and abstraction overhead
  - Programming single-kernel applications
    - ✓ No opportunity to leverage asynchronous execution and multiple devices
- What MCL is really for:
  - Programming multi-device, multi-device class systems (**extremely heterogeneous systems**)
    - ✓ Automatic scaling out and management of heterogeneous resources
  - Programming applications with complex dependencies and many tasks
    - ✓ Relieve programmers from tracking dependencies
    - ✓ Relieve programmers from assigning tasks to resources and track data dependencies
  - Programming complex workflows on heterogeneous systems

Extremely Heterogeneous Sytems: PNNL Junction

- Compute cluster
  - 48 nodes
  - Each node consists of:
    - ✓ 2x AMD CPU
    - ✓ 1x Xilinx Versal
    - ✓ 1x AMD GPU
    - ✓ 1x Xilinx SmartNIC
- Current status
  - AMD CPU
  - AMD GPU
  - Xilinx Versal
  - Xilinx SmarNIC

A test case: NWChem-Proxy

- CCSD(1) method from NWChem
  - Coupled cluster (CC) methods are commonly used in the post Hartree-Fock ab initio quantum chemistry and in nuclear physics computation.
  - The CC workflow is composed of iterative set of excitation (singles (S), doubles (D), triples (T), and quadruples (Q)) calculations
- Tensor Contractions are the main computational kernels:
  - Often reformulated as TTGT to take advantage of high-performance GEMM kernels
- Testbed:
  - NVIDIA DGX-1 V100
  - 2x Intel Xeon E5-2680, 768GB memory
  - 8x NVIDIA V100, 16GM memory, NVLink

```cpp
#include <iostream>
#include "taco.h"
#include "utils.h"

using namespace taco;

int main(int argc, char* argv[]) {
  if (argc != 2){
    std::cout << "Please enter input problem size" << "\n";
    exit(1);
  }

  int idim = atoi(argv[1]);

  Format csr({Dense,Sparse});
  Format csf({Sparse,Sparse,Sparse});
  Format  sv({Sparse});

  Format dense2d({Dense,Dense});
  Format dense4d({Dense,Dense, Dense, Dense});

  Tensor<double> i0("i0", {idim,idim}, dense2d);
  Tensor<double> F("F", {idim, idim}, dense2d);
  Tensor<double> V("V", {idim, idim, idim, idim}, dense4d);
  Tensor<double> t1("t1", {idim,idim}, dense2d);
  Tensor<double> t2("t2", {idim, idim, idim, idim}, dense4d);

// Initialization...

  IndexVar i, m, n, a, e, f;

  std::cout << "Computation started" << "\n";
  i0(a, i) = F(a, i);                                                          //#1
  i0(a, i) +=  -2.0 * F(m, e) * t1(a, m) * t1(e, i) +  F(a, e) * t1(e, i);    //#2
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t2(a, f, m, n) * t1(e, i);              //#3
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t1(a, m) * t1(f, n) * t1(e, i);         //#4
  i0(a, i) +=   V(n, m, e, f) * t2(a, f, m, n) * t1(e, i);                    //#5
  i0(a, i) +=   V(n, m, e, f) * t1(a, m) * t1(f, n) * t1(e, i);               //%6
  i0(a, i) +=  -1.0 * F(m, i) * t1(a, m);                                     //#7
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t2(e, f, i, n) * t1(a, m);              //#8
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t1(e, i) * t1(f, n) * t1(a, m);         //#9
  i0(a, i) +=  V(m, n, f, e) * t2(e, f, i, n) * t1(a, m);                     //#10
  i0(a, i) +=  V(m, n, f, e) * t1(e, i) * t1(f, n) * t1(a, m);                //#11
  i0(a, i) +=  2.0 * F(m, e) * t2(e, a, m, i);                                //#12
  i0(a, i) +=  -1.0 * F(m, e) * t2(e, a, i, m);                               //#13
  i0(a, i) +=  F(m, e) * t1(e, i) * t1(a, m);                                 //#14
  i0(a, i) +=  4.0 * V(m, n, e, f) * t1(f, n) * t2(e, a, m, i);               //#15
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t1(f, n) * t2(e, a, i, m);              //#16
  i0(a, i) +=  2.0 * V(m, n, e, f) * t1(f, n) * t1(e, i) * t1(a, m);          //#17
  i0(a, i) +=  -2.0 * V(m, n, f, e) * t1(f, n) * t2(e, a, m, i);              //#18
  i0(a, i) +=  V(m, n, f, e) * t1(f, n) * t2(e, a, i, m);                     //#19
  i0(a, i) +=  -1.0 * V(m, n, f, e) * t1(f, n) * t1(e, i) * t1(a, m);         //#20
  i0(a, i) +=  2.0 * V(m, a, e, i) * t1(e, m);                                //#21
  i0(a, i) +=  -1.0 * V(m, a, i, e) * t1(e, m);                               //#22
  i0(a, i) +=  2.0 * V(m, a, e, f) * t2(e, f, m, i);                          //#23
  i0(a, i) +=  2.0 * V(m, a, e, f) * t1(e, m) * t1(f, i);                     //#24
  i0(a, i) +=  -1.0 * V(m, a, f, e) * t2(e, f, m, i);                         //#25
  i0(a, i) +=  -1.0 * V(m, a, f, e) * t1(e, m) * t1(f, i);                    //#26
  i0(a, i) +=  -2.0 * V(m, n, e, i) * t2(e, a, m, n);                         //#27
  i0(a, i) +=  -2.0 * V(m, n, e, i) * t1(e, m) * t1(a, n);                    //#28
  i0(a, i) +=  V(n, m, e, i) * t2(e, a, m, n);                                //#29
  i0(a, i) +=  V(n, m, e, i) * t1(e, m) * t1(a, n);                           //#30

  i0.compile();
  i0.assemble();
  i0.compute();
}
```

CCSD Skeleton

```cpp
#include <iostream>
#include "taco.h"
#include "utils.h"

using namespace taco;

int main(int argc, char* argv[]) {
  if (argc != 2){
    std::cout << "Please enter input problem size" << "\n";
    exit(1);
  }

  int idim = atoi(argv[1]);

  Format csr({Dense,Sparse});
  Format csf({Sparse,Sparse,Sparse});
  Format  sv({Sparse});

  Format dense2d({Dense,Dense});
  Format dense4d({Dense,Dense, Dense, Dense});

  Tensor<double> i0("i0", {idim,idim}, dense2d);
  Tensor<double> F("F", {idim, idim}, dense2d);
  Tensor<double> V("V", {idim, idim, idim, idim}, dense4d);
  Tensor<double> t1("t1", {idim,idim}, dense2d);
  Tensor<double> t2("t2", {idim, idim, idim, idim}, dense4d);

// Initialization...

  IndexVar i, m, n, a, e, f;

  std::cout << "Computation started" << "\n";
  i0(a, i) = F(a, i);                                                   //#1
  i0(a, i) +=  -2.0 * F(m, e) * t1(a, m) * t1(e, i) +  F(a, e) * t1(e, i);  //#2
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t2(a, f, m, n) * t1(e, i);         //#3
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t1(a, m) * t1(f, n) * t1(e, i);    //#4
  i0(a, i) +=   V(n, m, e, f) * t2(a, f, m, n) * t1(e, i);               //#5
  i0(a, i) +=   V(n, m, e, f) * t1(a, m) * t1(f, n) * t1(e, i);          //%6
  i0(a, i) +=  -1.0 * F(m, i) * t1(a, m);                                //#7
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t2(e, f, i, n) * t1(a, m);         //#8
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t1(e, i) * t1(f, n) * t1(a, m);    //#9
  i0(a, i) +=  V(m, n, f, e) * t2(e, f, i, n) * t1(a, m);                //#10
  i0(a, i) +=  V(m, n, f, e) * t1(e, i) * t1(f, n) * t1(a, m);           //#11
  i0(a, i) +=  2.0 * F(m, e) * t2(e, a, m, i);                           //#12
  i0(a, i) +=  -1.0 * F(m, e) * t2(e, a, i, m);                          //#13
  i0(a, i) +=  F(m, e) * t1(e, i) * t1(a, m);                            //#14
  i0(a, i) +=  4.0 * V(m, n, e, f) * t1(f, n) * t2(e, a, m, i);          //#15
  i0(a, i) +=  -2.0 * V(m, n, e, f) * t1(f, n) * t2(e, a, i, m);         //#16
  i0(a, i) +=  2.0 * V(m, n, e, f) * t1(f, n) * t1(e, i) * t1(a, m);     //#17
  i0(a, i) +=  -2.0 * V(m, n, f, e) * t1(f, n) * t2(e, a, m, i);         //#18
  i0(a, i) +=  V(m, n, f, e) * t1(f, n) * t2(e, a, i, m);                //#19
  i0(a, i) +=  -1.0 * V(m, n, f, e) * t1(f, n) * t1(e, i) * t1(a, m);    //#20
  i0(a, i) +=  2.0 * V(m, a, e, i) * t1(e, m);                           //#21
  i0(a, i) +=  -1.0 * V(m, a, i, e) * t1(e, m);                          //#22
  i0(a, i) +=  2.0 * V(m, a, e, f) * t2(e, f, m, i);                     //#23
  i0(a, i) +=  2.0 * V(m, a, e, f) * t1(e, m) * t1(f, i);                //#24
  i0(a, i) +=  -1.0 * V(m, a, f, e) * t2(e, f, m, i);                    //#25
  i0(a, i) +=  -1.0 * V(m, a, f, e) * t1(e, m) * t1(f, i);               //#26
  i0(a, i) +=  -2.0 * V(m, n, e, i) * t2(e, a, m, n);                    //#27
  i0(a, i) +=  -2.0 * V(m, n, e, i) * t1(e, m) * t1(a, n);               //#28
  i0(a, i) +=  V(n, m, e, i) * t2(e, a, m, n);                           //#29
  i0(a, i) +=  V(n, m, e, i) * t1(e, m) * t1(a, n);                      //#30

  i0.compile();
  i0.assemble();
  i0.compute();
}
```

CCSD (COMET DSL)

C1 = A1 * B1
C2 = A2 * B2
…
Cn = An * Bn

Reduction

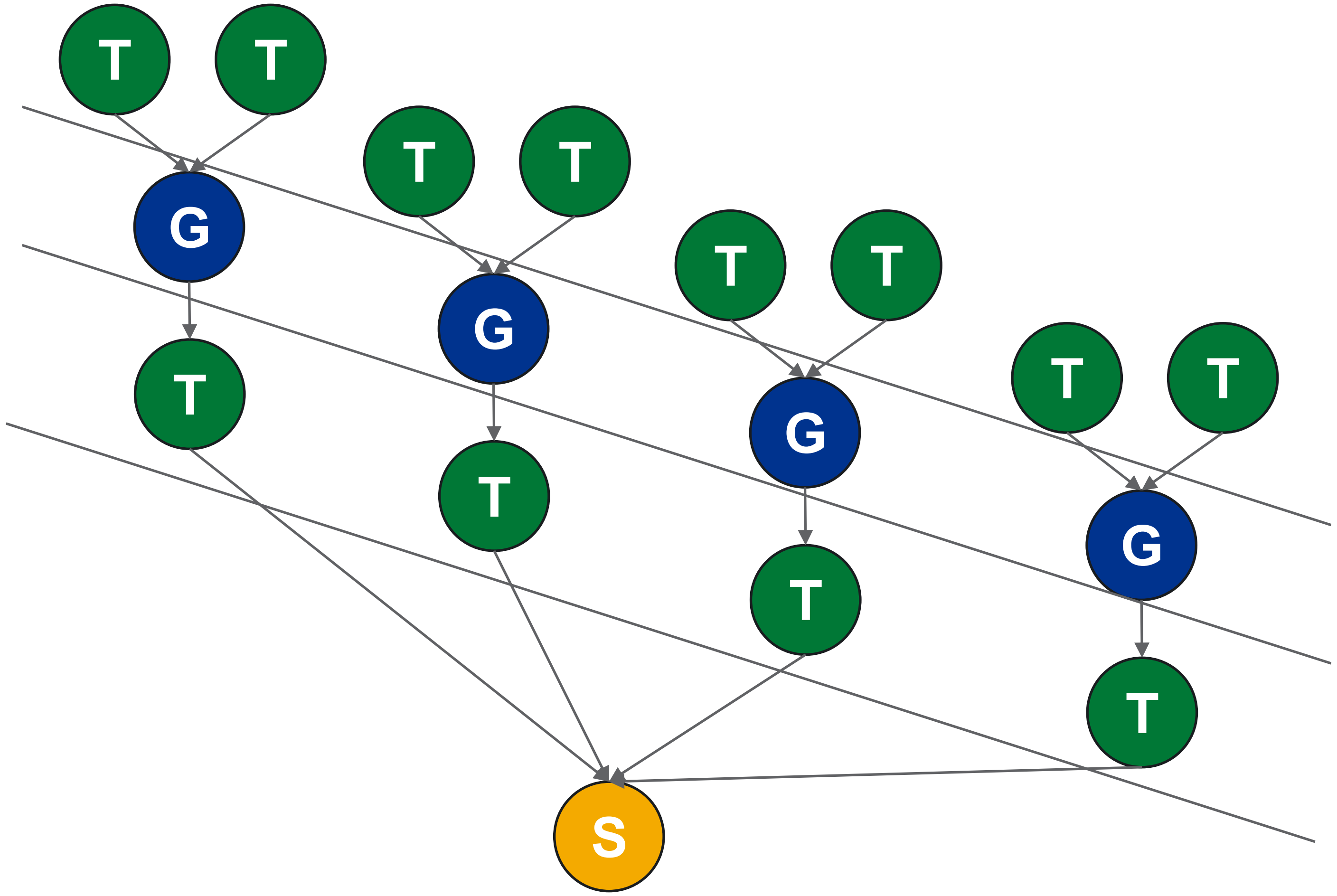
Skeleton code

TC => TTGT

A1t = transpose(A1)
B2t = transpose(B1)
C1t = GEMM(A1t,B1t)
C1 = transpose(C1t)

TTGT Optimization

MCL Implementation of CCSD



- TC reformulated as TTGT
- There are ~30 contractions in CCSD(1)
- Can use wavefront algorithm
- Many tasks run in parallel on multiple devices

GPU Task

FPGA Task

CPU Task

* This is only a functional implementation meant to test Junction

MCL CCSD Proxy Application 1/2

```c
int main(int argc, char** argv)
{
        float *A, *B, *C;
        float *AT, *BT, *CT;
        unsigned long i, j;
        int ret;
        struct tc_struct_hdl* tc_hdl;

        mcl_banner("Tensor Contraction Skeleton");
        parse_global_opts(argc, argv);

        mcl_init(1, 0x0);

        A  = (float*) malloc(size * size * sizeof(float));
        AT = (float*) malloc(size * size * sizeof(float));
        B  = (float*) malloc(size * size * sizeof(float));
        BT = (float*) malloc(size * size * sizeof(float));
        C  = (float*) malloc(size * size * sizeof(float));
        CT = (float*) malloc(size * size * sizeof(float));
        tc_hdl = (struct tc_struct_hdl*) malloc (rep * sizeof(struct tc_struct_hdl));

        if(!A || !B || !C || !AT || !BT || !CT || !tc_hdl){
                printf("Error allocating vectors. Aborting.");
                ret = -1;
                goto err;
        }

        srand48(13579862);
        for(i=0; i<size; ++i){
                for(j=0; j<size; ++j){
                        A[i*size+j] = (float)(0.5 + drand48()*1.5);
                }
        }
```

```c
for(i=0; i<size; ++i){
        for(j=0; j<size; ++j){
                B[i*size+j] = (float)(0.5 + drand48()*1.5);
        }
}

memset((char*) C, 0x0, size*size*sizeof(float));

mcl_prg_load("./src/transpose.cl", "", MCL_PRG_SRC);
mcl_prg_load("./src/matrixMul.cl", "", MCL_PRG_SRC);
mcl_prg_load("./build_dir.hw.xilinx_vck5000_gen3x16_xdma_1_202120_1/
           matrixMul.xclbin", "", MCL_PRG_BIN);
```

Load programs. The same kernel can be in different programs…

MCL CCSD Proxy Application 2/2

```c
printf("------------------------------------------\n");
    printf("\t Launching transposes...\n");
    for(i=0; i<rep; i++){
            transpose(&(tc_hdl[i].hdl[0]), A, AT, size);
            transpose(&(tc_hdl[i].hdl[1]), B, BT, size);
    }

    for(i=0; i<rep; i++){
            mcl_wait(tc_hdl[i].hdl[0]);
            mcl_wait(tc_hdl[i].hdl[1]);
            gemm(&(tc_hdl[i].hdl[2]),CT, AT, BT, size);
    }

    for(i=0; i<rep; i++){
            mcl_wait(tc_hdl[i].hdl[2]);
            transpose(&(tc_hdl[i].hdl[3]), CT, C, size);
    }

    mcl_wait_all();

          ...

    mcl_finit();
    return 0;
}
```

Start all transpose

For each TTGT, wait for pairs of transposes to complete, then start GEMM[1]

For each TTGT, wait for GEMM to complete, then start transpose

Accumulate results

Establish task dependencies

[1] For simplicity, mcl_test() have been replaced with mcl_wait()

Transpose

```
inline void transpose(mcl_handle** hdl, float* in, float* out, size_t n)
{
        int ret;
        size_t bsize = n * n * sizeof(float);
        int offset = 0;
        size_t szGlobalWorkSize[3] = { n, n, 1};
        size_t szLocalWorkSize[3]  = {BLOCK_DIM, BLOCK_DIM, 1};

        *hdl = mcl_task_create();
        assert(*hdl);

        ret = mcl_task_set_kernel(*hdl, "transpose", 6);
        assert(!ret);

        ret  = mcl_task_set_arg(*hdl, 0, (void*) out, bsize, MCL_ARG_OUTPUT | MCL_ARG_BUFFER);
        ret |= mcl_task_set_arg(*hdl, 1, (void*) in,  bsize, MCL_ARG_INPUT | MCL_ARG_BUFFER);
        ret |= mcl_task_set_arg(*hdl, 2, (void*) &offset, sizeof(int), MCL_ARG_INPUT | MCL_ARG_SCALAR);
        ret |= mcl_task_set_arg(*hdl, 3, (void*) &n, sizeof(int), MCL_ARG_INPUT | MCL_ARG_SCALAR);
        ret |= mcl_task_set_arg(*hdl, 4, (void*) &n, sizeof(int), MCL_ARG_INPUT | MCL_ARG_SCALAR);
        ret |= mcl_task_set_arg(*hdl, 5, NULL, (BLOCK_DIM + 1) * BLOCK_DIM * sizeof(float), MCL_ARG_LOCAL);
        assert(!ret);

        ret = mcl_exec(*hdl, szGlobalWorkSize, szLocalWorkSize, MCL_TASK_GPU);
        assert(!ret);

}
```

Transpose kernel

Transpose kernel

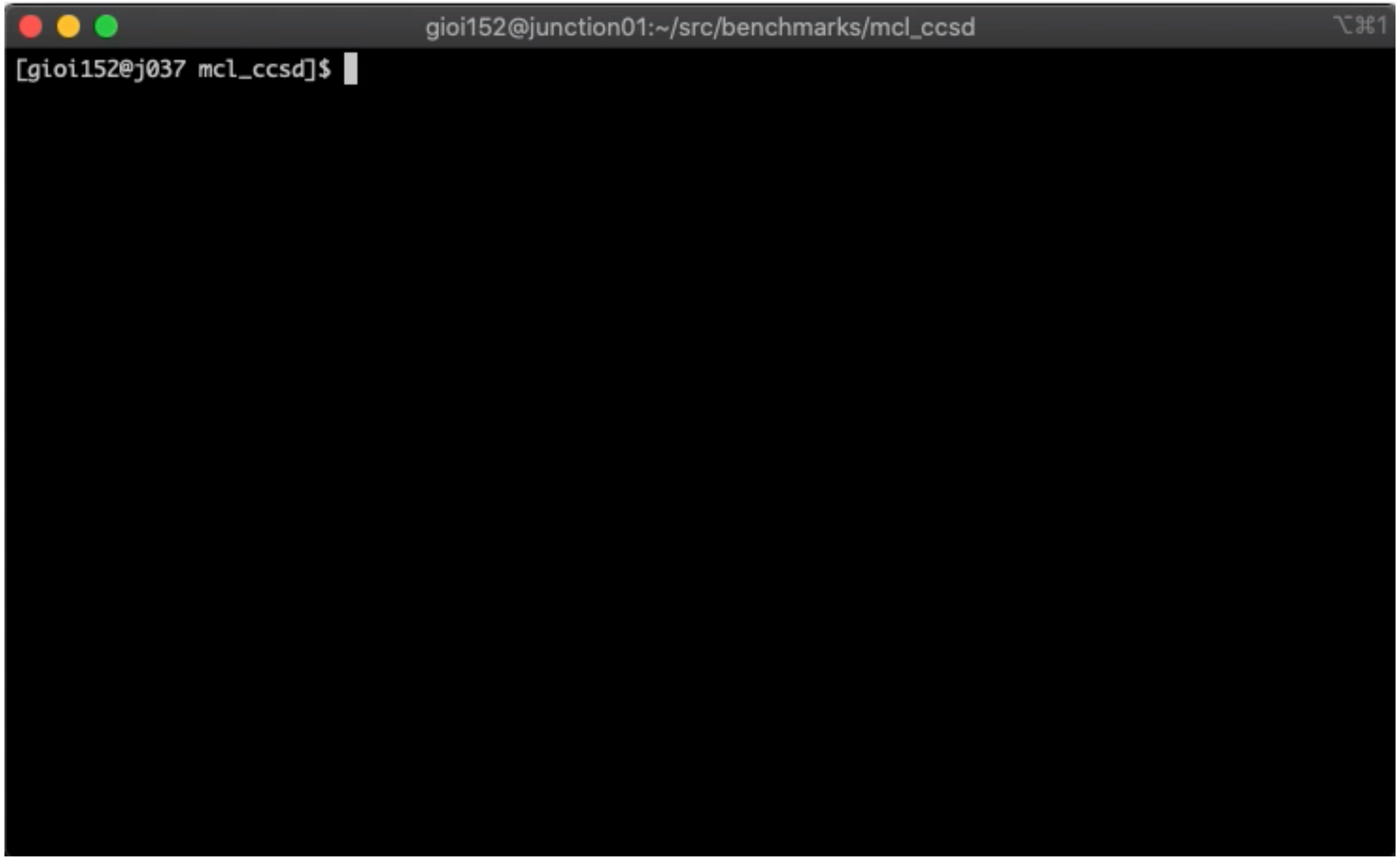
GEMM

```
inline void gemm(mcl_handle** hdl, float* C, float* A, float* B, size_t n)
{
        int ret;
        size_t bsize = n * n * sizeof(float);
        size_t szGlobalWorkSize[3] = { n, n, 1};
        size_t szLocalWorkSize[3]  = {BLOCK_DIM, BLOCK_DIM, 1};

        *hdl = mcl_task_create();
        assert(*hdl);

        ret = mcl_task_set_kernel(*hdl, "matrixMul", 8);
        assert(!ret);

        ret  = mcl_task_set_arg(*hdl, 0, (void*) C, bsize, MCL_ARG_OUTPUT | MCL_ARG_BUFFER);
        ret |= mcl_task_set_arg(*hdl, 1, (void*) A, bsize, MCL_ARG_INPUT | MCL_ARG_BUFFER);
        ret |= mcl_task_set_arg(*hdl, 2, (void*) B, bsize, MCL_ARG_INPUT | MCL_ARG_BUFFER);
        ret |= mcl_task_set_arg(*hdl, 3, NULL, sizeof(float) * BLOCK_DIM *BLOCK_DIM, MCL_ARG_LOCAL);
        ret |= mcl_task_set_arg(*hdl, 4, NULL, sizeof(float) * BLOCK_DIM *BLOCK_DIM, MCL_ARG_LOCAL);
        ret |= mcl_task_set_arg(*hdl, 5, (void*) &n, sizeof(int), MCL_ARG_INPUT | MCL_ARG_SCALAR);
        ret |= mcl_task_set_arg(*hdl, 6, (void*) &n, sizeof(int), MCL_ARG_INPUT | MCL_ARG_SCALAR);
        ret |= mcl_task_set_arg(*hdl, 7, (void*) &n, sizeof(int), MCL_ARG_INPUT | MCL_ARG_SCALAR);
        assert(!ret);

        ret = mcl_exec(*hdl, szGlobalWorkSize, szLocalWorkSize, MCL_TASK_FPGA);
        assert(!ret);

}
```

GEMM kernel

Execute on FPGA. This could also be MCL_TASK_GPU or MCL_TASK_GPU | MCL_TASK_FPGA

MCL CCSD Proxy Demo



Thank you