Pacific Northwest
NATIONAL LABORATORY

Hands-on Session

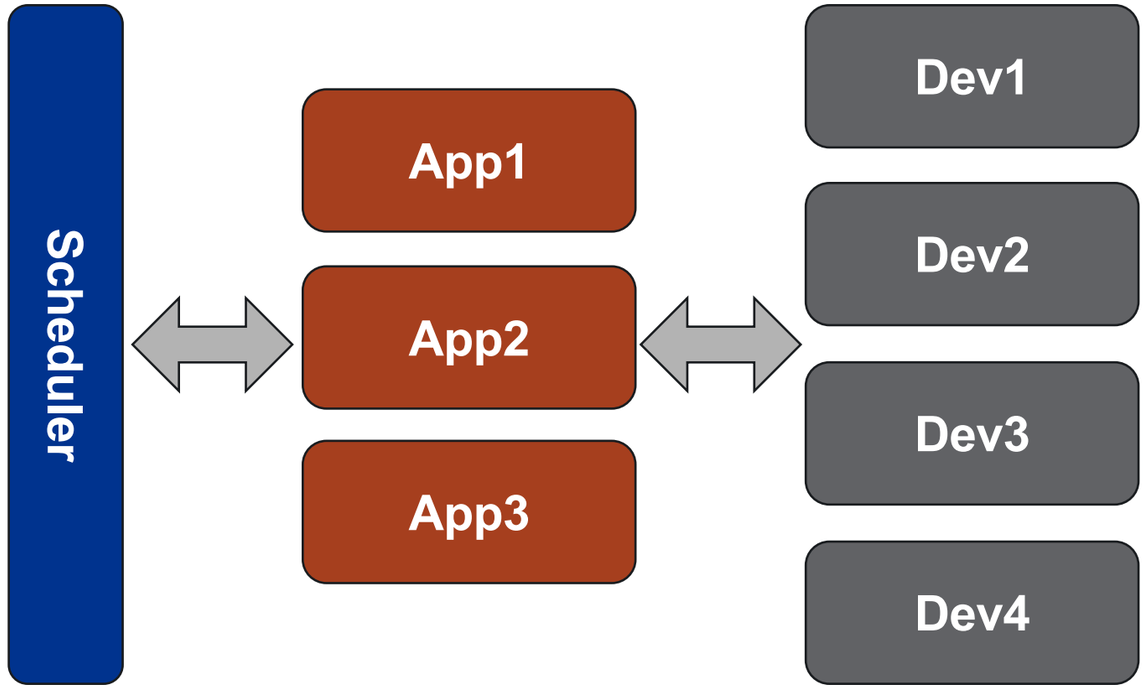
Roberto Gioiosa

U.S. DEPARTMENT OF ENERGY BATTELLE

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

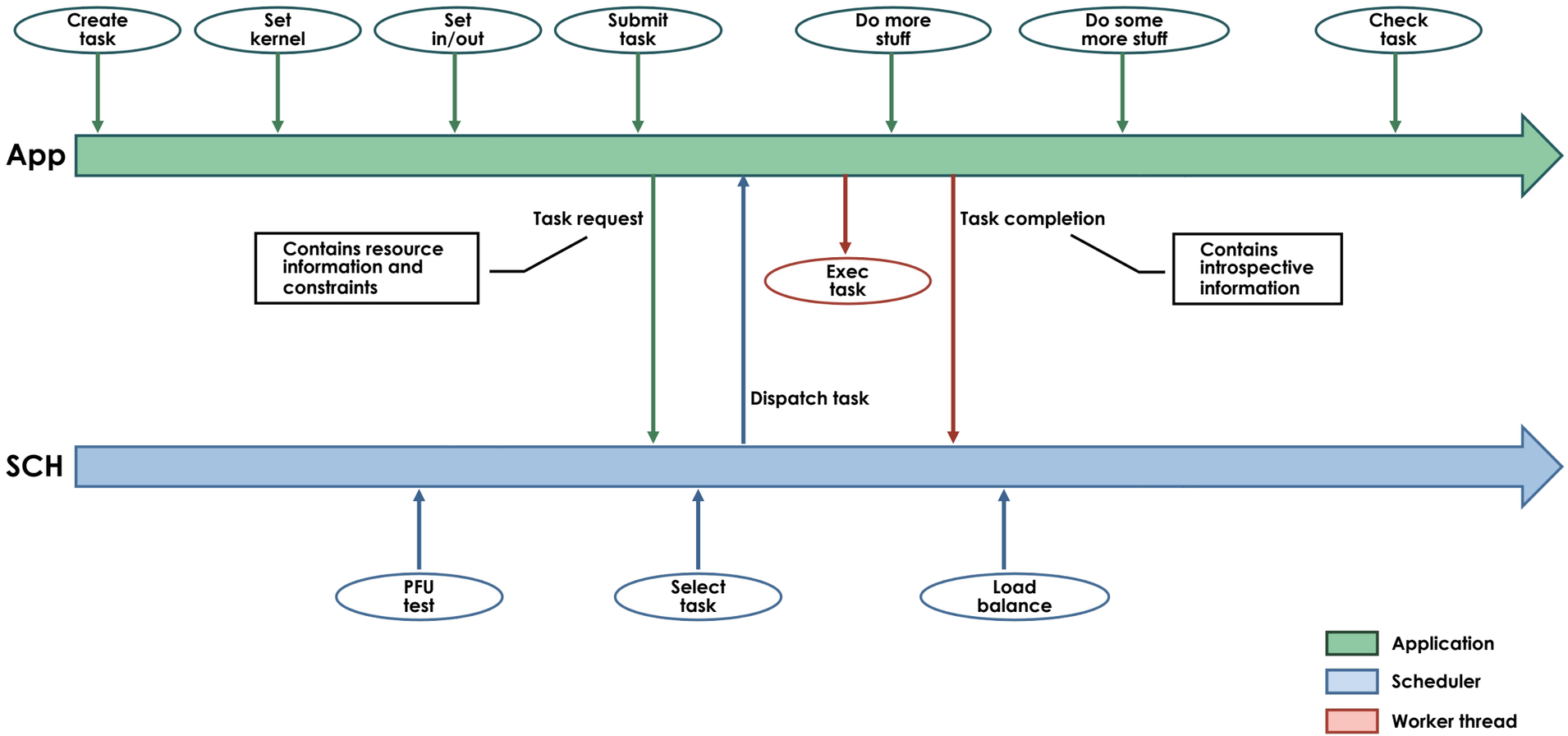
MCL Environment

- 1 system-level scheduler
- 1+ MCL applications
  - Applications interact directly with hardware devices
  - No additional data copy between applications and scheduler
- 1+ (heterogeneous) devices
  - CPU cores can also be used as devices



Pacific Northwest NATIONAL LABORATORY

Simple Execution Trace



MCL Scheduler

- Manage hardware resources
- Perform load balancing
- Track memory objects allocated on devices
- Implement scheduling framework:
  - Multiple scheduling algorithms
    Additional schedulers can be added (MCL Scheduler ABI)
- Generally runs in background
- Trace resource utilization

Scheduler	Objective
First Fit (ff)	Power efficiency
Round-robin (rr)	Load balancing
Delay	Locality
Hybrid	Load Balancing + Locality

Current Scheduling Algorithms

Want to know more about MCL scheduler for multi-device? See last year tutorial!

```
Usage: ./src/sched/mcl_sched [options]
        -s, --sched-class {fifo|fffs}  Select scheduler class (def = 'fifo')
        -p, --res-policy {ff|rr|delay|hybrid}  Select resource policy (def = class dependent)
        -h, --help                     Show this help
```

Anatomy of an MCL application 1/4

```
#include <minos.h>
int main(int argc, char** argv){
        mcl_handle**    hdl = NULL;
        uint64_t        pes[MCL_DEV_DIMS] = {N, N, 1};
        const size_t    msize    = N * N * sizeof(double);

        mcl_init(workers,0x0);
        A = (double*) malloc(size * size * sizeof(double));
        B = (double*) malloc(size * size * sizeof(double));
        C = (double*) malloc(size * size * sizeof(double));
        hdl = (mcl_handle**) malloc(sizeof(mcl_handle*) * rep);
        mcl_prg_load("./gemmN.cl","-DDOUBLE_PRECISION", MCL_PRG_SRC);

        for(int i=0; i<rep; i++){
                hdl[i] = mcl_task_create();
                mcl_task_set_kernel(hdl[i], "gemmN", 4);
                mcl_task_set_arg(hdl[i], 0, (void*) A, msize,        MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 1, (void*) B, msize,        MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 2, (void*) &n, sizeof(int), MCL_ARG_INPUT|MCL_ARG_SCALAR);
                mcl_task_set_arg(hdl[i], 3, (void*) C, msize,        MCL_ARG_OUTPUT|MCL_ARG_BUFFER);

                mcl_exec(hdl[i], pes, NULL, MCL_TASK_GPU);
        }
        mcl_wait_all();

        for(i=0; i<rep; i++)
                mcl_hdl_free(hdl[i]);
        free(hdl);
        mcl_finit();
        return 0;
}
```

MCL API header file

Init function:

- Define # MCL worker threads
- Register app w/ scheduler
- Device discovery

Finit function:

- Check pending tasks
- De-register app w/ scheduler

Anatomy of an MCL application 2/4

```
#include <minos.h>
int main(int argc, char** argv){
        mcl_handle**    hdl = NULL;
        uint64_t        pes[MCL_DEV_DIMS] = {N, N, 1};
        const size_t    msize    = N * N * sizeof(double);

        mcl_init(workers,0x0);
        A = (double*) malloc(size * size * sizeof(double));
        B = (double*) malloc(size * size * sizeof(double));
        C = (double*) malloc(size * size * sizeof(double));
        hdl = (mcl_handle**) malloc(sizeof(mcl_handle*) * rep);
        mcl_prg_load("./gemmN.cl","-DDOUBLE_PRECISION", MCL_PRG_SRC);

        for(int i=0; i<rep; i++){
                hdl[i] = mcl_task_create();
                mcl_task_set_kernel(hdl[i], "gemmN", 4);
                mcl_task_set_arg(hdl[i], 0, (void*) A, msize,        MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 1, (void*) B, msize,        MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 2, (void*) &n, sizeof(int), MCL_ARG_INPUT|MCL_ARG_SCALAR);
                mcl_task_set_arg(hdl[i], 3, (void*) C, msize,        MCL_ARG_OUTPUT|MCL_ARG_BUFFER);

                mcl_exec(hdl[i], pes, NULL, MCL_TASK_GPU);
        }
        mcl_wait_all();

        for(i=0; i<rep; i++)
                mcl_hdl_free(hdl[i]);
        free(hdl);
        mcl_finit();
        return 0;
}
```

Task handle
- Track status
- Report errors
- Provide stats

Create task
- Allocate task resources

Remove task handle

Anatomy of an MCL application 3/4

```
#include <minos.h>
int main(int argc, char** argv){
        mcl_handle**    hdl = NULL;
        uint64_t        pes[MCL_DEV_DIMS] = {N, N, 1};
        const size_t    msize    = N * N * sizeof(double);

        mcl_init(workers,0x0);
        A = (double*) malloc(size * size * sizeof(double));
        B = (double*) malloc(size * size * sizeof(double));
        C = (double*) malloc(size * size * sizeof(double));
        hdl = (mcl_handle**) malloc(sizeof(mcl_handle*) * rep);
        mcl_prg_load("./gemmN.cl","-DDOUBLE_PRECISION", MCL_PRG_SRC);

        for(int i=0; i<rep; i++){
                hdl[i] = mcl_task_create();
                mcl_task_set_kernel(hdl[i], "gemmN", 4);
                mcl_task_set_arg(hdl[i], 0, (void*) A, msize,       MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 1, (void*) B, msize,       MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 2, (void*) &n, sizeof(int), MCL_ARG_INPUT|MCL_ARG_SCALAR);
                mcl_task_set_arg(hdl[i], 3, (void*) C, msize,       MCL_ARG_OUTPUT|MCL_ARG_BUFFER);

                mcl_exec(hdl[i], pes, NULL, MCL_TASK_GPU);
        }
        mcl_wait_all();

        for(i=0; i<rep; i++)
                mcl_hdl_free(hdl[i]);
        free(hdl);
        mcl_finit();
        return 0;
}
```

Load program:
- Source file
- Compiler flags
- Source type

Select kernel:
- Kernel name
- # args

Set kernel argument
- Arg ID
- Host address
- Size
- Input/output + scalar/buffer

Anatomy of an MCL application 4/4

```
#include <minos.h>
int main(int argc, char** argv){
        mcl_handle**    hdl = NULL;
        uint64_t        pes[MCL_DEV_DIMS] = {N, N, 1};
        const size_t    msize    = N * N * sizeof(double);

        mcl_init(workers,0x0);
        A = (double*) malloc(size * size * sizeof(double));
        B = (double*) malloc(size * size * sizeof(double));
        C = (double*) malloc(size * size * sizeof(double));
        hdl = (mcl_handle**) malloc(sizeof(mcl_handle*) * rep);
        mcl_prg_load("./gemmN.cl","-DDOUBLE_PRECISION", MCL_PRG_SRC);

        for(int i=0; i<rep; i++){
                hdl[i] = mcl_task_create();
                mcl_task_set_kernel(hdl[i], "gemmN", 4);
                mcl_task_set_arg(hdl[i], 0, (void*) A, msize,        MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 1, (void*) B, msize,        MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 2, (void*) &n, sizeof(int), MCL_ARG_INPUT|MCL_ARG_SCALAR);
                mcl_task_set_arg(hdl[i], 3, (void*) C, msize,        MCL_ARG_OUTPUT|MCL_ARG_BUFFER);

                mcl_exec(hdl[i], pes, NULL, MCL_TASK_GPU);
        }
        mcl_wait_all();

        for(i=0; i<rep; i++)
                mcl_hdl_free(hdl[i]);
        free(hdl);
        mcl_finit();
        return 0;
}
```

Queue a task:

- # PEs (global, local)
- Device class or ANY
- Return immediately

Wait for completion

- Block until all tasks are completed

Computational Kernel

- OpenCL source code
  - Same code, many devices
- SPIRV binary IR
  - Same IR, many devices
- Binary (FPGA, NVDLA)
  - Device specific

```
#ifdef DOUBLE_PRECISION
#define FPTYPE double
#else
#define FPTYPE float
#endif

__kernel void gemmN( const __global FPTYPE* A,
                     const __global FPTYPE* B, int N,
                     __global FPTYPE* C)
{
          // Thread identifiers
          const int globalRow = get_global_id(0); // Row ID of C (0..N)
          const int globalCol = get_global_id(1); // Col ID of C (0..N)

          // Compute a single element (loop over K)
          FPTYPE acc = 0.0f;
          for (int k=0; k<N; k++) {
                    acc += A[globalRow*N + k] * B[k*N + globalCol];
          }
          // Store the result
          C[globalRow*N + globalCol] = acc;
}
```

GemmN.cl

MCL “Hello World” 1/2

- Kernel NxN GEMM
- MCL workers: 1,8
- Testbed: NVIDIA DGX-1 V100
  - 8 V100 GPUs
- Device Class: GPU

```
int main(int argc, char** argv)
{
                double *A, *B, *C;
                int i, j, ret = -1;

                mcl_banner("GEMM N Test");
                parse_global_opts(argc, argv);
                mcl_init(1,0x0);

                A = (double*) malloc(size * size * sizeof(double));
                B = (double*) malloc(size * size * sizeof(double));
                C = (double*) malloc(size * size * sizeof(double));
                if(!A || !B || !C){
                                printf("Error allocating vectors. Aborting.");
                                goto err;
                }

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

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

                ret = test_mcl(A,B,C,size);

                mcl_finit();

                free(A);
                free(B);
                free(C);
err:
                return ret;
}
```

MCL “Hello World” 2/2

```
int test_mcl(double* A, double* B, double* C, size_t N)
{
        struct timespec start, end;
        mcl_handle* hdl = NULL;
        uint64_t pes[MCL_DEV_DIMS] = {N, N, 1};
        const size_t msize = N * N * sizeof(double);
        uint64_t i;
        unsigned int errs = 0;
        float rtime;
        int ret;

        printf("MCL Test...");
        clock_gettime(CLOCK_MONOTONIC,&start);

        mcl_prg_load(hdl, "./gemmN.cl", "-DDOUBLE_PRECISION", MCL_PRG_SRC);
        hdl = mcl_task_create();
        mcl_task_set_kernel(hdl, "gemmN", 4);
        mcl_task_set_arg(hdl, 0, (void*) A, msize, MCL_ARG_INPUT|MCL_ARG_BUFFER);
        mcl_task_set_arg(hdl, 1, (void*) B, msize, MCL_ARG_INPUT|MCL_ARG_BUFFER);
        mcl_task_set_arg(hdl, 3, (void*) C, msize, MCL_ARG_OUTPUT|MCL_ARG_BUFFER);
        ret = mcl_exec(hdl, pes, NULL, MCL_TASK_GPU));
        mcl_wait(hdl);
        clock_gettime(CLOCK_MONOTONIC, &end);

        if(hdl->ret == MCL_RET_ERROR){
                printf("Error executing task %"PRIu64"!\n", i);
                errs++;
        }
        if(errs)
                printf("Detected %u errors!\n",errs);
        else{
                rtime = ((float)tdiff(end,start))/BILLION;
                printf("Done.\n Test time : %f seconds\n", rtime);
                printf(" Throughput: %f tasks/s\n", ((float)rep)/rtime);
        }
        mcl_hdl_free(hdl);
        return errs;
}
```

- Execute on a GPU class device
- Could use MCL_TASK_ANY to execute on any device class
- Use either MCL_TASK_ANY (or MCL_TASK_CPU) if running in the tutorial container

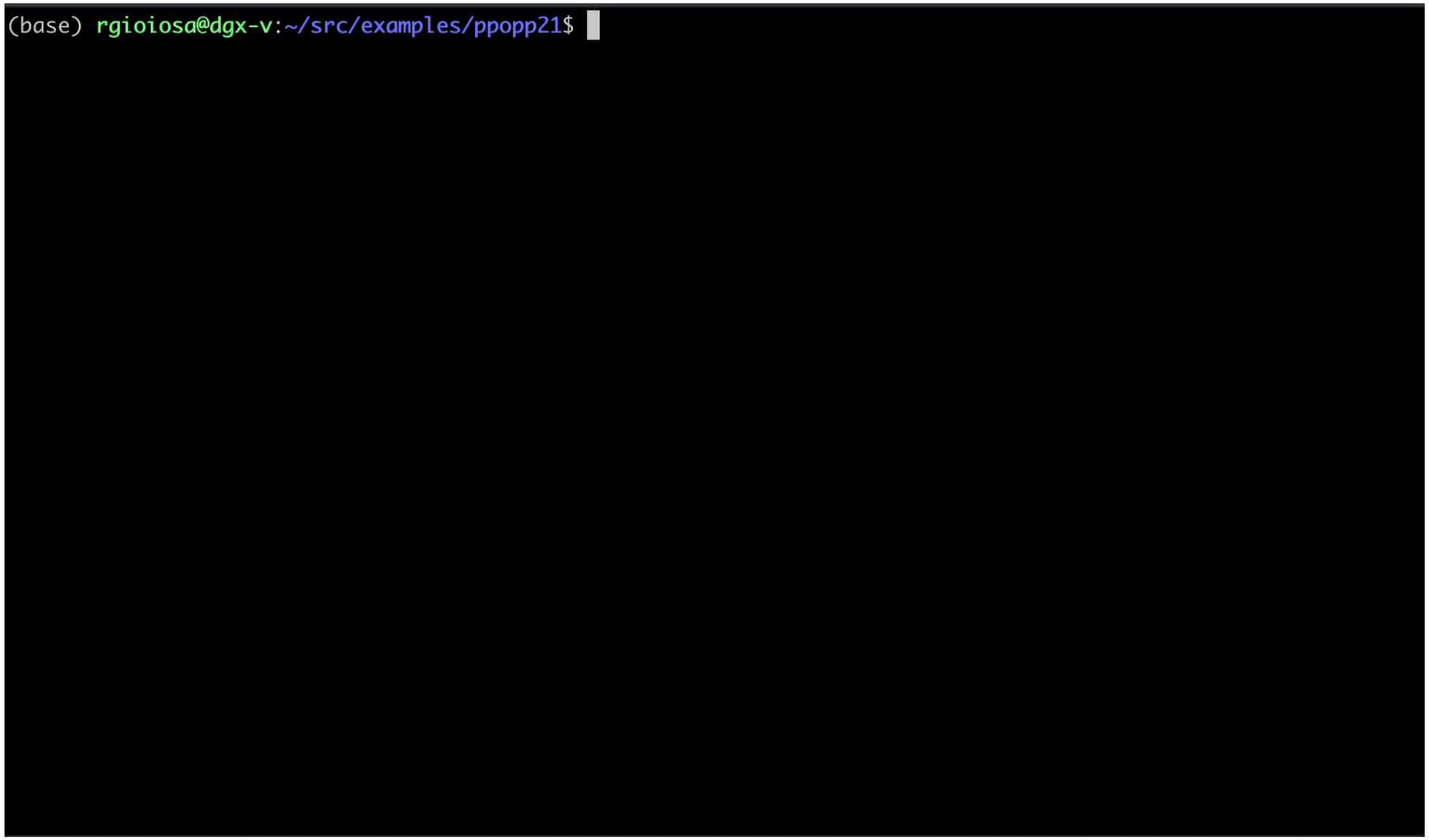
Running MCL "Hello World!"

- Compile application:
  - `gcc -Wall -O2 -I$HOME/include -I. -o example1 example1.c utils.c -L$HOME/lib -lmcl -lOpenCL -lm -lrt`
- Launching the scheduler
  - `mcl_sched -p rr &`
- Run application
  - `./example1`

```
============================================
    Minos Computing Library
    GEMM N Test
============================================
Version:    0.5
Start time: Fri Feb 26 00:42:07 2021
--------------------------------------------
Parsed options:
            Number of workers      = 1
            Type of test           = Async
            Matrix size            = 64
            Number of repetitions  = 1
            Type of PEs            = 1
            Verify test            = No
MCL Test...Done.
  Test time : 0.007895 seconds
  Throughput: 126.655640 tasks/s
```

Pacific Northwest
NATIONAL LABORATORY

MCL Demo



MCL Improved “Hello World”

```
int test_mcl(double* A, double* B, double* C, size_t N)
{
                struct timespec start, end;
                mcl_handle** hdl = NULL;
                uint64_t pes[MCL_DEV_DIMS] = {N, N, 1};
                const size_t msize = N * N * sizeof(double);
                uint64_t i;
                unsigned int errs = 0;
                float rtime;
                int ret;

                printf("MCL Test...");

                hdl = (mcl_handle**) malloc(sizeof(mcl_handle*) * rep);
                mcl_prg_load(hdl, "./gemmN.cl", "-DDOUBLE_PRECISION", MCL_PRG_SRC);
                clock_gettime(CLOCK_MONOTONIC,&start);
                for(i=0; i<rep; i++){
                                hdl[i] = mcl_task_create();
                                mcl_task_set_kernel(hdl[i], "gemmN", 4);
                                mcl_task_set_arg(hdl[i], 0, (void*) A, msize, MCL_ARG_INPUT|MCL_ARG_BUFFER);
                                mcl_task_set_arg(hdl[i], 1, (void*) B, msize, MCL_ARG_INPUT|MCL_ARG_BUFFER);
                                mcl_task_set_arg(hdl[i], 2, (void*) &N, sizeof(int), MCL_ARG_INPUT|MCL_ARG_SCALAR);
                                mcl_task_set_arg(hdl[i], 3, (void*) C, msize, MCL_ARG_OUTPUT|MCL_ARG_BUFFER);

                                ret = mcl_exec(hdl[i], pes, NULL, MCL_TASK_GPU);

                                mcl_wait(hdl[i]);
                }
                clock_gettime(CLOCK_MONOTONIC, &end);

                rtime = ((float)tdiff(end,start))/BILLION;
                printf("Done.\n Test time : %f seconds\n", rtime);
                printf(" Throughput: %f tasks/s\n", ((float)rep)/rtime);

                for(i=0; i<rep; i++)
                                mcl_hdl_free(hdl[i]);
                free(hdl);

                return 0;
}
```

Execute multiple tasks across all GPUs

Synchronous execution

Pacific Northwest NATIONAL LABORATORY

Running MCL Improved "Hello World!"

- Compile application:
  - `gcc -Wall -O2 -I$HOME/include -I. -o example2 example2.c utils.c -L$HOME/lib -lmcl -lOpenCL -lm -lrt`
- Launching the scheduler
  - `mcl_sched -p rr &`
- Run application
  - `./example2 -r 1024`

```
==========================================
    Minos Computing Library
    GEMM N Test
==========================================
Version:    0.5
Start time: Fri Feb 26 00:42:22 2021
------------------------------------------
Parsed options:
            Number of workers     = 1
            Type of test          = Async
            Matrix size           = 64
            Number of repetitions = 1024
            Type of PEs           = 1
            Verify test           = No
MCL Test...Done.
  Test time : 0.354675 seconds
  Throughput: 2887.153076 tasks/s
```

MCL Asynchronous “Hello World”

```
int test_mcl(double* A, double* B, double* C, size_t N)
{
        struct timespec start, end;
        mcl_handle** hdl = NULL;
        uint64_t pes[MCL_DEV_DIMS] = {N, N, 1};
        const size_t msize = N * N * sizeof(double);
        uint64_t i;
        unsigned int errs = 0;
        float rtime;
        int ret;

        printf("MCL Test...");

        hdl = (mcl_handle**) malloc(sizeof(mcl_handle*) * rep);
        mcl_prg_load(hdl, "./gemmN.cl", "-DDOUBLE_PRECISION", MCL_PRG_SRC);
        clock_gettime(CLOCK_MONOTONIC,&start);
        for(i=0; i<rep; i++){
                hdl[i] = mcl_task_create();
                mcl_task_set_kernel(hdl[i], "gemmN", 4);
                mcl_task_set_arg(hdl[i], 0, (void*) A, msize, MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 1, (void*) B, msize, MCL_ARG_INPUT|MCL_ARG_BUFFER);
                mcl_task_set_arg(hdl[i], 2, (void*) &N, sizeof(int), MCL_ARG_INPUT|MCL_ARG_SCALAR);
                mcl_task_set_arg(hdl[i], 3, (void*) C, msize, MCL_ARG_OUTPUT|MCL_ARG_BUFFER);

                ret = mcl_exec(hdl[i], pes, NULL, MCL_TASK_GPU);
        }
        mcl_wait_all();

        clock_gettime(CLOCK_MONOTONIC, &end);

        rtime = ((float)tdiff(end,start))/BILLION;
        printf("Done.\n Test time : %f seconds\n", rtime);
        printf(" Throughput: %f tasks/s\n", ((float)rep)/rtime);

        for(i=0; i<rep; i++)
                mcl_hdl_free(hdl[i]);
        free(hdl);

        return 0;
}
```

Execute multiple tasks across all GPUs

```
mcl_init(8,0x0);
```

Multiple workers run/check tasks in parallel

Asynchronous execution

Pacific Northwest NATIONAL LABORATORY

Running MCL Asynchronous "Hello World!"

- Compile application:
  - `gcc -Wall -O2 -I$HOME/include -I. -o example3 example3.c utils.c -L$HOME/lib -lmcl -lOpenCL -lm -lrt`
- Launching the scheduler
  - `mcl_sched -p rr &`
- Run application
  - `./example3 -r 1024 -w 8`

```
============================================
    Minos Computing Library
    GEMM N Test
============================================
Version:    0.5
Start time: Fri Feb 26 01:13:51 2021
--------------------------------------------
Parsed options:
            Number of workers      = 8
            Type of test           = Async
            Matrix size            = 64
            Number of repetitions = 1024
            Type of PEs            = 1
            Verify test            = No
MCL Test...Done.
  Test time : 0.038343 seconds
  Throughput: 26706.570312 tasks/s
```

Pacific Northwest
NATIONAL LABORATORY

Thank you