GPU Teaching Kit

Accelerated Computing

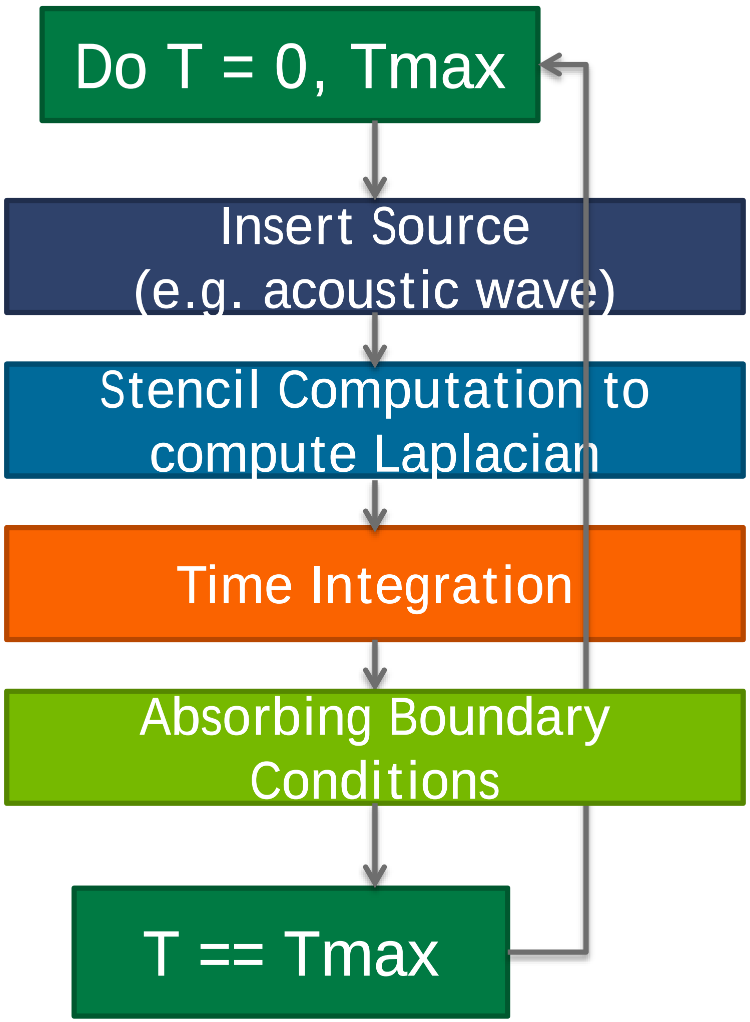
Module 18 – Related Programming Models: MPI

Lecture 18.2 – Introduction to MPI-CUDA Programming

Objective

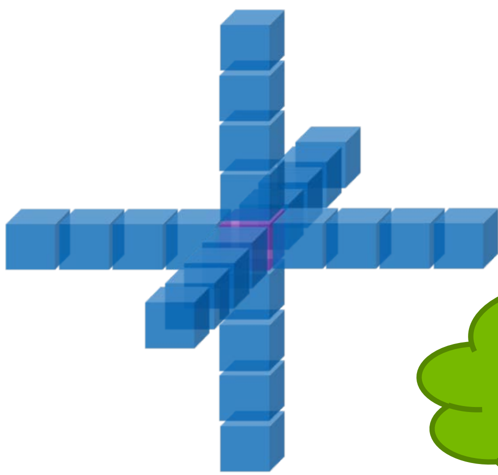
- To learn to write an MPI-CUDA application
  - Vector addition example
  - Wave propagation stencil example
  - MPI Barriers

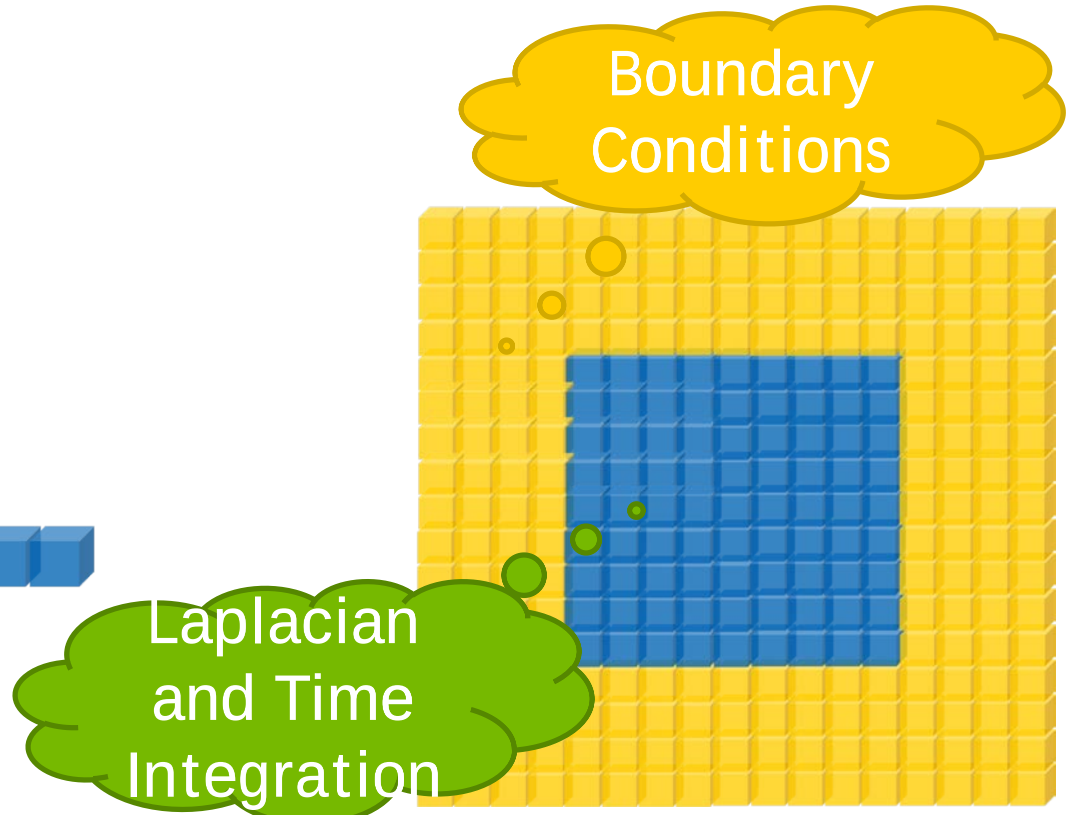
A Typical Wave Propagation Application



Review of Stencil Computations

- Example: wave propagation modeling
- $\nabla^2 U - \frac{1}{v^2}\frac{\partial U}{\partial t} = 0$
- Approximate Laplacian using finite differences





Wave Propagation: Kernel Code

```
/* Coefficients used to calculate the laplacian */
__constant__ float coeff[5];

__global__ void wave_propagation(float *next, float *in,
                    float *prev, float *velocity, dim3 dim)
{
    unsigned x = threadIdx.x + blockIdx.x * blockDim.x;
    unsigned y = threadIdx.y + blockIdx.y * blockDim.y;
    unsigned z = threadIdx.z + blockIdx.z * blockDim.z;

    /* Point index in the input and output matrixes */
    unsigned n = x + y * dim.z + z * dim.x * dim.y;

    /* Only compute for points within the matrixes */
    if(x < dim.x && y < dim.y && z < dim.z) {

        /* Calculate the contribution of each point to the laplacian */
        float laplacian = coeff[0] + in[n];
```

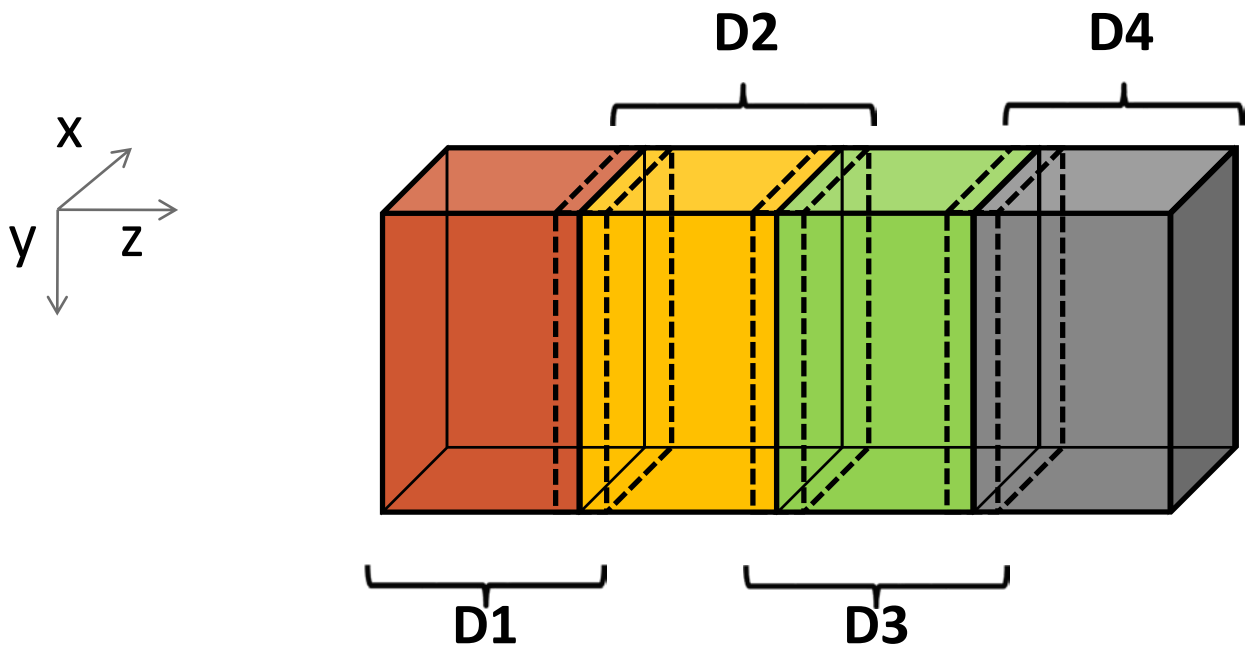
Wave Propagation: Kernel Code

```
        for(int i = 1; i < 5; ++i) {
            laplacian += coeff[i] *
                                 (in[n – i] + /* Left */
                                 in[n + i] + /* Right */
                                 in[n – i * dim.x] + /* Top */
                                 in[n + I * dim.x] + /* Bottom */
                                 in[n – i * dim.x * dim.y] + /* Behind */
                                 in[n + i * dim.x * dim.y]);  /* Front */
        }

        /* Time integration */
        next[n] = velocity[n] * laplacian + 2 * in[n] – prev[n];
    }
}
```

Stencil Domain Decomposition

- Volumes are split into tiles (along the Z-axis)
  - 3D-Stencil introduces data dependencies



Wave Propagation: Main Process

```
int main(int argc, char *argv[]) {
    int pad = 0, dimx  = 480+pad, dimy  = 480, dimz  = 400, nreps = 100;
    int pid=-1, np=-1;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    if(np < 3) {
        if(0 == pid) printf("Nedded 3 or more processes.\n");
        MPI_Abort( MPI_COMM_WORLD, 1 ); return 1;
    }
    if(pid < np - 1)
        compute_node(dimx, dimy, dimz / (np - 1), nreps);
    else
        data_server( dimx,dimy,dimz, nreps );

    MPI_Finalize();
    return 0;
}
```

Stencil Code: Server Process (I)

```
void data_server(int dimx, int dimy, int dimz, int nreps) {
    int np, num_comp_nodes = np – 1, first_node = 0, last_node = np - 2;
    unsigned int num_points = dimx * dimy * dimz;
    unsigned int num_bytes  = num_points * sizeof(float);
    float *input=0, *output = NULL, *velocity = NULL;
    /* Set MPI Communication Size */
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    /* Allocate input data */
    input = (float *)malloc(num_bytes);
    output = (float *)malloc(num_bytes);
    velocity = (float *)malloc(num_bytes);
    if(input == NULL || output == NULL || velocity == NULL) {
        printf(“Server couldn't allocate memory\n");
        MPI_Abort( MPI_COMM_WORLD, 1 );
    }
    /* Initialize input data and velocity */
    random_data(input, dimx, dimy ,dimz , 1, 10);
    random_data(velocity, dimx, dimy ,dimz , 1, 10);
```

Stencil Code: Server Process (II)

```
/* Calculate number of shared points */
int edge_num_points = dimx * dimy * (dimz / num_comp_nodes + 4);
int int_num_points  = dimx * dimy * (dimz / num_comp_nodes + 8);
float *input_send_address = input;

/* Send input data to the first compute node */
MPI_Send(send_address, edge_num_points, MPI_REAL, first_node,
         DATA_DISTRIBUTE, MPI_COMM_WORLD );
send_address += dimx * dimy * (dimz / num_comp_nodes - 4);

/* Send input data to "internal" compute nodes */
for(int process = 1; process < last_node; process++) {
    MPI_Send(send_address, int_num_points, MPI_FLOAT, process,
             DATA_DISTRIBUTE, MPI_COMM_WORLD);
    send_address += dimx * dimy * (dimz / num_comp_nodes);
}

/* Send input data to the last compute node */
MPI_Send(send_address, edge_num_points, MPI_REAL, last_node,
         DATA_DISTRIBUTE, MPI_COMM_WORLD);
```

Stencil Code: Server Process (II)

```
float *velocity_send_address = velocity;

/* Send velocity data to compute nodes */
for(int process = 0; process < last_node + 1; process++) {
    MPI_Send(send_address, edge_num_points, MPI_FLOAT, process,
            DATA_DISTRIBUTE, MPI_COMM_WORLD);
    send_address += dimx * dimy * (dimz / num_comp_nodes);
}

/* Wait for nodes to compute */
MPI_Barrier(MPI_COMM_WORLD);

/* Collect output data */
MPI_Status status;
for(int process = 0; process < num_comp_nodes; process++)
    MPI_Recv(output + process * num_points / num_comp_nodes,
        num_points / num_comp_nodes, MPI_FLOAT, process,
        DATA_COLLECT, MPI_COMM_WORLD, &status );
}
```

Stencil Code: Server Process (III)

```
        /* Store output data */
    store_output(output, dimx, dimy, dimz);

    /* Release resources */
    free(input);
    free(velocity);
    free(output);
}
```

Stencil Code: Compute Process (I)

```
void compute_node_stencil(int dimx, int dimy, int dimz, int nreps ) {
    int np, pid;
    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
    MPI_Comm_size(MPI_COMM_WORLD, &np);

    unsigned int num_points       = dimx * dimy * (dimz + 8);
    unsigned int num_bytes        = num_points * sizeof(float);
    unsigned int num_ghost_points = 4 * dimx * dimy;
    unsigned int num_ghost_bytes  = num_ghost_points * sizeof(float);

    int left_ghost_offset   = 0;
    int right_ghost_offset  = dimx * dimy * (4 + dimz);

    float *input = NULL, *output = NULL, *prev = NULL, *v = NULL;

    /* Allocate device memory for input and output data */
    gmacMalloc((void **)&input,  num_bytes);
    gmacMalloc((void **)&output, num_bytes);
    gmacMalloc((void **)&prev, num_bytes);
    gmacMalloc((void **)&v, num_bytes);
```

Stencil Code: Compute Process (II)

```
MPI_Status status;
int left_neighbor  = (pid > 0) ? (pid - 1) : MPI_PROC_NULL;
int right_neighbor = (pid < np - 2) ? (pid + 1) : MPI_PROC_NULL;
int server_process = np - 1;

/* Get the input data from server process */
float *rcv_address = input + num_ghost_points * (0 == pid);
MPI_Recv(rcv_address, num_points, MPI_FLOAT, server_process,
         DATA_DISTRIBUTE, MPI_COMM_WORLD, &status );

/* Get the velocity data from server process */
rcv_address = h_v + num_ghost_points * (0 == pid);
MPI_Recv(rcv_address, num_points, MPI_FLOAT, server_process,
         DATA_DISTRIBUTE, MPI_COMM_WORLD, &status );
```

GPU Teaching Kit

Accelerated Computing