ECE408 / CS483/CSE408 Spring 2020

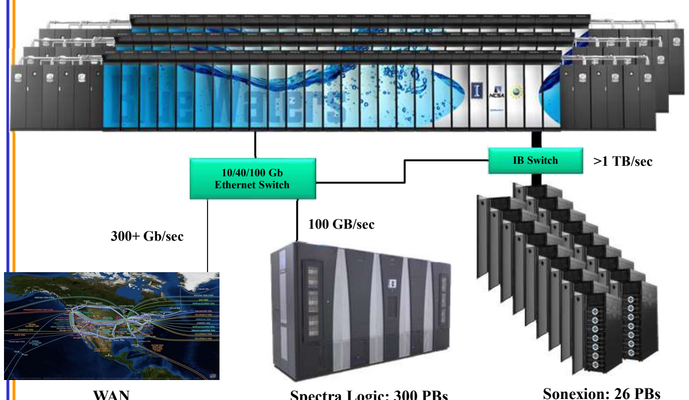
Applied Parallel Programming

Lecture 23: Joint CUDA-MPI Programming

Objective

- To be familiar with simple MPI-CUDA heterogeneous applications
  - understand the key sections of an MPI application
  - understand explicit communication in parallel computing applications

Blue Waters Computing System

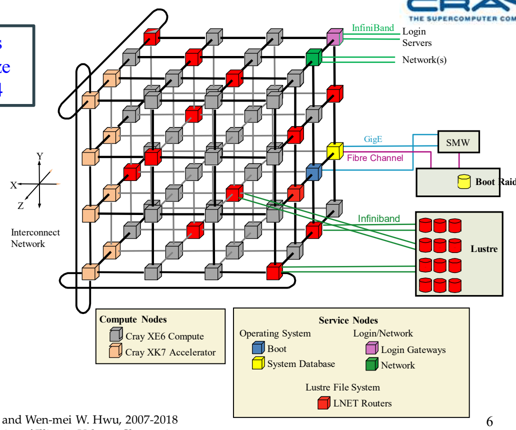


Blue Waters and Titan Computing Systems

System Attribute	**NCSA Blue Waters**	**ORNL Titan**
Vendors	Cray/AMD/NVIDIA	Cray/AMD/NVIDIA
Processors	Interlagos/Kepler	Interlagos/Kepler
Total Peak Performance (PF)	11.1	27.1
Total Peak Performance (CPU/GPU)	7.1/4	2.6/24.5
Number of CPU Chips	48,352	18,688
Number of GPU Chips	3,072	18,688
Amount of CPU Memory (TB)	1511	584
Interconnect	3D Torus	3D Torus
Amount of On-line Disk Storage (PB)	26	13.6
Sustained Disk Transfer (TB/sec)	>1	0.4-0.7
Amount of Archival Storage	300	15-30
Sustained Tape Transfer (GB/sec)	100	7

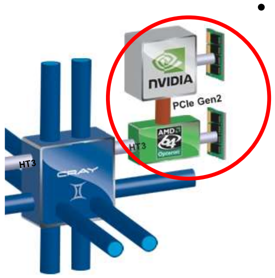
Gemini Interconnect Network

Blue Waters
3D Torus Size
23 x 24 x 24



Science Area	Number of Teams	Codes	Struct Grids	Unstruct Grids	Dense Matrix	Sparse Matrix	N-Body	Monte Carlo	FFT	PIC	Significant I/O
Climate and Weather	3	CESM, GCRM, CM1/WRF, HOMME	X	X		X		X			X
Plasmas/Magnetosphere	2	H3D(M),VPIC, OSIRIS, Magtail/UPIC	X				X		X		X
Stellar Atmospheres and Supernovae	5	PPM, MAESTRO, CASTRO, SEDONA, ChaNGa, MS-FLUKSS	X			X	X	X		X	X
Cosmology	2	Enzo, pGADGET	X			X	X				
Combustion/Turbulence	2	PSDNS, DISTUF	X						X		
General Relativity	2	Cactus, Harm3D, LazEV	X			X					
Molecular Dynamics	4	AMBER, Gromacs, NAMD, LAMMPS				X	X		X		
Quantum Chemistry	2	SIAL, GAMESS, NWChem			X	X	X	X			X
Material Science	3	NEMOS, OMEN, GW, QMCPACK			X	X	X	X			
Earthquakes/Seismology	2	AWP-ODC, HERCULES, PLSQR, SPECFEM3D	X	X			X				X
Quantum Chromo Dynamics	1	Chroma, MILC, USQCD	X		X	X					
Social Networks	1	EPISIMDEMICS									
Evolution	1	Eve									
Engineering/System of Systems	1	GRIPS,Revisit						X			
Computer Science	1			X	X	X			X		X

Cray XK7 Nodes

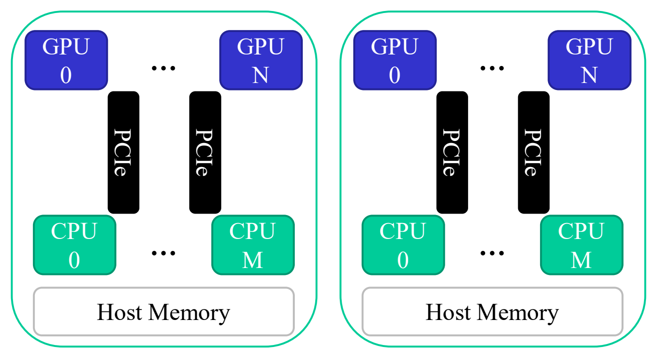


- Dual-socket Node
  - One AMD Interlagos chip
    - 8 core modules, 32 threads
    - 156.5 GFs peak performance
    - 32 GBs memory
      - 51 GB/s bandwidth
  - One NVIDIA Kepler chip
    - 1.3 TFs peak performance
    - 6 GBs GDDR5 memory
      - 250 GB/sec bandwidth
  - Gemini Interconnect
    - Same as XE6 nodes

Blue Waters contains 4,224 Cray XK7 compute nodes.

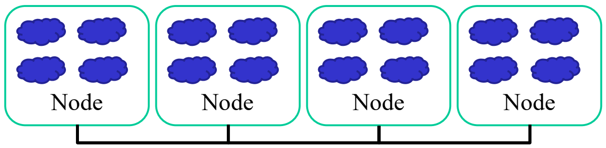
CUDA-based cluster

- Each node contains *N* GPUs



MPI Model

- Many processes distributed in a cluster



- Each process computes part of the output
- Processes communicate with each other through message passing (not global memory)
- Processes can synchronize through messages

MPI Initialization, Info

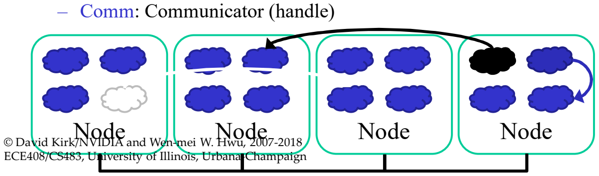
- `User launches an MPI job with X processes by executing in the command shell`
  - `MPIrun -np X`
- `int MPI_Init(int *argc, char ***argv)`
  - Initialize MPI
- `MPI_COMM_WORLD`
  - MPI group formed with all allocated nodes
- `int MPI_Comm_rank(MPI_Comm comm, int *rank)`
  - Rank of the calling process in group of comm
- `int MPI_Comm_size(MPI_Comm comm, int *size)`
  - Number of processes in the group of comm

Vector Addition: Main Process

```
int main(int argc, char *argv[]) {
    int vector_size = 1024 * 1024 * 1024;
    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(vector_size);
    else
        data_server(vector_size);

    MPI_Finalize();
    return 0;
}
```


MPI Sending Data

- `int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`
  - Buf: Initial address of send buffer
  - Count: Number of elements in send buffer (nonnegative integer)
  - Datatype: Datatype of each send buffer element
  - Dest: Rank of destination (integer)
  - Tag: Message tag (integer)
  - Comm: Communicator (handle)



MPI Receiving Data

- `int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`
  - Buf: Starting address of receive buffer
  - Count: Maximum number of elements in receive buffer (non-negative integer)
  - Datatype: Datatype of each receive buffer element
  - Source: Rank of source (integer)
  - Tag: Message tag (integer)
  - Comm: Communicator (handle)
  - Status: Status object

Vector Addition: Server Process (I)

```
void data_server(unsigned int vector_size) {  // runs on rank 0 only
    int          np;
    unsigned int num_bytes = vector_size * sizeof(float);

    /* Set MPI Communication Size */
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    int          num_nodes = np - 1;
    unsigned int vector_part = vector_size / num_nodes;

    /* Allocate input data */
    float* input_a = (float *)malloc(num_bytes);
    float* input_b = (float *)malloc(num_bytes);
    float* output  = (float *)malloc(num_bytes);
    if(input_a == NULL || input_b == NULL || output == NULL) {
        printf("Server couldn't allocate memory\n");
        MPI_Abort( MPI_COMM_WORLD, 1 );
    }
    /* Initialize input data */
    random_data(input_a, vector_size, 1, 10);
    random_data(input_b, vector_size, 1, 10);
```


Vector Addition: Server Process (II)

```
    /* Send data to compute nodes */
    for(int process = 0; process < num_nodes; process++) {

        MPI_Send(input_a + process * vector_part, vector_part,
                 MPI_FLOAT, process, DATA_DISTRIBUTE, MPI_COMM_WORLD);

        MPI_Send(input_b + process * vector_part, vector_part,
                 MPI_FLOAT, process, DATA_DISTRIBUTE, MPI_COMM_WORLD);
    }

    // Example of a barrier in MPI.  Here we want to avoid doing a
    // barrier, since sends and receives are blocking anyway, and we
    // want to avoid synchronizing the workers’ attempts to use the
    // incoming network bandwidth to the “server,” which is the
    // communication bottleneck here.

    // MPI_Barrier(MPI_COMM_WORLD);
```


Vector Addition: Server Process (III)

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

    /* Store output data */
    store_output(output, vector_size);

    /* Release resources */
    free(input_a);
    free(input_b);
    free(output);
}
```


Vector Addition: Compute Process (I)

```
void compute_node(unsigned int vector_size) {
    int          np;

    MPI_Comm_size(MPI_COMM_WORLD, &np);
    int          server_process = np - 1; // also # of compute processes
    unsigned int vector_part = vector_size / server_process;
    unsigned int num_bytes = vector_part * sizeof(float);

    /* Alloc host memory */
    float* input_a = (float *)malloc(num_bytes);
    float* input_b = (float *)malloc(num_bytes);
    float* output  = (float *)malloc(num_bytes);

    /* Get the input data from server process */
    MPI_Status   status;
    MPI_Recv(input_a, vector_part, MPI_FLOAT, server_process,
            DATA_DISTRIBUTE, MPI_COMM_WORLD, &status);
    MPI_Recv(input_b, vector_part, MPI_FLOAT, server_process,
            DATA_DISTRIBUTE, MPI_COMM_WORLD, &status);
```


20

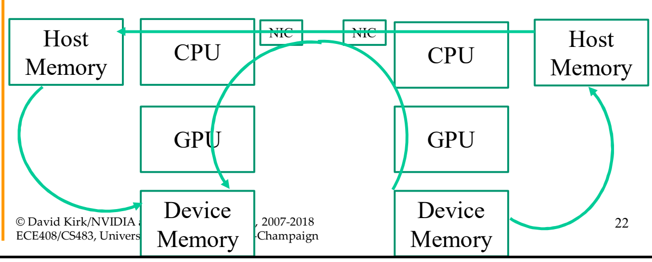
Vector Addition: Compute Process (II)

```
    /* Compute the partial vector addition */
    for(int i = 0; i < vector_part; ++i) {
        output[i] = input_a[i] + input_b[i];
    }

    /* Or, can offload to GPU here */
    /* cudaMalloc(), cudaMemcpy(), kernel launch, SYNCHRONIZE */

    // Example of a barrier in MPI (want to avoid here)
    // MPI_Barrier(MPI_COMM_WORLD);

    /* Send the output */
    MPI_Send(output, vector_part, MPI_FLOAT,
            server_process, DATA_COLLECT, MPI_COMM_WORLD);

    /* Release memory */
    free(input_a);
    free(input_b);
    free(output);
}
```


21

Current Trends

- GPU Direct
  - MPI_Send and MPI_Receive deals directly with GPU memory
  - Requires support from NIC vendors



22

QUESTIONS?

23