		授業計画	課題
05/08	第９回	~~密行列の直接解法~~ CUDA行列積	LU 分解の原理を理解し，その最適化・並列化 と LINPACK ベンチマークの特徴を理解できる
05/11	第１０回	~~密行列の固有値解析~~ CPU行列積(BLISLAB)	固有値・固有ベクトルの求め方を習得し対角化 正規直交化の高速化手法を理解できる
05/15	第１１回	~~疎行列の直接解法~~ CUDA+MPI	AMD や Nested dissection などの並べ替え法と skyline・multifrontal 法の高速化手法を理解
05/18	第１２回	~~疎行列の反復解法~~ 深層学習	正定値行列や条件数の概念を理解し，Jacobi 法 CG 法，GMRES 法の相違点を理解
05/22	第１３回	~~反復法の前処理~~	前処理による条件数やスペクトル半径への影響や 前処理された CG 法の効果を理解できる
05/25	第１４回	~~マルチグリッド法~~	V-cycle における緩和・縮約・補間の役割を理解し 前処理法としての効果を理解できる
05/29	第１５回	~~FMM，H 行列~~	多重極展開，低ランク近似の概念を理解し 木構造の果たす役割を理解できる

並列プログラミング言語: SIMD, OpenMP, MPI, GPU

並列計算ライブラリ: BLAS, LAPACK, FFTW

高性能計算支援ツール: Compiler flags, Profiler, Debugger

TSUBAME job submission

MPIの復習

```
#include "mpi.h"
#include <cstdio>
int main(int argc, char ** argv) {
  MPI_Init(&argc, &argv);
  int mpisize, mpirank;
  MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
  printf("rank: %d/%d\n",mpirank,mpisize);
  MPI_Finalize();
}
```

```
> mpicxx step01.cpp
> mpirun -np 2 ./a.out
```

CUDAの復習

```
#include <cstdio>

__global__ void mykernel(void) {
}

int main() {
  mykernel<<<1,1>>>();
  printf("Hello CPU\n");
  return 0;
}
```

```
> nvcc step01.cu
> ./a.out
```

CUDA+MPI

```
#include <mpi.h>
#include <cstdio>

__global__ void mykernel(void) {
}

int main(int argc, char **argv) {
  MPI_Init(&argc, &argv);
  int mpisize, mpirank;
  MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
  mykernel<<<1,1>>>();
  printf("rank: %d/%d\n",mpirank,mpisize);
  MPI_Finalize();
}
```

CUDA+MPI

```
> mpicxx step01.cu
step01.cu: file not recognized: File format not recognized
> nvcc step01.cu
step01.cu:1:17: error: mpi.h: No such file or directory
> export CPATH=$CPATH:/usr/apps.sp3/isv/intel/ParallelStudioXE/
ClusterEdition/2016-Update3/compilers_and_libraries_2016.3.210/
linux/mpi/intel64/include
> nvcc step01.cu
/tmp/tmpxft_000066d1_00000000-17_step02.o: In function `main':
tmpxft_000066d1_00000000-4_step02.cudafe1.cpp:(.text+0x165): undefined reference to `MPI_Init’
tmpxft_000066d1_00000000-4_step02.cudafe1.cpp:(.text+0x173): undefined reference to `MPI_Comm_size'
tmpxft_000066d1_00000000-4_step02.cudafe1.cpp:(.text+0x181): undefined reference to `MPI_Comm_rank'
tmpxft_000066d1_00000000-4_step02.cudafe1.cpp:(.text+0x191): undefined reference to
`MPI_Get_processor_name'
tmpxft_000066d1_00000000-4_step02.cudafe1.cpp:(.text+0x1d8): undefined reference to `MPI_Barrier'
tmpxft_000066d1_00000000-4_step02.cudafe1.cpp:(.text+0x28b): undefined reference to `MPI_Finalize'
> export LIBRARY_PATH=$LIBRARY_PATH:/usr/apps.sp3/isv/intel/
ParallelStudioXE/ClusterEdition/2016-Update3/
compilers_and_libraries_2016.3.210/linux/mpi/intel64/lib
> nvcc step01.cu -lmpi
```

step02.cu

```
#include <mpi.h>
#include <stdio.h>

__global__ void GPU_Kernel() {
  printf(" GPU block  : %d / %d  GPU thread : %d / %d\n",
         blockIdx.x, gridDim.x, threadIdx.x, blockDim.x);
}

int main(int argc, char **argv) {
  int mpisize, mpirank, gpusize, gpurank;
  cudaGetDeviceCount(&gpusize);
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
  cudaSetDevice(mpirank % gpusize);
  cudaGetDevice(&gpurank);
  for (int irank=0; irank!=mpisize; irank++) {
    MPI_Barrier(MPI_COMM_WORLD);
    if (mpirank == irank) {
      printf("MPI rank    : %d / %d  GPU device : %d / %d\n",
             mpirank, mpisize, gpurank, gpusize);
      GPU_Kernel<<<2,2>>>();
      cudaThreadSynchronize();
    }
  }
  MPI_Finalize();
}
```

参考資料

https://www.cs.utexas.edu/users/hfingler/GPU.pdf

http://on-demand.gputechconf.com/gtc/2016/presentation/s6142-jiri-kraus-multi-gpu-programming-mpi.pdf

http://tsubame.gsic.titech.ac.jp/docs/guides/tsubame2/html/programming.html

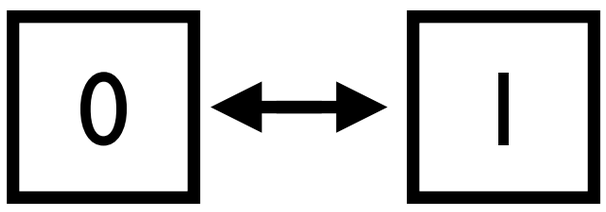
http://mug.mvapich.cse.ohio-state.edu/static/media/mug/presentations/2016/MUG16_GPU_tutorial_V5.pdf

http://readthedocs.org/projects/chainermn/downloads/pdf/latest/

http://news.mynavi.jp/articles/2015/04/16/gtc2015_multigpu02/

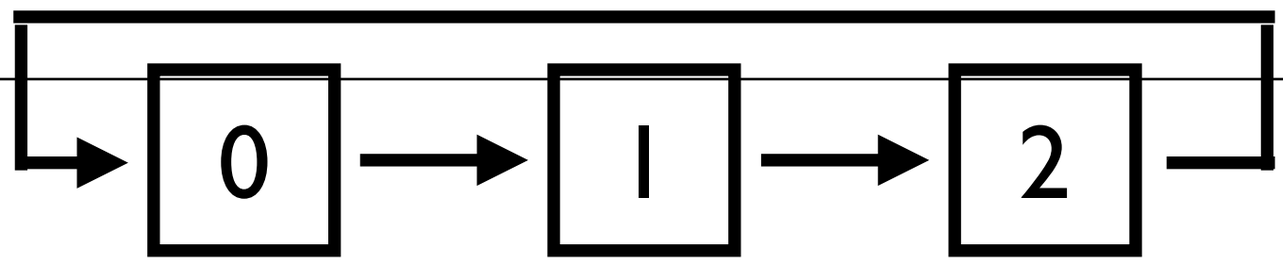
mpi/step10.cpp

```
#include "mpi.h"
#include <cstdio>
int main(int argc, char ** argv) {
  MPI_Init(&argc, &argv);
  int mpisize, mpirank;
  MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
  int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
  for(int i=0; i<4; i++)
    send[i] = mpirank+10*i;
  if(mpirank==0) {
    MPI_Send(send, 4, MPI_INT, 1, 0, MPI_COMM_WORLD);
    MPI_Recv(recv, 4, MPI_INT, 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  } else if(mpirank==1) {
    MPI_Send(send, 4, MPI_INT, 0, 1, MPI_COMM_WORLD);
    MPI_Recv(recv, 4, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }
  printf("rank%d: send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
         send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
  MPI_Finalize();
}
```



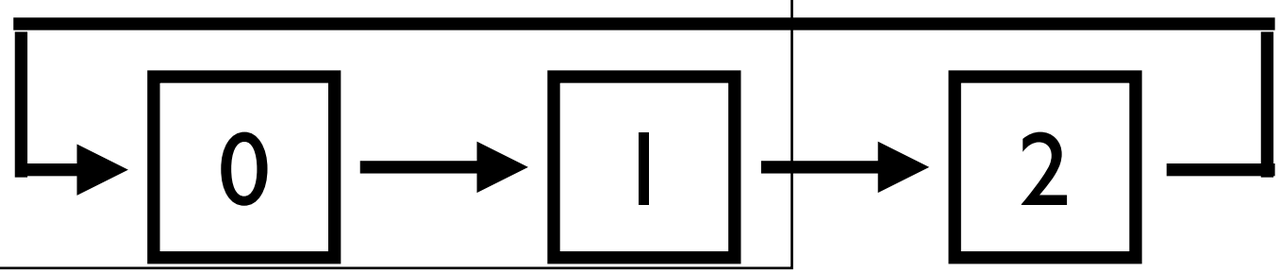
step03.cu

```
#include "mpi.h"
#include <cstdio>
int main(int argc, char ** argv) {
  MPI_Init(&argc, &argv);
  int mpisize, mpirank;
  MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
  int send[4] = {0,0,0,0}, recv[4] = {0,0,0,0};
  for(int i=0; i<4; i++)
    send[i] = mpirank+10*i;
  int sendrank = (mpirank + 1) % mpisize;
  int recvrank = (mpirank - 1 + mpisize) % mpisize;
  MPI_Send(send, 4, MPI_INT, sendrank, 0, MPI_COMM_WORLD);
  MPI_Recv(recv, 4, MPI_INT, recvrank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  for (int irank=0; irank<mpisize; irank++) {
    MPI_Barrier(MPI_COMM_WORLD);
    if (irank == mpirank) {
      printf("rank%d: send_rank=%d, recv_rank=%d\n", mpirank, sendrank, recvrank);
      printf("send=[%d %d %d %d], recv=[%d %d %d %d]\n",mpirank,
             send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    }
  }
  MPI_Finalize();
}
```



step04.cu

```
#include <mpi.h>
#include <stdio.h>
__global__ void GPU_Kernel(int *send) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  send[i] += 10 * i;
}
int main(int argc, char **argv) {
  int mpisize, mpirank;
  int size = 4 * sizeof(int);
  int *send = (int *)malloc(size);
  int *recv = (int *)malloc(size);
  int *d_send, *d_recv;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
  for(int i=0; i<4; i++)
    send[i] = mpirank;
  cudaSetDevice(mpirank % mpisize);
  cudaMalloc((void **) &d_send, size);
  cudaMalloc((void **) &d_recv, size);
  cudaMemcpy(d_send, send, size, cudaMemcpyHostToDevice);
  GPU_Kernel<<<2,2>>>(d_send);
  cudaMemcpy(send, d_send, size, cudaMemcpyDeviceToHost);
  int sendrank = (mpirank + 1) % mpisize;
  int recvrank = (mpirank - 1 + mpisize) % mpisize;
  MPI_Send(send, 4, MPI_INT, sendrank, 0, MPI_COMM_WORLD);
  MPI_Recv(recv, 4, MPI_INT, recvrank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  for (int irank=0; irank<mpisize; irank++) {
    MPI_Barrier(MPI_COMM_WORLD);
    if (mpirank == irank) {
      printf("rank%d: send_rank=%d, recv_rank=%d\n", mpirank, sendrank, recvrank);
      printf("send=[%d %d %d %d], recv=[%d %d %d %d]\n",
             send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    }
  }
  free(send); free(recv);
  cudaFree(d_send); cudaFree(d_recv);
  MPI_Finalize();
}
```



step05.cu

```
#include <mpi.h>
#include <stdio.h>
__global__ void GPU_Kernel(int *send) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  send[i] += 10 * i;
}
int main(int argc, char **argv) {
  int mpisize, mpirank;
  int size = 4 * sizeof(int);
  int *send = (int *)malloc(size);
  int *recv = (int *)malloc(size);
  int *d_send, *d_recv;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &mpisize);
  MPI_Comm_rank(MPI_COMM_WORLD, &mpirank);
  for(int i=0; i<4; i++)
    send[i] = mpirank;
  cudaSetDevice(mpirank % mpisize);
  cudaMalloc((void **) &d_send, size);
  cudaMalloc((void **) &d_recv, size);
  cudaMemcpy(d_send, send, size, cudaMemcpyHostToDevice);
  GPU_Kernel<<<2,2>>>(d_send);
  cudaMemcpy(send, d_send, size, cudaMemcpyDeviceToHost);
  int sendrank = (mpirank + 1) % mpisize;
  int recvrank = (mpirank - 1 + mpisize) % mpisize;
  MPI_Send(send, 4, MPI_INT, sendrank, 0, MPI_COMM_WORLD);
  MPI_Recv(recv, 4, MPI_INT, recvrank, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  for (int irank=0; irank<mpisize; irank++) {
    MPI_Barrier(MPI_COMM_WORLD);
    if (mpirank == irank) {
      printf("rank%d: send_rank=%d, recv_rank=%d\n", mpirank, sendrank, recvrank);
      printf("send=[%d %d %d %d], recv=[%d %d %d %d]\n",
             send[0],send[1],send[2],send[3],recv[0],recv[1],recv[2],recv[3]);
    }
  }
  free(send); free(recv);
  cudaFree(d_send); cudaFree(d_recv);
  MPI_Finalize();
}
```