		授業計画	課題
04/06	第１回	微分方程式の離散化	前進・後退・中心差分，高次の差分を用いて 微分方程式を離散化し，誤差を評価できる
04/10	第２回	有限差分法	時間積分の安定性や高次精度の積分を理解し 移流・拡散・波動方程式を解析できる
04/13	第３回	有限要素法	Galerkin 法，テスト関数，isoparametric 要素 の概念を理解し，弾性方程式を解析できる
04/17	第４回	~~スペクトル法~~ Python ctypes	Fourier・Chebyshev・Legendre・Bessel などの 直交基底関数による離散化の利点を説明できる
04/20	第５回	~~境界要素法~~ OpenMP	逆行列と δ 関数・Green 関数の関係を理解し 境界積分方程式を用いた解析ができる
04/24	第６回	~~分子動力学法~~ MPI	時間積分の symplectic 性や熱浴の概念を理解し 分子間に働く保存力の動力学を解析できる
04/27	第７回	~~Smooth particle hydrodynamics (SPH) 法~~ SIMD	微分演算子の動径基底関数による離散化と その保存性・散逸性を評価できる
05/01	第８回	~~Particle mesh 法~~ GPU	粒子と格子の両方の離散化を組み合わせる場合の 離散点からの補間法と高次モーメントの保存法

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

step01.cpp: parallel for

```
#include <cstdlib>
#include <cstdio>
#include <sys/time.h>
int main(int argc, char ** argv) {
  struct timeval tic, toc;
  int n = atoi(argv[1]);
  double * a = new double [n];
  double * b = new double [n];
  gettimeofday(&tic, NULL);
#pragma omp parallel for
  for (int i=1; i<n; i++) {
    b[i] = (a[i] + a[i-1]) / 2.0;
  }
  gettimeofday(&toc, NULL);
  printf("%lf s\n",toc.tv_sec-tic.tv_sec+(toc.tv_usec-tic.tv_usec)*1e-6);
  delete[] a;
  delete[] b;
}
```

```
>g++ -fopenmp step01.cpp
>./a.out 1000000
```

step02.cpp: barrier

```
#include <cstdio>
#include <omp.h>
int main() {
  int x = 2;
#pragma omp parallel num_threads(2) shared(x)
  {
    if (omp_get_thread_num() == 0) {
      x = 5;
    } else {
      printf("1: Thread# %d: x = %d\n", omp_get_thread_num(),x );
    }
#pragma omp barrier
    if (omp_get_thread_num() == 0) {
      printf("2: Thread# %d: x = %d\n", omp_get_thread_num(),x );
    } else {
      printf("3: Thread# %d: x = %d\n", omp_get_thread_num(),x );
    }
  }
}
```

step03.cpp: nowait

```
#include <cmath>
#include <cstdlib>
#include <cstdio>
#include <sys/time.h>
int main(int argc, char ** argv) {
  struct timeval tic, toc;
  int n = atoi(argv[1]);
  double * a = new double [n];
  double * b = new double [n];
  double * y = new double [n];
  double * z = new double [n];
  gettimeofday(&tic, NULL);
#pragma omp parallel
  {
#pragma omp for nowait
    for (int i=1; i<n; i++)
      b[i] = (a[i] + a[i-1]) / 2.0;
#pragma omp for nowait
    for (int i=0; i<n; i++)
      y[i] = sqrt(z[i]);
  }
  gettimeofday(&toc, NULL);
  printf("%lf s\n",toc.tv_sec-tic.tv_sec+(toc.tv_usec-tic.tv_usec)*1e-6);
  delete[] a;
  delete[] b;
  delete[] y;
  delete[] z;
}
```

step04.cpp: lastprivate

```
#include <cstdio>
#include <omp.h>
int main() {
  int jlast, klast;
#pragma omp parallel
  {
#pragma omp for collapse(2) lastprivate(jlast, klast)
    for (int k=1; k<=2; k++) {
      for (int j=1; j<=3; j++) {
        jlast = j;
        klast = k;
      }
    }
#pragma omp single
    printf("%d %d\n", klast, jlast);
  }
}
```

step05.cpp: sections

```
#include <cstdio>
#include <omp.h>
int main() {
  int section_count = 0;
  omp_set_dynamic(0);
  omp_set_num_threads(2);
#pragma omp parallel
#pragma omp sections
  {
#pragma omp section
    {
      section_count++;
      printf("section_count %d\n",section_count);
    }
#pragma omp section
    {
      section_count++;
      printf("section_count %d\n",section_count);
    }
  }
}
```

step06.cpp: Fibonacci

```
#include <cstdlib>
#include <cstdio>
int fib(int n) {
  int i,j;
  if (n<2) return n;
#pragma omp task shared(i)
  i = fib(n-1);
#pragma omp task shared(j)
  j = fib(n-2);
#pragma omp taskwait
  return i+j;
}

int main(int argc, char ** argv) {
  int n = atoi(argv[1]);
  printf("%d\n",fib(n));
}
```

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

step07.cpp: depend

```
#include <cstdio>
int main() {
  int x = 1;
#pragma omp task shared(x) depend(out: x)
  x = 2;
#pragma omp task shared(x) depend(in: x)
  printf("x + 1 = %d\n", x+1);
#pragma omp task shared(x) depend(in: x)
  printf("x + 2 = %d\n", x+2);
}
```

step08.cpp: atomic

```
#include <cstdio>
int main() {
  float x[10];
  int index[1000];
  for (int i=0; i<1000; i++) {
    index[i] = i % 10;
  }
  for (int i=0; i<10; i++)
    x[i] = 0.0;
#pragma omp parallel for shared(x, index)
  for (int i=0; i<1000; i++) {
#pragma omp atomic update
    x[index[i]]++;
  }
  for (int i=0; i<10; i++)
    printf("%d %f\n",i, x[i]);
}
```

step09.cpp: ordered

```
#include <cstdio>
int main() {
#pragma omp parallel for ordered schedule(dynamic)
  for (int i=0; i<100; i+=5) {
#pragma omp ordered
    printf("%d\n",i);
  }
}
```

step10.cpp: threadprivate

```
#include <cstdio>
int counter = 0;
#pragma omp threadprivate(counter)

int main() {
#pragma omp parallel for
  for (int i=0; i<100; i++) {
    counter++;
  }
  printf("%d\n",counter);
}
```