ハイパフォーマンスコンピューティング(5)

並列化による弊害が起きる例

- 並列化したい処理に依存関係がある場合

```
for (i = 1; i < n; i++) {
    a[i] = a[i-1] + i;
}
```

この場合は、i番目の処理に (i-1)番目の結果が必要なため

下記のようにすると結果がおかしくなる。

```
#pragma  omp  parallel  for
for (i = 1; i < n; i++) {
    a[i] = a[i-1] + i;
}
```

練習問題：ループ処理に依存関係がある場合の並列化による弊害を見てみよう。

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

int a[10];

int main(void)
{
  int i, n = 10;

  printf("serial\n");
  /* initializaiton */
  for (i = 0; i < n; i++) {
    a[i] = 0;
  }

  for (i = 1; i < n; i++) {
    a[i] = a[i-1] + i;
  }

  for (i = 0; i < n; i++) {
    printf("a[%d] = %d\n", i, a[i]);
  }
```

```
  printf("parallel\n");
  /* initializaiton */
  for (i = 0; i < n; i++) {
    a[i] = 0;
  }

  #pragma omp parallel for
  for (i = 1; i < n; i++) {
    a[i] = a[i-1] + i;
  }

  for (i = 0; i < n; i++) {
    printf("a[%d] = %d\n", i, a[i]);
  }

  return 0;
}
```

依存関係を除去できる例

- 依存関係がある場合でも、それを除去可能なときがある。

```
for (i = 0; i < n - 1; i++) {
    a[i] = a[i+1] + i;
}
```

この場合も依存関係があるため、このまま並列化すると結果がおかしくなるが、事前の処理によって、依存関係を除去できる。

```
#pragma omp parallel for
for (i = 0; i < n; i++) {
    b[i] = a[i];
}

#pragma omp parallel for
for (i = 0; i < n - 1; i++) {
    a[i] = b[i+1] + i;
}
```

事前にデータを
コピーしておけば・・

依存関係がなくなる！

演習問題：依存関係を除去してみよう。

(1) 下記のプログラムを作成し、何度か実行して結果が不定になることを確認する。
(2) 依存関係を除去した並列化のコードを追加してみよう。

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

int a[5];

int main(void)
{
  int i, n = 5;

  printf("serial\n");
  /* initializaiton */
  for (i = 0; i < n; i++) {
    a[i] = 1;
  }
  for (i = 0; i < n - 1; i++) {
    a[i] = a[i+1] + i;
  }
  for (i = 0; i < n; i++) {
    printf("a[%d] = %d\n", i, a[i]);
  }
```

```
  printf("parallel 1\n");
  /* initializaiton */
  for (i = 0; i < n; i++) {
    a[i] = 1;
  }

  #pragma omp parallel for
  for (i = 0; i < n - 1; i++) {
    a[i] = a[i+1] + i;
  }

  for (i = 0; i < n; i++) {
    printf("a[%d] = %d\n", i, a[i]);
  }
```

(2)でコードを追加

```
  return 0;
}
```

バリア同期

- #pragma omp barrier：すべてのスレッドの足並みをそろえる

⇒全てのスレッドの処理が終わる前に、別の処理が先に進むと困る場合や、時間の計測をする場合に、バリア同期を使う。

```
#pragma omp parallel
{
  ...

  #pragma omp barrier      ←バリア同期
  tic = omp_get_wtime();

   ...（時間計測したい処理）

  #pragma omp barrier      ←バリア同期
  toc = omp_get_wtime();


   ...

}
printf("time = %f\n", toc - tic);
```

練習問題：バリア同期をする場合としない場合で結果がどうなるか見てみよう。（実行時のタイミングによるので、何度か実行してみよう）

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

#define n 4

int main(void)
{
  int i, s, a[n], id;
  double v;

  #pragma omp parallel num_threads(n) private(id)
  {
    id = omp_get_thread_num();
    a[id] = id + 1; /* a = [1,2,...,n] */

    /* barrier sync */
    #pragma omp barrier
```

1スレッドで実行→

```
    #pragma omp single
    {
      s = 0;
      for (i = 0; i < n; i++) {
        s += a[i];
      }
      v = (double)s/n;
    }
  }
  printf("average = %f\n", v);

  return 0;
}
```

←バリア同期

OpenMPの補助指示文

- private
- firstprivate
- lastprivate
- shared
- num_threads
- など

OpenMPの補助指示文： private

■変数 j を各スレッドで別の変数として確保しながら実行される。

```
#pragma omp parallel for private(j)
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        y[i] += A[i][j]*x[j];
    }
}
```

・並列化対象のfor文のカウンタ i は、強制的にprivateとなる。
・何も指定されていない変数は、shared（共有）となる。

OpenMPによる行列積の並列化(C言語)

■最も外側のループについて並列化を行う

```
#pragma omp parallel for private(j, k)
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        for (k=0; k<n; k++) {
            C[i][j] += A[i][k]*B[k][j];
        }
    }
}
```

内側のループの並列化を行うと、
外側のループが回る度に
スレッドの生成を行うので
オーバーヘッドが大きい

OpenMPによる行列積の並列化(Fortran)

■最も外側のループについて並列化を行う

```
!$omp  parallel  do  private(j, k)
do i=1,n
    do j=1,n
        do k=1,n
            C(i,j) = C(i,j) + A(i,k)*B(k,j);
        end do
    end do
end do
```

演習問題

- まず、行列積をOpenMPを用いて並列化し、さらに、アンローリングやブロック化などのチューニングを行い、高速化しましょう。
- 時間の計測にはomp_get_wtime()を用いること.