



[目次](#) >> OpenMP

## OpenMP

OpenMPは複数のCPU(複数コアを含む)を持った計算機上での並列化に威力を発揮する。OpenMPを使う最大の利点は、OpenMPに対応したコンパイラであれば、非常に簡単に並列化できる点である。

現在、gcc、Visual C++、およびIntelコンパイラなど主要なコンパイラはOpenMPに対応している。

習得も他の並列化技法に比べて比較的容易である。

なお、速度を最優先にする場合、単一コンピュータ上で動かした場合でも、メモリのローカリティのためかOpenMPよりMPIの方が効率のよいことが多い。MPIに関しては[こちら](#)を参照。

なお、インテルがOpenMP初心者向けに非常にわかりやすい文書を公開している。

### OpenMPプログラムのコンパイル

```
include "omp.h"
```

OpenMPの各種関数を使わない場合、`#pragma omp`で始まる指示をソースコード内に書き込み、下記のコンパイルスイッチをつけてコンパイルするだけで、並列化される。

#### インテルコンパイラ

インテルコンパイラ(icpc)の場合は、Linux用では`-openmp`を、Windows用では、`/Qopenmp`をつけてコンパイルする。なお、インテルコンパイラは、バージョン9よりAMDのプロセッサにも対応するようになったが、バージョン9とAMDプロセッサの組み合わせでのOpenMPはなぜか実行効率が非常に悪い。これはバージョン10にすることにより解決される。

#### Visual C++

Visual C++(cl)の場合は、`/openmp`をつけてコンパイルする。

Visual C++は2005よりOpenMPがサポートされている。なお、無料のExpressバージョンには含まれていないが、Windows SDK for Windows Server 2008 and .NET Framework 3.5を導入することによって、使えるようになる。詳細は[こちら](#)。

#### GCC

gcc(g++)の場合は、`-fopenmp`をつけてコンパイルする。GCCはバージョン4.2から正式にサ

ポートされるようになった。

## OpenMPを使った並列計算

ここではOpenMPを使った並列計算について解説する。

### OpenMPを使ってコンパイルしているかをプログラム中で知るには

OpenMPを使ってコンパイルしているかをプログラム中で知るには、`_OPENMP`が定義されているかで判別する。具体的には、

```
#ifdef _OPENMP
    //OpenMPを使ったコード
#else
    //OpenMPを使わない場合のコード
#endif
```

### forループを並列化するには？

OpenMPで並列化する場合、最も使われるのがforループの並列化。forループの並列化には`#pragma omp parallel for`を使う。

```
#ifdef _OPENMP
#pragma omp parallel for
#endif
for(int i=0; i<N; i++)
{
    //ここの処理を書く
}
```

### 複数のforループが連続する場合は？

下記のように書くと毎回forループに入るたびにスレッドの生成コストが発生する。

```
#ifdef _OPENMP
#pragma omp parallel for
#endif
for(int i=0; i<N; i++)
{
    //ここの処理を書く
}
#ifdef _OPENMP
#pragma omp parallel for
#endif
for(int i=0; i<N; i++)
{
    //ここの処理を書く
}
#ifdef _OPENMP
```

```
#pragma omp parallel for
#endif
for(int i=0;i<N;i++)
{
    //ここの処理を書く
}
```

下記のようにすると、スレッド生成は一回ですむ。

```
#ifdef _OPENMP
#pragma omp parallel
#endif
{
    #ifdef _OPENMP
    #pragma omp for
    #endif
    for(int i=0;i<N;i++)
    {
        //ここの処理を書く
    }
    #ifdef _OPENMP
    #pragma omp for
    #endif
    for(int i=0;i<N;i++)
    {
        //ここの処理を書く
    }
    #ifdef _OPENMP
    #pragma omp for
    #endif
    for(int i=0;i<N;i++)
    {
        //ここの処理を書く
    }
}
```

## ブロックごとに並列化するには？

いくつかの異なった処理を並列して行いたい場合は、`#pragma omp parallel` および `#pragma omp sections` を使う。

```
#pragma omp parallel
#pragma omp sections
{
    #pragma omp section
    {
        //並列させたい処理1
    }
    #pragma omp section
    {
        //並列させたい処理2
    }
    #pragma omp section
    {
        //並列させたい処理3
    }
}
```

## プロセッサの数を取得するには？

omp\_get\_num\_procs()を使う。実際にはプロセッサの数\*コア数が返される。Hyper-threadingが有効になっているCPUでは、それも含めてカウント(すなわち2倍)される。

```
#ifdef _OPENMP
cout<<"The number of processors is "<<omp_get_num_procs()<<endl;
#endif
```

## スレッド数を取得するには？

現在の並列数を取得するには、omp\_get\_max\_threads()関数を使う。

```
#ifdef _OPENMP
cout<<"OpenMP : Enabled (Max # of threads = "<<omp_get_max_threads()<<")"<<endl;
#endif
```

## 並列数を変更するには？

通常は、環境変数OMP\_NUM\_THREADSから取得して並列数が決まる。  
omp\_set\_num\_threads関数を使うと、プログラム中で指定できる。

```
#ifdef _OPENMP
omp_set_num_threads(4);
#endif
```

## 最適なスケジューリング方法を選択する

タスクの内容によっては、スケジューリング法を変えることによって、より効率的に多くのCPUを使うことができる。

スケジューリングの指定方法は、

```
#pragma omp parallel for schedule(type[, size])
```

の様に、#pragma omp parallel forに続けて指定する。

type指定できるスケジューリングには

- static
- dynamic
- guided
- runtime

の4つがある。このうち、runtimeは実行時に環境変数でしているという意味で、実際のスケジューリング方法としては3つである。

staticはforループが開始される時点で何番目のループがどのスレッドで実行するかが、あらかじめ決まっている。sizeで指定したチャンク数で、順次実行されていく。デフォルトのチャンク数はループ数/プロセッサ数である。もし各スレッドの計算量が均等であれば、もっとも効率がよい。

dynamicはsizeで指定したチャンク数ごとに、あいているスレッドに割り当てられていく。デフォルトのチャンク数は1。これは各スレッドの計算量がばらばらな時に最も有効である。

guidedははじめ大きなチャンク数で割り当て、徐々にチャンク数が小さくなっていくものである。初期にチャンク数が大きいため効率がよい上、最後の方ではチャンク数を小さくして、各プロセッサに極力均等に割り当てるようにするもので、staticとdynamicの中間といったところである。

---

本文中のFC4はFedora ProjectのFedora Core 4を、FC5はFedora Core 5を、FC6はFedora Core 6をopenSUSEはNovellのSUSE Linux OSSを表します。Fedora7以降は、単にFedora7、Fedora8、Fedora9、Fedora10、Fedora11、Fedora12、Fedora13、Fedora14、Fedora15と表示しています。Ubuntuは、必要に応じて10.04、11.04のようにバージョン番号をつけて区別しています。MandrivaはMandriva Linuxを表します。

ここに登場するドメイン名やIPアドレスなどはフィクションです。実在の人物・団体等とは一切関係がありません。

実際に使用する際は、各自の環境に合わせて書き換えてください。

もし何か間違いなどありましたら[こちら](#)からご連絡ください

リンクに許可は不要です。

Copyright (C) 2011 [Chikuma Engineering Co., Ltd.](#) All Rights Reserved.