OpenMP* 4.x における拡張

OpenMP 4.0 と 4.5 の機能拡張

sugatech

内容

- OpenMP* 3.1 から 4.0 への拡張
- OpenMP* 4.0 から 4.5 への拡張

追加された機能（3.1 -> 4.0)

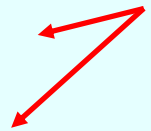
- C/C++ 配列シンタックスの拡張
- SIMD と SIMD 対応関数
- デバイスオフロード
- task 構文の依存性
- taskgroup 構文
- cancel 句と cancellation point 句
- 追加されたランタイム API
- 追加された環境変数

other¥task_dep.c

Task の依存性

Task 構文で depend 節を使用し依存性を明記

```
#include <stdio.h>
int main() {
   int x = 0, y = 0;
   #pragma omp parallel
   #pragma omp single
   {
      #pragma omp task shared(x) depend(out: x)
        x = 1;
      #pragma omp task shared(x) depend(in: x)
        y = x + 1;
        printf("x = %d y = %d¥n", x, y);
   }
   return 0;
}
```



タスク構文間でのデータ依存を明示的に指定できるようになりました。
この例では、x =2 がセットされるまで、printf は待機します

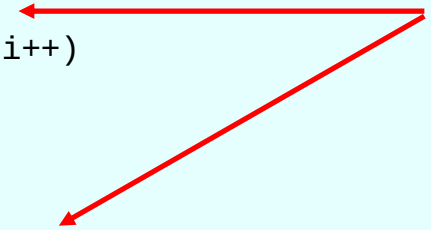
フロー依存（out,in)、アンチ依存(in,out)、出力依存(out,out) を制御できます

Taskgroup 構文

other¥taskgroup.c

Taskgroup 句は、taskgroup 内で生成された子タスク（孫タスク）の完了を待機することを指示できます

```
int main() {
  int i; tree_type tree;
  init_tree(tree);
  #pragma omp parallel
  #pragma omp single
  {
    #pragma omp task
      start_background_work();
    for (i = 0; i < max_steps; i++)
    {
    #pragma omp taskgroup
        {
           #pragma omp task
             compute_tree(tree);
        } // ここですべての compute_tree の完了を待機
        check_step();
    }
  } // ここで background_work が完了していることが要求される
  print_results();
  return 0;
}
```



この 2 つのタスクは同時に実行され、start_background_work() は、compute_tree() の同期の影響を受けない

Cancel 句

指定された構文タイプの最も内側の並列領域の要求をキャンセルします
if 文、while 文、do 文、switch 文とラベルの後には指定できません：

#pragma omp cancel [構文タイプ] [[,] if 節]

構文タイプ： parallel、sections、for、taskgroup

if 節： if(スカラー式)

注：構文に到達したとき、デッドロックを引き起こす可能性があるロックやその他のデータ構造を解放する必要があります。ブロックされたスレッドは取り消すことができません

Cancel 句の例

```
#define N 10000

void example() {
   std::exception *ex = NULL;
#pragma omp parallel shared(ex)
   {
#pragma omp for
      for (int i = 0; i < N; i++) {
         // no 'if' that prevents compiler optimizations
         try {
            causes_an_exception();
         }
         catch (std::exception *e) {
            // 後で例外を処理するため ex にステータスをストア
#pragma omp atomic write
            ex = e;
// for ワークシェアリングをキャンセル
#pragma omp cancel for
         }
      }
  // 例外が発生したら parallel 構文をキャンセル
      if (ex) {
#pragma omp cancel parallel
      }
      phase_1();
#pragma omp barrier
      phase_2();
   }
   // 例外がワークシェア・ループ内でスローされている場合は継続
   if (ex) {
      // ex の例外処理
   }
}
```

例外をキャッチしたら
for ワークシェア並列処理
をキャンセル

例外をキャッチしたら
parallel 並列処理を
キャンセル

Cancellation point 句

指定された構文タイプの最も内側の並列領域のキャンセルが要求された場合に、キャンセルをチェックする位置を指定：

#pragma omp cancellation point [構文タイプ]

構文タイプ： parallel、sections、for、taskgroup

制約事項：
- この構文は、実行文が許可されている場所にのみ追加できます
- if 文のアクション文として使用したり、プログラムで参照されるラベルの実行文として使用することはできません

Cancellation point 句の例

```
subroutine example(n, dim)
  integer, intent(in) :: n, dim(n)
  integer :: i, s, err
  real, allocatable :: B(:)
  err = 0
!$omp parallel shared(err)
! ...
!$omp do private(s, B)
  do i=1, n
!$omp cancellation point do
    allocate(B(dim(i)), stat=s)
    if (s .gt. 0) then
!$omp atomic write
      err = s
!$omp cancel do
    endif
!   ...
! deallocate private array B
    if (allocated(B)) then
      deallocate(B)
    endif
  enddo
!$omp end parallel
end subroutine
```

他のスレッドはこの位置でキャンセルをチェック

allocate 文からエラーが返された時に cancel do をアクティブにします

追加されたランタイム API

- omp_get_cancellation
- omp_get_proc_bind
- omp_set_default_device
- omp_get_default_device
- omp_get_num_devices
- omp_get_num_teams
- omp_get_team_num
- omp_is_initial_device

追加された環境変数

- OMP_CANCELLATION
- OMP_DEFAULT_DEVICE
- OMP_DISPLAY_ENV
- OMP_PLACES

追加された機能（4.0 -> 4.5)

- オフロード
  - 非構造化データ割り当て、非同期オフロード、MAP 節の拡張
- SIMD ベクトル
  - ループ構造への linear 節の追加
  - SIMD 構造への simdlen 節の追加
- Task 構造の依存性制御と priority 制御
- Taskloop (taskloop simd) 構文の追加*

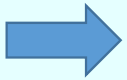
- 追加されたランタイム API
- 追加された環境変数

*インテル® コンパイラー V16 update 1 では未サポート

Taskloop 句の追加

*インテル® コンパイラー V16 update 1 では未サポート

For (C/C++) または Do (Fortran) ループ構造を、OpenMP のタスクを使用して並列化します。インテル® Cilk™ Plus の _Cilk_for (_Cilk_for _Simd) に類似しています

#pragma omp for [simd]  #pragma omp taskloop [simd]

指定可能な節：
if ([taskloop :] スカラー式)、shared(list)、private(list)、firstprivate(list)、lastprivate(list)、default(shared | none)、grainsize(grain size)、num_task(num task)、collapse(n)、final(スカラー式)、priority(priority 値)、untiled、mergable、nogroup

スケジュールは、for ワークシェアの schedule 節とは異なり、grainsize と num_task で制御

追加されたランタイム API

- omp_get_max_task_priority

追加された環境変数

- OMP_MAX_TASK_PRIORITY

付録

- インテル® C/C++ および Fortran コンパイラー V15 と V16 でサポートされる OpenMP 4.x の機能

インテル® C/C++ および Fortran コンパイラー V15 と V16

- PLACE と THREAD アフィニティー (15.0, 16.0)
- SIMD 拡張の主要機能 (15.0)
- デバイス/アクセラレーター拡張の主要機能 (15.0, 16.0)
- Simd,parallel,target,teams,distribute の複合構文 (15.0, 16.0)
- taskgroup と task depend 文 (15.0, 16.0)
- Atomic のシーケンシャル一貫性 (15.0, 16.0)
- エラー処理 (15.0, 16.0)
- ユーザー定義リダクション (16.0 C++, Fortran は今後のリリースで)
- Fortran 2003 サポート (15.0)
- ユーザー定義ベクトル異種関数 (16.0)
- __intel_simd_lane() 関数 (15.0, 16.0)
- #pragma omp ordered [simd | thread] (16.0)

sugatech