前回までの議論内容（1/2）

- lock/unlockは指示文で記述
- 例1

```
xmp_lock_t stack_lock:[*]
:
#pragma xmp lock (stack_lock:[image])
// なんらかの処理
#pragma xmp unlock (stack_lock:[image])
```

- 例1のCAFの例

```
type(lock_type), private :: stack_lock[*]
:
lock (stack_lock[image])
// なんらかの処理
unlock (stack_lock[image])
```

前回までの議論内容（2/2）

- 例2：no-wait lock

```
boolean success;
:
#pragma xmp lock (stack_lock:[image]) acquired_lock(success)
```

- 例2のCAFの例

```
logical :: success
lock(stack_lock,acquired_lock=success)
```

節にacquired_lockがありロックオブジェクトがロックされていた場合、
変数successにfalseが代入されて、そのまま進む (ロックしない)。
対象ロックオブジェクトがロックされていない場合は、
ロックして変数successにtrueが代入される。

今回の検討項目

- align指示文を使って、ロックオブジェクトを整列させる
- 場所を指定しないロック

align + ロック

- 分散配列の各要素と関連してロックしたい場合が多いと考えられるので、
  それをサポートする機能
- 例：5ノード利用時で、分散配列100要素に対して10個の分散ロックオブジェクト

align + ロック

```
#pragma xmp nodes p(5)
int a[100];
#pragma xmp template t(0:99)
#pragma xmp align a[i] with t(i)
xmp_lock_t a_lock[10];
#pragma xmp template t_lock(0:9)
#pragma xmp align a_lock[i] with t_lock(i)
   :
#pragma xmp lock (a_lock[k/10])
  me = a[k]:[i]; // 何らかの操作
#pragma xmp unlock (a_lock[k/10])
```

既存の方法を使った場合

```
#pragma xmp nodes p(5)
int a[100];
#pragma xmp template t(0:99)
#pragma xmp align a[i] with t(i)
xmp_lock_t a_lock[2]:[*];
   :
int div = 10*xmp_num_nodes();
#pragma xmp lock (a_lock[k/div]:[i])
  me = a[k]:[i]; // 何らかの操作
#pragma xmp unlock (a_lock[k/div]:[i])
```

場所を指定しないロック

- 例：

```
#pragma xmp lock (image[, tag])
  // 何らかの操作
#pragma xmp unlock (image[, tag])
```

- tagは省略した場合は、tag=0
- 利点：
  - ロックオブジェクトを宣言しなくて良い
  - 実装上、ロックのためのオブジェクトはどこにあっても良い