# XcalabeMP2.0の タスク並列機能

理研R-CCS

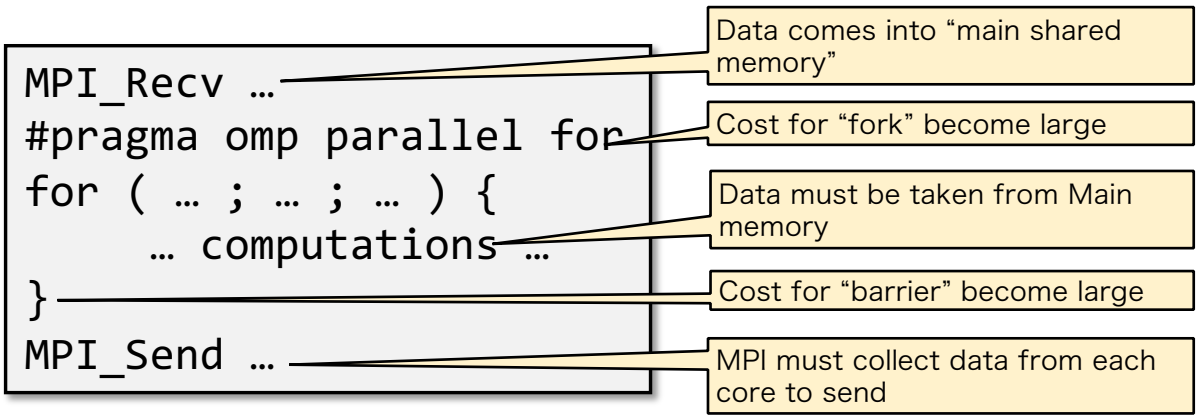
村井 均

# はじめに (1)

- XMP1.x spec. is now almost "converged."
  - focused on distributed-memory data parallelism based on the global- and local-view model
  - Upcoming ver.1.4 includes:
    - errata
    - a few extensions mainly for higher performance
    - draft of tasklets specification as an appendix

# はじめに (2)

- Rise of many-core processors
  - reveals limitations of existing programming model (XMP/MPI+OpenMP)




➡ XcalableMP2.0 for many-core

# XMP2.0の目標

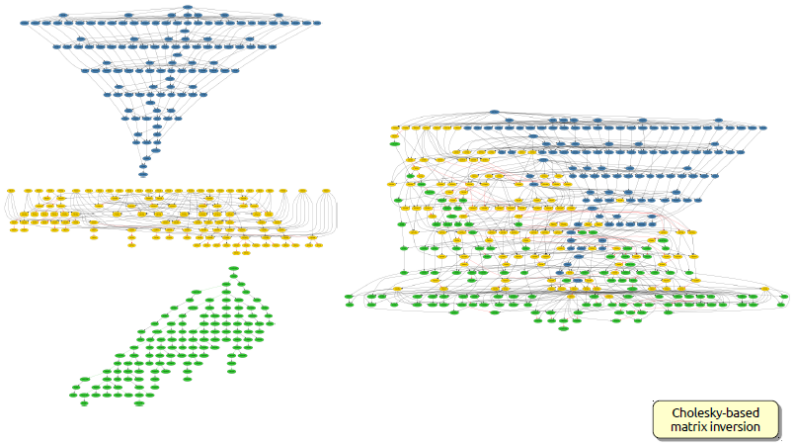
- Multitasking for many-core processors (with PGAS)
- Higher performance than MPI with explicit and aggressive optimization
- Expansion of the base language
  Fortran 2008 / C99 / C++11
- XMPT tool I/F

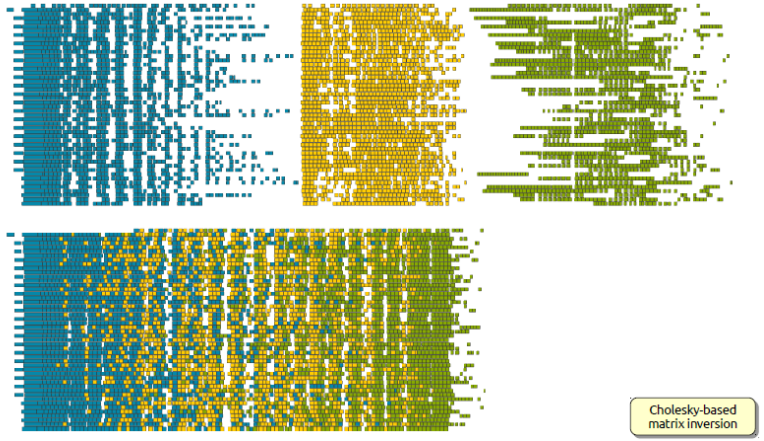
➡ Higher productivity and higher performance on many-core processors

# XMP2.0におけるマルチタスキング

- 「タスクレット」
  - created dynamically and executed in parallel on nodes.
  - can depend on and interact with each other between nodes.
- Escape from Bulk Synchronous Parallel (BSP) model
  - not barrier but P2P synchronization.
- Facilitates overlapping comm. and comp.

# 動的マルチタスキングの効果

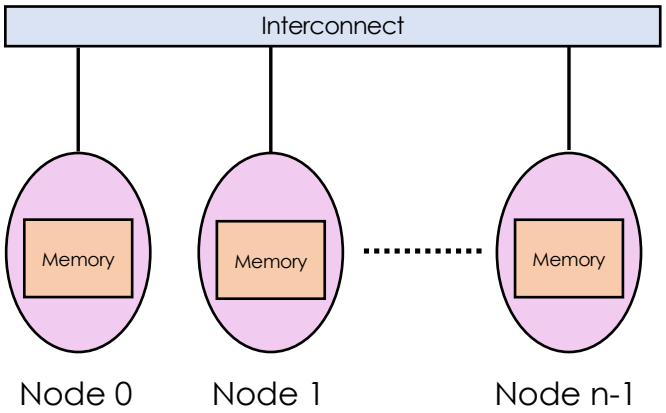



From PLASMA/QUARK slides by ICL, U. Teneessee

# XMP1.xとXMP2.0の実行モデル

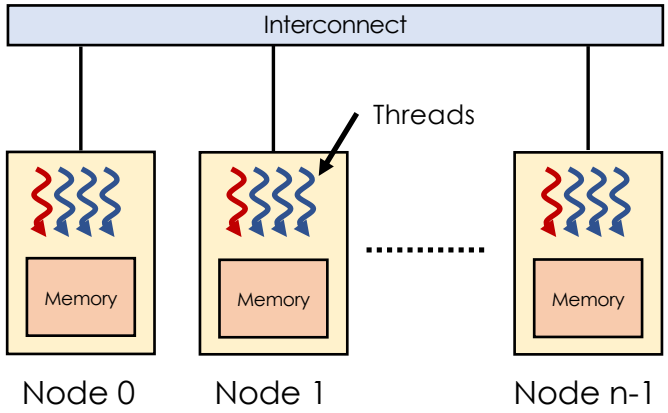
- XMP1.x




*XMP nodes* execute an XMP program in cooperation with each other.

排他的

- XMP2.0 (taskletsリージョンの中)




- An XMP program consists of tasklets.
- Threads execute tasklets on each node in cooperation with each other.

# XMP2.0におけるタスク依存関係の指定

- How to specify dependency between?
  - Within a node: `in/out/inout` clauses (cf. OpenMP's `depend` clause)
  - Between nodes:
    - Clauses for one-sided sync. (cf. `post/wait`, or `notify/query`) for the notification of ready-for-comm. and ack.
    - The interactions can be specified with any one-sided comms. (e.g. coarray, `MPI_Put/Get`, XMP's `gmove in/out`)

# Tasklet Directives

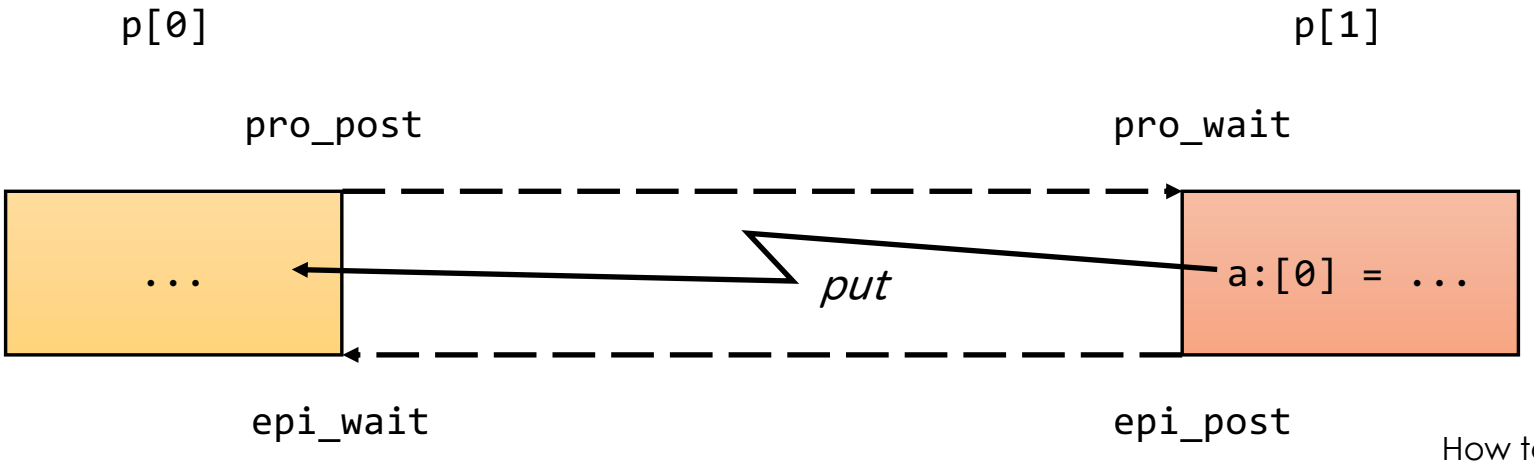
- `tasklets` Construct
  - defines a region for tasklet execution.
  - *Global constructs* cannot be put in a `tasklets` region.
- `tasklet` Construct
  - `in / out / inout`
  - `pro_post / pro_wait`
  - `epi_post / epi_wait`
  - `remote_in / remote_out`
  - `accept_remote_in / accept_remote_out`
- Communication Tasklet Constructs
  - `tasklet reflect, tasklet gmove, tasklet bcast, ...`

# Sync. between Tasklets

NOTE: `pro_wait` and `epi_wait` may influence tasklet scheduling.

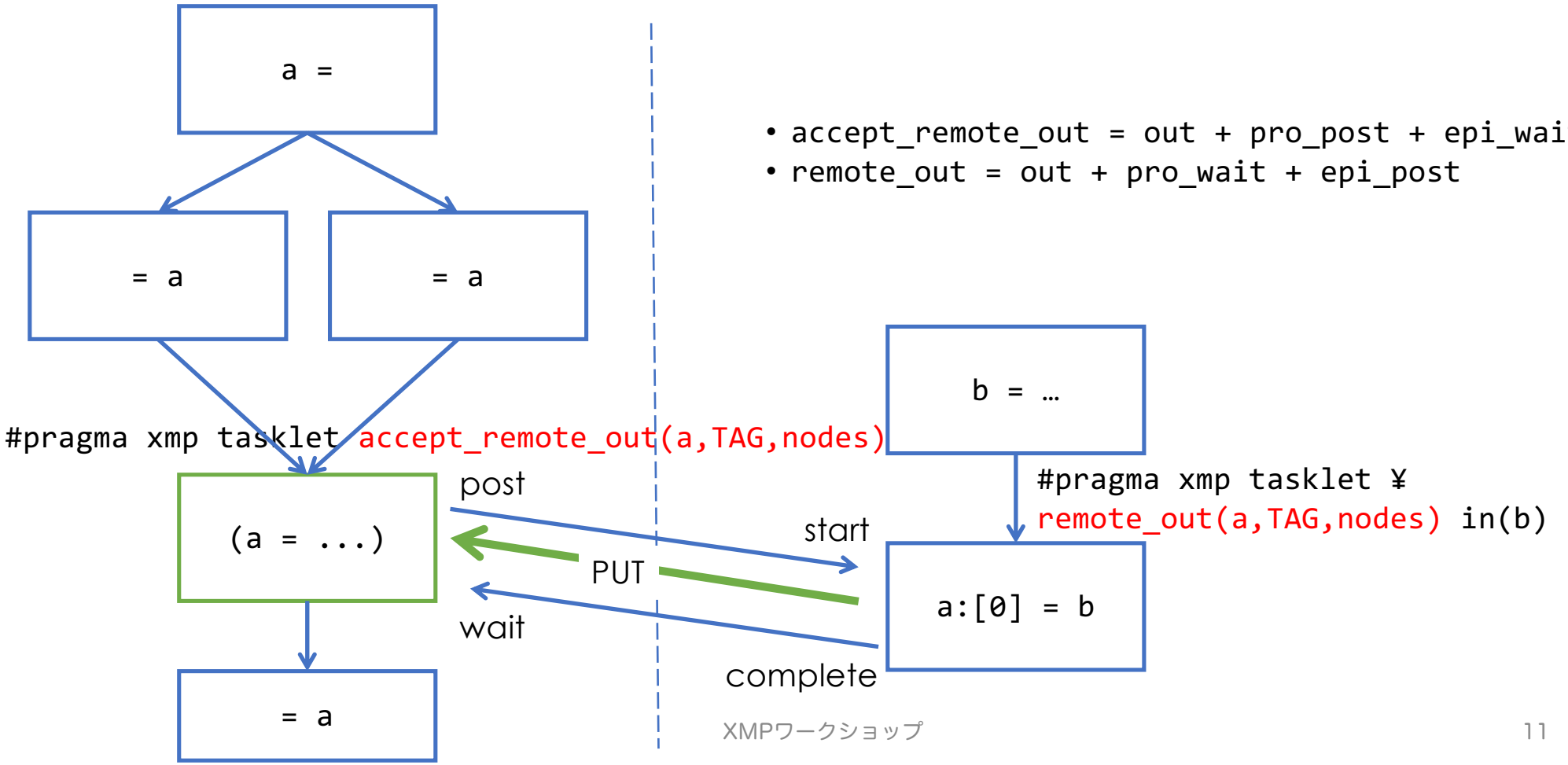
```
#pragma xmp tasklet pro_post(p(2), tag0) ¥
                    epi_wait(p(2), tag1) ¥
                    on p[0]
{
  ...
}
```

```
#pragma xmp tasklet pro_wait(p(1), tag0) ¥
                    epi_post(p(1), tag1) ¥
                    on p[1]
{
  a:[0] = ...;
}
```




How to implement TAGs is an open issue.

# Dependency between Nodes

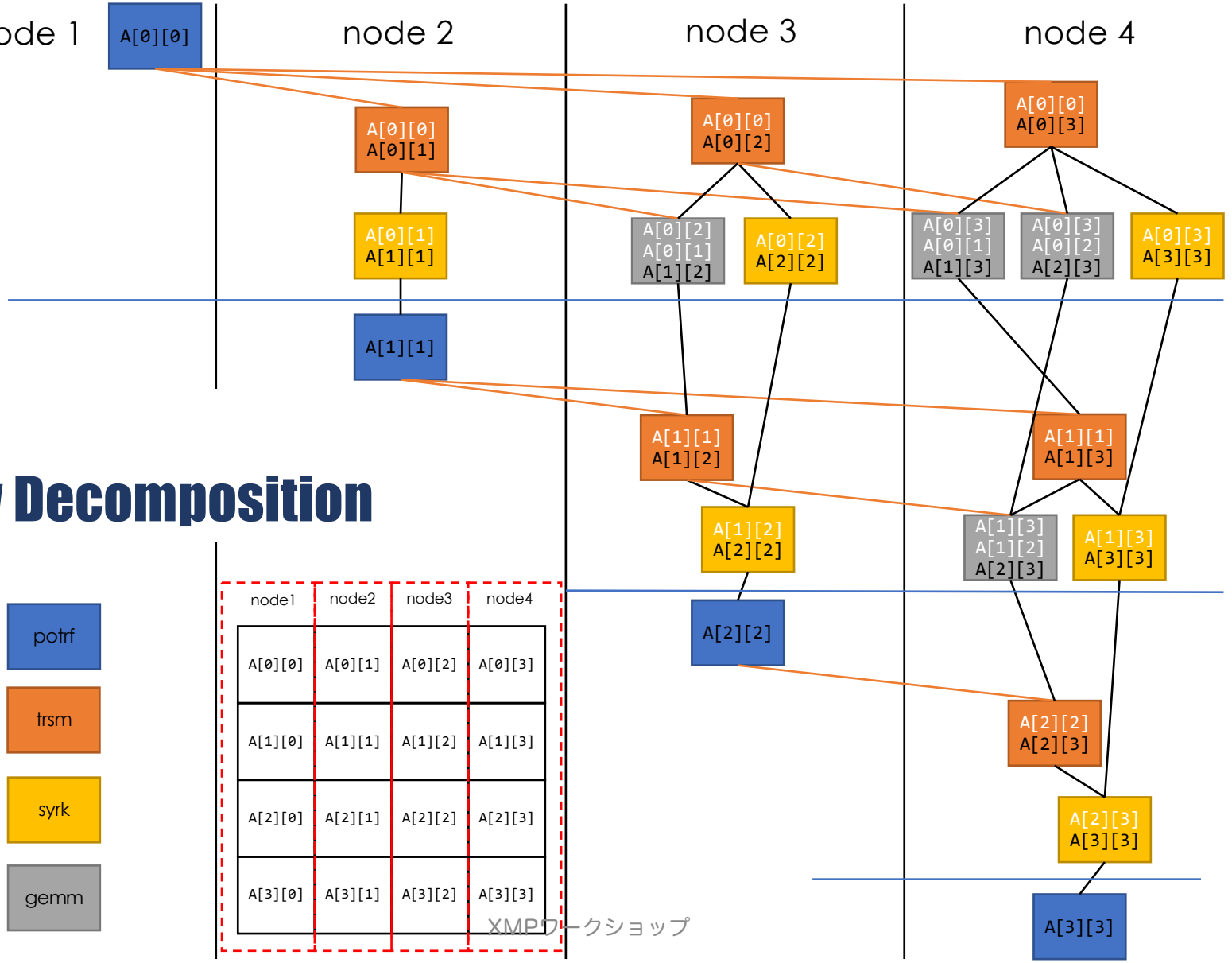


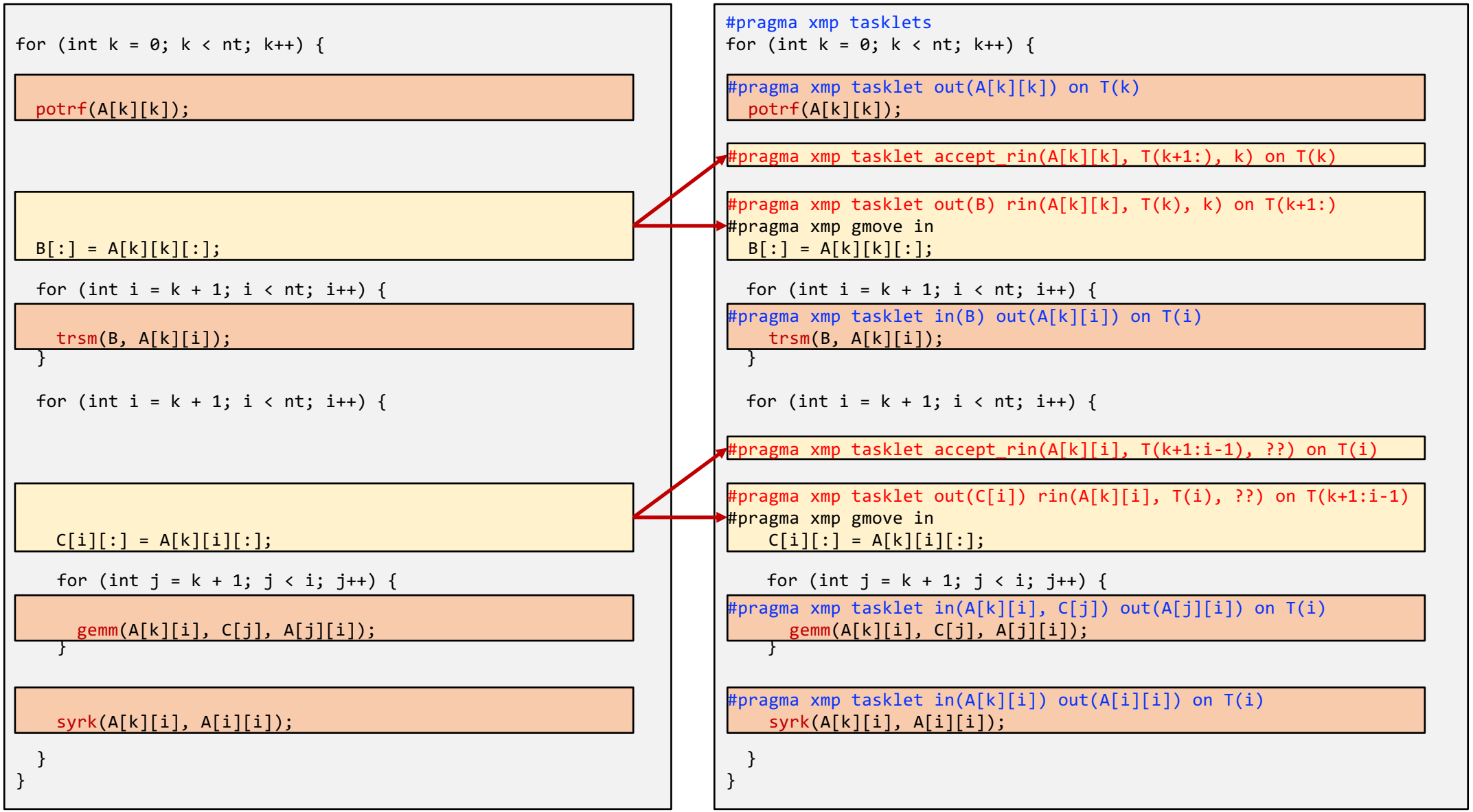

- accept_remote_out = out + pro_post + epi_wait
- remote_out = out + pro_wait + epi_post

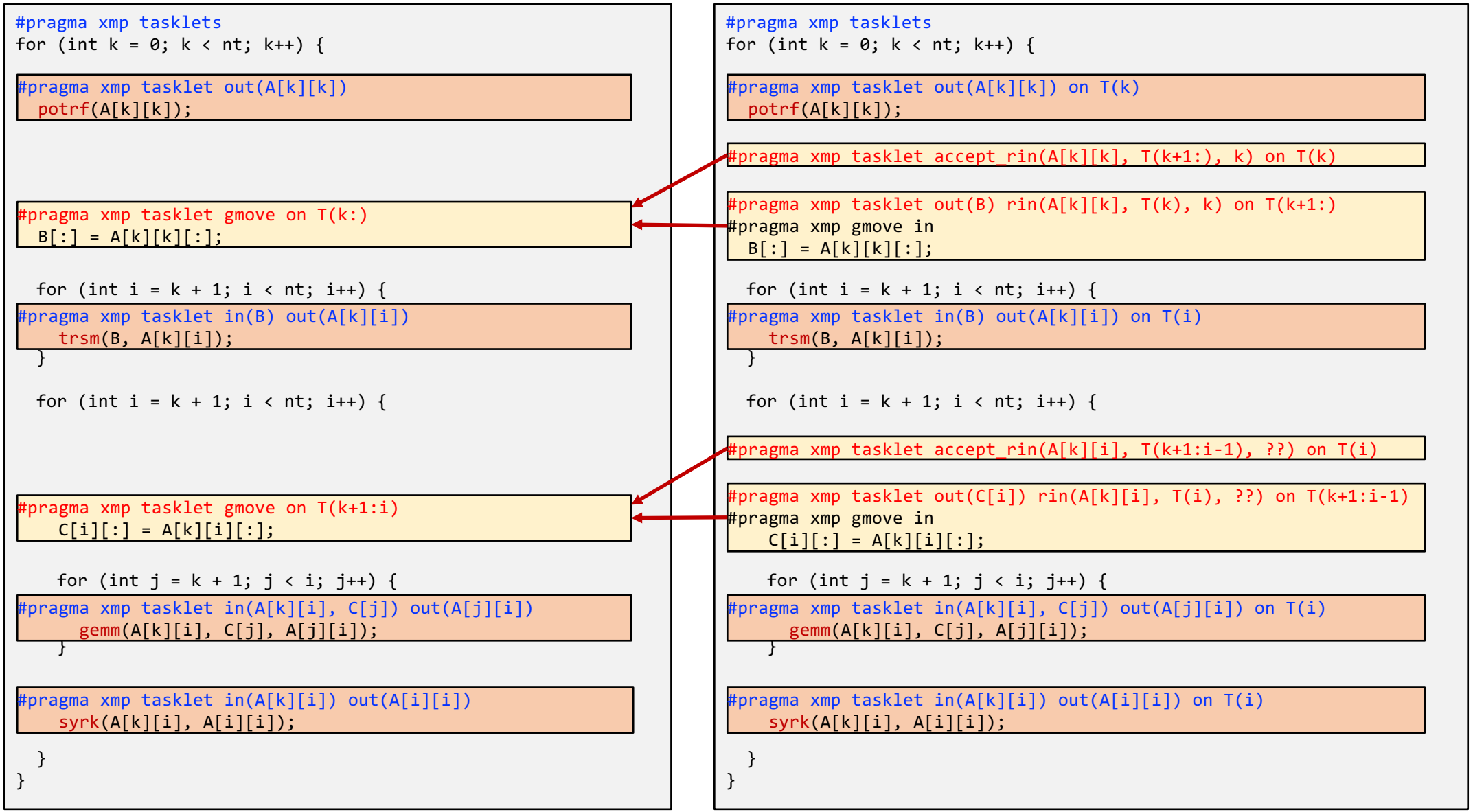
# Communication Tasklet Constructs

- Define inter-node interactions between tasklets.
- Produce the same effect as corresponding global communication directives (e.g. `reflect`).
- Non-collective (i.e. supposed to be implemented as p2p operations)
  - `tasklet reflect`
  - `tasklet gmove`
  - `tasklet bcast`
  - `...`

# Example: Cholesky Decomposition




for (int k = 0; k < nt; k++) {
potrf(A[k][k]);
B[:] = A[k][k][:];
for (int i = k + 1; i < nt; i++) {
trsm(B, A[k][i]);
}
for (int i = k + 1; i < nt; i++) {
C[i][:] = A[k][i][:];
for (int j = k + 1; j < i; j++) {
gemm(A[k][i], C[j], A[j][i]);
}
syrk(A[k][i], A[i][i]);
}
}
#pragma xmp tasklets
for (int k = 0; k < nt; k++) {
#pragma xmp tasklet out(A[k][k]) on T(k)
potrf(A[k][k]);
#pragma xmp tasklet accept_rin(A[k][k], T(k+1:), k) on T(k)
#pragma xmp tasklet out(B) rin(A[k][k], T(k), k) on T(k+1:)
#pragma xmp gmove in
B[:] = A[k][k][:];
for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet in(B) out(A[k][i]) on T(i)
trsm(B, A[k][i]);
}
for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet accept_rin(A[k][i], T(k+1:i-1), ??) on T(i)
#pragma xmp tasklet out(C[i]) rin(A[k][i], T(i), ??) on T(k+1:i-1)
#pragma xmp gmove in
C[i][:] = A[k][i][:];
for (int j = k + 1; j < i; j++) {
#pragma xmp tasklet in(A[k][i], C[j]) out(A[j][i]) on T(i)
gemm(A[k][i], C[j], A[j][i]);
}
#pragma xmp tasklet in(A[k][i]) out(A[i][i]) on T(i)
syrk(A[k][i], A[i][i]);
}
}


#pragma xmp tasklets
for (int k = 0; k < nt; k++) {
#pragma xmp tasklet out(A[k][k])
potrf(A[k][k]);
#pragma xmp tasklet gmove on T(k:)
B[:] = A[k][k][:];
for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet in(B) out(A[k][i])
trsm(B, A[k][i]);
}
for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet gmove on T(k+1:i)
C[i][:] = A[k][i][:];
for (int j = k + 1; j < i; j++) {
#pragma xmp tasklet in(A[k][i], C[j]) out(A[j][i])
gemm(A[k][i], C[j], A[j][i]);
}
#pragma xmp tasklet in(A[k][i]) out(A[i][i])
syrk(A[k][i], A[i][i]);
}
}
#pragma xmp tasklets
for (int k = 0; k < nt; k++) {
#pragma xmp tasklet out(A[k][k]) on T(k)
potrf(A[k][k]);
#pragma xmp tasklet accept_rin(A[k][k], T(k+1:), k) on T(k)
#pragma xmp tasklet out(B) rin(A[k][k], T(k), k) on T(k+1:)
#pragma xmp gmove in
B[:] = A[k][k][:];
for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet in(B) out(A[k][i]) on T(i)
trsm(B, A[k][i]);
}
for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet accept_rin(A[k][i], T(k+1:i-1), ??) on T(i)
#pragma xmp tasklet out(C[i]) rin(A[k][i], T(i), ??) on T(k+1:i-1)
#pragma xmp gmove in
C[i][:] = A[k][i][:];
for (int j = k + 1; j < i; j++) {
#pragma xmp tasklet in(A[k][i], C[j]) out(A[j][i]) on T(i)
gemm(A[k][i], C[j], A[j][i]);
}
#pragma xmp tasklet in(A[k][i]) out(A[i][i]) on T(i)
syrk(A[k][i], A[i][i]);
}
}

# おわりに

- XMP2.0は、メニーコアに向けた動的マルチタスキングの機能をサポートする。
- `tasklet`構文
  - ノード内の依存関係 → OpenMPのタスク機能に基づく
  - ノード間の依存関係 → p2pの同期に基づく
- タスクレット間のインタラクションは、任意の片側通信に依る。
- 通信タスクレット構文は、XMP1.xのグローバル通信に基づき、タスクレット間のインタラクションを指定する。

# 今後の課題

- グローバルビュー/PGASとマルチタスキングの融合
  - taskletloop構文 (cf. OpenMPのtaskloop)

```
!$xmp taskletloop on t(i) grainsize(5)
do i = 1, n
  a(i) = ...
end do
```

テンプレートtの分散に従って繰り返し空間が分散された後、各ノード上で5回ずつの繰り返しを一つのタスクレットとして生成する。