XcalableMP 2.0 案の動的タスク並列機能

津金 佳祐[†1]，中尾 昌広[†2]，李 珍泌[†2]，村井 均[†2]，佐藤 三久[†1,2]

†1 筑波大学大学院 システム情報工学研究科
†2 国立研究開発法人 理化学研究所 計算科学研究機構

目次

- 研究背景・目的
- プログラミングモデル
  - XcalableMP
  - OpenMP
- 動的タスク機能
  - tasklet in/out 指示文
  - tasklet post/wait 指示文
- 評価
  - ブロックコレスキー分解
  - 性能・生産性
- まとめ・今後の課題

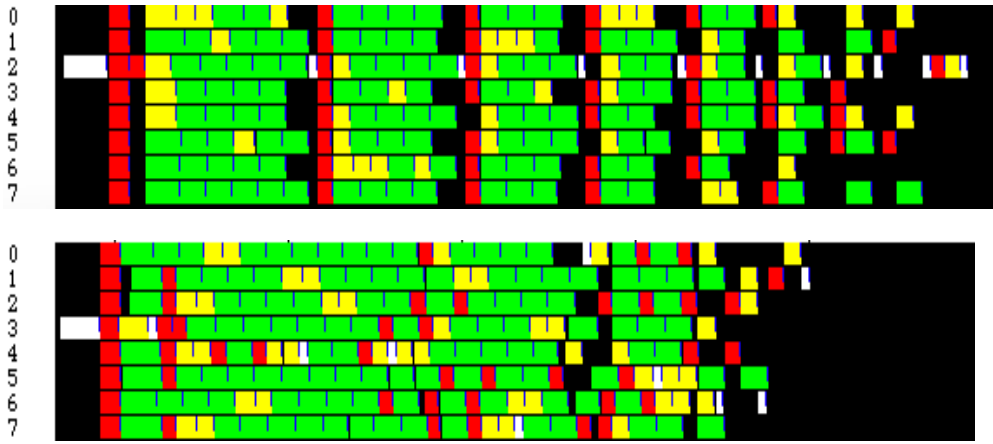
目次

- 研究背景・目的
- プログラミングモデル
  - XcalableMP
  - OpenMP
- 動的タスク機能
  - tasklet in/out 指示文
  - tasklet post/wait 指示文
- 評価
  - ブロックコレスキー分解
  - 性能・生産性
- まとめ・今後の課題

研究背景

- メニーコアプロセッサを用いた大規模システムの登場
  - Intel Xeon Phi
    - Tianhe-2 (#2), Stampede (#12), COMA (#133) : 2016/6, Top500
    - Oakforest-PACS (筑波大/東京大)
- メニーコアプロセッサにおけるプログラミング
  - 大量のコアをどのように効率よく使用するか
    - 通信と演算のオーバラップ
    - コア単位での細粒度な同期

➢ タスク並列に注目



全体同期とタスク依存の同期によるコレスキー分解の実行例

目的

- XcalableMP における動的タスク並列機能を提案
- Omni XcalableMP Compiler のランタイムを実装
  - 軽量スレッドライブラリ Argobots + MPI で実装（省略）
- 性能・生産性の評価
  - ブロックコレスキー分解

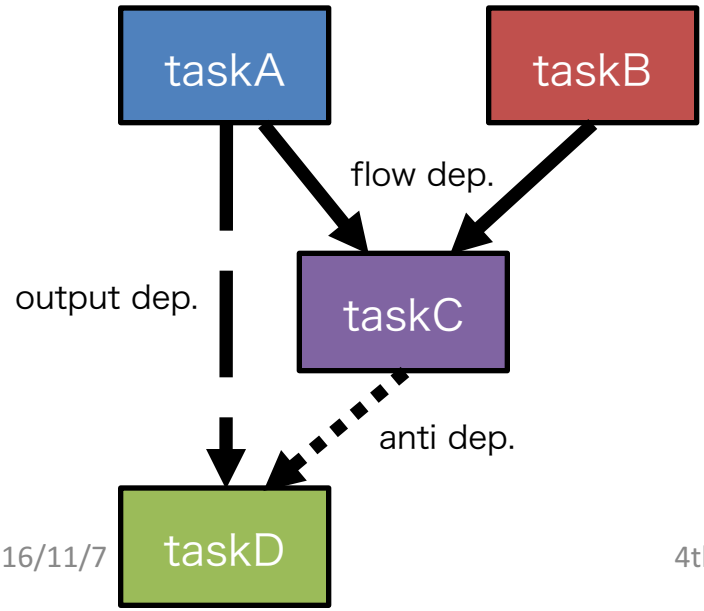
- XMP における動的タスク機能は XMP 規格部会にて現在検討中
- 本研究はその一つの提案

目次

- 研究背景・目的
- プログラミングモデル
  - ~~XcalableMP~~
  - OpenMP
- 動的タスク機能
  - tasklet in/out 指示文
  - tasklet post/wait 指示文
- 評価
  - ブロックコレスキー分解
  - 性能・生産性
- まとめ・今後の課題

OpenMP task 指示文

- 仕様 3.0 から登場したタスク並列処理のための機能
  - 演算が動的に決定する場合に用いられる
- 仕様 4.0 からはタスク間の依存関係を記述可能
  - depend 節 の in / out にデータ依存を記述
  - タスクの生成順序を考慮

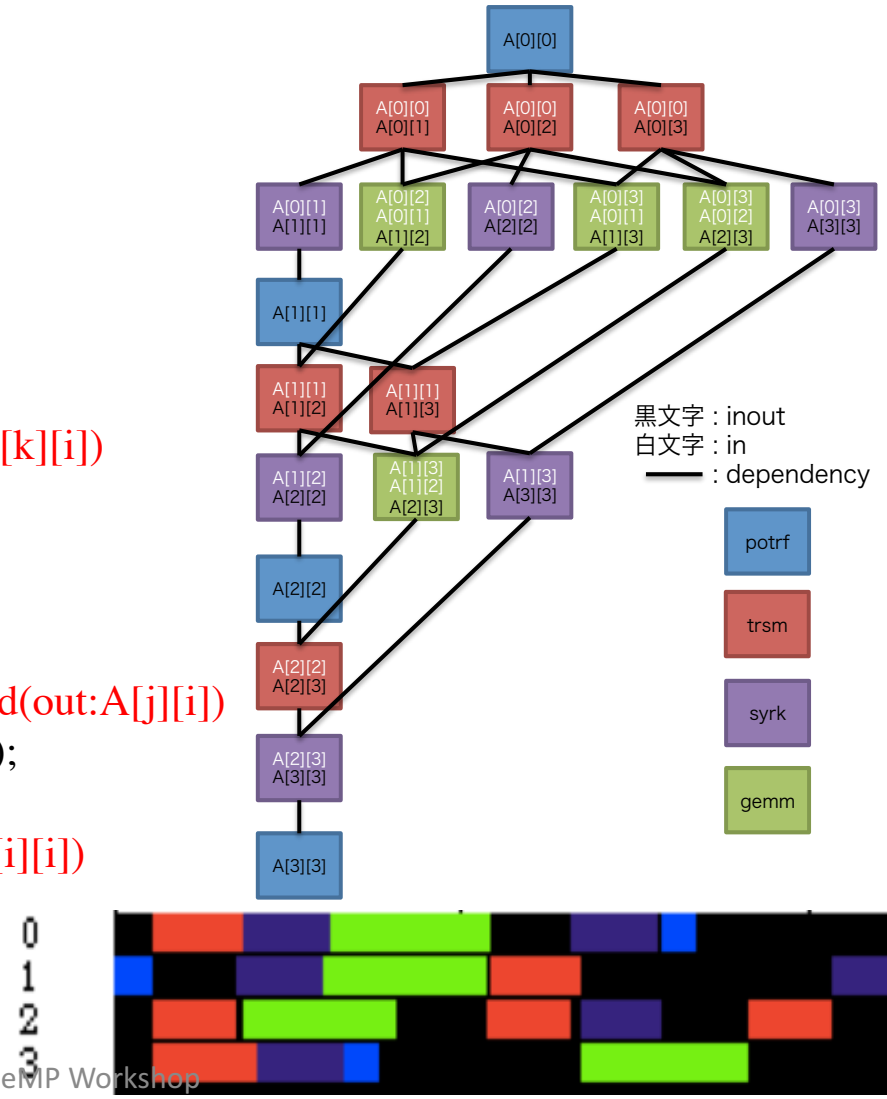



```
#pragma omp parallel
#pragma omp single
{
   int A, B, C;
#pragma omp task depend(out:A)
   A = 1;                    /* taskA */
#pragma omp task depend(out:B)
   B = 2;                    /* taskB */
#pragma omp task depend(in:A, B) depend(out:C)
   C = A + B;                /* taskC */
#pragma omp task depend(out:A)
   A = 2;                    /* taskD */
#pragma omp taskwait
}
```

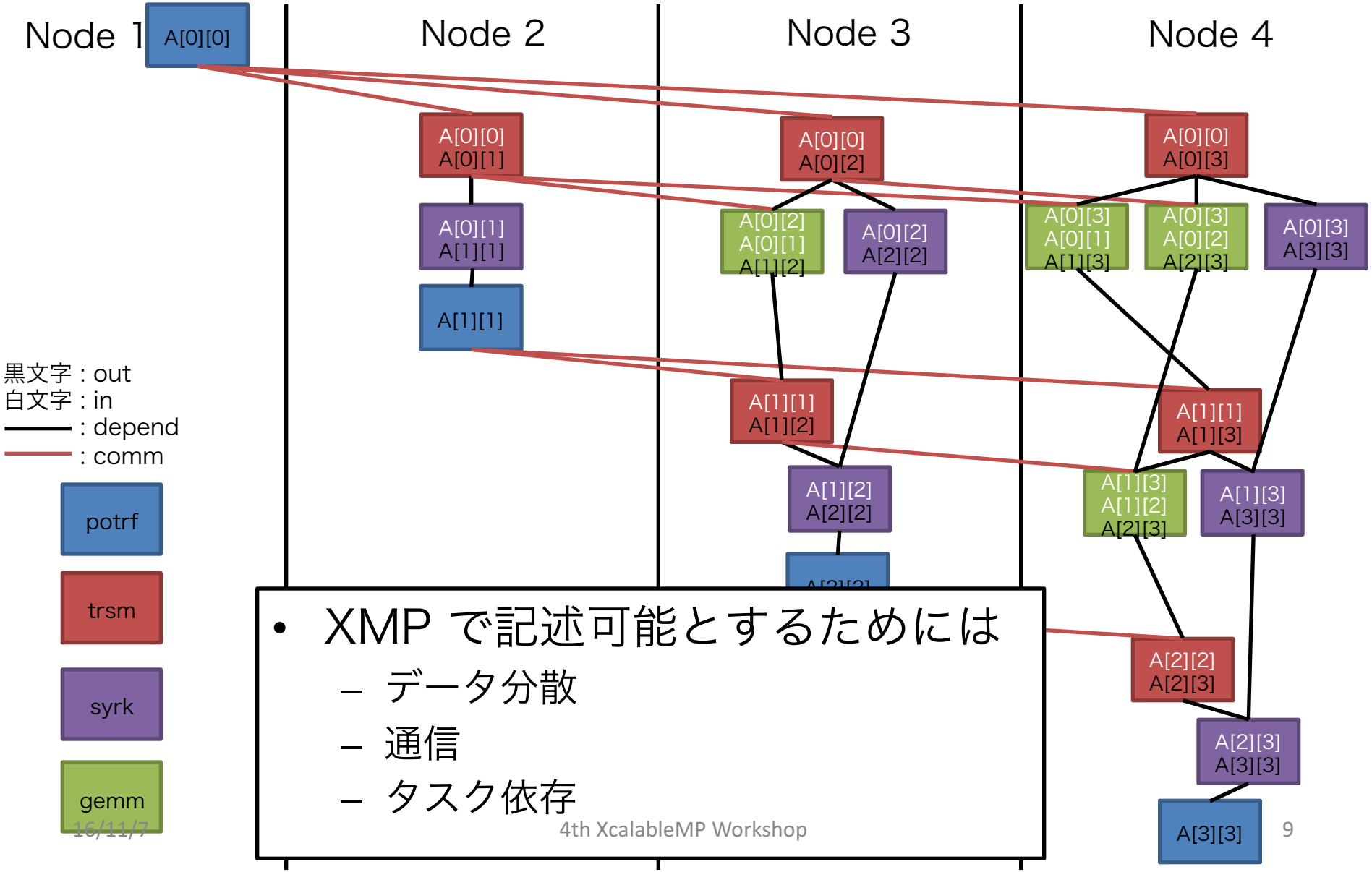
OpenMP task 指示文による ブロックコレスキー分解

```
#pragma omp parallel
#pragma omp single
{
   for (int k = 0; k < nt; k++) {
#pragma omp task depend(out:A[k][k])
        omp_potrf (A[k][k], ts, ts);

     for (int i = k + 1; i < nt; i++) {
#pragma omp task depend(in:A[k][k]) depend(out:A[k][i])
          omp_trsm (A[k][k], A[k][i], ts, ts);
     }
     for (int i = k + 1; i < nt; i++) {
       for (int j = k + 1; j < i; j++) {
#pragma omp task depend(in:A[k][i], A[k][j]) depend(out:A[j][i])
            omp_gemm (A[k][i], A[k][j], A[j][i], ts, ts);
       }
#pragma omp task depend(in:A[k][i]) depend(out:A[i][i])
          omp_syrk (A[k][i], A[i][i], ts, ts);
     }
  }
#pragma omp taskwait
}
```



分散メモリ環境上のブロックコレスキー分解



- XMP で記述可能とするためには
  - データ分散
  - 通信
  - タスク依存

目次

- 研究背景・目的
- プログラミングモデル
  - XcalableMP
  - OpenMP
- 動的タスク機能
  - tasklet in/out 指示文
  - tasklet post/wait 指示文
- 評価
  - ブロックコレスキー分解
  - 性能・生産性
- まとめ・今後の課題

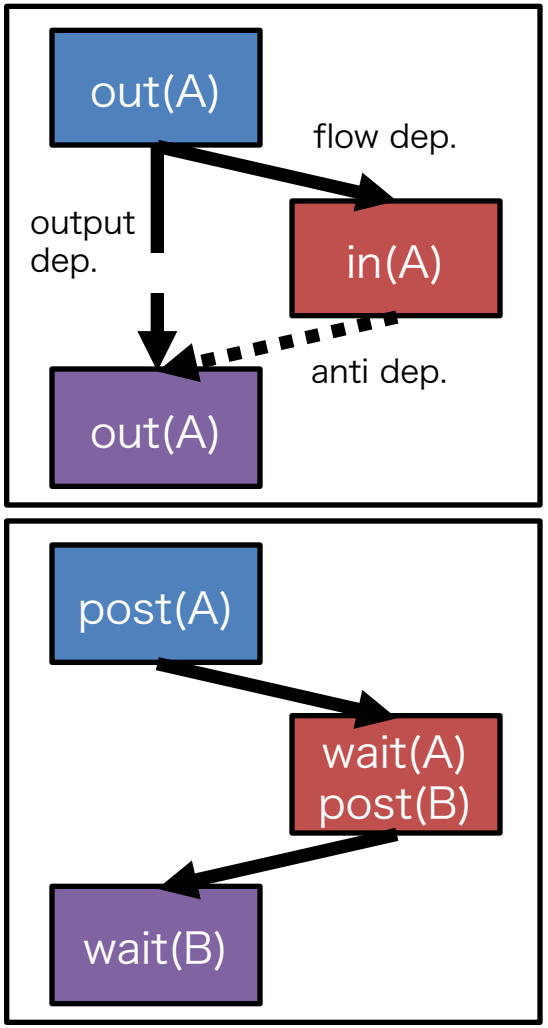
tasklet 指示文の記述案

#pragma xmp tasklet *tasklet-format[, ...] [*on {*node-ref* | *template-ref*}*]*
(structured-block)

- データ分散
  - グローバルビューモデルのデータ分散記述を使用
  - 実行ノードを on 節で記述
- 通信
  - ユーザが記述
  - gmove in/out 指示文や coarray
- タスク依存
  - OpenMP task depend のように指示文の節として記述

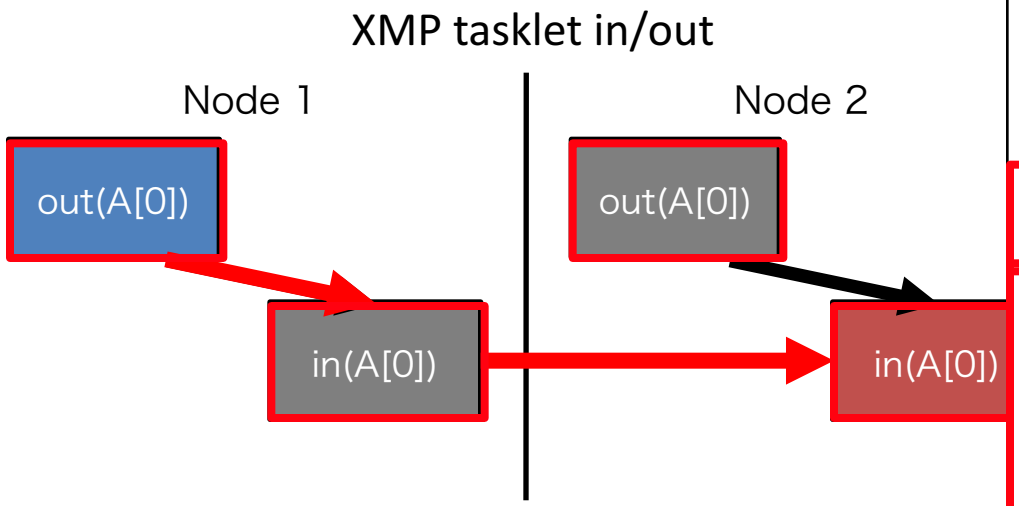
tasklet 指示文の記述案

- 2 種類の記述方法（節）を提案
- in / out 節
  - OpenMP task depend 節に則った記述案
  - データ依存 (フロー，出力，反依存)
  - 既存のコードを再利用が可能
- post / wait 節
  - データフローを post -> wait で記述
  - 直接的な記述



tasklet in/out 指示文

- OpenMP task depend と同様にデータ依存を記述
  - ノード内 / 間の依存は分散配列によりランタイムが判断
  - 全てのノードが全ての tasklet 指示文に到達する必要がある

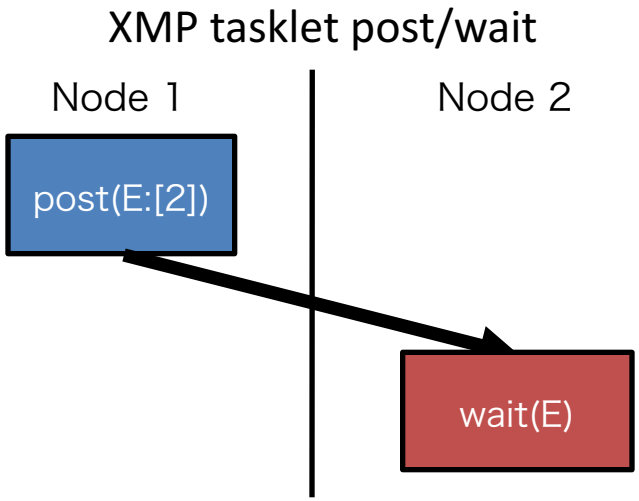



```
int A[2], B;
#pragma xmp nodes P(2)
#pragma xmp template T(0:1)
#pragma xmp distribute T(block) onto P
#pragma xmp align A[i] with T(i)

#pragma xmp tasklet out(A[0]) on T(0)
    funcA(A[0]);
#pragma xmp tasklet in(A[0]) out(A[1]) on T(1)
{
#pragma xmp gmove in
    B = A[0];
    funcB(B, A[1]);
}
#pragma xmp taskletwait
```

tasklet post/wait 指示文

- イベント変数を用いてデータフローを post -> wait で記述
- ノード間の依存関係を直接ユーザが指定
  - coarray のような記法で post 節のみ指定可能
  - 例 : post (E:[2])




```
int A[2], B;
xmp_event_t E:[*];
#pragma xmp nodes P(2)
#pragma xmp template T(0:1)
#pragma xmp distribute T(block) onto P
#pragma xmp align A[i] with T(i)

#pragma xmp tasklet post(E:[2]) on T(0)
    funcA(A[0]);
#pragma xmp tasklet wait(E) on T(1)
{
#pragma xmp gmove in
    B = A[0];
    funcB(B, A[1]);
}
#pragma xmp taskletwait
```

目次

- 研究背景・目的
- プログラミングモデル
  - XcalableMP
  - OpenMP
- 動的タスク機能
  - tasklet in/out 指示文
  - tasklet post/wait 指示文
- 評価
  - ブロックコレスキー分解
  - 性能・生産性
- まとめ・今後の課題

ブロックコレスキー分解 (XMP tasklet in/out)

```
double A[nt][nt][ts*ts];

for (int k = 0; k < nt; k++) {

    potrf (A[k][k]);

    for (int i = k + 1; i < nt; i++) {

        trsm (A[k][k], A[k][i]);
    }
/* … */
}
```

ブロックコレスキー分解 (XMP tasklet in/out)

```
double A[nt][nt][ts*ts];

#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

for (int k = 0; k < nt; k++) {

    potrf (A[k][k]);



    for (int i = k + 1; i < nt; i++) {

        trsm (A[k][k], A[k][i]);
    }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

ブロックコレスキー分解 (XMP tasklet in/out)

```
double A[nt][nt][ts*ts];

#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

for (int k = 0; k < nt; k++) {
#pragma xmp tasklet on T(k)
    potrf (A[k][k]);

    for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet on T(i)
        trsm (A[k][k], A[k][i]);
    }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

ブロックコレスキー分解 (XMP tasklet in/out)

```
double A[nt][nt][ts*ts], B[ts*ts];

#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

for (int k = 0; k < nt; k++) {
#pragma xmp tasklet on T(k)
    potrf (A[k][k]);

#pragma xmp tasklet on T(k:)
#pragma xmp gmove in
    B[:] = A[k][k][:];

    for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet on T(i)
        trsm (B, A[k][i]);
    }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

通信を記述

ブロックコレスキー分解 (XMP tasklet in/out)

```
double A[nt][nt][ts*ts], B[ts*ts];

#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

for (int k = 0; k < nt; k++) {
#pragma xmp tasklet out(A[k][k]) on T(k)
    potrf (A[k][k]);

#pragma xmp tasklet in(A[k][k]) out(B) on T(k:)
#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]);
    }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

通信を記述

依存関係を記述

ブロックコレスキー分解 (XMP tasklet in/out)

```
double A[nt][nt][ts*ts], B[ts*ts];

#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

for (int k = 0; k < nt; k++) {
#pragma xmp tasklet out(A[k][k]) on T(k)
    potrf (A[k][k]);

#pragma xmp tasklet in(A[k][k]) out(B) on T(k:)
#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]);
    }
/* … */
}
#pragma xmp taskletwait
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

通信を記述

依存関係を記述

同期

ブロックコレスキー分解 (XMP tasklet post/wait)

```
double A[nt][nt][ts*ts], B[ts*ts];


#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)


for (int k = 0; k < nt; k++) {
#pragma xmp tasklet on T(k)
   potrf (A[k][k]);

#pragma xmp tasklet on T(k:)
#pragma xmp gmove in
   B[:] = A[k][k][:];

   for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet on T(i)
      trsm (B, A[k][i]);
   }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

通信を記述

ブロックコレスキー分解 (XMP tasklet post/wait)

```
double A[nt][nt][ts*ts], B[ts*ts];

xmp_event_t EA[nt], EB[nt], EC[nt][nt];
#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)


for (int k = 0; k < nt; k++) {
#pragma xmp tasklet post(EA[k]) on T(k)
   potrf (A[k][k]);

#pragma xmp tasklet post(EA[k]) wait(EB[k]) on T(k:)
#pragma xmp gmove in
   B[:] = A[k][k][:];

   for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet post(EB[k]) wait(EC[k][i]) on T(i)
      trsm (B, A[k][i]);
   }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

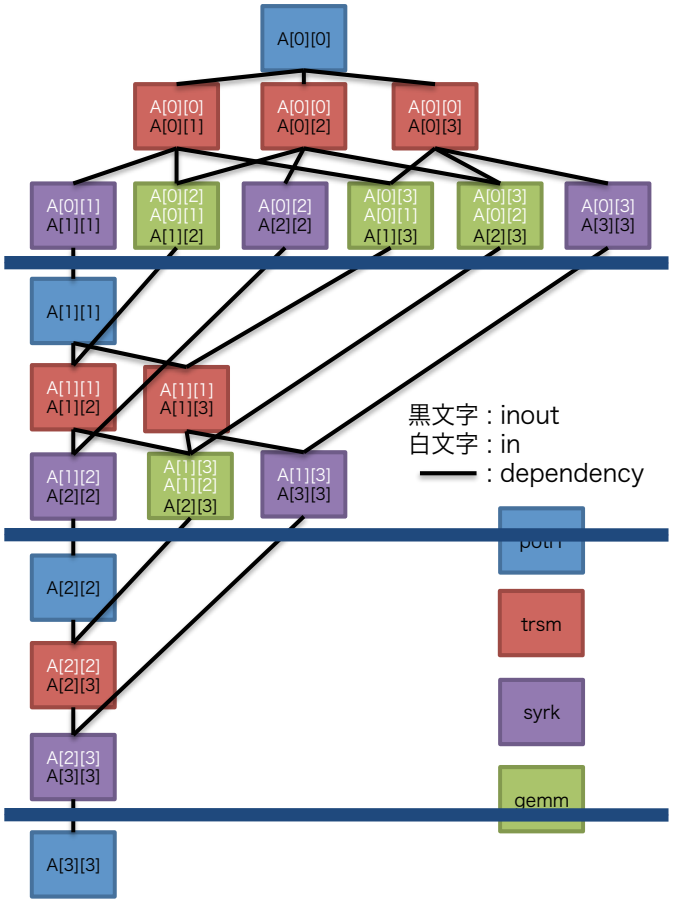
通信を記述

イベントを記述

OpenMP task 指示文による ブロックコレスキー分解 (再掲)

```
#pragma omp parallel
#pragma omp single
{
   for (int k = 0; k < nt; k++) {
#pragma omp task depend(out:A[k][k])
        omp_potrf (A[k][k], ts, ts);

     for (int i = k + 1; i < nt; i++) {
#pragma omp task depend(in:A[k][k]) depend(out:A[k][i])
          omp_trsm (A[k][k], A[k][i], ts, ts);
     }
     for (int i = k + 1; i < nt; i++) {
       for (int j = k + 1; j < i; j++) {
#pragma omp task depend(in:A[k][i], A[k][j]) depend(out:A[j][i])
            omp_gemm (A[k][i], A[k][j], A[j][i], ts, ts);
       }
#pragma omp task depend(in:A[k][i]) depend(out:A[i][i])
          omp_syrk (A[k][i], A[i][i], ts, ts);
     }
  }
#pragma omp taskwait
}
```



ブロックコレスキー分解 (XMP tasklet post/wait)

```
double A[nt][nt][ts*ts], B[ts*ts];

xmp_event_t EA[nt], EB[nt], EC[nt][nt], ED[nt+1];
#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

xmp_enable_event(ED[0]);
for (int k = 0; k < nt; k++) {
#pragma xmp tasklet wait(ED[k]) post(EA[k]) on T(k)
   potrf (A[k][k]);

#pragma xmp tasklet post(EA[k]) wait(EB[k]) on T(k:)
#pragma xmp gmove in
   B[:] = A[k][k][:];

   for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet post(EB[k]) wait(EC[k][i]) on T(i)
      trsm (B, A[k][i]);
   }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

通信を記述

イベントを記述

wait を無視したい場合

ブロックコレスキー分解 (XMP tasklet post/wait)

```
double A[nt][nt][ts*ts], B[ts*ts];

xmp_event_t EA[nt]:[*], EB[nt], EC[nt][nt]:[*], ED[nt+1];
#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

xmp_enable_event(ED[0]);
for (int k = 0; k < nt; k++) {
#pragma xmp tasklet wait(ED[k]) post(EA[k]:[:]) on T(k)
   potrf (A[k][k]);

#pragma xmp tasklet post(EA[k]) wait(EB[k]) on T(k:)
#pragma xmp gmove in
   B[:] = A[k][k][:];

   for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet post(EB[k]) wait(EC[k][i]:[:]) on T(i)
      trsm (B, A[k][i]);
   }
/* … */
}
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

通信を記述

イベントを記述

wait を無視したい場合

ノード間の依存関係

ブロックコレスキー分解 (XMP tasklet post/wait)

```
double A[nt][nt][ts*ts], B[ts*ts];

xmp_event_t EA[nt]:[*], EB[nt], EC[nt][nt]:[*], ED[nt+1];
#pragma xmp nodes P(*)
#pragma xmp template T(0:nt-1)
#pragma xmp distribute T(cyclic) onto P
#pragma xmp align A[*][i][*] with T(i)

xmp_enable_event(ED[0]);
for (int k = 0; k < nt; k++) {
#pragma xmp tasklet wait(ED[k]) post(EA[k]:[:]) on T(k)
   potrf (A[k][k]);

#pragma xmp tasklet post(EA[k]) wait(EB[k]) on T(k:)
#pragma xmp gmove in
   B[:] = A[k][k][:];

   for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet post(EB[k]) wait(EC[k][i]:[:]) on T(i)
      trsm (B, A[k][i]);
   }
/* … */
}
#pragma xmp taskletwait
```

グローバルビューの指示文でデータ分割を指定

演算をタスク化

通信を記述

イベントを記述

wait を無視したい場合

ノード間の依存関係

同期

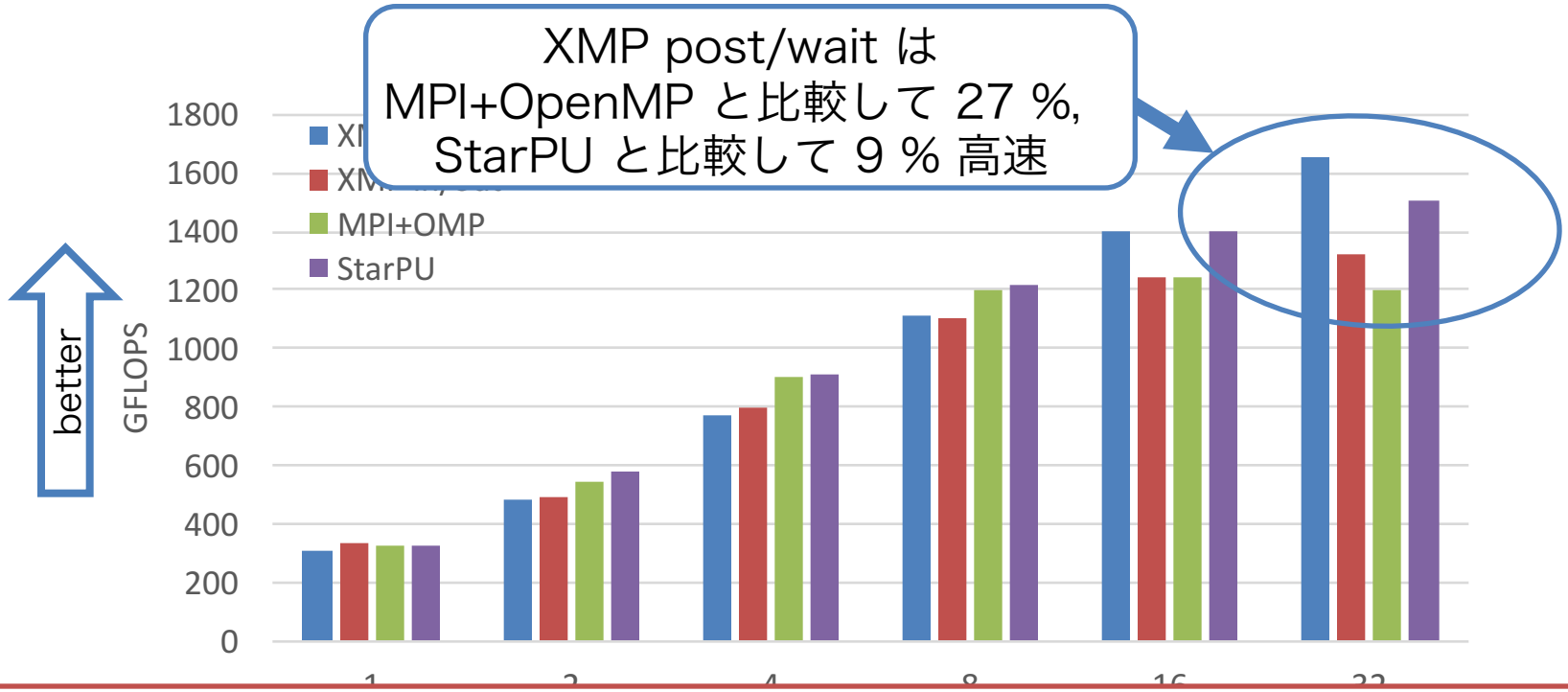
実験環境

- ベンチマーク：ブロックコレスキーコード
- 比較
  - XMP tasklet in/out
  - XMP tasklet post/wait
  - MPI + OpenMP
  - StarPU
    - INRIA が開発を進めている分散メモリ環境におけるタスク並列処理を記述可能なランタイムライブラリ
- 環境
  - HA-PACS/TCA 部（筑波大学計算科学研究センター）

CPU	Intel Xeon-E5 2680v2 2.8GHz × 2 (20 cores)
Memory	DDR3 1866MHz × 4 channel, 128GB
Interconnect	InfiniBand: Mellanox Connect-X3 Dual-port QDR
Software	Intel Compiler 16.0.2, Intel MPI 5.1.3, Intel MKL 11.3.2, StarPU 1.2

性能評価

- 行列サイズ：16k x 16k，ブロックサイズ：512 x 512
- 16 スレッド/プロセス，1 プロセス/ノード



XMP in/out は依存先の tasklet 指示文に到達した時にノード間の依存があると判断するため，直接通信相手を指定する XMP post/wait と比較して性能が低い

生産性

- 定量的な評価
  - delta SLOC : 逐次実装からの変更（修正，追加，削除）行数の比較

	Serial	OMP	MPI+OMP	StarPU	XMP in/out	XMP post/wait
modified	-	0	13	31	2	2
added	-	12	264	176	16	26
deleted	-	0	0	0	0	0
SLOC	318	330	582	494	334	344

  - XMP post/wait はイベント変数を定義するため XMP in/out と比較して行数が増加
- 定性的な評価
  - データ分散を指示文で簡易に実装
  - 通信記述以外は，ほぼ OpenMP と同等のタスク依存の記述

目次

- 研究背景・目的
- プログラミングモデル
  - XcalableMP
  - OpenMP
- 動的タスク機能
  - tasklet in/out 指示文
  - tasklet post/wait 指示文
- 評価
  - ブロックコレスキー分解
  - 性能・生産性
- まとめ・今後の課題

まとめ

- XcalableMP における動的タスク並列機能を提案
  - tasklet in/out，post/wait 指示文を提案
- Omni XMP Compiler のランタイムを実装
- 性能・生産性を評価
  - XMP post/wait は MPI+OpenMP と比較して 27 %，StarPU は 9 % 高速
  - 指示文ベースの記述により大幅な行数削減
    - 通信やデータ分散を除けばほぼ OpenMP と同等な記述

今後の課題

- tasklet 指示文の仕様の決定
- Omni XMP Compiler のコード変換部の実装
- 他のアプリケーションでの評価
- メニーコア上での評価