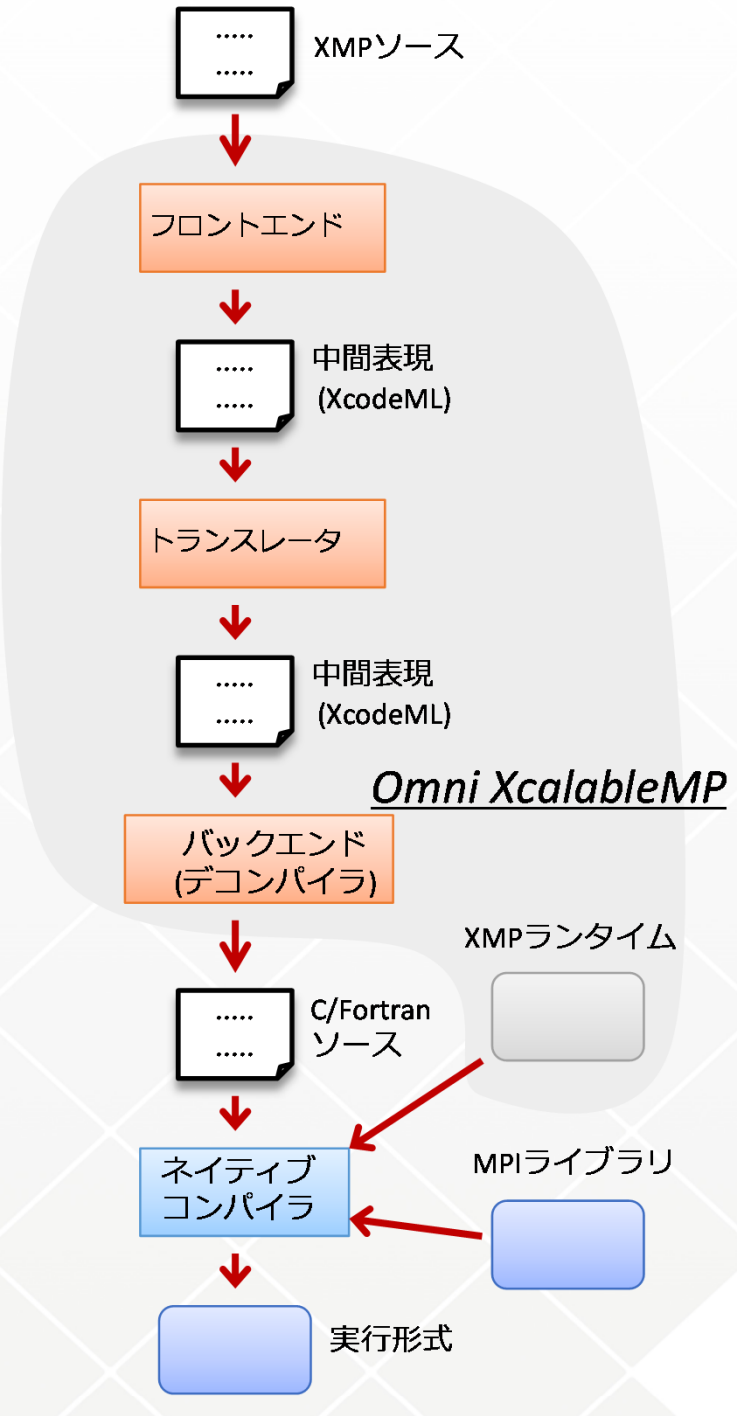
Omni コンパイラの概要

Omni XcalableMP

- **理研AICSと筑波大で開発中のXMP処理系**
  - XMP/C
  - XMP/Fortran
- **オープンソース**
- **トランスレータ + ランタイム(MPIベース)**
- **OpenACC、XcalableACC対応**
  - OpenMP 4　を計画
- **Front-end**
  - Fortran 2008：　今年度、ほぼ終了
  - C++：　LLVM　clangベースで開発中

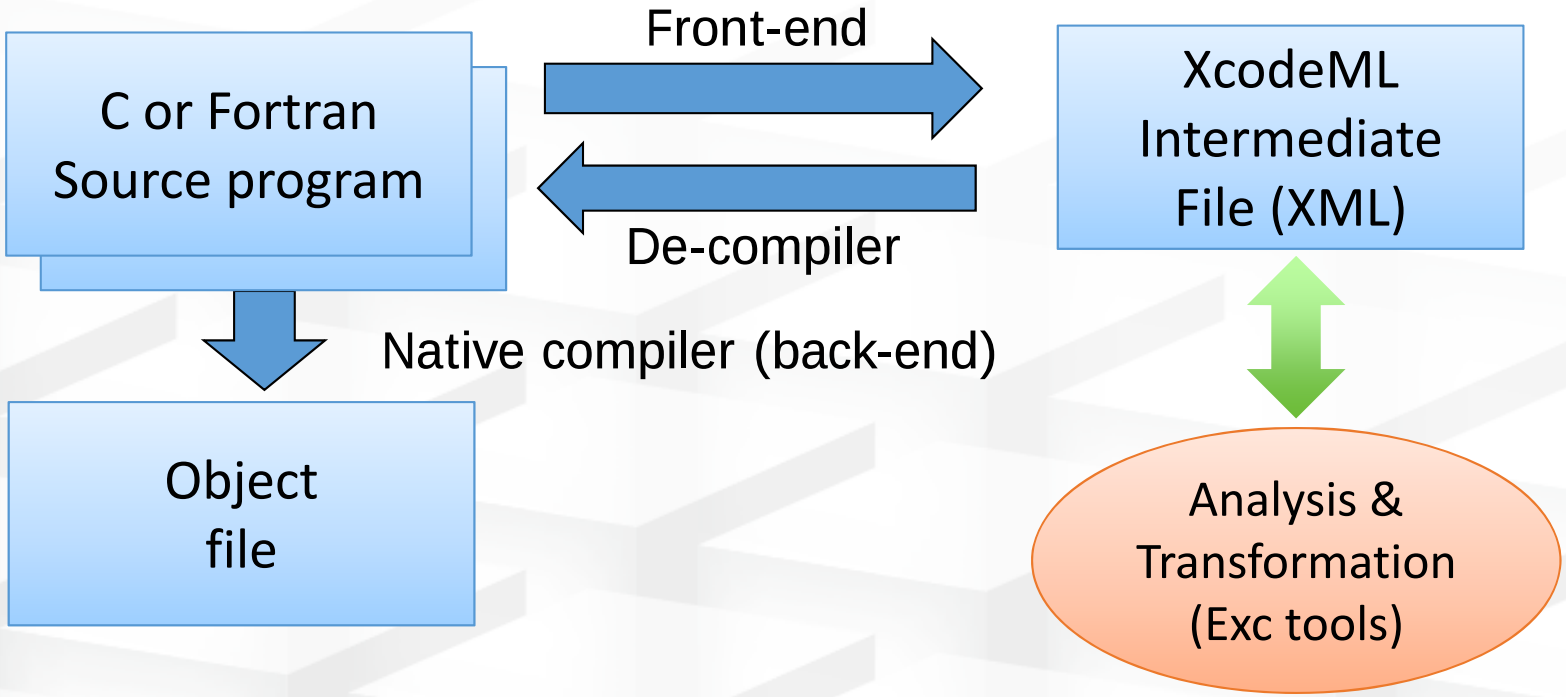


What is “Omni Compiler”?

- **Omni compiler is a collection of programs and libraries that allow users to build code transformation compilers.**
- **Omni Compiler is to translate C and Fortran programs with directive-based extensions such as XcalableMP and OpenACC directives into codes compiled by a native back-end compiler linked with runtime libraries.**
- **Supported directives:**
  - OpenMP (C, F95)
  - OpenACC (C)
  - XcalableMP (C, F95)
  - XcalableACC (C)

Overview of Omni Compiler

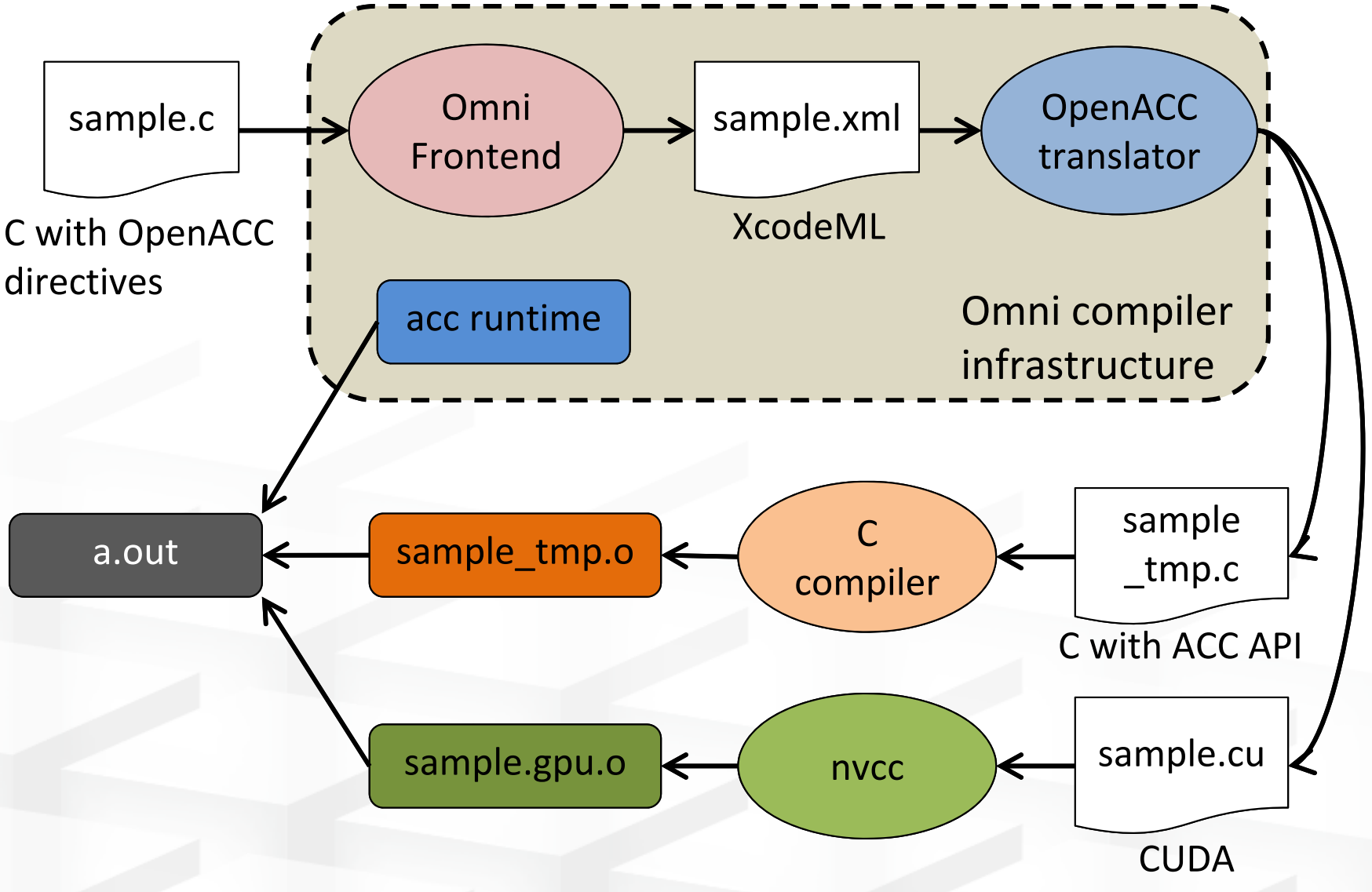
- **The source programs in C and Fortran are translated into an XML intermediate form call XcodeML by the front-end programs, C-front and F-front.**
  - Currently, only C and Fortran90 are supported.
- **The XcodeML is an intermediate form designed to be translated back to the source program.**
- **Exc tools are a collection of Java classes to analyze and transform programs.**



Brief History of Omni Compiler Project

- **1999-2003: Omni OpenMP compiler project started at RWC (Real World Computing) project (1992-2002)**
  - OpenMP 1.0 (C and Fortran77 (f2c)) supported
  - Cluster-enabled OpenMP (on top of DSM for Distributed memory)
  - Proprietary intermediate code (called Xcode) for C
- **(2003: supported by U. Tsukuba)**
- **2007: XcalableMP Specification Group launched**
- **2008-2011: e-science project**
  - C-front Revised (“gcc” based parser, gnu-extension supported)
  - F95 parser
  - XcodeML for C and F95
  - XcalableMP Spec v 1.0 released
  - Omni XcalableMP compiler for C released
- **2012 - : supported by RIKEN AICS, 2014- : adopted for post-K project**
  - Omni XcalableMP compiler for F95 released
  - Omni OpenACC (by U. Tsukuba)
  - 2016: Omni XcalableMP compiler v 1.0

Example: Omni OpenACC Compiler



Objective and Design of XcodeML

- **XML for an intermediate code**
  - Human-readable format in XML
  - It contains all information to translate the intermediate code back to the source program
  - Support for C and Fortran 90 (currently, working also for C++)
  - Syntax errors are supposed to be detected in front-end.
  - Separate front-end and transformation phase, and back-end
    - Rich tools to handle XML can be used.

- **Structures**
  - Type definitions
  - Symbol tables for global declarations
  - Program (function definitions) in ASTs

Exc tool kits: A Java Class Library for Analysis and Transformation

- Currently, the tools consist of the following two Java package.
  - **Package exc.object**: this package defines the data structure. “Xobject”, which is an AST (Abstract Syntax Tree) for XcodeML intermediate form. It contains XcodeML input/output, classes and methods to build and access AST represented by “Xobjects”. It is useful to scan and analyze the program.
  - **Package exc.block**: this package defines data structure called “Block” to represents block structures of programs. It is useful to analyze and transform the programs.

- Classes to implement languages
  - Pacakge exc.openmp: A translator for OpenMP compiler.
  - Package. exc.xcalablemp, exc.xmpf: A translater for XcalableMP compiler.

XcodeMLのデモ

Design patterns in exc tool kits (1)

- **XobjectIterator is an iterator to traverse all Xobjects represented in Xobject class.**
- **Subclass:**
  - bottomupXobjectIterator
  - topdownXobjectIterator

```
XobjectIterator i = new topdownXobjectIterator(x);
for(i.init(); !i.end(); i.next(){
    x = i.getXobject();
    ...; // some operations ...
}
```

Design patterns in exc tool kits (2)

- **Class XobjectDef is a container of external definitions in Xobject File.**
- **Use visitor pattern to access XobjectDef in XobjectFile.**
  - iterateDef method of XobjectFile traverses XobjectDef.

```
class MyFuncVistor implements XobjectDefVisitor {
   public void doDef(XobjectDef d){
       ... do something on definition of d ...
   }
   ...
}

/* in main routine */
XobjectVisitor op = new MyFuncVisitor;
f.iterateFuncDef(op);
```

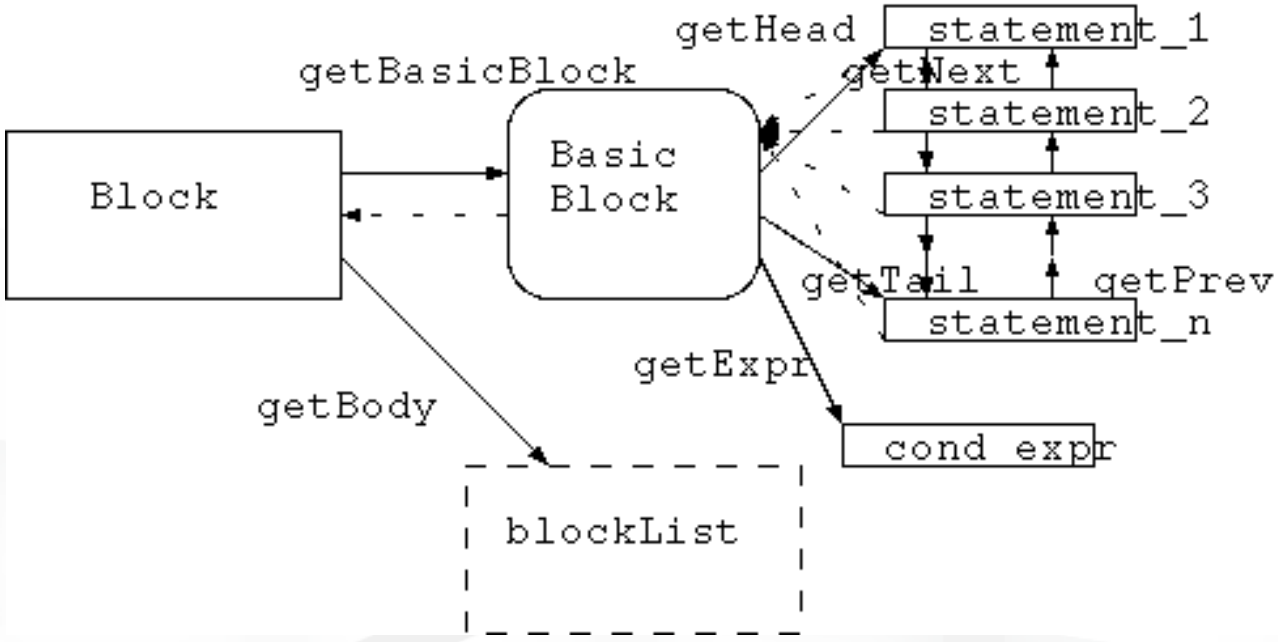
Simple Example using exc tools: print all variable reference

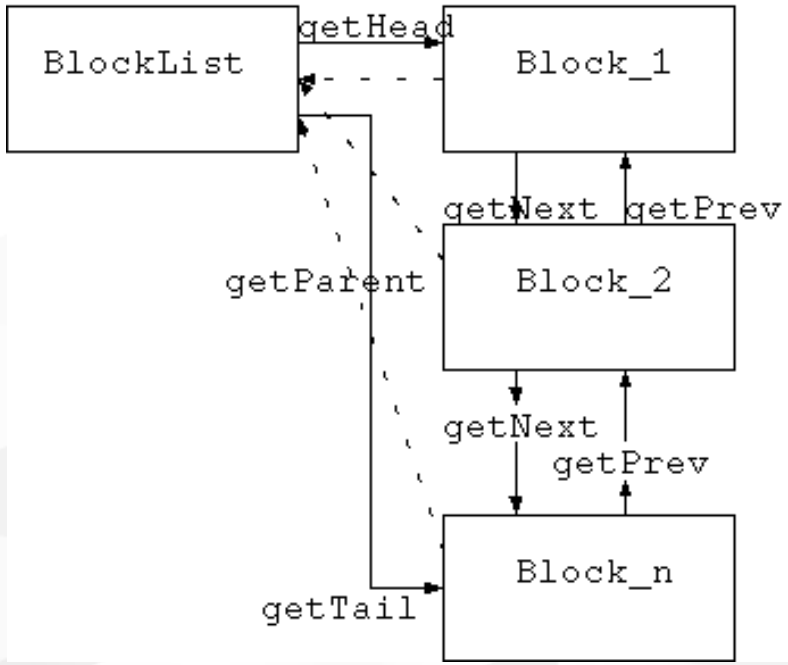
- **Use XobjectIterator class to scan AST**
- **Use visitor pattern to access XobjectDef in XobjectFile.**

```
class PrintVarRef implements XobjectDefVisitor {
   public void doDef(XobjectDef d){
        String fname = d.getName();
           XobjectIterator i = new topdownXobjectIterator(d.getFunBody());
           for(i.init();!i.end(); i.next()){
            Xobject x = i.getXobject();
            if(x.isVariable() || x.isVarAddr())
                   System.out.println("Variable '"+v.getName()+
                      "' is referenced from Function '"+fname+"'");
        }
   }
   public static void main(String args[]){
       XobjectFile f = new XobjectFile();
       f.Input("foo.x");
       XobjectVisitor op = new PrintVarName();
       f.iterateFuncDef(op);
   }
}
```

Block and BlockList in exc.block

- **Data structure for code transformation**
  - Double-linked list to insert/delete.
  - Create new function definition





Example 1: Translation of OpenACC parallel construct

```
#pragma acc parallel num_gangs(1) vector_length(128)
{
      /* codes in parallel region */
}
```

```
__global__ static  void _ACC_GPU_FUNC_0_DEVICE( ... )
{
      /* codes in parallel region */
}
```

GPU kernel function

```
extern "C" void _ACC_GPU_FUNC_0( ... )
{
      dim3 _ACC_block(1, 1, 1), _ACC_thread(128, 1, 1);
      _ACC_GPU_FUNC_0_DEVICE<<<_ACC_block,_ACC_thread>>>( ..
. );
      _ACC_GPU_M_BARRIER_KERNEL();
}
```

kernel launch function

Example 2: Translation of OpenACC loop construct

```
/* inner parallel region */
#pragma acc loop vector
for(i = 0; i < N; i++){
      a[i]++;
}
```

virtual index : _ACC_idx
virtual index range : _ACC_init, cond, step

```
/* inner gpu kernel code */
int i, _ACC_idx;
int _ACC_init, _ACC_cond, _ACC_step;
_ACC_gpu_init_thread_x_iter(&_ACC_init, &_ACC_cond, &_ACC_step, 0, N, 1);
for(_ACC_idx = _ACC_init; _ACC_idx < _ACC_cond; _ACC_idx += _ACC_step){
      _ACC_gpu_calc_idx(_ACC_idx, &i, 0, N, 1);
      a[i]++;
}
```

calculate the range of virtual index

virtual index

range variables

calculate 'i' from virtual index

loop body

Projects using Omni Compiler/XcodeML

- XcalableMP
- XcalableACC (XcalableMP + OpenACC)
- CLAW Fortran Compiler (Meteo Swiss)

Current Status and Plan

- **On going:**
  - Documentation !!!!!
  - Fortran 2003 support
    - We got support from the vender partner. They provide us test-suite of F2003.
  - C++ support
    - Using Clang/LLVM front-end to convert Clang AST to XcodeML

- **Plan**
  - XMP 2.0 (tasklet construct for multithreaded execution in manycore)
  - XcalableMP for C++ (object-oriented features)
  - OpenACC (XcalableACC) for Fortran 2003

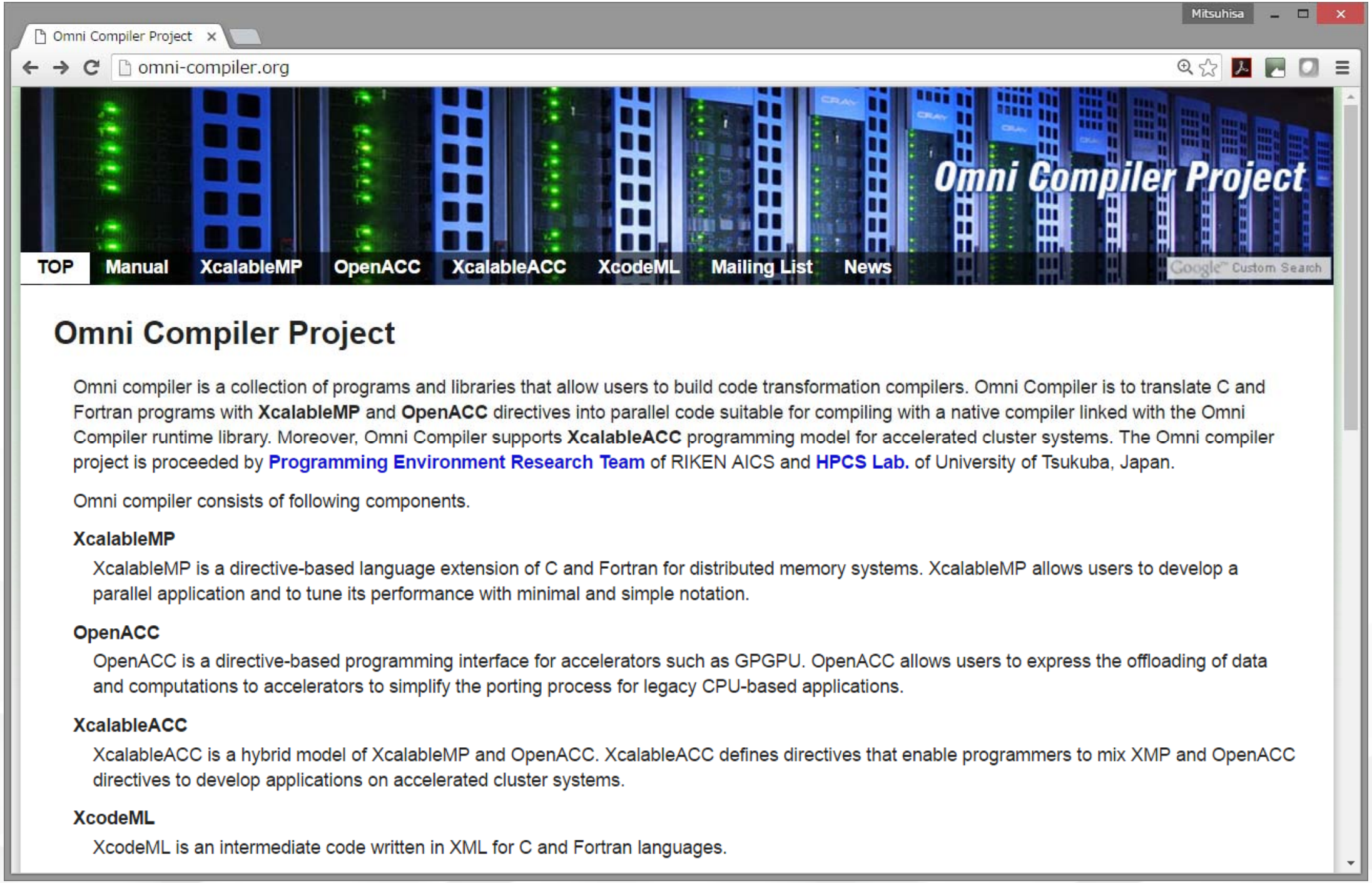
“メタプログラミング” in Fortran?

- SWoPP 2017で発表

Omniコンパイラ基盤における HPC向けメタプログラミング機能の構想

村井 均[1]　佐藤 三久[1]　李 珍泌[1]　中尾 昌広[1]　岩下英俊[1]

概要：Omni は，Fortran および C をターゲットとする，ソース to ソース変換に基づくコンパイラ基盤である．Omni は，理研と筑波大によって開発されており，Omni XcalableMP コンパイラの基盤としても用いられている．現在，我々は，主にプログラム生産性の向上を目的として，Omni における HPC 向けメタプログラミング機能の設計を進めている．本報告では，本機能の設計および予備的実装を示すとともに，本機能の実現可能性および有効性を検証する．ループ・アンローリングおよび構造型のデータレイアウト最適化に関する適用例を用いた評価の結果，本機能により種々の変換を記述することが可能であることを確認できた．


Mitsuhisa
Omni Compiler Project
omni-compiler.org
Omni Compiler Project
TOP
Manual
XcalableMP
OpenACC
XcalableACC
XcodeML
Mailing List
News
Google Custom Search
Omni Compiler Project
Omni compiler is a collection of programs and libraries that allow users to build code transformation compilers. Omni Compiler is to translate C and Fortran programs with XcalableMP and OpenACC directives into parallel code suitable for compiling with a native compiler linked with the Omni Compiler runtime library. Moreover, Omni Compiler supports XcalableACC programming model for accelerated cluster systems. The Omni compiler project is proceeded by Programming Environment Research Team of RIKEN AICS and HPCS Lab. of University of Tsukuba, Japan.
Omni compiler consists of following components.
XcalableMP
XcalableMP is a directive-based language extension of C and Fortran for distributed memory systems. XcalableMP allows users to develop a parallel application and to tune its performance with minimal and simple notation.
OpenACC
OpenACC is a directive-based programming interface for accelerators such as GPGPU. OpenACC allows users to express the offloading of data and computations to accelerators to simplify the porting process for legacy CPU-based applications.
XcalableACC
XcalableACC is a hybrid model of XcalableMP and OpenACC. XcalableACC defines directives that enable programmers to mix XMP and OpenACC directives to develop applications on accelerated cluster systems.
XcodeML
XcodeML is an intermediate code written in XML for C and Fortran languages.

XMPと他の言語との連携

～Pythonと連携するPGAS言語 XcalableMPのプログラミングモデル～

Masahiro Nakao, Hitoshi Murai, Taisuke Boku and Mitsuhisa Sato, “Linkage of XcalableMP and Python languages for high productivity on HPC cluster system - Application to Graph Order/degree Problem”, PGAS-EI workshop, HPCAisa 2018.

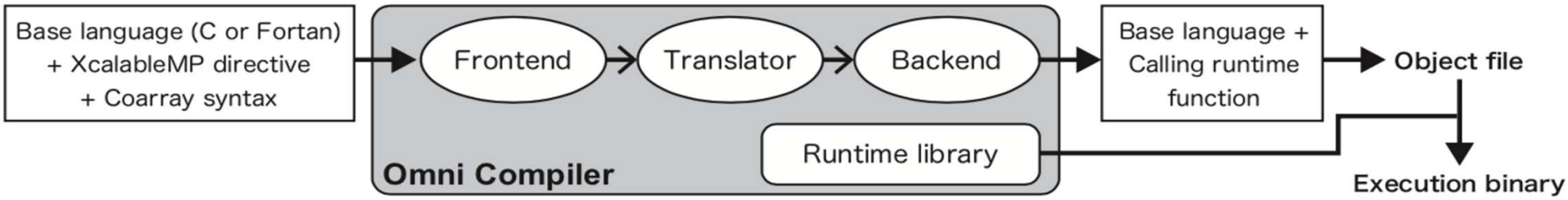
研究目的

- PGAS言語の目的は、プログラマの生産性を上げること
- そのためには、他の言語・ライブラリ・ツールなどとの連携が重要
  - 過去の研究で、BLASやプロファイリングツールとの連携は行った

PythonとXMPの連携機能の作成

- Pythonを利用するメリット
  - 既存のPythonライブラリの有効活用
    - SciPyやNumPyといった科学技術計算用ライブラリが充実
  - HPCでユーザが多い

Omni compilerの構成



- Source-to-Source コンパイラ
  - ユーザが書いた指示文と拡張構文があるコードは、Omni Compilerが用意しているランタイムの呼び出しを含むコードに変換される
  - Omni Compilerのランタイムは，基本的にはC + MPIを利用

MPIプログラムからXMPプログラムを利用する機能

- 下記関数を実装（xmp.hで定義）

Language	Return Value Type	Function	Description
XMP/C	void	xmp_init(MPI_Comm)	XMP 環境の初期化処理
XMP/F	(None)	xmp_init(Integer)	
XMP/C	void	xmp_finalize(void)	XMP 環境の完了処理
XMP/F	(None)	xmp_finalize()	

MPIプログラム（mpi.c）

```
#include <xmp.h>
#include <mpi.h>

int main(int argc, char **argv){
  MPI_Init(&argc, &argv);

  xmp_init(MPI_COMM_WORLD);
  call_xmp();
  xmp_finalize();
```

XMPプログラム（xmp.c）

```
void call_xmp(){
#pragma xmp nodes p[3]
      :
```

```
$ xmpcc xmp.c -c
$ mpicc mpi.c -c
$ xmpcc xmp.o mpi.o -o a.out
$ mpirun -np 3 a.out
```

XMPプログラムからMPIプログラムを利用する機能

- 下記関数を実装（xmp.hで定義）

Language	Return Value Type	Function	Description
XMP/C	void	xmp_init_mpi(int*, char***)	MPI 環境の初期化処理
XMP/F	(None)	xmp_init_mpi()	
XMP/C	MPI_Comm	xmp_get_mpi_comm(void)	実行中のノード集合に対する MPI コミュニケータの取得
XMP/F	integer	xmp_get_mpi_comm()	
XMP/C	void	xmp_finalize_mpi(void)	MPI 環境の完了処理
XMP/F	(None)	xmp_finalize_mpi()	

XMPプログラム（xmp.c）

```
#include <xmp.h>
#include <mpi.h>
#pragma xmp nodes p[3]

int main(int argc, char **argv){
  xmp_init_mpi(&argc, &argv);
  MPI_Comm comm = xmp_get_mpi_comm();
  call_mpi(comm);
  xmp_finalize_mpi();
```

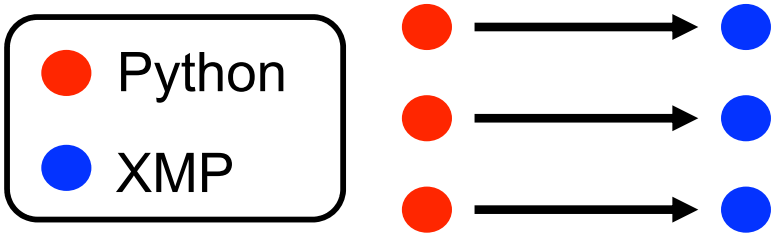
MPIプログラム（mpi.c）

```
#include <mpi.h>

void call_mpi(MPI_Comm comm){
  int rank, size;
  MPI_Comm_rank(comm, &rank);
  MPI_Comm_size(comm, &size);
```

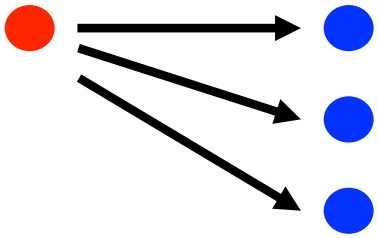
```
$ xmpcc xmp.c -c
$ mpicc mpi.c -c
$ xmpcc xmp.o mpi.o -o a.out
$ mpirun -np 3 a.out
```

XMPとPythonの連携機能

1. 並列Pythonプログラム（mpi4py）から、並列XMPプログラムを呼び出す
   - 前ページまでのMPIとの連携と概念は同じ（SPMD）



2. 逐次Pythonプログラムから、並列XMPプログラムを呼び出す
   - 1つの逐次プログラムが並列プログラムを呼び出す



並列Pythonプログラムから、並列XMPプログラムを呼び出す

- PythonのXMPパッケージを作成．下記の関数を提供

Function	Description
init(sharedFile, MPI.Intracomm)	*XMP環境の初期化*
finalize()	*XMP環境の完了処理*

Pythonプログラム（a.py）

```
import xmp
from mpi4py import MPI
import numpy  # 引数が必要な場合だけ

lib = xmp.init('xmp.so', MPI.COMM_WORLD)
arg = numpy.array([1,2])
lib.call_xmp(arg.ctypes)
xmp.finalize()
```

XMPプログラム（xmp.c）

```
void call_xmp(long arg[2]){
#pragma xmp nodes p[3]
      :
```

```
$ xmpcc -fPIC -shared xmp.c -o xmp.so
$ mpirun -np 3 python a.py
```

逐次Pythonプログラムから、並列XMPプログラムを呼び出す

- PythonのXMPパッケージを作成．下記の関数を提供

Function	*Description*
spawn(sharedfile, nodes, funcName, arguments)	*XMP プログラム実行*

Pythonプログラム（a.py）

```
import xmp
import numpy # 引数の必要な場合だけ

arg = numpy.array([1,2])
xmp.spawn("xmp.so", 3, "call_xmp", arg)
```

1プロセスで動作する

XMPプログラム（xmp.c）

```
void call_xmp(long arg[2]){
#pragma xmp nodes p[3]
      :
```

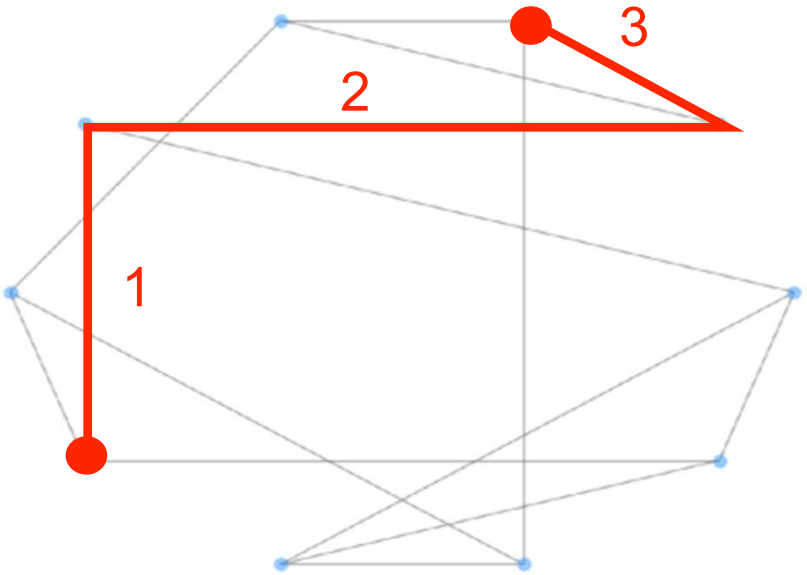
3プロセスで動作する

```
$ xmpcc -fPIC -shared xmp.c -o xmp.so
$ mpirun -np 1 python a.py
```

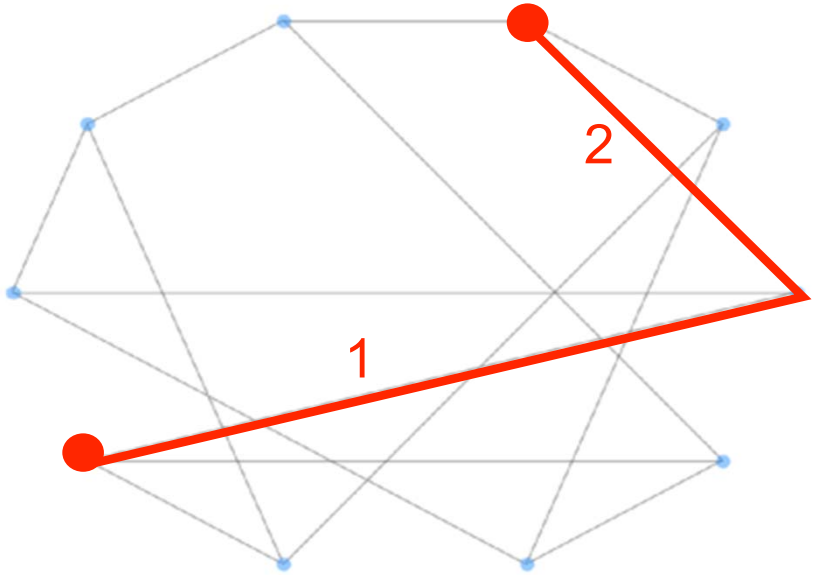
1プロセスで実行

Graph Order/degree問題とは

- 無向グラフにおいて与えられた頂点数と次数を持つ「直径」と「頂点間の平均距離」を最小化する問題のこと（低遅延な相互結合網の設計に役立つ [藤原 2015]）
  - 頂点数と次数から，最小の直径と平均距離は算出できる[V. G. Cerf 1974]が，どのようなグラフになるかはわからない（おそらくNP困難）
- 頂点数=10，次数=3の場合の例：



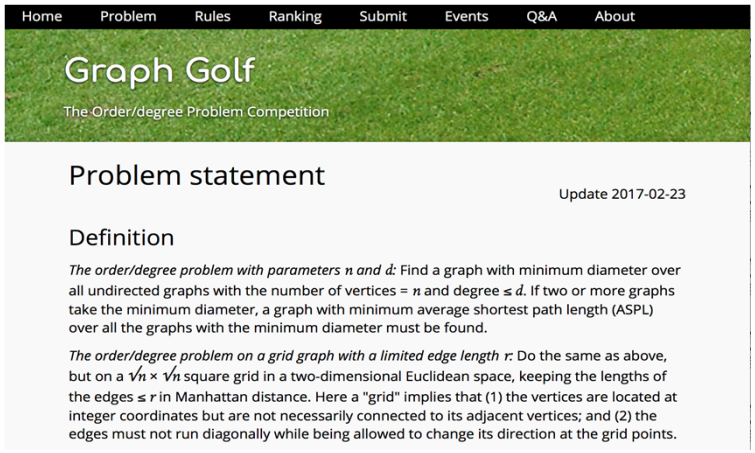
直径=3，平均距離=1.89（ランダム）



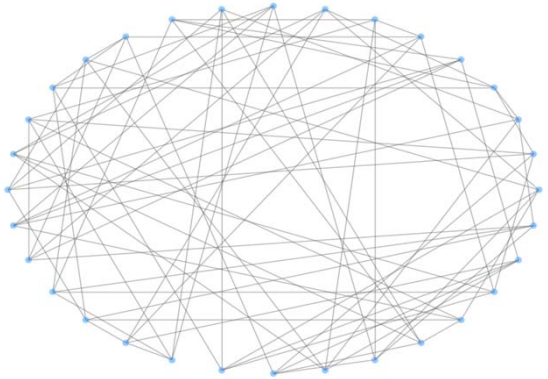
直径=2，平均距離=1.67（最適解）

Graph Golf@NII

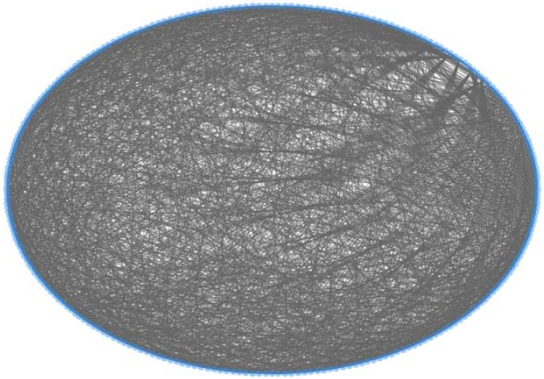
- Graph Order/degree問題に対するコンペティションをNIIが開催
  - 2015年から毎年開催（http://research.nii.ac.jp/graphgolf）
  - 複数の頂点数と次数の組合せを出題し，最小の直径と平均距離を出したグループを表彰する



頂点数=32，次数=5



頂点数=256，次数=18

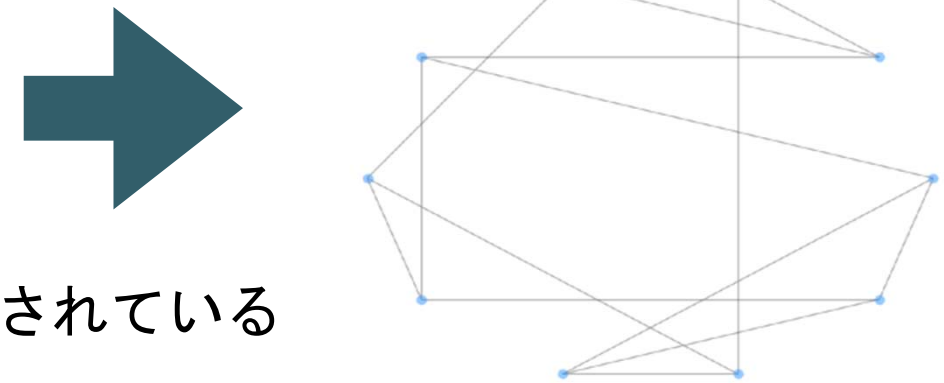


2017 年の General Graph Category の頂点と次数の組合せ

頂点数（n）	32	256	576	1344	4896	9344	88128	98304	100000	100000
次数（d）	5	18	30	30	24	10	12	10	32	64

XMPとPythonを組み合わせる利点

- Graph Golfの公式サイトで，本問題に対するPythonプログラムが公開されている
  - http://research.nii.ac.jp/graphgolf/py/create-random.py （100行くらい）
  - 下記を出力する
    - ランダムな初期解
    - ある解の直径と平均距離の出力
    - ある解をPNG形式で保存
  - Pythonのnetworkxパッケージが利用されている



直径=3，平均距離=1.89（ランダム）

- 課題
  - グラフの初期解を生成するが，最適解の探索は行わない
  - 直径と平均距離の算出に時間がかかる

この2つの課題に対する機能をXMPを使って作成する

Graph Order/degree solver in Python + XMP

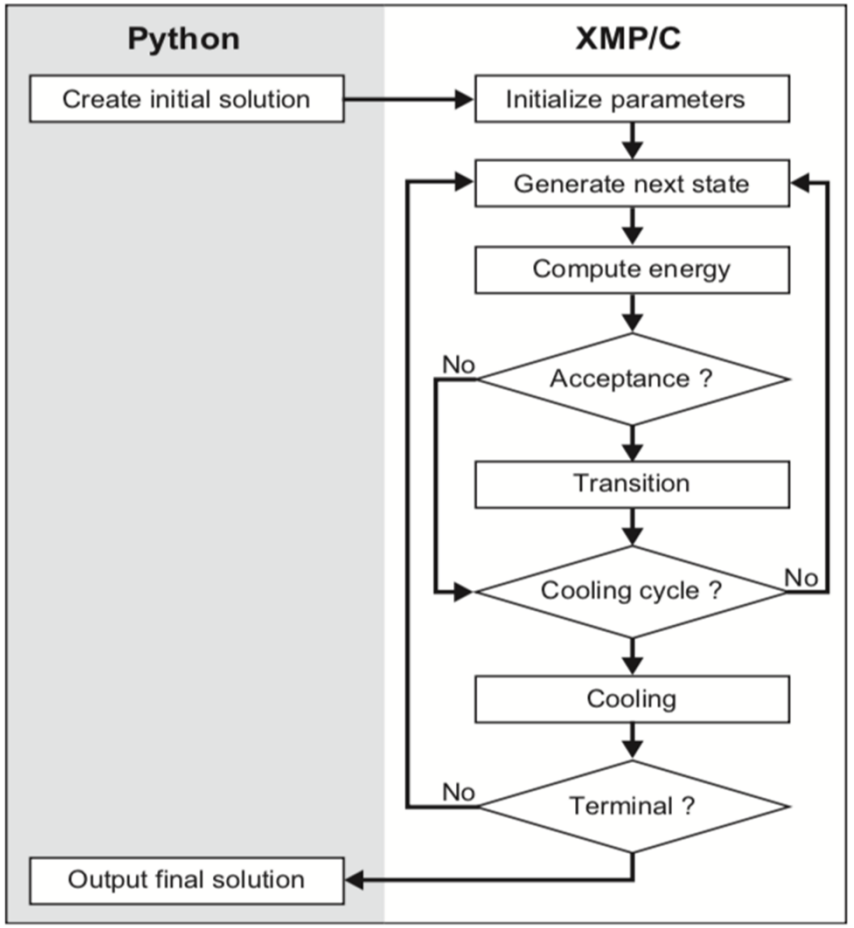
- 実装方針
  - 初期解の生成と図の出力は，既存のPythonコードを利用
  - 最適解の探索（直径と平均距離の計算を含む）はXMPを利用
    - 最適解探索はSimulated Annealingを用いる

Python

```
import xmp
  :
xmp.init("xmp.so", MPI.COMM_WORLD)
lib.sa(c_int(lines), edge, c_int(vertices))
xmp.finalize()
```

XMP/C

```
void sa(int lines, int edge[lines][2], int
vertices){
   :
#pragma xmp loop on t[i]
  for(int i=0;i<vertices;i++){
     : // 各頂点における直径と平均距離を
     : // 幅優先探索で算出
  }
#pragma xmp reduction(max:diameter)
#pragma xmp reduction(+:average_distance)
}
```



評価環境

- COMA@筑波大を利用

CPU	Intel Xeon-E5 2670v2 2.8 GHz x 2 Sockets
Memory	DDR3 1866MHz 59.7GB/s 64GB
Network	InfiniBand FDR 7GB/s
Software	intel/16.0.2, intelmpi/5.1.1, Omni Compiler 1.2.1 Python 2.7.9, networkx 1.9

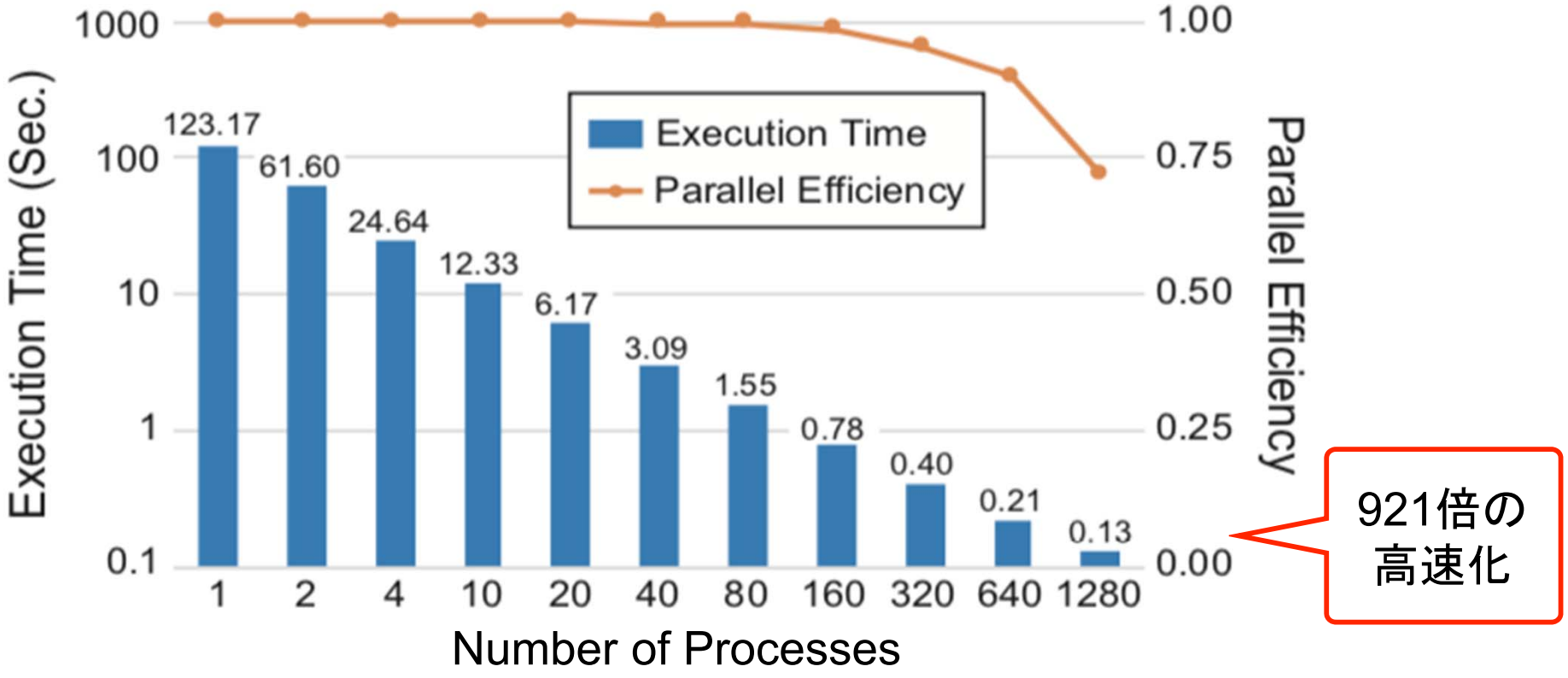
- 1回の直径と頂点間平均距離を計算する

2017 年の General Graph Category の頂点と次数の組合せ

頂点数（n）	32	256	576	1344	4896	9344	88128	98304	100000	100000
次数（d）	5	18	30	30	24	10	12	10	32	64

結果

- flat-MPIで実行
- 20プロセスまで1ノード内
- オリジナルのPythonコード（networkxパッケージ）で要する時間は148.83秒

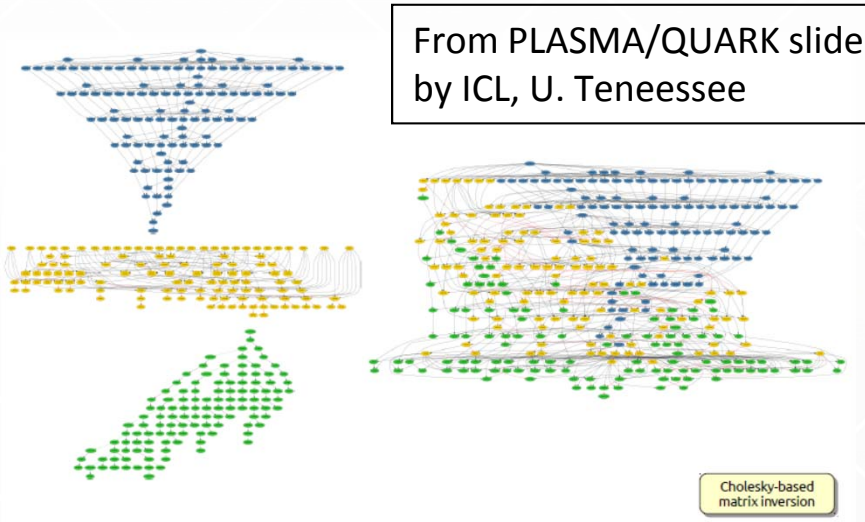


XcalableMP 2.0 に向けて
～ 分散メモリ上タスク並列モデル ～

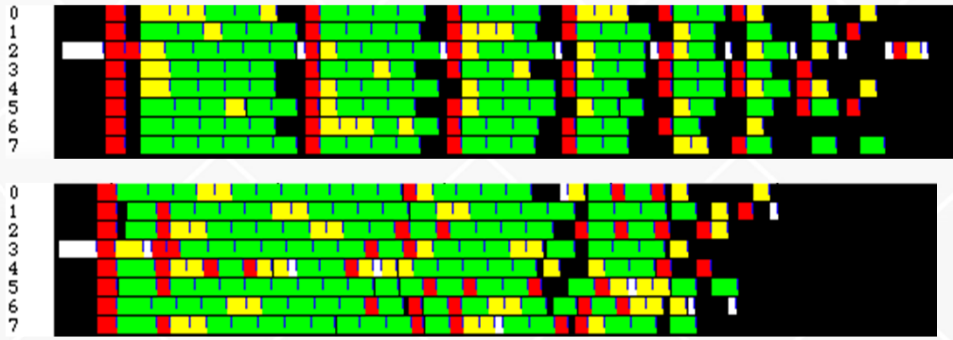
- 「PGAS モデルに基づく大規模並列クラスタ向け高生産並列プログラミングモデルに関する研究」筑波大学　津金佳祐　学位論文
- Keisuke Tsugane, Jinpil Lee, Hitoshi Murai, and Mitsuhisa Sato. “Multi-tasking Execution in PGAS Language XcalableMP and Communication Optimization on Many-core Clusters. In HPC Asia 2018: International Conference on High Performance Computing in Asia-Pacic Region, January 28–31, 2018, Chiyoda, Tokyo, Japan.

タスク並列プログラミングモデル

- **共有メモリ向けの様々なタスク並列モデル**
  - Cilk Plus，TBB，OpenMP
- **データ依存に基づくタスク並列モデルの登場**
  - OpenMP 4.0，Quark，StarPU，OmpSs



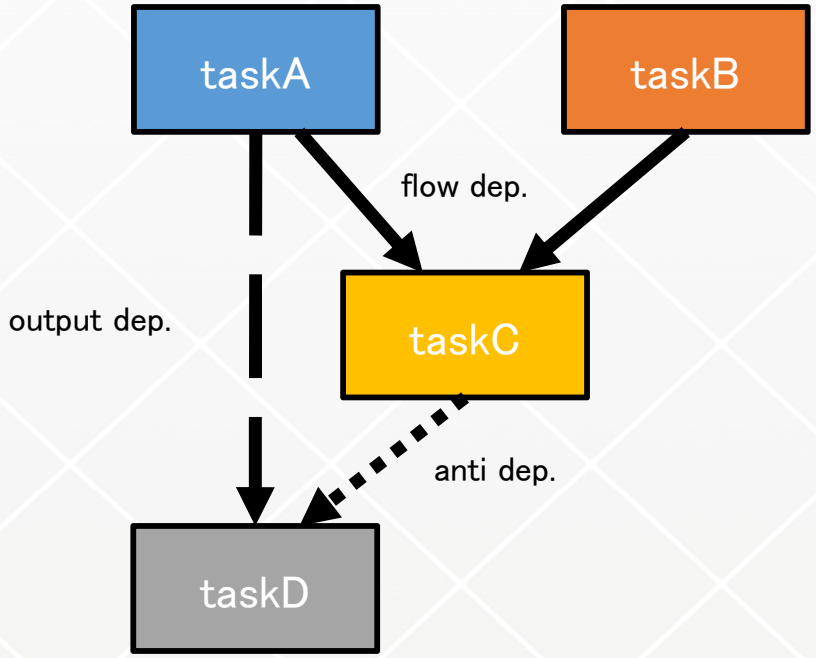
- **コアの多いメニーコアプロセッサにて高い性能が期待される**
  - 全体同期からデータ依存によるタスク間の細粒度な同期へ
  - 通信と演算のオーバラップが容易



Execution results of Cholesky Factorization in OpenMP 4.0 task

OpenMP task 指示文

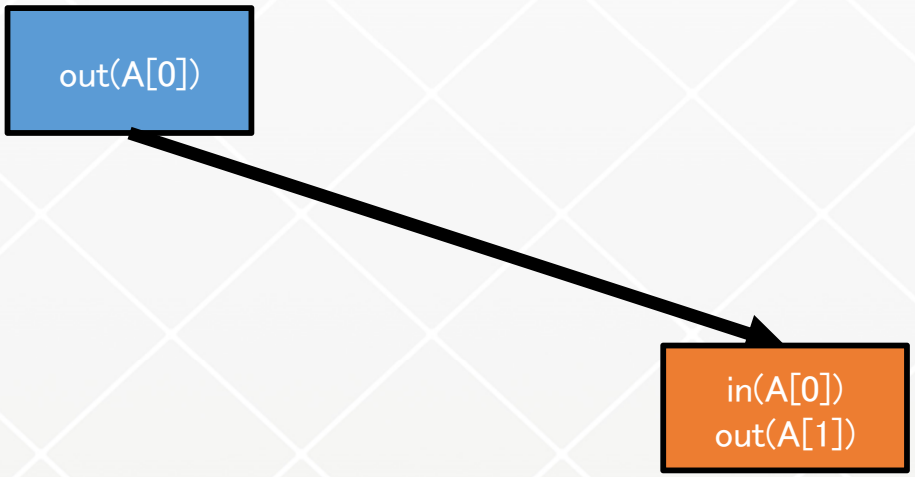
- **仕様 4.0 からはタスク間の依存関係を記述可能**
  - depend 節 の in，out，inout にデータ依存を記述
  - 逐次実行を基にした依存モデル（フロー，出力，反依存）
    - 異なるノードの並列動作するスレッド上のタスク依存生成は困難




```
#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 */
}
```

ノード間のタスク依存

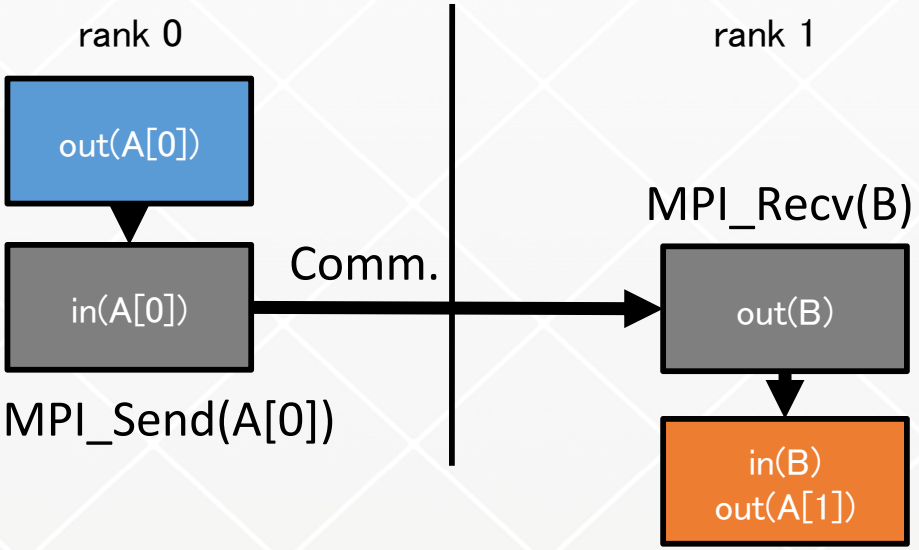
- **通信による依存関係の指定**
  - ノード間の依存関係 ＝ データの移動（通信）と定義
  - 通信をタスク化し通信対象をデータ依存とする
    - 依存関係が満たされれば（通信が終了すれば）演算タスクが実行




```
    int A[2];
#pragma omp parallel
#pragma omp single
{
#pragma omp task depend(out:A[0])
    funcA(A[0]);
#pragma omp task depend(in:A[0]) depend(out:A[1])
    funcB(A[0], A[1]);
}
```

ノード間のタスク依存 (cont'd)

- **通信による依存関係の指定**
  - ノード間の依存関係 = データの移動（通信）と定義
  - 通信をタスク化し通信対象をデータ依存とする
    - 依存関係が満たされれば（通信が終了すれば）演算タスクが実行




```
    int A[2], B;
if (rank == 0) {
#pragma omp task depend(out:A[0])
    funcA(A[0]);
#pragma omp task depend(in:A[0])
    MPI_Send(A[0], 1, …);

} else if (rank == 1) {
#pragma omp task depend(out:B)
    MPI_Recv(B, 0, …);
#pragma omp task depend(in:B) depend(out:A[1])
    funcB(B, A[1]);
}
```

Send は in

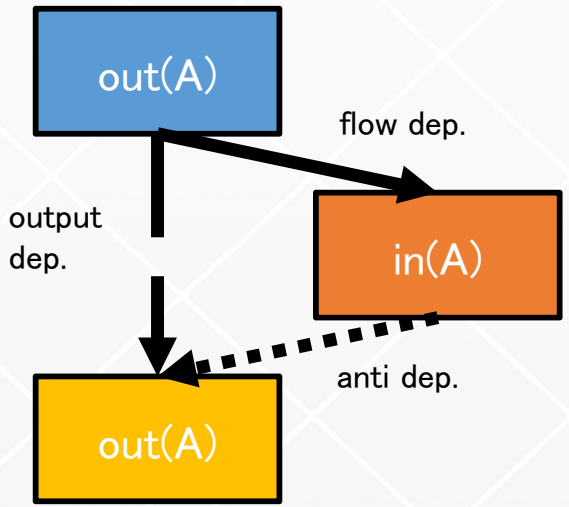
Recv は out

XMP tasklet 指示文の記述案

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

where *clause* is :
{in | out | inout} *(event_variable[, …])*

- **実行モデル**
  - OpenMP のタスク依存と同様
  - 各ノードがそれぞれタスクを生成，並列実行
    - tasklets 指示文で囲われた範囲のみ並列実行
  - on 節による実行ノードの指定
- **ノード内/間のタスク依存の記述が必要**
  - ノード内は OpenMP
  - ノード間は通信で記述
    - XMP の各モデルの通信を依存関係として使えるように拡張



グローバルビューにおけるtasklet 指示文

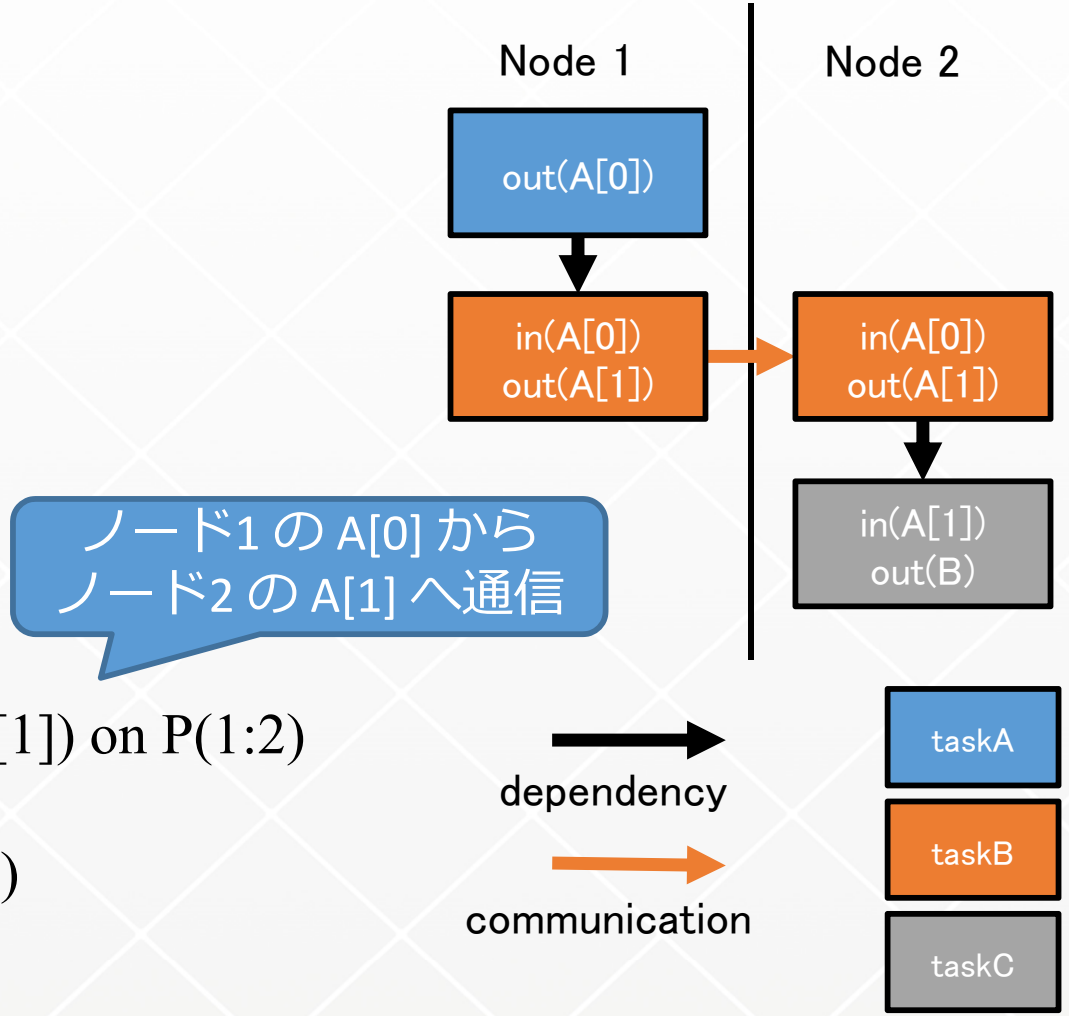
#pragma xmp tasklet gmove *[clause[, ...]] [*on {*node-ref* | *template-ref*}*]*
(an assignment statement)

where *clause* is :
{in | out | inout} *(event_variable[, …])*

- **tasklet gmove，tasklet reflect 指示文を提案**
- **タスク内で gmove を実行**
  - 配列代入文に分散配列が含まれる場合に通信が発生
  - 通信相手は実行ノード集合内で決定
  - 実行ノード集合内での暗黙の同期を含まない
    - 基本的に 1 対 1 通信とし，同期は各通信単位で実行
- **tasklet 指示文同様に in，out，inout 節を提供**

グローバルビューにおけるtasklet 指示文の例

```
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 tasklets
{
#pragma xmp tasklet out(A[0]) on P(1)
    A[0] = 0;        /* taskA */
#pragma xmp tasklet gmove in(A[0]) out(A[1]) on P(1:2)
    A[1] = A[0];   /* taskB */
#pragma xmp tasklet in(A[1]) out(B) on P(2)
    B = A[1];        /* taskC */
}
```



ローカルビューにおけるtasklet 指示文

#pragma xmp tasklet *[clause[, ...]]* [on {*node-ref* | *template-ref*}]
(an assignment statement)

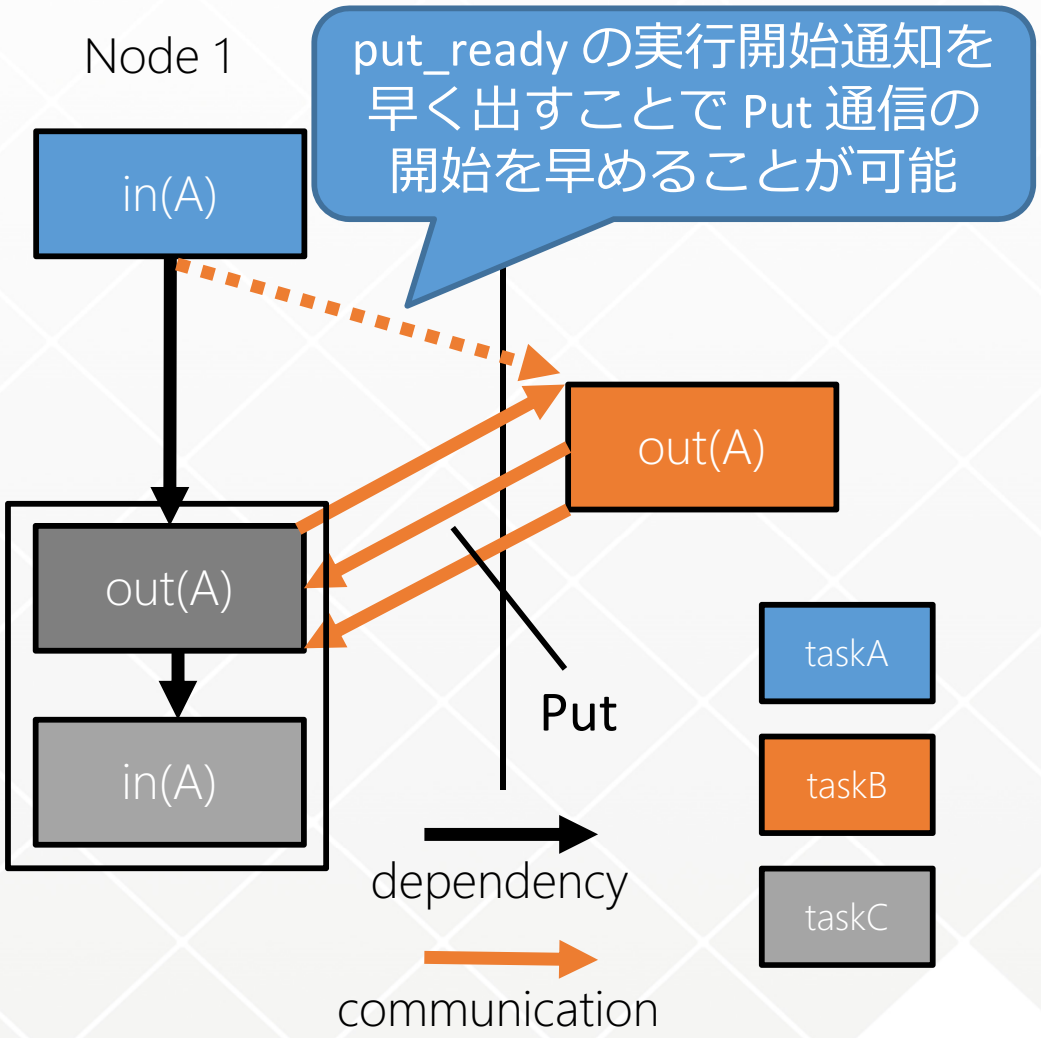
where *clause* is :
{put | get} (*tag*)
or
{put_ready | get_ready} (*variable*, {*node-ref* | *template-ref*}, *tag*)

- **タスク内でユーザ記述の coarray を実行可能**
  - タスク間の片側通信なので，”通信可能通知”，”片側通信の実行（coarray）”，”通信完了通知”が必要
- **片側通信に合わせた，”通信可能通知”/”通信完了通知”を記述する節を提供（put_ready/put，get_ready/get 節）**

ローカルビューにおけるtasklet 指示文の例

- ノード2 の taskB が ノード1 の変数 A に Put を実行

```
#pragma xmp nodes P(2)
    int A:[*], B, C, tag;
#pragma xmp tasklets
{
#pragma xmp tasklet in(A) out(B) on P(1)
    B = A;          /* taskA */

#pragma xmp tasklet out(A) put(tag) on P(2)
    A:[1] = 1;     /* taskB */

#pragma xmp tasklet in(A) out(D) ¥
                put_ready(A, P(1), tag) on P(1)
    C = A;          /* taskC */
}
```



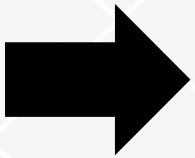
試作実装

- **Omni XMP コンパイラに tasklet 指示文を実装**
  - MPI+OpenMP へのコード変換
  - 軽量スレッドライブラリ Argobots@ANL による実装：任意地点で実行可能なタスクスイッチの機能を実装
    - スレッド内における通信と計算のオーバラップによる高速化
- **MPI+OpenMP コードへの変換**
  - 提案指示文を対応する OpenMP 指示文へと変換
    - tasklet 指示文 -> task 指示文
  - 基本的に XMP ランタイムの gmove，coarray 実装を使用
    - MPI 通信は以下のように変更

```
MPI_Send(…);
```

or

```
MPI_Isend(…);
MPI_Wait(…);
```



```
MPI_Isend(…);
MPI_Test(&comp, …);
while (!comp) {
#pragma omp taskyield
    MPI_Test(&comp, …);
}
```

非同期に通信完了を確認

別のタスクへスイッチ

評価：実験環境

- **実験環境**
  - Oakforest-PACS（最先端共同 HPC 基盤施設：JCAHPC）
- **評価項目**
  - ブロックコレスキーベンチマーク
  - 問題サイズ：32k × 32k，ブロックサイズ：512 × 512
  - 最大 32 ノード，64 スレッド / ノード で実行

CPU	Intel Xeon Phi 7250 (68 cores)
Memory	16 GB (MCDRAM) 96 GB (DDR4)
Interconnect	Intel Omni-Path Architecture
Software	Intel Compiler ver. 17.0.1 Intel MPI library 2017 update 1

Example

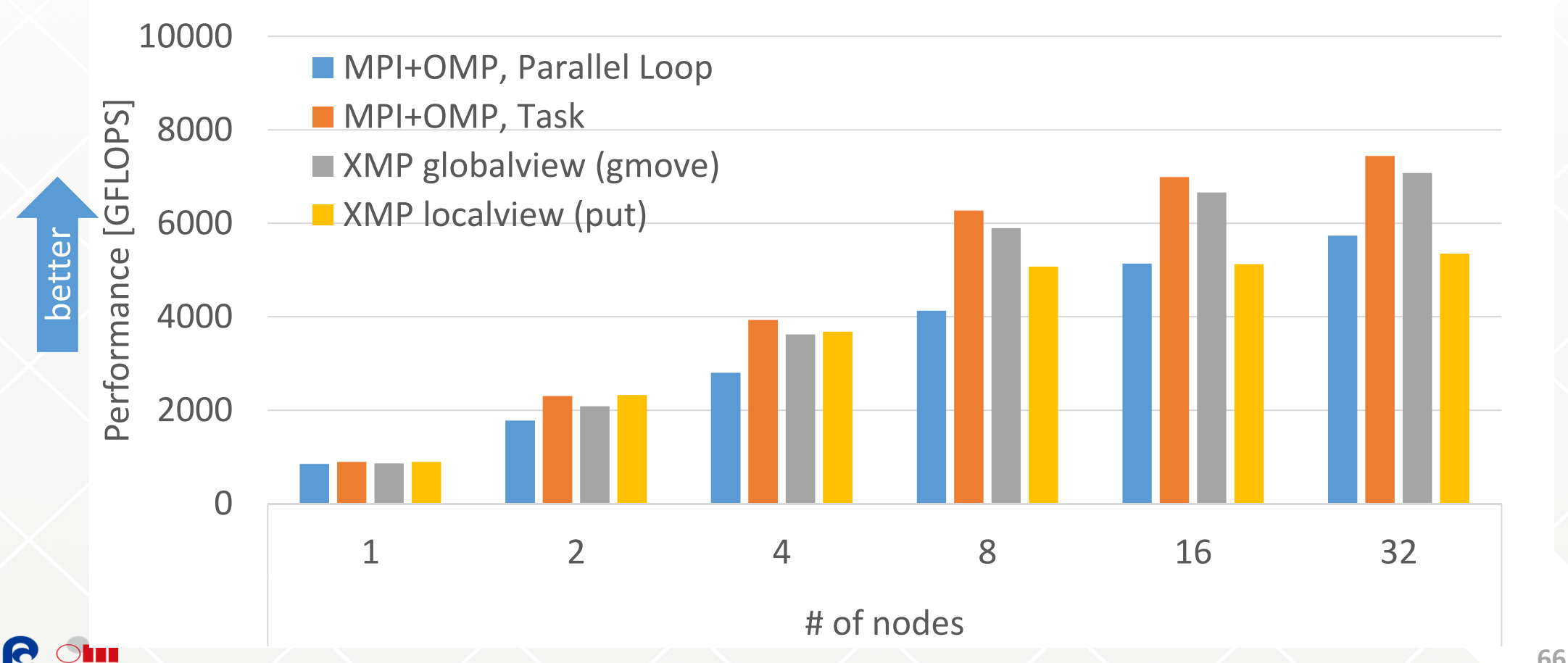
- **Block Cholesky Factorization**

```
  double A[nt][nt][ts*ts], B[ts*ts], C[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 out(A[k][k]) ¥
                              get ready(A[k][k], T(k:), k*nt+k) on T(k)
    potrf(A[k][k]);

#pragma xmp tasklet in(A[k][k]) out(B) get(k*nt+k) 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]) ¥
                      get ready(A[k][i], T(i:), k*nt+i) on T(i)
      trsm(A[k][k], A[k][i]);
    }
    for (int i = k + 1; i < nt; i++) {
#pragma xmp tasklet in(A[k][i]) out(C[i]) get(k*nt+i) on T(i:)
#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(j)
        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 taskletwait
#pragma xmp barrier
```

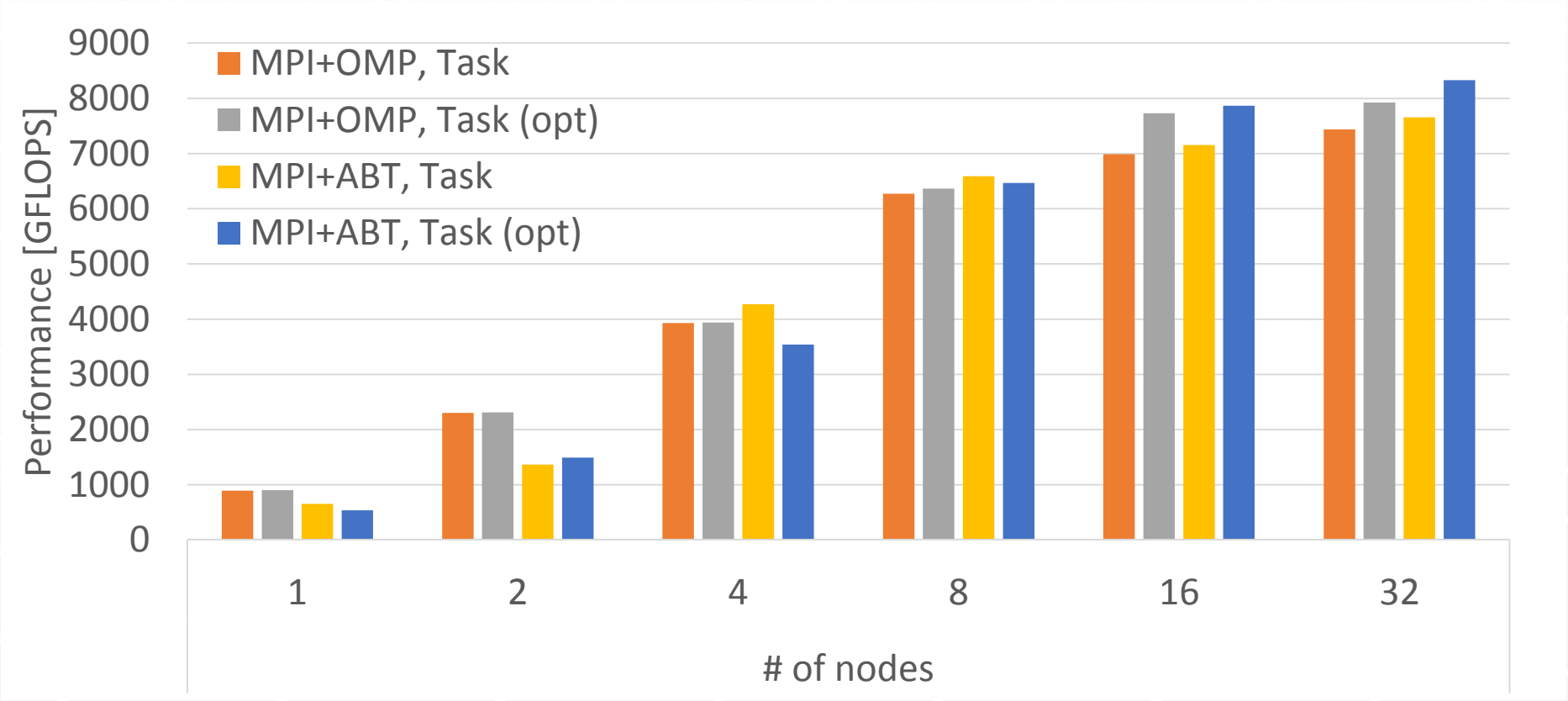
ブロックコレスキーベンチマークの結果

- ループ並列とタスク並列の比較
  - MPI+OMP (Parallel Loop) と MPI+OMP (task)
- XMP と MPI+OMP のタスク版の比較
  - XMP global-view/local-view と MPI+OMP (task)



メニーコア（OFP)における最適化

- **MPI_THREAD_MULTIPLEが遅かったので、通信を１スレッドに集めた（opt)**
- **Argobotで、スレッドのスイッチをできるようにした。**
  - OpenMPのthread_yieldは効かない。



これからのPGASプログラミングモデルの動向

PGAS applications workshop (PAW) at SC17, Keynote “How does PGAS “collaborate” with MPI+X?” by M. Sato, より。

“MPI+X” for exascale?

- **X is OpenMP!**
- **“MPI+Open” is now a standard programming for high-end systems.**
  - I’d like to celebrate that OpenMP became “standard” in HPC programming
- **Questions:**
  - “MPI+OpenMP” is still a main programming model for exa-scale?
  - How does PGAS “collaborate” with MPI+X?
  - Can PGAS replace(beat!?) “MPI” for exascale?

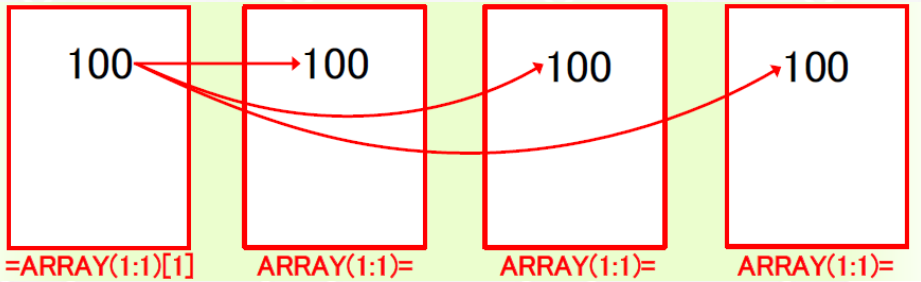
誤解：すべてをPGASで書かなくてはならないわけではない

- MPI broadcast operation can be written in CAF easily, …
- But, is it efficient?

```
REAL(8),DIMENSION(:),ALLOCATABLE :: ARRAY
・・・
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NIMG,IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,ID,IERR)
・・・
CALL MPI_BCAST(ARRAY, MSGSIZE, MPI_REAL8, 0,
MPI_COMM_WORLD,IERR)
```

- Should use “co_broadcast” in CAF.
- Sophisticated collective communication libraries of “matured” MPI are required
- Obviously, PGAS need to **collaborate** with MPI.

```
REAL(8),DIMENSION(:),CODIMENSION[:],ALLOCATABLE :: ARRAY
・・・
NIMG= NUM_IMAGES()
ID= THIS_IMAGE()
・・・
SYNC ALL
IF(ID /= 1) THEN
ARRAY(1:MSGSIZE) = ARRAY(1:MSGSIZE)[1]
END IF
SYNC ALL
```



PGAS とremote memory access (RMA)/one-sided comm.の違い

- PGAS is a programming model relating to distributed memory system with a shared address space that distinguishes between local (cheap) and remote (expensive) memory access.
  - Easy and intuitive to describe remote data access, for not only one side-comm, but also stride comm.
- RMA is a mechanism (operation) to access data in remote memory by giving address in (shared) address space.
  - RDMA is a mechanism to directly access data in remote memory without involving the CPU or OS at the destination node.
  - Recent networks such as Cray and Fujitsu Tofu support remote DMA operation which strongly support efficient one-sided communication.
- PGAS is implemented by RMA providing light-weight one-sided communication and low overhead synchronization semantics.
- For programmers, both PGAS and RMA are programming interfaces and offer several constructs such as remote read/write and synchronizations.
  - MPI3 provides several RMA (one-sided comm.) APIs as library interface.

PGASは、MPIに勝てるのか？
Can PGAS replace MPI? / Can PGAS be faster than MPI?

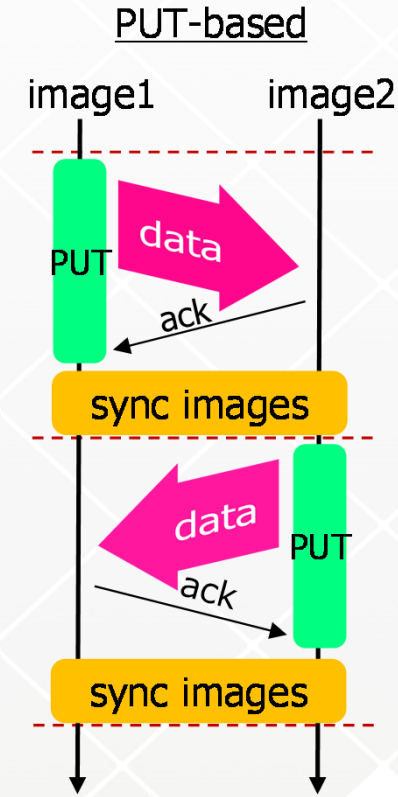
- Advantages of RMA/RDMA Operations
  - (Note: Assume MPI RMA is an API for PGAS)
  - multiple data transfers can be performed with a single synchronization operation
  - Some irregular communication patterns can be more economically expressed
  - Significantly faster than send/receive on systems with hardware support for remote memory access
    - Recently, many kinds of high-speed interconnect have hardware support for RDMA, including Infiniband, … as well as Cray and Fujitsu.

Case study: “Ping-pong” in CAF and RDMA

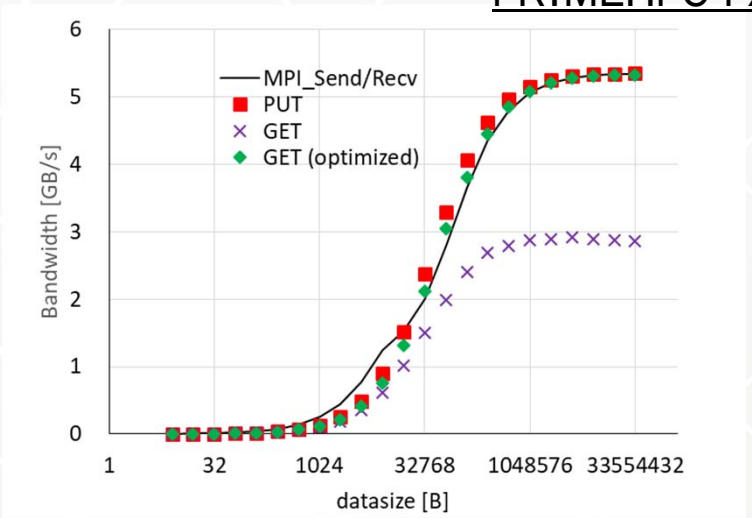
- Ping-pong: single data transfer with synchronization operation
- EPCC Fortran Coarray micro-benchmark

	image1	image2
Ping phase	x(1:ndata)[image2] = x(1:ndata) sync images	sync images
Pong phase	sync images	x(1:ndata)[image1] = x(1:ndata) sync images

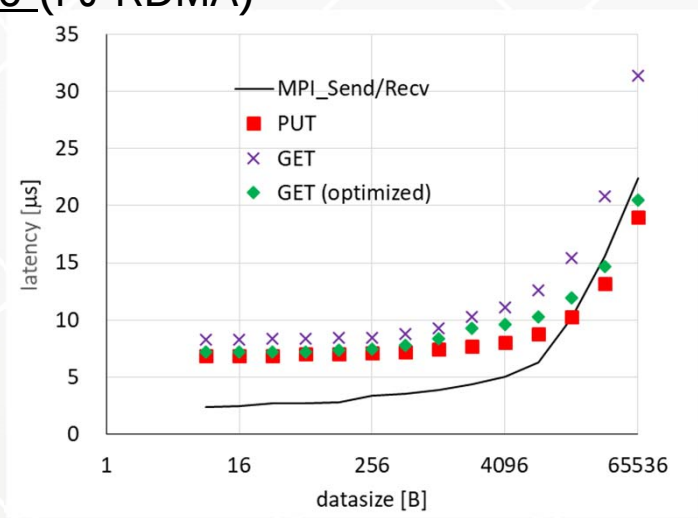
- Bandwidth of PUT-based achieves almost the same performance of send/recv. But, latency of small messages is bad.
- PUT exec time = data transfer + wait ack + sync_image
  - Sync_images exchange sync between nodes. It can be optimized by one-direction sync: post/wait (XMP), notify/query (Rice U, CAF2.0), event post/event wait (Fortran 2015)



PRIMEHPC FX100 (FJ-RDMA)



bandwidth

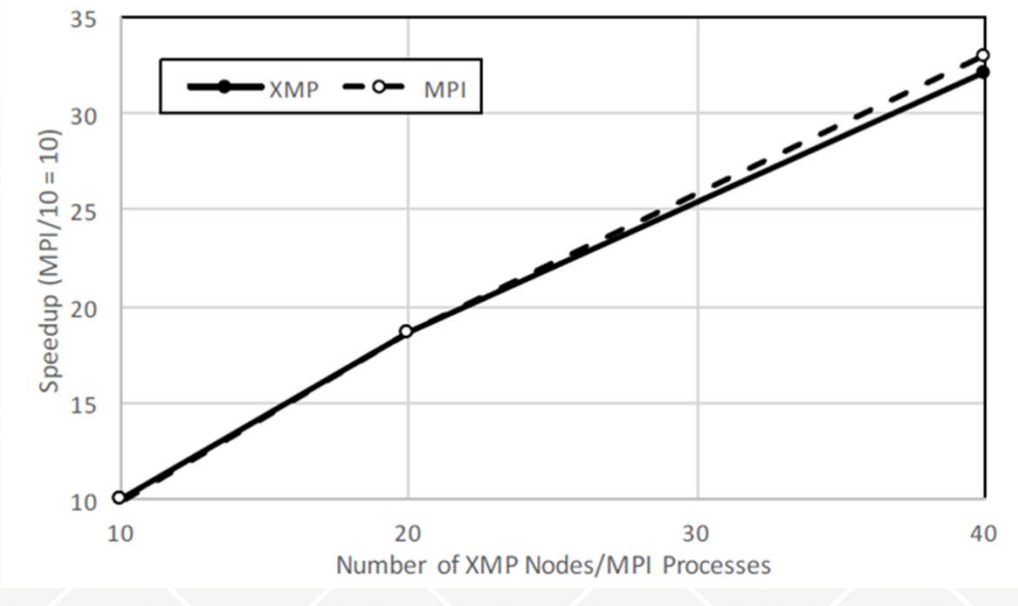


latency

Fiber min-apps in CAF/XMP

- Hitoshi Murai, Masahiro Nakao, Hidetoshi Iwashita and Mitsuhisa Sato, "Preliminary Performance Evaluation of Coarray-based Implementation of Fiber Miniapp Suite using XcalableMP PGAS Language“, PAW2017 (in morning)
- Replace of MPI with Coarray operation by simple rewriting rule.
- Our coarray-based implementations of three (NICAM-DC, NTChem-MINI, and FFB-MINI) of the five miniapps were comparable to their original MPI implementations.
- However, for the remaining two miniapps (CCS QCD and MODYLAS-MINI) due to communication buffer management

Result of NICAM-DC



Simple re-writing rule

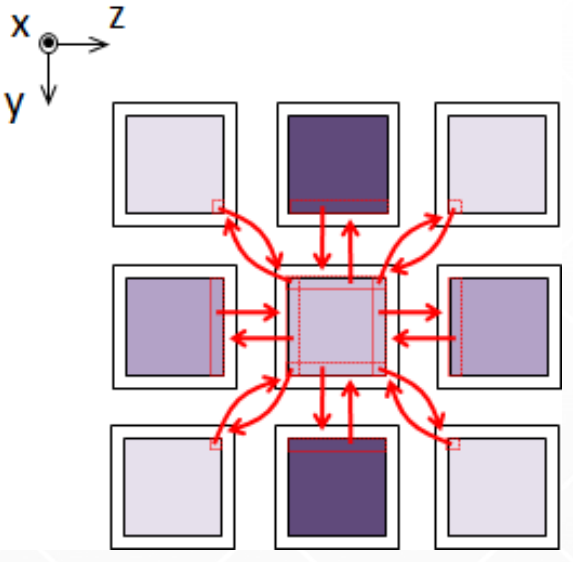
```
real a(8), b(8), c(8)

if (myrank == 0) then
  call MPI_Isend(a, 4, ..., 1, ...)
else if (myrank == 1) then
  call MPI_Irecv(b, 4, ..., 0, ...)
end if

call MPI_Wait(...)

call MPI_Bcast(c, 8, ..., 0, ...)
```

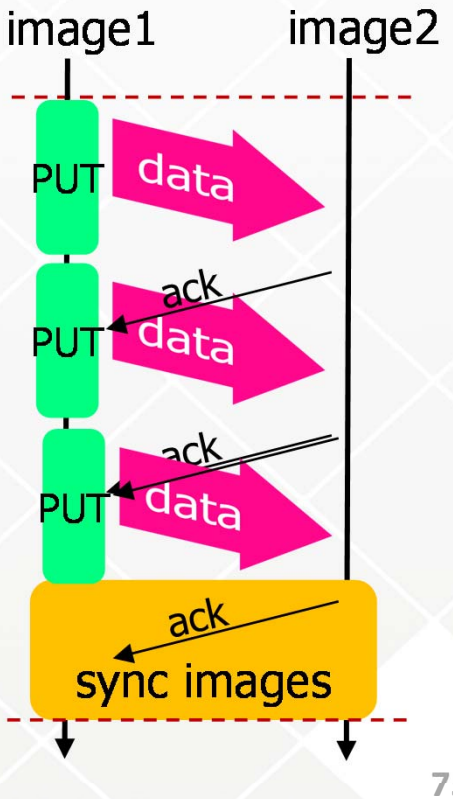
(a) MPI Program

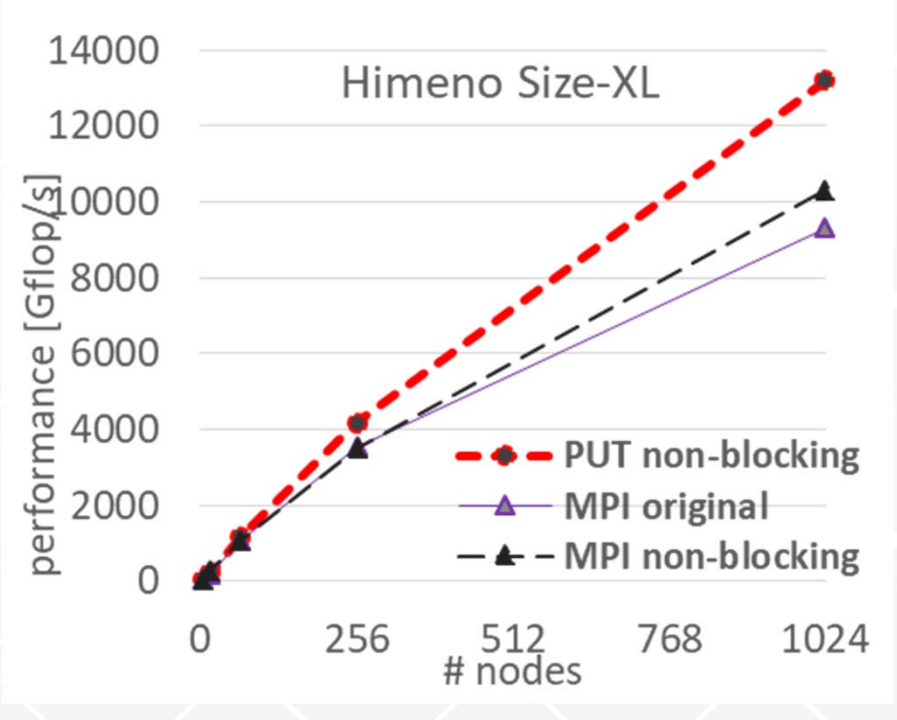
```
real a(8), b(8)[*], c(8)[*]

if (this_image() == 1) then
  ! put operation
  b(1:4)[2] = a(1:4)
else if (this_image() == 2) then
  ! recv removed
end if

sync all

call co_broadcast(c(1:8), 1)
```

Case study(2): stencil communication

- Typical communication pattern in domain-decomposition.
- Advantage of PGAS: Multiple data transfers with a single synchronization operation at end
- PUT non-blocking outperforms MPI in Himeno Benchmark!
  - Don’t wait ack before sending the next data (by FJ-RDMA)







NOTE: The detail of this results is to be presented in HPCAisa 2018: Hidetoshi Iwashita, Masahiro Nakao, Hitoshi Murai, Mitsuhisa Sato, “A Source-to-Source Translation of Coarray Fortran with MPI for High Performance”

PGASの得意な通信パターン

- When the communication pattern is expressed in some defined forms, PGAS comm. can be optimized.
  - Coarray assignment by array section syntax. Runtime select better stride comm, pack/unack or direct RDMA.

```
A(1, 1:10)[1] = B(1:10)    // put from image2 to image1
```

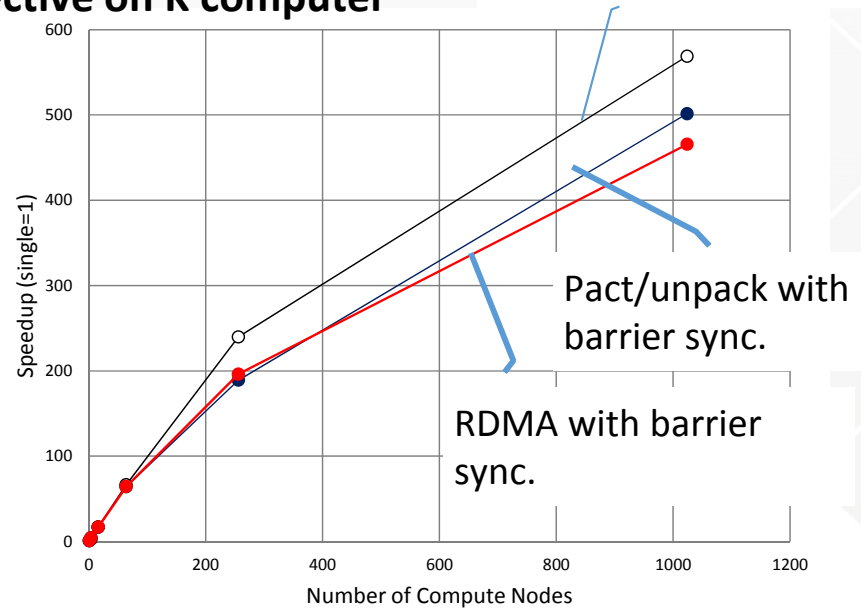
  - Reflect operation in XMP global view model (stencil neighbor comm)

- Irregular Communication Patterns:
  - If communication *pattern* is not known *a priori*, but the data locations are known, send-recv program needs an extra step to make pairs of send-recv. BUT, RMA can handle it easily because only the source or destination process needs to perform the put or get call

```
!$xmp shadow a(2:2, 1:1)
!$xmp reflect (a) width (/periodic/1:1, 0:0) async (0)
!$xmp wait_async (0)
```

- **shadow: declares width of shadow**
- **reflect: executes stencil comm.**
  - width: specifies shadow width to be updated. /periodic/: updates shadow periodically. async: asynchronous comm.
- **wait_async: completes async. reflects**

Optimized stencil communication by reflect directive on K computer



Target: a prototype dynamical core of a climate model SCALE-LES

MPI RMA は、PGASの低レベルの通信レイヤとして適当か？

- **“MPI is too low and too high API for communication”. (Prof. Marc Snir, JLESC 7th WS)**
  - MPI RMA APIs offer their PGAS model rather than “primitives” for other PGAS.

- **In case of our XMP Coarray implementation:**
  - Using “passive target”
  - MPI flush operation and synchronization do not match to implement “sync_images”.
  - Complex “window” management to expose the memory as a coarray.
  - (We need more study for better usage of MPI RMA)
  - Fujitsu RDMA interface is much faster in K-computer.

“Compiler-free” アプローチ：最近のトレンド

- Library approach: MPI3 RMA, OpenShmem, GlobalArray, …
- C++ Template approach: UPC++, DASH, …
- This approach may increase portability, clean separation from base compiler optimization, … but sometimes hard to debug in C++ template…

- But, approach by compiler will give:
  - New language, or language extension provides easy-to-use and intuitive feature resulting in better productivity.
  - Enable compiler analysis for further optimization: removal of redundant sync and selection of efficient communication, etc, …

  - But, in reality, compiler-approach is not easy to be accepted for deployment, and support many sites, …