

**PCクラスターコンソーシアム
第1回 XcalableMP ワークショップ**

**XcalableMPの仕様ワーキンググループと
Omniプロジェクト**

佐藤 三久

筑波大学・計算科学研究センター
理化学研究所・計算科学研究機構

目次



- XcalableMPプロジェクトの概要
 - e-science プロジェクト
 - PGAS並列プログラミング言語の現状・動向
 - XcalableMP規格部会 経緯・趣旨
- レファレンス実装の概要
 - Omni コンパイラプロジェクトとXcodeML
- XMP部会の状況と、XMP関連の研究プロジェクト
- エクサスケール時代のプログラミング言語について

“Petascale” 並列プログラミング WG (XcalableMP WG)



■ 目的

- “標準的な”並列プログラミングのためのペタスケールを目指した並列プログラミング言語の仕様を策定する
- “標準化”を目指して、“world-wide” communityに提案する。

■ Members（発足当時）

- Academia: M. Sato, T. Boku (compiler and system, U. Tsukuba), K. Nakajima (app. and programming, U. Tokyo), Nanri (system, Kyusyu U.), Okabe, Yasugi(HPF, Kyoto U.)
- Research Lab.: Watanabe and Yokokawa (RIKEN), Sakagami (app. and HPF, NIFS), Matsuo (app., JAXA), Uehara (app., JAMSTEC/ES)
- Industries: Iwashita and Hotta (HPF and XPFortran, Fujitsu), Murai and Seo (HPF, NEC), Anzaki and Negishi (Hitachi)

■ 2007年12月にkick-off, 現在、e-scienceプロジェクトの並列プログラミング検討委員会に移行

■ メーカーからのコメント・要望（活動開始時）

- 科学技術アプリケーション向けだけでなく、組み込みのマルチコアでも使えるようなものにするべき。
- 国内の標準化だけでなく、world-wideな標準を目指す戦略を持つべき
- 新しいものをつくるのであれば、既存の並列言語(HPF やXPFortranなど)からの移行パスを考えてほしい

「シームレス高生産・高性能プログラミング環境」(代表 東京大学 石川裕、H20-23, 3.5年)

「並列アプリケーション生産性拡大のための道具」の開発

PCクラスタから大学情報基盤センター等に設置されているスパコンまで、ユーザに対するシームレスなプログラミング環境を提供

■ 高性能並列プログラミング言語処理系

- 逐次プログラムからシームレスに並列化および高性能化を支援する並列実行モデルの確立とそれに基づく並列言語コンパイラの開発

■ 高生産並列スクリプト言語

- 最適パラメータ探索など粗粒度の大規模な階層的並列処理を、簡便かつ柔軟に記述可能で処理効率に優れたスクリプト言語とその処理系の開発

■ 高効率・高可搬性ライブラリの開発

- 自動チューニング(AT)機構を含む数値計算ライブラリの開発
- PCクラスタでも基盤センタースパコン(1万規模CPU)でも単一実行時環境を提供するSingle Runtime Environment Image環境の提供

高性能並列プログラミング言語処理系の開発

筑波大
学
東京大
学

次世代並列プログラミング言語仕様検討会
(主査：佐藤三久@筑波大)
NEC、富士通、日立、JAXA、
JAMSTEC、核融合研、筑波大、
東大、京大、九大

高生産並列スクリプト言語の開発

京都大
学

富士通研究所

高効率・高可搬性ライブラリ開発

東京大学

富士通研究所

日立中央研究所

■ 現状と課題

- 並列プログラムの大半はMPI通信ライブラリによるプログラミング
 - 生産性が悪く、並列化のためのコストが高い
- 並列プログラミングの教育のための簡便で標準的な言語がない(MPIでの教育にとどまっている)
- 研究室のPCクラスタから、センター、ペタコンまでに到るスケーラブルかつポータブルな並列プログラミング言語が求められている

■ 目標

- 既存言語を指示文により拡張し、これからの大規模並列システム(分散メモリシステムと共有メモリノード)でのプログラミングを助け、生産性を向上させる並列プログラミング言語を設計・開発する。
- 標準化をすることを前提に、ユーザのわかりやすさを第一にどこでも使えるということを重視し、開発ならびに普及活動を進める。

Current Problem ?!

```
int array[YMAX][XMAX];

main(int argc, char**argv){
  int i,j,res,temp_res,dx,llimit,ulimit,size,rank;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  dx = YMAX/size;
  llimit = rank * dx;
  if(rank != (size - 1)) ulimit = llimit + dx;
  else ulimit = YMAX;

  temp_res = 0;
  for(i = llimit; i < ulimit; i++){
    for(j = 0; j < 10; j++){
      array[i][j] = func(i,j);
      temp_res += array[i][j];
    }
  }

  MPI_Allreduce(&temp_res,&res,1,MPI_INT,MPI_SUM,MPI_COMM_WORLD);
  MPI_Finalize();
}
```

MPIしか、使えるものがない
MPIの並列プログラムはむずかしい... いっぱい書き換えな
いといけないし、時間がかかる、
デバックもむずかしいし、...



We need better solutions!!

```
#pragma xmp template T[10]
#pragma xmp distributed T[block]

int array[10][10];
#pragma xmp aligned array[i][*] to T[i]

main(){
  int i,j,res;
  res = 0;
  #pragma xmp loop on T[i] reduction(+)
  for(i = 0; i < 10; i++){
    for(j = 0; j < 10; j++){
      array[i][j] = func(i,j);
      res += array[i][j];
    }
  }
}
```

data
add to
incremen

work sharing and data
synchronization

いまのプログラムに指示文を加える
ただだから、簡単！性能チュー
ニングも可能、...
どこでも使えるから安心、並列プ
ログラミングも習得にもお勧め！

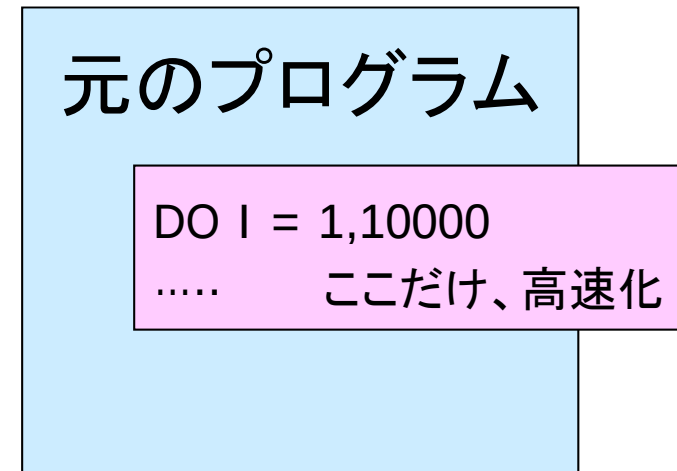


並列処理の問題点：並列化はなぜ大変か



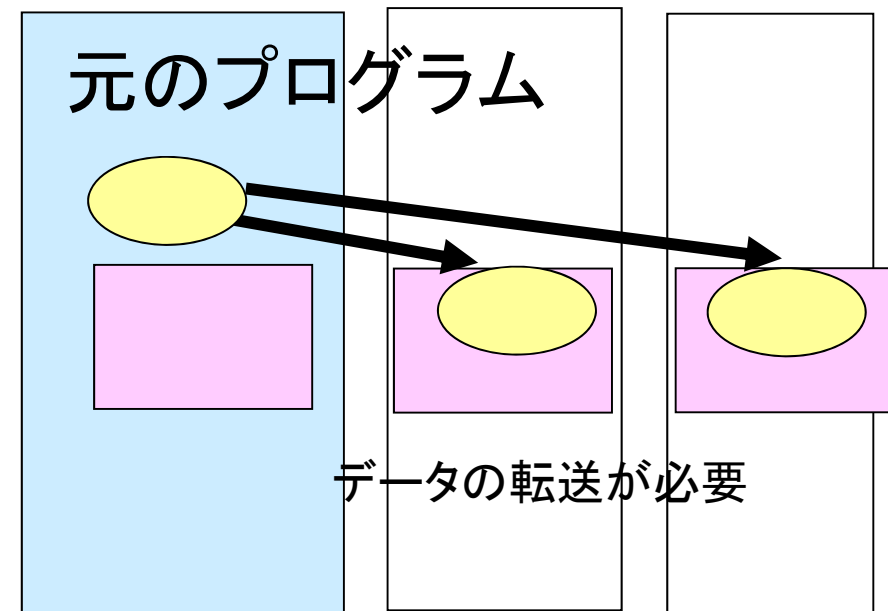
■ ベクトルプロセッサ

- あるループを依存関係がなくなるように記述
- ローカルですむ
- 高速化は数倍



■ 並列化

- 計算の分割だけでなく、通信（データの配置）が本質的
- データの移動が少なくなるようにプログラムを配置
- ライブラリ的なアプローチが取りにくい
- 高速化は数千倍～数万

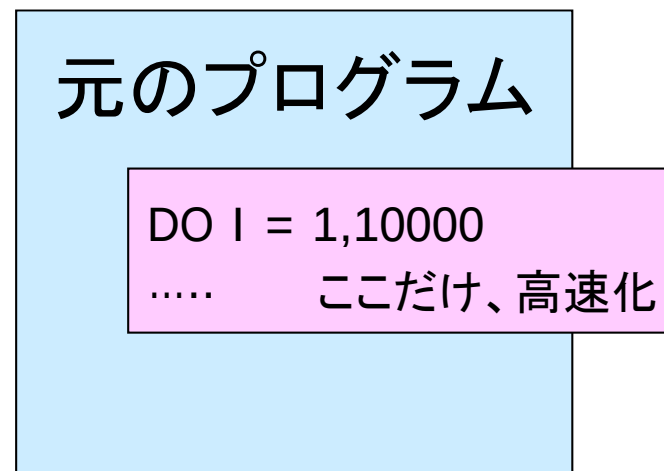


並列処理の問題点：並列化はなぜ大変か



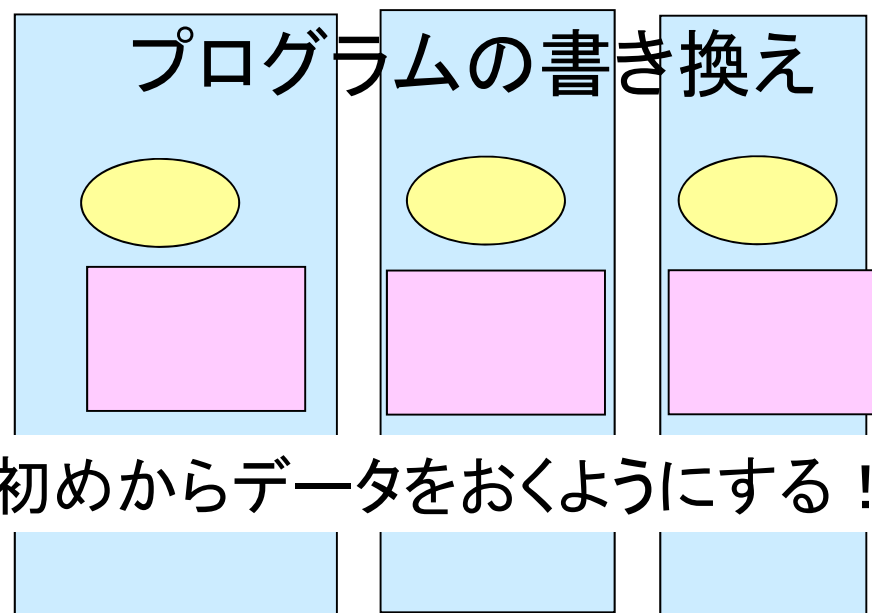
■ ベクトルプロセッサ

- あるループを依存関係がなくなるように記述
- ローカルですむ
- 高速化は数倍



■ 並列化

- 計算の分割だけでなく、通信（データの配置）が本質的
- データの移動が少なくなるようにプログラムを配置
- ライブラリ的なアプローチが取りにくい
- 高速化は数千倍～数万

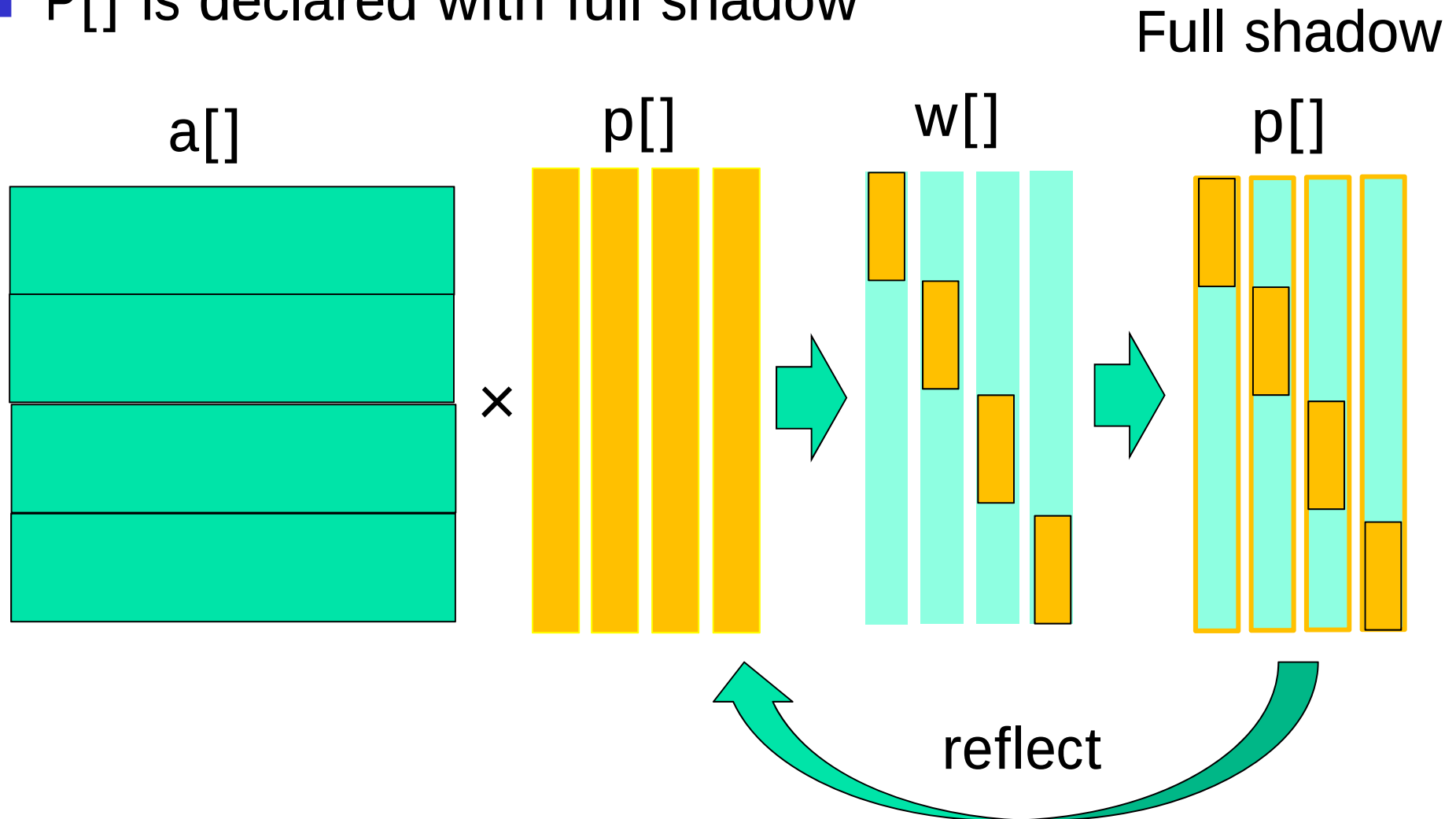


並列化と並列プログラミング

- 理想：自動並列コンパイラがあればいいのだが、...
- 「並列化」と並列プログラミングは違う！
- なぜ、並列プログラミングが必要か
 - ベクトル行列積を例に、...

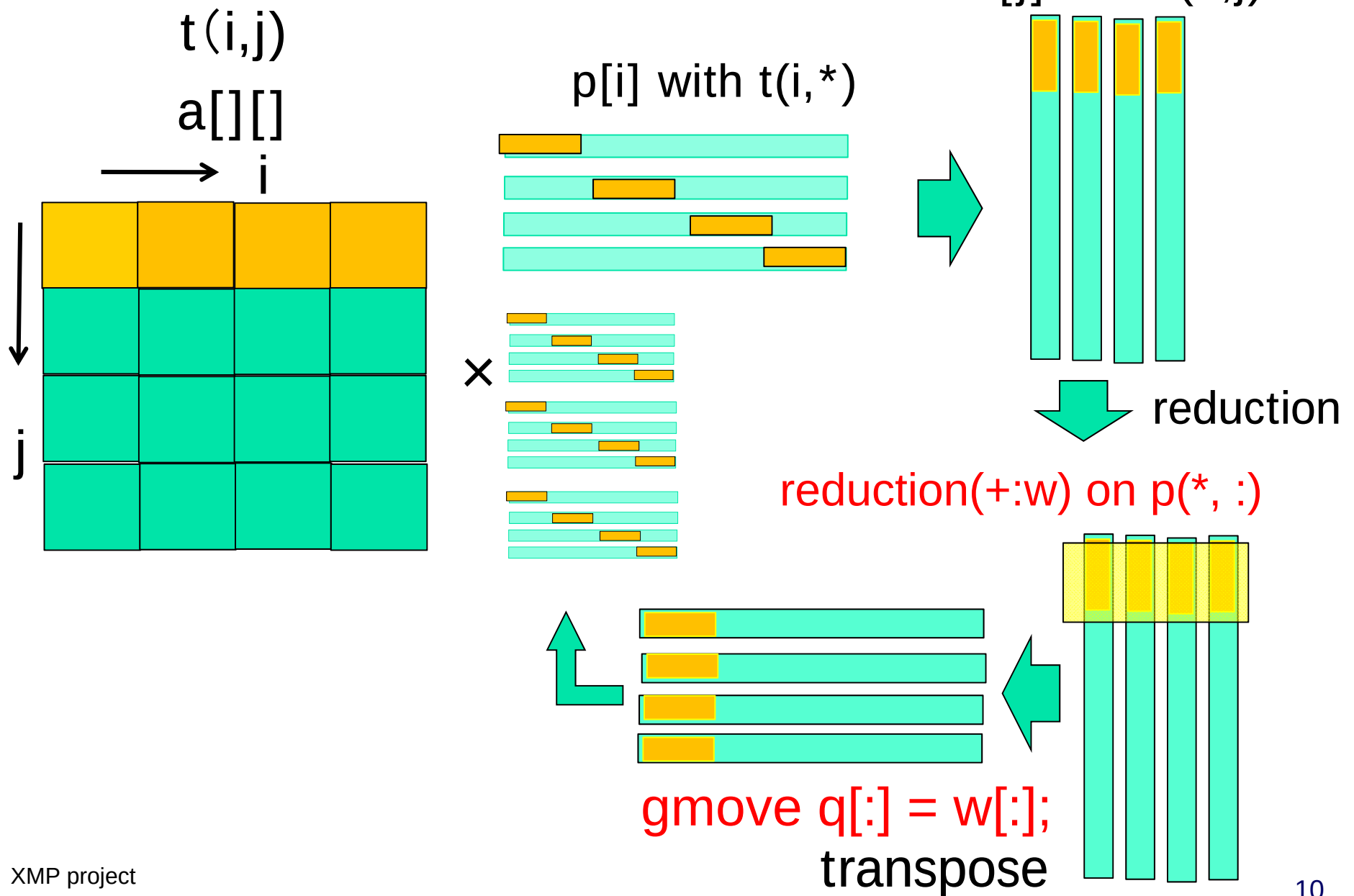
1次元並列化

- P[] is declared with full shadow



2次元並列化

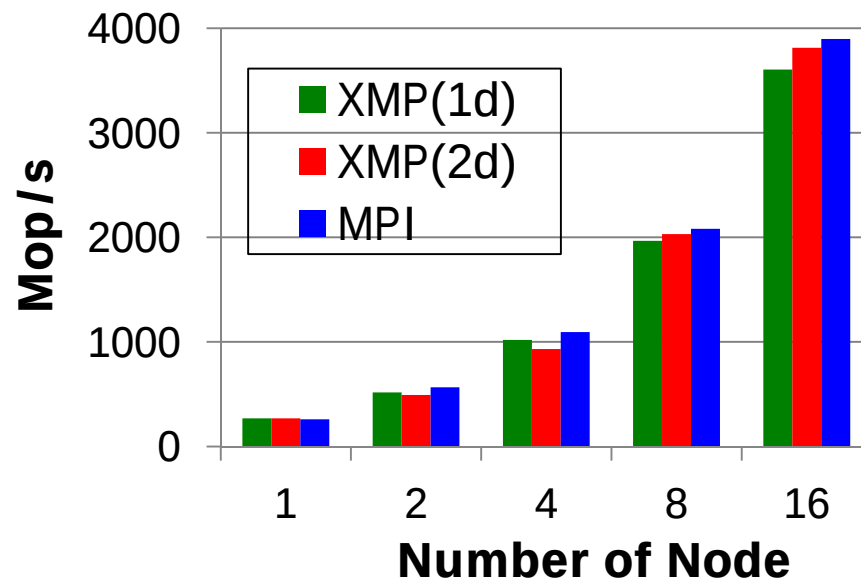
X_{calable}MP



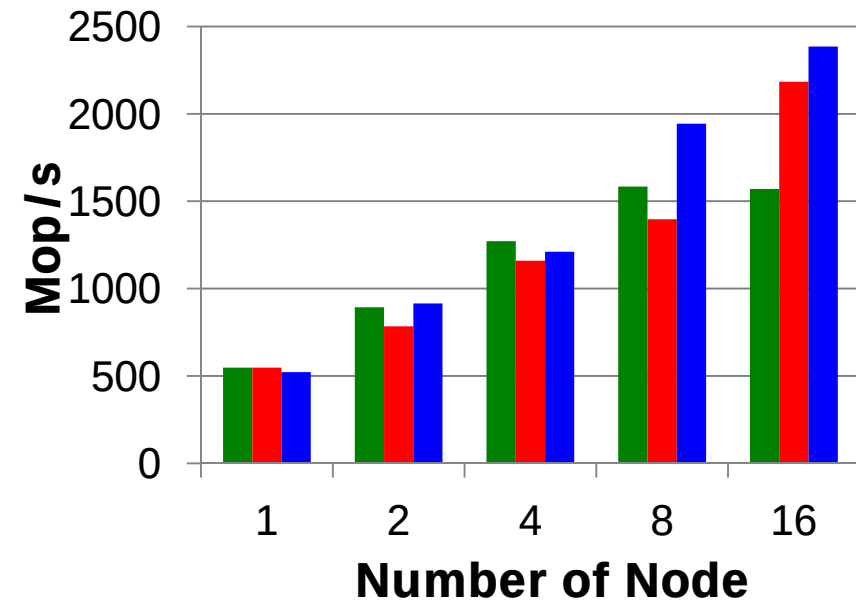
Performance Results : NPB-CG



T2K Tsukuba System



PC Cluster



The results for CG indicate that the performance of **2D. parallelization in XMP** is comparable to that of MPI.

■ HPFの教訓 (by 坂上@核融合研、村井@NEC)

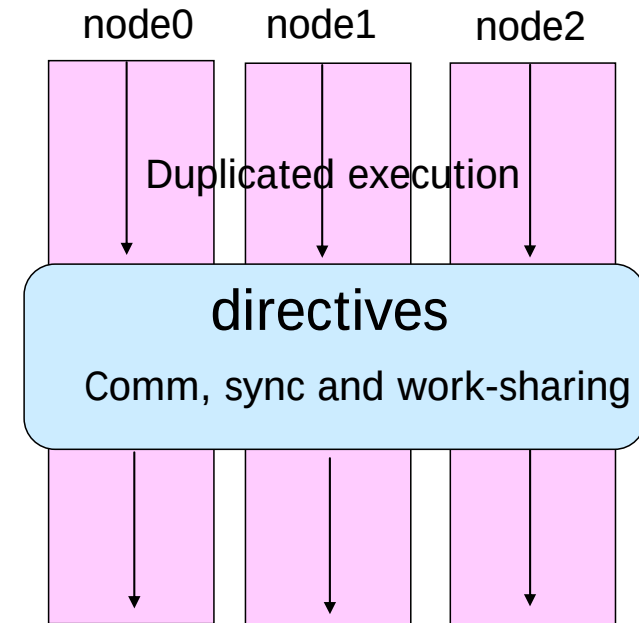
- 並列性とデータ分散を書いて、自動的に生成するという方針は理想的だったが、必ずしも性能は上がらなかった。期待が大きかった分、失望も大きかった
- ベース言語としたF90が未熟だった。Fortranだけだった。
- 必要な情報をユーザで指示文で補ってもらうという方針だったが、どこをどうすれば最適なコードになるかが明らかでなかった。
- 自動であるがために、通信がどこでおこっているのか、どうやってチューニングすればいいのか、ユーザに手段が与えられていなかった。
- 完全性を求めるあまり不必要な仕様があり、実装の障害になっていた。
- レファレンス実装が不在。教育が考慮されていない。

■ 90年代の並列プログラミング言語

- 多くはプログラミング言語の研究が主で、実際のアプリで使われることが少なかった。
- 組織的な普及活動、標準化、教育活動がない。

XcalableMP : directive-based language eXtension for Scalable and performance-tunable Parallel Programming

- **Directive-based language extensions** for familiar languages F90/C/C++
 - コードの書き換えや教育のコストを抑えること
- **“Scalable” for Distributed Memory Programming**
 - SPMDが基本的な実行モデル。
 - MPIのように、各ノードでスレッドが独立に実行を開始する。
 - 指示文(directive)がなければ、重複実行
 - タスク並列のためのMIMD実行も
- **“performance -aware” for explicit communication and synchronization.**
 - 指示文を実行するときに、Work-sharing や通信・同期がおきる。
 - すべての同期・通信操作は、指示文によって起きる。HPFと異なり、パフォーマンスのチューニングがわかりやすくなる。



XcalableMP Code Example (Fortran)

```
program xmp_example
Integer array(XMAX,MAX)
!$XMP nodes p(4)
!$XMP template t(YMAX)
!$XMP distribute t(block) onto p
!$XMP align array(*,j) with t(j)
```

data distribution

```
integer :: i, j, res
res = 0
```

add to the serial code : incremental parallelization

```
!$XMP loop on t(j) reduction(+:res)
do j = 1, 10
  do i = 1, 10
    array(i,j) = func(i, j)
    res += array(i,j)
  enddo
enddo
```

work mapping and data synchronization

XcalableMP Code Example (C)



```
int array[YMAX][XMAX];
```

```
#pragma xmp nodes p(4)  
#pragma xmp template t(YMAX)  
#pragma xmp distribute t(block) on p  
#pragma xmp align array[i][*] with t(i)
```

data distribution

```
main(){  
  int i, j, res;  
  res = 0;
```

add to the serial code : incremental parallelization

```
#pragma xmp loop on t(i) reduction(+:res)  
  for(i = 0; i < 10; i++){  
    for(j = 0; j < 10; j++){  
      array[i][j] = func(i, j);  
      res += array[i][j];  
    }  
  }
```

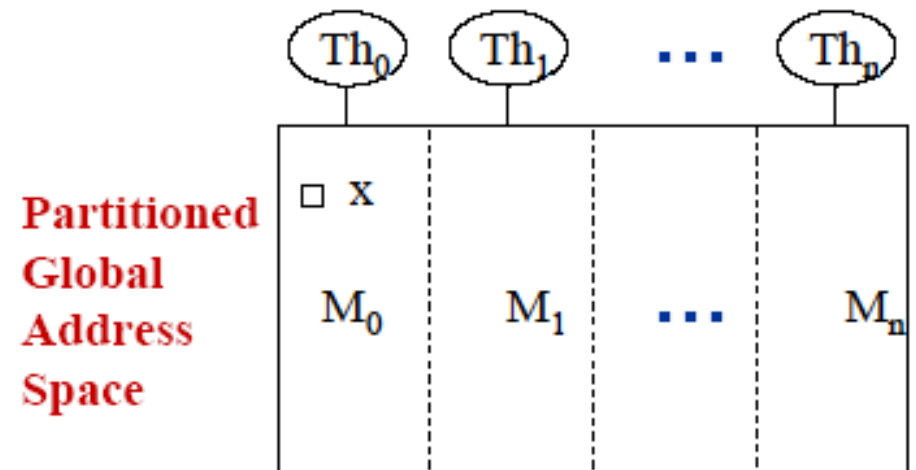
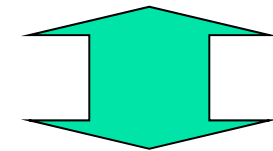
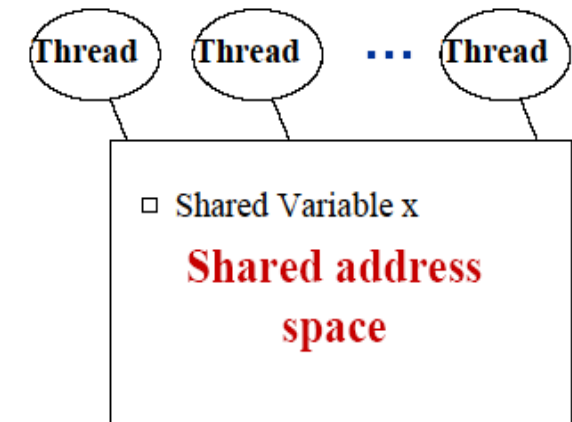
work mapping and data synchronization

XcalableMP as evolutionary approach

- 現状のコードからの移行を重視
 - 既存の言語C/Fortranに指示文を加えて並列プログラミング
- 過去の経験を取り入れる
 - 我が国のHPFの歴史
- コミュニティで言語を策定: e-science プロジェクトで組織。
- PGAS モデルをベースに言語を設計
- 難しいところはCoarray (from CAF) で。
- 演算加速機構への対応 XMP-dev 等、リサーチビークルとしても利用

Partitioned Global Address Space Model Programming

- ユーザがlocal/globalを宣言する(意識する)
- Partitioned Global Address Space (PGAS) model
- スレッドと分割されたメモリ空間は、対応づけられている(affinity)
 - 分散メモリモデルに対応
- 「shared/global」の発想は、いろいろなところから同時に出てきた。
 - Split-C
 - PC++
 - UPC
 - CAF: Co-Array Fortran
 - (EM-C for EM-4/EM-X)
 - (Global Array)



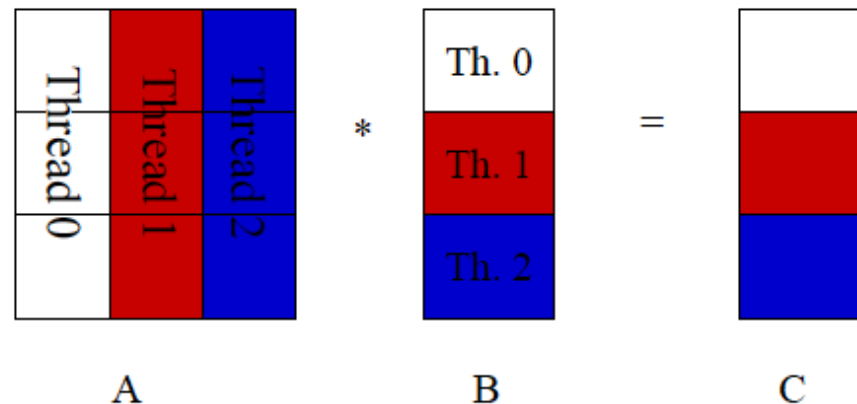
UPC: Unified Parallel C

- Unified Parallel C
 - Lawrence Berkeley National Lab.を中心に設計開発
- Private/Shared を宣言
- SPMD
 - MYTHREADが自分のスレッド番号
 - 同期機構
 - Barriers
 - Locks
 - Memory consistency control
- User's view
 - 分割されたshared spaceについて、複数のスレッドが動作する。
 - ただし、分割されたshared spaceはスレッドに対してaffinityを持つ。

行列積の例

```
#include <upc_relaxed.h>
shared int a[THREADS][THREADS];
shared int b[THREADS], c[THREADS];

void main (void) {
    int i, j;
    upc_forall(i=0;i<THREADS;i++;i){
        c[i] = 0;
        for (j=0; j<THREADS; j++)
            c[i] += a[i][j]*b[j];
    }
}
```



CAF: Co-Array Fortran

- Global address space programming model

- one-sided communication (GET/PUT)

- SPMD 実行を前提

- Co-array extension

- 各プロセッサで動くプログラムは、異なる”**image**”を持つ。

```
real, dimension(n)[*] :: x,y
```

```
x(:) = y(:)[q]
```

qのimageで動くyのデータをローカルなxにコピーする(get)

- プログラムは、パフォーマンス影響を与える要素に対して制御する。

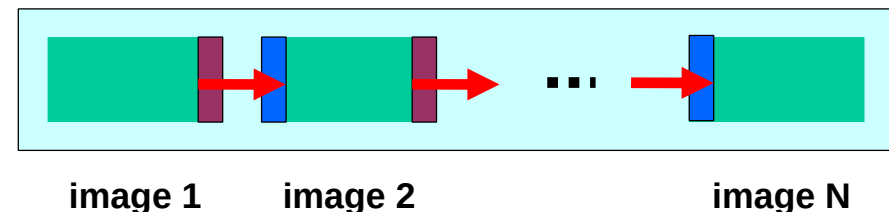
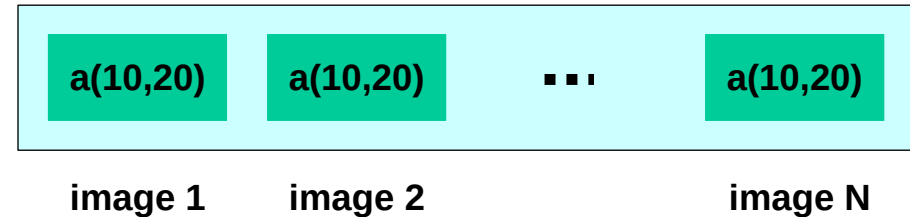
- データの分散配置
- 計算の分割
- 通信をする箇所

- データの転送と同期の

言語プリミティブをもっている。

- amenable to compiler-based communication optimization

```
integer a(10,20)[*]
```

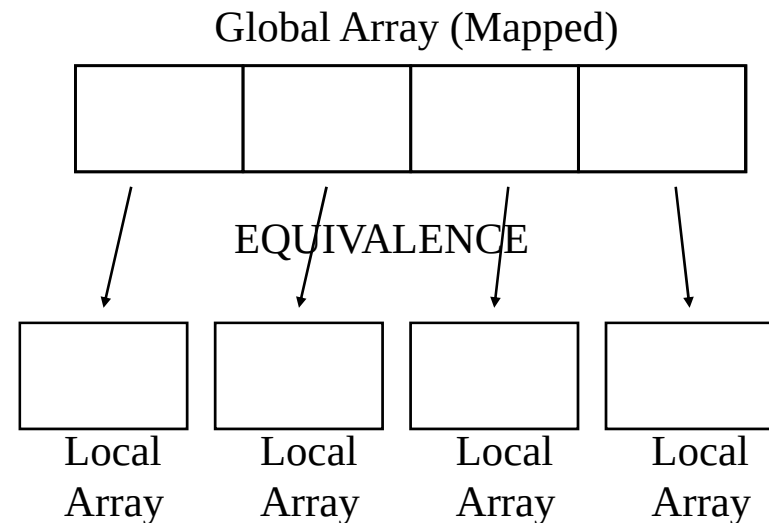


```
if (this_image() > 1)
  a(1:10,1:2) =
    a(1:10,19:20)[this_image()-1]
```

XPFortran (VPP Fortran)

- NWT (Numerical Wind Tunnel)向けに開発された言語、実績あり
- localとglobalの区別をする。
- インデックスのpartitionを指定、それを用いてデータ、計算ループの分割を指示
- 逐次との整合性のある程度保つことができる。言語拡張はない。

```
!XOCL PROCESSOR P(4)
  dimension a(400),b(400)
!XOCL INDEX PARTITION D=
  (P,INDEX=1:1000)
!XOCL GLOBAL a(/D(overlap=(1,1))), b(/D)
!XOCL PARALLEL REGION
!XOCL SPREAD DO REGIDENT(a,b) /D
  do i = 2, 399
    dif(i) = u(i+1) - 2*u(i) + u(i-1)
  end do
!XOCL END SPREAD
!XOCL END PARALLEL
```



PGAS言語の利点・欠点

- MPIとHPFの中間に位置する
 - わかり易いモデル
 - 比較的、プログラミングが簡単、MPIほど面倒ではない。
 - ユーザから見えるプログラミングモデル。通信、データの配置、計算の割り当てを制御できる
 - MPIなみのtuningもできる。
 - プログラムとしてpack/unpackをかいてもいい。
- 欠点
 - 並列のために言語を拡張しており、逐次には戻れない。(OpenMPのようにincrementalではない)
 - やはり、制御しなくてはならないか。
 - 性能は？

PGAS言語の現状・動向



■ UPC

- UCB/LBNLを中心にUPCコンソーシアムを組織して、開発普及活動を行っている
- Cray, IBM等のマシンで利用可能
- Research activityは高い。最近では、UPC task等の拡張も行っている。
- 標準化活動はいまいちか。

■ CAF

- Fortran 2008に規格として入ることになった。
- Crayのマシンでは使える。Intelもサポート。
- 研究活動としては、Rice University, University of Hustonで行われている。

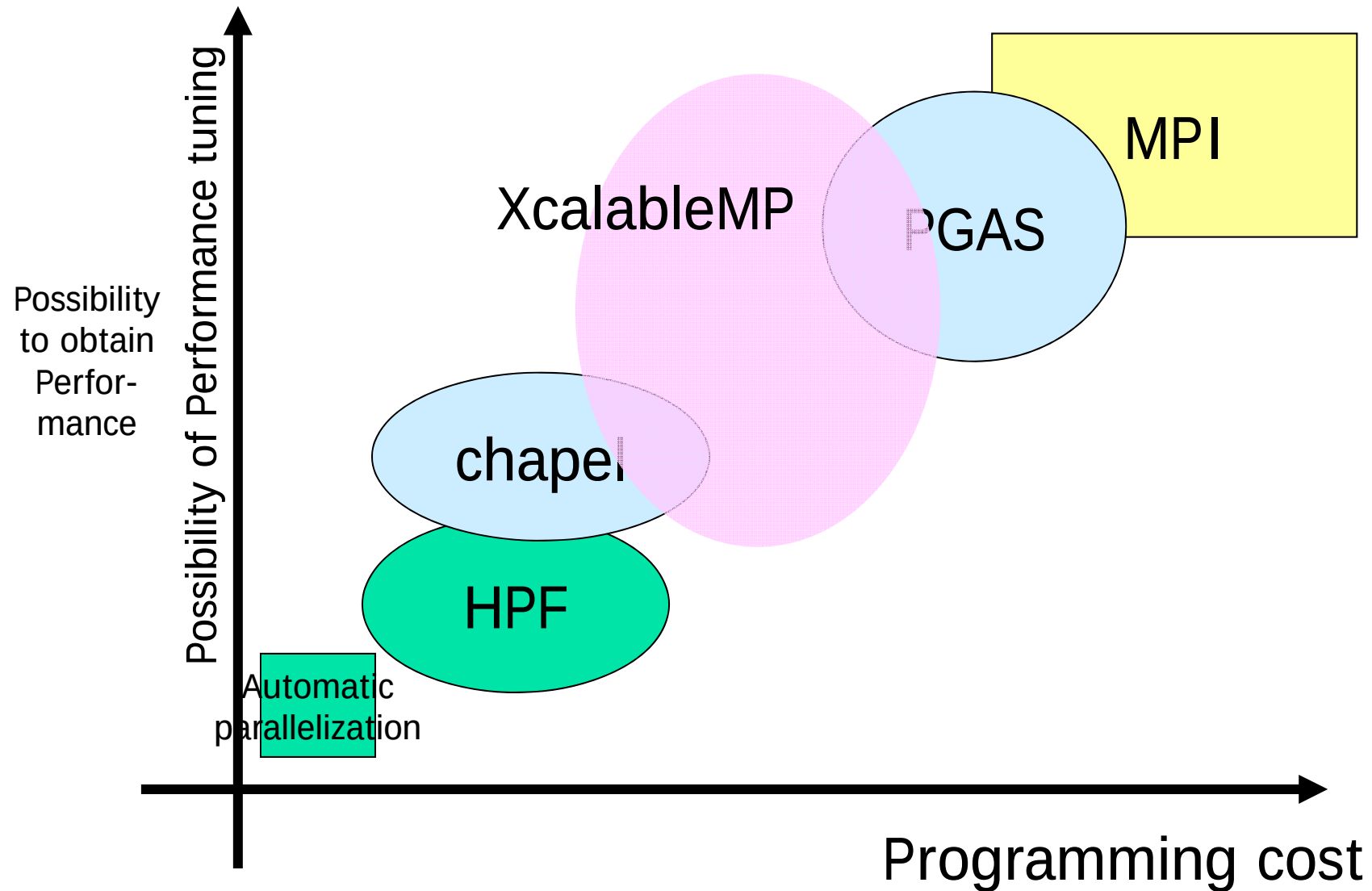
■ PGAS全体の動向

- 大規模化、RDMAハードウェアサポート(Cray XE6)など、注目は高まりつつある
- One-sidedはスケーラブルといわれている(が本当か?)
- SCでは、BOFとBooth(名前は?)で活動
- PGAS conferenceで、議論。
- 標準化とは? 低レベルの通信ライブラリの標準化? OpenShmem?

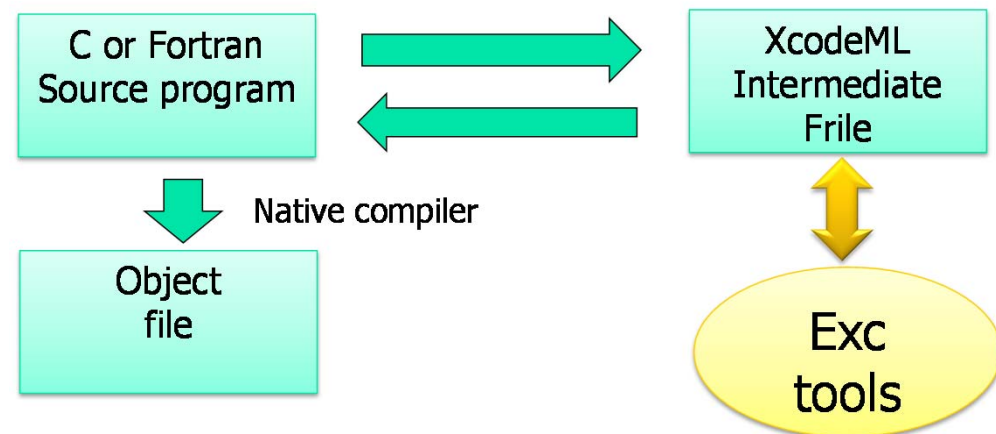
■ その他

- X10? Chapel?

Target area of XcalableMP



- XcalableMPのリファレンス実装
- 元々、OpenMPコンパイラを開発するために作ったもの
 - "Omni Compiler Software is a collection of programs and libraries, written in C and Java, that allow researchers to build code transformation systems (compilers). Omni OpenMP compiler is a part of this software that translates C and Fortran77 programs with OpenMP pragmas into C code suitable for compiling with a native compiler linked with the Omni OpenMP runtime library."
- 現在、以下の要素から構成されている
 - C フロントエンド (gcc対応)
 - F95 フロントエンド
 - XcodeML : フロントエンドのXMLによる中間フォーマット
 - XcodeMLの変換を行うためのjava ツールキット
 - C, F95のde-compiler
 - コンパイルドライバー



- XMLによるASTレベルの中間コード表現
 - human readable, well-defined
 - perl等、色々な言語で処理が可能
 - XcalableMP, OpenMPコンパイラでは、javaのツールキットで処理
- タイプ情報、シンボルテーブルなど、元のソースコードを復元できる情報を持つように設計

XcodeML 例 (1/3)



```
typedef struct complex {
    double real;
    double img;
} complex_t;

complex_t x;
complex_t complex_add(complex_t x, double y);

main()
{
    complex_t z;

    x.real = 1.0;
    x.img = 2.0;

    z = complex_add(x, 1.0);

    printf("z=(%f,%f)\n", z.real, z.img);
}

complex_t complex_add(complex_t x, double y)
{
    x.real += y;
    return x;
}
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<XcodeProgram source="t3.c">
<typeTable>
  <pointerType type="P0" ref="S0"/>
  <pointerType type="P1" ref="S0"/>
  <pointerType type="P2" ref="S0"/>
  <pointerType type="P3" ref="S0"/>
  <pointerType type="P4" ref="S0"/>
  <pointerType type="P5" ref="F0"/>
  <pointerType type="P6" is_restrict="1" ref="char"/>
  <pointerType type="P7" ref="F2"/>
  <structType type="S0">
    <symbols>
      <id type="double">
        <name>real</name>
      </id>
      <id type="double">
        <name>img</name>
      </id>
    </symbols>
  </structType>
  <functionType type="F0" return_type="S0">
    <params>
      <name type="S0">x</name>
      <name type="double">y</name>
    </params>
  </functionType>
  <functionType type="F1" return_type="int">
    <params/>
  </functionType>
  <functionType type="F2" return_type="int">
    <params/>
  </functionType>
  <functionType type="F3" return_type="S0">
    <params>
      <name type="S0">x</name>
```

XcodeML例 (2/3)



```
<functionType type="F3" return_type="S0">
  <params>
    <name type="S0">x</name>
    <name type="double">y</name>
  </params>
</functionType>
</typeTable>
<globalSymbols>
  <id type="F0" sclass="extern_def">
    <name>complex_add</name>
  </id>
  <id type="S0" sclass="extern_def">
    <name>x</name>
  </id>
  <id type="F1" sclass="extern_def">
    <name>main</name>
  </id>
  <id type="F2" sclass="extern_def">
    <name>printf</name>
  </id>
  <id type="S0" sclass="typedef_name">
    <name>complex_t</name>
  </id>
  <id type="S0" sclass="tagname">
    <name>complex</name>
  </id>
</globalSymbols>
```

```
<globalDeclarations>
  <varDecl>
    <name>x</name>
  </varDecl>
  <funcDecl>
    <name>complex_add</name>
  </funcDecl>
  <functionDefinition>
    <name>main</name>
    <symbols>
      <id type="S0" sclass="auto">
        <name>z</name>
      </id>
    </symbols>
    <params/>
    <body>
      <compoundStatement>
        <symbols>
          <id type="S0" sclass="auto">
            <name>z</name>
          </id>
        </symbols>
        <declarations>
          <varDecl>
            <name>z</name>
          </varDecl>
        </declarations>
        <body>
          <exprStatement>
            <assignExpr type="double">
              <memberRef type="double" member="real">
                <varAddr type="P0" scope="local">x</varAddr>
              </memberRef>
              <floatConstant type="double">1.0</floatConstant>
            </assignExpr>
```

XcodeML 例 (3/3)



```
<exprStatement>
  <assignExpr type="double">
    <memberRef type="double" member="img">
      <varAddr type="P1" scope="local">x</varAddr>
    </memberRef>
    <floatConstant type="double">2.0</floatConstant>
  </assignExpr>
</exprStatement>
<exprStatement>
  <assignExpr type="S0">
    <Var type="S0" scope="local">z</Var>
    <functionCall type="S0">
      <function>
        <funcAddr type="P5">complex_add</funcAddr>
      </function>
      <arguments>
        <Var type="S0" scope="local">x</Var>
        <floatConstant type="double">1.0</floatConstant>
      </arguments>
    </functionCall>
  </assignExpr>
</exprStatement>
<exprStatement>
  <functionCall type="int">
    <function>
      <funcAddr type="F2">printf</funcAddr>
    </function>
    <arguments>
      <stringConstant>z=(%f,%f)¥n</stringConstant>
      <memberRef type="double" member="real">
        <varAddr type="P2" scope="local">z</varAddr>
      </memberRef>
      <memberRef type="double" member="img">
        <varAddr type="P3" scope="local">z</varAddr>
      </memberRef>
    </arguments>
  </functionCall>
</exprStatement>
```

部会の設置の経緯



- e-scienceプロジェクトは2011年度で終了
- その時点の成果として、Version 1.0を決定・公開
- 開発は、筑波大とAICSで継続
 - 「京」@AICSおよび情報基盤センターへの展開
- 普及・今後の展開のために、PCクラスタコンソーシアムに部外を設置

- 並列プログラミング言語XcalableMP(XMP)の仕様について、検討し、決定する。本部会では、アプリケーション開発者がさまざまな計算環境で並列プログラム言語を容易に利用することができるように、プログラミング言語に要求される機能を検討するとともに、計算機メーカ及びユーザ・コミュニティが合意できる標準的な言語仕様を定めることを目的とする。これにより、並列プログラミングの生産性を向上させるとともに、コミュニティからの支援により、さまざまなプラットフォームにおいて利用できる言語の普及を目指し活動を行う。
- 活動内容
 - ①並列プログラミング言語XcalableMPの言語仕様の検討・決定
 - ②並列プログラミング言語XcalableMPの普及活動(講習会、ワークショップ、コンテストの開催等)
 - ③PCクラスタ向け並列プログラミング言語の動向の調査

組織



- 部会長: 佐藤三久 (筑波大学・理化学研究所AICS)
- 副部会長: 岩下英俊 (富士通(株))、林 康晴(日本電気(株))
- 部会員: 並列言語の開発に関心のある会社、並列言語を利用に関心のあるHPC研究者および会社、並列言語・プログラミングに関する研究者
- 参加資格
 - 最終的な規格決定についての投票権は正会員に限る(但し、部会に一定回数参加していること)
 - 規格について検討を行う部会や技術討論会の参加者については、会員であるかどうかは問わない。
- 活動期間
 - 3年後に見直しを行う。

これまでの活動



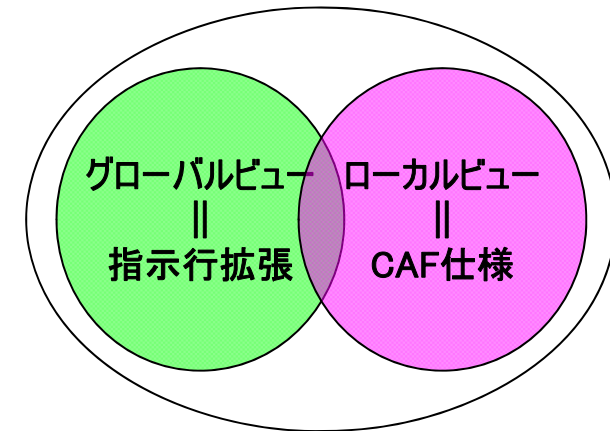
- ▶ 2011年11月、V1.0仕様を公開
 - ▶ 文科省e-Scienceプロジェクトの一環として
- ▶ 2011年6月、XMP規格部会発足
WG開催は、2011年12月第1回 ~ 2012年12月第9回
 - ▶ PCCC会員以外の参加も自由。ただし投票権は正会員のみ
 - ▶ 開催通知案内をご希望の方は、事務局(pccc-sec@pccluster.org)まで
 - ▶ V1.0仕様の修正と拡張を議論
 - ▶ 影舞による進捗管理 
- ▶ 2012年11月、V1.1仕様を公開 
 - ▶ V1.0からの主な拡張
 - ▶ Coarray機能対応 XMPによる並列化との混在の意味付け
 - ▶ プライマリノード配列 物理ノードのトポロジへの対応
 - ▶ マッピング問合せ関数 数値計算libやMPIとのインタフェース
 - ▶ 並列I/O仕様の詰め アトミック性などについての議論

Coarray機能対応：V1.1仕様では



- ▶ Coarray機能は、XMPに必要
 - ▶ Fortranでは今後標準。無視できない。
 - ▶ ローカルビューであり、XMPの指示行拡張とは相補的に使えると期待
- ▶ 議論の末、解決した課題
 - ▶ Fortran2008仕様との互換性保障
 - ▶ Coarrayで書かれた関数・サブルーチンは、そのままXMPから呼出し可能
 - ▶ Coarrayの「イメージ」とXMPの「ノード」との対応付けを定義
 - ▶ XMPプログラム中に、coarray記法が混在可能
 - ▶ XMP/Cでも、同じ機能をサポート

XMP V1.1言語仕様



XMP/Cでのcoarray記述例

```
float a[100]:[*], b[100];  
  
me = xmp_node_num();  
  
if (me>0) b[0] = a[99]:[me-1];  
xmp_sync_all(&stat);
```

並列ライブラリインタフェース

- すべてをXMPで書くことは現実的ではない。他のプログラミングモデルとのインタフェースが重要
- MPIをXMPから呼び出すインタフェース
- (MPIからXMPを呼び出すインタフェース)
- XMPから、MPIで記述された並列ライブラリを呼び出す方法
 - ScalapackやMUMPSを対象
 - XMPの分散配列記述から、Scalapackのディスクリプタを作る
 - XMPで配列を設定、ライブラリを呼び出す
 - その場合、直によびだすか、wrapperをつくるか。

■ Design

- Provide efficient IO for global distributed arrays directly from lang.
- Mapping to MPI-IO for efficiency
- Provide compatible IO mode for sequential program exec.

■ IO modes

- (native local IO)
- Global collective IO (for global distributed arrays)
- Global atomic IO
- Single IO to compatible IO to seq. exec

今年度の活動(1/2)



- 2013年4月以降
 - 規格部会開催(第12回:4/23, 第13回:6/7, 第14回:9/30)
 - 配列処理関数, 不規則問題などについて議論
 - 11月に仕様書v1.2を公開予定
 - 第3回XMP Challengeの開催(2013年2月 - 4月)
 - XMP利用講習会の開催(7/18, 9/13, 12/4)
 - 計算科学振興財団(FOCUS)の講習会
 - http://www.j-focus.jp/event_seminar/entry-394.html
 - 7月と9月は初級編. ハンズオンで講義を行う
 - 12月は中級編の予定
 - XMPワークショップ@秋葉原UDX(11/1の予定)



■ マルチコア対応

■ 現状

- ほとんどのクラスタがいまや、マルチコアノード(SMPノード)
- 小規模では格コアにMPIを走らせるflat MPIでいいが、大規模ではMPI数を減らすためにOpenMPとのハイブリッドになっている。
 - ハイブリッドにすると(時には)性能向上も。メモリ節約も。
- しかし、ハイブリッドはプログラミングのコストが高い。

■ XMPとOpenMPをhybridに記述、その規則を定義

■ collective communicationのライブラリを定義

- sin,cosなどの算術関数の要素並列
- transposeなどの形状変換関数

リファレンス実装について



- Omni XMP Compiler v0.6.1をリリース(2013年3月)
 - Coarray機能のストライド通信の対応
 - 京コンピュータにおけるCoarray機能のサポート
 - <http://www.hpcs.cs.tsukuba.ac.jp/omni-compiler/xcalablemp/>
- 2013年11月にOmni XMP Compiler v0.7をリリース予定
 - OpenMP指示文やOpenACC指示文との混在利用
 - 京コンピュータにおけるハードウェアサポートを利用した実装
 - XMP/Fortranの片側通信機能のサポート

などを予定しています。



HOME
Download
Document
Publication
Mailing List
Links

Omni XcalableMP Compiler

ABOUT

The Omni XcalableMP Compiler is a reference implementation of [XcalableMP](#).
This compiler has been developed by [University of Tsukuba HPCS Lab.](#) and [RIKEN AICS Programming Environment Research Team](#).

NEWS

- ▶ 2012.11.30 : Nightly build version is released.
- ▶ 2012.11.29 : This site is open.

今年度の活動(2/2)



- SC13@Denverでブース展示(11/18 - 21)
- HPC Challenge Class 2に応募
 - HPCシステムの性能を評価するためのベンチマーク集
 - HPL, FFT, RandomAccess, Stream
 - Class 1は性能のみを評価
 - Class 2は生産性(実装のエレガントさ)と性能の2つを統合的に評価
 - XMPでHPC Challengeベンチマークを実装し, 京コンピュータ上で計測
 - SC13のBoFで発表予定
 - SC13後にXMPのHPC Challengeベンチマークを公開予定

現在の(研究)活動



- フランスでのチュートリアル COMPAR@グルノーブル
 - フランスとのYMLとの統合プロジェクト(日仏戦略FP3Cプロジェクト)
- NEC SXやIBM BG/Qなどへの移植 (AICSの活動の一部)
 - そのうちに、調達の「仕様書」にいらてもらえるように
- XMPを使った並列プログラムのチュートリアルなどの人材育成活動(AICS)
- GPUのサポート、XMP-devはあるが、OpenACCとの混在で再検討(朴CREST@筑波大、AICS)
- 研究
 - Xeon Phiでの性能評価(筑波大)
 - XMP向けのshared memoryモデル、one-sided通信(筑波大)
 - 有限要素法などの不規則メッシュのための拡張(AICS)
 - XMP上の動的なタスク生成(AICS)

これからのプログラミングモデル

エクサスケールコンピューティングに向けて

エクサ時代のプログラミングモデル
～性能か、生産性か?～

SS研パネルより

並列プログラミング言語・モデルの研究・開発・普及

- まず、エクサに限った話ではない。今でも、十分に問題。
- 実用的な言語・モデルが一般に普及するには、(成功したとして)10年かかる。
 - OpenMPも10年、それにきっかけが必要
- アプリが言語で書かれるためには、標準化(デファクトでもいい)が必要
 - どこでも動くという保証が必要
 - 多くのプラットフォームで使えることが必要
- 普及には、教育が必要
 - 新しい言語の普及へのハードルは高い

各報告書から

■ IESP roadmap

- Programming models: Research is needed into a variety of promising programming models for exascale computing, including system-wide models that provide a uniform approach to application development across an entire platform, as well as hybrid programming models that combine two or more programming APIs. Such models will need to provide a range of means for the expression of high levels of concurrency and locality, and may be capable of supporting application-specific fault tolerance. Both enhancements to existing programming interfaces as well as new programming approaches should be explored. For new models, interoperability with existing HPC programming interfaces is highly desirable. Programming models that facilitate productive application development are to be encouraged. Other desirable characteristics are performance transparency and the ability to support incremental application migration.
- Compilers: With substantial support from the runtime library, they may also be required to support the balancing of the workload across the system components. Compilers for node programming models may be required to generate code that runs across a large collection of general-purpose cores, or across a node that may be configured with general-purpose cores along with one or more specialized accelerators.
- Compilers: Memory hierarchies will be highly complex; memory will be distributed across the nodes of exascale systems and will be NUMA within the individual nodes, with many levels of cache and possibly scratchpad memory.
- Compilers: [Alternatives R&D] ... By ensuring interoperability between different languages and programming models, compilers can be key to mitigating the risk involved in selecting an emerging programming model and may increase the adoption of new models by offering an incremental path from existing or proposed models (UPC, CAF ...)
- それほど、びっくりしたことが書いてあるわけではないが、当然のことを書いてある

各報告書から

■ EESI

- [Software EcoSystem W4.2] Programming models that provide the right balance between expressiveness, portability, performance and transition path for applications
- [Software EcoSystem W4.2] Unified runtime system, providing a unified API to deal with threads and lightweight tasks (together with their integration with MPI/PGAS communication systems)
- [Scientific Software Engineering W4.4] Development of Domain Specific Languages (DSL) targeting specific application areas, e.g. climate, engineering, materials. High-level abstraction enables application development to be protected and insulated from architectural issues at the Exascale
- [Scientific Software Engineering W4.4] Integration of coupling and workflow technologies with the development of efficient and generic coupling/workflow software for Exascale architectures
- 体制的な話が多い
- DSLは、Scientific Software Engineering に位置づけ

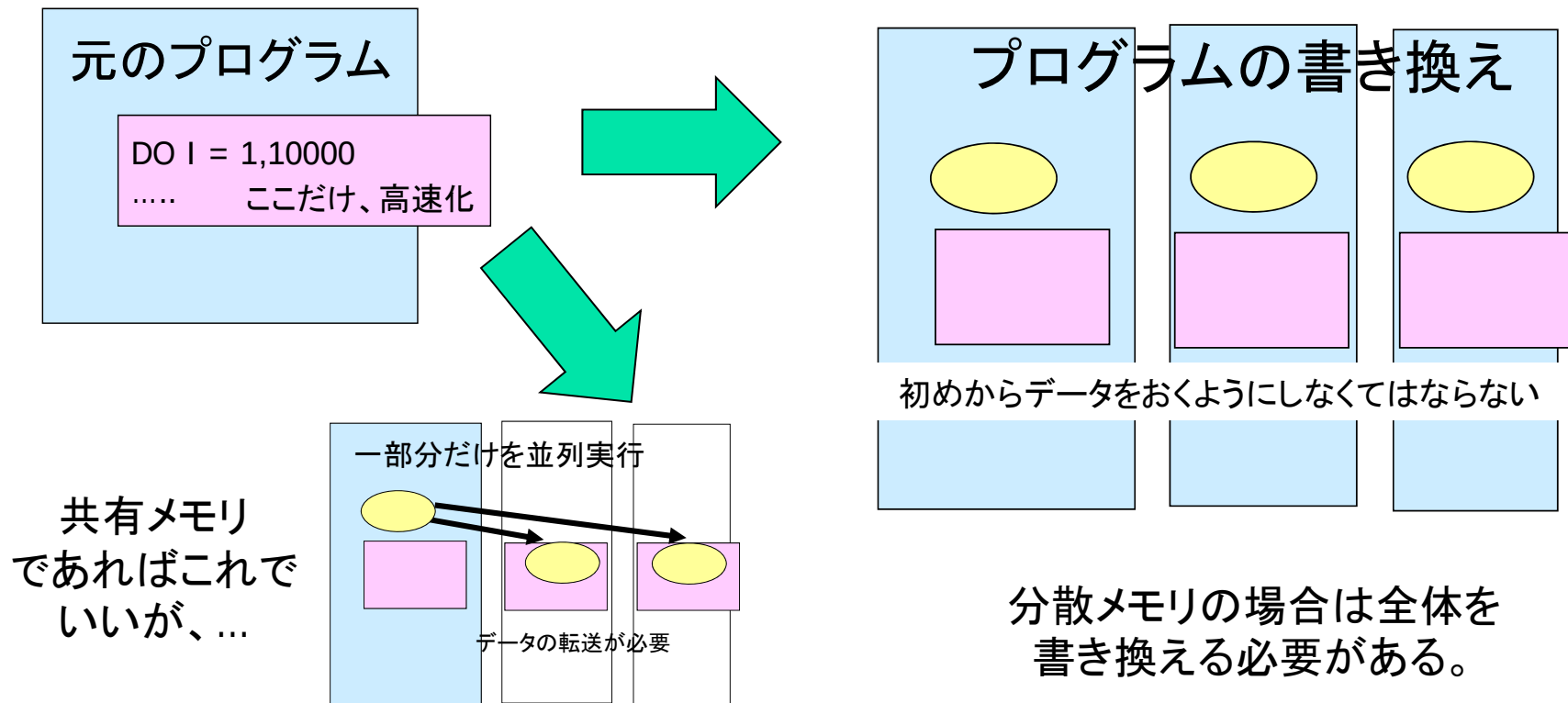
■ SDHPC

- (アプローチ1) 領域固有、アルゴリズム固有な言語(Domain Specific Language; DSL)の設計
- (アプローチ2) 汎用性を持った基盤言語の設計
- Revolution vs. Evolution

並列プログラミングは何が難しいか

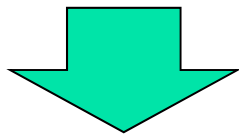
- All about memory! キーポイントは、メモリモデル
- 第一の波： 逐次プログラム ⇒ ベクトルプロセッサ
 - 並列性記述、データのレイアウトを変える
- 第2の波： ベクトルプロセッサ ⇒ 分散メモリ, MPP
 - MPI、プログラムの構造を変えることが必要
- 第3の波： 分散メモリ + 加速機構オフロード
 - 必要なデータをオフロード
 - 並列性は、kernelで書かなくてもOpenMP/OpenACC的なアプローチがある。

分散メモリと並列プログラミング

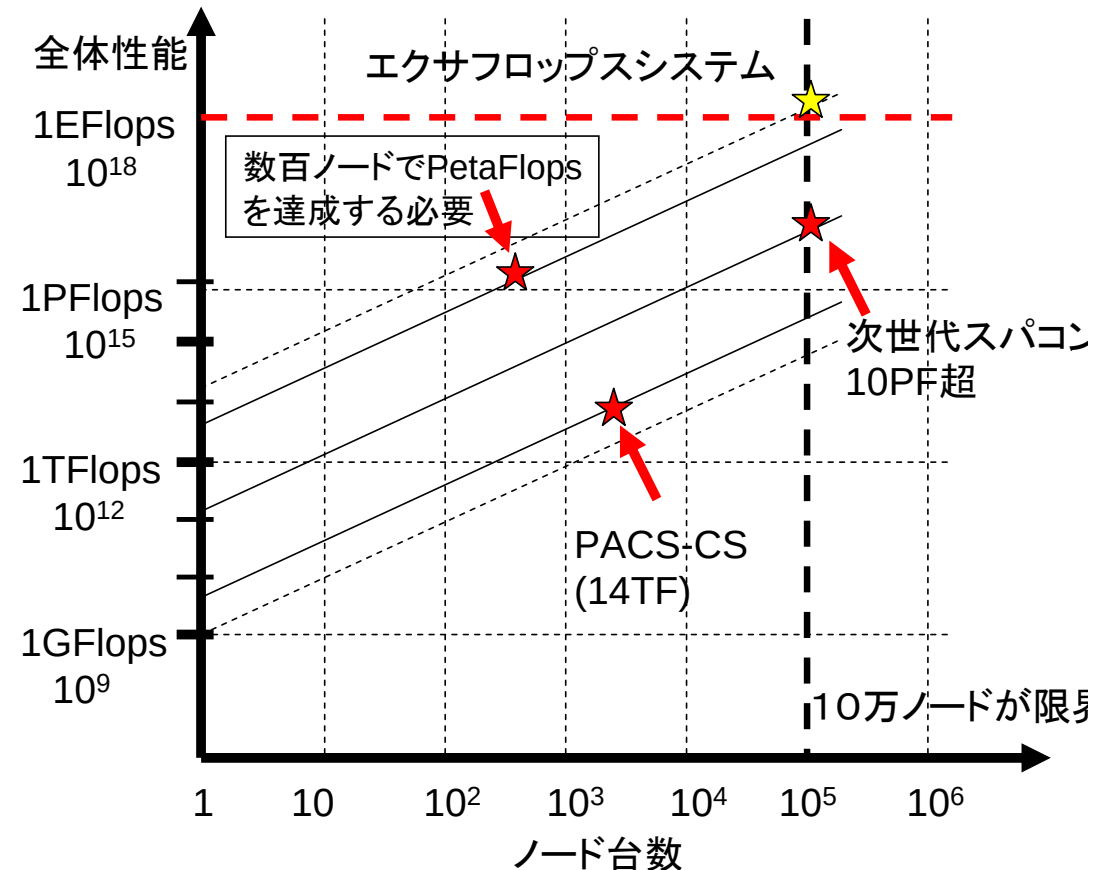


エクサスケールコンピューティングのアーキテクチャ

- 各国のペタスケールプロジェクト
 - コアは数百万
 - 電力は数十MW
- 結局、プロセッサ数を増やすことによって、weak-scalingで性能を上げてきた。
- これからは、Strong-scalingできるノードとそれに耐えるネットワークが必要



- 演算加速機構 (GPUとは限らない) が有望なソリューション
 - ただし、用途が限定されていく
 - メモリは別のことが多い



MPIで、いいのか？

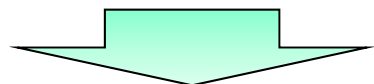
いつまで、MPIでプログラミングしなくてはならないのか？

What's a role of PGAS for exascale computing

- Exploiting locality is getting more important for exascale computing.
- Remote memory access facility (such as RDMA) supporting PGAS may be more scalable than MPI for several thousands of process.
- Global address space may increase productivity of complex parallel programs.
- In our project (HPCI-FS), the description of alignment of data in global address space (XMP) is useful for code generation for each PE in the PACS-G SIMD processor.

「演算加速機構を持つ将来のHPCIシステムに関する調査研究」

- ナノテクやライフサイエンスの進歩、気候気象予測や地震・防災への対処には計算科学は不可欠かつ有効な手段
 - そのためにはさらなる計算能力が要請されている。
 - 設置面積、消費電力等の制限からノード数の増加による並列システムの性能向上には限界
- ライフサイエンスの分子シミュレーション等、多様な分野で比較的小さい一定サイズの問題の高速化が望まれている(いわゆる強スケーリング)
 - 対応した研究開発の例: ANTON, MDGRAPE-4



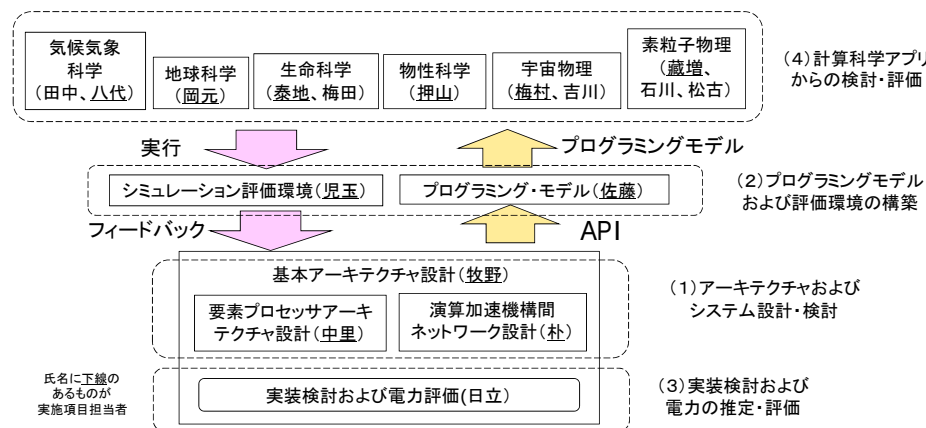
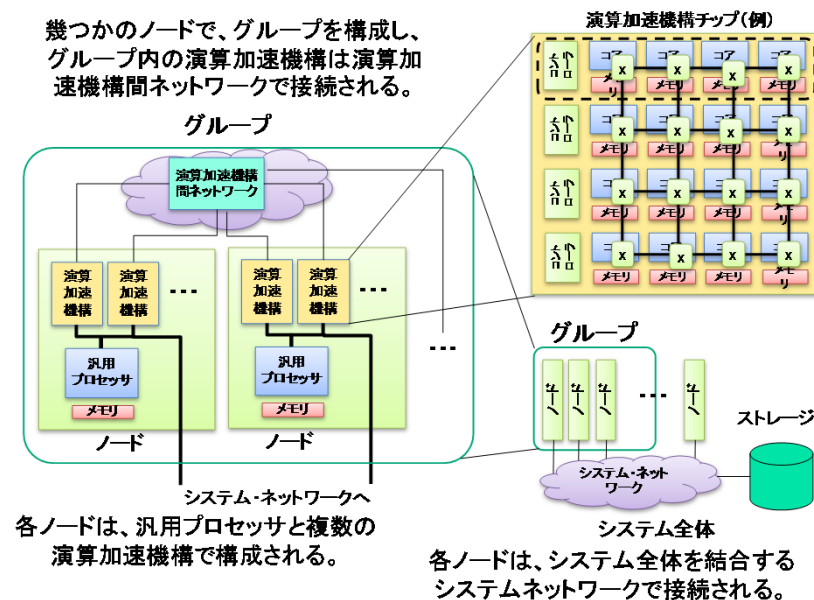
電力効率の大幅な効率化と強スケーリング問題の高速化による新たな計算科学の展開を目指して、演算加速機構による並列大規模システムについて調査研究を行う。

平成23年度文部科学省アプリケーション＆コンピュータアーキテクチャ・コンパイラ・システムソフトウェア合同作業部会において、まとめられた「今後のHPCI技術開発に関する報告書」の中で、分類されたシステム構成のうち「**メモリ容量削減**」および「**演算重視**」のシステムを主な調査研究の対象とする。

- 主管事業実施機関: 筑波大学 計算科学研究センター
- 共同事業参画機関: 東京工業大学、理化学研究所、会津大学、日立製作所
- 協力機関: 東京大学、広島大学、高エネルギー加速器研究機構
- 調査研究を、以下の4つの項目に分けて実施
 - (1)アーキテクチャおよびシステムの設計・検討
 - (2)プログラミング・モデルおよび評価環境の構築
 - (3)実装検討および電力の推定・評価
 - (4)計算科学アプリからの検討・評価

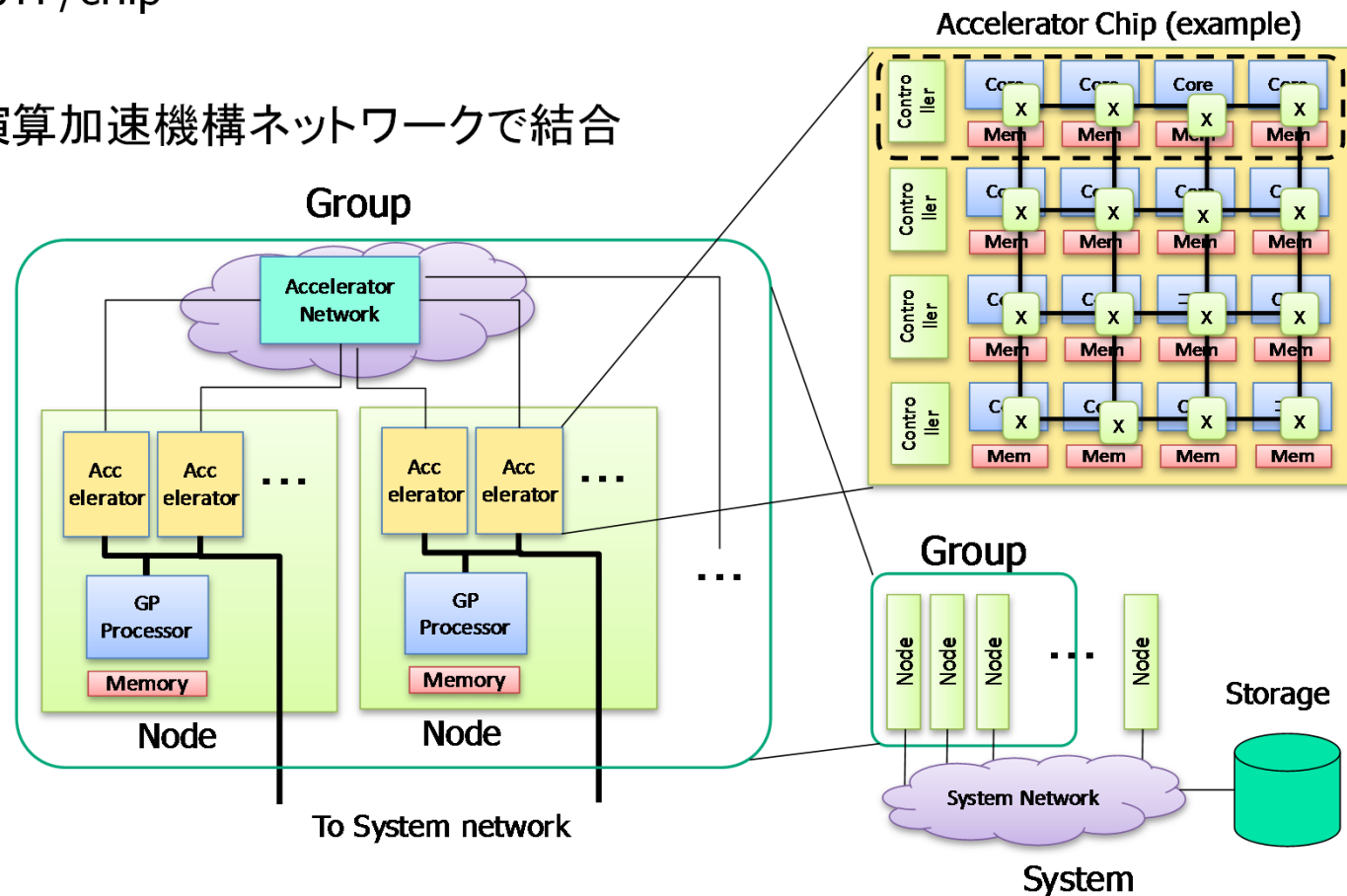
多数の演算コアを内蔵したチップによる演算加速機構が汎用プロセッサで構成された並列システムの各ノードに接続もしくは内蔵されているヘテロロジーニアスな並列システムを想定

演算加速機構は、多数のスループットコアにより構成。スループットコアは、チップ内ネットワークにより結合される。
図に示したものは一つの例。



PACS-G: a straw man architecture

- 以下のアーキテクチャを、**Straw man**(たたき台) アーキテクチャとして設定
- 演算集約型とメモリ削減型のステンシル計算を両立させるアーキテクチャ(プロセッサ、ネットワーク)をターゲットに設定
- チップの基本アーキテクチャは、大規模SIMD PEはローカルメモリで動作
- 目標は、4096PE 16TF/chip
- 2048 チップ/group,
Group内のチップは演算加速機構ネットワークで結合



PACS-G: a straw man architecture

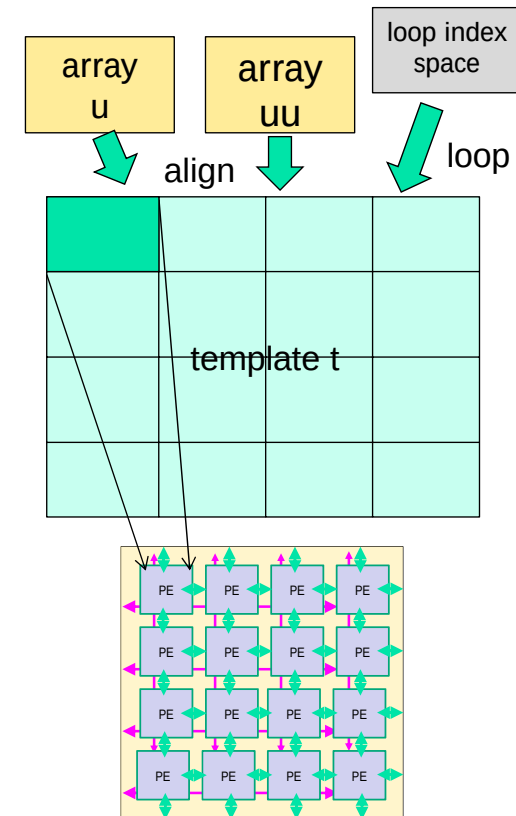
- Programming model: XcalableMP + OpenACC/OpenMP 4.0
 - Use OpenACC/OpenMP 4.0 to specify offloaded fragment of code and data movement
 - To align data and computation to core, we use the concept "template" of XcalableMP (virtual index space). We can generate code for each PE.
 - (And data parallel lang. like C*)

```
#pragma xmp template t(0:XSIZE+2, 0:YSIZE+2)

double u[XSIZE+2][YSIZE+2],uu[XSIZE+2][YSIZE+2];
#pragma xmp align u[i][j] with t(i,j)
#pragma xmp align uu[i][j] with t(i,j)

#pragma xmp loop on t(x,y)
  for(x = 1; x <= XSIZE; x++)
    for(y = 1; y <= YSIZE; y++)
      u[x][y] = (uu[x-1][y] + uu[x+1][y]
                + uu[x][y-1] + uu[x][y+1])/4.0;

sum = 0.0;
#pragma xmp loop on t(x,y) reduction(+:sum)
  for(x = 1; x <= XSIZE; x++)
    for(y = 1; y <= YSIZE; y++)
      sum += (uu[x][y]-u[x][y]);
```



Domain-specific Language(DSL), フレームワーク vs. 汎用言語

- DSL、フレームワークだけでなく、汎用のプログラミング言語・モデルも必要
 - 通信モデル、アーキテクチャのabstractionレイヤとして
- DSL、フレームワークがあったとしても、これからのエクサに対応できるかは別
 - 抽象度の高さは、利点だが。
- 誰が書くのか
 - DSLを作るためのDSL/フレームワーク
 - 結局、ドメインのコミュニティの問題か？

ライブラリ

- ・ある分野の有用かつ汎用的なサブルーチン群
- ・アプリケーションの構築法には立ち入らない
- ・例: 行列計算ライブラリ, FFT ライブラリ

フレームワーク

- ・特定種のアプリに再利用できるサブルーチン群
- ・アプリケーションアーキテクチャを形成する
- ・例: 流体計算フレームワーク

エクサ時代のプログラミングモデル ～性能か、生産性か?～

- 当然書きにくくなる ⇒ 生産性が低下
- メモリをちゃんと扱えるプログラミングモデルを！
(PGAS?) ⇒ 性能
- DSLで隠してしまえばいいが、...

ご清聴ありがとうございました。