

ペタスケールシステムへの 並列プログラミング言語の課題

～並列プログラミング言語検討会からの中間報告～

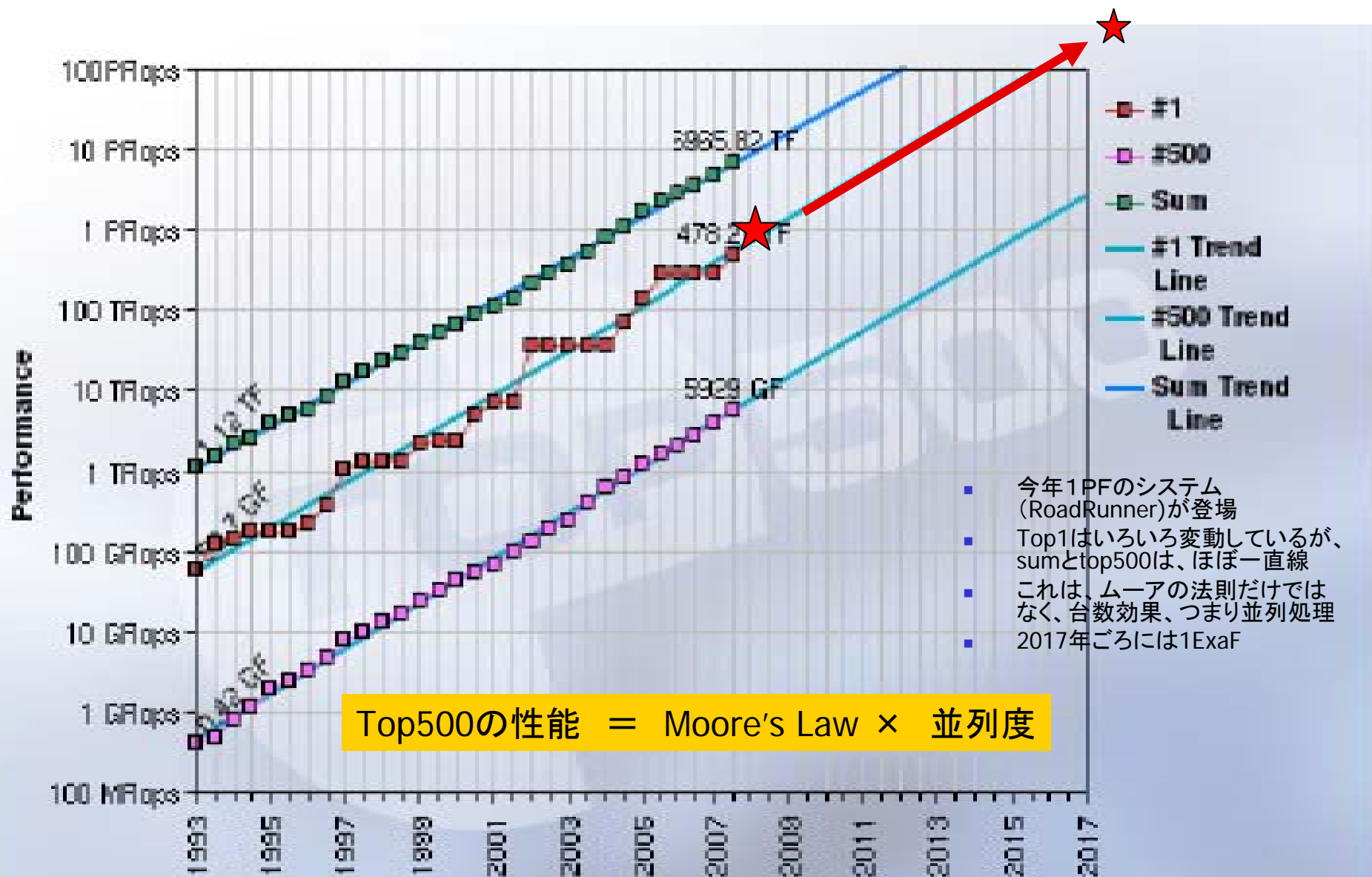
佐藤 三久

筑波大学 計算科学研究センター

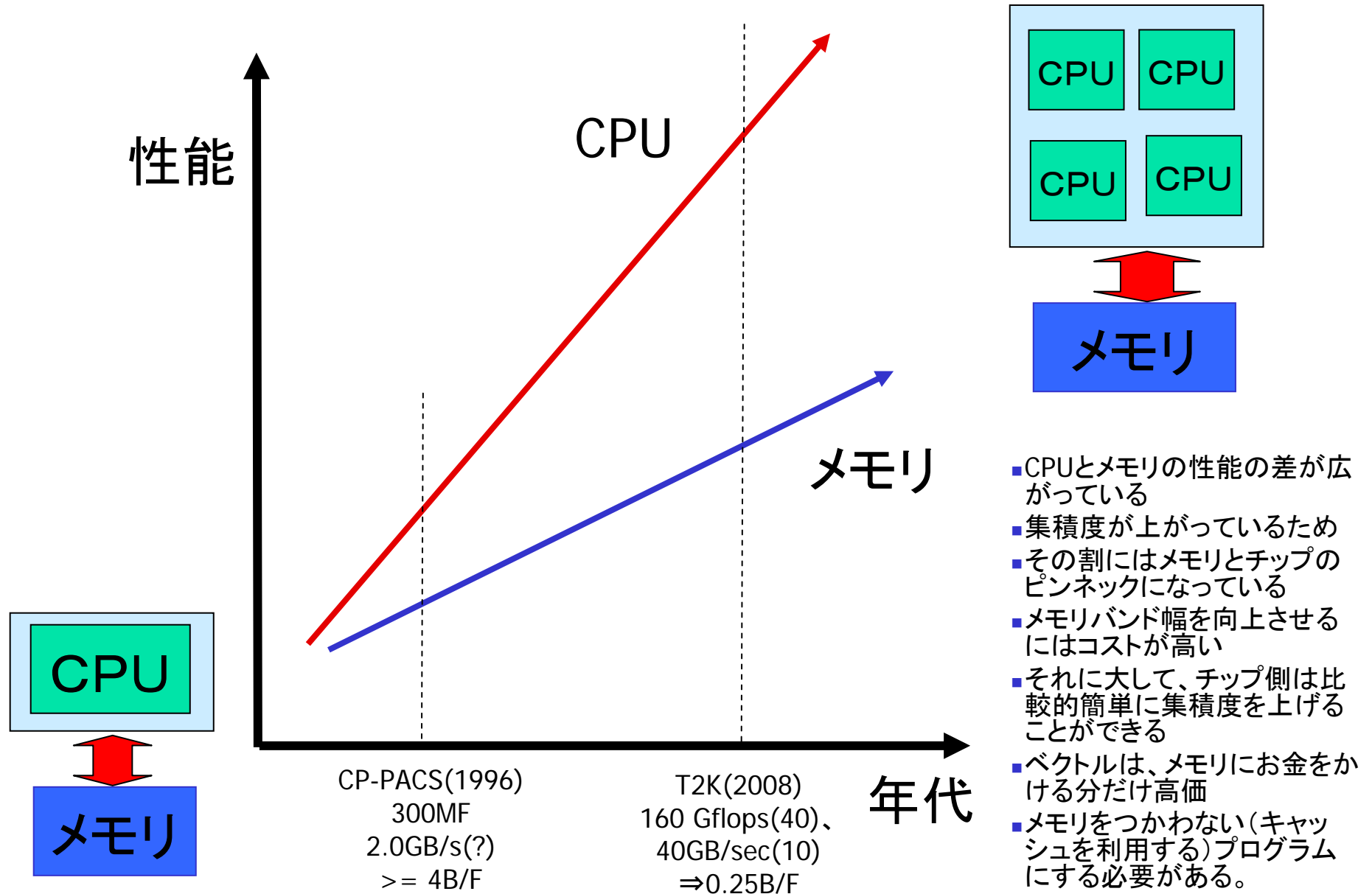
もくじ

- なぜ、並列化は必要なのか
- 現状の並列プログラミング言語について
 - (OpenMP), UPC, CAF, HPF, XPF, ...
- 並列プログラミング言語検討会
 - 動機、経緯
 - 検討状況
 - 計画

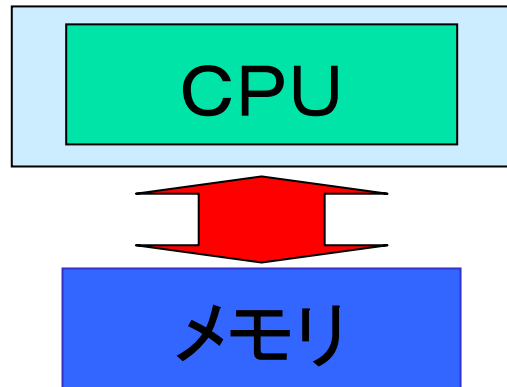
TOP500: 全世界のスパコンランキング500位



スパコン(高性能システム)のトレンド

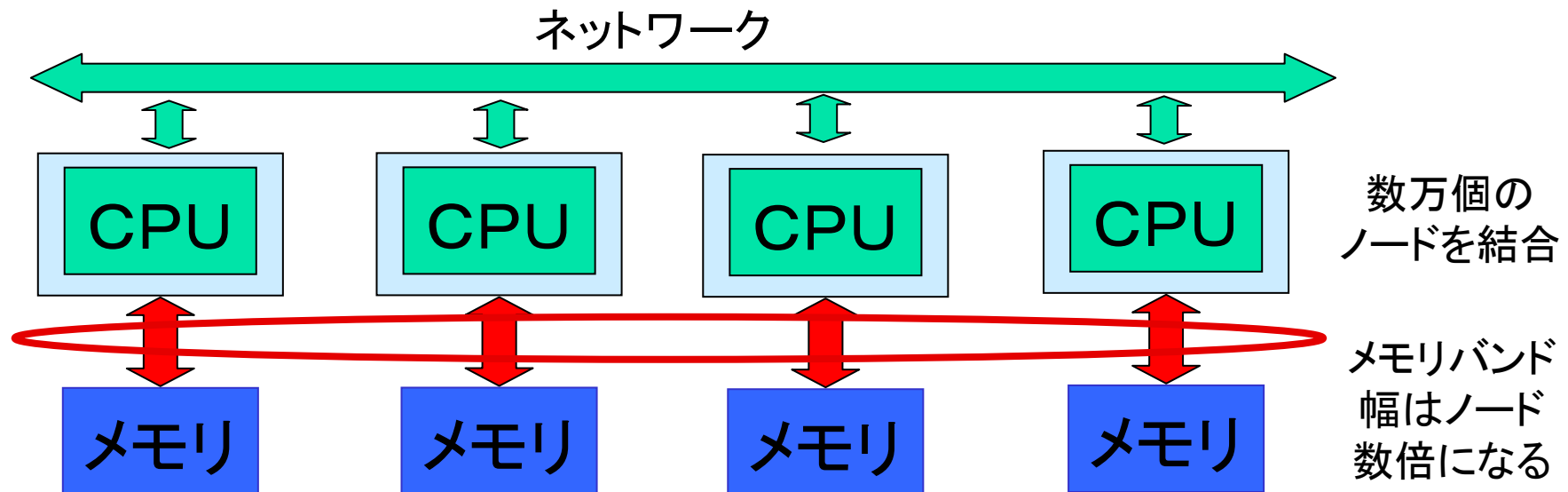


並列システムはなぜ有効か



ベクトルプロセッサで
CPUの性能を上げた
分だけ、メモリバンド
幅を上げる
⇒ しかし、高々数倍

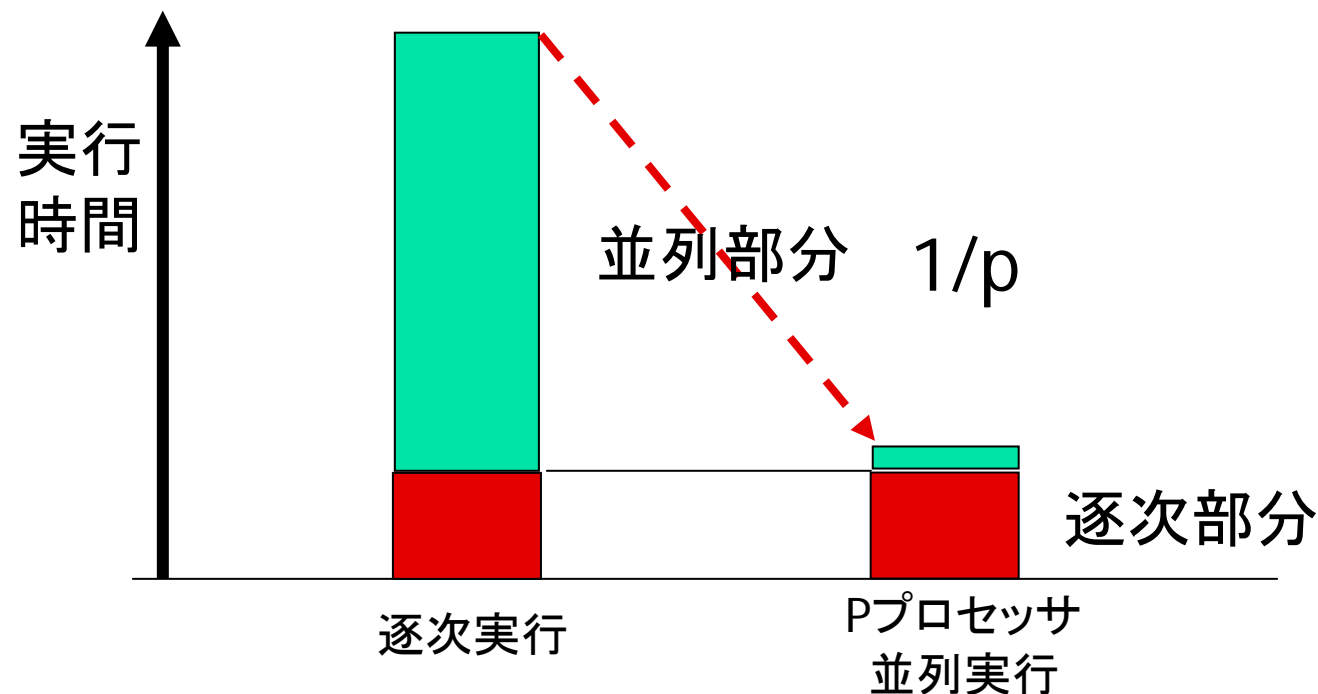
- 複数のノードを結合することにより、実行のメモリバンド幅 (aggregate) を増やすことができる。
- これからは、数万ノードの時代
- もちろん、ノードの性能を引き出すことも重要
- 安いCPUのテクノロジーを利用できる。
- ネットワークの性能が重要になる。



並列処理の問題点:「アムダールの法則」の呪縛

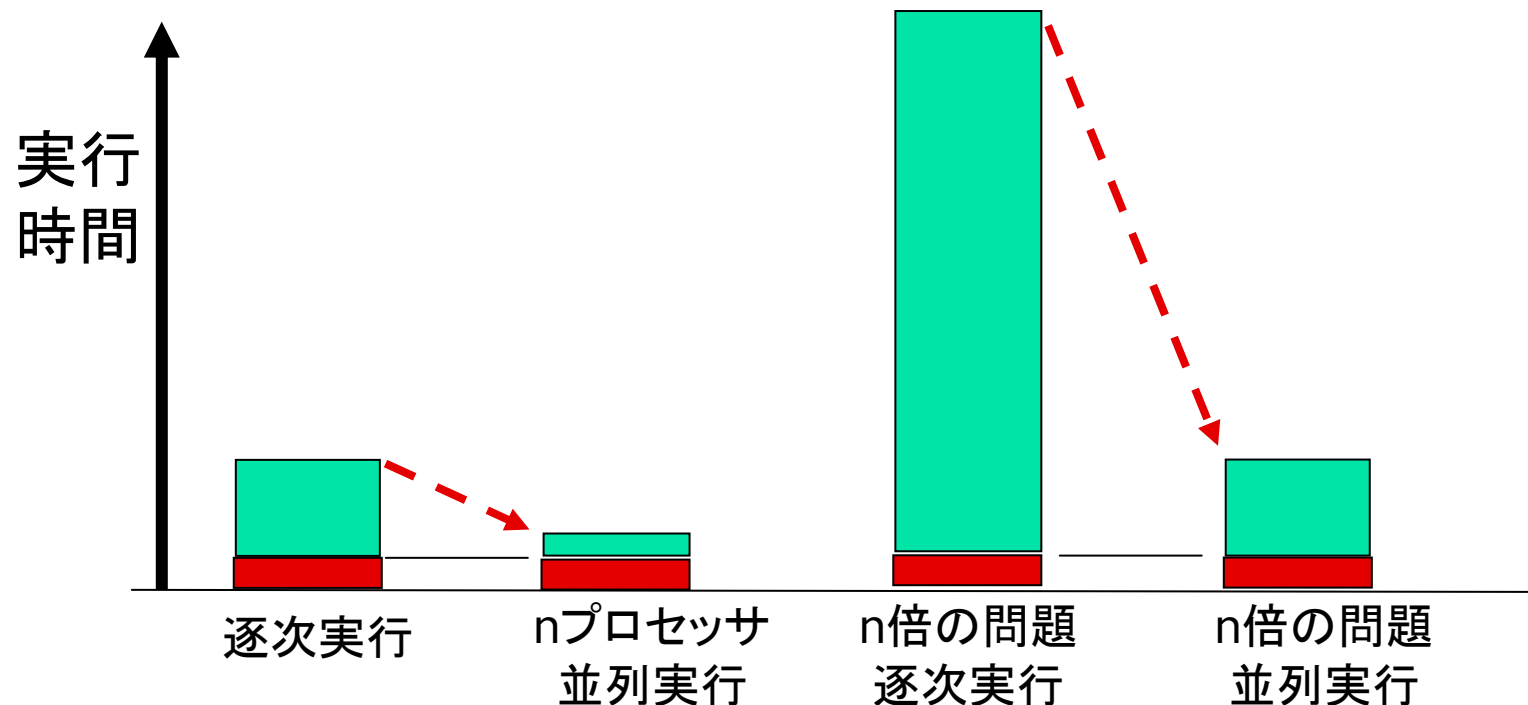
■ アムダールの法則

- 逐次処理での実行時間を T_1 , 逐次で実行しなくてはならない部分の比率が α である場合、 p プロセッサを用いて実行した時の実行時間(の下限) T_p は、 $T_p = \alpha * T_1 + (1 - \alpha) * T_1 / p$
- つまり、逐次で実行しなくてはならない部分が10%でもあると、何万プロセッサを使っても、高々10倍にしかない。



並列処理の問題点:「アムダールの法則」の呪縛

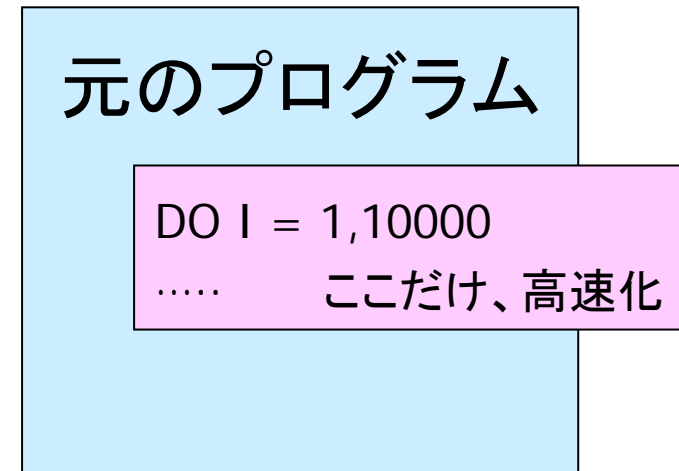
- 「Gustafsonの法則」: では実際のアプリではどうか？
 - 並列部分は問題規模によることが多い
 - 例えば、ノード数 n の場合、 n 倍の大きい問題を解けばよい。 n 倍の問題は、計算量が n になると、並列処理部分は一定
 - Weak scaling – プロセッサあたりの問題を固定 ← 大規模化は可能
 - Strong scaling – 問題サイズを固定 ← こちらはプロセッサが早くなくてはならない。



並列処理の問題点：並列化はなぜ大変か

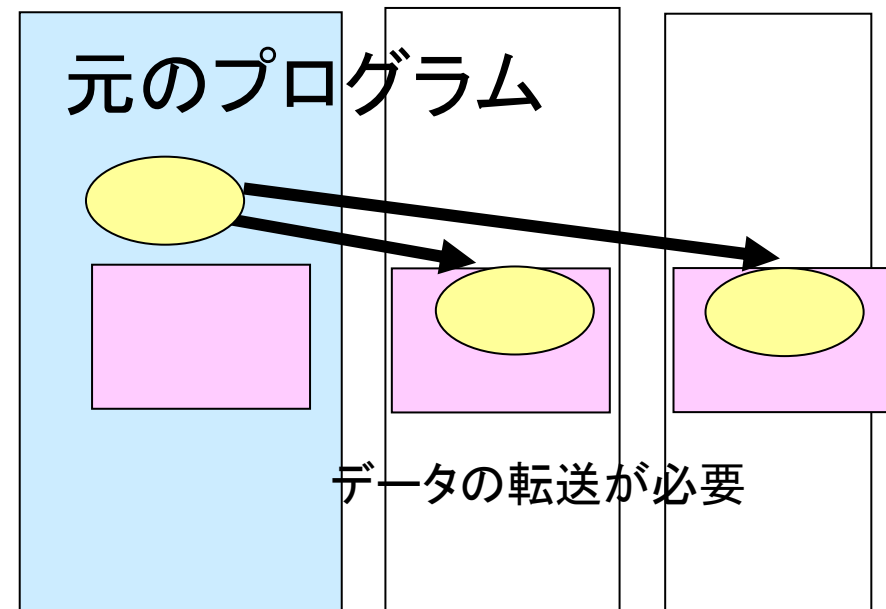
■ ベクトルプロセッサ

- 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

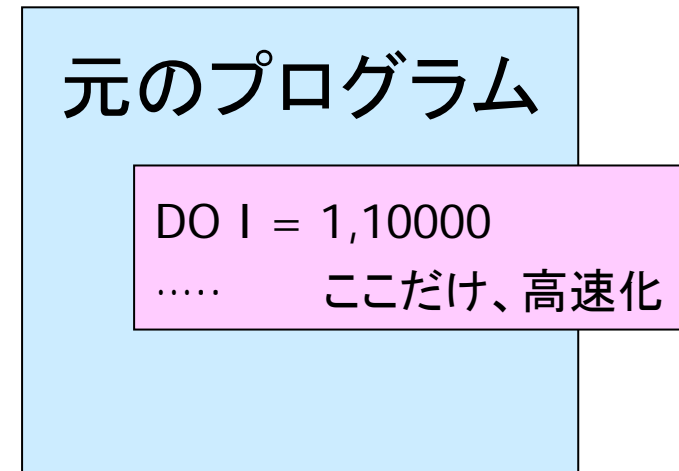
- 計算の分割だけでなく、通信（データの配置）が本質的
- データの移動が少なくなるようにプログラムを配置
- ライブラリ的なアプローチが取りにくい
- 高速化は数千倍～数万



並列処理の問題点：並列化はなぜ大変か

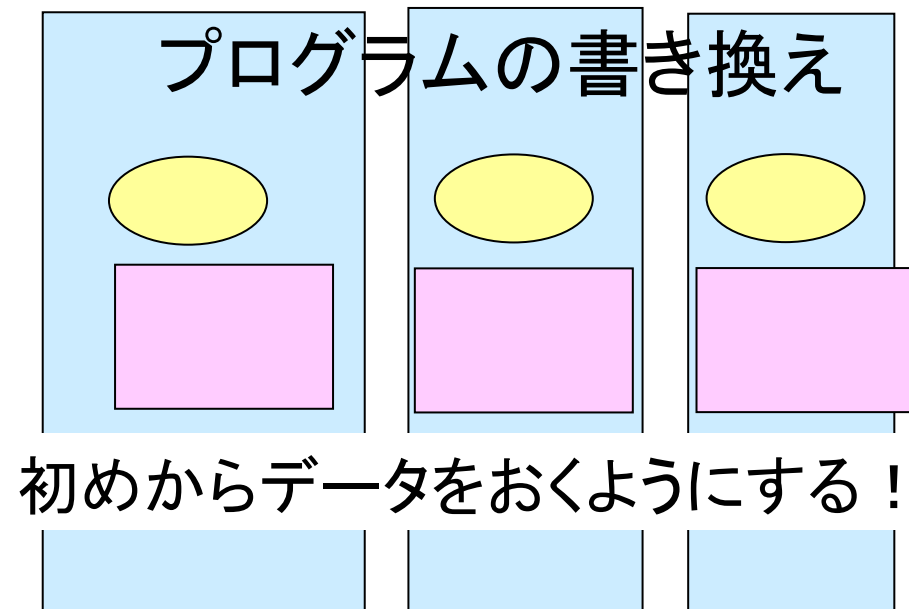
■ ベクトルプロセッサ

- 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



~~これからのスパコン~~

~~次世代スパコン~~はあなたの(逐次)プログラムを
速く実行してくれるか？



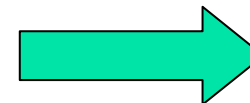
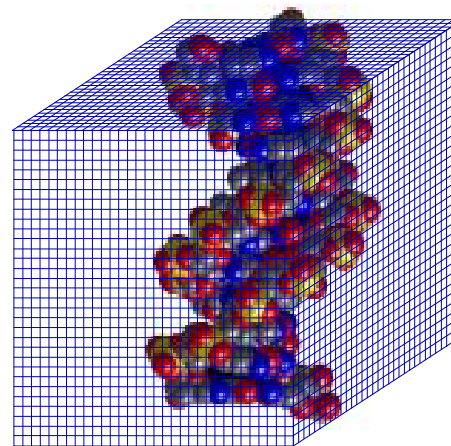
(多分)**No!**

- 10PFLOPS超のシステムの実現に当たって最も重要な課題は
消費電力と設置スペース
 - 現在のPC(10GFLOPS/100W)で10PFLOPSシステムを作ると、100MW必要
 - 1PFLOPSならば、現状の延長線にあるが、10PFLOPSでは再検討が必要
- 個々のプロセッサはむしろ遅くなる傾向
 - 低電力化のため、電圧を下げる必要
 - ➡ 逐次プログラムはむしろ遅くなる(こともある)
- プロセッサの演算性能に比べてメモリバンド幅が足りなくなる

これからのスパコン

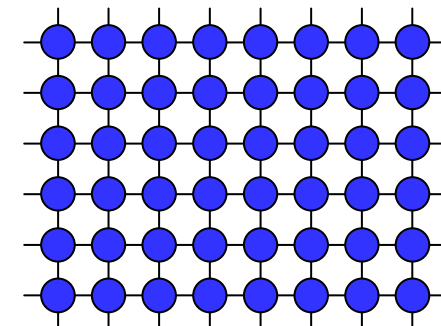
~~次世代スパコン~~での高速化には、並列化は必須

- まず、並列化ができる計算科学の人材の育成が必要
 - 並列プログラミングMPI, OpenMP ...
- 計算科学の研究者と計算機科学の研究者が協調して、問題の定式化、アルゴリズムの再検討を行いアプリケーション開発を進めることが必要
 - PFLOPSの性能を必要とする問題は潜在的に膨大な並列性を内在する(はず)
 - 例: FFT(全対全通信)を必要としないスケーラブルなアルゴリズム、FMOなど
- 近接相互作用を中心に並列処理を行う「実空間アプローチ」(超並列向き)
 - 物理の様々な問題に対して応用できる
 - 境界条件を現実に合わせて設定できる
 - 例; 実空間密度汎関数法(RS-DFT)

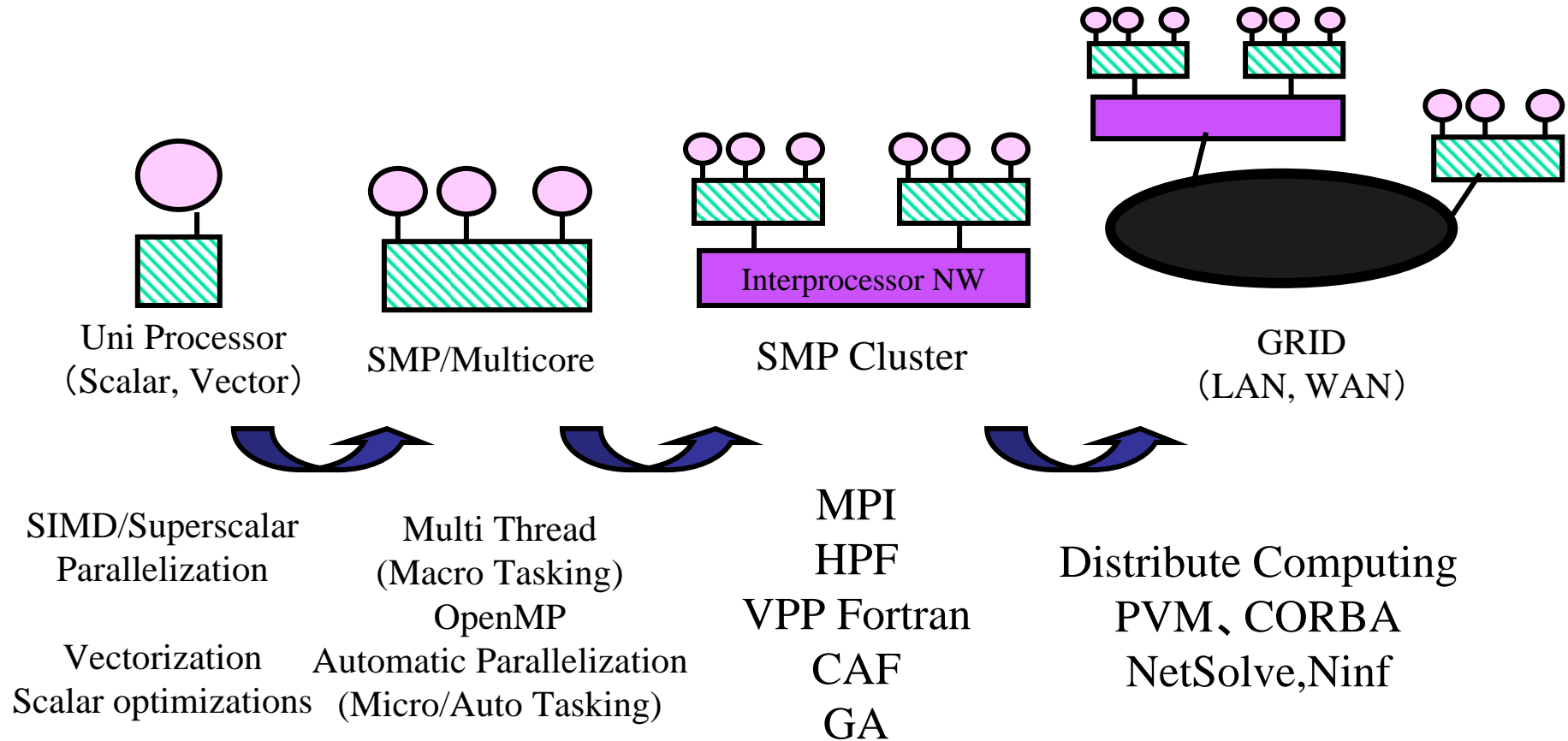


実空間離散化
から直接マップ

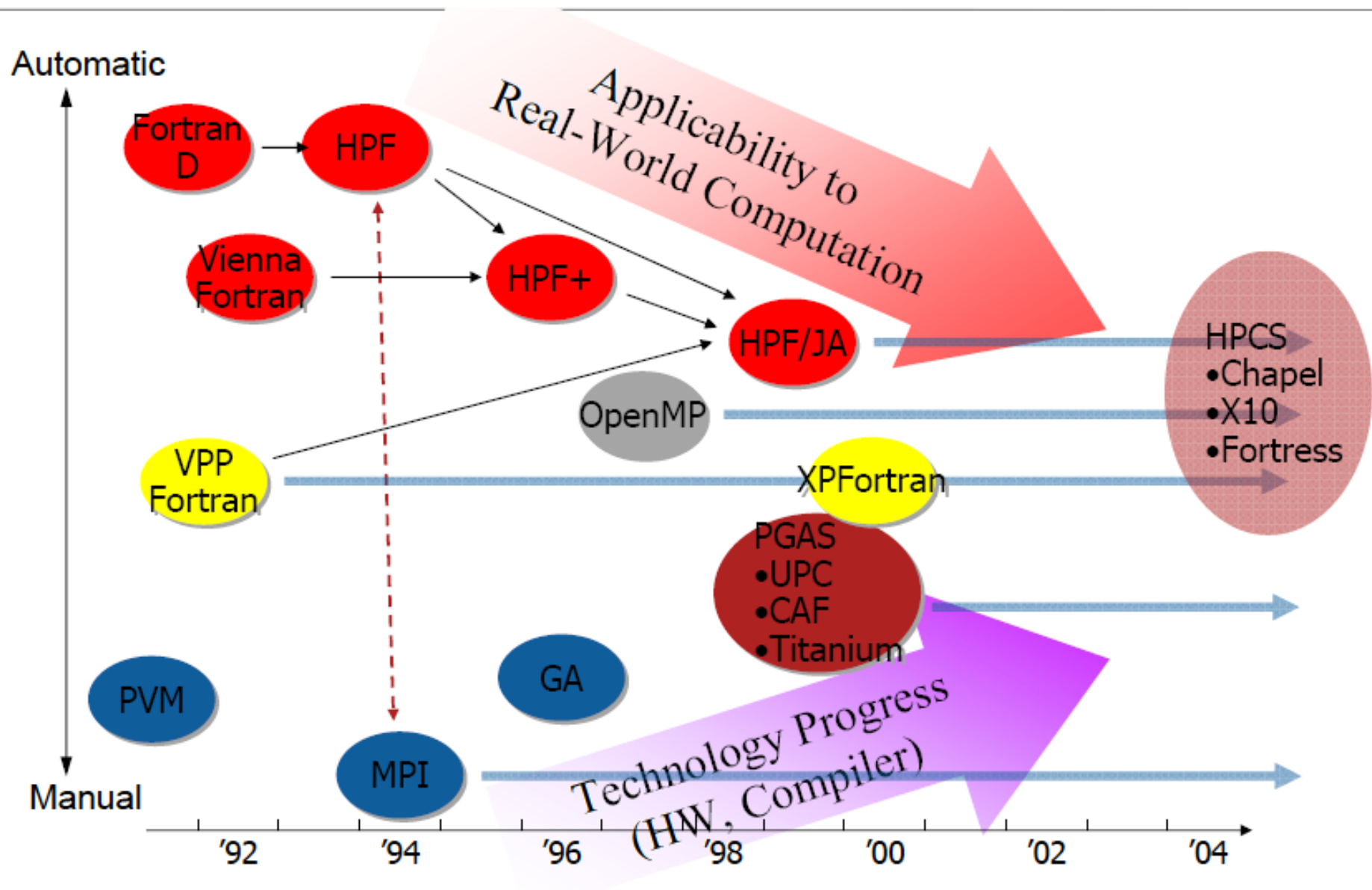
物理空間を直接
シミュレーション



並列プログラミングモデルとPlatform

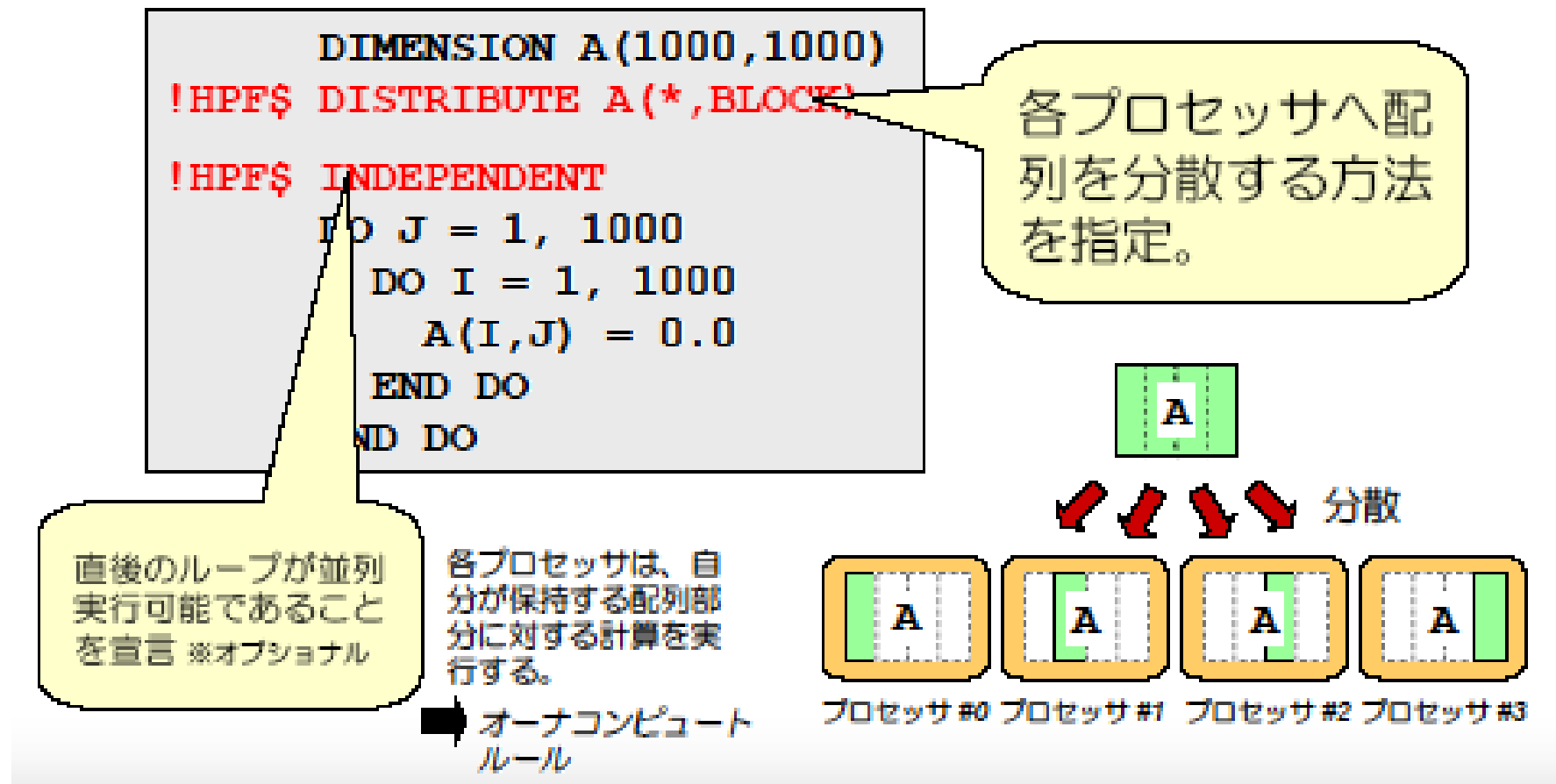


History and Trends for Parallel Programming Languages



HPF: High Performance Fortran

- Data Mapping: ユーザが分散を指示
- 計算は、owner-compute rule
- データ転送や並列実行制御はコンパイラが生成



HPF/JA、HPF/ES

■ HPF/JA

■ データ転送制御directiveの拡張

- Asynchronous Communication, Shift optimization, Communication schedule reuse

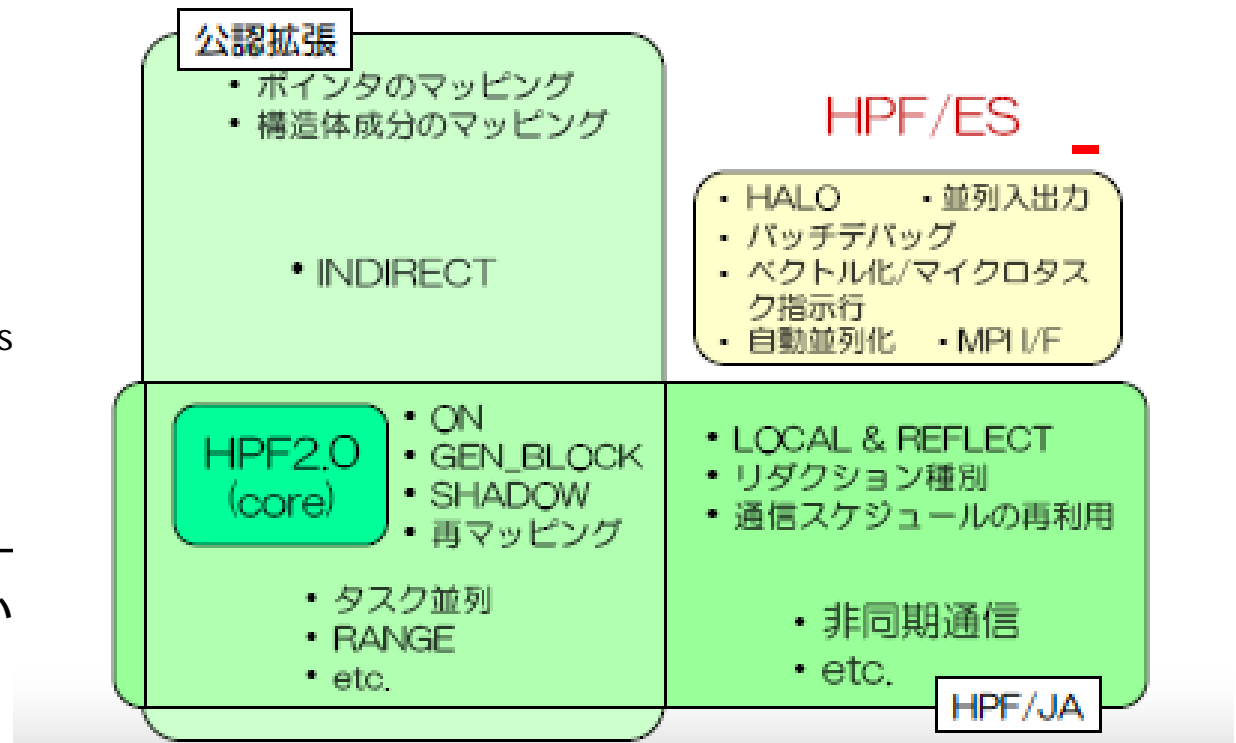
■ 並列化支援の強化(reduction等)

■ HPF/ES

■ HALO, Vectorization/Parallelization handling, Parallel I/O

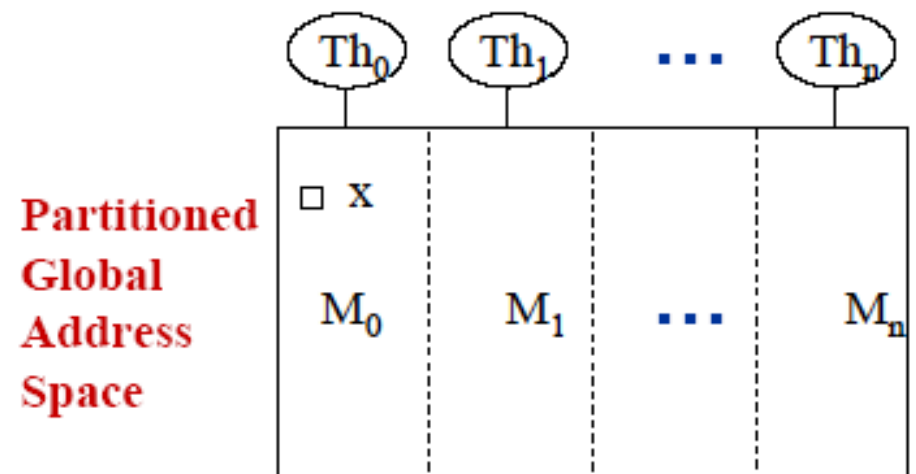
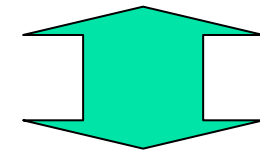
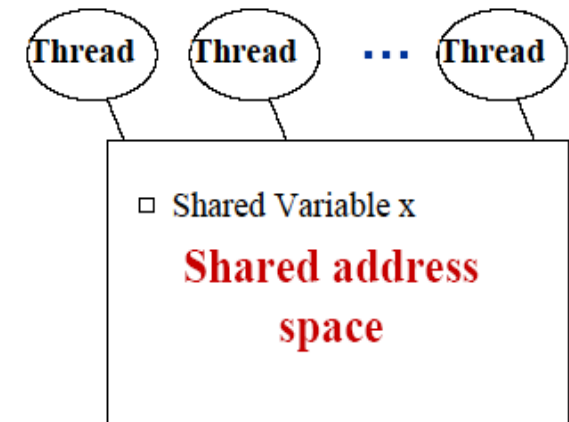
■ 現状

- HPFは、日本(HPFPC (HPF推進協議会))でサポートされている
- SC2002 Gordon Bell Award
 - 14.9 Tflops Three-dimens Fluid Simulation for Fusion Science with HPF on the Earth Simulator
 - HPFそのままではない (ト
- 国内ベンダーはサポートしてい
- 米国dHPF at Rice U.



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)



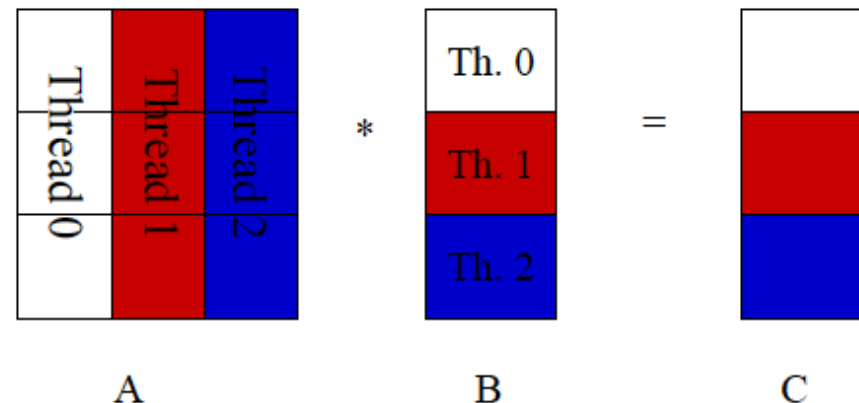
UPC: Unified Parallel C

- 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];
    }
}
```



Shared宣言

- Sharedというqualifierを導入する
- Shared array の要素とブロックはthreadのメモリ空間に分散配置される。

```
shared int x[THREADS] /*One element per thread */  
shared int y[10][THREADS] /*10 elements per thread */
```

- sharedなスカラー変数

```
shared int a; /*One item on system (affinity to thread 0) */  
int b; /* one private b at each thread */
```

- Shared Pointer

```
shared int *p;
```

THREADS = 3の場合

```
shared int x;  
/*x will have affinity to thread 0*/  
shared int y[THREADS];  
int z;
```

Thread 0

x

y[0]

z

Thread 1

y[1]

z

Thread 2

y[2]

z

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)

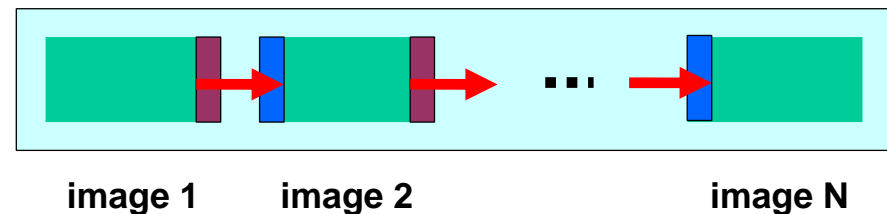
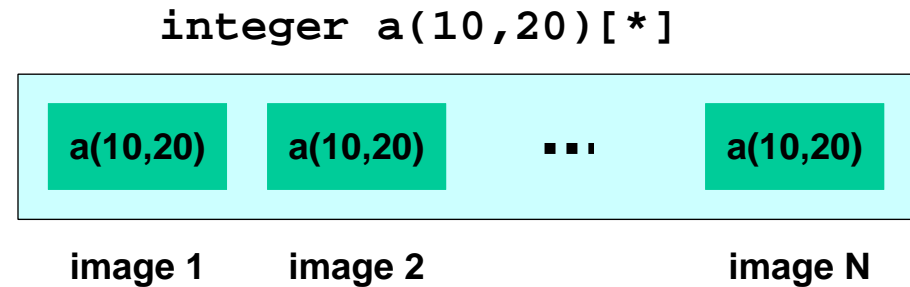
- プログラムは、パフォーマンス影響を与える要素に対して制御する。

- 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



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

CAF Programming Model

- SPMD process images
 - fixed number of images during execution
 - images operate asynchronously
- Both private and shared data
 - **real x(20, 20)** a private 20x20 array in each image
 - **real y(20, 20)[*]** a shared 20x20 array in each image
- Simple one-sided shared-memory communication
 - **x(:,j:j+2) = y(:,p:p+2)[r]** copy columns from image **r** into local columns
- Synchronization intrinsic functions
 - **sync_all** – a barrier and a memory fence
 - **sync_mem** – a memory fence
 - **sync_team**([team members to notify],
[team members to wait for])
- Pointers and (perhaps asymmetric)
- dynamic allocation

2次元のco-Array notationも可能
2次元のCo-dimension

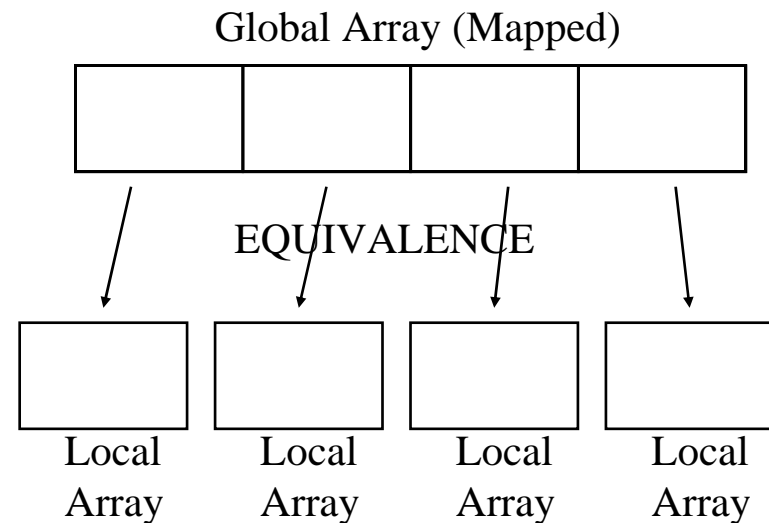
```
real,dimension(n,n)[p,*] :: a,b,c
do k=1,n
  do q=1,p
    c(i,j)[myP,myQ] = c(i,j)[myP,myQ]
      + a(i,k)[myP, q]*b(k,j)[q,myQ]
  enddo
enddo
```

行列積の例

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
```



Global Address Space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。

現状のまとめ

- MPI
 - これが残念ながら、現状！
 - これでいいのか...!?
- OpenMP:
 - 簡単。incrementalに並列化できる。
 - 共有メモリ向け、100プロセッサまで
 - incrementalでいいのだが、そもそも分散メモリには対応していない。
 - MPIコードがすでにある場合は、Mixed OpenMP-MPI はあまり必要ないことが多い
- HPF:
 - 使えるようになってきた(HPF for PC cluster)
 - が、実用的なプログラムは難しいし、問題点もある
 - コンパイラに頼りすぎ。実行のイメージが見えない
 - そもそも、...
- Global Address Space 言語
 - 米国では、だんだん広まりつつある。
 - MPIよりはまし。そこそこの性能もでる...
 - 基本的に、プログラムを書き換える必要がある。
 - そもそも、このくらいで手をうってでもいいのか...
- 自動並列化コンパイラ
 - 究極。共有メモリにはそこそこ使えるようになってきている。が、分散メモリは、むずかしい。

Think about MPI, ...

- Why was MPI accepted and so successful?
 - Portability: Most parallel computing platforms can run MPI programs (even in SMP).
 - Many free and portable software such as MPICH.
 - Education: MPI Standard allows many programmers to learn MPI parallel programming.
 - In university
 - By book
- The cost of parallelization is also important for acceptance by application programmers.
 - Easy to transfer from an original sequential program.
 - What application programmers need to learn must be small.

Many Parallel programming languages ...

- So far, many parallel programming languages were proposed in computer science community.
- Are they actually used by application users?
- Where were they gone?
- What is missing in them?
- Why?

Jede
HPC++
mpc++
HPF
Linda
Mentat
Fortran M
Occam
APL
SAL

pC++
SISAL
NESL
Clik
pHaskel
Prolog
Orca
mpC
C*
dataparallel C

Split-C
Fortran D
V
Charm++
CODE
ZPL
Fortran X3H5
.....

“Petascale” Parallel language design working group

■ Objectives

- Making a draft on “petascale” parallel language for “standard” parallel programming
- To propose the draft to “world-wide” community as “standard”

■ Members

- Academia: M. Sato, T. Boku (compiler and system, U. Tsukuba), K. Nakajima (app. and programming, U. Tokyo), Nanri (system, Kyusyu U.), Okabe (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)

■ 4 WG meetings were held (Dec. 13/2007 for kick-off, Feb. 1, March 18, May 12)

■ Requests from Industry (at the moment before starting activities)

- Not only for scientific applications, also for embedded multicore systems
- Not only for Japanese standard, and should have a strategy for world-wide “standard”
- Consider a transition path from existing languages such as HPF and XPFortran.

The rise and fall of High Performance Fortran

-- Lessons learned from HPF --

“with love” ☺ by Sakagami@NIFS and Murai@NEC

■ Background

- MPI is (still now) an **obstacle** for programming a distributed memory systems.
 - Debugging MPI code is not easy, and update/modification of MPI program forces a tough work for application people.
 - If MPI is only a solution to parallel machine, nobody wants to use parallel machines. (EP is ok, but ...)
- There was a great demand for parallel programming languages!

■ Why HPF?

- Application people want just easy parallel programming environment with reasonable (not necessarily perfect) performance.
- OpenMP is just for shared memory systems.
- Not practical alternative solutions. (New languages by US HPCS project are research prototype, not yet available for practical use.)

HPF history in Japan

- Still survive in Japan, supported by HPF promotion consortium
- HPF Users Group Meeting (HUG from 1996-2000), HFP intl. workshop (in Japan, 2002 and 2005)
- Compiler Availability
 - HPF/ES (HPF+HPF/JA+some extension for Earth Simulator)
 - HPF/SX, HPF/VPP, HPF/ES for PC clusters, fhpf (free software distributed by HPF consortium)
 - ADAPTOR (GMD, Germany)
 - SHPF (U. Vienna, Austria)
 - PGHPF (PGI, US)

“Pitfall” in Design policy of HPF

- “Ideal” design policy of HPF
 - A user gives a small information such as data distribution and parallelism.
 - The compiler generates “good” communication and work-sharing automatically.
 - By ignoring directives, parallelized code can be considered as the original sequential code.

- **“Don’t give too much expectation to users which the technology could not meet.”**
 - This “ideal” design policy had generated a great “expectation” from users!
 - But, the reality was not ...

“Pitfall” in the base language

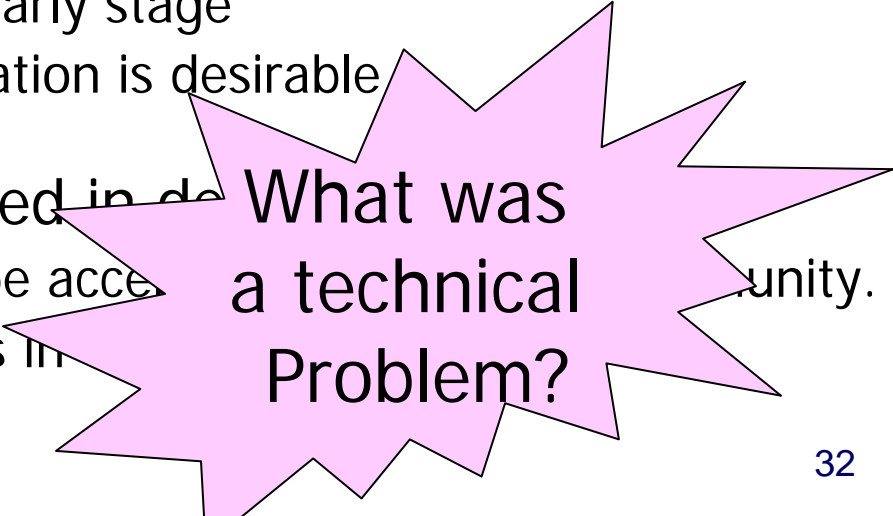
- The base language of HPF was Fortran 90.
- A bad thing was that at the moment of HPF announced (mid 90's), F90 was still immature.
- Many application people have to rewrite programs in F90 in order to use HPF
 - Their code was often written in F77.
 - Re-write from F77 to F90 was not easy work.
- *“Application people don't want to rewrite their programs. They are very conservative”*
- Sometimes, they complained that “I re-wrote my program by spending a lot time, but the performance was not good!”
 - The reason why the performance of HPF was not so good was sometimes due to the immaturity of F90 implementation.

“Pitfall” in compiler optimization and quality

- No explicit mean for performance tuning .
 - Everything depends on compiler optimization.
- Users can specify more detail directives, but no information how much performance improvement will be obtained by additional information
 - INDEPENDENT for parallel loop
 - PROCESSOR + DISTRIBUTE
 - ON HOME
- The performance is too much dependent on the compiler quality, resulting in “incompatibility” due to compilers.
 - “By using compiler A, I got very good performance. But It is not by compiler B.”
- **“Specification must be clear. Programmers want to know what happens by giving directives”**
 - “Small kindness” causes just a “Big trouble” (in Japanese maxim)

Other Lessons ...

- Large specifications were included to satisfy “theoretical” completeness of the language model.
 - Useless complex combination of data distributions
 - “Specification must be clear and simple”
- Immaturity of early compilers disappointed users.
 - “Don’t release a bad immature compiler in early stage”
- No reference implementation of HPF like MPICH in MPI standard.
 - Many different interpretations in early stage
 - Free and Open-source implementation is desirable
- “Education” should be considered in development.
 - How will a new language should be accepted by the community.
 - No text book, no tutorial materials in early stage.



What was
a technical
Problem?

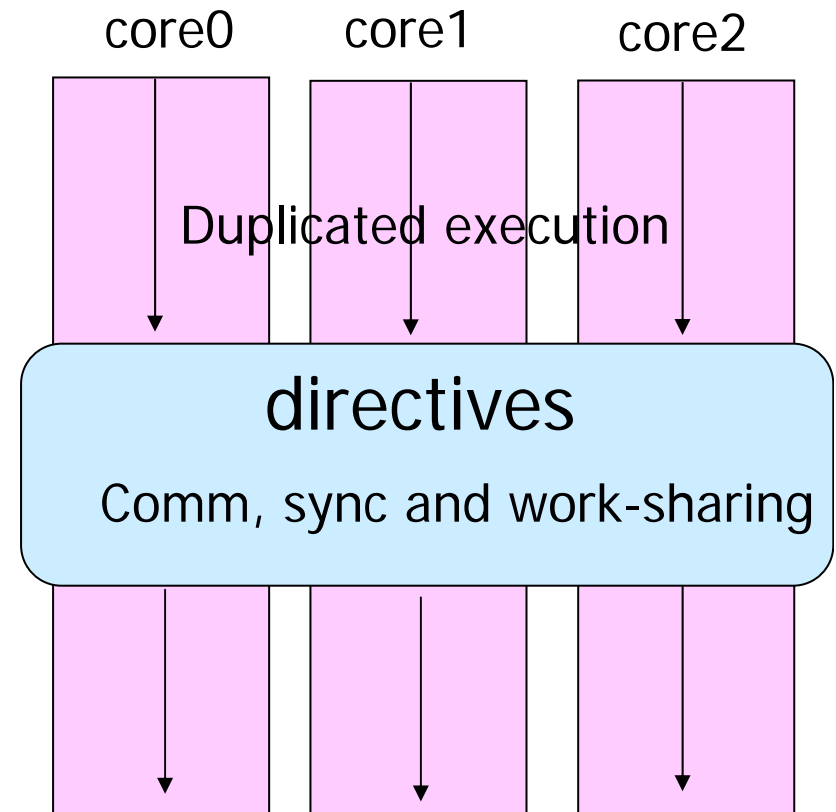
Status of Parallel Language WG

- What we agreed in the last meeting:
 - Basic execution model (SPMD)
 - Directive extension
 - Explicit communication for user's performance tuning
 - Support for common communication pattern
 - Need of the interface to MPI at low level to allow wider range of distributed parallel programming

- What we don't agree yet:
 - Global view vs. Local view
 - One-sided communication vs. Two-sided communication

Objectives and What we agreed (1)

- Easy to use for beginners. Easier than MPI.
 - Use the existing languages (C/Fortran) as the base language and extend them by the directives (like OpenMP)
 - Minimum extension of the base language if required
 - (Education cost is cheaper than a “new language”)
- Communication must be explicit
 - Communication happens when specification specifies it.
 - It is important for performance tuning by the expert users.
- SPMD model as a basic execution model
 - Duplicated execution if no directive specified.
 - Work-sharing when directives are encountered.



Objectives and What we agreed (2)

- Support common pattern (communication and work-sharing) for data parallel programming
 - Reduction and scatter/gather
 - Communication of sleeve (shadow) area
 - Like OpenMPD, HPF/JA, XFP
 - Still we have a problem how to view & describe distributed data (See next ...)
- Need to interface to MPI at low level
 - Allows the programmer to use MPI
 - Support for wider-range of distributed-memory parallel programming
 - It can be useful to program for large-scale parallel machine.

Under discussion (not agreed yet)

- Global view vs. Local view



- Global View (HPF-like)
 - Easy-to-use for global operations for Array
 - Good for array-based computation (e.g. CFD)
- Local view (PGAS, CAF, UPC)
 - SPMD, local and external memory operations are explicit
 - Good for FEM-type applications.
- One-sided communication vs. Two-sided communication
 - One-side comm. is easy-use, but sometime requires HW for high performance
 - Semantics of One-sided communication is not clear.

「並列プログラミング言語プロジェクト」の現状のまとめ

- 標準的な並列プログラミング言語を目指す
 - 新規の言語でなく、通常の言語をベースにする。
- 初心者に使しやすい、HPF/XPFユーザに受け入れやすい
 - Global view を提供
 - Alignment, affinityの提供
 - 集団通信をサポート
- MPIユーザが受け入れ安い、MPIとの混在、性能チューニングが容易
 - Local viewを提供
 - Get/put
 - Send/recv
 - ライブラリインタフェース
- 大規模並列プログラム
 - タスク並列
 - タスク動的並列

おわりに

- 並列プログラミング言語検討会の目的は、標準的な言語を決めること
 - 超並列マシンの並列プログラミングにはいろいろな課題はあるが、... 生産性(productivity)をあげることが重要
 - 「MPIよりもましなプログラミング環境を！」
- これからの計画
 - 年内に初版の仕様を決定
 - 文部科学省次世代IT基盤構築のための研究開発「e-サイエンス実現のためのシステム統合・連携ソフトウェアの研究開発」の中で、筑波大が中心に開発・実装を進める
 - 国際的な標準化活動を展開

並列プログラミング言語検討委員会(予定)

- 佐藤 三久 筑波大学計算科学研究センター センター長・教授
- 横川 三津夫 理化学研究所次世代スーパーコンピュータ開発実施本部
- 朴 泰祐 筑波大学計算科学研究センター 教授
- 坂上 仁志 核融合科学研究所 教授
- 岡部 寿男 京都大学学術情報メディアセンター 教授
- 中島 研吾 東京大学情報基盤センター 特任教授
- 南里 豪志 九州大学情報基盤研究開発センター 准教授
- 松尾 裕一 宇宙航空技術研究開発機構
- 上原 均 海洋研究開発機構
- 堀田 耕一郎 富士通(株)
- 岩下 英俊 富士通(株)
- 左近 彰一 日本電気(株)
- 村井 均 日本電気(株)
- 根岸 清 (株)日立製作所
- 安崎 篤郎 (株)日立製作所