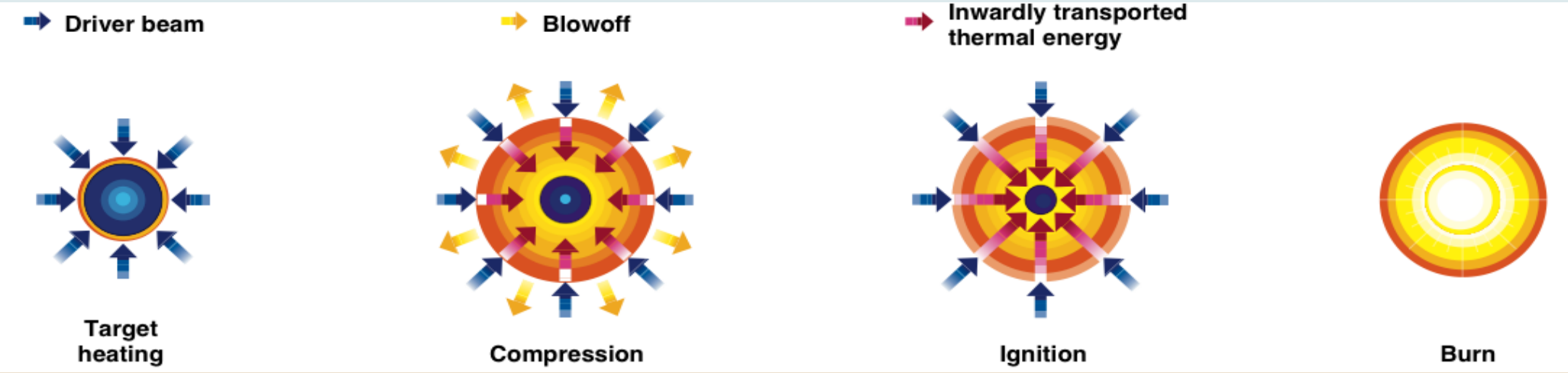
3次元流体コードによる「京」におけるXcalableMPの性能評価

核融合科学研究所
基礎物理シミュレーション研究系
坂上仁志

XcalableMP

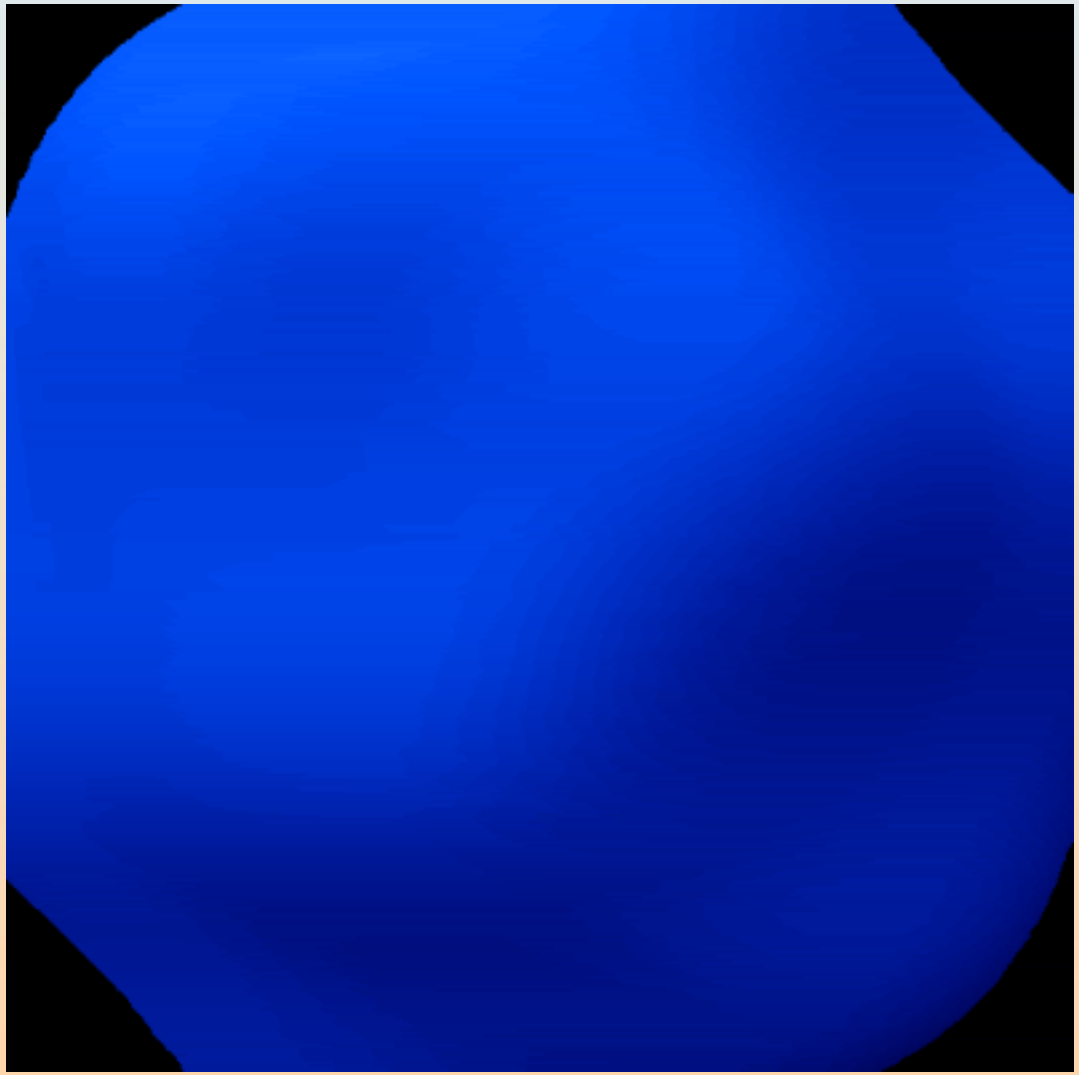
レーザー核融合の概要



(1) DT燃料を含んだターゲットを四方八方からレーザーで照射して急速に加熱する.
(2) ターゲット表面がプラズマ化して噴出し,その反作用で燃料が圧縮／爆縮される.(ロケット効果)
(3) 燃料コアの温度が10keV,密度が固体密度の1000倍になると,自己点火が起こる.
(4) 圧縮された燃料全体に核融合燃焼が迅速に広がり,投入したエネルギー以上のエネルギーが得られる.

困難な均一爆縮

- レイリーテイラー不安定性の成長



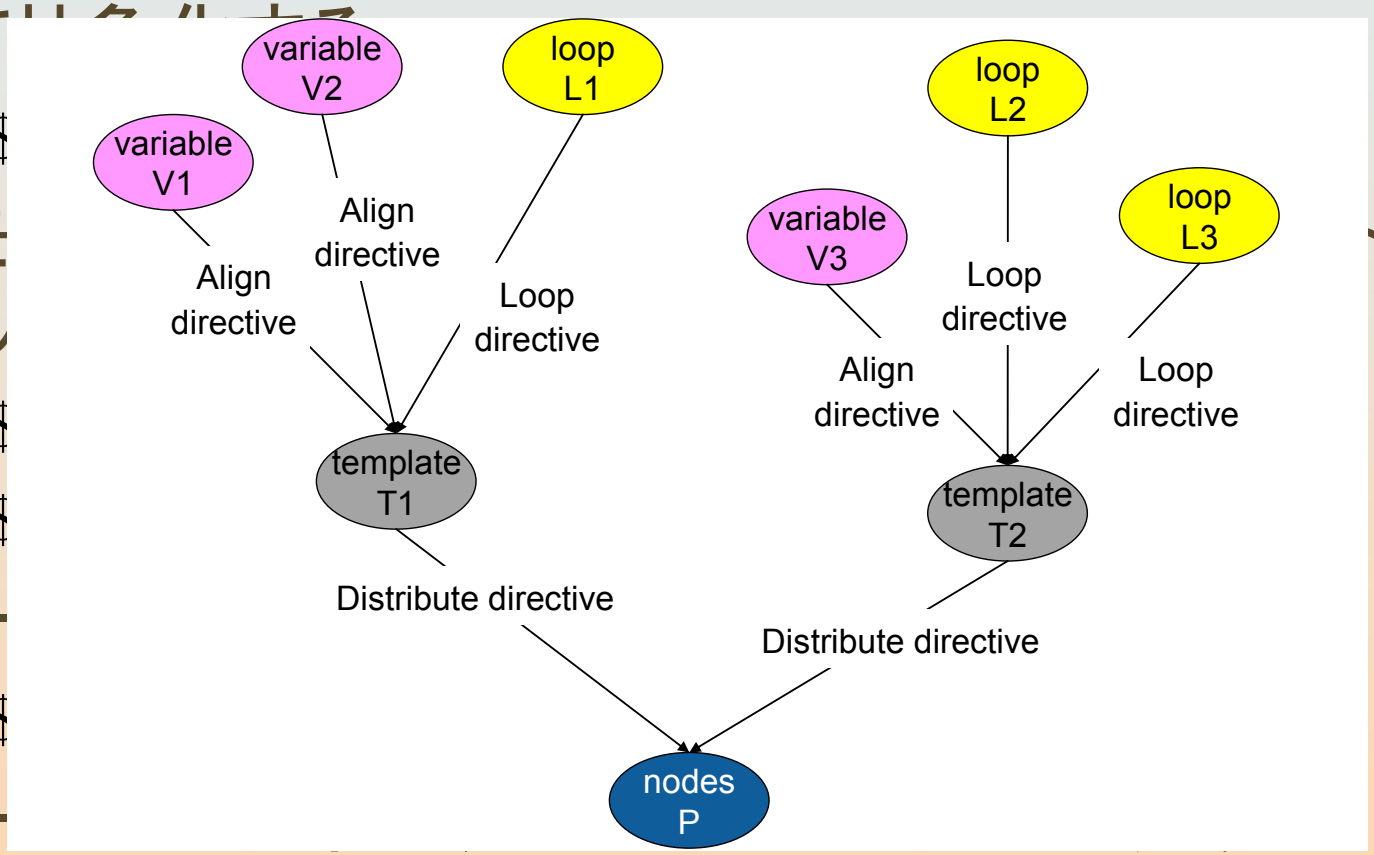
XcalableMP

3次元流体コード

- レーザー核融合の爆縮過程をシミュレーション
- 3次元流体方程式(オイラー方程式)
  - 非粘性
  - 圧縮性
- 3次元カーテシアン座標系
  - 原点を跨ぐ非対称な流れ
- 5点差分TVDスキームによる空間微分
  - 典型的なステンシル計算
- 陽的解法による時間積分
  - 多次元の時間発展は分ステップ法

XMPのグローバルビューモデル

- 分散メモリ環境のプロセッサ（複数）とメモリをノードとして
  - !$
- メモ
  - !$
  - !$
- デー
  - !$
- ルー
  - !$xmp loop on t(i)

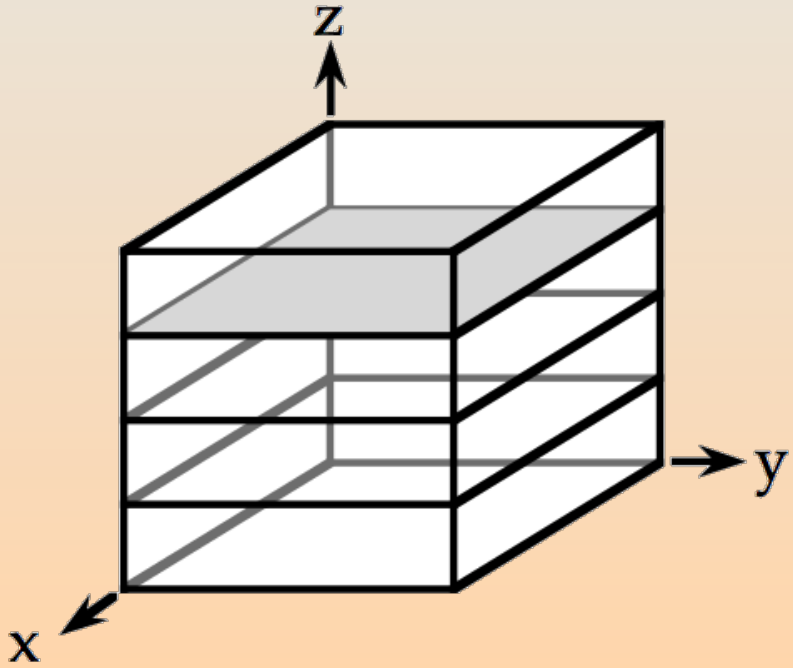


Z方向のみの領域分割

```
integer parameter :: lx=2048, ly=2048, lZ=2048
integer parameter :: nz=256
!$XMP NODES proc(nz)
!$XMP TEMPLATE t(lx,ly,lz)
!$XMP DISTRIBUTE t(*,*,BLOCK) ONTO proc
real*8 :: physval(6,lx,ly,lz)
!$XMP ALIGN (*,*,*,i) WITH t(*,*,i) :: physval
!$XMP SHADOW (0,0,0,1) :: physval

!$XMP LOOP (iz) ON t(*,*,iz)
!$OMP PARALLEL DO REDUCTION(max:wram) PRIVATE(iy,ix,...)
do iz = 1, lz-1
  do iy = 1, ly
    do ix = 1, lx
      ...
end do
!$XMP REDUCTION(max:wram)

!$XMP REFLECT (physval)
!$XMP LOOP (iz) ON t(*,*,iz)
!$OMP PARALLEL DO PRIVATE(iy,ix)
do iz = 1, lz-1
  do iy = 1, ly
    do ix = 1, lx
```

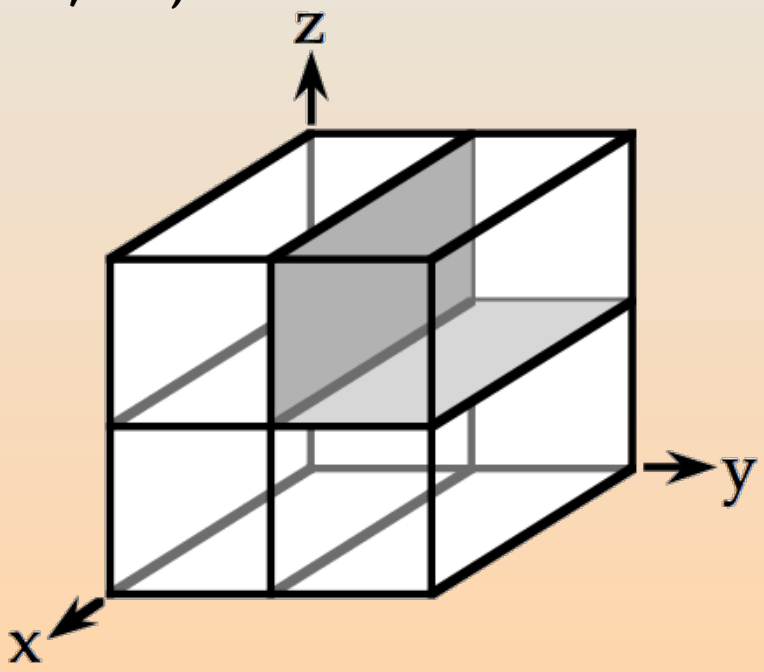


YとZ方向の領域分割

```
integer parameter :: lx=2048, ly=2048, lZ=2048
integer parameter :: ny=16, nz=16
!$XMP NODES proc(ny,nz)
!$XMP TEMPLATE t(lx,ly,lz)
!$XMP DISTRIBUTE t(*,BLOCK,BLOCK) ONTO proc
real*8 :: physval(6,lx,ly,lz)
!$XMP ALIGN (*,*,i,j) WITH t(*,i,j) :: physval
!$XMP SHADOW (0,0,1,1) :: physval

!$XMP LOOP (iy,iz) ON t(*,iy,iz)
!$OMP PARALLEL DO REDUCTION(max:wram) PRIVATE(iy,ix,...)
do iz = 1, lz-1
  do iy = 1, ly
    do ix = 1, lx
      ...
end do
!$XMP REDUCTION(max:wram)

!$XMP REFLECT (physval) width(0,0,0,1)
!$XMP LOOP (iy,iz) ON t(*,iy,iz)
!$OMP PARALLEL DO PRIVATE(iy,ix)
do iz = 1, lz-1
  do iy = 1, ly
    do ix = 1, lx
```



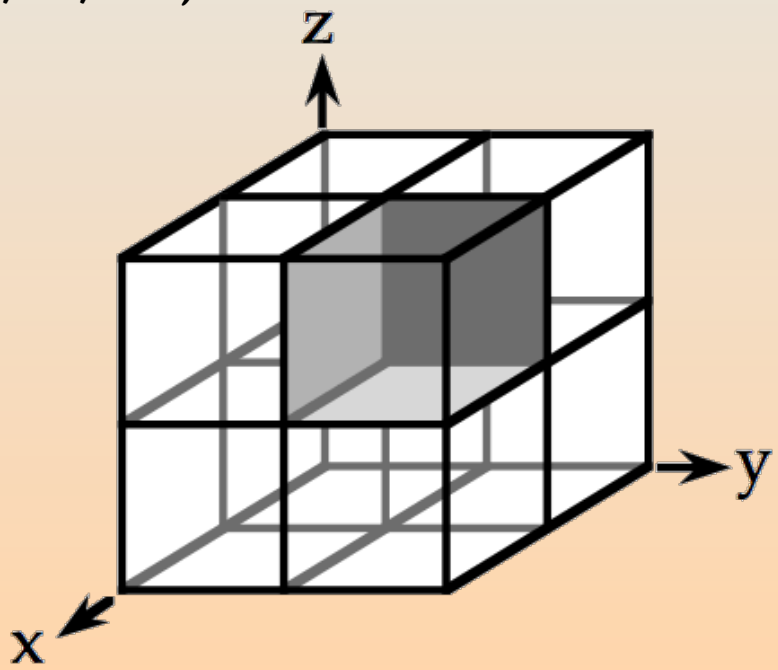
X,Y,Z全方向の領域分割

```
integer parameter :: lx=2048, ly=2048, lZ=2048
integer parameter :: nx=8, ny=8, nz=4
!$XMP NODES proc(nz)
!$XMP TEMPLATE t(lx,ly,lz)
!$XMP DISTRIBUTE t(BLOCK,BLOCK,BLOCK) ONTO proc
real*8 :: physval(6,lx,ly,lz)
!$XMP ALIGN (*,i,j,k) WITH t(i,j,k) :: physval
!$XMP SHADOW (0,1,1,1) :: physval

!$XMP LOOP (ix,iy,iz) ON t(ix,iy,iz) REDUCTION(max:wram)
!$OMP PARALLEL DO REDUCTION(max:wram) PRIVATE(iy,ix,...)
do iz = 1, lz-1
  do iy = 1, ly
    do ix = 1, lx
      ...
end do


!$XMP REFLECT (physval) width(0,0,0,1)
!$XMP LOOP (ix,iy,iz) ON t(ix,iy,iz)
!$OMP PARALLEL DO PRIVATE(iy,ix)
do iz = 1, lz-1
  do iy = 1, ly
    do ix = 1, lx
```

*最新バージョンでは，修正済み



通信回数と通信量

- 多方向に分散するほど,1プロセスが通信するトータルな通信量は減るが,通信回数は増える.
- このため,最適な領域分割方法は,並列コンピュータのノード間ネットワークの性能および特性に依存する.
  - ただし,一方向のみの分割では大規模並列は難しい.
- Z方向のみの領域分割
  - (lx·ly)×4回×2回
- YとZ方向の領域分割
  - (lx·ly)/ny×4回×2回+(lx·lz)/nz×4回×2回
- X,Y,Z全方向の領域分割
  - (lx·ly)/(nx·ny)×4回×2回+(lx·lz)/(nx·nz)×4回×2回+ (ly·lz)/(ny·nz)×4回×3回

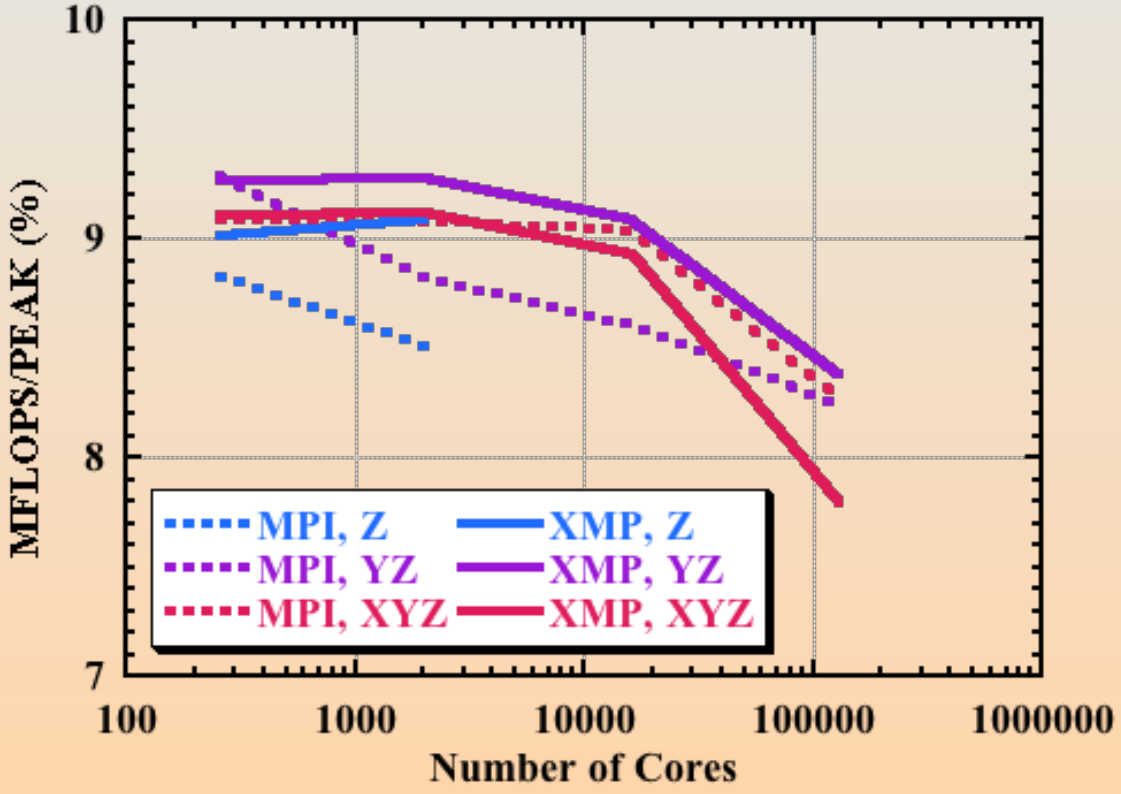
「京」における実行環境

- 「京」は,ノード当たり8コアなので,1ノードに1プロセスを1D的に割り当てて,8スレッド並列とした.
- ハードウェアモニタにより取得したMFLOPS/PEAKの値を用いて,weak scaleの性能評価を行った.
- Omni XcalableMP 0.7.0, Fujitsu Fortran K-1.2.0-15を利用した.

#cores	lx=ly=lz	Zのみ	YとZ		X,Y,Z全て		
		nz	ny	nz	nx	ny	nz
256	1024	32	8	4	4	4	2
2048	2048	256	16	16	8	8	4
16384	4096		64	32	16	16	8
131072	8192		128	128	32	32	16

MPIコードとの比較

- XMPコードの性能は,手で書いたMPIコードの性能とほぼ同等であり,分割方法による差は小さい.
- しかし,「京」ピーク性能の8~9%しか得られていない.



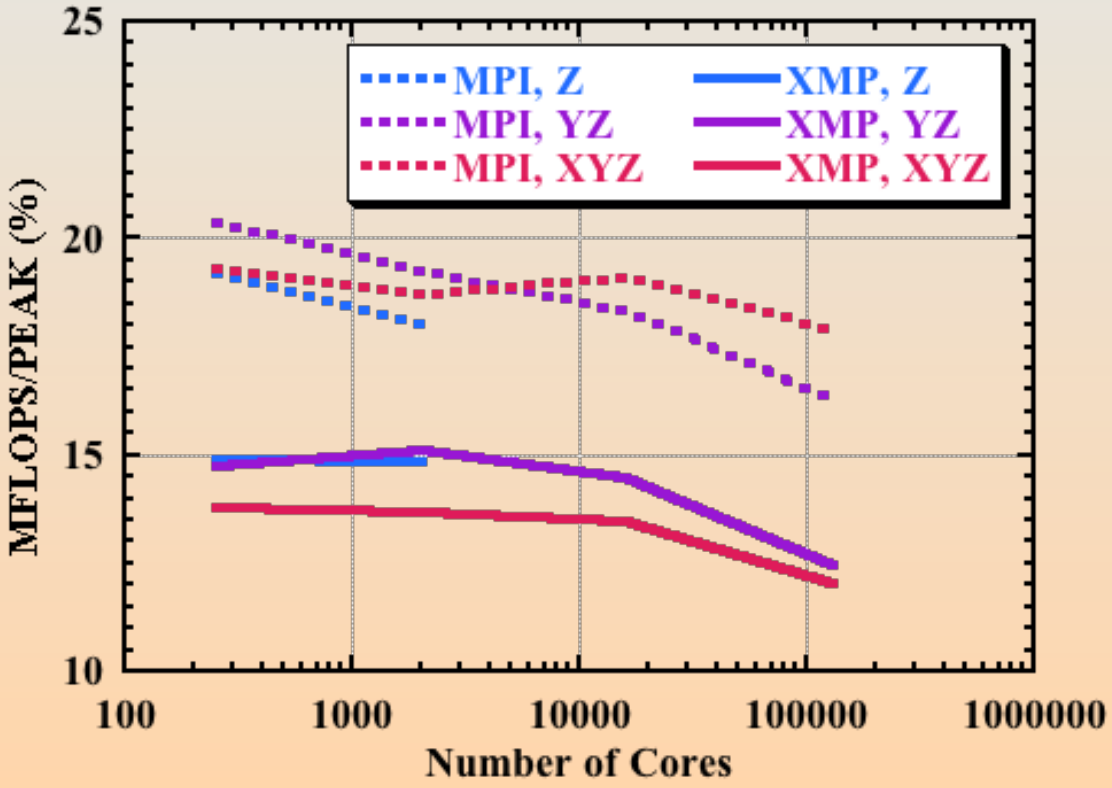
SIMD化の阻害要因

- これは,計算負荷の大きなループがSIMD化されていないことが原因だと考えられる.(HMでのSIMD実行率は5%以下)
  - SIMD化の阻害要因はループ中のIF文である.
  - 流速の方向変化や極めて小さな流速を扱うために,IF文をなくすことはできない.

```
if( walfa1(ix,iy,iz-1) * walfa1(ix,iy,iz) .gt. szero2 ) then
  if( walfa1(ix,iy,iz) .gt. 0.0d0 ) then
    wg1(ix,iy,iz) = min( walfa1(ix,iy,iz-1), walfa1(ix,iy,iz) )
  else
    wg1(ix,iy,iz) = max( walfa1(ix,iy,iz-1), walfa1(ix,iy,iz) )
  end if
  wtmp1(ix,iy,iz) = somga * &
           abs(      walfa1(ix,iy,iz)   -      walfa1(ix,iy,iz-1)   ) &
            / ( abs( walfa1(ix,iy,iz) ) + abs( walfa1(ix,iy,iz-1) ) )
else
  wg1(ix,iy,iz) = 0.0d0
  wtmp1(ix,iy,iz) = 0.0d0
end if
```

SIMD化の強制

- IF文の真率は高いので,SIMD化を強制する.
  - Ksimd=2 コンパイラオプションの追加指定(SIMD実行率は約50%)
- MPIコードの性能は20%に向上したが,XMPコードの向上は15%に留まっている.



XMPコードの性能低下 ～OpenMPのfork/join～

- MPIコードでは,OpenMPのオーバーヘッドを小さくするため,サブルーチンの先頭でforkし,それぞれのDOループを並列実行した後,returnの直前でjoinする.
- XMPコードでは,並列ループ毎にfork/joinする.
  - XMP LOOP指示文の外側には,OMP指示文を書けない.

```
!$XMP LOOP (ix,iy,iz) ON t(ix,iy,iz)
!$OMP PARALLEL DO PRIVATE(iy,ix)
do iz = 1, lz-1
  do iy = 1, ly
    do ix = 1, lx
```

- そこで,MPIコードも並列ループ毎にfork/joinするように書き換えたが,顕著な性能低下は見られなかった.

XMPコードの性能低下 ～ソフトウェアパイプライニング～

- MPIコードでは,すべてのループがソフトウェアパイプライニングされているが,XMPコードではされていない.

```
jwd6002s-i  "adv3dx.F90", line 104: このDOループをSIMD化しました.
jwd8204o-i  "adv3dx.F90", line 104: ループをソフトウェアパイプライニングしました.
jwd8205o-i  "adv3dx.F90", line 104: このループは,繰返し数が25回以上の時,ソフトウェアパイプライニングが有効になります.

jwd6002s-i  "adv3dx.F90", line 100: このDOループをSIMD化しました.
jwd8662o-i  "adv3dx.F90", line 100: 良いスケジューリング結果を得られなかったため,ソフトウェアパイプライニングを適用できません.
jwd8202o-i  "adv3dx.F90", line 100: ループを展開数2回でアンローリングしました.
```

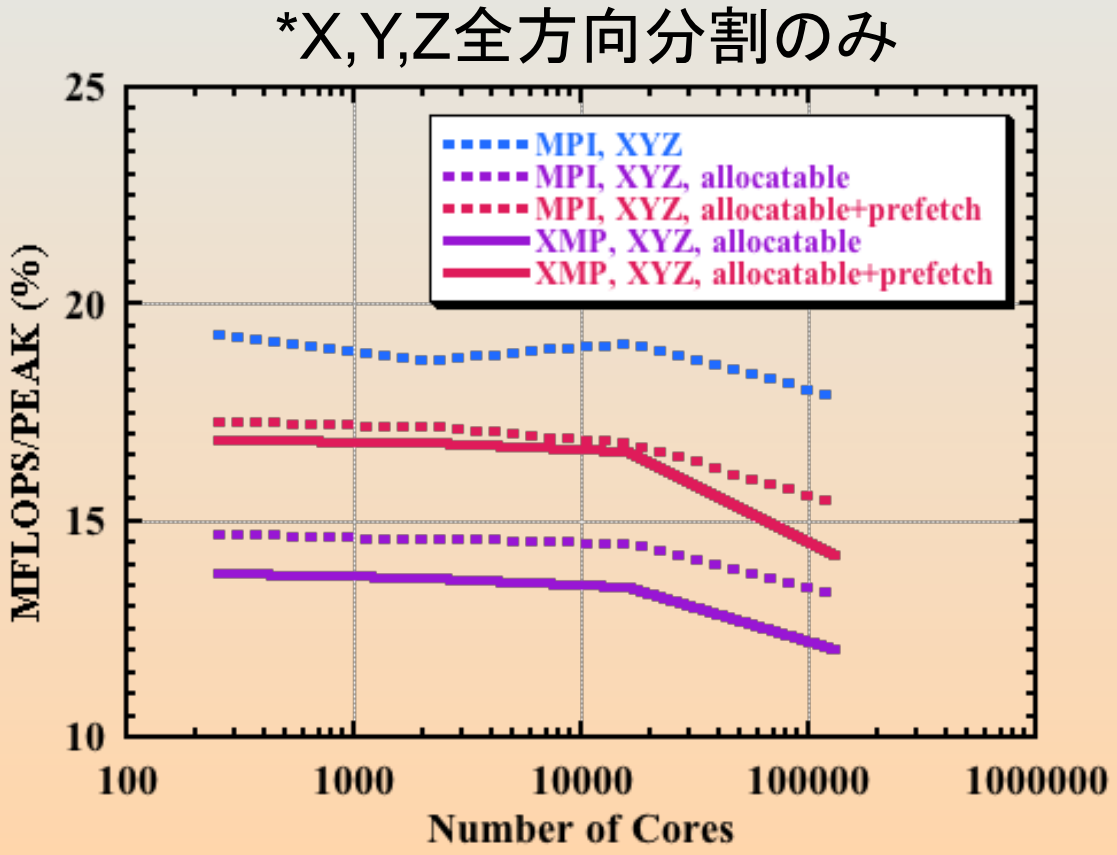
- これは,XMPコードの変換後DO文について,コンパイラの解析能力が不十分であるようだ.
  - do i = is, ie -----> do i1 = xmp_s1, xmp_e1, xmp_d1
- そこで,ステップが1の場合は,do i1 = xmp_s1, xmp_e1, 1 となるように改修した結果,すべてのループがソフトウェアパイプライニングされたが,性能向上は見られなかった.

XMPコードの性能低下 ～プリフェッチ～

- MPIコードでは静的配列を使っているため,適切にプリフェッチされているが,XMPコードでは,すべての配列がallocatableに変換されるため,プリフェッチされていない場合があった.
  - HMのMem. Th./PEAK%は,55 vs. 37.(SIMD% 56 vs. 52)
  - Kprefetch_stride コンパイラオプションの追加指定
- プリフェッチの強制によりXMPコードの性能は17%程度に向上したが,MPIコードには及ばない.
- そこで,MPIコードの静的配列をallocatableに変えると,性能低下が見られた.
  - 富士通Fortranコンパイラによるallocatable配列の最適化が不十分であるようだ.

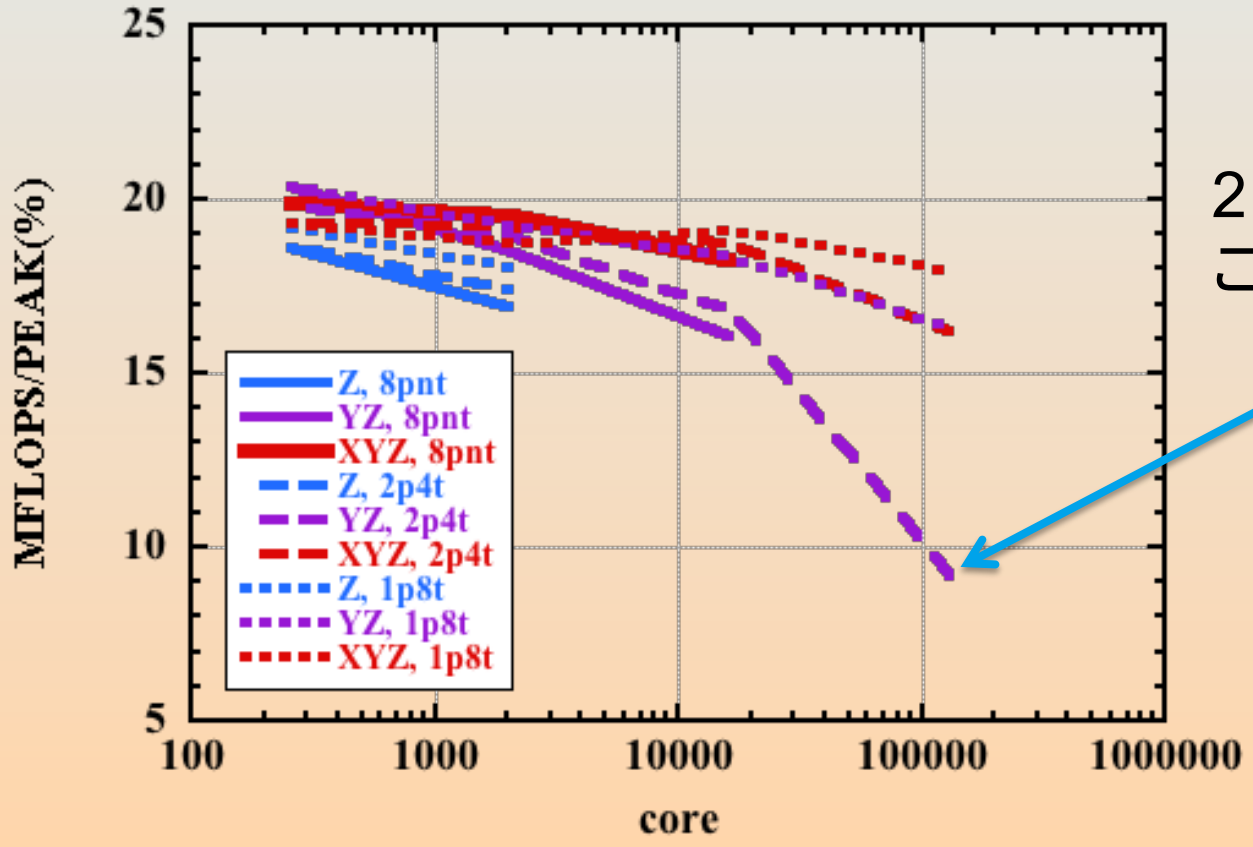
XMPコードの性能低下 ～allocatable array～

- allocatable配列同士で比較すると,MPIコードとXMPコードでは,ほぼ同等な性能が得られた.



MPIコードでのフラット並列

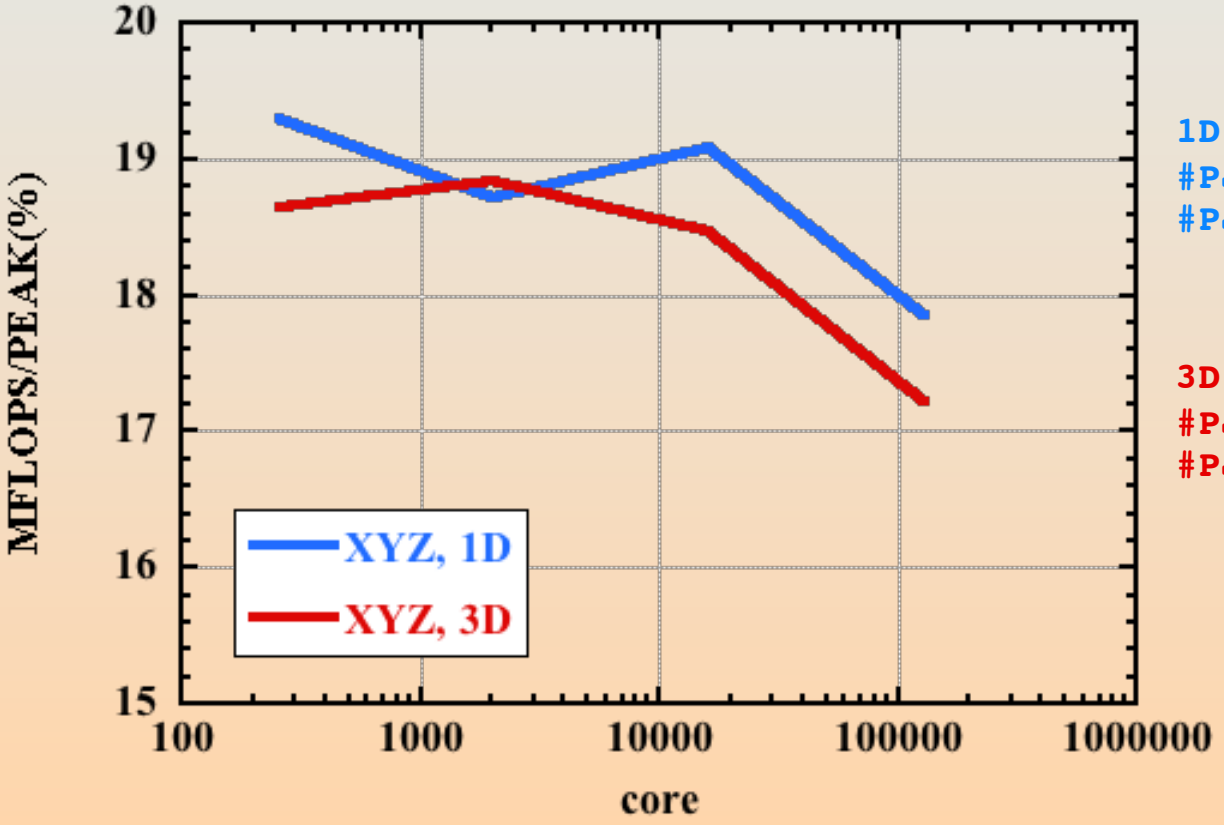
- 8プロセスのみ（フラット並列）,2プロセス4スレッド,1プロセス8スレッドの違いは,あまり大きくなかった.



2回計測したが,
これだけ低い.

MPIコードでのTOFUトポロジー

- プロセスのノードへの割当について,TOFUトポロジーを考慮したが,性能差は見られなかった.




```
1D
#PJM --rsc-list "node=2048"
#PJM --mpi "shape=2048,proc=2048"

3D
#PJM --rsc-list "node=16x16x8"
#PJM --mpi "shape=16x16x8,proc=2048"
```

まとめ

- 3次元流体コードをXMPによって並列化し,「京」上でMPIコードとの性能比較を行った.
- XMPコンパイラによって変換されたコードは,Fortranコンパイラの最適化を阻害する要因を含んでしまうため,性能低下が見られた.
- しかし,阻害要因を克服するコンパイラオプションを追加することにより,MPIコードと同等な性能が得られた.
  - ただし,静的配列を全てallocatable配列に変換することは,現時点では,コンパイラの最適化に大きな影響を与える.
  - Fortranコンパイラの改良 vs. XMPコンパイラでの対応?

*Part of the research was funded by MEXT's program for the Development and Improvement for the Next Generation Ultra High-Speed Computer System, under its Subsidies for Operating the Specific Advanced Large Research Facilities.