

東北大学サイバーサイエンスセンター講習会

はじめてのスパコン

～ スーパーコンピュータAOBAの紹介と利用法入門 ～



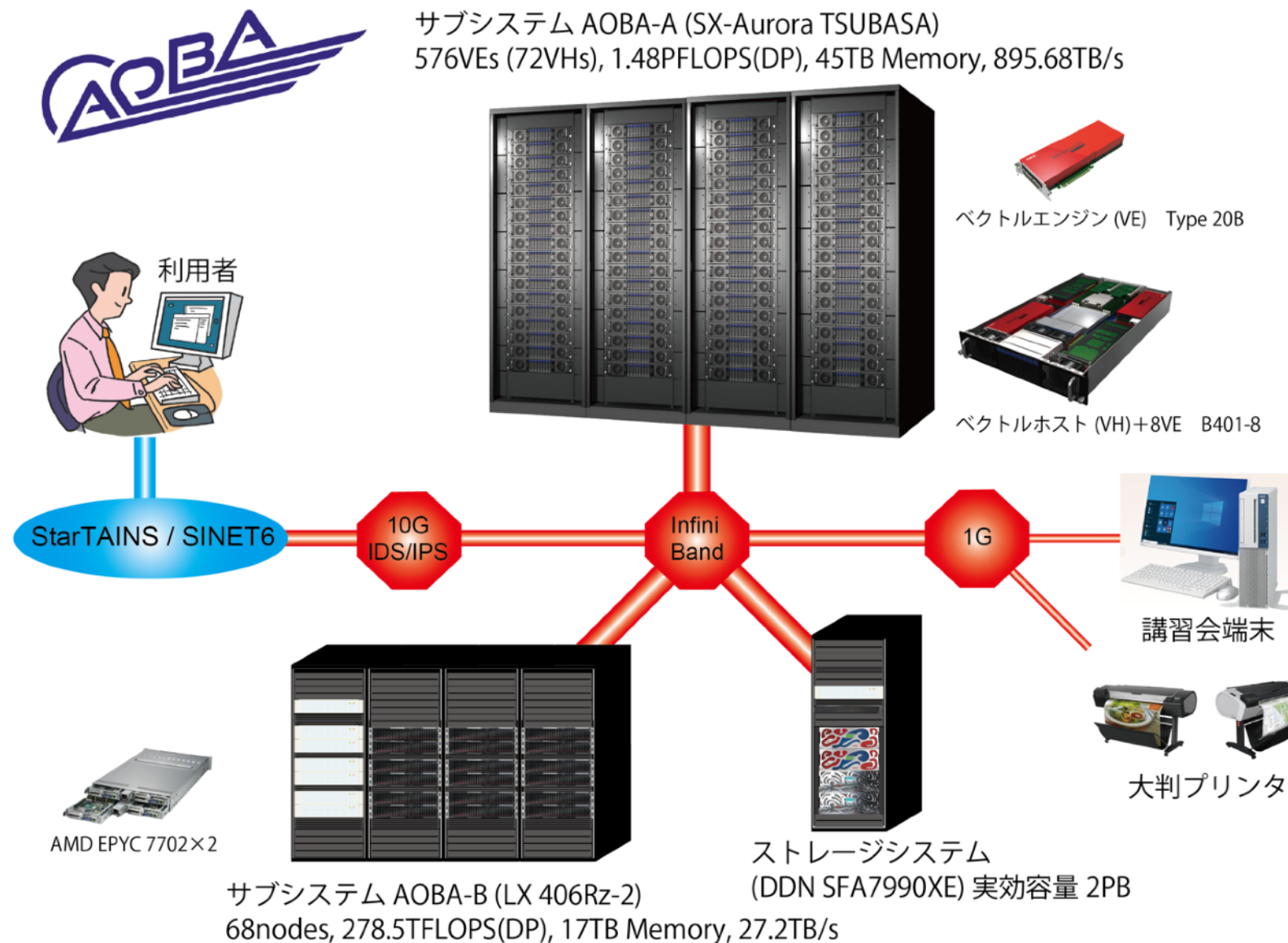
2022年5月26日

東北大学 サイバーサイエンスセンター
東北大学 情報部 情報基盤課

- スーパーコンピュータAOBAの紹介
- スーパーコンピュータとは？
- サイバーサイエンスセンターでの応用例
- 計算機の種類
- AOBAシステムの特徴
- AOBA-AとAOBA-Bどちらを使うか？
- 利用者向けウェブサイトとポータルサイト
- プログラム開発の流れ（実習）
- 並列実行とは
- プログラムのコンパイル方法
- ジョブの実行方法と実行キュー
- 負担金制度と利用申請方法
- 利用者講習会・利用相談・高速化支援

スーパーコンピュータAOBAの紹介

- サブシステムAOBA-A : SX-Aurora TSUBASA
- サブシステムAOBA-B : LX 406Rz-2
- ストレージシステム : DDN SFA7990XE (実効容量 2PB)
- 利用者向けサーバー : ログインサーバ, フロントエンドサーバ, ファイル転送サーバ
- センター内施設 : 講習会端末, 大判プリンタ (A0判)



- スーパーコンピュータ (Supercomputer、略称：スパコン) とは、内部の演算処理速度がその時代の一般的なコンピュータより非常に高速な計算機 (コンピュータ) のこと。
- HPCサーバ(High Performance Computing Server)と呼ばれることもある。

スーパーコンピュータとは？ (2/4)

- top500 (www.top500.org)
- 世界中の高性能なコンピュータシステムのランキングサイト
- 年2回 (6月と11月) 発表
- HPLベンチマークによるランキング
- 実アプリケーションとの性能乖離も指摘される

Flop/s (Floating point number Operations Per Second)

- 1秒間に浮動小数点演算が何回できるかを表す性能指標
- CPUの性能を表す際に用いられる

TOP500 LIST - NOVEMBER 2021

$R_{\max}$ and R_{peak} values are in TFlops. For more details about other fields, check the TOP500 description.

R_{peak} values are calculated using the advertised clock rate of the CPU. For the efficiency of the systems you should take into account the Turbo CPU clock rate where it applies.

←	1-100	101-200	201-300	301-400	401-500	→
---	-------	---------	---------	---------	---------	---

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438

スーパーコンピュータとは？ (3/4)

- HPCGベンチマーク (<http://hpcg-benchmark.org/>)
- 実アプリケーションの挙動に近い新しい指標
- 密行列、疎行列ベクトル計算をしており、実アプリケーションに近い多様な計算が含まれる

HPCG - NOVEMBER 2021

The TOP500 list has incorporated the High-Performance Conjugate Gradient (HPCG) Benchmark results, which provide an alternative metric for assessing supercomputer performance and is meant to complement the HPL measurement.

≡ HPCG Release

THE LIST

The HPCG results here are very similar to the last list. Fugaku was the clear winner with 16.0 HPCG-petaflops, while Summit retained its No. 2 spot with 2.93 HPCG-petaflops. Perlmutter, a USA machine housed at Lawrence Berkeley National Laboratory, took the No. 3 spot with 1.91 HPCG-petaflops.

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	HPCG (TFlop/s)
1	1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	16004.50
2	2	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	2925.75
3	5	Perlmutter - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	706,304	64,590.0	1905.44

スーパーコンピュータとは？（4/4）

- Green500(<http://www.green500.org/>)
- 電力効率の良い高性能計算機のランキング
- HPLベンチマークによる演算性能をシステム電力で除した、消費電力1Wあたりの性能

NOVEMBER 2021

The system to claim the No. 1 spot for the Green500 was MN-3 from Preferred Networks in Japan. Relying on the MN-Core chip and an accelerator optimized for matrix arithmetic, this machine was able to achieve an incredible 39.38 gigaflops/watt power-efficiency. This machine provided a performance 29.7-gigaflops/watt on the last list, clearly showcasing some impressive improvement. It also enhanced its standing on the TOP500 list, moving from No. 337 to No. 302.

Green500 Data

≡ Green500 Release

THE LIST

GREEN500 LIST (EXCEL)

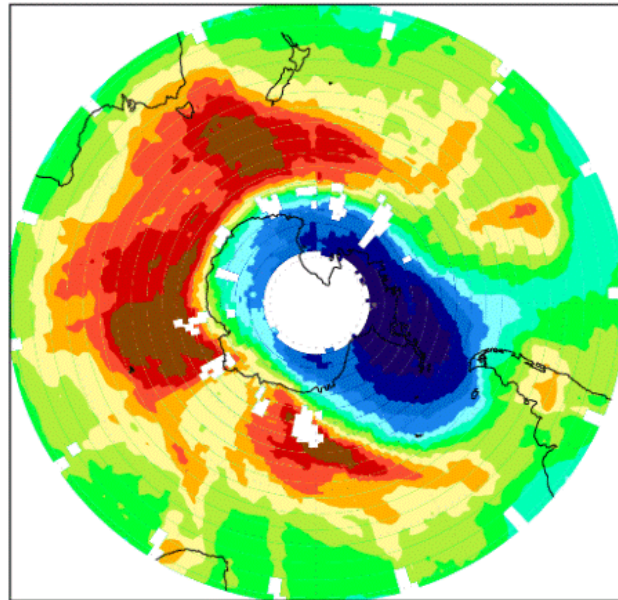
Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	301	MN-3 - MN-Core Server, Xeon Platinum 8260M 24C 2.4GHz, Preferred Networks MN-Core, MN-Core DirectConnect, Preferred Networks Japan	1,664	2,181.2	55	39.379
2	291	SSC-21 Scalable Module - Apollo 6500 Gen10 plus, AMD EPYC 7543 32C 2.8GHz, NVIDIA A100 80GB, Infiniband HDR200, HPE Samsung Electronics South Korea	16,704	2,274.1	103	33.983
3	295	Tethys - NVIDIA DGX A100 Liquid Cooled Prototype, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100 80GB, Infiniband HDR, Nvidia NVIDIA Corporation United States	19,840	2,255.0	72	31.538

- 航空・宇宙
- 気象・地球環境
- 電磁解析
- 分子化学・分子設計
- 原子力・核融合
- ライフサイエンス
- 資源探索、軍事利用、金融 等

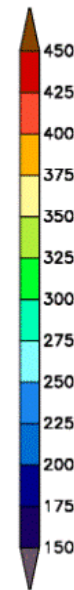
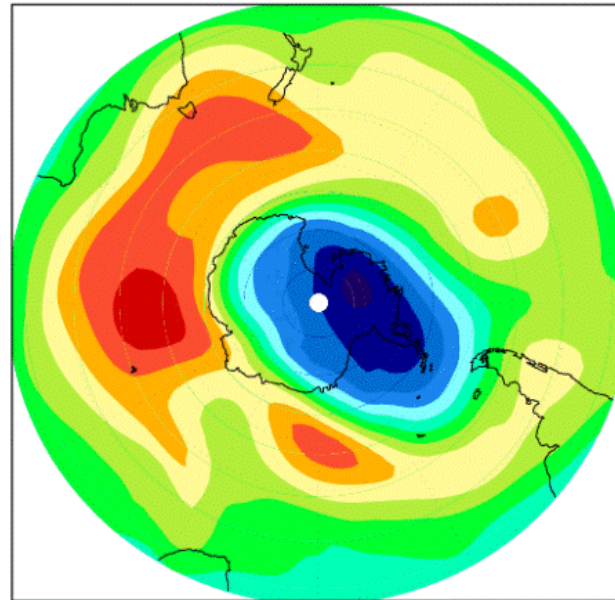
複雑な物理現象の数値シミュレーションに用いられる

気象 シミュレーション

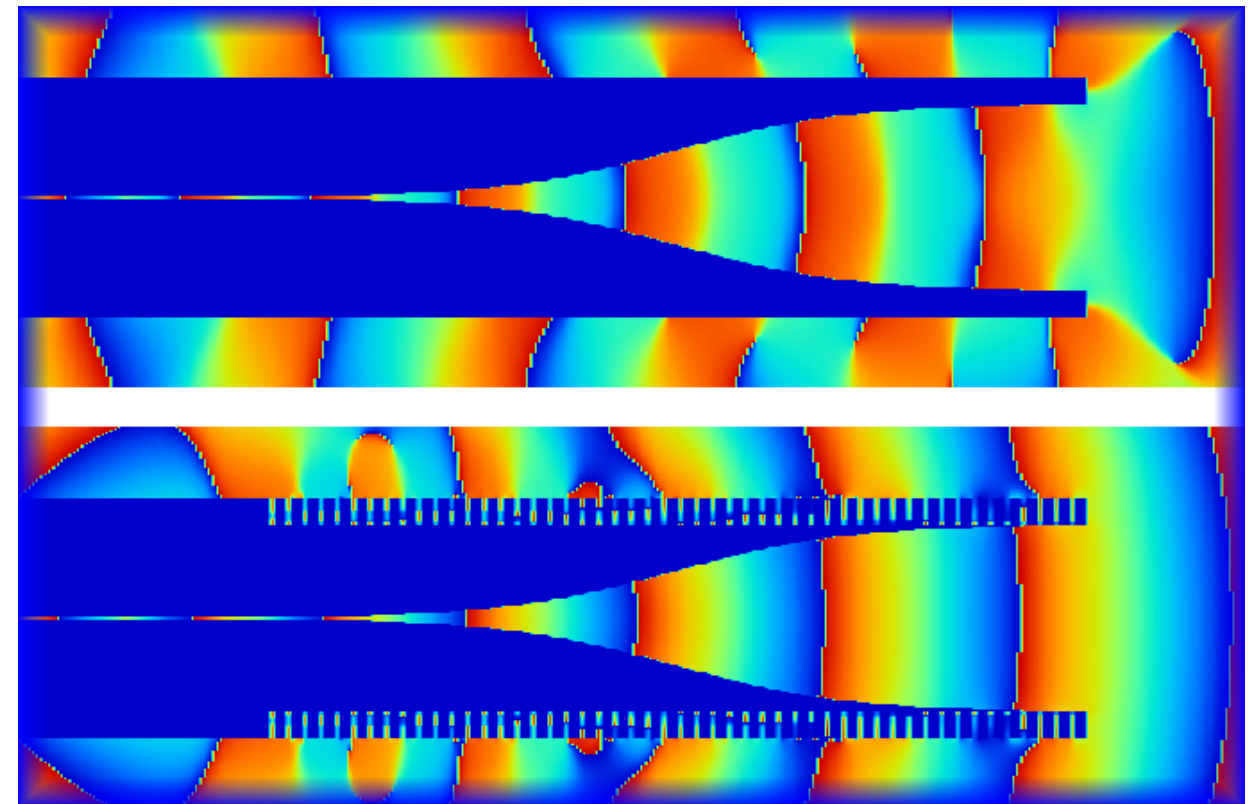
TOMS [DU] 2002 9/10



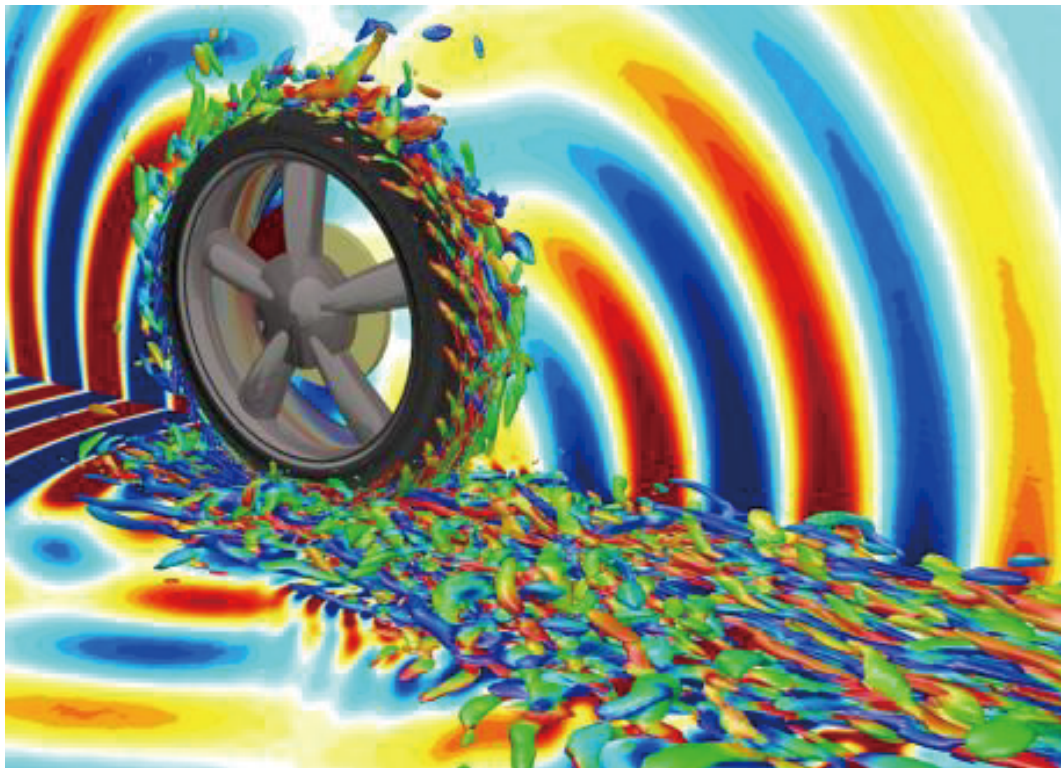
CTM(UVT) [DU] 09/10



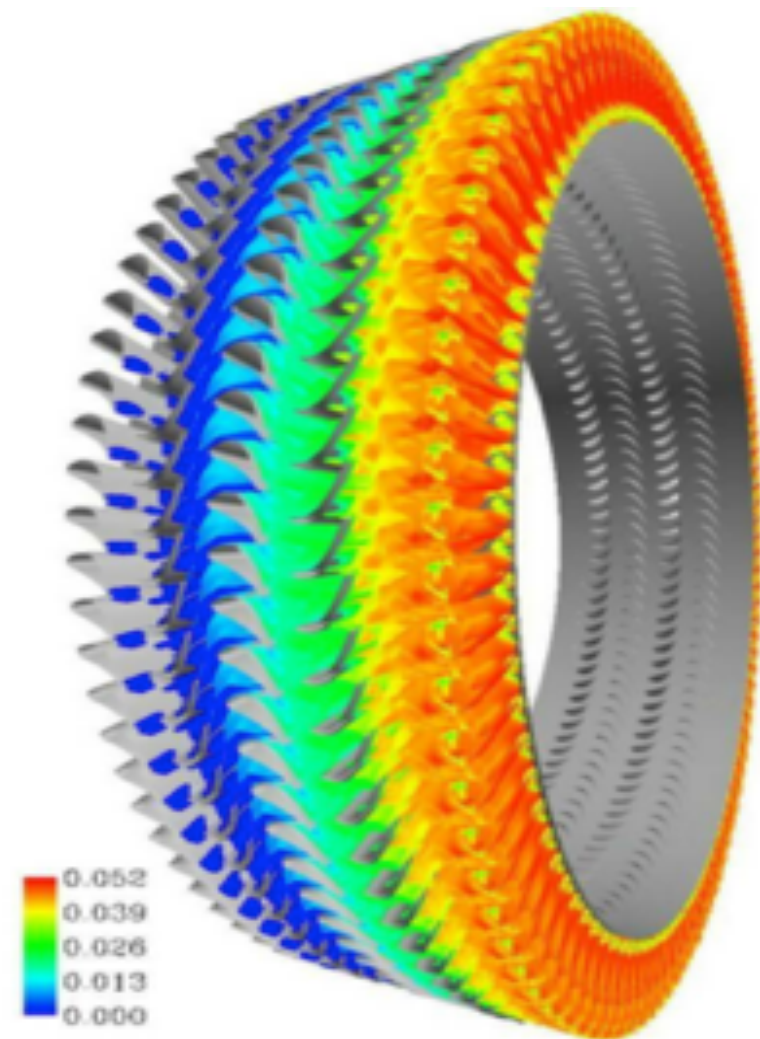
アンテナ開発



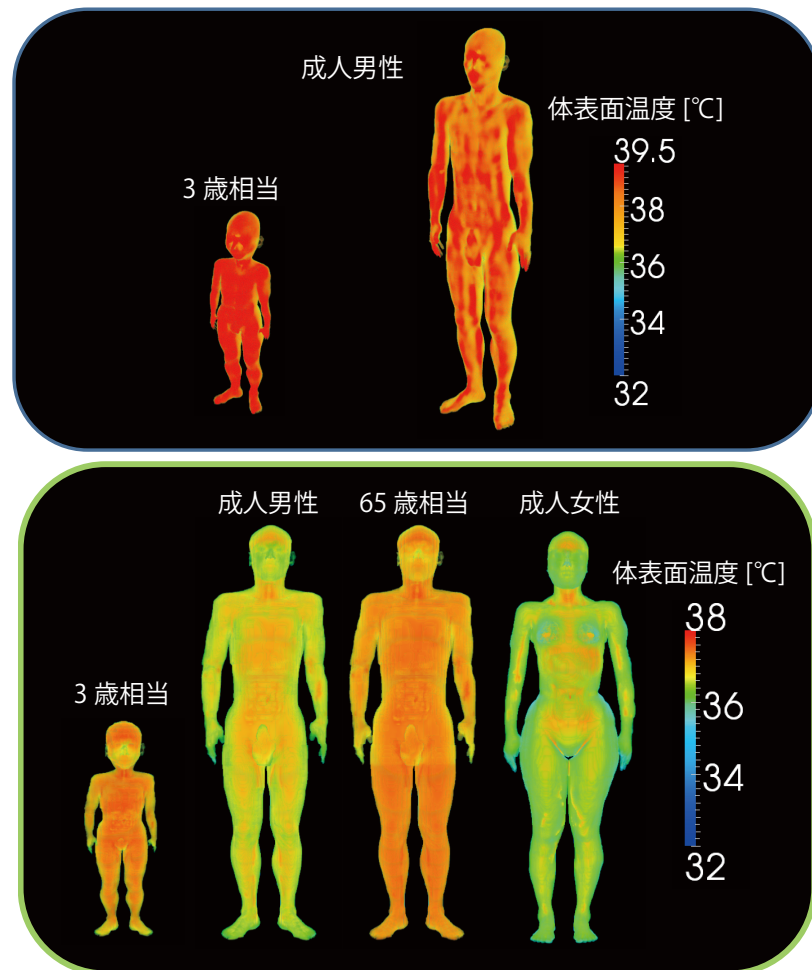
低騒音タイヤ開発



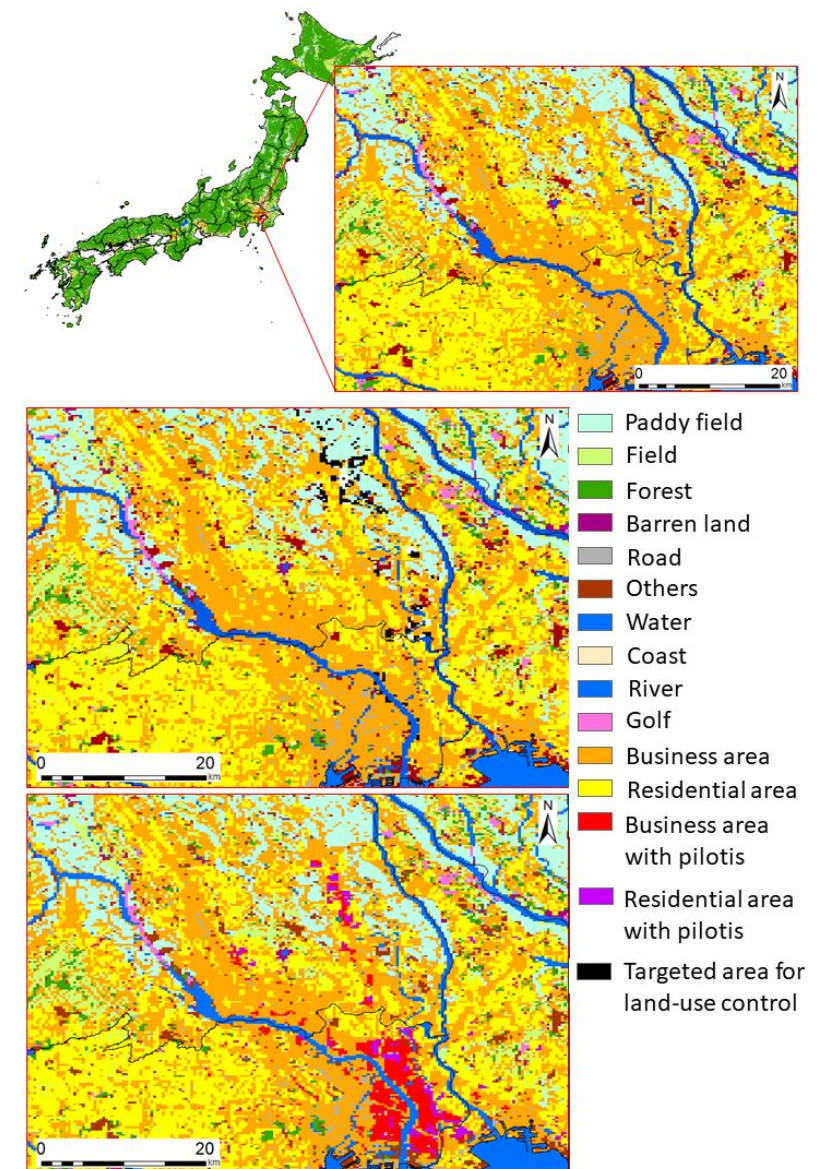
CO2削減 タービン設計



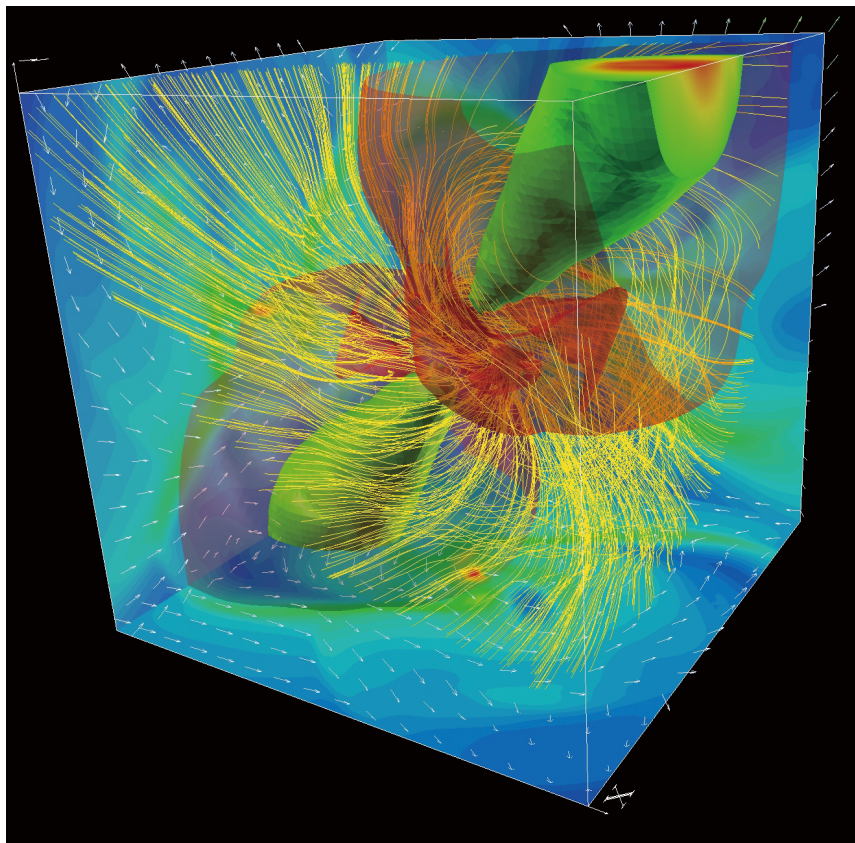
熱中症 シミュレーション



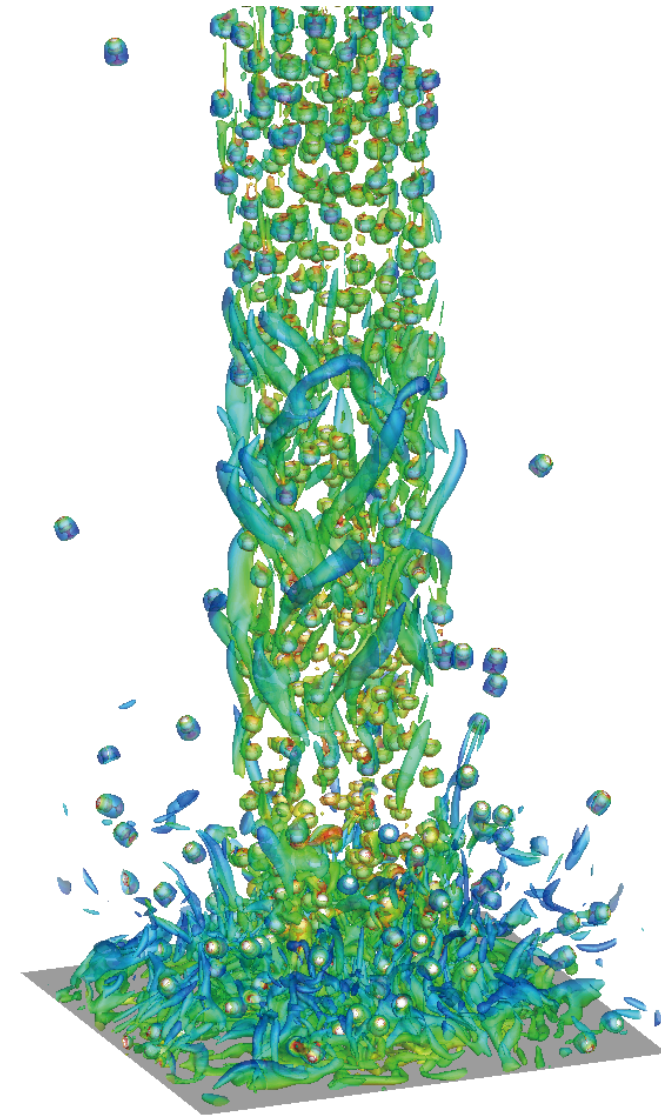
洪水被害予測 シミュレーション



星と惑星形成の
大規模シミュレーション



大規模混相流解析



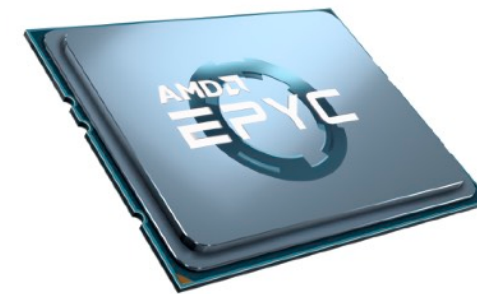
①プロセッサによる分類

a. ベクトル計算機



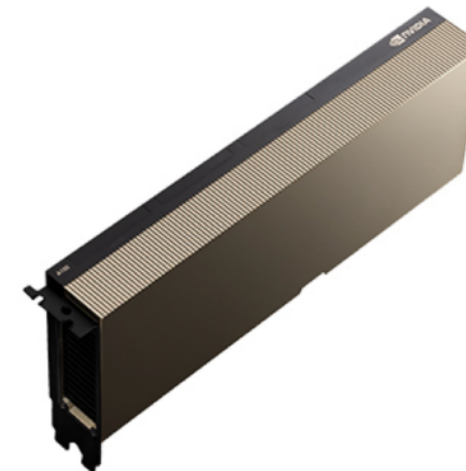
NEC SX-Aurora Type 20B
(AOBA-A)

b. スカラ計算機



AMD EPYC 7720
(AOBA-B)

c. GPU(Graphics Processing Unit)



NVIDIA A100

②メモリ構成による分類

a. 共有メモリ型

b. 分散メモリ型

①-a. ベクトル計算機

- ベクトル演算のための専用ハードウェアを持つプロセッサを搭載した計算機
- SXシリーズ (NEC) 、地球シミュレータ (NEC)
- 1980年代から1990年代にかけて、スパコンと言えばベクトル計算機のことだった
- SX-Aurora TSUBASAはx86 CPUとの組み合わせが必要
- AOBA-Aに採用

①-b. スカラ計算機

- 汎用プロセッサを搭載した計算機
- 汎用プロセッサは Xeon(Intel), EPYC(AMD), POWER(IBM), ARM(ARM)等、普及品化しているプロセッサ
- 1990年代中盤以降、安価な汎用プロセッサを複数搭載した並列計算機が主流となる
- HPCサーバ、PCクラスタ
- AOBA-Bに採用

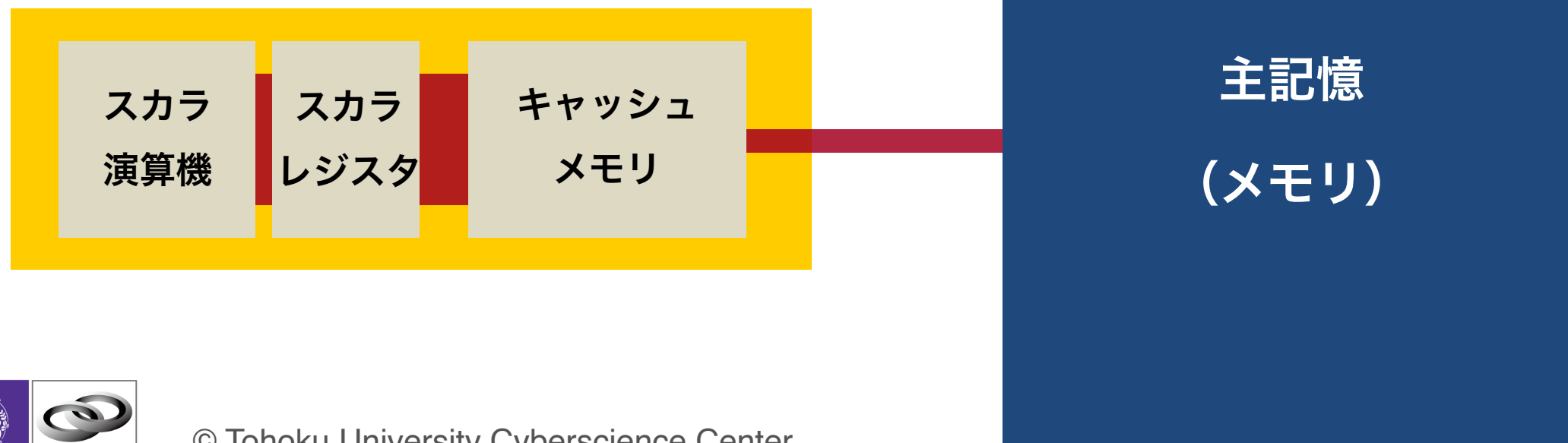
①-c. GPU

- コンピュータゲームに代表されるリアルタイム画像処理に特化した演算装置
- GPUのハードウェアを、より一般的な計算に活用
- AI00(NVIDIA), INSTINCT(AMD)
- 映像出力端子を持たない専用製品や、深層学習ベースのAI向けに特化した演算器を搭載
- 汎用CPUとの組み合わせが必要

■ベクトルプロセッサ

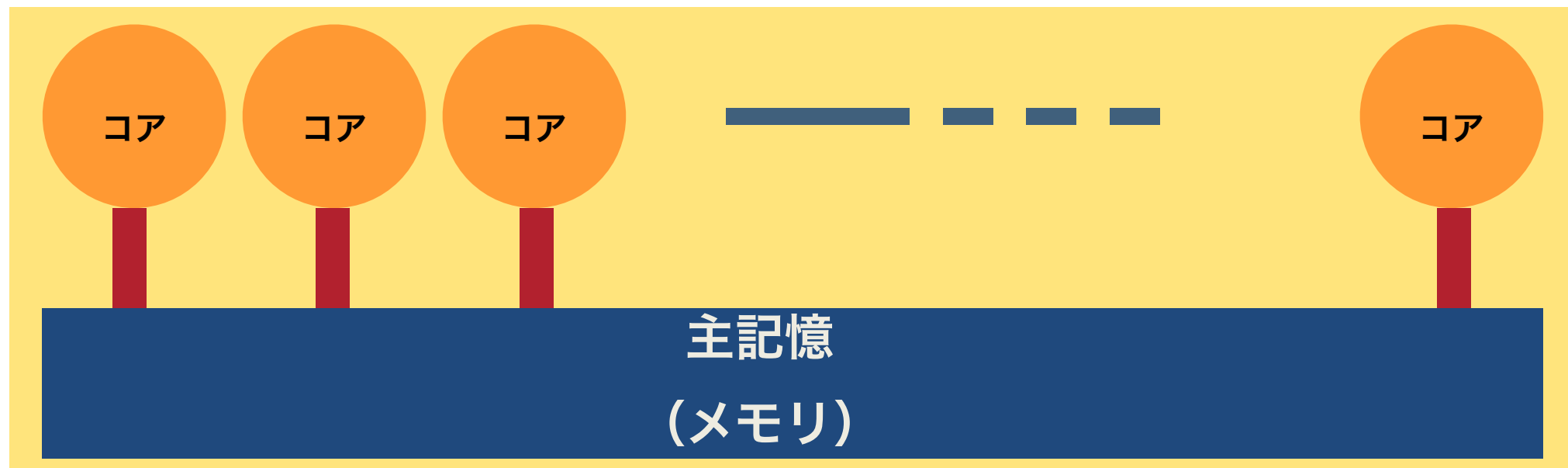


■ス칼ラプロセッサ



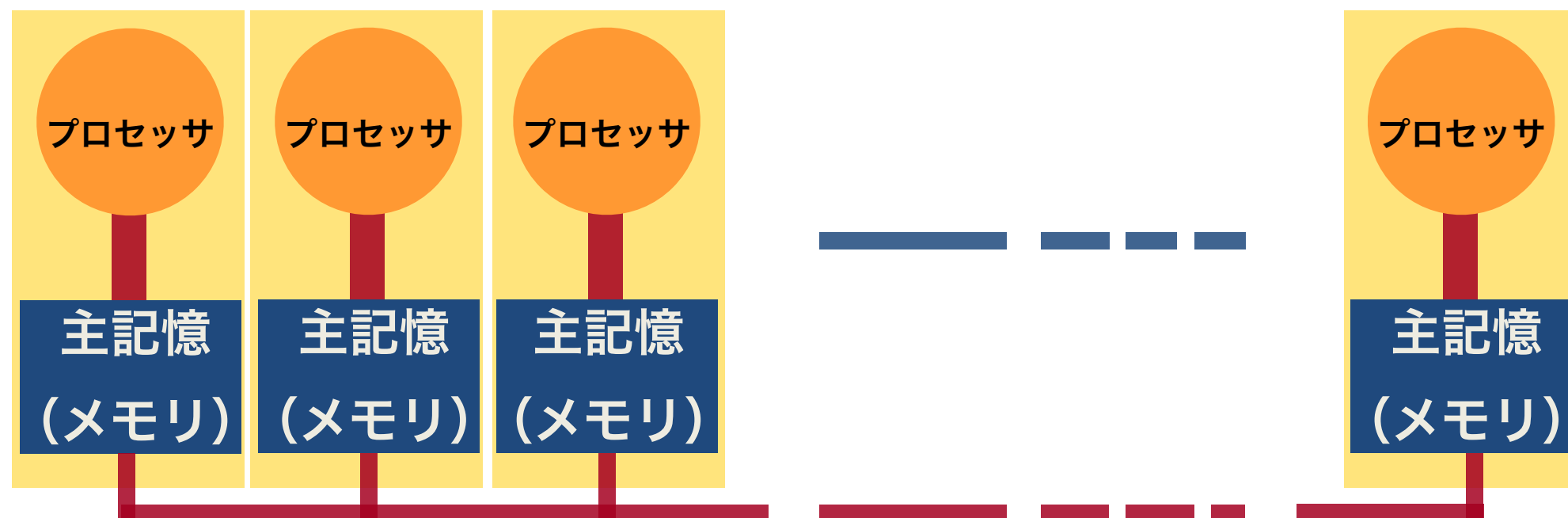
②-a. 共有メモリ型

- 1つの物理メモリを複数のプロセッサで共有するタイプ (SMP : Symmetric Multiple Processor)
- 自動並列化およびOpenMPによる並列化実行が可能
- MPI(Message Passing Interface)による並列化実行も可能
- AOBA-Aの各**VE内**並列、AOBA-Bの各**ノード内**並列



②-b. 分散メモリ型

- プロセッサごとにローカルな物理メモリを持ち、それらをInfiniBandなどのネットワークで複数接続するタイプ
- MPIにより並列実行する
- 自動並列化およびOpenMPによる並列化実行も同時利用が可能
- AOBA-Aの**複数VE**並列、AOBA-Bの**複数ノード**並列



サブシステムAOBA-A : SX-Aurora TSUBASA 【ハードウェア】

- ベクトルプロセッサ+x86/Linuxアーキテクチャの構成

ベクトルエンジン (**VE**) は演算処理を行う

→ 本システムでは Type 20B を8個搭載

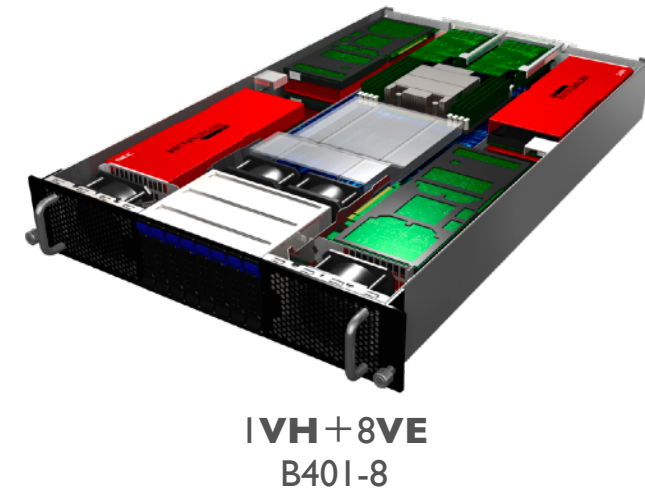
ベクトルホスト (**VH**) はOS処理, VE制御, I/O制御等を行う

→ 本システムでは EPYC 7402P を1個搭載

基本システム単位は 1**VH** + 8**VE**

システム全体では 72**VH**+576**VE** 最大利用は32**VH**+256**VE**

ノード間接続は InfiniBand HDR (200Gbps) ×2



- VE**はSX-ACEを継承するベクトルアーキテクチャ

マルチコアベクトルプロセッサ (8コア) とHBM2メモリを搭載

ベクトル演算による高い演算性能: **VE**あたり 2.45TFLOPS (DP) 4.91TFLOPS (SP)

コア間共有メモリ: **VE**あたり 48GB

高いメモリバンド幅: **VE**あたり 1.53TB/s

演算性能とデータ転送性能のバランス: 0.65B/F

- VH**のLinux OS環境とVEを連携した利用が可能

24コア 1.075TFLOPS (DP) 256GBメモリを搭載 (利用者ジョブは16コア、136GBまで)

【**VH**でプリ処理 → **VE**で演算処理 → **VH**でポスト処理】を1ジョブで完結するなどの利用が可能

- ・ 自動ベクトル化・自動並列化機能機能を備えた, Fortran/C/C++コンパイラを利用可能
x86向けに開発されたFortran/C/C++ソースコードも, コンパイラがベクトル性能を引き出す
SX-ACE向けに開発されたアプリケーションの移植もサポート
MPIライブラリによる分散メモリ並列実行に対応
GNU互換環境を装備 (SX-ACE向けコンパイラからオプション, 指示行に仕様の変更あり)
- ・ ベクトルアーキテクチャに最適化された科学技術計算ライブラリ (Fortran/C)
BLAS, FFTW, LAPACK, ScaLAPACKのインターフェースをそのまま利用可能
ASLインターフェースも利用可能
- ・ 実行時性能解析ツールを利用可能
PROGING, FTRACE, Ftrace Viewer
- ・ 一部の量子化学分野のアプリもインストール済
Quantum Espressoの一部アプリを**VE**でも利用可能

サブシステムAOBA-B : LX 406Rz-2 【ハードウェア・ソフトウェア】

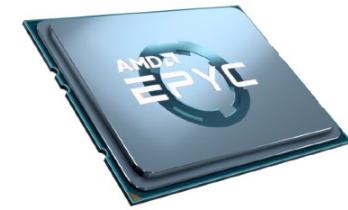
- x86/Linuxアーキテクチャ構成

1ノードにAMD EPYC 7702 (64コア) を2個, 256GBメモリを搭載

ノード性能は 4.096TFLOPS (DP) 8.192TFLOPS (SP) 409.6GB/s

システム全体では68ノード 最大利用は16ノード

ノード間接続は InfiniBand HDR (200Gbps) ×1



AMD EPYC 7702

- AMDコンパイラ, GNUコンパイラ, Intelコンパイラが利用可能

AOCC (AMD Optimizing C/C++ Compiler, Fortran) , OpenMPI

GNU Compiler Collection, OpenMPI

Intel OneAPI ベース&HPCツールキット (ライセンス数限定)

- Linux OSに対応したアプリを準備中

Gaussian16, GRRM17, Quantum Espresso, OpenFOAM

(利用者限定商用アプリ) Mathematica, MATLAB

RISTが整備する国プロアプリ

他OSSなどもユーザ領域にインストール可能



LX 406Rz-2 4ノード

	AOBA-A (SX)	AOBA-B (LX)
利用 最小 単位	1VE	1ノード
コア数	8	128
理論演算性能 [TFLOPS]	2.45	4.09
メモリ容量 [GB]	48	256
コア性能 [GFLOPS]	307	32
メモリ転送性能 [TB/s]	1.53	0.40

	AOBA-A (SX)	AOBA-B (LX)
利用 最大 単位	256VE	16ノード
コア数	2,048	2,048
理論演算性能 [TFLOPS]	627.2	65.5
メモリ容量 [TB]	12	4

AOBA-Aを選択

- ・ シングルコア実行のプログラム
- ・ メモリ転送性能が律速になるプログラム
- ・ MPIによる大規模並列を行う場合

AOBA-Bを選択

- ・ ノード内並列化（OpenMP並列・自動並列）のみされているプログラム
- ・ ノード内でメモリ容量を多く使うプログラム

AOBA-Aを選択

- ・ SX-ACEで利用していたプログラム
- ・ ベクトル化率が99.0%以上, 平均ベクトル長が128以上のプログラム
 - 実行時性能の取得方法と, さらなるベクトル高速化については利用相談をご利用下さい。
- ・ お試しで使いたい
 - IVEを1時間まで利用できる無料のキューがあります。(同時実行数1)

AOBA-Bを選択

- ・ 商用アプリ, OSSなどソースコードを改変しにくいプログラム
- ・ ベクトル性能が出ないと分かっているプログラム
- ・ AOBA-A向けにコンパイル出来ないプログラム
- ・ Fortran/C/C++以外のコンパイラを使用するプログラム
- ・ Gaussian16, GRRM17, MATLAB (バッチ処理), OpenFOAMを使う場合

両方で実行を試した後に選択

- ・ AOBA-A, AOBA-Bの両方で実行できるアプリケーション (Quantum Espresso6.3のpw.x)

利用者向けウェブサイトとポータルサイト

- ・サイバーサイエンスセンター 大規模科学計算システムのウェブサイト <https://www.ss.cc.tohoku.ac.jp/>

- システムの利用マニュアル
- 運用についてのお知らせ
- 利用相談などの連絡先
- 講習会予定

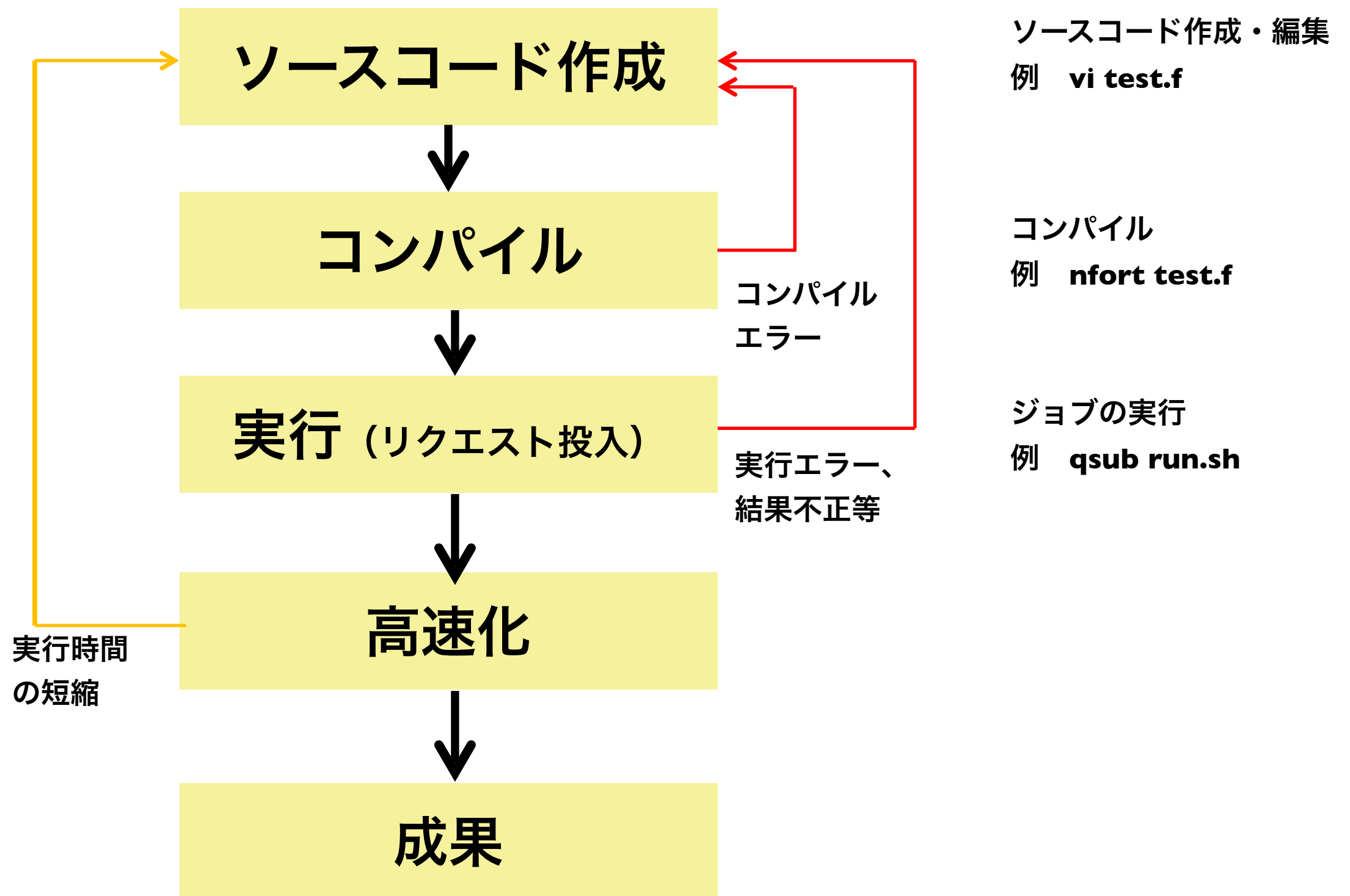
- ・利用申請からログインまでについては以下を参照

<https://www.ss.cc.tohoku.ac.jp/first-use/>

- ・利用者用ポータルサイト（LDAP認証連携）

- 公開鍵・秘密鍵ペアの作成
- 利用状況（負担金，合計課金対象時間，ジャーナルレコード等）の確認





逐次実行

front\$ (プロンプト) に続くコマンドを入力します。

【ソースコード】

```
ソースコードのコピー (ディレクトリごと)
front$ cd ~
front$ cp -r /mnt/stfs/ap/lecture/super/prog1 ./
```

【コンパイル】

```
AOBA-A (SX) 向けにコンパイル
front$ cd prog1
front$ nfort vec.f90 (逐次実行Fortranプログラム)
      (コンパイルメッセージが表示)
front$ ls (実行ファイル a.out が作成されていることを確認)
```

【リクエスト投入】

```
AOBA-A (SX) にバッチリクエストファイルの投入
front$ qsub run.sh
      (投入先のプロジェクトコードを確認)
```

【リクエストの状況確認】

```
バッチリクエストの実行待ち、実行中を確認
front$ reqstat

      (ジョブが終了するとなにも表示されません)
```

【結果の確認】

標準出力ファイルの確認
front\$ ls（標準出力ファイル名の確認）
front\$ cat run.sh.o12345

vc(1,1)= 5.1393758152283769E+05

【実効性能の確認】

標準エラー出力ファイルの確認
front\$ ls（標準エラー出力ファイル名の確認）
front\$ cat run.sh.e12345

```
***** Program Information *****
Real Time (sec) : 0.571793
User Time (sec) : 0.564301
Vector Time (sec) : 0.563507
Inst. Count : 698325691
V. Inst. Count : 318832677
V. Element Count : 81621164350
V. Load Element Count : 5368709152
FLOP Count : 65498251483
MOPS : 187271.718039
MOPS (Real) : 184722.874918
MFLOPS : 116129.751073
MFLOPS (Real) : 114549.178629
A. V. Length : 255.999997
V. Op. Ratio (%) : 99.640710
L1 Cache Miss (sec) : 0.000299
CPU Port Conf. (sec) : 0.000000
V. Arith. Exec. (sec) : 0.495812
V. Load Exec. (sec) : 0.067678
VLD LLC Hit Element Ratio (%) : 0.009778
FMA Element Count : 23622320128
Power Throttling (sec) : 0.000000
Thermal Throttling (sec) : 0.000000
Memory Size Used (MB) : 33342.000000
Non Swappable Memory Size Used (MB) : 96.000000

Start Time (date) : Wed May 25 08:54:57 2022 JST
End Time (date) : Wed May 25 08:54:57 2022 JST
```

自動並列実行

【ソースコード】

ソースコードのコピー (ディレクトリごと)

```
front$ cd ~
```

```
(front$ cp -r /mnt/stfs/ap/lecture/super/prog1 ./) (逐次実行プログラムと同じ)
```

【コンパイル】

AOBA-A (SX) 向けにコンパイル

```
front$ cd prog1
```

```
front$ nfort -mparallel vec.f90 (自動並列実行Fortranプログラム)
```

(コンパイルメッセージが表示)

```
front$ ls (実行ファイル a.out が作成されていることを確認)
```

【リクエスト投入】

AOBA-A (SX) にバッチリクエストファイルの投入

```
front$ qsub run.sh
```

(投入先のプロジェクトコードを確認)

【リクエストの状況確認】

バッチリクエストの実行待ち、実行中を確認

```
front$ reqstat
```

(ジョブが終了するとなにも表示されません)

OpenMP並列実行

【ソースコード】

ソースコードのコピー (ディレクトリごと)

```
front$ cd ~  
front$ cp -r /mnt/stfs/ap/lecture/super/prog2 ./
```

【コンパイル】

AOBA-A (SX) 向けにコンパイル

```
front$ cd prog2  
front$ nfort -fopenmp omp.f90 (OpenMP並列実行Fortranプログラム)  
(コンパイルメッセージが表示)  
ls (実行ファイル a.out が作成されていることを確認)
```

【リクエスト投入】

AOBA-A (SX) にバッチリクエストファイルの投入

```
front$ qsub run.sh  
(投入先のプロジェクトコードを確認)
```

【リクエストの状況確認】

バッチリクエストの実行待ち、実行中を確認

```
front$ reqstat  
  
(ジョブが終了するとなにも表示されません)
```

MPI並列実行

【ソースコード】

ソースコードのコピー (ディレクトリごと)

```
front$ cd ~  
front$ cp -r /mnt/stfs/ap/lecture/super/prog3 ./
```

【コンパイル】

AOBA-A (SX) 向けにコンパイル

```
front$ cd prog3  
front$ mpinfort mpi.f90 (MPI並列実行Fortranプログラム)  
(コンパイルメッセージが表示)  
ls (実行ファイル a.out が作成されていることを確認)
```

【リクエスト投入】

AOBA-A (SX) にバッチリクエストファイルの投入

```
front$ qsub run.sh  
(投入先のプロジェクトコードを確認)
```

【リクエストの状況確認】

バッチリクエストの実行待ち、実行中を確認

```
front$ reqstat  
  
(ジョブが終了するとなにも表示されません)
```

計算機と実行形式と演算結果および演算時間の比較

AOBA-A (SX) 1VE 8core 2.45TFLOPS	コンパイルコマンド	全体演算時間 [sec]	演算部のみ[sec]
逐次実行	nfort vec.f90	0.567	—
自動並列実行(8core)	nfort -mparallel vec.f90	0.161	—
OpenMP並列実行(8core)	nfort -fopenmp omp.f90	0.195	—
MPI並列実行 (1VE)	mpinfort mpi.f90	0.491	0.133
MPI並列実行 (8VE)	mpinfort mpi.f90	0.744	0.014

AOBA-B (LX) 1node 128core 4.096TFLOPS	コンパイルコマンド ※	全体演算時間 [sec]	演算部のみ[sec]
逐次実行	ifort vec.f90 -mcmodel=medium -march=core-avx2	7.093	—
自動並列実行(128core)	ifort -parallel vec.f90 -mcmodel=medium -march=core-avx2	0.872	—
OpenMP並列実行(128core)	ifort -fopenmp omp.f90 -march=core-avx2	1.611	—
MPI並列実行(1node)	mpiifort mpi.f90 -march=core-avx2	13.303	1.232
MPI並列実行(2node)	mpiifort mpi.f90 -march=core-avx2	29.822	0.599

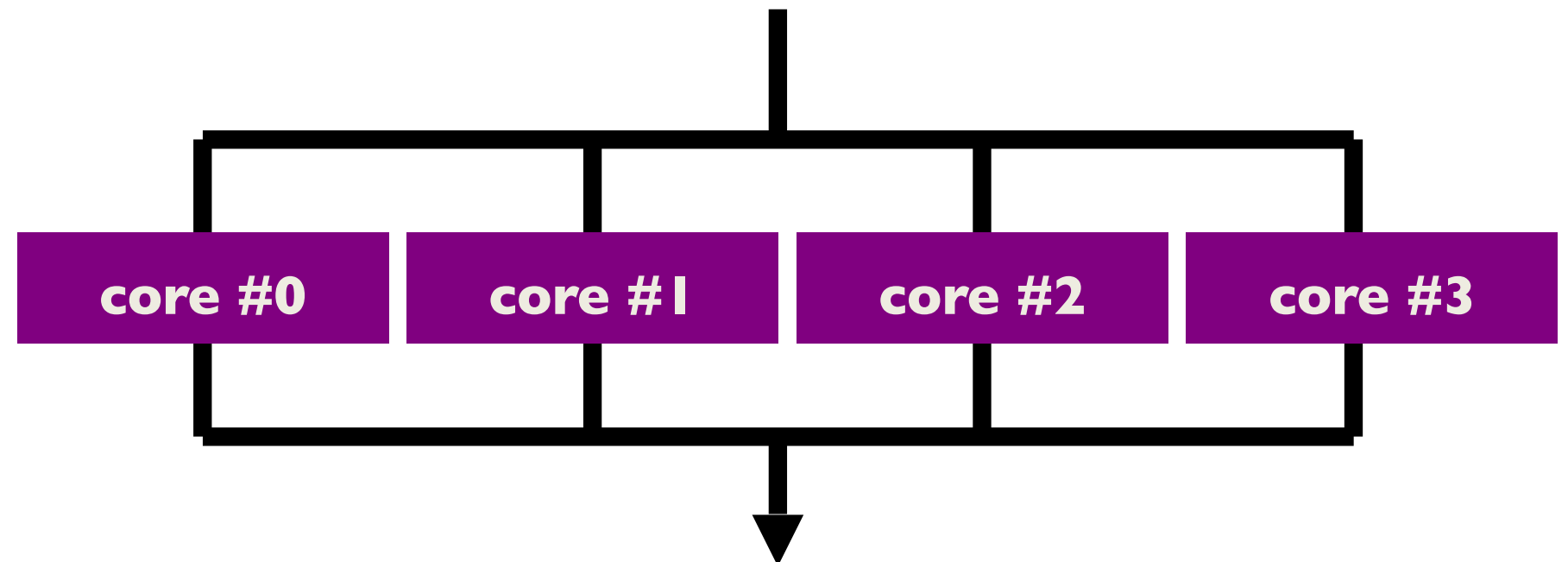
※ Intel OneAPIの利用は、bash環境 (/bin/bashの実行) で
source /opt/oneapi/setvars.sh intel64 コマンドの実行が必要

- 演算範囲を複数のプロセッサ（コア）で分割し、時間的に並列処理すること

逐次実行



4並列実行



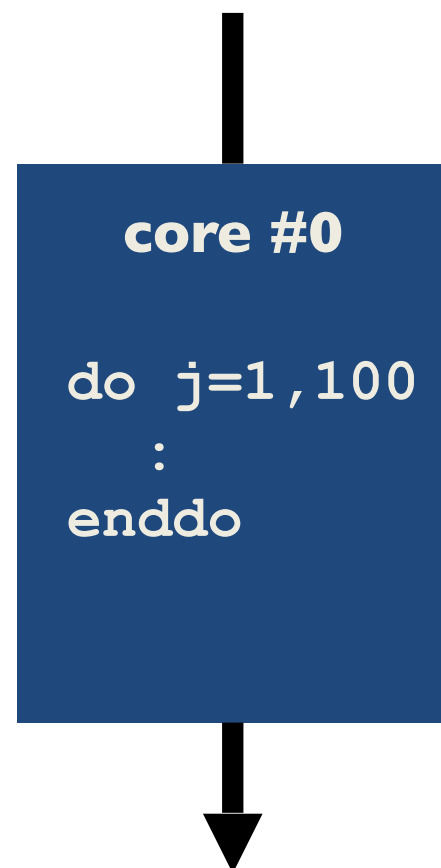
• 自動並列化の例

```
do j=1,100  
  do i=1,2048  
    a(i,j)=b(i,j)+c(i,j)  
  enddo  
enddo
```

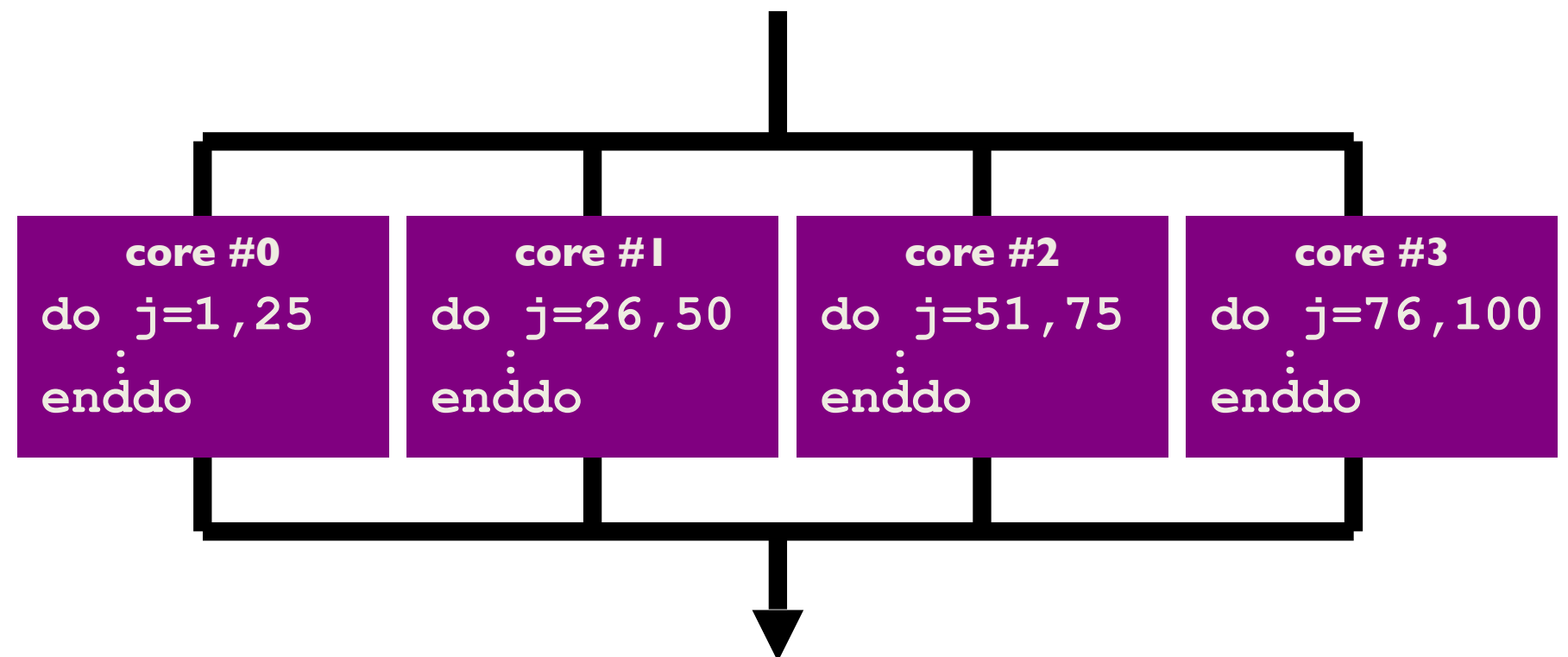
基本方針

内側のループはベクトル実行
外側のループを並列化

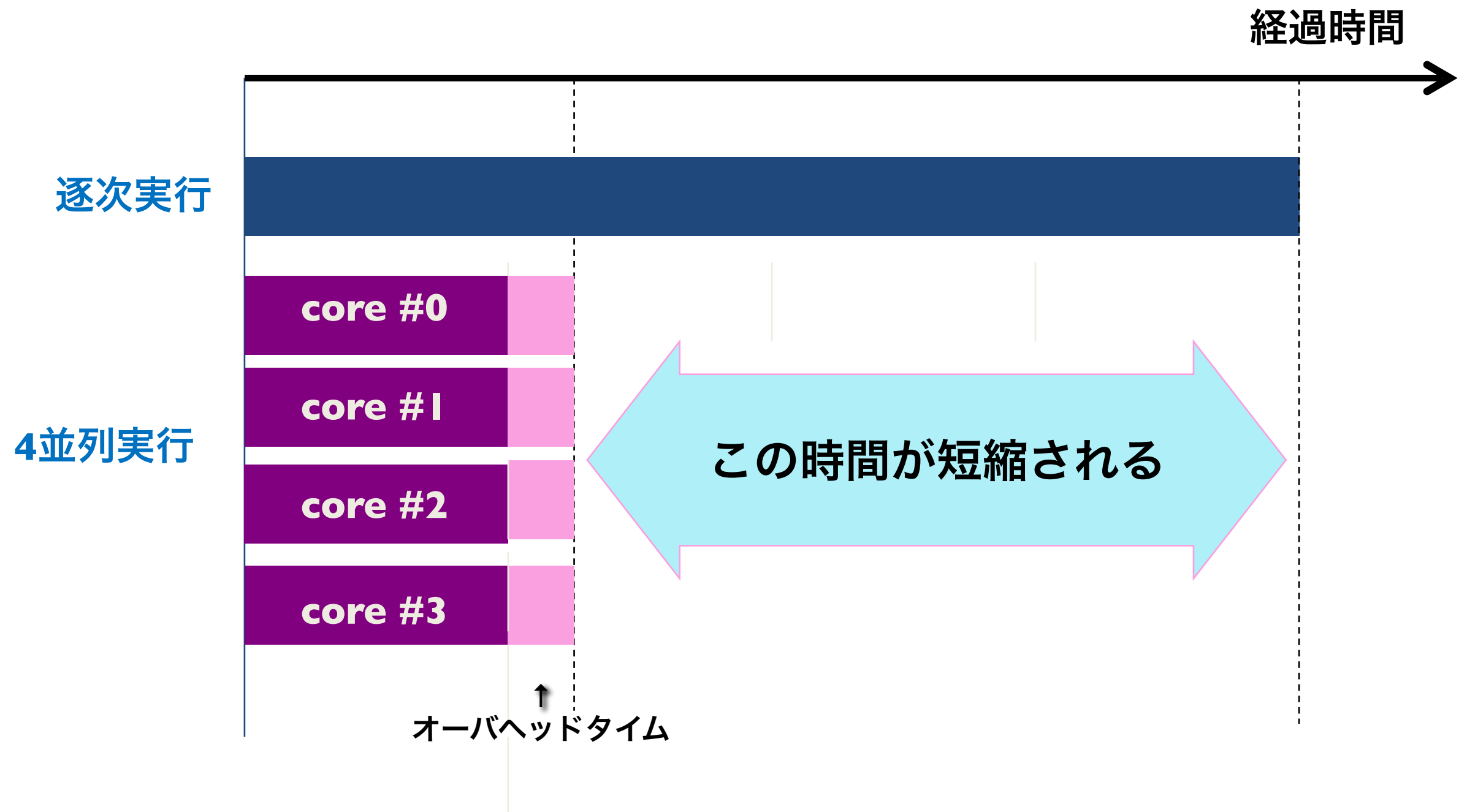
逐次実行



4並列実行



- CPU時間が短縮されるのではなく、経過時間が短縮される



- 並列処理の方法

- 自動並列

- コンパイラが並列可能な箇所（ループ）を見つけ自動的に並列化する。
実行並列数は、割り当てられたプロセッサ数で決定する。

- OpenMP

- ユーザが、ソースコード中の並列化する箇所に指示文（ディレクティブ）を指定する。実行並列数は、割り当てられたプロセッサ数で決定する。

- MPI(Message Passing Interface)

- メッセージ交換用ライブラリによりプロセッサ間データ通信を行う。
データの分割、処理方法等の並列処理の手順を、ユーザが明示的にプログラミングしなければならない。

• 並列処理の特徴

	既存のソースコード (逐次処理用) を 利用可能か？	並列化に要する作業	複数ノードでの実行 が可能か？	その他
自動並列	○	ない コンパイルオプション を付ける	× MPI並列化が必要	コンパイラまかせのためお手軽だが、高効率 で実行するにはユーザの一手間が必要な場合 あり。センターで最も利用されている並列化 手法。
OpenMP	指示行の挿入が必要	ある 並列化可能な箇所をユ ーザが判断し、ソース コードに指示文を追加 する	× MPI並列化が必要	ユーザの指示により並列化を行うため、意図 したとおりに動作する。演算結果の検証が必 要。自動並列の手動版的な手法。
MPI	MPI用のソースコードに 改変が必要	多い MPI用のソースコードを 作成する必要あり	○	MPI用プログラムを作成する必要があるが、大 規模並列化には不可欠。分散メモリ、共有メ モリともに利用可能。

- 自動並列化の条件

- 自動並列化の対象ループで、かつそのループ中の文、データ型、演算が自動並列化対象であること
- ループ中のデータに依存関係がないこと
- 並列化により演算順序が変わっても正しい演算結果となること
- 並列化によって性能向上が期待できること

NEC Software Development kit for Vector Engine

・ 逐次実行

front\$ **nfort** コンパイルオプション Fortranソースファイル名

front\$ **ncc** コンパイルオプション Cソースファイル名

front\$ **nc++** コンパイルオプション C++ソースファイル名

・ 自動並列化

front\$ **nfort -mparallel** コンパイルオプション Fortranソースファイル名

front\$ **ncc -mparallel** コンパイルオプション Cソースファイル名

front\$ **nc++ -mparallel** コンパイルオプション C++ソースファイル名

・ OpenMP並列化

front\$ **nfort -fopenmp** コンパイルオプション Fortranソースファイル名

front\$ **ncc -fopenmp** コンパイルオプション Cソースファイル名

front\$ **nc++ -fopenmp** コンパイルオプション C++ソースファイル名

・ MPI並列化 (自動並列化, OpenMP並列化の併用も可能)

front\$ **mpinfort** コンパイルオプション Fortranソースファイル名

front\$ **mpincc** コンパイルオプション Cソースファイル名

front\$ **mpinc++** コンパイルオプション C++ソースファイル名

AMD Optimizing C/C++ Compiler (AOCC)

・ 逐次実行

front\$ **flang** コンパイルオプション Fortranソースファイル名

front\$ **clang** コンパイルオプション Cソースファイル名

front\$ **clang++** コンパイルオプション C++ソースファイル名

・ OpenMP並列化

front\$ **flang -fopenmp** コンパイルオプション Fortranソースファイル名

front\$ **clang -fopenmp** コンパイルオプション Cソースファイル名

front\$ **clang++ -fopenmp** コンパイルオプション C++ソースファイル名

・ MPI並列化 (OpenMPIを利用, OpenMP並列化の併用も可能)

front\$ **mpifort** コンパイルオプション Fortranソースファイル名

front\$ **mpicc** コンパイルオプション Cソースファイル名

front\$ **mpic++** コンパイルオプション C++ソースファイル名

最適化のコンパイルオプション -march=znver2 でRome向け最適化コンパイル

Intel OneAPI ベース&HPCツールキット

Intelコンパイラ環境に切り替えるために、bash環境で以下のコマンドを実行する必要がある

```
front$ source /opt/oneapi/setvars.sh intel64
```

- ・ 逐次実行

```
front$ ifort コンパイルオプション Fortranソースファイル名
```

```
front$ icc コンパイルオプション Cソースファイル名
```

```
front$ icpc コンパイルオプション C++ソースファイル名
```

- ・ 自動並列化

```
front$ ifort -parallel コンパイルオプション Fortranソースファイル名
```

```
front$ icc -parallel コンパイルオプション Cソースファイル名
```

```
front$ icpc -parallel コンパイルオプション C++ソースファイル名
```

- ・ OpenMP並列化

```
front$ ifort -qopenmp コンパイルオプション Fortranソースファイル名
```

```
front$ icc -qopenmp コンパイルオプション Cソースファイル名
```

```
front$ icpc -qopenmp コンパイルオプション C++ソースファイル名
```

- ・ MPI並列化 (OpenMPIを利用, OpenMP並列化の併用も可能)

```
front$ mpiifort コンパイルオプション Fortranソースファイル名
```

```
front$ mpiicc コンパイルオプション Cソースファイル名
```

```
front$ mpiicpc コンパイルオプション C++ソースファイル名
```

リクエストの実行方法と実行キュー (1/5)

- ・ ジョブスクリプトファイルを作成し, qsub コマンドでリクエストを投入
front\$ **qsub** ジョブスクリプトファイル名
- ・ バッチリクエストの実行状況, リクエストIDは, reqstat コマンドで確認
front\$ **reqstat**
- ・ バッチリクエストのキャンセル, 途中終了はqdelコマンド
front\$ **qdel** リクエストID

【逐次実行】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q sx --venode 1</code>	#SX-Auroraを1VE使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>cd \$PBS_O_WORKDIR</code>	# qsubを実行したディレクトリに移動
<code>./a.out</code>	#カレントディレクトリのa.outを実行

【自動並列／OpenMP並列実行】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q sx --venode 1</code>	#SX-Auroraを1VE使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -v OMP_NUM_THREADS=8</code>	#1VEあたり8コアで実行 (1～8)
<code>cd \$PBS_O_WORKDIR</code>	# qsubを実行したディレクトリに移動
<code>./a.out</code>	#カレントディレクトリのa.outを実行

【MPI実行】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q sx --venode 8</code>	#SX-Auroraを8VE使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>cd \$PBS_O_WORKDIR</code>	# qsubを実行したディレクトリに移動
<code>mpirun -np 64 ./a.out</code>	#カレントディレクトリのa.outを64プロセスで実行

【MPIと自動並列／OpenMP並列
の同時利用】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q sx --venode 8</code>	#SX-Auroraを8VE使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -v OMP_NUM_THREADS=8</code>	#1VEあたり8コアで実行 (1～8)
<code>cd \$PBS_O_WORKDIR</code>	# qsubを実行したディレクトリに移動
<code>mpirun -np 8 ./a.out</code>	#カレントディレクトリのa.outを8プロセスx8コア並列で実行

AOCCの場合**【逐次実行】**

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 1</code>	#LX 460Rz-2を1ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>./a.out</code>	#カレントディレクトリのa.outを実行

【OpenMP並列実行】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 1</code>	#LX 460Rz-2を1ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -v OMP_NUM_THREADS=128</code>	#1ノードあたり128コアで実行 (1~128)
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>./a.out</code>	#カレントディレクトリのa.outを実行

【MPI実行】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 2</code>	#LX 460Rz-2を2ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -T openmpi</code>	#OpenMPIライブラリを使うことを指定
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>mpirun \$NQSVMPIOPTS -np 256 ./a.out</code>	#カレントディレクトリのa.outを256プロセスで実行

**【MPIとOpenMP並列
の同時利用】**

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 2</code>	#LX 460Rz-2を2ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -T openmpi</code>	#OpenMPIライブラリを使うことを指定
<code>#PBS -v OMP_NUM_THREADS=64</code>	#1ノードあたり64コアで実行 (1~128)
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>mpirun \$NQSVMPIOPTS --map-by ppr:2:node -np 4 ./a.out</code>	#カレントディレクトリのa.outを4プロセスx64コア並列で実行

Intel OneAPIの場合**【逐次実行】**

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 1</code>	#LX 460Rz-2を1ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>source /opt/oneapi/setvars.sh intel64</code>	#プログラム実行環境をIntelコンパイラに切替え
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>./a.out</code>	#カレントディレクトリのa.outを実行

【OpenMP並列実行】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 1</code>	#LX 460Rz-2を1ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -v OMP_NUM_THREADS=128</code>	#1ノードあたり128コアで実行（1～128）
<code>source /opt/oneapi/setvars.sh intel64</code>	#プログラム実行環境をIntelコンパイラに切替え
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>./a.out</code>	#カレントディレクトリのa.outを実行

【MPI実行】

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 2</code>	#LX 460Rz-2を2ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -T intmpi</code>	#Intel MPIライブラリを使うことを指定
<code>source /opt/oneapi/setvars.sh intel64</code>	#プログラム実行環境をIntelコンパイラに切替え
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>mpirun -np 256 ./a.out</code>	#カレントディレクトリのa.outを256プロセスで実行

**【MPIとOpenMP並列実行
の同時利用】**

<code>#!/bin/sh</code>	#シェルを指定
<code>#PBS -q lx -b 2</code>	#LX 460Rz-2を2ノード使用する
<code>#PBS -l elapstim_req=2:00:00</code>	#実行時間を2時間に設定
<code>#PBS -T intmpi</code>	#Intel MPIライブラリを使うことを指定
<code>#PBS -v OMP_NUM_THREADS=64</code>	#1ノードあたり64コアで実行（1～128）
<code>source /opt/oneapi/setvars.sh intel64</code>	#プログラム実行環境をIntelコンパイラに切替え
<code>cd \$PBS_O_WORKDIR</code>	#qsubを実行したディレクトリに移動
<code>mpirun -hostfile \${PBS_NODEFILE} -ppn 2 -np 4 ./a.out</code>	#カレントディレクトリのa.outを4プロセスx64コア並列で実行

リクエストの実行方法と実行キュー（5/5）

サブシステムAOBA-A（SX-Aurora TSUBASA）

実行キュー名	利用可能VE数	実行時間制限 規定値／最大値	ジョブの実行形態
sxf	1	1時間／1時間	1VEジョブ 1時間無料 (VH を共用する)
SX	1	72時間／720時間	1VEジョブ (VH を共用する)
	2～256		8VE 単位で確保 (VH を共用しない)
sxmix	2～8		1VE 単位で確保 (VH を共用する)
個別設定	契約VE数	720時間	占有利用

サブシステムAOBA-B（LX 406Rz-2）

実行キュー名	利用可能ノード数	実行時間制限 規定値／最大値	ジョブの実行形態
lx	1～16	72時間／720時間	ノードを共用しない
個別設定	契約ノード数	720時間	占有利用

- ・バイナリを作成するのに利用したコンパイラと、投入する計算機のキューの確認が必要
→ GNUコンパイラで作成したバイナリをSXのキューに投入すると、VH（EPYC）で実行されてしまう

負担金制度（1/3）

- ・ 利用者番号（アカウント）の初期登録料，年間維持費**なし**
 - 従量課金を基本とするため，計算機を利用しない場合の負担金請求は**0円**
 - ※ 利用者番号は年度を超える場合も自動継続され，ホーム領域（/uhome）のデータも保存されます。
- ・ 計算機利用負担金
 - 共有利用・従量
課金対象時間（利用VH数または利用ノード数と，利用時間の積）に比例した課金方式
 - 共有利用・定額
利用負担金の先払いにより，負担額の課金対象時間相当まで計算機を利用可能
年度途中に定額負担金の追加も可能
 - 占有利用
3ヶ月単位でAOBA-A（8VE単位）またはAOBA-B（1ノード単位）を占有して利用
特定利用者と計算資源を占有するため，他利用者のジョブ待ちが無い
- ・ ストレージ負担経費
 - ホーム領域 5TBまで無料
 - 追加1TBにつき年額3,000円
- ・ 出力負担経費
 - センターの大判プリンタ1枚につき ソフトクロス紙 1,200円 光沢紙 600円
- ・ 民間企業利用については 成果公開型は2倍，成果非公開型は4倍の課金単価

負担金制度 (2/3)

サブシステムAOBA-A (SX-Aurora TSUBASA)

利用形態	負担額および利用可能課金対象時間
共有 (無料)	利用 VE 数 1 (実行数, 実行時間の制限あり) 無料
共有 (従量)	課金対象時間 = (利用 VE 数 ÷ 8 を切り上げた数) × 経過時間 (秒) 課金対象時間の合計 1 時間につき 125 円 課金対象時間は半期毎 (4～9 月および 10～3 月) に合計し, 1 時間未満を切上げて負担金を請求する。
共有 (定額)	負担額 10 万円 につき 課金対象時間の合計 800時間
占有	利用 VE数8 , 利用期間 3ヶ月 につき 270,000円

サブシステムAOBA-B (LX 406-Rz2)

利用形態	負担額および利用可能課金対象時間
共有 (従量)	課金対象時間 = 利用ノード数 × 経過時間 (秒) 課金対象時間の合計 1 時間につき 22 円 課金対象時間は半期毎 (4～9 月および 10～3 月) に合計し, 1 時間未満を切上げて負担金を請求する。
共有 (定額)	負担額 10 万円 につき 課金対象時間の合計 4,600時間
占有	利用 ノード数1 , 利用期間 3ヶ月 につき 47,000円

・ 民間企業利用については 成果公開型は**2倍**, 成果非公開型は**4倍**の課金単価

利用申請方法（利用者番号の取得）

センターに利用申請 <https://www.ss.cc.tohoku.ac.jp/apply-for-use/> をご参照ください。

- ・ 大学・学術利用

- ☐ 負担金請求あり， 随時申請可能

- ・ 民間企業利用（成果公開型／成果非公開型）

- ☐ 負担金請求あり， 課題審査あり， トライアルユースあり， 随時申請可能

- ・ センターとの共同研究（大学・学術・民間企業利用対象）

- ☐ 負担金請求あり， 課題審査あり、負担金助成あり， 応募期間あり（締切3月頃）

各機関での課題募集

- ・ 学際大規模情報基盤共同利用・共同研究拠点公募型共同研究（JHPCN） <https://jhpcn-kyoten.itc.u-tokyo.ac.jp/>

- ☐ 採択予算超過の場合に負担金請求あり， 応募期間あり（締切1月頃）

- ・ 革新的ハイパフォーマンズ・コンピューティング・インフラ（HPCI） <http://www.hpci-office.jp/>

- ☐ 負担金請求なし（採択資源量まで利用可能）， 応募期間あり（締切11月頃）

利用者支援 <https://www.ss.cc.tohoku.ac.jp/support/> をご参照ください。

- ・利用者講習会

- ☐ システムの利用法, コードの高速化・並列化, ネットワーク・セキュリティ, アプリケーション利用方法について, 年間10回程度開催
- ☐ センター内端末機室および遠隔配信で実施

- ・利用相談

- ☐ 利用申請の方法, システムの利用方法, コンパイルエラー, ジョブの投入方法, コードの高速化など
- ☐ 利用相談フォームで受け付け <https://www.ss.cc.tohoku.ac.jp/consultation/>
- ☐ メールで継続的にサポート
- ☐ 年間約100件

- ・高速化支援

- ☐ コードを預かり, 利用者, センター教職員, ベンダーが協力してコードの高速化・並列化を実施
- ☐ コード大規模化のサポート, JHPCN課題, HPCI課題へのステップアップを支援
- ☐ 1997年から継続的な取り組み
- ☐ 年間5～10件程度を実施
- ☐ SX-ACEシステム（2015年度～2020年度）ではベクトル高速化またはMPI並列化を30件実施
(平均16.7倍) (平均約2.4倍)