



# アクセラレータを備えた並列計算機 のためのプログラミング言語 XcalableACC

村井 均  
理研 AICS

# はじめに(1)



- アクセラレータ・クラスタ(e.g. GPUクラスタ)の普及
  - TOP500のトップ10のうち4つ
- MPI+CUDAによるプログラミングが主流。
  - 生産性の問題(難しい)
- 指示文ベースのアクセラレータ向けプログラミングモデル **OpenACC**  
Directives for Accelerators
  - ➔ XMPとOpenACCを組み合わせよう!

# はじめに(2)

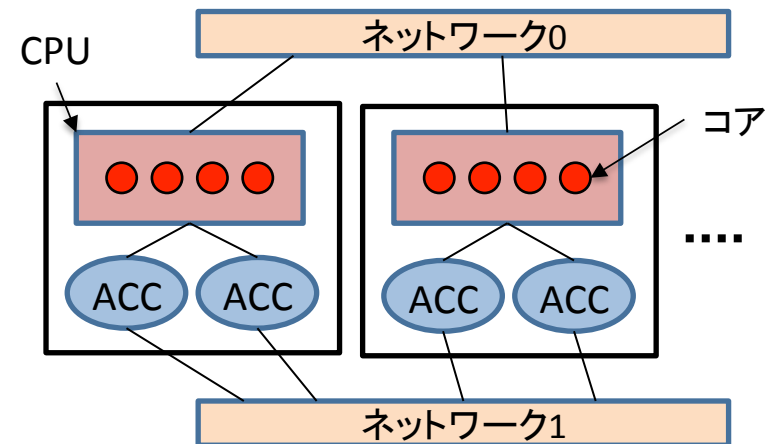
- アクセラレータ・クラスタの課題

- 多階層並列性

- ノード間 → XMP
- (ノード内 → OpenMP)
- デバイス間 → ?
- アクセラレータ → OpenACC

- デバイス間直接通信(物理 or 論理)

- Tightly Coupled Accelerator (TCA)
- NVIDIA GPUDirect



# 目的

- アクセラレータ・クラスタのための新しいプログラミング言語XcalableACC (XACC)の提案
  - 多階層並列性
  - デバイス間直接通信
- Omni XcalableACCコンパイラの開発
  - ➔ 高性能と高生産性の両方を実現

# OpenACC

- オフロードモデルに基づくアクセラレータのためのプログラミングモデル
- 指示文ベース
- C/C++/Fortran
- Cray, CAPS, PGI (NVIDIA)

# サンプルコード

```
#pragma acc data copyin(A)
{
    #pragma acc kernels
    {
        #pragma acc loop independant
        for (int i = 0; i < N; ++i){
            A[i][0] = ...;
        }
        ...
    }

    #pragma acc update host(A)
    ...
}
```

Aをデバイス上に割付け

デバイスへオフロードする処理  
(compute region)

デバイス上でスレッド並列処理

Aをホストへコピー

# OpenACCの特徴

- オフロードモデル
  - ホスト(CPU)が、データおよびワークをデバイスに「オフロード」する。
  - デバイスのあらゆる動作は、ホストにより (compute regionの外側で) 指示される。
  - 実行キューに基づく非同期処理 (cf. CUDAストリーム)
- マルチデバイスを意図した機能は、実質的に存在しない。

# XACCの基本方針(1)

- XACC = XMP + OpenACC + 新規拡張

XMP指示文	ノード間の分散メモリ並列処理
OpenACC指示文	デバイス上の並列処理
XACC拡張	階層並列性、デバイス間直接通信

cf. XMP + OpenMPの場合に比べ、XMP指示文とOpenACC指示文の関連は密であり、XACC処理系は各種指示文を解釈する必要がある。

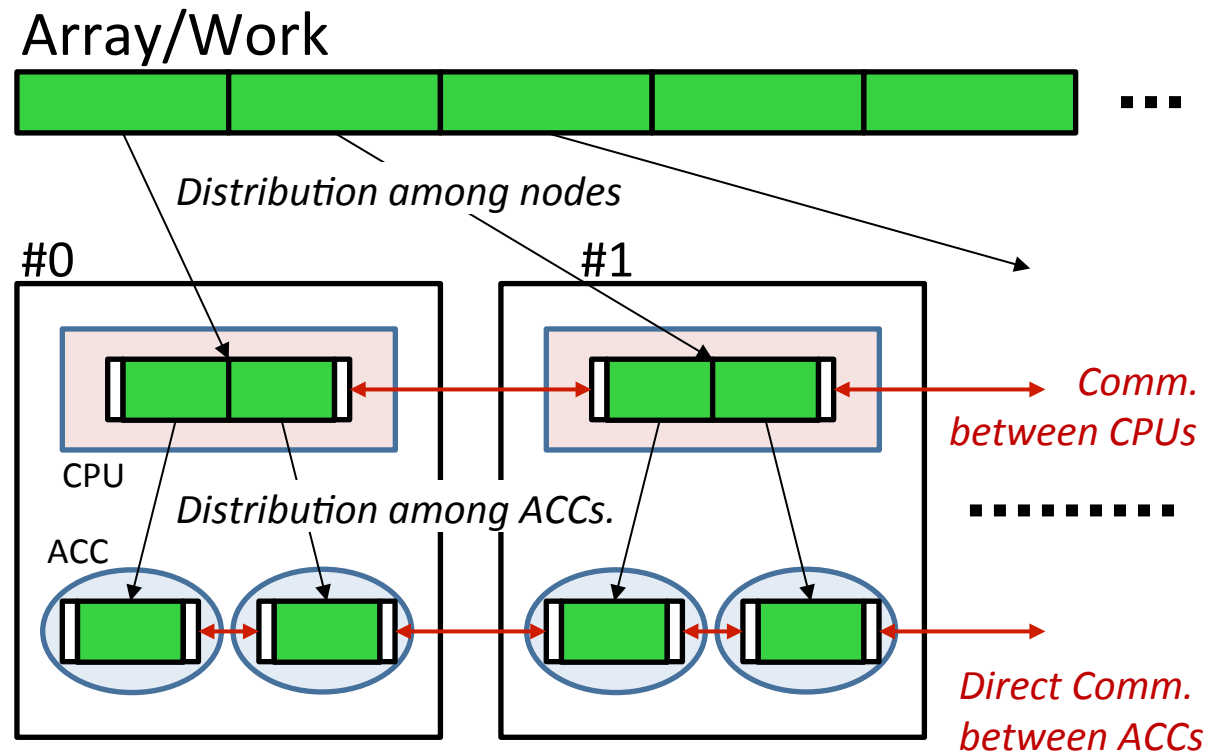


# XACCの基本方針(2)

- 原則として、XMP構文(ノード間並列)の内側にOpenACC指示文(アクセラレータ処理)を置く → 後で再検討
- 2種類の指示文(prefix)
  - (拡張された)XMP指示文: `#pragma xmp`
  - (拡張された)OpenACC指示文: `#pragma acc`

※ デバイス間直接通信 → XMP拡張  
階層並列性 → OpenACC拡張

# XACCの実行モデル



# XACC拡張

- XMP
  - acc節
- OpenACC
  - device指示文
  - on\_device節
  - layout節
  - shadow節
  - barrier\_device指示文

# XMP通信指示文の拡張 (acc節)

- acc節を伴うXMP通信指示文は、デバイス上のデータを対象にする。

```
#pragma xmp reflect (a) acc
```

※ acc節がない場合は、処理系が選択する。

- acc節を伴うbarrier指示文は、デバイス間のバリア同期を行う。

```
#pragma xmp barrier acc
```

# OpenACC拡張: device指示文

- 一次元、1-originの「デバイス配列」を宣言する → `on_device`節の引数
- “=”節なしで宣言した場合、暗黙にデバイスの型と個数が決定される(実装依存)。
- “=”節には、処理系依存のpredefinedなデバイス配列を指定できる。

```
#pragma acc device d0(*)  
#pragma acc device d1(*) = nvidia(1:4)
```

# OpenACC拡張: `on_device`節

- 各ACC指示文の動作の対象となるデバイス配列を指定する。
  - `data / enter data / exit data / declare / update`
  - `parallel / kernel / parallel loop / kernels loop`
  - `wait`
  - `barrier_device` **New!**
- 特に、(非同期でない) `wait` 指示文は全デバイスにおける処理が完了するまでブロックする。

# OpenACC拡張: layout節とshadow節

- layout節
  - 対象:
    - declare
    - loop / parallel loop / kernels loop
  - 配列のインデックスおよびループのイタレーションの、デバイス配列への分散方法を指定する。
  - 分散形式はblockと"\*"のみ。
- shadow節
  - 対象: declare
  - デバイス配列への分散に関するシャドウを指定する。

## XMPコード (マルチノード)

```
void foo(){  
  
#pragma xmp nodes p(4)  
  
#pragma xmp template t(0:99)  
#pragma xmp distribute t(block) onto p  
  
float a[100][100];  
#pragma xmp align a[i][*] with t(i)  
#pragma xmp shadow a[1:1][0]  
  
#pragma xmp reflect (a)  
  
#pragma xmp loop (i) on t(i)  
for (int i = 0; i < 100; i++){  
    for (int j = 0; j < 99; j++){  
        a[i][j+1] = 1;  
    }  
}  
}
```

4ノード上に配列aを分散

ステンシル通信

外側ループをノード間で並列化



## XACCコード (マルチノード+マルチデバイス)

4ノード上に配列aを分散

```
void foo(){  
  
#pragma xmp nodes p(4)  
#pragma acc device d(*)  
  
#pragma xmp template t(0:99)  
#pragma xmp distribute t(block) onto p  
  
float a[100][100];  
#pragma xmp align a[i][*] with t(i)  
#pragma xmp shadow a[1:1][0]  
#pragma acc declare copy(a) layout([*][block]) shadow([0][1:1]) on_device(d)  
  
#pragma xmp reflect (a) acc  
  
#pragma xmp loop (i) on t(i)  
for (int i = 0; i < 100; i++){  
#pragma acc parallel loop layout(a[*][j+1]) on_device(d)  
for (int j = 0; j < 99; j++){  
a[i][j+1] = 1;  
}  
}  
}
```

デバイス配列dを宣言

配列aをd上に分散

ステンシル通信

外側ループをノード間で並列化

内側ループをデバイス間で並列化

## XACCコード (マルチデバイス)

```
void foo(){
```

```
#pragma acc device d(*)
```

デバイス配列dを宣言

```
float a[100][100];
```

配列aをd上に分散

```
#pragma acc declare copy(a) layout([*][block]) shadow([0][1:1]) on_device(d)
```

```
#pragma xmp reflect (a) acc
```

ステンシル通信

```
for (int i = 0; i < 100; i++){
```

```
#pragma acc parallel loop layout(a[*][j+1]) on_device(d)
```

```
for (int j = 0; j < 99; j++){
```

```
    a[i][j+1] = 1;
```

```
    }
```

```
  }
```

```
}
```

内側ループをデバイス間で並列化

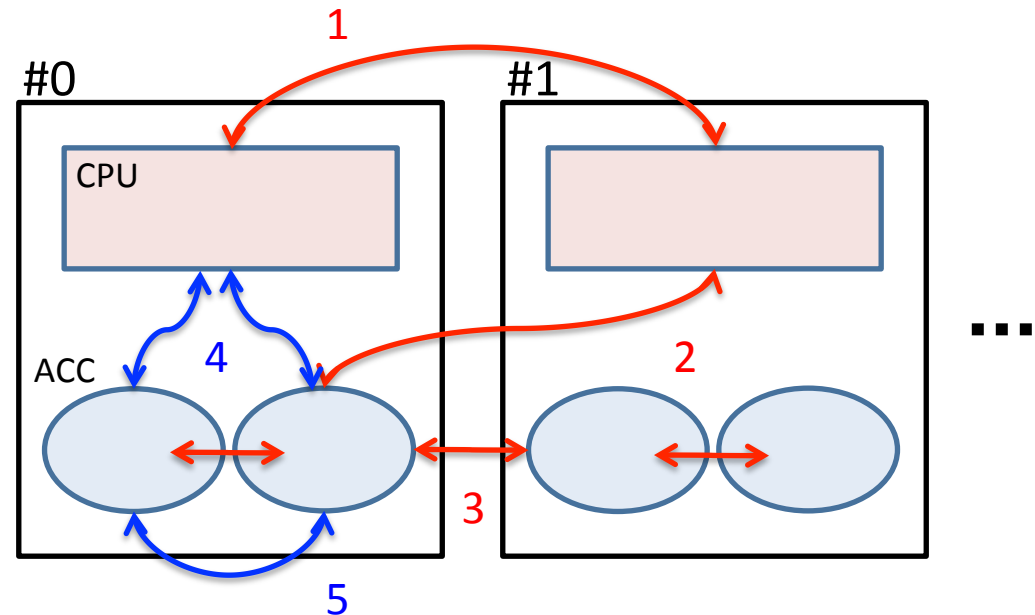
# 同期モデル(検討中)

- ノード間

1. ホスト-ホスト
2. ホスト-デバイス
3. デバイス-デバイス

- ノード内

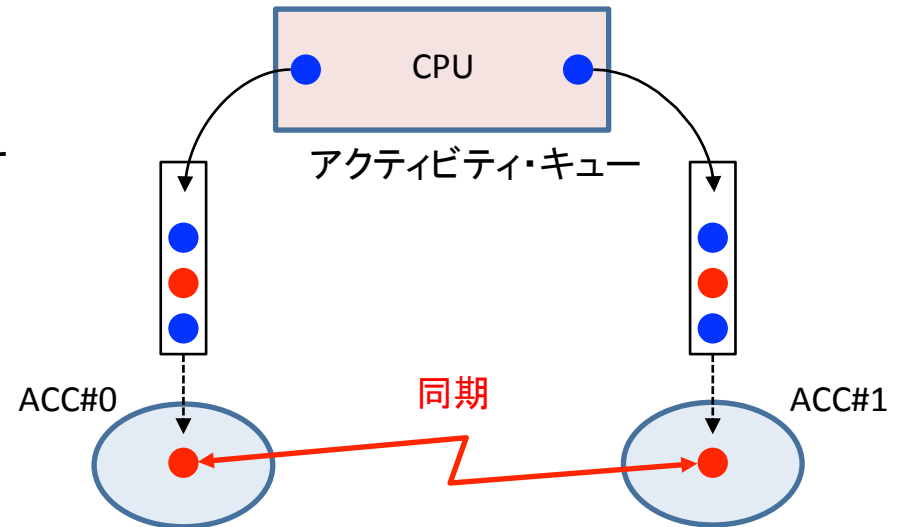
4. ホスト-デバイス
5. デバイス-デバイス



# 同期モデル(検討中)

- デバイス間の「同期」とは？(3と5)
  - デバイスが非同期処理を実行する順序に関する「制約」→ 特別な非同期処理(イベント)
  - ホストにより指定されるが、ホストは関与しない。

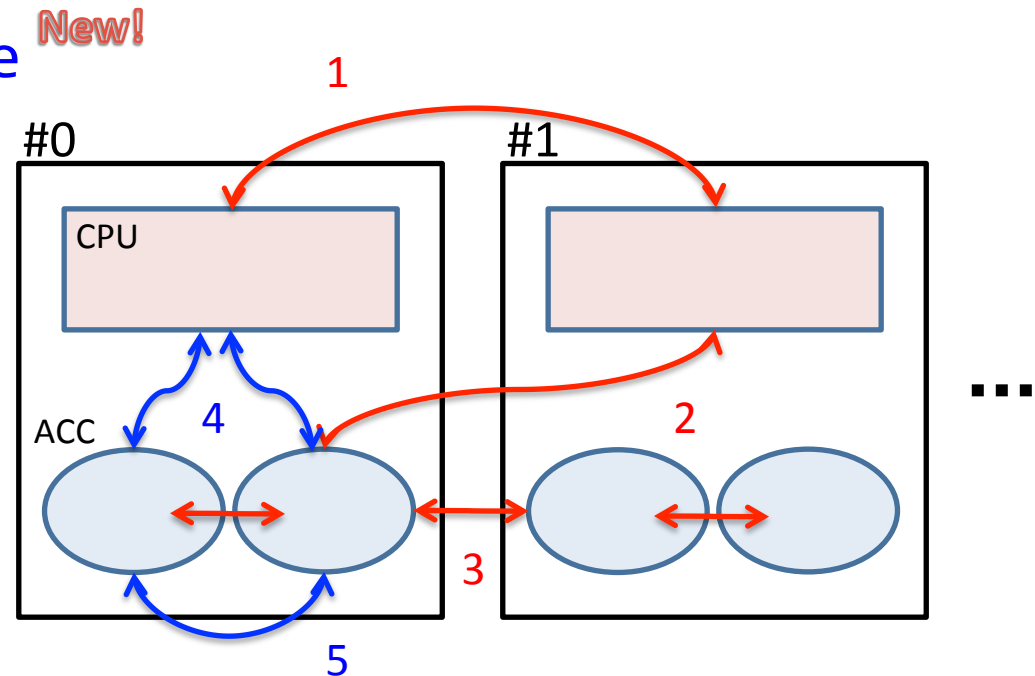
※ 同期処理の場合(4):  
ホストが処理を発行する順序に関する「制約」として実現。



# 同期モデル(検討中)

1. xmp barrier
2. 不要?
3. xmp barrier acc **New!**
4. acc wait
5. acc barrier\_device **New!**

※ 3は5の機能を包含する。したがって、5は主にローカルビュー  
並列が対象。



# 要検討

- XMP構文とACC構文のネストの順序
  - acc data構文の内側にXMP構文
  - compute regionの内側にxmp loop構文
  - compute regionの内側にXMP通信・同期開始構文 → デバイスから通信または同期をキックしたい。
- syntactic sugar ?

```
#pragma xmp reflect_init (a)

#pragma acc kernels on_device(d)
{

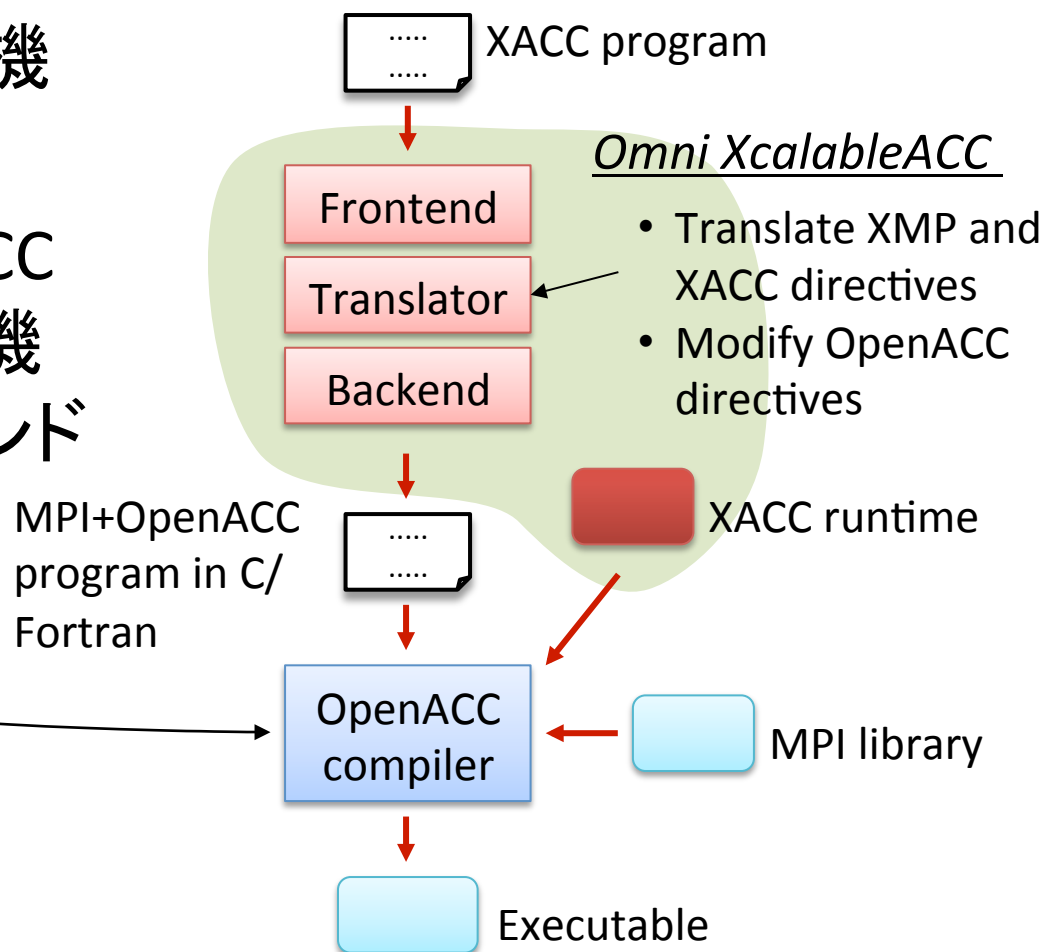
    #pragma xmp loop (i) on t(i)
    for (int i = 0; i < 100; i++){
        #pragma acc loop
        for (int j = 0; j < 100; j++){
            a[i][j] = ...
        }
    }

    #pragma xmp reflect_do (a)

    #pragma xmp loop (i) on t(i)
    for (int i = 0; i < 100; i++){
        #pragma acc loop
        for (int j = 0; j < 100; j++){
            b[i][j] = ...
        }
    }
}
```

# Omni XscalableACC

- Omni XMPの「拡張機能」として開発中。
- Omni自身もOpenACCコンパイラとしての機能を持つ(バックエンドとして利用可能)。





# 予備評価

- 姫野ベンチマークをXACCで実装
- HA-PACS/TCAで評価

※ デバイス間の並列処理は  
未実装

```
float p[MIMAX][MJMAX][MKMAX];  
// XMP指示文を用いて分散配列の定義  
#pragma xmp shadow p[1:1][1:1][0]  
  
#pragma acc data copy(p) ..  
{  
..  
#pragma xmp reflect (p) acc  
..  
#pragma xmp loop (k,j,i) on t(k,j,i)  
#pragma acc parallel loop collapse(3) ..  
for(i=1 ; i<MIMAX ; ++i)  
  for(j=1 ; j<MJMAX ; ++j){  
    for(k=1 ; k<MKMAX ; ++k){  
      S0 = p[i+1][j][k] * ..;
```

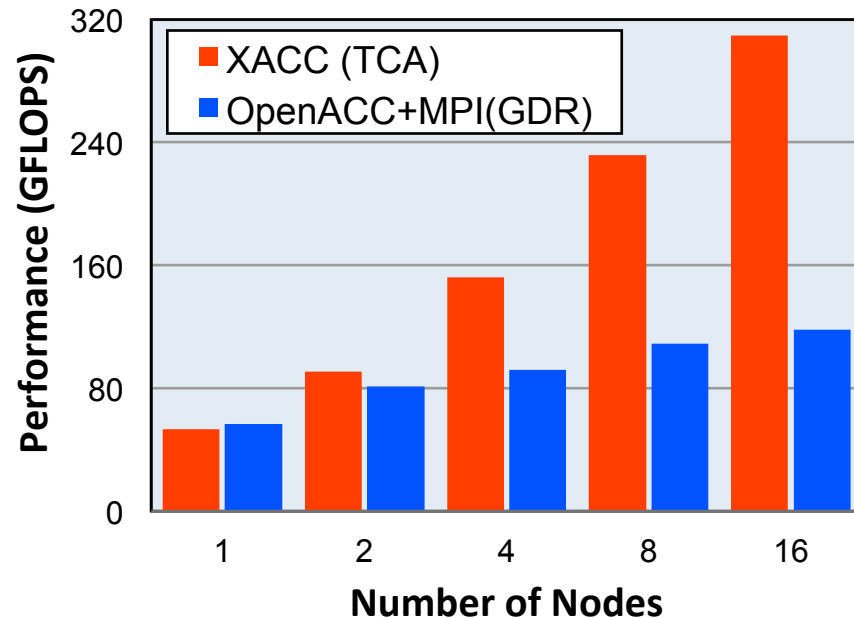
← 袖領域の定義

← アクセラレータに分散配列を転送

← アクセラレータメモリの袖交換

← ループの分散処理

# 予備評価: 性能



- OpenACC+MVAPICH2-GDRと比較。
- サイズM(128x128x256)で、最大2.7倍の性能
- TCAの制約により、サイズを大きくすると差は小さくなる。

# 予備評価: 生産性

- 行数 (source lines of codes)

	トータル	XMP	OpenACC	指示文以外
XACC	213	28	9	176
OpenACC + MPI	488	-	15	473

- XACCの行数はOpenACC + MPIの半分以下  
∵ ループ文のインデックス計算や、ステンシル通信のための計算が不要
- グローバルビュー

# まとめ

- アクセラレータ・クラスタ向けプログラミング言語XcalableACCを開発中。
- XACC = XMP + OpenACC + XACC拡張
- マルチデバイスとデバイス間直接通信をサポート。
- 予備評価により、高性能と高生産性を確認。