AICS公開ソフトウェア講習会 15回

- 表題　通信ライブラリとI/Oライブラリ
- 場所　AICS R104-2
- 時間　2016/03/23 (水) 13:30-17:00
  - 13:30 - 13:40 全体説明
  - 13:40 - 14:10 PRDMA
  - 14:10 - 14:40 MPICH
  - 14:40 - 15:10 PVAS
  - 15:10 - 15:30 休憩
  - 15:30 - 16:00 Carp
  - 16:00 - 16:30 MPI-IO
  - 16:30 - 17:00 Darshan

AICS公開ソフトウェア講習会
MPICH

畑中正行

2016/03/23

K computer

目次

- MPICH とは?
- MPICH on 京
- 使用上の注意
- MPI-3 隣接集団通信 （紹介）
- まとめ
- 今後の展開

MPICH とは?

- メジャーな MPI 実装
  - 主に Open MPI と MPICH。それをベースに各社製品化
    - Open MPI　：富士通(京)
    - MPICH　　：インテル，マイクロソフト，…

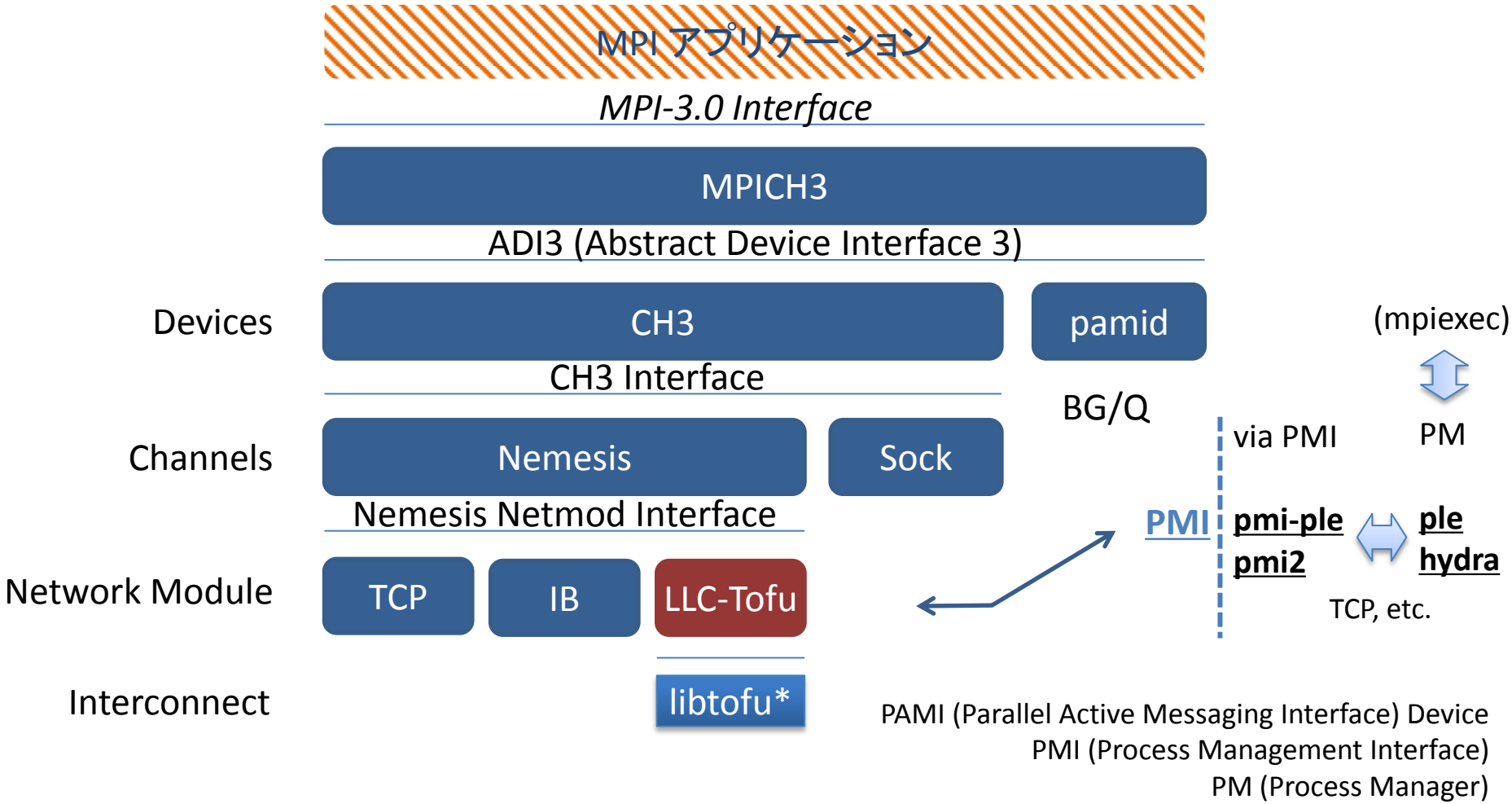
- 「京」のサポート
  - 富士通製：Open MPI ベース (MPI-2.2 準拠)
    - 現在 MPI 標準の最新版は、MPI-3.1 版

MPICH on 京 - 使い方

- 公開場所
  - 「京」 MPICH_HOME=/opt/aics/mpich/mpich-3.1.4
- コンパイル(ログインノード; cross compiler)
  - ${MPICH_HOME}/bin
  - mpicc (富士通 fccpx の wrapper; mpifccpx)
  - mpif90 (富士通 frtpx の wrapper; mpifrtpx)
  - mpicxx (富士通 FCCpx の wrapper; mpiFCCpx)
    - 計算ノード用 own compiler は現在未提供
- 実行(計算ノード)
  - mpichexec (mpiexec の wrapper)
    - Staging 等の基本オプションは同じ
    - 但し、-mca 等 Open MPI 固有オプションは使えない

MPICH on 京 – 基本構成

- MPICH 基本構成



MPICH on 京 – サポート状況

- サポート状況
  - mpich-3.1.4 (stable)
  - MPI-3.0 仕様準拠
  - mpich 同梱テストセット完走済 (517 項目; spawn除く)

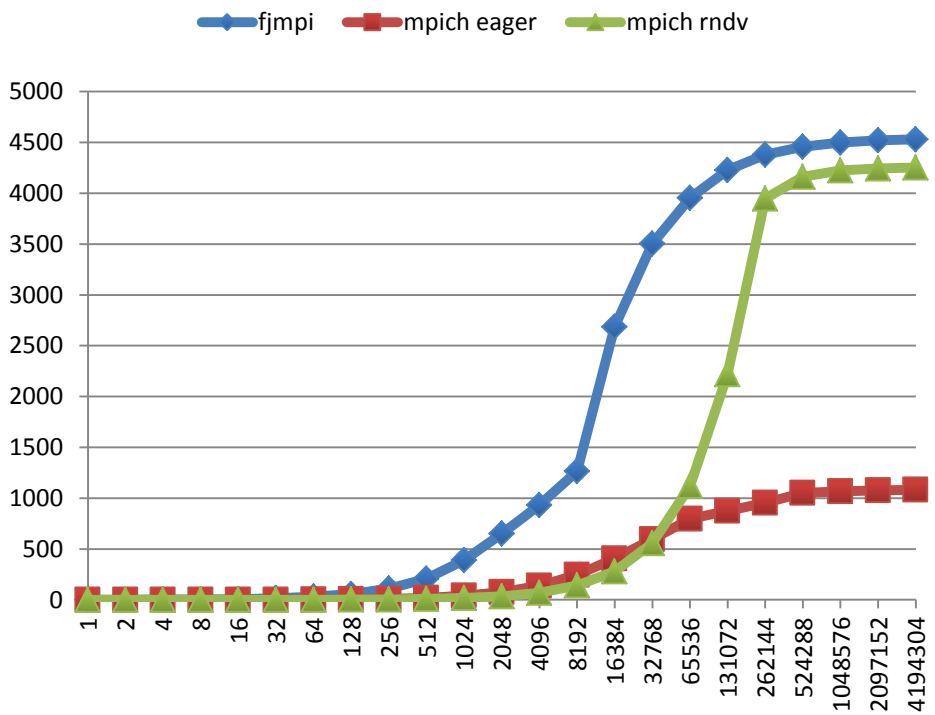
MPICH on 京 – 使用上の注意

- DPC (Dynamic Process Creation) は未サポート
  - MPI_Comm_spawn (と、その派生呼出し)
  - MPI_Comm_connect/accept (と、その関連呼出し)
    - Process Manager が Hydra/KVS でなく、富士通 PLE のため
    - PMI (Process Manager Interface) に強い制限あり
- MPI-IO での FEFS アクセスは未サポート
  - FEFS は「京」の並列ファイルシステム (Lustreベース)
  - MPI-IO 実装の romio/adio に fefs 対応なし
- 富士通拡張なし
  - FJRDMA_ 関数セットなし
  - 集団通信は「京」最適化なし
    - mpich の汎用実装のみ

MPICH on 京 – 性能

- IMB-4.1 (Intel MPI Benchmarks 4.1)
  - Uniband (縦軸: スループット [MB/s], 横軸: 転送サイズ [Bytes])

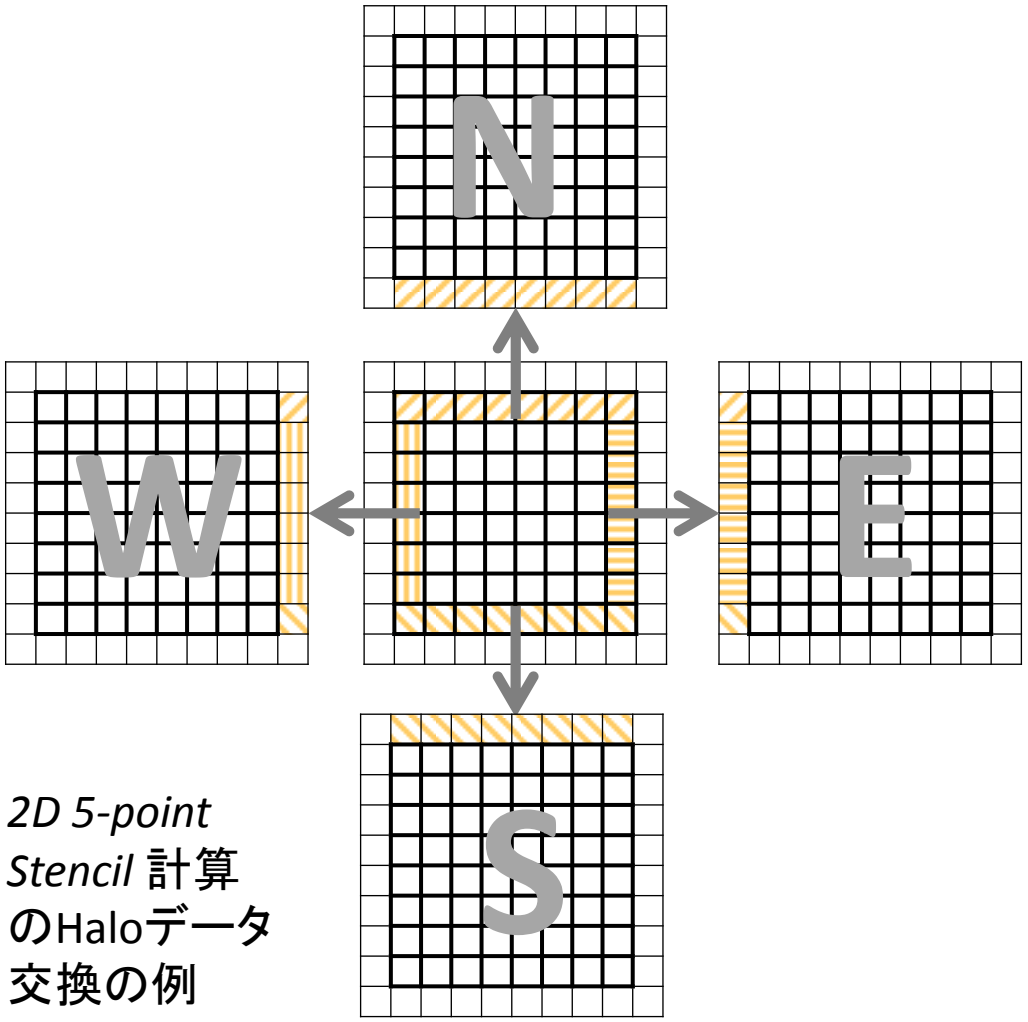
  - 転送サイズが大きいところでは、ピーク性能がまずまず出ている
  - 立ち上がりが悪い原因は、Rendezvous プロトコル(制御用)の短メッセージの処理が遅いため



→ EAGER 性能改善

隣接集団通信 (1)

- 隣接通信



2D 5-point Stencil 計算のHaloデータ交換の例

- 隣接通信
  - 通信範囲が隣接プロセスに限定
- 重要な通信パターン
  - 袖通信
    - Halo データ交換
    - Ghost 領域更新
    - のりしろ…
- MPI-3.0 版仕様で集団通信としてサポート

隣接集団通信 (2)

京の既存システム
→ MPI-2.2 仕様準拠
(MPI-3.0未サポート)

- MPI 3.0 仕様
  - 2012 年 09 月に公開
  - MPICH や Open MPI 等主要な MPI 実装でサポート済

- MPI-3 MPI_Neighbor_alltoallw() 隣接集団通信関数
  - 引数は既存の MPI_Alltoallw() と同じ
  - MPI_Cart_create() (or MPI_Graph_create 系)から生成されたコミュニケータ
    - 隣接関係が決定できるコミュニケータであること

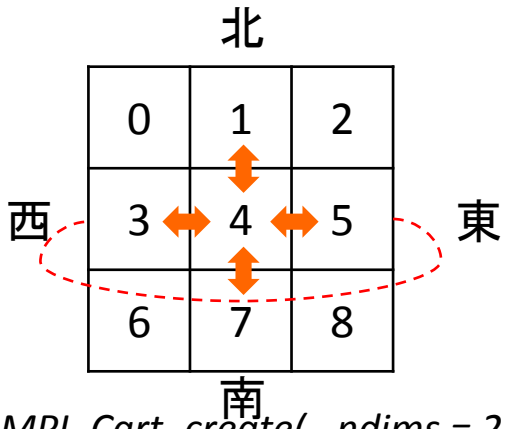
```
int MPI_Neighbor_alltoallw(
  const void *sendbuf, const int sendcounts[], const MPI_Aint sdispls[], const MPI_Datatype sendtypes[],
  void *       recvbuf, const int  recvcounts[], const MPI_Aint  rdispls[], const MPI_Datatype  recvtypes[],
  MPI_Comm comm);
```

特殊な
コミュニケータ

- 通信相手ごとに、バッファの先頭、データ型、データ個数を指定できる

隣接集団通信 (3)

- MPI_Neighbor_alltoallw
  - 引数 comm
    - MPI_Cart_create() で生成
      - MPI 1.0 仕様から導入されている
      - コミュニケータに隣接関係を記述する情報を追加



MPI_Cart_create(, ndims = 2, dims = { 3, 3 }, , , &cart) の例

```
int MPI_Cart_create( MPI_comm comm_old,  int ndims, const int dims[], const int periods[], int reorder,
  MPI_Comm *comm_cart);
```

- 引数 sendcounts[], sdipls[], sendtypes[] (recv も同様)

	[0]	[1]	[2]	[3]	[4]	[5]
ndims=1	西	東				
ndims=2	北	南	西	東		
ndims=3	下	上	北	南	西	東

ここでは、通信相手（隣接プロセス）を東西南北で呼ぶことにする

東西南北上下の定義

1. MPI_Cart_shift(, ndims – 1, 1, 西, 東)
2. MPI_Cart_shift(, ndims – 2, 1, 北, 南)
3. MPI_Cart_shift(, ndims – 3, 1, 下, 上)

隣接集団通信 (4)

- 隣接集団通信　小まとめ
  - 実装側が Tag Matching 等、isend/irecv の複雑なセマンティックスから解放される
  - PRDMA では MPI_Startall を使って、暗に通信パターンを教えていたが、実装側に明に伝えられる
  - コミュニケータの付加情報を使って、通信トポロジも伝えられる

    → 最適化が効きやすい

  - MPI_Ineighbor_alltoallw を使えば、計算と通信のオーバラップを狙った、非ブロッキング呼出しが可能

隣接用 コミュニケータ (1)

- 2D 5-point ステンシル通信パターンの定義 by MPI_Cart_create

```
MPI_Comm comm_orig = MPI_COMM_WORLD; /* or comm_split */
MPI_Comm comm_cart = MPI_COMM_NULL;
int ndim = 2, dims[2], cycs[2], rord;

dims[0] = 3; dims[1] = 3; /* 3 x 3   */
cycs[0] = 1; cycs[1] = 1; /* periods */
rord = 0; /* reorder */

MPI_Cart_create(comm_orig, ndim, dims, cycs, rord, &comm_cart);
```

- MPI_Cart_create
  - MPI プロセスのトポロジが格子状 (Cartesian) であることを定義

  - 引数 ndim　　次元数 (ここでは 2 次元)
  - 引数 dims　　各次元のサイズ (ここでは 3x3)
  - 引数 cycs　　格子境界で巡回するか？ (ここでは「巡回」)
  - 引数 rord　　ランクを振り直すか？ (ここでは「しない」)

隣接用 コミュニケータ (2)

- MPI_Cart_create の問題
  - 1D 3-point, 2D 5-point, 3D 7-point ステンシルしか定義できない
  - 斜め方向を含む 2D 9-point (star) ステンシルはどう定義するのか?

- MPI_Neighbor_alltoallw で使用可能なもう一つの MPI_Dist_graph_create 系のコミュニケータを使う
- ここでは、MPI_Cart_create から、MPI_Dist_graph_create_adjacent で MPI_DIST_GRAPH タイプのコミュニケータを生成する例を示す

隣接用 コミュニケータ (3)

- 2D 9-point ステンシル通信パターンの定義

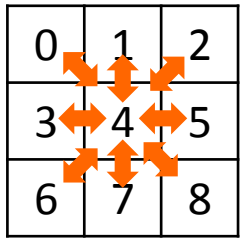
```
int nsrc = 8, srcs[8], ndst = 8, dsts[8];
int ndim = 2, dims[2], cycs[2], ords[2];

MPI_Cart_get(comm_cart, ndim, dims, cycs, ords);
for (ii = 0; ii < 8; ii++) {
  int cord[2] = { ords[0], ords[1] }, rank_peer;

  switch (ii) {
  case 0 /* nn */: cord[0] -= 1; break;
  case 1 /* ss */: cord[0] += 1; break;
  case 2 /* ww */: cord[1] -= 1; break;
  case 3 /* ee */: cord[1] += 1; break;
  case 4 /* nw */: cord[0] -= 1; cord[1] -= 1; break;
  case 5 /* se */: cord[0] += 1; cord[1] += 1; break;
  case 6 /* ne */: cord[0] -= 1; cord[1] += 1; break;
  case 7 /* sw */: cord[0] += 1; cord[1] -= 1; break;
  }

  if((cycs[0] == 0)
     && ((cord[0] < 0)||(cord[0] >= dims[0]))) {
    rank_peer = MPI_PROC_NULL;
  }
  else if((cycs[1] == 0)
     && ((cord[1] < 0)||(cord[1] >= dims[1]))) {
    rank_peer = MPI_PROC_NULL;
  }
  else {
    MPI_Cart_rank(comm_cart, cord, &rank_peer);
  }
  srcs[ii] = dsts[ii] = rank_peer;
}

MPI_Dist_graph_create_adjacent(comm_cart,
  nsrc, srcs, swts, ndst, dsts, dwts,
  info, rord, &comm_dg);
```



- 2D 9-pnt. Stencil の プロセス・トポロジの生成
  - 座標計算による MPI_Dist_graph_create_adjacent
- MPI_CART→ MPI_DIST_GRAPH
  - MPI_Cart ユーティリティ関数が利用可能
  - 簡単かつポータビリティ高
- MPI_Cart_get()
  - ords[2]：そのカルテジアンにおける自ランクの座標
  - cord[2]：各隣接ランクの座標
- MPI_Cart_rank()
  - 座標 → ランク

```
int srcs[8] /* = { nn, ss, ww, ee, nw, se, ne, sw } */;
int dsts[8] /* = { nn, ss, ww, ee, nw, se, ne, sw } */;
```

隣接用 コミュニケータ (4)

- 2D 9-point ステンシル通信パターンの定義 （つづき）
  - 前スライドの送受信の通信ランクの定義 (MPI_Dist_graph_create_adjacent)

```
int srcs[8] /* = { nn, ss, ww, ee, nw, se, ne, sw } */;
int dsts[8] /* = { nn, ss, ww, ee, nw, se, ne, sw } */;
```

  - 上記のランク配列の index（通信相手）に従って、MPI_Neighbor_alltoallw の引数を生成

```
int MPI_Neighbor_alltoallw(
  const void *sendbuf, const int sendcounts[], const MPI_Aint sdispls[], const MPI_Datatype sendtypes[],
  void *       recvbuf, const int  recvcounts[], const MPI_Aint  rdispls[], const MPI_Datatype  recvtypes[],
  MPI_Comm comm);
```

    - srcs[] は recv 側引数に対応、dsts[] は send 側引数に対応する。
    - sendbuf / sdispls[] で各送信相手のバッファの先頭が決定される。recvbuf rdispls[] も同様。それ以外は MPI_Isend / Irecv と同じ

隣接用 コミュニケータ (5)

- 2D 9-point ステンシル通信パターンの定義 （まとめ）

```
MPI_Cart_create( comm_orig, ... , &comm_cart );
MPI_Dist_graph_create_adjacent ( comm_cart, ... , &comm_dg );
MPI_Neighbor_alltoallw ( ... , comm_dg);
```

- MPI_Dist_graph_create_adjacent

```
int MPI_Dist_graph_create_adjacent( MPI_comm comm_old,
  int indegree, const int sources[], const int sourceweights[],
  int outdegree, const int destinations[], const int destweights[],
  MPI_Info info, int reorder,  MPI_Comm *comm_dist_graph);
```

  - より一般化 (Cartesian (長方形格子等) の要件を緩和)した MPI_DIST_GPRAH タイプのトポロジ対応コミュニケータを生成

MPI-4 永続集団通信

- MPI-3 仕様で、非同期集団通信が導入された

```
int MPI_Ineighbor_alltoallw(
  const void *sendbuf, const int sendcounts[], const MPI_Aint sdispls[], const MPI_Datatype sendtypes[],
  void *       recvbuf, const int recvcounts[], const MPI_Aint rdispls[], const MPI_Datatype recvtypes[],
  MPI_Comm comm, MPI_Request *request );
```

– MPI_Isend /Irecv のように、通信プログレスの問題が発生 (MPI_Testall ...)

– MPI-4 では「永続集団通信」が検討されている

```
int MPI_Neighbor_alltoallw_init(
  const void *sendbuf, const int sendcounts[], const MPI_Aint sdispls[], const MPI_Datatype sendtypes[],
  void *       recvbuf, const int recvcounts[], const MPI_Aint rdispls[], const MPI_Datatype recvtypes[],
  MPI_Comm comm, MPI_Request *request );
int MPI_Start( MPI_Request *request );
int MPI_Wait ( MPI_Request *request );
```

まとめ

- まとめ
  - 「京」でも、MPI-3 実装が利用可能
  - 使い方は、コンパイラの変更と mpichexec だけ
    - 機能面でいくつか制限が存在
  - MPI-3 では、MPI_Dist_graph_create_adjacent + MPI_Neighbor_alltoallw を使って、「ユーザ定義の通信パターン」の集団通信が可能
    - 重要な通信パターンだが、集団通信関数がなかった Halo 領域交換にも適用可

今後の展開 (1)

- マイルストーン

OFI	Open Fabric Interface by Open Fabrics Alliance (infiniband) IB Verbs replacement
tlib	Tofu Libraries by Fujitsu in top of Tofu Hardware

今後の展開 (2)

- 非ブロッキング集団通信
  - 計算と通信の同時並行が集団通信でも可
- 隣接集団通信
  - 「隣接」に限らず、固定通信パターンをユーザ定義して集団通信化可能
- 片側通信
  - MPI_UNIFIED 新メモリモデル片方向通信
    - MPI_SEPARATE (MPI-2) メモリモデルでは、public ウィンドウと private ウィンドウに分かれ、同期に時間がかかった
    - MPI_UNIFIED：RDMA 通信向き public = privateメモリモデル
    - RDMA を駆使した高速実装も可

今後の展開 (3)

- 次期 MPI-4 仕様　実装早期提供
  - Persistent Collective：
    - MPIX_Neigibor_alltoallw_init()
    - MPI_Start()
    - MPI_Wait()
    - MPI_Request_free()

Backup Slides