FUJITSU

shaping tomorrow with you

低レベル通信ライブラリACPのPGASランタイム向け機能

2014年10月24日

富士通株式会社、JST CREST
安島 雄一郎

本発表の構成

FUJITSU

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

ACEプロジェクトとACPライブラリ

FUJITSU

- エクサスケールの時代はプロセッサのメニーコア化が進む
  - コアあたりのメモリ容量は減少

- Advanced Communication for Exa（ACE）プロジェクト
  - 米澤CREST第二期 南里T

    http://ace-project.kyushu-u.ac.jp/index.html
  - 新しい通信システムソフトウェア技術の創出
  - 低遅延だけでなく、省メモリにも重点を置く

- Advanced Communication Primitives（ACP）ライブラリ
  - ACEプロジェクトの中核技術となる低レベル通信ライブラリ
  - メモリ消費量を明示的に制御

ACPライブラリの特徴

- 低レベル通信ライブラリ
  - インターコネクト・デバイスを直接サポート
    - Ethernet（UDP）
    - InfiniBand
    - Tofuインターコネクト
  - 上位層には通信ライブラリ、並列言語処理系のランタイムを想定
- 省メモリプロトコル、アルゴリズムを提供
  - エクサスケールの実行環境は1ノード多プロセス
  - 各プロセスにおける、通信のメモリ使用量削減が重要
- ACP自身も省メモリ

本発表の構成

FUJITSU

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

ACP-1.0 のインタフェース構成

- チャネル
  - 明示的に生成、破棄して使用する、プロセス間片方向通信チャネル
- ベクタ、リスト
  - 複数プロセスに跨って生成、操作、破棄できるデータ構造
- メモリアロケータ
  - 任意のプロセスからメモリを確保、解放
- アドレス変換
  - ローカルメモリにグローバルアドレスを割り当て
- グローバルメモリ参照
  - グローバルアドレス上のデータ転送、アトミック演算

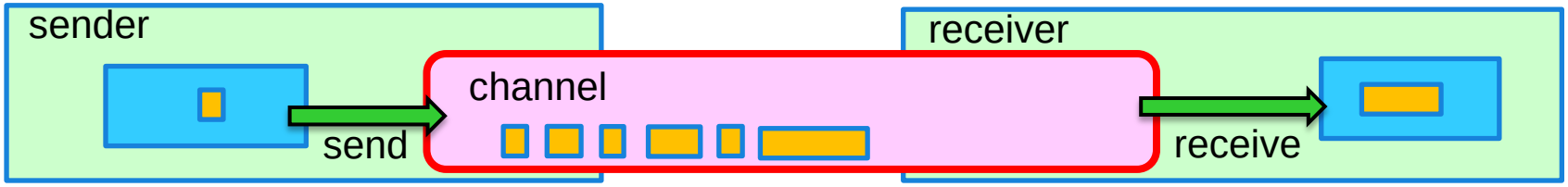
本発表の構成

FUJITSU

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

チャネル

- プロセス間の片方向通信チャネル
- 明示的に生成、破棄
  - 「必要十分な期間」だけ、通信バッファを割当て
  - 早期の明示的なチャネル生成による、オーバーヘッド隠蔽を期待




```
acp_ch_t ch = acp_create_ch( … );
loop {
        acp_wait_ch(acp_nbsend(ch, … ));
}
acp_free_ch(ch);
```

```
acp_ch_t ch = acp_create_ch( … );
loop {
        acp_wait_ch(acp_nbrecv(ch, … ));
}
acp_free_ch(ch);
```

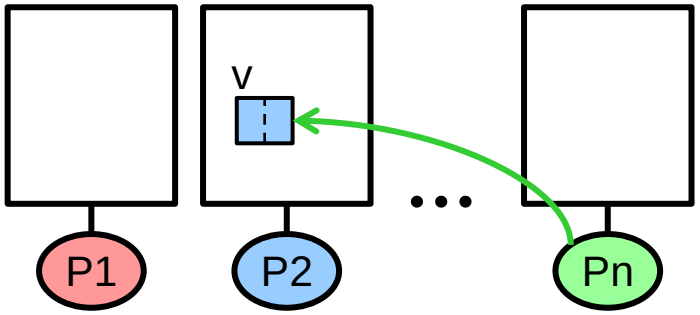
- チャネルは用途毎に生成することを想定
  - 複数用途で通信バッファを共有するためのタグ機能は付けない

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

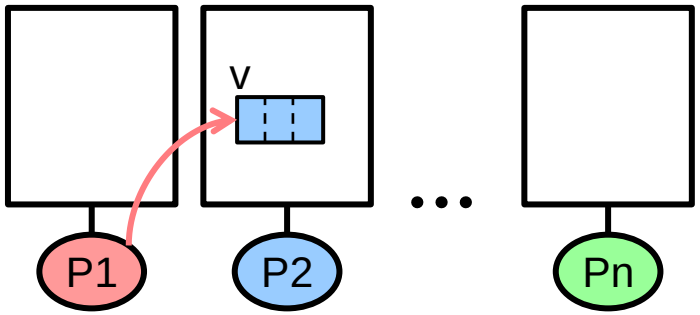
ベクタ型データ構造

- 任意のプロセスのメモリ上に生成できる可変長配列
  - 単一プロセスのメモリ上で閉じるので、グローバル配列ではない



v = acp_create_vector(… , 2)

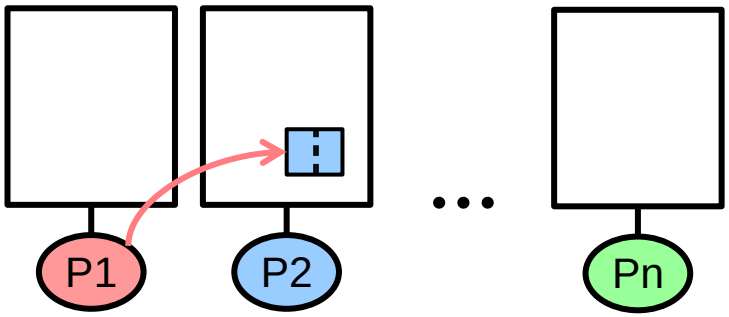
- 他のプロセスから参照、要素追加、要素削除が可能
  - ただし、生成したプロセスからベクタのグローバルアドレスを得る必要あり



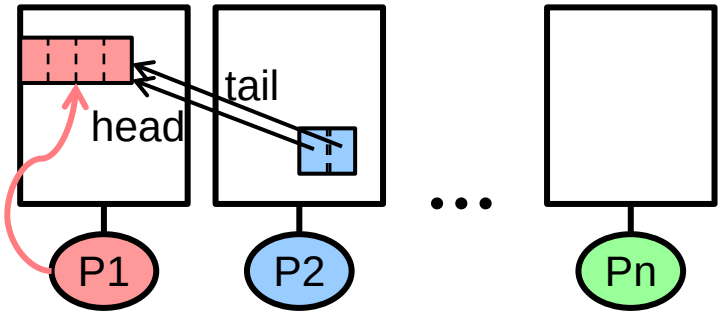
acp_push_back_vector(v, …)

リスト型データ構造

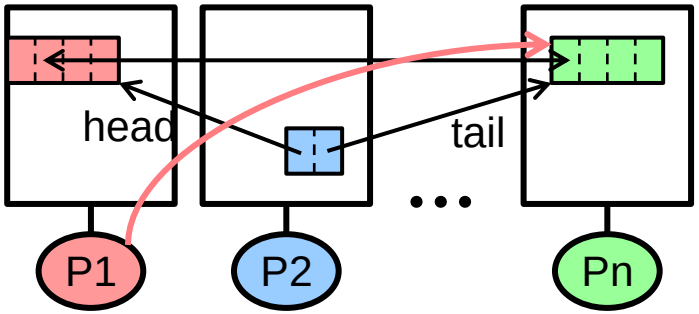
- 要素毎に異なるプロセスに配置できる双方向リスト
- 管理情報（空リスト）を配置するプロセスも指定可能
- 要素サイズは固定、空リスト生成時に決定



acp_create_list(… , 2)



acp_push_back_list(… , 1)



acp_push_back_list(… , n)

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

メモリアロケータ関数

- 割当: acp_malloc()
  - サイズと割当プロセス番号を指定してメモリ割当て
  - 割り当てた領域の先頭グローバルアドレスを返す
- 解放: acp_free()
  - 解放する領域の先頭グローバルアドレスを指定

名称	定義
割当	acp_ga_t **acp_malloc**(size_t *size*, int *rank*);
解放	void **acp_free**(acp_ga_t *ga*);

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

アドレス変換関数

- グローバルアドレスは64ビット
  - グローバルアドレス(ポインタ)に対するアトミック演算が可能
  - 16エクサバイトをアドレッシング可能で、容量は十分

- グローバルアドレス取得手順
  1. 論理アドレスとサイズを指定して、メモリを登録する
  2. メモリを登録するとアドレス変換キーが得られる
  3. 登録済みメモリのグローバルアドレスは、アドレス変換キーを使用して取得

名称	定義
メモリ登録	acp_atkey_t **acp_register_memory** (void* *addr*, size_t *size*, int *color*);
グローバルアドレス取得	acp_ga_t **acp_query_ga**(acp_atkey_t *atkey*, void* *addr*);
メモリ登録解除	int **acp_unregister_memory**(acp_atkey_t *atkey*);

グローバルアドレス関連関数

- 逆引き
  - グローバルアドレスからローカルアドレス、ランク番号に変換可能
- スターターメモリ
  - ACPライブラリが予め用意するグローバルメモリ
  - 静的共有変数の配置場所
  - 任意のプロセスのスターターアドレスを取得できる

名称	定義
論理アドレス取得	void* **acp_query_address**(acp_ga_t *ga*);
ランク番号取得	int **acp_query_rank**(acp_ga_t *ga*);
スターターアドレス取得	acp_ga_t **acp_query_starter_ga**(int *rank*);

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

グローバルメモリ参照（GMA）関数

- コピー: acp_copy()
  - 転送元と転送先に任意のグローバルアドレスを指定
  - 非ブロッキング、GMAハンドルを返す
  - 開始条件（GMA間依存関係）を指定可能
- 不可分比較交換: acp_cas8(), acp_cas4()
  - コピーと同様だが、サイズの代わりに比較交換のオペランドを指定
- GMA完了: acp_complete()
  - GMAが終了するまで待ち合わせる

名称	定義
コピー	acp_handle_t **acp_copy**(acp_ga_t *dst*, acp_ga_t *src*, size_t *size*, acp_handle_t *order*);
不可分比較交換	acp_handle_t **acp_cas8**(acp_ga_t *dst*, acp_ga_t *src*, uint64_t *oldval*, uint64_t *newval*, acp_handle_t *order*);
GMA完了	void **acp_complete**(acp_handle_t *handle*);

GMA間依存関係指定のサンプルコード

FUJITSU

- 非ブロッキングのAllGather集団通信
  - 各プロセスがデータを二分木で全プロセスにBroadcast
    - Broadcastは非ブロッキング・集中制御型

⇒ 全プロセスがBroadcastすることで、全体としてAllGather通信を実現

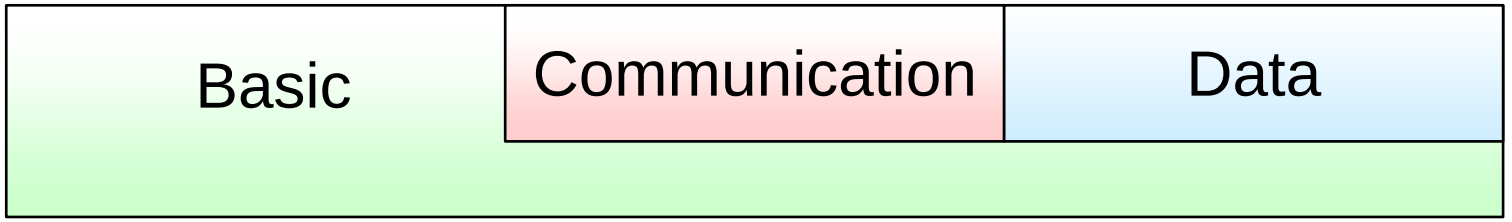
```
rank = acp_rank();
procs = acp_procs();
handle[rank] = ACE_HANDLE_NULL;
for ( i = 1 ; i < procs ; i++ ) {
    dst = (rank +  i    ) % procs;
    src = (rank + (i>>1)) % procs;
    handle[dst] = acp_copy(
        acp_query_startar_ga(dst) + n * rank,
        acp_query_startar_ga(src) + n * rank,
        n,
        handle[src]
    );
}
acp_complete(ACP_HANDLE_ALL);
acp_sync();
```

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

ACPのモジュール構成

- 基本層（Basic Layer）
  - アドレス変換、グローバルメモリ参照を実装
  - デバイス依存
- コミュニケーション・ライブラリ（Communication Library）
  - チャネルを実装
  - デバイス非依存
- データ・ライブラリ（Data Library）
  - ベクタ、リスト、メモリアロケータを実装
  - デバイス非依存



ACPライブラリの構成

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

メモリ消費量

- 基本層のメモリ消費量を机上計算
  - プロセス数は100万、メモリ登録数は128を想定
- Tofu版はプロセスあたり約70MB
  - ノードあたり8プロセスの場合、ノードあたり合計0.55GB

	Tofu	UDP
プロセス数比例	69MB 1プロセスあたり • コマンド受信バッファ 64B • Tofuアドレステーブル 4B • Tofu経路テーブル 1B	18MB 1プロセスあたり • 待ち受けIPアドレス 4B • 待ち受けポート番号 2B • 送信シーケンス番号 4B • 受信シーケンス番号 4B • ランク番号 4B
メモリ登録数比例	7KB 1メモリ登録あたり • 登録メモリ管理テーブル 40B • 論理アドレス検索テーブル 16B	
その他、固定	264KB • コマンドキュー兼コマンド送信バッファ 64B × 4,096 • 論理アドレス検索テーブル 8B × 256	647KB • コマンドキュー 80B × 4,096 • コマンドステーション 1,504B × 64 • デリゲートステーション 3,480 × 64
合計	**約70MB**	**約19MB**

評価環境

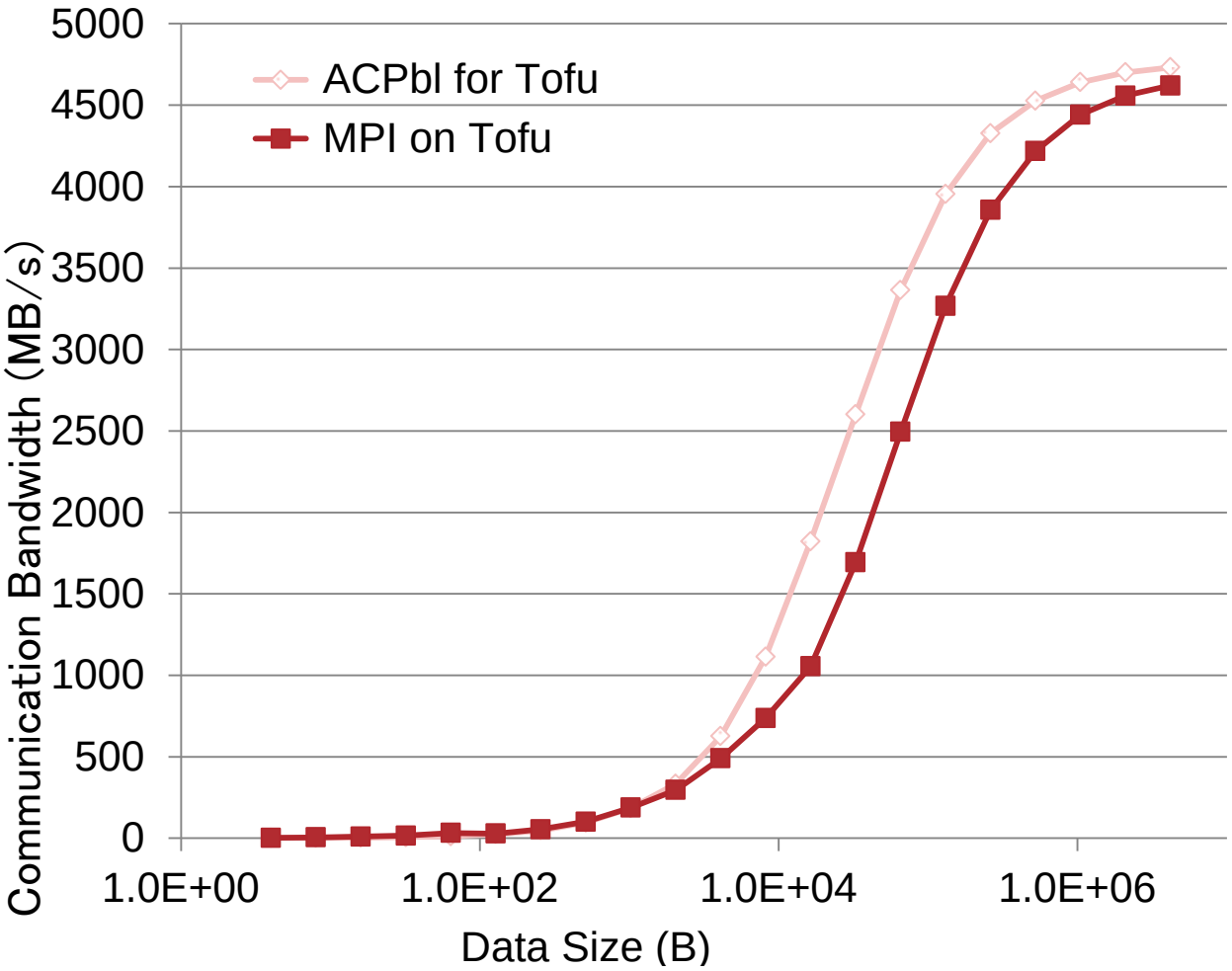
FUJITSU

- PCクラスタとスーパーコンピュータで評価

	PC Cluster	Supercomputer
Node	Fujitsu PRIMERGY RX200 S5	Fujitsu PRIMEHPC FX10
CPU	Intel Xeon E5520, 2 sockets (4 cores, 2.27 GHz)	Fujitsu SPARC64TM IXfx (16 cores, 1.848 GHz)
Memory	48GB, DDR3 1066MHz	64GB, DDR3 1333MHz
Network	Gigabit Ethernet (125 MB/s)	Tofu interconnect (5.0 GB/s)
OS	Linux version 2.6.32	Linux version 2.6.52.8

- PCクラスタ（GbE）ではUDP版基本層を使用
- スーパーコンピュータ（FX10）ではTofu版基本層を使用

基本層のスループット



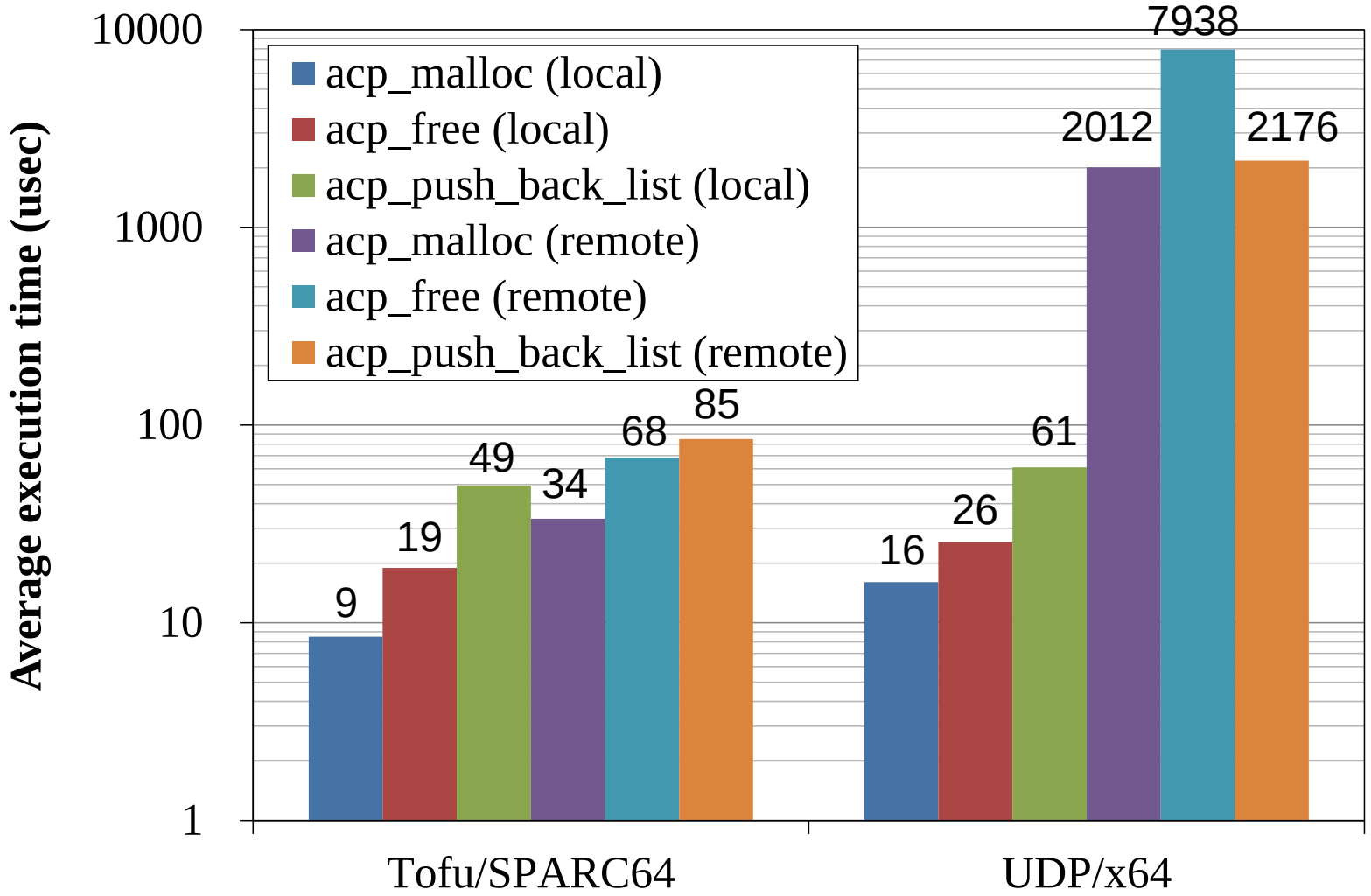
- 約10KB以上のデータでは、ACPはMPIよりスループットが高い
  - MPIのオーバーヘッドはRendezvousプロトコルが原因と考えられる

関数実行時間

- 評価対象関数
  - グローバルメモリ割当関数 acp_malloc()
  - グローバルメモリ解放関数 acp_free()
  - リスト要素追加関数 acp_push_back_list()

- 評価方法
  - 割当: 100回連続で呼び出し、平均実行時間を計測
    - 割当先プロセス: 他ノードのプロセス(remote)と自ノードのプロセス(local)
    - 割当サイズ: 1バイト以上32768バイト以下の乱数で毎回変更
  - 解放: 割当てた100個のメモリを連続で解放し、平均実行時間を計測
    - 解放順序: 割当順ではなく、ランダムな順番
  - 要素追加: 100回連続で呼び出し、平均実行時間を計測
    - 要素配置プロセス: 他ノードのプロセス(remote)と自ノードのプロセス(local)

関数実行時間評価結果



- Tofu版ではlocalとremoteで実行時間は2～4倍しか変わらない
- UDP版ではlocalとremoteで実行時間が30～300倍遅くなる

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

Ruby, PythonへのCoarray拡張実装

FUJITSU

- Rubyの内部DSLとしてCoArrayを拡張

```
array = LLPGAS::CoArray.new(:int, 5) # CoArrayの生成は集団的
x = array[0]                         # ローカル配列の参照
array.at(1)[3] = y                   # リモート配列への代入
```

- 通信部分はACPを呼び出し
  - 内部でRemote-to-remoteコピーも実現
  - Proxyオブジェクトにより、グローバルアドレスの評価を代入まで遅延
- 実装量
  - インターフェース部(FFI)を除くと500行程度
  - このうちProxyオブジェクトは40行程度
- Python版実装量
  - Proxyオブジェクト以外はRubyと同程度
  - Proxyオブジェクト部は100行程度

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

今後の課題

- 集団通信、隣接通信インタフェースの拡張
  - 複数プロセス間を接続する有向チャネル
  - チャネル系インタフェースの高級版
- データ構造系インタフェースの拡張
  - キュー、集合、連想配列
- グローバルメモリアロケータのアルゴリズム改良
- 基本層のTofuインターコネクト2対応

本発表の構成

- 概要
- インタフェース
  - チャネル
  - ベクタ、リスト
  - メモリアロケータ
  - アドレス変換
  - グローバルメモリ参照
- モジュール構成
- メモリ消費量と性能評価
- 利用例
- 今後の課題
- まとめ

まとめ

- ACPは低レベル通信ライブラリ
  - インターコネクト・デバイスを直接サポートし、低オーバーヘッド
  - 省メモリプロトコル、アルゴリズムを提供
- 明示的にメモリを使用するインタフェース
  - チャネル、ベクタ、リスト、メモリアロケータ
  - アドレス変換、グローバルメモリ参照
- モジュール化実装
  - アドレス変換、グローバルメモリ参照を含む基本層のみ、デバイス依存
- 100万プロセス時に、プロセスあたり数十MBのメモリ使用量
- 性能評価
  - Tofu版ではlocalとremoteで実行時間は2～4倍しか変わらない
  - UDP版ではlocalとremoteで実行時間が30～300倍遅くなる
- Ruby, PythonのCoArray拡張を500行程度の実装量で実現

FUJITSU

shaping tomorrow with you