Rhpcパッケージ入門

Ei-ji Nakama

July 21, 2026

1 概要

Rhpcパッケージは、Message Passing Interface (MPI) を利用して、Rでハイパフォーマンスコンピューティング (HPC) を実現するためのインターフェースを提供します。ソケットや fork に依存する標準的な並列処理ライブラリとは異なり、MPIへの直接的かつ低レイテンシなアクセスを提供するため、マルチコアPCから大規模な計算クラスタまで対応可能です。

1.1 主な仕様

- **MPI専用API**: すべての関数に `Rhpc_` プレフィックスが付与されており、名前空間の衝突を避け、MPI バックエンドであることを明示します（例: `Rhpc_lapply`, `Rhpc_worker_call`）。
- **動的なワーカー管理**: `mpirun` による静的起動と、実行時の `MPI_Comm_spawn` による動的生成の両方をサポートします。
- **ハイブリッド通信**: 命令の配布には一括送信 (Collective communication) を、結果の回収には個別送信 (Point-to-Point) を使用し、スループットを最大化します。
- **Windows 対応**: `fakemaster` 機構により、RGui や RStudio などの GUI 環境からでも複雑なコマンドライン起動なしに MPI を利用できます。
- **long vector 対応**: やや大きなデータにも対応できるよう設計されています。

2 インストールと起動方法

2.1 Linux / Unix

MPI ライブラリ（OpenMPI、MPICH2 など）が必要です。パッケージをビルドしてインストールします。

```
R CMD INSTALL Rhpc_0.26.1.tar.gz
```

Fujitsu MPI など他の MPI 実装を使う場合は、`-configure-args` でコンパイル・リンクフラグを指定します。

2.1.1 mpirun による起動（バッチ実行）

パッケージ内に生成される Rhpc シェルを使い、複数プロセスで R を起動します。

```
mpirun -n 4 ~/R/.../Rhpc/Rhpc CMD BATCH -q --no-save test.R
```

2.2 Windows (MS-MPI)

Microsoft MPI (MS-MPI) をインストールし、環境変数 MSMPI_INC、MSMPI_LIB32、MSMPI_LIB64 を設定します。 RGui や RStudio から Rhpc_initialize() を呼ぶだけで、mpiexec と fakemaster が自動起動されます。

3 基本的な使い方

Rhpcを使用するには、まず MPI 環境を初期化し、ワーカーのハンドル (cl) を取得します。 Rhpc_getHandle() の動作は呼び出し方によって異なります。

- **引数なし**: R がすでに mpirun や mpiexec 下で起動されている場合、既存のワーカープロセスを取得します。
- **数値引数**: 指定した数のワーカープロセスを動的に生成します（例: Rhpc_getHandle(4)）。

典型的なワークフローは Rhpc_initialize() → Rhpc_getHandle() → 並列処理 → Rhpc_finalize() です。

```
> library(Rhpc)
> # Rhpc 環境の初期化
> Rhpc_initialize()
> # ハンドルの取得（mpirun 下では引数なし、動的生成時は Rhpc_getHandle(4) など）
> cl <- Rhpc_getHandle()
> # 全ワーカーで関数を実行し、マスターに結果を集約
> pids <- Rhpc_worker_call(cl, Sys.getpid)
> print(pids)
> # 結果を返さずにワーカー上でコードを実行する場合
> Rhpc_worker_noback(cl, function() {
+     cat("worker process:", Sys.getpid(), "\n")
+ })
> # 終了処理
> Rhpc_finalize()
```

4 Windows でのバックエンド管理

Windows では Rhpc_initialize() が次の処理を内部で行います。

1. 現在の R プロセスが `mpiexec` 下で動いていなければ `fakemaster` を起動する。
2. `mpiexec` 経由で `fakemaster.exe` を起動し、名前付きパイプで R プロセスと接続する。
3. `fakemaster` が生成した MPI 環境変数を R プロセスに取り込む。
4. これにより GUI ベースの R セッションでも `Rhpc_getHandle()` や `Rhpc_worker_call()` が利用可能になる。

注意点:

- `Rhpc_initialize()` 実行時に `mpiexec` や `fakemaster` のウィンドウがバックグラウンドで起動します。これらを閉じると MPI 接続が切れ、処理が失敗します。
- `options(Rhpc.mpiexec="mpiexec -n 1")` で `mpiexec` のコマンドラインを指定できます。

```
> library(Rhpc)
> oldoptions <- options(Rhpc.mpiexec = "mpiexec -n 1")
> Rhpc_initialize()
> cl <- Rhpc_getHandle(3)
> worker_env_info <- Rhpc_worker_call(cl, function() {
+     list(
+         computer_name = Sys.getenv("COMPUTERNAME"),
+         username = Sys.getenv("USERNAME")
+     )
+ })
> print(worker_env_info)
> Rhpc_finalize()
> options(oldoptions)
```

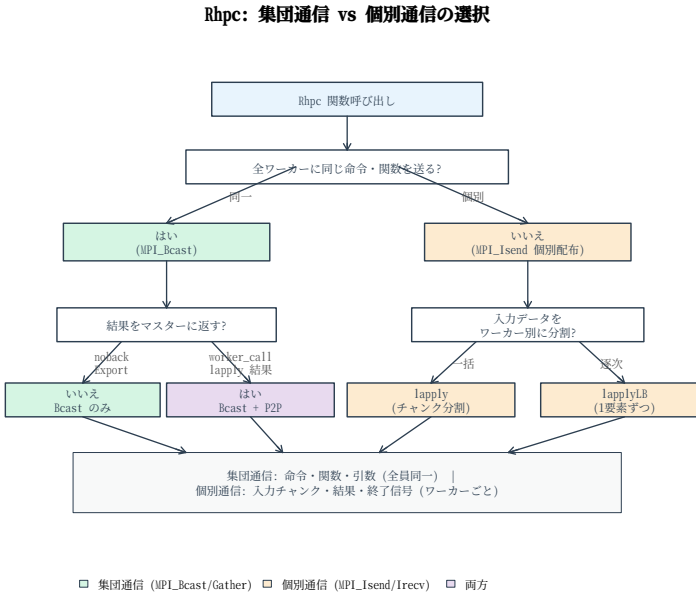
5 集団通信と個別通信の選択

`Rhpc` は MPI の **集団通信**（Collective）と **個別通信**（Point-to-Point, P2P）を用途に応じて使い分けます。選択の基本原則は次のとおりです。

- **全ワーカーに同じ内容を送る** → 集団通信 (`MPI_Bcast`)
- **ワーカーごとに内容が異なる** → 個別通信 (`MPI_Isend` / `MPI_Irecv`)
- **結果のサイズがワーカーごとに異なる** → 個別通信（非同期受信）
- **結果を返さない** → 集団通信のみで完結

5.1 通信方式の選択フロー

次の図は、Rhpc が各操作でどの通信方式を選ぶかを示すフローチャートです。



5.2 MPI 関数と役割

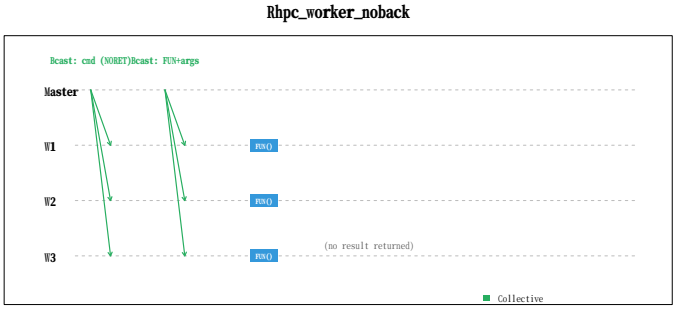
通信種別	MPI 関数	Rhpc での用途
集団通信	`MPI_Bcast`	命令・関数・引数を全ワーカーへ一括配布
集団通信	`MPI_Gather`	各ワーカーの結果サイズ情報をマスターへ集約
個別通信	`MPI_Isend`	マスター→ワーカー（入力チャンク）、ワーカー→マスター（結果）
個別通信	`MPI_Irecv`	マスターが各ワーカーから結果を非同期受信
個別通信	`MPI_Probe`	`lapply`/`lapplyLB` で完了したワーカーを検出

5.3 関数別シーケンス図

以下に、主要関数ごとの通信シーケンス図を示します。図はすべて R の `plot` で描画しています。

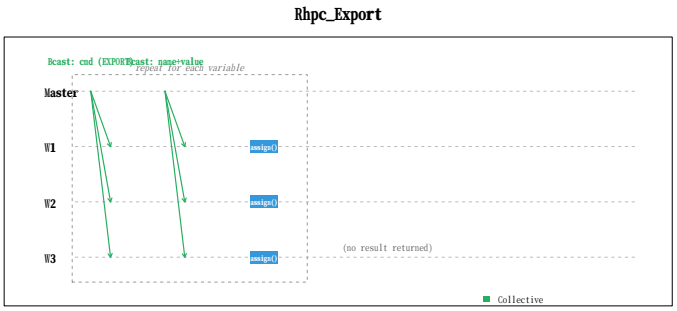
5.3.1 Rhpc_worker_noback()

結果を返さないワーカー実行。集団通信 (`MPI_Bcast`) のみで完結します。



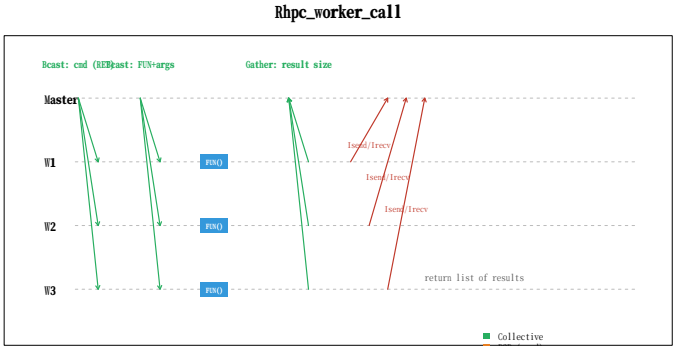
5.3.2 Rhpc_Export()

変数を全ワーカーのグローバル環境へ配布。`Rhpc_worker_noback` と同様 `MPI_Bcast` のみですが、ワーカー側では `assign()` が実行されます。変数ごとに Bcast が繰り返されます。



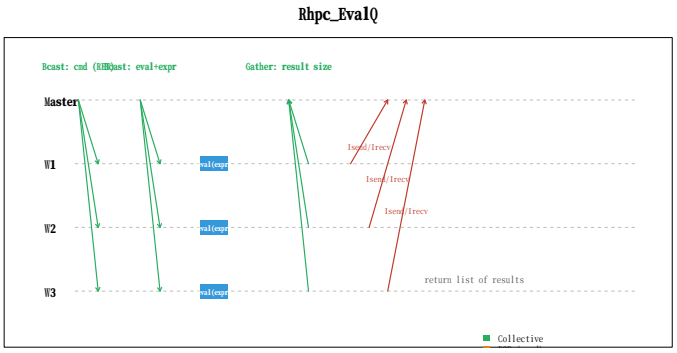
5.3.3 Rhpc_worker_call()

全ワーカーで同一関数を実行し、結果をマスターへ返します。命令配布は Bcast、結果サイズは Gather、結果本体は P2P (Isend/Irecv) です。



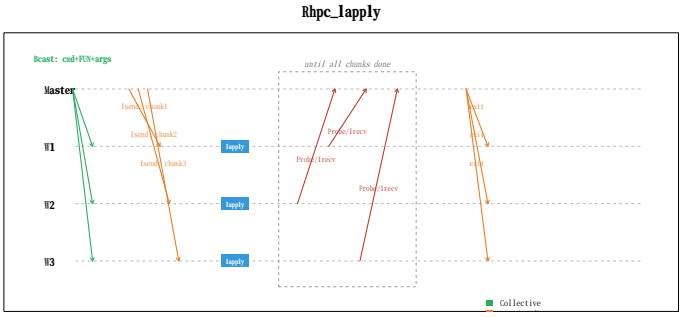
5.3.4 Rhpc_EvalQ()

`Rhpc_worker_call(cl, eval, substitute(expr))` のラッパーです。通信パターンは `Rhpc_worker_call` と同一です。



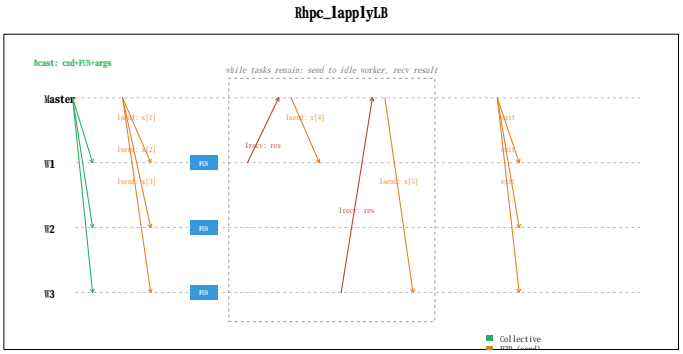
5.3.5 Rhpc_lapply()

入力 `X` をワーカー数に応じてチャンク分割し、関数は Bcast、入力チャンクは P2P で配布します。結果は `MPI_Probe`/`MPI_Irecv` で完了順に回収し、最後に終了信号を P2P 送信します。



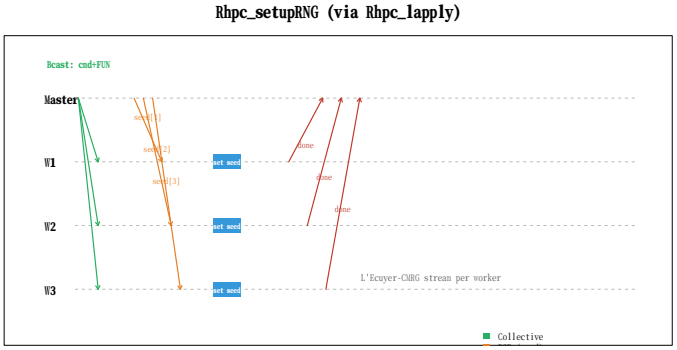
5.3.6 Rhpc_lapplyLB()

関数は Bcast しますが、入力は **1要素ずつ** 空きワーカーへ P2P 配布します。ワーカーが完了するたびに次の要素を送り、結果も逐次 P2P で回収します。



5.3.7 Rhpc_setupRNG()

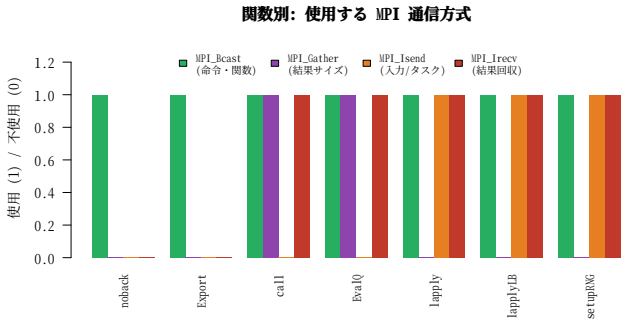
`Rhpc_getHandle()` 内で自動呼び出されます。内部で `Rhpc_lapply` を使い、各ワーカーへ異なる乱数シードを P2P 相当で配布します（`lapply` と同じ通信パターン）。



5.4 通信方式の比較

次の図は、主要関数ごとの通信方式の違いを一覧で示します。

```
> draw_comm_comparison <- function() {
+     funcs <- c("noback", "Export", "call", "EvalQ", "lapply", "lapplyLB", "setupRNG")
+     bcast <- c(1, 1, 1, 1, 1, 1, 1)
+     gather <- c(0, 0, 1, 1, 0, 0, 0)
+     isend_in <- c(0, 0, 0, 0, 1, 1, 1)
+     irecv_out <- c(0, 0, 1, 1, 1, 1, 1)
+
+     mat <- rbind(
+         "MPI_Bcast\n(命令・関数)" = bcast,
+         "MPI_Gather\n(結果サイズ)" = gather,
+         "MPI_Isend\n(入力/タスク)" = isend_in,
+         "MPI_Irecv\n(結果回収)" = irecv_out
+     )
+     colnames(mat) <- funcs
+
+     oldpar <- par(mar = c(6, 8, 3, 2))
+     on.exit(par(oldpar))
+     barplot(mat, beside = TRUE, col = c("#27AE60", "#8E44AD", "#E67E22", "#C0392B"),
+             border = NA, las = 2, cex.names = 0.85,
+             main = "関数別: 使用する MPI 通信方式",
+             ylab = "使用 (1) / 不使用 (0)", ylim = c(0, 1.35))
+     legend("top", inset = 0.02, horiz = TRUE, bty = "n", cex = 0.75,
+            legend = rownames(mat),
+            fill = c("#27AE60", "#8E44AD", "#E67E22", "#C0392B"))
+ }
> draw_comm_comparison()
```



5.5 なぜこの使い分けか

- **Bcast を命令に使う理由**: 関数と引数は全ワーカーで同一です。1回のシリアライズと木構造配布 ($O(\log N)$) で、各ワーカーへ順次送信する $O(N)$ より効率的です。
- **P2P を入力データに使う理由 (`lapply`)**: 各ワーカーへの入力チャンクは内容・サイズが異なります。Bcast は全員に同じデータを送るため不適切です。
- **P2P を結果に使う理由**: 各ワーカーの結果サイズは実行時まで不明で、ワーカー間で異なります。`MPI_Irecv` による非同期受信で、速いワーカーの結果を先に処理できます。
- **`lapplyLB` が P2P 逐次配布を選ぶ理由**: タスクの実行時間にばらつきがある場合、事前チャンク分割 (`lapply`) より空きワーカーへの動的割当 (`lapplyLB`) の方が CPU 利用率が高くなります。

6 主要な関数一覧

6.1 環境管理

- `Rhpc_initialize()`: MPI 環境を初期化する。
- `Rhpc_finalize()`: MPI 環境を終了する。
- `Rhpc_getHandle(procs)`: ワーカークラスタのハンドルを取得する。`procs` に数値を指定すると動的生成する。
- `Rhpc_numberOfWorker(cl)`: ワーカー数を返す。

6.2 ワーカー上での実行

- `Rhpc_worker_call(cl, FUN, ...)`: 全ワーカーで `FUN` を実行し、結果のリストを返す。

- `Rhpc_worker_noback(cl, FUN, ...)`: 全ワーカーで `FUN` を実行するが、結果は返さない。
- `Rhpc_EvalQ(cl, expr)`: 全ワーカーで式 `expr` を評価する（`clusterEvalQ` 相当）。

6.3 並列 *apply 系

- `Rhpc_lapply(cl, X, FUN, ...)`: `lapply` と同様にリスト/ベクトルを並列処理する。入力をチャンク分割して各ワーカーに配布する。
- `Rhpc_lapplyLB(cl, X, FUN, ...)`: ロードバランシング版。タスクを空いたワーカーに逐次割り当て、実行時間にばらつきがある場合に有効。
- `Rhpc_sapply(cl, X, FUN, ...)`: `sapply` 相当。結果をベクトルや行列に簡略化できる。
- `Rhpc_sapplyLB(cl, X, FUN, ...)`: ロードバランシング版 `sapply`。
- `Rhpc_apply(cl, X, MARGIN, FUN, ...)`: `apply` 相当。配列の指定次元に沿って並列適用する。

6.4 データ共有とユーティリティ

- `Rhpc_Export(cl, variableNames)`: マスターで定義した変数を全ワーカーのグローバル環境に配布する。
- `Rhpc_setupRNG(cl, iseed)`: 各ワーカーに独立した乱数ストリーム (L'Ecuyer-CMRG) を設定する。`Rhpc_getHandle()` 内で自動呼び出される。
- `Rhpc_enquote(...)`: 引数をクォートする内部ユーティリティ。
- `Rhpc_splitList(var, num)`: リストを指定数に分割する。
- `Rhpc_serialize()` / `Rhpc_unserialize()`: カスタムシリアライズ関数。

7 並列マッピング (Rhpc_lapply)

`Rhpc_lapply` は `lapply` と同じインターフェースで、入力データをワーカー数に応じてチャンク分割し並列実行します。

```
> Rhpc_initialize()
> cl <- Rhpc_getHandle()
> input_data <- 1:10
> results <- Rhpc_lapply(cl, input_data, function(x) {
+     return(x^2)
```

```
+ })
> print(unlist(results))
> Rhpc_finalize()
```

8　ロードバランシング (Rhpc_applyLB)

各要素の実行時間にばらつきがある場合、Rhpc_applyLB が有効です。事前チャンク分割ではなく、空いたワーカーにタスクを逐次割り当てるため、CPU 利用率が向上します。

```
> Rhpc_initialize()
> cl <- Rhpc_getHandle()
> input_data <- 1:10
> results_lb <- Rhpc_applyLB(cl, input_data, function(x) {
+     Sys.sleep(runif(1, 0, 1))  # 実行時間のばらつきを模擬
+     return(x * 10)
+ })
> print(unlist(results_lb))
> Rhpc_finalize()
```

9　sapply / apply の利用

Rhpc_sapply や Rhpc_apply も parallel パッケージと同様の使い方ができます。

```
> Rhpc_initialize()
> cl <- Rhpc_getHandle()
> # sapply 相当
> res <- Rhpc_sapply(cl, 1:10000, sqrt)
> # apply 相当
> df <- data.frame(a = 1:4, b = 5:8)
> Rhpc_apply(cl, df, 1, max)  # 行方向
> Rhpc_apply(cl, df, 2, max)  # 列方向
> Rhpc_finalize()
```

10　変数のワーカーへのエクスポート

ワーカー側でマスターで定義した変数を使う場合、Rhpc_Export でグローバル環境に配布します。

```
> Rhpc_initialize()
> cl <- Rhpc_getHandle()
> multiplier <- 10
> Rhpc_Export(cl, "multiplier")
```

```
> res <- Rhpc_worker_call(cl, function() {
+     return(multiplier * 2)
+ })
> print(res)
> Rhpc_finalize()
```

11 オプションと MPI 属性

Rhpc_initialize() 後、次のオプションが options() に設定されます。

- Rhpc.mpi.rank: 現在の MPI ランク
- Rhpc.mpi.procs: 総プロセス数
- Rhpc.mpi.c.comm: C 言語 MPI コミュニケータ（ポインタ）
- Rhpc.mpi.f.comm: Fortran MPI コミュニケータ

usequote 引数（デフォルト TRUE）は options(Rhpc.usequote=FALSE) で無効化でき、パフォーマンス向上が期待できます。

12 まとめ

Rhpcを利用することで、R の統計プログラミングの柔軟性と MPI の高いパフォーマンスを両立できます。主な利点は次のとおりです。

- **効率性**: ソケットベースの並列処理と比べ通信オーバーヘッドが小さい。
- **スケーラビリティ**: ローカルマルチコアから大規模クラスタへの移行がスムーズ。
- **簡便性**: lapply や apply と同じ R スタイルの構文で並列化できる。
- **Windows 対応**: GUI 環境からでも fakemaster により MPI を利用可能。