CPM: A software tool for Communication Performance Modelling

Version 1.1.0 (Revision 226)

Generated by Doxygen 1.7.1

Sun Sep 12 2010 11:11:30

Contents

1 Introduction

Traditionally, communication performance models for high performance computing are analytical and built for homogeneous clusters. The basis of these models is a point-to-point communication model characterized by a set of integral parameters, having the same value for each pair of processors. The execution time of other operations (which are, in fact, collective), is expressed as a combination of the point-to-point parameters, and is analytically predicted for different message sizes and numbers of processors involved. The core of this approach is the choice of such a point-to-point model that is the most appropriate to the targeted platform, allowing for easy and natural expression of different algorithms of collective operations. For homogeneous clusters, the point-to-point parameters are found statistically from communication experiments between any two processors. Typical experiments include sending and receiving messages of different sizes, with the communication execution time being measured on one side.

A homogeneous communication model can be applied to a cluster of heterogeneous processors by averaging values obtained for every pair of processors. In this case, the heterogeneous cluster will be treated as homogeneous in terms of the performance of communication operations. If some processors or links in the heterogeneous cluster significantly differ in performance, predictions based on the homogeneous communication model may become inaccurate. More accurate performance models would not average the point-to-point communication parameters. The use of such heterogeneous communication models in model-based optimization of MPI collective operations on heterogeneous clusters do improve their performance.

The traditional models use a small number of parameters to describe communication between any two processors. The number of these parameters and their use in the model are always defined in a way that allows for their accurate estimation with a set of point-to-point communication experiments between these two processors. The price to pay is that such a traditional point-to-point communication model is not intuitive. The meaning of its parameters is not clear. Different sources of the contribution into the execution time are artificially and non-intuitively mixed and spread over a smaller number of parameters. This makes the models difficult to use for accurate modelling of collective communications.

The alternative approach is to use original point-to-point heterogeneous models that allow for easy and intuitive expression of the execution time of collective communication operations such as this model designed for switched heterogeneous clusters. While more accurate, this heterogeneous model has a significantly larger number of parameters. This will result in a higher cost of their estimation. In particular, when applied to the heterogeneous communication model, the statistical methods of finding the point-to-point parameters, traditionally used in the case of homogeneous communication models, will require a significantly larger number of measurements. For our target architecture, which is a heterogeneous cluster based on a switched network, we can address this problem by performing most of the communication experiments in parallel, using the fact that the network switches provide no-contention point-to-point communications, appropriately forwarding packets between sources and destinations.

We present the software tool that automates the estimation of the heterogeneous communication performance models of clusters based on a switched network [8]. The software tool can also be used in the high-level model-based optimization of MPI collective operations. This is particularly important for heterogeneous platforms where the users typically have neither authority nor knowledge for making changes in hardware or basic software settings.

1.1 Authors

Alexey Lastovetsky, Vladimir Rychkov, Maureen O'Flynn, Kiril Dichev

Heterogeneous Computing Laboratory

School of Computer Science and Informatics, University College Dublin

Belfield, Dublin 4, Ireland

http://hcl.ucd.ie

{alexey.lastovetsky, vladimir.rychkov}@ucd.ie

References

[1] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: incorporating long messages into the LogP model – one step closer towards a realistic model for parallel computation. In *SPAA '95: Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 95–105, New York, NY, USA, 1995. ACM. 15

[2] Prashanth B. Bhat, C. S. Raghavendra, and Viktor K. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63(3):251 – 263, 2003. 24, 30

[3] E.W. Chan, M.F. Heimlich, A. Purkayastha, and R.A. van de Geijn. On optimizing collective communication. *Cluster Computing, 2004 IEEE International Conference on*, pages 145–155, Sept. 2004. 24

[4] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12, 1993. 14

[5] J. Hatta and S. Shibusawa. Scheduling algorithms for efficient gather operations in distributed heterogeneous systems. *Parallel Processing, 2000. Proceedings. 2000 International Workshops on*, pages 173–180, 2000. 24, 30

[6] Roger W. Hockney. The communication challenge for MPP: Intel Paragon and Meiko CS-2. *Parallel Comput.*, 20(3):389–398, 1994. 10, 12

[7] Thilo Kielmann, Henri E. Bal, and Kees Verstoep. Fast measurement of LogP parameters for message passing platforms. In *IPDPS '00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 1176–1183, London, UK, 2000. Springer-Verlag. 10, 14, 15

[8] A. Lastovetsky, V. Rychkov, and M. OFlynn. A software tool for accurate estimation of parameters of heterogeneous communication models. In A. Lastovetsky, T. Kechadi, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 15th European PVM/MPI User's Group Meeting*, volume 5205, pages 43–54, Dublin, Ireland, September 7-10 2008. Springer-Verlag Berlin Heidelberg. 2, 10

[9] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005. 24

2 Installation

```
Installation
============

Included software (configured, built and installed together with CPM):
1. MPIBlib (MPI Benchmark library) - required (for benchmarking MPI communication
      operations)
2. logp_mpi (The MPI LogP Benchmark, version 1.4) - required (for the PLogP model
     )

Required software:
1. any C/C++ and MPI (MPICH-1 does not support shared libraries)
2. GSL (GNU Scientific Library, version 1.11)
3. Boost (The Boost C++ libraries: Graph, version 1.36) -
   optional (for tree-based collectives)
4. R (The R Project for Statistical Computing, version 2.6.1) - optional
   (for estimation of the LMO threshold parameters)
5. Gnuplot (An Interactive Plotting Program) -
   optional (for performance diagrams)
6. Graphviz (Graph Visualization Software: dot) -
   optional (for tree visualization)
7. Doxygen (Source code documentation generator tool) and any TeX -
   optional (for reference manual)

GSL
If GSL is installed in a non-default directory
```

```
$ export LD_LIBRARY_PATH=DIR/lib:$LD_LIBRARY_PATH

Boost
1. Boost should be configured with at least the Graph library
   (default: all)
$ ./configure --prefix=DIR --with-libraries=graph
2. Default installation:
 - DIR/include/boost_version/boost
 - DIR/lib/libboost_library_versions.*
Create symbolic links:
$ cd DIR/include; ln -s boost_version/boost
$ cd DIR/lib; ln -s libboost_[library]_[version].[a/so] libboost_[library].[a/so]

$ export LD_LIBRARY_PATH=DIR/lib:$LD_LIBRARY_PATH

R
1. R should be configured as a shared library
$ ./configure --prefix=DIR --enable-R-shlib=yes
$ make install
2. Set up environment
$ export R_HOME=DIR/lib/R
3. Install required packages
$ DIR/bin/R
> install.packages(c("sandwich", "strucchange", "zoo"))
4. If R is installed in a non-default directory
$ export LD_LIBRARY_PATH=$R_HOME/lib:$LD_LIBRARY_PATH

For developers
--------------

Required software:
1. Subversion
2. GNU autotools
3. Doxygen
4. Graphviz

$ svn co https://hcl.ucd.ie/repos/CPM/trunk CPM
$ cd CPM
$ svn log -v > ChangeLog
$ cd MPIBlib
$ svn log -v > ChangeLog
$ cd ..
$ autoreconf --install --force
$ mkdir build
$ cd build
$ ../configure --enable-debug
$ make all install check

To create a package:
$ make dist

For users
---------

Download and untar the latest package from http://hcl.ucd.ie/project/cpm

$ mkdir build
$ cd build
$ ../configure
$ make all install check

Configuration
-------------

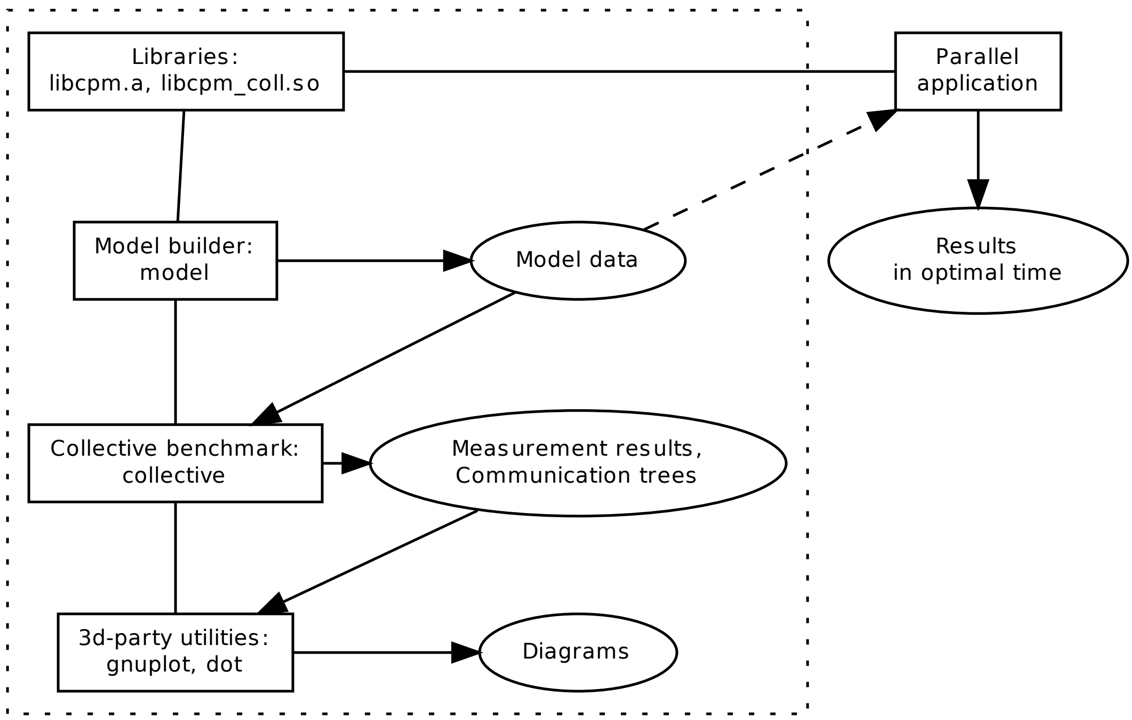
Packages:
  --with-gsl-dir=DIR      GNU Scientific Library directory
  --with-boost-dir=DIR    The Boost C++ libraries directory
```

```
  --with-r-dir=DIR        The R Project for Statistical Computing directory

Check configure options:
$ ../configure -h
```

3 The software design

CPM is implemented in C/C++ on top of MPI. The package consists of libraries, tools and tests. The libraries implements heterogeneous communication performance models and model-based collectives. The tools estimates the parameters of the models and evaluates the performance of the model-based collective communication operations.

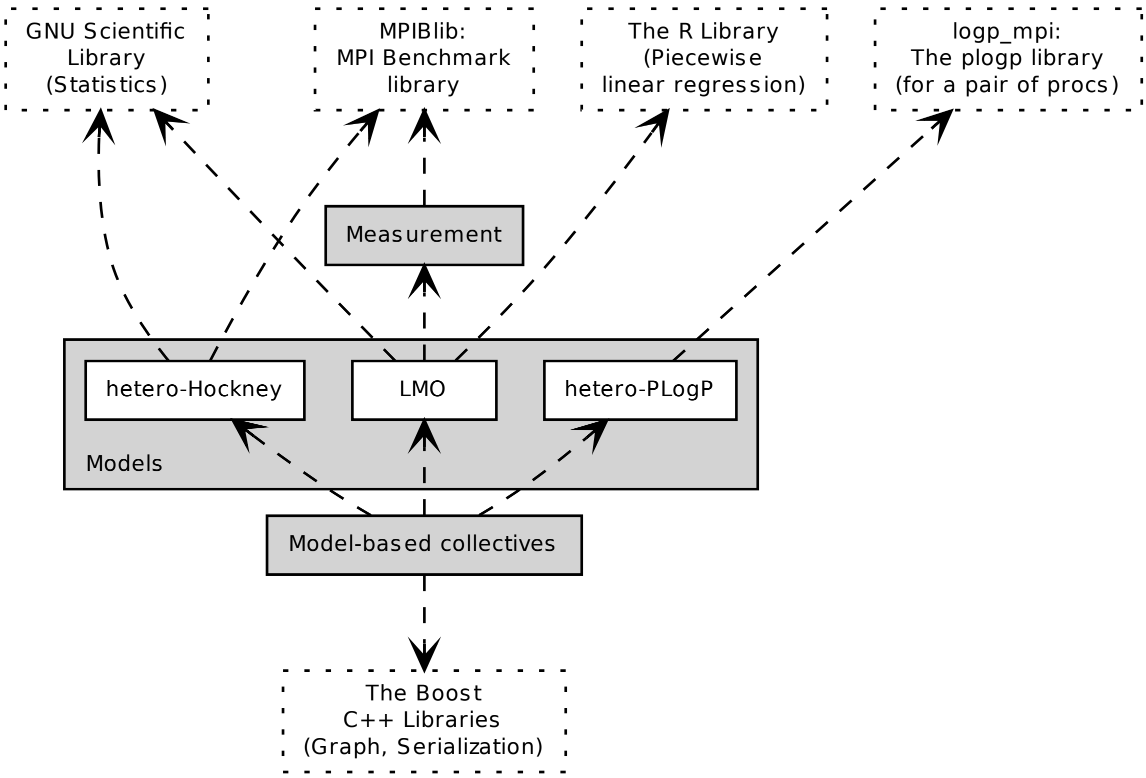


3.1 Models library

A static library `libcpm.a` implements heterogeneous communication performance models and consists of the following modules:

- Measurement: benchmarking specific communication experiments
- Communication performance models

- Prediction of communication time



3.1.1 Tools

- tools/model.c - model builder, performs measurements, estimates parameters of the model and stores the model data.

Usage:

```
$ mpirun [mpi options] model -M LMO -o lmo.mod > lmo.out
$ mpirun [mpi options] collective -O CPM_Gather_linear_opt -i lmo.mod > collecti
    ve.out
$ gnuplot collective.plot
```

3.2 Collectives library

Shared library `libcpm_coll.so` implements different model-based algorithms of collectives:

- Generic model-based collectives
- Hockney-based collective operations

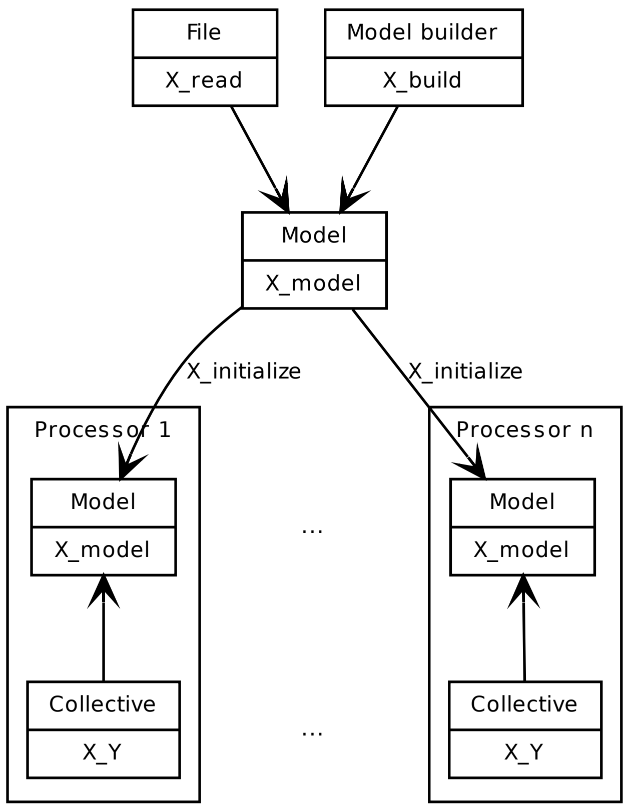
- PLogP-based collective operations
- LMO-based collective operations

Command-line arguments:

- **verbose** verbose mode (default: no)
- **sgv** scatterv/gatherv mode
  - 0 propagation (default)
  - 1 broadcast
  - 2 allover
- **model** *S* communication model: Hockney, PLogP, LMO (required for model-based collectives)
- **file** *S* model data file (required for model-based collectives)

In order to preserve the original MPI interface, all model-based implementations use the global variables that provide model parameters. The interface of the implementation of a collective communication operation `Y` based on the model `X` includes the following components:

- `void X_Y(standard args)` is the model-based collective operation itself (for example, Hockney_Scatter_bfs_binomial_min)
- `X_model* X_model_instance` is a global variable providing the model parameters (must be available at all processors in the MPI communicator)
- `void X_initialize(MPI_Comm, X_model* model), void X_finalize(MPI_Comm)` are functions responsible for allocation and deallocation of the model instance at all processors. The `model` argument encapsulates the model parameters obtained either from a file or the model builder.



All model-based algorithms are divided into two groups: model-specific and generic.

Model-specific collectives depend on certain communication performance models, using parameters specific for these models only. For example, LMO_Gather_split_flat directly uses the LMO_model_instance global variable and its threshold parameters to split the medium size messages and perform a series of linear gathers with small messages, in order to avoid escalations of the execution time on the clusters with the TCP/IP communication layer. A model-specific collective operation `Y` is implemented in the following way:

```
int X_Y(standard args) {
    if (condition with X_model_instance->param)
        return ...;
}
```

Generic collectives depend on the prediction of the execution time of some communication operation (communication primitive); they are parameterized by predictions, which can be provided by any model. The communication primitive can be either the collective operation itself or some other simple operation. See Generic model-based collectives.

3.2.1 Tools

- MPIBlib/tools/collective_test - performs a universal or operation-specific collective benchmark with given accuracy and efficiency

- MPIBlib/tools/collective - verifies implementations of collective communication operations

3.2.2 Tests

- tests/pgemm.c - heterogeneous parallel matrix-matrix product, using different algorithms of scatter-v/gatherv.

4 Module Documentation

4.1 Measurement: benchmarking specific communication experiments

Functions

- void CPM_measure_p2pp (MPI_Comm comm, int M, int parallel, MPIB_precision precision, MPIB_result *results)
- void CPM_measure_p2pp_p2p (MPI_Comm comm, int M, int parallel, MPIB_precision precision, MPIB_result *results_p2pp, MPIB_result *results_p2p[2])

4.1.1 Detailed Description

This module extends the MPIBlib functionality in order to measure the execution time of some specific communication experiments required to estimate the parameters of communication performance models.

4.1.2 Function Documentation

4.1.2.1 void CPM_measure_p2pp (MPI_Comm *comm,* int *M,* int *parallel,* MPIB_precision *precision,* MPIB_result * *results*)

Measures the 1-to-2 execution times.

Measures the execution times of

- $i \xleftrightarrow[0]{M} jk$,
- $j \xleftrightarrow[0]{M} ik$,
- $k \xleftrightarrow[0]{M} ij$

in the communicator, $i < j < k$.

Parameters

comm communicator, number of nodes ≥ 3

M message size

parallel several non-overlapped p2pp communications at the same time if non-zero

precision measurement parameters

results array of $3C_n^3$ measurement results (significant only at root)

4.1.2.2 void CPM_measure_p2pp_p2p (MPI_Comm *comm*, int *M*, int *parallel*, MPIB_precision *precision*, MPIB_result $*$ *results_p2pp*, MPIB_result $*$ *results_p2p[2]*)

Measures the 1-to-2 and 1-to-1 execution times.

Measures the execution times of

- $i \xleftrightarrow[0]{M} jk$, $i \xleftrightarrow[0]{M} j$, $i \xleftrightarrow[0]{M} k$,
- $j \xleftrightarrow[0]{M} ik$, $j \xleftrightarrow[0]{M} i$, $j \xleftrightarrow[0]{M} k$,
- $k \xleftrightarrow[0]{M} ij$, $k \xleftrightarrow[0]{M} i$, $k \xleftrightarrow[0]{M} j$

in the communicator, $i < j < k$.

Parameters

comm communicator, number of nodes ≥ 3

M message size

parallel several non-overlapped point-to-point communications at the same time if non-zero

precision measurement parameters

results_p2pp array of $3C_n^3$ measurement results (significant only at root)

results_p2p 2 arrays of $3C_n^3$ measurement results (significant only at root)

4.2 Communication performance models

Classes

- struct CPM_predictor

Defines

- #define CPM_ERR_MODEL MPI_ERR_LASTCODE + 100

4.2.1 Detailed Description

This module provide the following models:

- The heterogeneous Hockney model [6]
- The heterogeneous PLogP model [7]
- LMO: an advanced heterogeneous communication performance model [8]

For each model, this module provides the following interface (`X` stands for the name of the model: `Hockney`, `PLogP` or `LMO`):

- ```
  struct X_model {
      CPM_predictor predictor;
      ... // parameters
  }
  ```
  is a data structure containing the parameters of the model and a set of the estimation functions defined by CPM_predictor.

- ```
  void X_build(MPI_Comm, MPIB_msgset, MPIB_precision, int parallel, X_model**);
  ```
  is a function responsible for building the model with given precision and message sizes. If the `parallel` argument is set to zero, the communication experiments will be performed consequently, otherwise in parallel.

- ```
  void X_read(FILE*, X_model**);
  void X_write(FILE*, const X_model*);
  ```
  are functions for input/output of the model data.

This interface can be used directly in parallel applications. It is also a basis for optimized implementations of MPI collective communication operations (see Collectives library).

4.2.2 Define Documentation

4.2.2.1 #define CPM_ERR_MODEL MPI_ERR_LASTCODE + 100

Invalid model instance. A return error code to be used in the model-based implementations of collective communication operations.

4.3 The heterogeneous Hockney model

Classes

- struct Hockney_model

Functions

- Hockney_model ∗ Hockney_alloc (int n)
- void Hockney_free (Hockney_model ∗model)
- void Hockney_read (FILE ∗stream, Hockney_model ∗∗model)
- void Hockney_write (FILE ∗stream, const Hockney_model ∗model)
- void Hockney_estimate (MPI_Comm comm, int M, MPIB_precision precision, int parallel, Hockney_model ∗∗model)
- void Hockney_estimate_regression (MPI_Comm comm, MPIB_msgset msgset, MPIB_precision precision, int parallel, Hockney_model ∗∗model)
- double Hockney_predict_p2p (void ∗_this, int i, int j, int M)
- double Hockney_predict_sg_flat_serial (void ∗_this, int root, int M)
- double Hockney_predict_sg_flat_parallel (void ∗_this, int root, int M)
- double Hockney_predict_sg_binomial (void ∗_this, int root, int M)
- double Hockney_hpredict_sg_flat_serial (void ∗_this, int M)
- double Hockney_hpredict_sg_flat_parallel (void ∗_this, int M)
- double Hockney_hpredict_sg_binomial (void ∗_this, int M)
- double Hockney_predict_linear_sgv (void ∗_this, int size, int root, int ∗size_bytes)
  ```

4.3.1 Detailed Description

This module provides building of the heterogeneous extension of the Hockney model and estimation of the execution time of point-to-point and collective communication operations.

In contrast to the original model [6], which is based on two point-to-point parameters, estimating the point-to-point execution time as $T(M) = \alpha + \beta M$, the heterogeneous model distinguishes the parameters of each pair of processors $T_{ij}(M) = \alpha_{ij} + \beta_{ij}M$.

4.3.2 Function Documentation

4.3.2.1 Hockney_model∗ Hockney_alloc (int *n*)

Allocates memory for the Hockney model.

Parameters

n number of processors

4.3.2.2 void Hockney_free (Hockney_model ∗ *model*)

Frees the Hockney model.

Parameters

model the Hockney model

4.3.2.3 void Hockney_read (FILE ∗ *stream,* Hockney_model ∗∗ *model*)

Reads the Hockney model.

4.3.2.4 void Hockney_write (FILE ∗ *stream,* const Hockney_model ∗ *model*)

Writes the Hockney model.

4.3.2.5 void Hockney_estimate (MPI_Comm *comm,* int *M,* MPIB_precision *precision,* int *parallel,* Hockney_model ∗∗ *model*)

Estimates the parameters of the Hockney model in two series of roundtrips with empty and non-empty messages. (accuracy depends on the precision of measurements):

- Measures the execution time $T_{ij}(0)$ of each $i \overset{0}{\underset{0}{\longleftrightarrow}} j$ roundtrip in the communicator, $i < j$, to find $\alpha_{ij} = T_{ij}(0)$. To obtain more accurate results performs a series of roundtrips and takes the average $T_{ij}(0)$.
- Measures the execution time $T_{ij}(M)$ of each $i \overset{M}{\underset{M}{\longleftrightarrow}} j$ roundtrip in the communicator, $i < j$, to find $\beta_{ij} = \frac{T_{ij}(M) - \alpha_{ij}}{M}$. To obtain more accurate results performs a series of roundtrips and takes the average $T_{ij}(M)$.

Parameters

comm communicator, number of nodes ≥ 2

M message size

precision measurement precision

parallel several non-overlapped point-to-point communications at the same time if non-zero

model Hockney model (significant only at root)

4.3.2.6 void Hockney_estimate_regression (MPI_Comm *comm*, MPIB_msgset *msgset*, MPIB_precision *precision*, int *parallel*, Hockney_model $**$ *model*)

Estimates the parameters of the Hockney model in a series of roundtrips with different message sizes (accuracy depends on the message set):

- Performs single communication experiments for different message sizes.
- Selects message sizes regularly TODO: Adaptive selection, comparing the result of measurement with the prediction based on the linear regression over all previous results.

Parameters

comm communicator, number of nodes ≥ 2

msgset message set

precision measurement precision

parallel several non-overlapped point-to-point communications at the same time if non-zero

model Hockney model (significant only at root)

4.3.2.7 double Hockney_predict_p2p (void $*$ *_this*, int *i*, int *j*, int *M*)

Predicts the execution time of a point-to-point communication as $T_{ij}(M) = \alpha_{ij} + \beta_{ij}M$.

4.3.2.8 double Hockney_predict_sg_flat_serial (void $*$ *_this*, int *root*, int *M*)

Heterogeneous prediction of flat-tree scatter/gather (sequential point-to-point communications) $\sum_{i=0,i\neq r}^{n-1} (\alpha_{ri} + \beta_{ri}M)$

4.3.2.9 double Hockney_predict_sg_flat_parallel (void $*$ *_this*, int *root*, int *M*)

Heterogeneous prediction of flat-tree scatter/gather (parallel point-to-point communications) $\max_{i=0,i\neq r}^{n-1} (\alpha_{ri} + \beta_{ri}M)$

4.3.2.10 double Hockney_predict_sg_binomial (void $*$ *_this*, int *root*, int *M*)

Heterogeneous prediction of binomial scatter/gather (point-to-point communications are sequential within the same root or parallel otherwise) $\overbrace{\alpha_{ri} + \beta_{ri} 2^{log_2(n-1)} M + \max S(log_2(n-1) - 1)}$

4.3.2.11 double Hockney_hpredict_sg_flat_serial (void ∗ *_this,* int *M*)

Homogeneous prediction of flat-tree scatter/gather (sequential point-to-point communications) $(n-1)(\alpha+\beta M)$

4.3.2.12 double Hockney_hpredict_sg_flat_parallel (void ∗ *_this,* int *M*)

Homogeneous prediction of flat-tree scatter/gather (parallel point-to-point communications) $\alpha+\beta M$

4.3.2.13 double Hockney_hpredict_sg_binomial (void ∗ *_this,* int *M*)

Homogeneous prediction of binomial scatter/gather (point-to-point communications are sequential within the same root or parallel otherwise) $(log_2 n)\alpha+(n-1)\beta M$

4.3.2.14 double Hockney_predict_linear_sgv (void ∗ *_this,* int *size,* int *root,* int ∗ *size_bytes*)

Hockney predictions of collectives based on P2P predictions

4.4 The heterogeneous PLogP model

Classes

- struct PLogP_model

Functions

- PLogP_model ∗ PLogP_alloc (int n)
- void PLogP_free (PLogP_model ∗model)
- void PLogP_read (FILE ∗stream, PLogP_model ∗∗model)
- void PLogP_write (FILE ∗stream, const PLogP_model ∗model)
- void PLogP_estimate (MPI_Comm comm, MPIB_precision precision, MPIB_msgset msgset, int parallel, PLogP_model ∗∗model)
- double PLogP_predict_p2p (void ∗_this, int i, int j, int M)
- double LogGP_predict_p2p (void ∗_this, int i, int j, int M)
- double PLogP_hpredict_sg_linear (void ∗_this, int M)
- double LogGP_hpredict_sg_linear (void ∗_this, int M)

4.4.1 Detailed Description

This module provides building of the heterogeneous extension of the parameterised LogP model and PLogP/LogGP predictions of the execution time of point-to-point and collective communication operations.

The PLogP model [7] is an extension of the LogP model [4]. The PLogP model is defined in terms of the following parameters:

- L - latency,
- $o_s(M)$ and $o_r(M)$ - sender and receiver overheads,
- $g(M)$ - gap per message (delay between consecutive communications),

- P - number of processors.

Some of these parameters are functions, which makes this model "parameterised". They are represented by piecewise linear functions. The PLogP parameters are estimated with help of the logp_mpi library [7]. The heterogeneous extension is a set of the PLogP models built for each pair of processors: $\{L_{ij}, o_{sij}(M), o_{rij}(M), g_{ij}(M)\}_{i \neq j=0}^{n-1}$.

The parameters of the LogP and LogGP (another extension of the LogP model proposed in [1], which considers message size and includes an extra parameter, G, gap per byte) models can be expressed via the PLogP parameters:

- $L = L^p + g^p(1) - o_s^p(1) - o_r^p(1)$
- $2 * o = o_s^p(1) + o_r^p(1)$
- $g = g^p(1)$
- $G = g^p(M_{max})/M_{max}$

The heterogeneous extension of the LogGP model can be obtained by applying the above equations to the PLogP parameters of each pair of processors. Therefore, this module provides both the PLogP and LogGP predictions of the execution time of collective communication operations.

4.4.2 Function Documentation

4.4.2.1 PLogP_model* PLogP_alloc (int *n*)

Allocates memory for PLogP model

4.4.2.2 void PLogP_free (PLogP_model * *model*)

Frees the PLogP model.

4.4.2.3 void PLogP_read (FILE * *stream*, PLogP_model ** *model*)

Reads the PLogP model.

Note

Creates a temporary file.

4.4.2.4 void PLogP_write (FILE * *stream*, const PLogP_model * *model*)

Writes the PLogP model.

Note

Creates a temporary file.

4.4.2.5 void PLogP_estimate (MPI_Comm *comm*, MPIB_precision *precision*, MPIB_msgset *msgset*, int *parallel*, PLogP_model ** *model*)

Estimates the PLogP model. Calls the logp_mpi library.

Parameters

- ***comm*** communicator
- ***precision*** measurement precision
- ***msgset*** message set
- ***parallel*** several non-overlapped point-to-point communications at the same time if non-zero
- ***model*** PLogP model

4.4.2.6 double PLogP_predict_p2p (void * *_this*, int *i*, int *j*, int *M*)

Predicts the execution time of a point-to-point communication as $L + g(M)$

4.4.2.7 double LogGP_predict_p2p (void * *_this*, int *i*, int *j*, int *M*)

Predicts the execution time of a point-to-point communication according to LogGP model as $L + 2 * o + G(M - 1)$

Parameters

- ***_this*** PLogP model
- ***i*** index of the process
- ***j*** index of the process
- ***M*** message size

4.4.2.8 double PLogP_hpredict_sg_linear (void * *_this*, int *M*)

Homogeneous PLogP prediction of linear scatter/gather: $L + (n - 1)g(M)$

4.4.2.9 double LogGP_hpredict_sg_linear (void * *_this*, int *M*)

Homogeneous LogGP prediction of linear scatter/gather: $L + 2o + (n - 1)(M - 1)G + (n - 2)g$

4.5 LMO: an advanced heterogeneous communication performance model

Classes

- struct LMO_model

Functions

- LMO_model ∗ LMO_alloc (int n)
- void LMO_free (LMO_model ∗model)
- void LMO_read (FILE ∗stream, LMO_model ∗∗model)
- void LMO_write (FILE ∗stream, const LMO_model ∗model)
- void LMO_estimate_p2p (LMO_model ∗model, MPI_Comm comm, int parallel, MPIB_precision precision)
- void LMO_estimate_one2many (LMO_model ∗model, MPI_Comm comm, int stride, int max_size, MPIB_precision precision)
- void LMO_estimate_many2one (LMO_model ∗model, MPI_Comm comm, int stride, int max_size, MPIB_precision precision)
- void LMO_estimate (MPI_Comm comm, MPIB_precision precision, MPIB_msgset msgset, int parallel, LMO_model ∗∗model)
- double LMO_predict_p2p (void ∗_this, int i, int j, int M)
- double LMO_predict_scatter_flat (void ∗_this, int root, int M)
- double LMO_predict_gather_flat (void ∗_this, int root, int M)

4.5.1 Detailed Description

This module provides building of the LMO model and estimation of the execution time of point-to-point and collective communication operations.

4.5.2 Function Documentation

4.5.2.1 LMO_model∗ LMO_alloc (int *n*)

Allocates the LMO model.

Parameters

n number of processors

Returns

LMO model

4.5.2.2 void LMO_free (LMO_model ∗ *model*)

Frees the LMO model.

Parameters

model LMO model

4.5.2.3 void LMO_read (FILE ∗ *stream,* LMO_model ∗∗ *model*)

Reads the LMO model.

4.5.2.4 void LMO_write (FILE ∗ *stream,* const LMO_model ∗ *model*)

Writes the LMO model.

4.5.2.5 void LMO_estimate_p2p (LMO_model ∗ *model,* MPI_Comm *comm,* int *parallel,* MPIB_precision *precision*)

Estimates the point-to-point parameters.

- Finds the fixed processing delays and latencies.

  For each 3 nodes $i < j < k$, measures execution times and solves systems of equations:

  $$\begin{cases} T_{ij}(0) = 2(C_i + L_{ij} + C_j) & i \xleftrightarrow[0]{0} j \\ T_{jk}(0) = 2(C_j + L_{jk} + C_k) & j \xleftrightarrow[0]{0} k \\ T_{ik}(0) = 2(C_i + L_{ik} + C_k) & i \xleftrightarrow[0]{0} k \\ T_{ijk}(0) = 2(2C_i + \max\limits_{x=j,k}(L_{ix} + C_x)) & i \xleftrightarrow[0]{0} jk \\ T_{jik}(0) = 2(2C_j + \max\limits_{x=i,k}(L_{jx} + C_x)) & j \xleftrightarrow[0]{0} ik \\ T_{kij}(0) = 2(2C_k + \max\limits_{x=i,j}(L_{kx} + C_x)) & k \xleftrightarrow[0]{0} ij \end{cases}$$

  averages solutions:

  $T_{ijk}(0) = 2(2C_i + \max\limits_{x=j,k}(L_{ix} + C_x)) = 2C_i + \max\limits_{x=j,k} T_{ix}(0)$

  $$\begin{cases} C_i = (T_{ijk}(0) - \max\limits_{x=j,k} T_{ix}(0))/2 \\ C_j = (T_{jik}(0) - \max\limits_{x=i,k} T_{jx}(0))/2 \\ C_j = (T_{kij}(0) - \max\limits_{x=i,j} T_{kx}(0))/2 \\ L_{ij} = T_{ij}(0)/2 - C_i - C_j \\ L_{jk} = T_{jk}(0)/2 - C_j - C_k \\ L_{ik} = T_{ik}(0)/2 - C_i - C_k \end{cases}$$

  and checks confidence intervals.

- Finds the variable processing delays and transmission rates.

  For each 3 nodes $i < j < k$, solves systems of equations:

  $$\begin{cases} T_{ij}(M) = 2(C_i + L_{ij} + C_j + M(t_i + \frac{1}{\beta_{ij}} + t_j)) & i \xleftrightarrow[M]{M} j \\ T_{jk}(M) = 2(C_j + L_{jk} + C_k + M(t_j + \frac{1}{\beta_{jk}} + t_k)) & j \xleftrightarrow[M]{M} k \\ T_{ik}(M) = 2(C_i + L_{ik} + C_k + M(t_i + \frac{1}{\beta_{ik}} + t_k)) & i \xleftrightarrow[M]{M} k \\ T_{ijk}(M) = 2(2C_i + Mt_i) + \max\limits_{x=j,k}(2(L_{ix} + C_x) + M(\frac{1}{\beta_{ix}} + t_x)) & i \xleftrightarrow[0]{M} jk \\ T_{jik}(M) = 2(2C_j + Mt_j) + \max\limits_{x=i,k}(2(L_{jx} + C_x) + M(\frac{1}{\beta_{jx}} + t_x)) & j \xleftrightarrow[0]{M} ik \\ T_{kij}(M) = 2(2C_k + Mt_k) + \max\limits_{x=i,j}(2(L_{kx} + C_x) + M(\frac{1}{\beta_{kx}} + t_x)) & k \xleftrightarrow[0]{M} ij \end{cases}$$

  averages solutions:

  $T_{ijk}(M) = 2(2C_i + Mt_i) + \max\limits_{x=j,k}(2(L_{ix} + C_x) + M(\frac{1}{\beta_{ix}} + t_x)) = 2C_i + Mt_i + \max\limits_{x=j,k}(T_{ix}(0) + T_{ix}(M))/2$

$$\begin{cases} t_i = (T_{ijk}(M) - \max\limits_{x=j,k}(T_{ix}(0) + T_{ix}(M))/2 - 2C_i)/M \\ t_j = (T_{jik}(M) - \max\limits_{x=i,k}(T_{jx}(0) + T_{jx}(M))/2 - 2C_j)/M \\ t_j = (T_{kij}(M) - \max\limits_{x=i,j}(T_{kx}(0) + T_{kx}(M))/2 - 2C_k)/M \\ \frac{1}{\beta_{ij}} = (T_{ij}(M)/2 - C_i - L_{ij} - C_j)/M - t_i - t_j \\ \frac{1}{\beta_{jk}} = (T_{jk}(M)/2 - C_j - L_{jk} - C_k)/M - t_j - t_k \\ \frac{1}{\beta_{ik}} = (T_{ik}(M)/2 - C_i - L_{ik} - C_k)/M - t_i - t_k \end{cases}$$

and checks confidence intervals.

It is called by LMO_estimate.

Parameters

model LMO model, must be allocated and filled by LMO_predict_scatter_flat and LMO_predict_-gather_flat (significant only at root)

comm communicator, number of nodes ≥ 3

parallel several non-overlapped point-to-point communications at the same time if non-zero

precision measurement parameters

4.5.2.6 void LMO_estimate_one2many (LMO_model * *model*, MPI_Comm *comm*, int *stride*, int *max_size*, MPIB_precision *precision*)

Estimates the parameters of one-to-many. Measures one-to-many execution time for the message sizes $0 < M < max\_size$ and performs the Bai & Perron algorithm over the F statistic for the data to find the S breakpoint in the piecewise linear regression $T \sim M$. R must be initialized by LMO_init_R at root beforehand.

It is called by LMO_estimate.

Parameters

model LMO model, must be allocated (significant only at root)

comm communicator

stride stride for message sizes

max_size maximum message size

precision measurement parameters

4.5.2.7 void LMO_estimate_many2one (LMO_model * *model*, MPI_Comm *comm*, int *stride*, int *max_size*, MPIB_precision *precision*)

Estimates the parameters of many-to-one. Measures many-to-one execution time for the message sizes $0 < M < max\_size$ and performs the Bai & Perron algorithm over the F statistic for the data to find the M_2 breakpoint in the piecewise linear regression $T \sim 1$. R must be initialized by LMO_init_R at root beforehand.

Then in the loop measures many-to-one execution time for the message sizes $0 < M < m$ with the stride reduced twice each time. m is a message size for which a tenfold escalation of the execution time has been observed on the previous step. As stride reaches 1-byte value m is truncated to a kb value.

It is called by LMO_estimate.

Parameters

model LMO model, must be allocated (significant only at root)

comm communicator

stride stride for message sizes

max_size maximum message size

precision measurement parameters

4.5.2.8 void LMO_estimate (MPI_Comm *comm*, MPIB_precision *precision*, MPIB_msgset *msgset*, int *parallel*, LMO_model $**$ *model*)

Estimates the parameters of the LMO model. Calls LMO_predict_scatter_flat, LMO_predict_gather_flat, LMO_estimate_p2p. R must be initialized by LMO_init_R at root beforehand.

Parameters

comm communicator

precision measurement precision

msgset message set

parallel several non-overlapped point-to-point communications at the same time if non-zero

model LMO model (significant only at root)

4.5.2.9 double LMO_predict_p2p (void $*$ *_this*, int *i*, int *j*, int *M*)

Predicts the execution time of point-to-point communication. The execution time of $i \xrightarrow{M} j$ is equal to $C_i + L_{ij} + C_j + M(t_i + \frac{1}{\beta_{ij}} + t_j)$

Parameters

_this LMO model

i index of the precessor

j index of the precessor

M message size

Returns

predicted execution time

4.5.2.10 double LMO_predict_scatter_flat (void $*$ *_this*, int *root*, int *M*)

Predicts the execution time of flat-tree scatter. $(n-1)(C_r + Mt_r) + \max_{i=1,i\neq r}^{n-1}(L_{ri} + C_i + M(\frac{1}{\beta_{ri}} + t_i))$

Parameters

_this LMO model

root root precessor

M message size

Returns

predicted execution time

4.5.2.11 double LMO_predict_gather_flat (void ∗ *_this*, int *root*, int *M*)

Predicts the execution time of flat-tree gather.

$$(n-1)(C_r + Mt_r) + \begin{cases} \max\limits_{i=1}^{n-1}(L_{ri} + C_i + M(\dfrac{1}{\beta_{ri}} + t_i)) & M < M_1 \\ \sum\limits_{i=1}^{n-1}(L_{ri} + C_i + M(\dfrac{1}{\beta_{ri}} + t_i)) & M > M_2 \end{cases}$$

Parameters

_this LMO model
root root precessor
M message size

Returns

predicted execution time

4.6 Prediction of communication time

Functions

- double CPM_predict_brsg (CPM_predictor ∗predictor, int size, int root, int size_bytes, CPM_coll_-ops operation)
- double CPM_predict_sgv (CPM_predictor ∗predictor, int size, int root, int ∗size_bytes, CPM_coll_-ops operation)
- double CPM_predict_flat_sgv (CPM_predictor ∗predictor, int size, int root, int ∗size_bytes)
- double CPM_predict_flat_sg (CPM_predictor ∗predictor, int size, int root, int size_bytes)
- double CPM_predict_flat_sgv_parallel (CPM_predictor ∗predictor, int size, int root, int ∗size_bytes)
- double CPM_predict_flat_sg_parallel (CPM_predictor ∗predictor, int size, int root, int size_bytes)
- double CPM_predict_flat_sgv_serial (CPM_predictor ∗predictor, int size, int root, int ∗size_bytes)
- double CPM_predict_flat_sg_serial (CPM_predictor ∗predictor, int size, int root, int size_bytes)

4.6.1 Detailed Description

This module provides generic predict functions TODO: implement generic predict functions (start with generic tree predictions) TODO: use generic predict functions with all models and compare their results with observations TODO: implement some model-specific predict functions and compare their results with the results of generic predict functions

4.6.2 Function Documentation

4.6.2.1 double CPM_predict_brsg (CPM_predictor ∗ *predictor*, int *size*, int *root*, int *size_bytes*, CPM_coll_ops *operation*)

Predicts the execution time of Bcast, Reduce, Scatter, Gather operations

4.6.2.2 double CPM_predict_sgv (CPM_predictor ∗ *predictor,* int *size,* int *root,* int ∗ *size_bytes,* CPM_coll_ops *operation*)

Predicts the execution time of Scatterv, Gatherv operations

4.6.2.3 double CPM_predict_flat_sgv (CPM_predictor ∗ *predictor,* int *size,* int *root,* int ∗ *size_bytes*)

Predicts the execution time of flat-tree Scatterv, Gatherv operations

4.6.2.4 double CPM_predict_flat_sg (CPM_predictor ∗ *predictor,* int *size,* int *root,* int *size_bytes*)

Predicts the execution time of flat-tree Scatter, Gather operations

4.6.2.5 double CPM_predict_flat_sgv_parallel (CPM_predictor ∗ *predictor,* int *size,* int *root,* int ∗ *size_bytes*)

return maximum of all the P2P transfer times

4.6.2.6 double CPM_predict_flat_sg_parallel (CPM_predictor ∗ *predictor,* int *size,* int *root,* int *size_bytes*)

return maximum of all the P2P transfer times

4.6.2.7 double CPM_predict_flat_sgv_serial (CPM_predictor ∗ *predictor,* int *size,* int *root,* int ∗ *size_bytes*)

return sum of all the P2P transfer times

4.6.2.8 double CPM_predict_flat_sg_serial (CPM_predictor ∗ *predictor,* int *size,* int *root,* int *size_bytes*)

return sum of all the P2P transfer times

4.7 Generic model-based collectives

Functions

- int CPM_Scatterv_sorted_flat (CPM_predictor ∗predictor, MPIB_sort_order order, void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Gatherv_sorted_flat (CPM_predictor ∗predictor, MPIB_sort_order order, void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Bcast_dfs_binomial (CPM_predictor ∗predictor, CPM_next_node_strategy next_node, void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

- int CPM_Bcast_bfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int CPM_Reduce_dfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int CPM_Reduce_bfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int CPM_Bcast_ucs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int CPM_Reduce_ucs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int CPM_Scatter_ucs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Gather_ucs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Scatter_bfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Scatter_dfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Gather_bfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Gather_dfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Scatterv_dfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Scatterv_bfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Scatterv_ucs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Gatherv_dfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Gatherv_bfs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Gatherv_ucs_binomial (CPM_predictor *predictor, CPM_next_node_strategy next_node, void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int CPM_Scatterv_Traff (CPM_predictor *predictor, void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

- int CPM_Gatherv_Traff (CPM_predictor ∗predictor, void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

4.7.1 Detailed Description

There are two typical examples of generic implementations: switch between algorithms and processor mapping.

- *Switch between algorithms* is an implementation that uses the prediction of the execution time of different algorithms of the collective operation, finds the fastest algorithm for given message size and number of processors, and switches between them. The prediction can be provided by any model. Different algorithms of the collective operations are communication primitives. [3], [9] .
- *Processor mapping* is an implementation of a tree-based algorithm of the collective operation that maps the processors to the nodes of the communication tree in accordance with the performance of the point-to-point communications. In this case, the point-to-point communication operation is a communication primitive, which execution time can be predicted by any model. [5], [2] .

The interface of a generic collective operation `Y` based on the prediction of the communication primitive `Z` consists of:

- a general, model-independent, implementation of collective operation:

  ```
  int CPM_Y(CPM_predictor* predictor, standard args) {
      if (condition with predictor->predict_Z(predictor, args))
          return ...;
  }
  ```

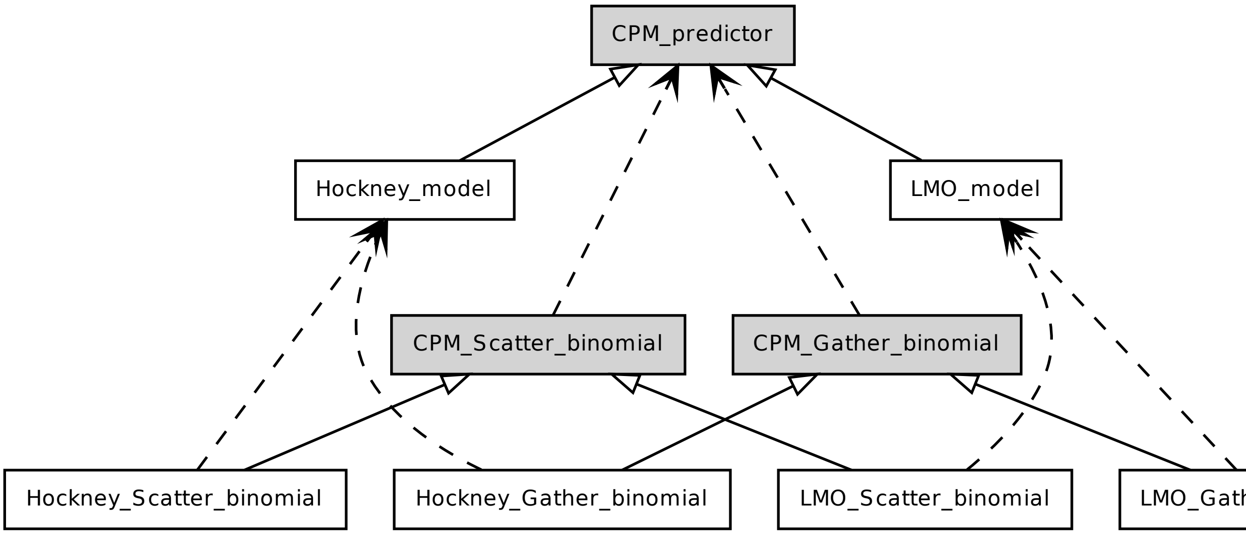
  which includes an extra parameter, a model-based predictor `predictor`, and calls its function `predict_Z` to predict the execution time of the communication primitive. For example, CPM_-Scatter_bfs_binomial is a generic binomial scatter based on the point-to-point predictions CPM_-predictor::predict_p2p.
- particular model-based implementations (`X` stands for the name of the model):

  ```
  int X_Y(standard args) {
      return CPM_Y(&X_model_instance->predictor, standard args);
  }
  ```

  For example, Hockney_Scatter_bfs_binomial_min is a derivative of the generic algorithm CPM_-Scatter_bfs_binomial that is based on the Hockney prediction of the point-to-point execution time.

This approach provides flexibility by reusing the same (general) implementations of a collective operation with different communication performance models.

A design of the generic collectives:



Generic model-based functions are designed for use in both C/C++.

4.7.2 Function Documentation

4.7.2.1 int CPM_Scatterv_sorted_flat (CPM_predictor * *predictor,* MPIB_sort_order *order,* void * *sendbuf,* int * *sendcounts,* int * *displs,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic sorted flat-tree scatterv. Starts from the end/beginning (order = DESC/ASC) of the sorted list.

4.7.2.2 int CPM_Gatherv_sorted_flat (CPM_predictor * *predictor,* MPIB_sort_order *order,* void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int * *recvcounts,* int * *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic sorted flat-tree gatherv. Starts from the beginning/end (order = DESC/ASC) of the sorted list.

4.7.2.3 int CPM_Bcast_dfs_binomial (CPM_predictor * *predictor,* CPM_next_node_strategy *next_node,* void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

Generic DFS binomial bcast.

4.7.2.4 int CPM_Bcast_bfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

Generic BFS binomial bcast.

4.7.2.5 int CPM_Reduce_dfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* void ∗ *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

Generic DFS binomial reduce.

4.7.2.6 int CPM_Reduce_bfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* void ∗ *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

Generic BFS binomial reduce.

4.7.2.7 int CPM_Bcast_ucs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

Generic UCS binomial bcast.

4.7.2.8 int CPM_Reduce_ucs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* void ∗ *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

Generic UCS binomial reduce.

4.7.2.9 int CPM_Scatter_ucs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic UCS binomial scatter.

4.7.2.10 int CPM_Gather_ucs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic UCS binomial gather.

4.7.2.11 int CPM_Scatter_bfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic BFS binomial scatter.

4.7.2.12 int CPM_Scatter_dfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic DFS binomial scatter.

4.7.2.13 int CPM_Gather_bfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic BFS binomial gather.

4.7.2.14 int CPM_Gather_dfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic DFS binomial gather.

4.7.2.15 int CPM_Scatterv_dfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic DFS binomial scatterv.

4.7.2.16 int CPM_Scatterv_bfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic BFS binomial scatterv.

4.7.2.17 int CPM_Scatterv_ucs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic UCS binomial scatterv.

4.7.2.18 int CPM_Gatherv_dfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic DFS binomial gatherv.

4.7.2.19 int CPM_Gatherv_bfs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic BFS binomial gatherv.

4.7.2.20 int CPM_Gatherv_ucs_binomial (CPM_predictor ∗ *predictor,* CPM_next_node_strategy *next_node,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic UCS binomial gatherv.

4.7.2.21 int CPM_Scatterv_Traff (CPM_predictor ∗ *predictor,* void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic Traff scatterv.

4.7.2.22 int CPM_Gatherv_Traff (CPM_predictor ∗ *predictor,* void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Generic Traff gatherv.

4.8 Hockney-based collective operations

Functions

- int Hockney_initialize (MPI_Comm comm, Hockney_model ∗model)
- int Hockney_finalize (MPI_Comm comm)
- int Hockney_Scatterv_sorted_flat_asc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_sorted_flat_asc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_sorted_flat_dsc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_sorted_flat_dsc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Bcast_dfs_binomial_min (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int Hockney_Reduce_dfs_binomial_min (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int Hockney_Bcast_dfs_binomial_max (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int Hockney_Reduce_bfs_binomial_min (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int Hockney_Reduce_bfs_binomial_max (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int Hockney_Bcast_bfs_binomial_min (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int Hockney_Bcast_bfs_binomial_max (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int Hockney_Reduce_dfs_binomial_max (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int Hockney_Bcast_ucs_binomial_min (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

- int Hockney_Reduce_ucs_binomial_min (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int Hockney_Bcast_ucs_binomial_max (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int Hockney_Reduce_ucs_binomial_max (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int Hockney_Scatter_dfs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gather_dfs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatter_dfs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gather_dfs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatter_ucs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gather_ucs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatter_ucs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gather_ucs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatter_bfs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gather_bfs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatter_bfs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gather_bfs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_dfs_binomial_min (void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_dfs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_dfs_binomial_max (void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_dfs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_bfs_binomial_min (void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_bfs_binomial_min (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_bfs_binomial_max (void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_bfs_binomial_max (void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_ucs_binomial_min (void *sendbuf, int *sendcounts, int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

- int Hockney_Gatherv_ucs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_ucs_binomial_max (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_ucs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Scatterv_Traff (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int Hockney_Gatherv_Traff (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

Variables

- Hockney_model ∗ Hockney_model_instance

4.8.1 Detailed Description

The implementations of tree-based algorithms are similar to [5] and [2] .

4.8.2 Function Documentation

4.8.2.1 int Hockney_initialize (MPI_Comm *comm*, Hockney_model ∗ *model*)

Initializes the instances of the Hockney model (Hockney_model_instance) at all processes in the communicator.

Parameters

comm MPI communicator

model Hockney model (significant only at root)

4.8.2.2 int Hockney_finalize (MPI_Comm *comm*)

Destroys the instances of the Hockney model (Hockney_model_instance) at all processes in the communicator.

4.8.2.3 int Hockney_Scatterv_sorted_flat_asc (void ∗ *sendbuf*, int ∗ *sendcounts*, int ∗ *displs*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Sorted flat-tree scatterv based on the Hockney model.

4.8.2.4 int Hockney_Gatherv_sorted_flat_asc (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Sorted flat-tree gatherv based on the Hockney model.

4.8.2.5 int Hockney_Scatterv_sorted_flat_dsc (void * *sendbuf,* int * *sendcounts,* int * *displs,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Sorted flat-tree scatterv based on the Hockney model.

4.8.2.6 int Hockney_Gatherv_sorted_flat_dsc (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int * *recvcounts,* int * *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Sorted flat-tree gatherv based on the Hockney model.

4.8.2.7 int Hockney_Bcast_dfs_binomial_min (void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

DFS binomial bcast based on the Hockney model.

4.8.2.8 int Hockney_Reduce_dfs_binomial_min (void * *sendbuf,* void * *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

DFS binomial reduce based on the Hockney model.

4.8.2.9 int Hockney_Bcast_dfs_binomial_max (void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

DFS binomial bcast based on the Hockney model.

4.8.2.10 int Hockney_Reduce_bfs_binomial_min (void * *sendbuf,* void * *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

BFS binomial reduce based on the Hockney model.

4.8.2.11 int Hockney_Reduce_bfs_binomial_max (void * *sendbuf,* void * *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

BFS binomial reduce based on the Hockney model.

4.8.2.12 int Hockney_Bcast_bfs_binomial_min (void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

BFS binomial bcast based on the Hockney model.

4.8.2.13 int Hockney_Bcast_bfs_binomial_max (void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

BFS binomial bcast based on the Hockney model.

4.8.2.14 int Hockney_Reduce_dfs_binomial_max (void * *sendbuf,* void * *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

DFS binomial reduce based on the Hockney model.

4.8.2.15 int Hockney_Bcast_ucs_binomial_min (void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

UCS binomial bcast based on the Hockney model.

4.8.2.16 int Hockney_Reduce_ucs_binomial_min (void * *sendbuf,* void * *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

UCS binomial reduce based on the Hockney model.

4.8.2.17 int Hockney_Bcast_ucs_binomial_max (void * *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

UCS binomial bcast based on the Hockney model.

4.8.2.18 int Hockney_Reduce_ucs_binomial_max (void * *sendbuf,* void * *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

UCS binomial reduce based on the Hockney model.

4.8.2.19 int Hockney_Scatter_dfs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

DFS binomial scatter based on the Hockney model.

4.8.2.20 int Hockney_Gather_dfs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

DFS binomial gather based on the Hockney model.

4.8.2.21 int Hockney_Scatter_dfs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

DFS binomial scatter based on the Hockney model.

4.8.2.22 int Hockney_Gather_dfs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

DFS binomial gather based on the Hockney model.

4.8.2.23 int Hockney_Scatter_ucs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial scatter based on the Hockney model.

4.8.2.24 int Hockney_Gather_ucs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial gather based on the Hockney model.

4.8.2.25 int Hockney_Scatter_ucs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial scatter based on the Hockney model.

4.8.2.26 int Hockney_Gather_ucs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial gather based on the Hockney model.

4.8.2.27 int Hockney_Scatter_bfs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

BFS binomial scatter based on the Hockney model.

4.8.2.28 int Hockney_Gather_bfs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

BFS binomial gather based on the Hockney model.

4.8.2.29 int Hockney_Scatter_bfs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

BFS binomial scatter based on the Hockney model.

4.8.2.30 int Hockney_Gather_bfs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

BFS binomial gather based on the Hockney model.

4.8.2.31 int Hockney_Scatterv_dfs_binomial_min (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial scatterv based on the Hockney model.

4.8.2.32 int Hockney_Gatherv_dfs_binomial_min (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial gatherv based on the Hockney model.

4.8.2.33 int Hockney_Scatterv_dfs_binomial_max (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial scatterv based on the Hockney model.

4.8.2.34 int Hockney_Gatherv_dfs_binomial_max (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial gatherv based on the Hockney model.

4.8.2.35 int Hockney_Scatterv_bfs_binomial_min (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial scatterv based on the Hockney model.

4.8.2.36 int Hockney_Gatherv_bfs_binomial_min (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial gatherv based on the Hockney model.

4.8.2.37 int Hockney_Scatterv_bfs_binomial_max (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial scatterv based on the Hockney model.

4.8.2.38 int Hockney_Gatherv_bfs_binomial_max (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial gatherv based on the Hockney model.

4.8.2.39 int Hockney_Scatterv_ucs_binomial_min (void ∗ *sendbuf*, int ∗ *sendcounts*, int ∗ *displs*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial scatterv based on the Hockney model.

4.8.2.40 int Hockney_Gatherv_ucs_binomial_min (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial gatherv based on the Hockney model.

4.8.2.41 int Hockney_Scatterv_ucs_binomial_max (void ∗ *sendbuf*, int ∗ *sendcounts*, int ∗ *displs*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial scatterv based on the Hockney model.

4.8.2.42 int Hockney_Gatherv_ucs_binomial_max (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial gatherv based on the Hockney model.

4.8.2.43 int Hockney_Scatterv_Traff (void ∗ *sendbuf*, int ∗ *sendcounts*, int ∗ *displs*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Scatterv based on modified Traff using the Hockney model.

4.8.2.44 int Hockney_Gatherv_Traff (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Gatherv based on modified Traff using the Hockney model.

4.8.3 Variable Documentation

4.8.3.1 Hockney_model∗ Hockney_model_instance

The global instance of Hockney model for use in the optimized scatter/gather. Must be initialized by Hockney_initialize and destroyed by Hockney_finalize at all processes.

4.9 PLogP-based collective operations

Functions

- int PLogP_initialize (MPI_Comm comm, PLogP_model ∗model)

- int PLogP_finalize (MPI_Comm comm)
- int PLogP_Scatterv_sorted_flat_asc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_sorted_flat_asc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_sorted_flat_dsc (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_sorted_flat_dsc (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Bcast_dfs_binomial_min (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int PLogP_Reduce_dfs_binomial_min (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int PLogP_Bcast_dfs_binomial_max (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int PLogP_Reduce_bfs_binomial_min (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int PLogP_Reduce_bfs_binomial_max (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int PLogP_Bcast_bfs_binomial_min (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int PLogP_Bcast_bfs_binomial_max (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int PLogP_Reduce_dfs_binomial_max (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int PLogP_Bcast_ucs_binomial_min (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int PLogP_Reduce_ucs_binomial_min (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int PLogP_Bcast_ucs_binomial_max (void ∗buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
- int PLogP_Reduce_ucs_binomial_max (void ∗sendbuf, void ∗recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
- int PLogP_Scatter_dfs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gather_dfs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatter_dfs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gather_dfs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatter_ucs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gather_ucs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatter_ucs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gather_ucs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatter_bfs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

- int PLogP_Gather_bfs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatter_bfs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gather_bfs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_dfs_binomial_min (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_dfs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_dfs_binomial_max (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_dfs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_bfs_binomial_min (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_bfs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_bfs_binomial_max (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_bfs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_ucs_binomial_min (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_ucs_binomial_min (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_ucs_binomial_max (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_ucs_binomial_max (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Scatterv_Traff (void ∗sendbuf, int ∗sendcounts, int ∗displs, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int PLogP_Gatherv_Traff (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int ∗recvcounts, int ∗displs, MPI_Datatype recvtype, int root, MPI_Comm comm)

Variables

- PLogP_model ∗ PLogP_model_instance

4.9.1 Function Documentation

4.9.1.1 int PLogP_initialize (MPI_Comm *comm*, PLogP_model ∗ *model*)

Initializes the instances of the PLogP model (PLogP_model_instance) at all processes in the communicator.

Parameters

comm MPI communicator

model PLogP model (significant only at root)

4.9.1.2 int PLogP_finalize (MPI_Comm *comm*)

Destroys the instances of the PLogP model (PLogP_model_instance) at all processes in the communicatior.

4.9.1.3 int PLogP_Scatterv_sorted_flat_asc (void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Sorted flat-tree scatterv based on the PLogP model.

4.9.1.4 int PLogP_Gatherv_sorted_flat_asc (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Sorted flat-tree gatherv based on the PLogP model.

4.9.1.5 int PLogP_Scatterv_sorted_flat_dsc (void ∗ *sendbuf,* int ∗ *sendcounts,* int ∗ *displs,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Sorted flat-tree scatterv based on the PLogP model.

4.9.1.6 int PLogP_Gatherv_sorted_flat_dsc (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int ∗ *recvcounts,* int ∗ *displs,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Sorted flat-tree gatherv based on the PLogP model.

4.9.1.7 int PLogP_Bcast_dfs_binomial_min (void ∗ *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

DFS binomial bcast based on the PLogP model.

4.9.1.8 int PLogP_Reduce_dfs_binomial_min (void ∗ *sendbuf,* void ∗ *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

DFS binomial reduce based on the PLogP model.

4.9.1.9 int PLogP_Bcast_dfs_binomial_max (void ∗ *buffer,* int *count,* MPI_Datatype *datatype,* int *root,* MPI_Comm *comm*)

DFS binomial bcast based on the PLogP model.

4.9.1.10 int PLogP_Reduce_bfs_binomial_min (void ∗ *sendbuf,* void ∗ *recvbuf,* int *count,* MPI_Datatype *datatype,* MPI_Op *op,* int *root,* MPI_Comm *comm*)

BFS binomial reduce based on the PLogP model.

4.9.1.11 int PLogP_Reduce_bfs_binomial_max (void * *sendbuf*, void * *recvbuf*, int *count*, MPI_Datatype *datatype*, MPI_Op *op*, int *root*, MPI_Comm *comm*)

BFS binomial reduce based on the PLogP model.

4.9.1.12 int PLogP_Bcast_bfs_binomial_min (void * *buffer*, int *count*, MPI_Datatype *datatype*, int *root*, MPI_Comm *comm*)

BFS binomial bcast based on the PLogP model.

4.9.1.13 int PLogP_Bcast_bfs_binomial_max (void * *buffer*, int *count*, MPI_Datatype *datatype*, int *root*, MPI_Comm *comm*)

BFS binomial bcast based on the PLogP model.

4.9.1.14 int PLogP_Reduce_dfs_binomial_max (void * *sendbuf*, void * *recvbuf*, int *count*, MPI_Datatype *datatype*, MPI_Op *op*, int *root*, MPI_Comm *comm*)

DFS binomial reduce based on the PLogP model.

4.9.1.15 int PLogP_Bcast_ucs_binomial_min (void * *buffer*, int *count*, MPI_Datatype *datatype*, int *root*, MPI_Comm *comm*)

UCS binomial bcast based on the PLogP model.

4.9.1.16 int PLogP_Reduce_ucs_binomial_min (void * *sendbuf*, void * *recvbuf*, int *count*, MPI_Datatype *datatype*, MPI_Op *op*, int *root*, MPI_Comm *comm*)

UCS binomial reduce based on the PLogP model.

4.9.1.17 int PLogP_Bcast_ucs_binomial_max (void * *buffer*, int *count*, MPI_Datatype *datatype*, int *root*, MPI_Comm *comm*)

UCS binomial bcast based on the PLogP model.

4.9.1.18 int PLogP_Reduce_ucs_binomial_max (void * *sendbuf*, void * *recvbuf*, int *count*, MPI_Datatype *datatype*, MPI_Op *op*, int *root*, MPI_Comm *comm*)

UCS binomial reduce based on the PLogP model.

4.9.1.19 int PLogP_Scatter_dfs_binomial_min (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial scatter based on the PLogP model.

4.9.1.20 int PLogP_Gather_dfs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

DFS binomial gather based on the PLogP model.

4.9.1.21 int PLogP_Scatter_dfs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

DFS binomial scatter based on the PLogP model.

4.9.1.22 int PLogP_Gather_dfs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

DFS binomial gather based on the PLogP model.

4.9.1.23 int PLogP_Scatter_ucs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial scatter based on the PLogP model.

4.9.1.24 int PLogP_Gather_ucs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial gather based on the PLogP model.

4.9.1.25 int PLogP_Scatter_ucs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial scatter based on the PLogP model.

4.9.1.26 int PLogP_Gather_ucs_binomial_max (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

UCS binomial gather based on the PLogP model.

4.9.1.27 int PLogP_Scatter_bfs_binomial_min (void * *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void * *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

BFS binomial scatter based on the PLogP model.

4.9.1.28 int PLogP_Gather_bfs_binomial_min (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial gather based on the PLogP model.

4.9.1.29 int PLogP_Scatter_bfs_binomial_max (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial scatter based on the PLogP model.

4.9.1.30 int PLogP_Gather_bfs_binomial_max (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial gather based on the PLogP model.

4.9.1.31 int PLogP_Scatterv_dfs_binomial_min (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial scatterv based on the PLogP model.

4.9.1.32 int PLogP_Gatherv_dfs_binomial_min (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial gatherv based on the PLogP model.

4.9.1.33 int PLogP_Scatterv_dfs_binomial_max (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial scatterv based on the PLogP model.

4.9.1.34 int PLogP_Gatherv_dfs_binomial_max (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

DFS binomial gatherv based on the PLogP model.

4.9.1.35 int PLogP_Scatterv_bfs_binomial_min (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial scatterv based on the PLogP model.

4.9.1.36 int PLogP_Gatherv_bfs_binomial_min (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial gatherv based on the PLogP model.

4.9.1.37 int PLogP_Scatterv_bfs_binomial_max (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial scatterv based on the PLogP model.

4.9.1.38 int PLogP_Gatherv_bfs_binomial_max (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

BFS binomial gatherv based on the PLogP model.

4.9.1.39 int PLogP_Scatterv_ucs_binomial_min (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial scatterv based on the PLogP model.

4.9.1.40 int PLogP_Gatherv_ucs_binomial_min (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial gatherv based on the PLogP model.

4.9.1.41 int PLogP_Scatterv_ucs_binomial_max (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial scatterv based on the PLogP model.

4.9.1.42 int PLogP_Gatherv_ucs_binomial_max (void * *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void * *recvbuf*, int * *recvcounts*, int * *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

UCS binomial gatherv based on the PLogP model.

4.9.1.43 int PLogP_Scatterv_Traff (void * *sendbuf*, int * *sendcounts*, int * *displs*, MPI_Datatype *sendtype*, void * *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Scatterv based on modified Traff using the PLogP model.

4.9.1.44 int PLogP_Gatherv_Traff (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int ∗ *recvcounts*, int ∗ *displs*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Gatherv based on modified Traff using the PLogP model.

4.9.2 Variable Documentation

4.9.2.1 PLogP_model∗ PLogP_model_instance

The global instance of PLogP model for use in the optimized scatter/gather. Must be initialized by PLogP_initialize and destroyed by PLogP_finalize at all processes.

4.10 LMO-based collective operations

Functions

- int LMO_initialize (MPI_Comm comm, LMO_model ∗model)
- int LMO_finalize (MPI_Comm comm)
- int LMO_Scatter_split_flat (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
- int LMO_Gather_split_flat (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Variables

- LMO_model ∗ LMO_model_instance

4.10.1 Function Documentation

4.10.1.1 int LMO_initialize (MPI_Comm *comm*, LMO_model ∗ *model*)

Initializes the instances of the LMO model (LMO_model_instance) on all processes in the communicatior.

Parameters

comm MPI communicator

model LMO model (significant only at root)

4.10.1.2 int LMO_Scatter_split_flat (void ∗ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void ∗ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Split flat-tree MPI_Scatter.

LMO_model_instance must be initialized.

4.10.1.3 int LMO_Gather_split_flat (void ∗ *sendbuf,* int *sendcount,* MPI_Datatype *sendtype,* void ∗ *recvbuf,* int *recvcount,* MPI_Datatype *recvtype,* int *root,* MPI_Comm *comm*)

Split flat-tree MPI_Gather.

LMO_model_instance must be initialized.

4.10.2 Variable Documentation

4.10.2.1 LMO_model∗ LMO_model_instance

The global instance of the LMO model for use in the optimized scatter/gather.

Must be initialized by LMO_initialize and destroyed by LMO_finalize on all processes.

5 Class Documentation

5.1 CPM_predictor Struct Reference

Public Attributes

- double(∗ predict_p2p)(void ∗_this, int i, int j, int M)

5.1.1 Detailed Description

An predictor of the communication execution time. Contains a set of the predict functions:

- `double (*predict_p2p)(void* _this, int i, int j, int M)` is a function predicting the execution time of the point-to-point communication operation, where `i` and `j` are the indices of the processors, `M` is a message size.
- `double (*predict_Y)(void* _this, int M, int root, int size)` is a function predicting the execution time of the algorithm of collective communication operation `Y` (for example, `Scatter_binomial`). The `root` parameter is significant for the collective operations with the root processor.

Each of the estimation functions includes a self pointer argument `_this` to be used in the derived data structures (communication performance models).

5.1.2 Member Data Documentation

5.1.2.1 double(∗ CPM_predictor::predict_p2p)(void ∗_this, int i, int j, int M)

Predicts the execution time of the point-to-point communication

Parameters

_this a self pointer

i an index of a processor involved in point-to-point communication

j an index of a processor involved in point-to-point communication

M a message size.

Returns

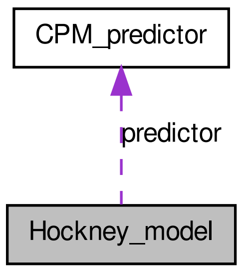
predicted execution time

The documentation for this struct was generated from the following file:

- models/cpm_models.h

5.2 Hockney_model Struct Reference

Collaboration diagram for Hockney_model:



Public Attributes

- CPM_predictor predictor
- int n
- double ∗ a
- double ∗ b

5.2.1 Detailed Description

The heterogeneous Hockney model. The point-to-point parameters for all pairs of processors.

5.2.2 Member Data Documentation

5.2.2.1 CPM_predictor Hockney_model::predictor

Predictor

5.2.2.2 int Hockney_model::n

Number of nodes

5.2.2.3 double* Hockney_model::a

Array of C_n^2 parameters: $\{\alpha_{ij}\}_{i \neq j=0}^{n-1}$

5.2.2.4 double* Hockney_model::b

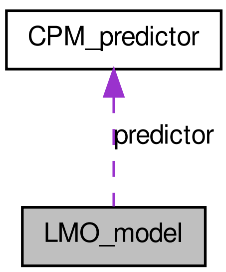
Array of C_n^2 parameters: $\{\beta_{ij}\}_{i \neq j=0}^{n-1}$

The documentation for this struct was generated from the following file:

- models/hockney.h

5.3 LMO_model Struct Reference

Collaboration diagram for LMO_model:



Public Attributes

- CPM_predictor predictor
- int n
- double * C
- double * L
- double * t
- double * b
- int S
- double one2many_small [2]
- double one2many_large [2]
- int M1
- int M2
- double many2one_small [2]
- double many2one_large [2]

5.3.1 Detailed Description

The LMO model

5.3.2 Member Data Documentation

5.3.2.1 CPM_predictor LMO_model::predictor

Predictor

5.3.2.2 int LMO_model::n

number of nodes

5.3.2.3 double* LMO_model::C

array of n average fixed processing delays

5.3.2.4 double* LMO_model::L

array of C_n^2 average to/from latencies ($L_{ij} = L_{ji}$)

5.3.2.5 double* LMO_model::t

array of n average variable processing delays

5.3.2.6 double* LMO_model::b

array of C_n^2 average to/from transmission rates ($\beta_{ij} = \beta_{ji}$)

5.3.2.7 int LMO_model::S

threshold between small and large message sizes for the one2many model

5.3.2.8 double LMO_model::one2many_small[2]

one2many linear model for small message sizes

5.3.2.9 double LMO_model::one2many_large[2]

one2many linear model for large message sizes

5.3.2.10 int LMO_model::M1

threshold between small and medium message sizes for the many2one model

5.3.2.11 int LMO_model::M2

threshold between medium and large message sizes for the many2one model

5.3.2.12 double LMO_model::many2one_small[2]

many2one linear model for small message sizes

5.3.2.13 double LMO_model::many2one_large[2]

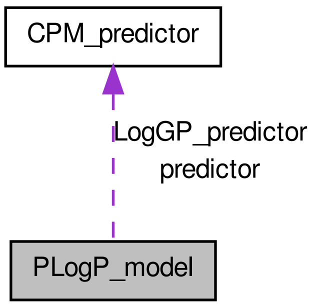
many2one linear model for large message sizes

The documentation for this struct was generated from the following file:

- models/lmo.h

5.4 PLogP_model Struct Reference

Collaboration diagram for PLogP_model:



Public Attributes

- CPM_predictor predictor
- CPM_predictor LogGP_predictor
- int n
- void ** logp

5.4.1 Detailed Description

The heterogeneous PLogP model. The point-to-point parameters for all pairs of processors.

5.4.2 Member Data Documentation

5.4.2.1 CPM_predictor PLogP_model::predictor

Predictor

5.4.2.2 CPM_predictor PLogP_model::LogGP_predictor

LogGP predictor

5.4.2.3 int PLogP_model::n

Number of nodes

5.4.2.4 void** PLogP_model::logp

Array of C_n^2 pointers to the PLogP parameters, which correspond to all pairs of processors. Each pointer refers to the logp_params data structure of the logp_mpi library.

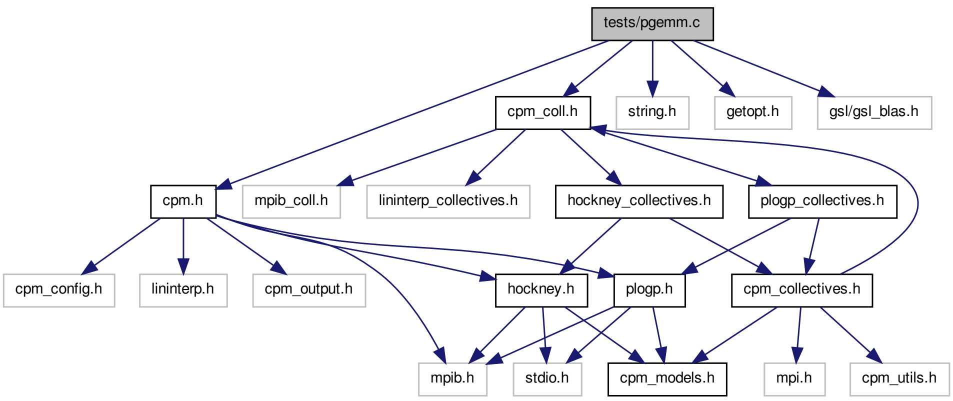
The documentation for this struct was generated from the following file:

- models/plogp.h

6 File Documentation

6.1 tests/pgemm.c File Reference

Include dependency graph for pgemm.c:



6.1.1 Detailed Description

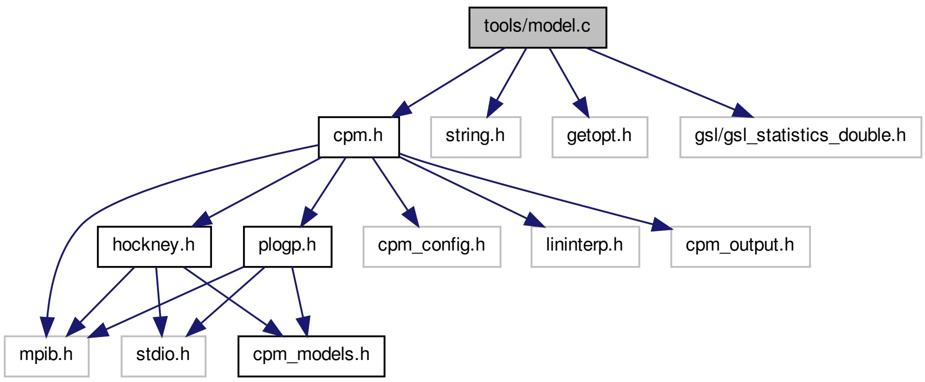
Heterogeneous parallel matrix-matrix product:

- Estimates the execution time of gsl_blas_dgemm on all processors
- Scatters the row blocks of matrix A proportionally to the speed of processors (scatterv)
- Broadcasts matrix B
- Gathers matrix C (gatherv)

6.2 tools/model.c File Reference

Builds the Hockney, PLogP and LMO models and estimates the execution time of point-to-point and collective communication operations. Arguments are described in the MPIBlib manual.

Include dependency graph for model.c:



6.2.1 Detailed Description

Builds the Hockney, PLogP and LMO models and estimates the execution time of point-to-point and collective communication operations. Arguments are described in the MPIBlib manual. **model.plot** draws the graph observation vs predictions.

- input: model.out (model output for different models), p2p.out (p2p output)
- output: p2p.eps (0-1)

Index