CPM: Communication Performance Model

Version 1.0.1 (Revision 108)

Generated by Doxygen 1.5.6

Wed Sep 24 11:31:37 2008

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.

1.1 Authors

```
Alexey Lastovetsky, Vladimir Rychkov, Maureen O'Flynn

Heterogeneous Computing Laboratory
School of Computer Science and Informatics, University College Dublin
Belfield, Dublin 4, Ireland
http://hcl.ucd.ie

{alexey.lastovetsky, vladimir.rychkov, maureen.oflynn}@ucd.ie
```

1.2 Changes

```
1.0.0 (105)
-----------

Initial release
```

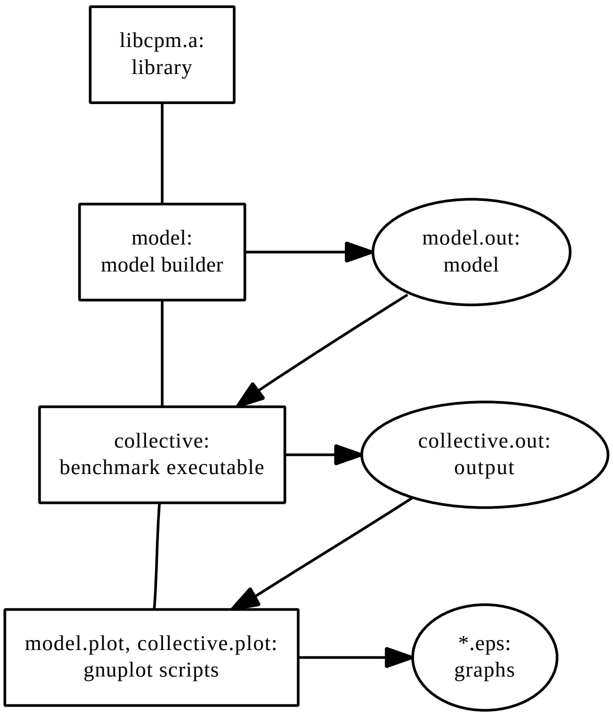
2 Installation

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

Required software:
1. any MPI implementation
2. GSL (GNU Scientific Library)
3. R (The R Project for Statistical Computing)
4. logp_mpi (The MPI LogP Benchmark) - optional
5. Gnuplot - optional

GSL
---
If GSL is intalled in a non-default directory
$ export LD_LIBRARY_PATH=DIR/lib:$LD_LIBRARY_PATH

R
-
1. R sould 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 intalled in a non-default directory
$ export LD_LIBRARY_PATH=$R_HOME/lib:$LD_LIBRARY_PATH

logp_mpi
--------
$ wget ftp://ftp.cs.vu.nl/pub/kielmann/logp_mpi.tar.gz
$ tar -zxvf logp_mpi.tar.gz
$ cd logp_mpi-1.4
$ make

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

Required software:
```

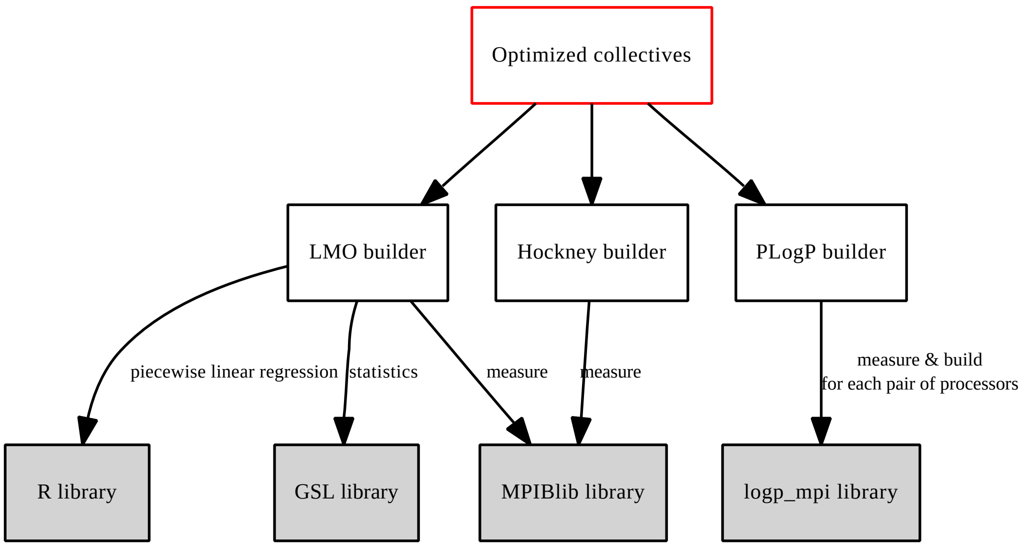
```
1. Subversion
2. GNU autotools
3. Doxygen
4. Graphvis

$ svn co https://hcl.ucd.ie/repos/CPM/trunk CPM
$ cd CPM
$ svn log -v > ChangeLog
$ autoreconf --install
$ mkdir build
$ cd build
$ ../configure --prefix=DIR --enable-debug
$ make install

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 --prefix=DIR
$ make install

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

Packages:
  --with-gsl-dir=DIR      GNU Scientific Library directory
  --with-r-dir=DIR        The R Project for Statistical Computing directory
  --with-logp_mpi-dir=DIR The MPI LogP Benchmark directory (default: without
                          logp_mpi)

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

3 Usage

The package consists of a library, executables and gnuplot scripts.



Library structure:



Benchmarking executables and gnuplot scripts are described in:

- model/main.c
- collective/main.c

Typical parameters of executables are listed in Options for executables.

Using the gnuplot

```
$ gnuplot script_name.plot
```

The gnuplot data files should have names script_name.out

4 Module Documentation

4.1 Collective

Heterogeneous implementations of MPI collective operations.

Functions

- void LMO_initialize (MPI_Comm comm, LMO_model ∗model)

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

- void LMO_finalize (MPI_Comm comm)

  Destroys the instances of the LMO model (LMO_instance) on all processes in the communicatior.

- int CPM_Scatter_linear_opt (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Optimized linear MPI_Scatter.

- int CPM_Gather_linear_opt (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Optimized linear MPI_Gather.

- void Hockney_initialize (MPI_Comm comm, Hockney_model ∗model)

  Initializes the instances of the Hockney model (Hockney_model_instance) on all processes in the communicatior.

- void Hockney_finalize (MPI_Comm comm)

  Destroys the instances of the Hockney model (Hockney_model_instance) on all processes in the communicatior.

- int CPM_Scatter_binomial_opt (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Optimized binomial MPI_Scatter based on the Hockney model.

- int CPM_Scatter_binomial_cor (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Corrupted binomial MPI_Scatter based on the Hockney model.

- int CPM_Gather_binomial_opt (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Optimized binomial MPI_Gather based on the Hockney model.

- int CPM_Gather_binomial_cor (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Corrupted binomial MPI_Gather based on the Hockney model.

- void CPM_p2p_initialize (MPI_Comm comm, int root, int opt)

  Initializes the instances of the optimized/corrupted communicator (CPM_comm_instance) on all processes in the communicatior.

- void CPM_p2p_finalize (MPI_Comm comm)

  Destroys the instances of the p2p execution times (CPM_comm_instance) on all processes in the communicatior.

- int CPM_Scatter_binomial (void ∗sendbuf, int sendcount, MPI_Datatype sendtype, void ∗recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  Binomial MPI_Scatter based on the reordered comm CPM_comm_instance.

Variables

- LMO_model ∗ LMO_instance

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

- Hockney_model ∗ Hockney_model_instance

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

- MPI_Comm CPM_comm_instance

  The global instance of the optimized/corrupted communicator for use in the optimized/corrupted scatter/gather.

4.1.1 Detailed Description

Heterogeneous implementations of MPI collective operations.

4.1.2 Function Documentation

4.1.2.1 void LMO_initialize (MPI_Comm *comm*, LMO_model ∗ *model*)

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

Parameters:

comm MPI communicator

model LMO model (significant only at root)

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

Optimized linear MPI_Scatter.

LMO_instance must be initialized.

4.1.2.3 int CPM_Gather_linear_opt (void $*$ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void $*$ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Optimized linear MPI_Gather.

LMO_instance must be initialized.

4.1.2.4 void Hockney_initialize (MPI_Comm *comm*, Hockney_model $*$ *model*)

Initializes the instances of the Hockney model (Hockney_model_instance) on all processes in the communicatior.

Parameters:

comm MPI communicator

model Hockney model (significant only at root)

4.1.2.5 int CPM_Scatter_binomial_opt (void $*$ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void $*$ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Optimized binomial MPI_Scatter based on the Hockney model.

Hockney_model_instance must be initialized.

4.1.2.6 int CPM_Scatter_binomial_cor (void $*$ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void $*$ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Corrupted binomial MPI_Scatter based on the Hockney model.

Hockney_model_instance must be initialized.

4.1.2.7 int CPM_Gather_binomial_opt (void $*$ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void $*$ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Optimized binomial MPI_Gather based on the Hockney model.

Hockney_model_instance must be initialized.

4.1.2.8 int CPM_Gather_binomial_cor (void $*$ *sendbuf*, int *sendcount*, MPI_Datatype *sendtype*, void $*$ *recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*)

Corrupted binomial MPI_Gather based on the Hockney model.

Hockney_model_instance must be initialized.

4.1.2.9 void CPM_p2p_initialize (MPI_Comm *comm*, int *root*, int *opt*)

Initializes the instances of the optimized/corrupted communicator (CPM_comm_instance) on all processes in the communicatior.

Parameters:

comm basic communicator

root root in the resulted communicator

opt optimized (1) or corrupted (0) communicator

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

Binomial MPI_Scatter based on the reordered comm CPM_comm_instance.

CPM_comm_instance must be initialized.

4.1.3 Variable Documentation

4.1.3.1 LMO_model∗ LMO_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.

4.1.3.2 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 on all processes.

4.1.3.3 MPI_Comm CPM_comm_instance

The global instance of the optimized/corrupted communicator for use in the optimized/corrupted scatter/gather.

Must be initialized by CPM_p2p_initialize and destroyed by CPM_p2p_finalize on all processes.

4.2 Options for executables

4.2.1 Detailed Description

As getopt cannot be reused, this module provides an interface for getopt.

Typical parameters of executables are as follows:

- **-h** help
- **-O** *S* collective operation (required):
  MPI_Scatter, MPIB_Scatter_linear, MPIB_Scatter_binomial
  MPI_Gather, MPIB_Gather_linear, MPIB_Gather_binomial
  MPI_Scatterv
  MPI_Gatherv
  CPM_Scatter_linear_opt
  CPM_Gather_linear_opt
  CPM_Scatter_binomial_opt, CPM_Scatter_binomial_cor
  CPM_Gather_binomial_opt, CPM_Gather_binomial_cor
- **-t** *S* timing: max, root, global (default: max)
- **-s** *I* message size stride (default: 1024)
- **-m** *I* maximum message size (default: 102400)
- **-p** 0/1 parallel p2p benchmarking (default: 1)

- **-r** *I* minimum number of repetitions (default: 5)
- **-R** *I* maximum number of repetitions (default: 100)
- **-c** *D* confidence level: $0 < D < 1$ (default: 0.95)
- **-e** *D* error: $0 < D < 1$ (default: 0.025)
- **-M** *S* model name (required)
- **-i** *S* input model file, stdin - standard input (default: none - model will be built)
- **-o** *S* output model file, stdout - standard output (default: stdout)

where:

- *S* - string
- *I* - integer
- *D* - double

4.3 Hockney

Computes the parameters of the Hockney model.

Data Structures

- struct Hockney_model

  Hockney model.

Functions

- void Hockney_build (MPI_Comm comm, int M, int parallel, MPIB_precision precision, Hockney_-model ∗∗model)

  Builds the Hockney model.

- Hockney_model ∗ Hockney_alloc (int n)

  Allocates memory for Hockney model.

- void Hockney_free (Hockney_model ∗model)

  Frees Hockney model.

- Hockney_model ∗ Hockney_load (const char ∗filename)

  Loads Hockney model.

- void Hockney_save (const char ∗filename, Hockney_model ∗model)

  Saves Hockney model.

- double Hockney_estimate (Hockney_model ∗model, int i, int j, int M)

  Estimates the execution time of a point-to-point communication.

- double Hockney_estimate_sg_linear_sequential (Hockney_model ∗model, int root, int M)

  Heterogeneous estimation of linear scatter/gather (sequential point-to-point communications).

- double Hockney_estimate_sg_linear_parallel (Hockney_model ∗model, int root, int M)

  Heterogeneous estimation of linear scatter/gather (parallel point-to-point communications).

- double Hockney_estimate_sg_binomial (Hockney_model ∗model, int root, int M)

  Heterogeneous estimation of binomial scatter/gather (point-to-point communications are sequential within the same root or parallel otherwise).

- double Hockney_hestimate_sg_linear_sequential (Hockney_model ∗model, int root, int M)

  Homogeneous estimation of linear scatter/gather (sequential point-to-point communications).

- double Hockney_hestimate_sg_linear_parallel (Hockney_model ∗model, int root, int M)

  Homogeneous estimation of linear scatter/gather (parallel point-to-point communications).

- double Hockney_hestimate_sg_binomial (Hockney_model ∗model, int root, int M)

  Homogeneous estimation of binomial scatter/gather (point-to-point communications are sequential within the same root or parallel otherwise).

4.3.1 Detailed Description

Computes the parameters of the Hockney model.

4.3.2 Function Documentation

4.3.2.1 void Hockney_build (MPI_Comm *comm*, int *M*, int *parallel*, MPIB_precision *precision*, Hockney_model ∗∗ *model*)

Builds the Hockney model.

- Measures the execution time $T_{ij}(0)$ of each $i \overset{0}{\underset{0}{\longleftrightarrow}} j$ roundtrip in the communicator, $i < j$, to find $\alpha = T_{ij}(0)$. 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 $\beta = \frac{T_{ij}(M) - \alpha}{M}$. 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
parallel	several non-overlapped point-to-point communications at the same time if non-zero
precision	measurement precision
model	Hockney model (significant only at root)

4.3.2.2 void Hockney_free (Hockney_model * *model*)

Frees Hockney model.

Parameters:

model Hockney model

4.3.2.3 Hockney_model* Hockney_load (const char * *filename*)

Loads Hockney model.

Parameters:

filename file name ("stdin" = stdin)

Returns:

Hockney model

4.3.2.4 void Hockney_save (const char * *filename*, Hockney_model * *model*)

Saves Hockney model.

Parameters:

filename file name ("stdout" = stdout)

model Hockney model

4.3.2.5 double Hockney_estimate_sg_linear_sequential (Hockney_model * *model*, int *root*, int *M*)

Heterogeneous estimation of linear scatter/gather (sequential point-to-point communications).

$$\sum_{i=0, i \neq r}^{n-1} (\alpha_{ri} + \beta_{ri} M)$$

4.3.2.6 double Hockney_estimate_sg_linear_parallel (Hockney_model * *model*, int *root*, int *M*)

Heterogeneous estimation of linear scatter/gather (parallel point-to-point communications).

$$\max_{i=0, i \neq r}^{n-1} (\alpha_{ri} + \beta_{ri} M)$$

4.3.2.7 double Hockney_estimate_sg_binomial (Hockney_model * *model*, int *root*, int *M*)

Heterogeneous estimation of binomial scatter/gather (point-to-point communications are sequential within the same root or parallel otherwise).

$$\overbrace{\sum_{k=log_2(n-1)}^{0} (\alpha_{ri} + \beta_{ri} 2^k M) + \max_{k=log_2(n-1)-1}^{0} (S(k))}^{S(log_2(n-1))}$$

4.3.2.8 double Hockney_hestimate_sg_linear_sequential (Hockney_model ∗ *model*, int *root*, int *M*)

Homogeneous estimation of linear scatter/gather (sequential point-to-point communications).

$(n-1)(\alpha+\beta M)$

4.3.2.9 double Hockney_hestimate_sg_linear_parallel (Hockney_model ∗ *model*, int *root*, int *M*)

Homogeneous estimation of linear scatter/gather (parallel point-to-point communications).

$\alpha+\beta M$

4.3.2.10 double Hockney_hestimate_sg_binomial (Hockney_model ∗ *model*, int *root*, int *M*)

Homogeneous estimation of binomial scatter/gather (point-to-point communications are sequential within the same root or parallel otherwise).

$(log_2 n)\alpha+(n-1)\beta M$

4.4 LMO

Computes the parameters of the LMO communication performance model.

Data Structures

- struct LMO_model

  LMO model.

Functions

- LMO_model ∗ LMO_alloc (int n)

  Allocates LMO model.

- void LMO_free (LMO_model ∗model)

  Frees LMO model.

- LMO_model ∗ LMO_load (const char ∗filename)

  Loads LMO model.

- void LMO_save (const char ∗filename, LMO_model ∗model)

  Saves LMO model.

- int CPM_initR ()

  Initializes R.

- void CPM_endR ()

  Finalizes R.

- void LMO_build_p2p (LMO_model ∗model, MPI_Comm comm, int parallel, MPIB_precision precision)

  Builds the heterogeneous point-to-point model.

- void LMO_build_one2many (LMO_model ∗model, MPI_Comm comm, int stride, int max_size, MPIB_precision precision)

  Computes the parameters of one-to-many.

- void LMO_build_many2one (LMO_model ∗model, MPI_Comm comm, int stride, int max_size, MPIB_precision precision)

  Computes the parameters of many-to-one.

- void LMO_build (MPI_Comm comm, int stride, int max_size, int parallel, MPIB_precision precision, LMO_model ∗∗model)

  Computes the parameters of LMO model.

- double LMO_estimate_p2p (LMO_model ∗model, int i, int j, int M)

  Estimates the execution time of point-to-point communication.

- double LMO_estimate_one2many (LMO_model ∗model, int root, int M)

  Estimates the execution time of one-to-many communication.

- double LMO_estimate_many2one (LMO_model ∗model, int root, int M)

  Estimates the execution time of many-to-one communication.

4.4.1 Detailed Description

Computes the parameters of the LMO communication performance model.

4.4.2 Function Documentation

4.4.2.1 LMO_model∗ LMO_alloc (int *n*)

Allocates LMO model.

Parameters:

n number of processors

Returns:

LMO model

4.4.2.2 void LMO_free (LMO_model ∗ *model*)

Frees LMO model.

Parameters:

model LMO model

4.4.2.3 LMO_model∗ LMO_load (const char ∗ *filename*)

Loads LMO model.

Parameters:

filename file name ("stdin" = stdin)

Returns:

LMO model

4.4.2.4 void LMO_save (const char ∗ *filename*, LMO_model ∗ *model*)

Saves LMO model.

Parameters:

filename file name ("stdout" = stdout)

model LMO model

4.4.2.5 void LMO_build_p2p (LMO_model ∗ *model*, MPI_Comm *comm*, int *parallel*, MPIB_-precision *precision*)

Builds the heterogeneous point-to-point model.

- 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:

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

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

  and checks confidence intervals.

- Finds the variable processing delays and transmission rates.

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

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

  averages solutions:

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

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

  and checks confidence intervals.

It is called by LMO_build.

Parameters:

model LMO model, must be allocated and filled by LMO_build_one2many and LMO_build_-many2one (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

4.4.2.6 void LMO_build_one2many (LMO_model * *model*, MPI_Comm *comm*, int *stride*, int *max_size*, MPIB_precision *precision*)

Computes 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 CPM_initR at root beforehand.

It is called by LMO_build.

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

4.4.2.7 void LMO_build_many2one (LMO_model $*$ *model*, MPI_Comm *comm*, int *stride*, int *max_size*, MPIB_precision *precision*)

Computes 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 CPM_initR 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 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_build.

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

4.4.2.8 void LMO_build (MPI_Comm *comm*, int *stride*, int *max_size*, int *parallel*, MPIB_precision *precision*, LMO_model $**$ *model*)

Computes the parameters of LMO model.

Calls LMO_build_one2many, LMO_build_many2one, LMO_build_p2p. R must be initialized by CPM_-initR at root beforehand.

Parameters:

comm communicator

stride stride for message sizes

max_size maximum message size

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

precision measurement parameters

model LMO model (significant only at root)

4.4.2.9 double LMO_estimate_p2p (LMO_model $*$ *model*, int *i*, int *j*, int *M*)

Estimates 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:

model LMO model

i index of the precessor

j index of the precessor

M message size

Returns:

predicted execution time

4.4.2.10 double LMO_estimate_one2many (LMO_model * *model*, int *root*, int *M*)

Estimates the execution time of one-to-many communication.

The execution time of $0 \xrightarrow{M} 1..n-1$ is equal to

$$(n-1)(C_0 + Mt_i) + \begin{cases} \max_{i=1}^{n-1}(L_{0i} + C_i + M(\frac{1}{\beta_{0i}} + t_i)) & M < S \\ \sum_{i=1}^{n-1}(L_{0i} + C_i + M(\frac{1}{\beta_{0i}} + t_i)) & M \geq S \end{cases}$$

Parameters:

model LMO model

root root precessor

M message size

Returns:

predicted execution time

4.4.2.11 double LMO_estimate_many2one (LMO_model * *model*, int *root*, int *M*)

Estimates the execution time of many-to-one communication.

The execution time of $0 \xrightarrow{M} 1..n-1$ is equal to

$$(n-1)(C_0 + Mt_i) + \begin{cases} \max_{i=1}^{n-1}(L_{0i} + C_i + M(\frac{1}{\beta_{0i}} + t_i)) & M < M_1 \\ \sum_{i=1}^{n-1}(L_{0i} + C_i + M(\frac{1}{\beta_{0i}} + t_i)) & M > M_2 \end{cases}$$

Parameters:

model LMO model

root root precessor

M message size

Returns:

predicted execution time

4.5 Measurement

Measures the execution time of point-to-point and collective communications.

Functions

- void CPM_measure_p2pp (MPI_Comm comm, int M, int parallel, MPIB_precision precision, MPIB_result ∗results)

  Measures the 1-to-2 execution times.

- 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.

4.5.1 Detailed Description

Measures the execution time of point-to-point and collective communications.

4.5.2 Function Documentation

4.5.2.1 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)

4.5.2.2 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)

4.6 PLogP

Computes the parameters of the parameterized LogP model.

Data Structures

- struct PLogP_model

  PLogP model.

Functions

- void PLogP_build (MPI_Comm comm, int stride, int max_size, int parallel, MPIB_precision precision, PLogP_model ∗∗model)

  Builds PLogP model.

- void PLogP_free (PLogP_model ∗model)

  Frees PLogP model.

- PLogP_model ∗ PLogP_load (const char ∗filename)

  Loads PLogP model.

- void PLogP_save (const char ∗filename, PLogP_model ∗model)

  Saves PLogP model.

- double PLogP_estimate (PLogP_model ∗model, int i, int j, int M)

  Estimates the execution time of a point-to-point communication.

- double PLogP_estimate_LogGP (PLogP_model ∗model, int i, int j, int M)

  Estimates the execution time of a point-to-point communication according to LogGP model.

4.6.1 Detailed Description

Computes the parameters of the parameterized LogP model.

PLogP is an extension of the LogP model that describes a network in terms of the following parameters:

- L : latency - this is the end-to-end latency between nodes
- o : overhead - for sending and receiving $o_s(m)$ and $o_r(m)$ respectively for message size m
- g : gap per message depends on message size $g(m)$
- P : number of nodes involved in communication

The PLogP model is defined in terms of these parameters, the end-to-end latency L, sender and receiver overheads, $o_s(m)$ and $o_r(m)$ respectively, gap per message g(m), and number of nodes involved in communication P. In this model sender and receiver overheads and gap per message depend on the message size. Time to send a message of size m between two nodes in the PLogP model is $L + g(m)$.

4.6.2 Function Documentation

4.6.2.1 void PLogP_build (MPI_Comm *comm*, int *stride*, int *max_size*, int *parallel*, MPIB_precision *precision*, PLogP_model ** *model*)

Builds PLogP model.

Calls logp_mpi library.

Parameters:

comm communicator
stride stride for message sizes
max_size maximum message size
parallel several non-overlapped point-to-point communications at the same time if non-zero
precision measurement precision
model PLogP model

4.6.2.2 PLogP_model* PLogP_load (const char * *filename*)

Loads PLogP model.

Parameters:

filename file name ("stdin" = stdin)

Returns:

PLogP model

4.6.2.3 void PLogP_save (const char * *filename*, PLogP_model * *model*)

Saves PLogP model.

Parameters:

filename file name ("stdout" = stdout)
model PLogP model

4.6.2.4 double PLogP_estimate_LogGP (PLogP_model ∗ *model*, int *i*, int *j*, int *M*)

Estimates the execution time of a point-to-point communication according to LogGP model.

$T = L + 2 * o + G(M - 1)$

The meaning of the parameters of LogGP and PLogP models:

- $L = L_p + g_p(1) - os_p(1) - or_p(1)$
- $2 * o = os_p(1) + or_p(1)$
- $G = g_p(M_{max})/M_{max}$

Parameters:

model PLogP model

i index of the process

j index of the process

M message size

4.7 Utilities

Utility functions.

Defines

- #define CPM_C3(n) (n) ∗ ((n) - 1) ∗ ((n) - 2) / 6

  C_n^3

- #define CPM_IJK2INDEX(n, i, j, k) (2 ∗ (n) - (i) - 1) ∗ ((i) - 1) ∗ (i) / 4 + (2 ∗ (n) - (i) - (j) + 1) ∗ ((j) - (i)) / 2 + (k) - (j) - 1

  The index of the (i, j, k) element in the array of C_n^3 elements, $i < j < k < n$.

4.7.1 Detailed Description

Utility functions.

4.7.2 Define Documentation

4.7.2.1 #define CPM_IJK2INDEX(n, i, j, k) (2 ∗ (n) - (i) - 1) ∗ ((i) - 1) ∗ (i) / 4 + (2 ∗ (n) - (i) - (j) + 1) ∗ ((j) - (i)) / 2 + (k) - (j) - 1

The index of the (i, j, k) element in the array of C_n^3 elements, $i < j < k < n$.

$$\frac{C_{n-1}^2 + C_{n-i}^2}{2} i + \frac{(n-i+1)+(n-j)}{2}(j-i) + (k-j-1)$$

5 Data Structure Documentation

5.1 Hockney_model Struct Reference

Hockney model.

```
#include <cpm_hockney.h>
```

Data Fields

- int n
- double $*$ a
- double $*$ b

5.1.1 Detailed Description

Hockney model.

$T = \alpha + \beta M$

5.1.2 Field Documentation

5.1.2.1 int Hockney_model::n

number of nodes

5.1.2.2 double$*$ Hockney_model::a

array of C_n^2 of α

5.1.2.3 double$*$ Hockney_model::b

array of C_n^2 of β

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

- src/cpm_hockney.h

5.2 LMO_model Struct Reference

LMO model.

```
#include <cpm_lmo.h>
```

Data Fields

- 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]

5.2.1 Detailed Description

LMO model.

See LMO_estimate_p2p, LMO_estimate_one2many, LMO_estimate_many2one

5.2.2 Field Documentation

5.2.2.1 int LMO_model::n

number of nodes

5.2.2.2 double$*$ LMO_model::C

array of n average fixed processing delays

5.2.2.3 double$*$ LMO_model::L

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

5.2.2.4 double$*$ LMO_model::t

array of n average variable processing delays

5.2.2.5 double$*$ LMO_model::b

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

5.2.2.6 int LMO_model::S

threshold between small and large message sizes for the one2many model

5.2.2.7 double LMO_model::one2many_small[2]

one2many linear model for small message sizes

5.2.2.8 double LMO_model::one2many_large[2]

one2many linear model for large message sizes

5.2.2.9 int LMO_model::M1

threshold between small and medium message sizes for the many2one model

5.2.2.10 int LMO_model::M2

threshold between medium and large message sizes for the many2one model

5.2.2.11 double LMO_model::many2one_small[2]

many2one linear model for small message sizes

5.2.2.12 double LMO_model::many2one_large[2]

many2one linear model for large message sizes

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

- src/cpm_lmo.h

5.3 PLogP_model Struct Reference

PLogP model.

`#include <cpm_plogp.h>`

Data Fields

- int n
- void $**$ logp

5.3.1 Detailed Description

PLogP model.

See PLogP_estimate.

5.3.2 Field Documentation

5.3.2.1 int PLogP_model::n

number of nodes

5.3.2.2 void$**$ PLogP_model::logp

array of C_n^2 p2p precision

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

- src/cpm_plogp.h

6 File Documentation

6.1 model/main.c File Reference

Builds the LMO, Hockney and PLogP models and estimates the execution time of ppoint-to-point and collective communication operations.

```
#include "cpm.h"
#include <string.h>
#include <getopt.h>
#include <stdio.h>
```

6.1.1 Detailed Description

Builds the LMO, Hockney and PLogP models and estimates the execution time of ppoint-to-point and collective communication operations.

model.plot draws the graph observation vs predictions.

- input: lmo.out, hockney.out, plogp.out (model output for different models), p2p.out (p2p output), scatter.out and gather.out (collective outputs for MPIB_Scatter_linear and MPIB_Gather_linear)
- output: p2p.eps (0-1), scatter.eps, gather.eps

6.2 collective/main.c File Reference

Collective benchmark.

```
#include "cpm.h"
#include <string.h>
#include <getopt.h>
#include <stdio.h>
```

6.2.1 Detailed Description

Collective benchmark.

Checks and benchmarks collective operations, including model-based ones.

Plot can be made by **MPIBlib/collective/collective.plot**

Index