CPM: Communication Performance Model

Version 1.0.0 (Revision 105)

Generated by Doxygen 1.5.6

Wed Sep 3 09:20:42 2008

Contents

1 Introduction

CPM (Communication Performance Model) provides the modeling of performance of MPI communications.

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
```


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 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 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

$ 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
-------------

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

3 Usage

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

3.1 Executables and gnuplot scripts

Benchmarking executables and gnuplot scripts that visualize the results are described in:

- model/main.c
- collective/main.c
- p2p/main.c

Typical parameters of executables are as follows:

- **-h** help
- **-i** *S* input model file, stdin - standard input (default: none - model will be built)
- **-o** *S* output model file, stdout - standard output (default: stdout)
- **-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_opt
  CPM_Gather_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** 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)

where:

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

Using the gnuplot

```
$ gnuplot script_name.plot
```

The gnuplot data files should have names script_name.out

4 Module Documentation

4.1 Executables

4.2 Collective

Heterogeneous implementations of MPI collective operations.

Functions

- void CPM_model_initialize (MPI_Comm comm, CPM_model ∗model)

  Initializes the instances of the CPM model (CPM_model_instance) on all processes in the communicatior.

- void CPM_model_finalize (MPI_Comm comm)

  Destroys the instances of the CPM model (CPM_model_instance) on all processes in the communicatior.

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

  Optimized MPI_Scatter.

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

  Optimized 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

- CPM_model * CPM_model_instance

  The global instance of the CPM 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.

4.2.1 Detailed Description

Heterogeneous implementations of MPI collective operations.

4.2.2 Function Documentation

4.2.2.1 void CPM_model_initialize (MPI_Comm *comm*, CPM_model * *model*)

Initializes the instances of the CPM model (CPM_model_instance) on all processes in the communicatior.

Parameters:

comm MPI communicator

model CPM model (significant only at root)

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

Optimized MPI_Scatter.

CPM_model_instance must be initialized.

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

Optimized MPI_Gather.

CPM_model_instance must be initialized.

4.2.2.4 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)

4.2.2.5 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.

4.2.2.6 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.

4.2.2.7 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.

4.2.2.8 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.

4.2.2.9 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

4.2.2.10 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.

4.2.3 Variable Documentation

4.2.3.1 CPM_model∗ CPM_model_instance

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

Must be initialized by CPM_model_initialize and destroyed by CPM_model_finalize on all processes.

4.2.3.2 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.

4.2.3.3 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.

4.3 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.

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 $i \xleftrightarrow[0]{0} j$ 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 $i \xleftrightarrow[M]{M} j$ 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

4.4 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.

4.4.1 Detailed Description

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

4.4.2 Function Documentation

4.4.2.1 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)

4.4.2.2 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)

4.5 Model

Computes the parameters of the communication performance model.

Data Structures

- struct CPM_model

  CPM model.

Functions

- CPM_model $*$ CPM_model_alloc (int n)

  Allocates CPM model.

- void CPM_model_free (CPM_model $*$model)

  Frees CPM model.

- CPM_model $*$ CPM_model_load (const char $*$filename)

  Loads CPM model.

- void CPM_model_save (const char $*$filename, CPM_model $*$model)

  Saves CPM model.

- int CPM_initR ()

  Initializes R.

- void CPM_endR ()

  Finalizes R.

- void CPM_build_p2p (CPM_model ∗model, MPI_Comm comm, int parallel, MPIB_precision precision)

  Builds the heterogeneous point-to-point model.

- void CPM_build_one2many (CPM_model ∗model, MPI_Comm comm, int stride, int max_size, MPIB_precision precision)

  Computes the parameters of one-to-many.

- void CPM_build_many2one (CPM_model ∗model, MPI_Comm comm, int stride, int max_size, MPIB_precision precision)

  Computes the parameters of many-to-one.

- void CPM_build (MPI_Comm comm, int stride, int max_size, int parallel, MPIB_precision precision, CPM_model ∗∗model)

  Computes the parameters of CPM model.

- double CPM_estimate_p2p (CPM_model ∗model, int i, int j, int M)

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

- double CPM_estimate_one2many (CPM_model ∗model, int root, int M)

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

- double CPM_estimate_many2one (CPM_model ∗model, int root, int M)

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

4.5.1 Detailed Description

Computes the parameters of the communication performance model.

4.5.2 Function Documentation

4.5.2.1 CPM_model∗ CPM_model_alloc (int *n*)

Allocates CPM model.

Parameters:

n number of processors

Returns:

CPM model

4.5.2.2 void CPM_model_free (CPM_model ∗ *model*)

Frees CPM model.

Parameters:

model CPM model

4.5.2.3 CPM_model* CPM_model_load (const char * *filename*)

Loads CPM model.

Parameters:

filename file name ("stdin" = stdin)

Returns:

CPM model

4.5.2.4 void CPM_model_save (const char * *filename*, CPM_model * *model*)

Saves CPM model.

Parameters:

filename file name ("stdout" = stdout)

model CPM model

4.5.2.5 void CPM_build_p2p (CPM_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:

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

  averages solutions:

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

  and checks confidence intervals.

- Finds the variable processing delays and transmission rates.

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

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

averages solutions:

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

and checks confidence intervals.

It is called by CPM_build.

Parameters:

model CPM model, must be allocated and filled by CPM_build_one2many and CPM_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

4.5.2.6 void CPM_build_one2many (CPM_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 CPM_build.

Parameters:

model CPM 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 CPM_build_many2one (CPM_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 CPM_build.

Parameters:

model CPM 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 CPM_build (MPI_Comm *comm*, int *stride*, int *max_size*, int *parallel*, MPIB_precision *precision*, CPM_model ** *model*)

Computes the parameters of CPM model.

Calls CPM_build_one2many, CPM_build_many2one, CPM_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 CPM model (significant only at root)

4.5.2.9 double CPM_estimate_p2p (CPM_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

$C_i + L_{ij} + C_j + M(t_i + \beta_{ij} + t_j)$

Parameters:

model CPM model

i index of the precessor

j index of the precessor

M message size

Returns:

predicted execution time

4.5.2.10 double CPM_estimate_one2many (CPM_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

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

Parameters:

model CPM model
root root precessor
M message size

Returns:

predicted execution time

4.5.2.11 double CPM_estimate_many2one (CPM_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

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

Parameters:

model CPM model
root root precessor
M message size

Returns:

predicted execution time

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 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 *parallel*, MPIB_precision *precision*, PLogP_model ** *model*)

Builds PLogP model.

Calls logp_mpi library.

Parameters:

comm communicator

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 CPM_model Struct Reference

CPM model.

```
#include <cpm_model.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]

5.1.1 Detailed Description

CPM model.

See CPM_estimate_p2p, CPM_estimate_one2many, CPM_estimate_many2one

5.1.2 Field Documentation

5.1.2.1 int CPM_model::n

number of nodes

5.1.2.2 double* CPM_model::C

array of n average fixed processing delays

5.1.2.3 double* CPM_model::L

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

5.1.2.4 double* CPM_model::t

array of n average variable processing delays

5.1.2.5 double* CPM_model::b

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

5.1.2.6 int CPM_model::S

threshold between small and large message sizes for the one2many model

5.1.2.7 double CPM_model::one2many_small[2]

one2many linear model for small message sizes

5.1.2.8 double CPM_model::one2many_large[2]

one2many linear model for large message sizes

5.1.2.9 int CPM_model::M1

threshold between small and medium message sizes for the many2one model

5.1.2.10 int CPM_model::M2

threshold between medium and large message sizes for the many2one model

5.1.2.11 double CPM_model::many2one_small[2]

many2one linear model for small message sizes

5.1.2.12 double CPM_model::many2one_large[2]

many2one linear model for large message sizes

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

- src/cpm_model.h

5.2 Hockney_model Struct Reference

Hockney model.

#include <cpm_hockney.h>

Data Fields

- int n
- double * a
- double * b

5.2.1 Detailed Description

Hockney model.

$T = \alpha + \beta M$

5.2.2 Field Documentation

5.2.2.1 int Hockney_model::n

number of nodes

5.2.2.2 double* Hockney_model::a

array of C_n^2 of α

5.2.2.3 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

5.3 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

Estimates the execution time of p2p and collective operations according to the model.

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

6.1.1 Detailed Description

Estimates the execution time of p2p and collective operations according to the model.

The model is built or read from file.

model.plot draws the graph of the execution time (sec) against message size (kb) with error bars: observation and prediction.

- input: model.out
- output: scatter.eps (MPIB_Scatter_linear), gather.eps (MPIB_Gather_linear), p2p.eps (0-1)

6.2 collective/main.c File Reference

Collective benchmark.

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

6.2.1 Detailed Description

Collective benchmark.

Checks and benchmarks collective operations

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

6.3 p2p/main.c File Reference

p2p benchmark and Hockney, PLogP/LogGP models

```
#include "cpm.h"
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
```

6.3.1 Detailed Description

p2p benchmark and Hockney, PLogP/LogGP models

Benchmarks p2p communications and build Hokney, PLogP/LogGP models for all pairs

p2p.plot draws the graph of the execution time (sec) against message size (kb).

- input: p2p.out
- output: 0-1.eps (0-1 with error bars, comparison with predictions), 0-1-2.eps (comparison of 0-1, 0-2, 1-2)

Index