PCU

4.0

Generated by Doxygen 1.8.6

Fri Feb 21 2014 19:15:38

Contents

Chapter 1

Introduction

PCU (the Parallel Control Utility) is a library for parallel computation based on MPI with additional support for hybrid MPI/thread environments. PCU provides three things to users:

1. A hybrid phased message passing system
2. Hybrid collective operations
3. A thread management system

Phased message passing is similar to Bulk Synchronous Parallel. All messages are exchanged in a phase, which is a collective operation involving all threads in the parallel program. During a phase, the following events happen in sequence:

1. All threads send non-blocking messages to other threads
2. All threads receive all messages sent to them during this phase PCU provides termination detection, which is the ability to detect when all messages have been received without prior knowledge of which threads are sending to which.

To write hybrid MPI/thread programs, PCU provides a function that creates threads within an MPI process, similar to the way mpirun creates multiple processes. PCU assigns ranks to these threads and has them each run the same function, with thread-specific input arguments to the function.

Once a program has created threads using PCU, it can call the message passing API from within threads, which will behave as if each thread were an MPI process. Threads have unique ranks and can send messages to one another, regardless of which process they are in.

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 pcu.c File Reference

Functions

- int PCU_Comm_Init (void)

  Initializes the PCU library.
- int PCU_Comm_Free (void)

  Frees all PCU library structures.
- int PCU_Comm_Self (void)

  Returns the communication rank of the calling thread.
- int PCU_Comm_Peers (void)

  Returns the number of threads in the program.
- void PCU_Comm_Begin (void)

  Begins a PCU communication phase.
- int PCU_Comm_Pack (int to_rank, const void *data, size_t size)

  Packs data to be sent to to_rank.
- int PCU_Comm_Send (void)

  Sends all buffers for this communication phase.
- bool PCU_Comm_Listen (void)

  Tries to receive a buffer for this communication phase.
- int PCU_Comm_Sender (void)

  *Returns in * from_rank the sender of the current received buffer.*
- bool PCU_Comm_Unpacked (void)

  Returns true if the current received buffer has been completely unpacked.
- int PCU_Comm_Unpack (void *data, size_t size)

  Unpacks a block of data from the current received buffer.
- void PCU_Add_Doubles (double *p, size_t n)

  Performs an Allreduce sum of double arrays.
- void PCU_Min_Doubles (double *p, size_t n)

  Performs an Allreduce minimum of double arrays.
- void PCU_Max_Doubles (double *p, size_t n)

  Performs an Allreduce maximum of double arrays.
- void PCU_Add_Ints (int *p, size_t n)

  Performs an Allreduce sum of integers.
- void PCU_Add_Longs (long *p, size_t n)

  Performs an Allreduce sum of long integers.

- void PCU_Exscan_Ints (int *p, size_t n)

  Performs an exclusive prefix sum of integer arrays.
- void PCU_Exscan_Longs (long *p, size_t n)

  See PCU_Exscan_Ints.
- void PCU_Min_Ints (int *p, size_t n)

  Performs an Allreduce minimum of int arrays.
- void PCU_Max_Ints (int *p, size_t n)

  Performs an Allreduce maximum of int arrays.
- int PCU_Thrd_Run (int nthreads, PCU_Thrd_Func function, void **in_out)

  Runs nthreads instances of function, each in a thread.
- int PCU_Thrd_Self (void)

  Returns the process-unique rank of the calling thread.
- int PCU_Thrd_Peers (void)

  Returns the number of threads running in the current process.
- int PCU_Comm_Rank (int *rank)

  Similar to PCU_Comm_Self, returns the rank as an argument.
- int PCU_Comm_Size (int *size)

  Similar to PCU_Comm_Peers, returns the size as an argument.
- int PCU_Comm_Start (PCU_Method method)

  Deprecated, see PCU_Comm_Begin.
- int PCU_Comm_Packed (int to_rank, size_t *size)

  *Returns in * size the number of bytes being sent to to_rank.*
- int PCU_Comm_Write (int to_rank, const void *data, size_t size)

  Packs a message to be sent to to_rank.
- int PCU_Comm_Receive (bool *done)

  Similar to PCU_Comm_Listen, returns the status as an argument.
- bool PCU_Comm_Read (int *from_rank, void **data, size_t *size)

  Receives a message for this communication phase.
- int PCU_Comm_From (int *from_rank)

  Similar to PCU_Comm_Sender, returns the rank as an argument.
- int PCU_Comm_Received (size_t *size)

  *Returns in * size the bytes in the current received buffer.*
- void * PCU_Comm_Extract (size_t size)

  Extracts a block of data from the current received buffer.

3.1.1 Function Documentation

3.1.1.1 int PCU_Comm_Init (void)

Initializes the PCU library.

This function must be called by all MPI processes before calling any other PCU functions. MPI_Init or MPI_Init_thread should be called before this function.

3.1.1.2 int PCU_Comm_Free (void)

Frees all PCU library structures.

This function must be called by all MPI processes after all other calls to PCU, and before calling MPI_Finalize.

3.1.1.3 int PCU_Comm_Self (void)

Returns the communication rank of the calling thread.

when called from a non-threaded MPI process, this function is equivalent to MPI_Comm_rank(MPI_COMM_WORLD,rank).

When called from a thread inside PCU_Thrd_Run, the rank is unique to a thread in the whole MPI job. Ranks are consecutive from 0 to $pt-1$ for a program with p processes and t threads per process. Ranks are contiguous within a process, so that the t threads in process i are numbered from ti to $ti+t-1$.

3.1.1.4 int PCU_Comm_Peers (void)

Returns the number of threads in the program.

when called from a non-threaded MPI process, this function is equivalent to MPI_Comm_size(MPI_COMM_WORLD,size).

When called from a thread inside PCU_Thrd_Run, the size is pt, where p is the number of MPI processes and t is the number of threads per process, which is the nthreads argument passed to PCU_Thrd_Run.

3.1.1.5 void PCU_Comm_Begin (void)

Begins a PCU communication phase.

This function must be called by all threads in the MPI job at the beginning of each phase of communication. After calling this function, each thread may call functions like PCU_Comm_Pack or PCU_Comm_Write.

3.1.1.6 int PCU_Comm_Pack (int *to_rank,* const void * *data,* size_t *size*)

Packs data to be sent to *to_rank*.

This function appends the block of *size* bytes starting at *data* to the buffer being sent to *to_rank*. This function should be called after PCU_Comm_Start and before PCU_Comm_Send.

3.1.1.7 int PCU_Comm_Send (void)

Sends all buffers for this communication phase.

This function should be called by all threads in the MPI job after calls to PCU_Comm_Pack or PCU_Comm_Write and before calls to PCU_Comm_Receive or PCU_Comm_Read. All buffers from this thread are sent out and receiving may begin after this call.

3.1.1.8 bool PCU_Comm_Listen (void)

Tries to receive a buffer for this communication phase.

Either this function or PCU_Comm_Read should be called at least once by all threads during the communication phase, after PCU_Comm_Send is called. The result will be false if and only if the communication phase is over and there are no more buffers to receive. Otherwise, a buffer was received. Its contents are retrievable through PCU_Comm_Unpack, and its metadata through PCU_Comm_Sender and PCU_Comm_Received. Users should unpack all data from this buffer before calling this function again, because the previously received buffer is destroyed by the call.

3.1.1.9 int PCU_Comm_Sender (void)

Returns in * *from_rank* the sender of the current received buffer.

This function should be called after a successful PCU_Comm_Receive.

3.1.1.10 bool PCU_Comm_Unpacked (void)

Returns true if the current received buffer has been completely unpacked.

This function should be called after a successful PCU_Comm_Receive.

3.1.1.11 int PCU_Comm_Unpack (void * *data*, size_t *size*)

Unpacks a block of data from the current received buffer.

This function should be called after a successful PCU_Comm_Receive. *data* must point to a block of memory of at least *size* bytes, into which the next *size* bytes of the current received buffer will be written. Subsequent calls to this function will begin unpacking where this call left off, so that the entire received buffer can be unpacked by a sequence of calls to this function. It is up to the user to ensure that there remains *size* bytes to be unpacked, PCU_Comm_Unpacked can help with this.

3.1.1.12 void PCU_Add_Doubles (double * *p*, size_t *n*)

Performs an Allreduce sum of double arrays.

This function must be called by all ranks at the same time. *p* must point to an array of *n* doubles. After this call, p[i] will contain the sum of all p[i]'s given by each rank.

3.1.1.13 void PCU_Exscan_Ints (int * *p*, size_t *n*)

Performs an exclusive prefix sum of integer arrays.

This function must be called by all ranks at the same time. *p* must point to an array of *n* integers. After this call, p[i] will contain the sum of all p[i]'s given by ranks lower than the calling rank.

3.1.1.14 int PCU_Thrd_Run (int *nthreads*, PCU_Thrd_Func *function*, void ** *in_out*)

Runs *nthreads* instances of *function*, each in a thread.

This function will create (*nthreads* - 1) new pthreads and use these as well as the caller thread to run *function*. The argument passed to thread i is *in_out* [i], and the return value of thread i is then stored in *in_out* [i]. If in_out is NULL, all threads will receive NULL as their argument.

Currently, PCU requires that this call is collective and homogeneous. This means that all processes in an MPI job should call PCU_Thrd_Run at the same time, and they should all pass the same number for *nthreads*. MPI_Init_thread should have been called before this function.

Any calls to PCU_Comm functions from within one of these threads will have access to the hybrid communication interface. This means that ranks will be unique to a thread in the whole MPI job, and messages are sent and received between threads. Phases will be synchronized across all threads in the MPI job.

3.1.1.15 int PCU_Thrd_Self (void)

Returns the process-unique rank of the calling thread.

When called from a thread inside PCU_Thrd_Run, the resulting rank will be unique only within the same process. Ranks are contiguous integers from 0 to nthreads-1, with the thread that called PCU_Thrd_Run being assigned rank 0.

3.1.1.16 int PCU_Thrd_Peers (void)

Returns the number of threads running in the current process.

When called from a thread inside PCU_Thrd_Run, returns the number of threads running in this process, which is equivalent to the nthreads argument to PCU_Thrd_Run.

3.1.1.17 int PCU_Comm_Packed (int *to_rank*, size_t * *size*)

Returns in * *size* the number of bytes being sent to *to_rank*.

This function returns the size of the buffer being sent to *to_rank*. This function should be called after PCU_Comm_Start and before PCU_Comm_Send.

3.1.1.18 int PCU_Comm_Write (int *to_rank*, const void * *data*, size_t *size*)

Packs a message to be sent to *to_rank*.

This function packs a message into the buffer being sent to *to_rank*. Messages packed by this function can be received using the function PCU_Comm_Read. This function should be called after PCU_Comm_Start and before PCU_Comm_Send. If this function is used, PCU_Comm_Pack should not be used.

3.1.1.19 bool PCU_Comm_Read (int * *from_rank*, void ** *data*, size_t * *size*)

Receives a message for this communication phase.

This function tries to receive a message packed by PCU_Comm_Write. If a the communication phase is over and there are no more messages to receive, this function returns false. Otherwise, * *from_rank* will be the rank which sent the message, *data* will point to the start of the message data, and *size* will be the number of bytes of message data. If this function is used, PCU_Comm_Receive should not be used. Note that the address * *data* points into a PCU buffer, so it is strongly recommended that this data be read and not modified.

3.1.1.20 int PCU_Comm_Received (size_t * *size*)

Returns in * *size* the bytes in the current received buffer.

This function should be called after a successful PCU_Comm_Receive. The size returned will be the total received size regardless of how much unpacking has been done.

3.1.1.21 void* PCU_Comm_Extract (size_t *size*)

Extracts a block of data from the current received buffer.

This function should be called after a successful PCU_Comm_Receive. The next *size* bytes of the current received buffer are unpacked, and an internal pointer to that data is returned. The returned pointer must not be freed by the user.

Index