MPI/RT:
A Real-Time Message Passing Standard Specification and API

Arkady Kanevsky, MITRE Corp.

Anna Rounbehler, SKY Computers, Inc.

Anthony Skjellum, Miss. State Univ.

Overview of Lecture, Part I.

- Introduction (Skjellum)
- How to Reach us / Stay Plugged in to MPI and MPI/RT
- Highlights of MPI, MPI/RT, collective operations
- General Principles, Techniques, Issues
- MPI/RT Philosophy (Kanevsky)
- Real-time Paradigms

Overview of Lecture, Part II.

- MPI/RT Unifying Principles
  - channels
  - buffer pools
  - synchronous clocks
  - early binding
- Details of the Individual Models
  - Time-driven model
  - Event-driven models (low-level, high-level)
  - Priority-driven model

Overview of Lecture, Part III.

- Instrumentation (Rounbehler)
- Fault Tolerance (Rounbehler)
- Conclusions (Skjellum)

Being Plugged-in to MPI/RT

- Reflector:
  - subscribe to *mpi-realtime-request@mcs.anl.gov* (make your message have the line “subscribe”)
  - send comments to *mpi-realtime@mcs.anl.gov*
- Latest MPI/RT: ftp://aurora.cs.msstate.edu/pub/mpi/rt-2-26feb97.ps
  (many older drafts are kept there too for completeness)
  Latest MPI-2 specification, under http://www.mcs.anl.gov/mpi
- Reaching us

  A. Skjellum, 601-325-8435, *tony@cs.msstate.edu*
  A. Kanevsky, 508-271-5352, *arkady@mitre.org*
  A. Rounbehler, 508-250-1920, *anna@sky.com*
- New MPI/RT Page (to be released with the examples from this lecture)
  http://www.erc.msstate.edu/hpclab/mpirt

Recent and Upcoming Activities (You are Welcome to Attend!)

- MPI/RT meeting (Feb 20-21@ NRaD)
- MPI Forum meeting (March 5-7@ O'Hare)
- MPI/RT meeting (April 25-26@ O'Hare)
- MPI Forum meeting (April 23-25@ O'Hare)
- Combination of meetings to finalize first complete draft by end of May
- Public review period will happen thereafter
- MPI/RT will keep meeting after MPI-2 terminates in May, working independently thereafter...

General MPI History, Part I.

- MPI is a de facto Message Passing Standard
- MPI-1 completed 5/94, revised 6/95 & 5/97
- MPI-2 involves extensions to MPI-1
- MPI/RT is a part of MPI-2 process but not part of standard to be released 5/97 - real-time specification and features
- MPI/RT is not specifically a subset or superset of MPI-1 or MPI-2.

MPI Forum (a.k.a. MPI-F)

- Existence since January, 1993 based on 1992 organization
- Ad hoc body, open forum, regular meetings
- Many institutions (industry, government, academia) represented
- Has developed one complete, widely used standard in HPC, MPI-1 (5/94)
- Is working on MPI-2, and MPI/RT presently
- Does not have the official sanction of a standards body, but has been successful so far in HPC/cluster uses
- Has specific voting rules, approach for accepting/rejecting proposals, etc., with inclusiveness for new participants

MPI/RT Forum Subcommittee

- Constituted informally in March/April, 1995
- Government, University, Commercial participants
- Has “special status” in MPI process to allow time for concepts to mature properly
- Has broad scope in terms of augmenting and modifying MPI
- Has focused goal of providing standardized programming environment for RT
- Meets at all MPI Forum meetings, and in between
- Like rest of MPI, an open forum

MPI-1 Technical Highlights, I.

- Reliable, large data point-to-point message passing and collective operations
- Group-oriented communication with safety
- Point-to-point operations (2-sided) scoped to ranks contained within groups contained in turn within communicators
- Non-blocking and blocking, synchronous and buffered point-to-point modes
- Blocking collective operations scoped to groups
- Dual-group (bi-partite) point-to-point
- Mapping of processes to graphs or Cartesian topologies

MPI-1 Technical Highlights, II.

- Environmental Inquiry
- Profiling Interface
- Supports SPMD/MPMD, but static process model
- Manageable (128 fns MPI-1.1, 129 fns MPI-1.2)
- Small number of concepts to master
- Language Independent Specification (LIS)
- Language Specific Bindings (LSB) for C, Fortran77
- Requirement for a thread-safe API

MPI-2 Technical Highlights, I.

- Superset of MPI-1.2
- Dynamic Process Management (Spawn, Client-Server, …)
- 1-sided communication (high-level put/get model to “windows”)
- Collective I/O for flat file structures
- Collective operations on bi-partite (2 group) structures
- LSB for C, Fortran77, C++, and Fortran90

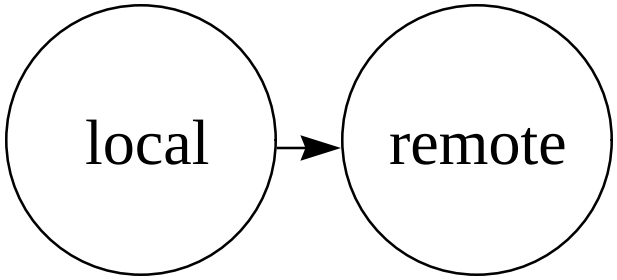
MPI-2 Technical Highlights, II.

- External Interface Specification for Tools
- Very large number functions
- Conceptually more ideas than MPI-1, but builds on it
- Thread compliance: Position on POSIX/Unix threads - MPI interactions

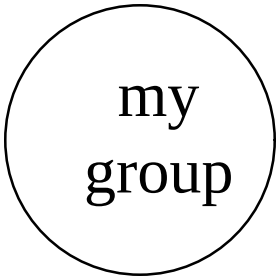
About Collective Ops.

- Collective operations supported by MPI:
  - clique versions (MPI-1)
  - bi-partite versions (MPI-2)
  - scoped to user-specified groups, not whole allocation
- Operations include:
  - MPI: gather, scatter, reduce, allreduce, alltoall, reducescatter, scan
  - MPI-2: alltoall generalization as well
  - MPI/RT: transpose-and-reshape to be proposed.

About Collective Ops., II.



bi-partite or intercommunicator operations (MPI-2, MPI/RT) MPI-1 only supports point to point between disjoint groups



clique or intracommunicator operations (all standards). Support for a variety of collective operations is offered. MPI/RT will extend this set semantically and may offer a transpose-reshape.

MPI/RT Technical Highlights

- Embraces MPI as base API and programming strategy, but modifies it for real-time
- Designed to support time-driven , priority-driven and event-driven paradigms
- Uses channels (point-to-point and collective) with quality of service
- Provides timeouts where appropriate
- Has profile stages to support cost-effective vendor development

Language-Independent Specification vs. Language-Specific Binding

- LSB for C uses form:
  - MPI_Function_name(args), or
  - MPIRT_Function_name(args)
    when describing function names
- LIS uses form
  - MPI_FUNCTION_NAME(args)
    where all are capital

Scope of MPI/RT Specification

- A full *draft* specification exists, under revision
- Timing requirements of implementation
- API for functions added or modified (to MPI)
- Channel, buffering, and admission test syntax/semantics
- Discussion of three real-time programming models
- Discussion of QoS specifications
- Fault tolerance / fault detection [early stages]
- Instrumentation
- Profiles
- LIS specification, LSB for C, F77, C++, F90

Scope of MPI/RT Functionality

- Channels
- Admission Tests
- Reliable, QOS-oriented data transfer and operations.
- Point-to-point operations on channels
- Collective operations on collective channels
- Admissions tests specified in a novel way to the system
- Both on-line and off-line mechanisms for establishing channels+QOS are to be entertained.
- Support for use of base MPI for non-real-time programming within the RT world.

Current Real-Time Development Philosophy

- Current application development process
  - Step 1 - Application developers become platform experts
    - Platform provides message passing API
  - Step 2 - Application developers are responsible for achieving desired application performance and real-time guarantees
    - It is up to application developers to get desired QoS and performance using platform specific API
- Inherently platform dependent - anti portable

MPI/RT Philosophy - Rationale

- Designers and implementors of a platform are better platform experts than users
  - Operating systems
  - Communication layers
  - Middleware with MPI/RT
- Platform and middleware designers know better what it takes to provide QoS

Rationale - QoS Guarantees

- Providing QoS guarantee is a difficult schedulability problem:
  - Involves scheduling different resources:
    - CPUs where send/receiver applications are running
    - Data buses
    - DMA engines
    - Memory
    - Network interface chips
    - Physical network (topology dependencies)
  - Different resources are scheduled using real-time paradigms and different scheduling algorithms

Rationale - MPI/RT Implementors Viewpoint

- Platform designers are in the best position to know all the resources involved in satisfying data transfer QoS guarantees and resource's interdependencies
- Platform designers can adjust run-time system parameters during initialization
- Different solutions for satisfying QoS guarantees for different architectures (hidden from the users)

MPI/RT Philosophy - Main Viewpoint

- Difficult for application developers to coordinate resource usage to guarantee application QoS
  - User solutions are not portable
- `MPI/RT cornerstone`

  Users provide detailed application info. - platform either satisfies user requests (dedicates resources) or states that it can not do it
  - Allows MPI/RT implementors to best utilize platform data transfer primitives: two-sided, one-sided, and zero-sided communications
  - MPI/RT encourages this viewpoint but does not require

MPI/RT Philosophy - Semantics

- MPI/RT provides both
  - Early binding
    - Supports implementation guaranteed user QoS - Channels
  - Late binding
    - Provides API to help application developers to satisfy user QoS requirements
- MPI/RT provides an API for timeouts, handlers, instrumentation

MPI/RT Paradigms

- Real-time paradigms - user perspective:
  - Time-Driven
  - Event Driven
  - Priority-Driven
  - Best Effort (Soft QoS)
- Each paradigm has specific QoS characteristics
- Each paradigm has unique additional functionality
- Common underlying principles

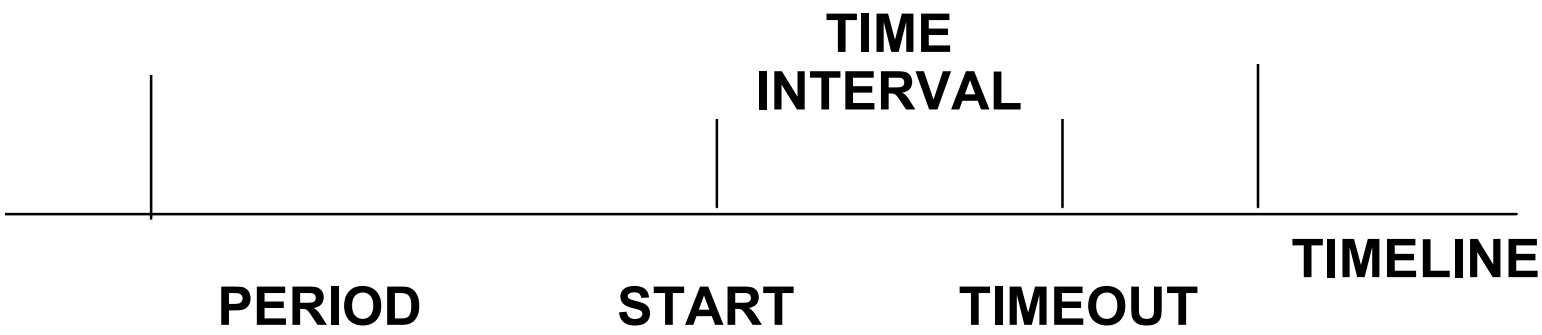
  `virtual channels` - for early binding
- Lower level QoS parameters that are resource specific are not part of the API:

  (bandwidth, jitter, latency)

  Presented as advice to implementors

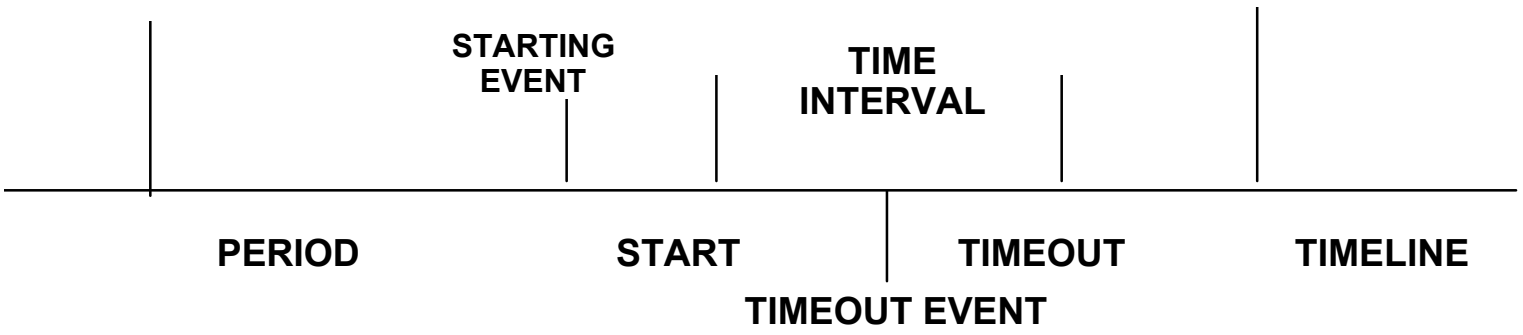
Time-Driven Real-Time Paradigm

- Application activities are satisfied within the requested time interval
- Quality of Service parameters:
  - Time interval for data transfer
  - Period of data transfer for periodic computations
  - Relationship between the time interval and the period



Event-Driven Real-Time Paradigm

- Application activities are bounded by time intervals, guarded by user specified events
- Quality of Service parameters:
  - Upper bound for an activation/deactivation time of an activity in response to an event:
- Two models: low level and high level event-driven paradigms
- Events can be local (for low-level and high-level event driven paradigms) or global (for high-level event-driven paradigm)



Priority-Driven Real-Time Paradigm

- User specified priority for data transfer
  - Early binding - Priority of a persistent channel
- Quality of Service parameters:
  - Integer priority of the persistent channel for message transmission

Time Specification

- MPIRT_TIME_OBJECT:
  - MPIRT_TIME_OBJECT_TYPE
    - ABSOLUTE
    - RELATIVE
  - MPIRT_TIME_OBJECT_TIME

    double precision floating point number

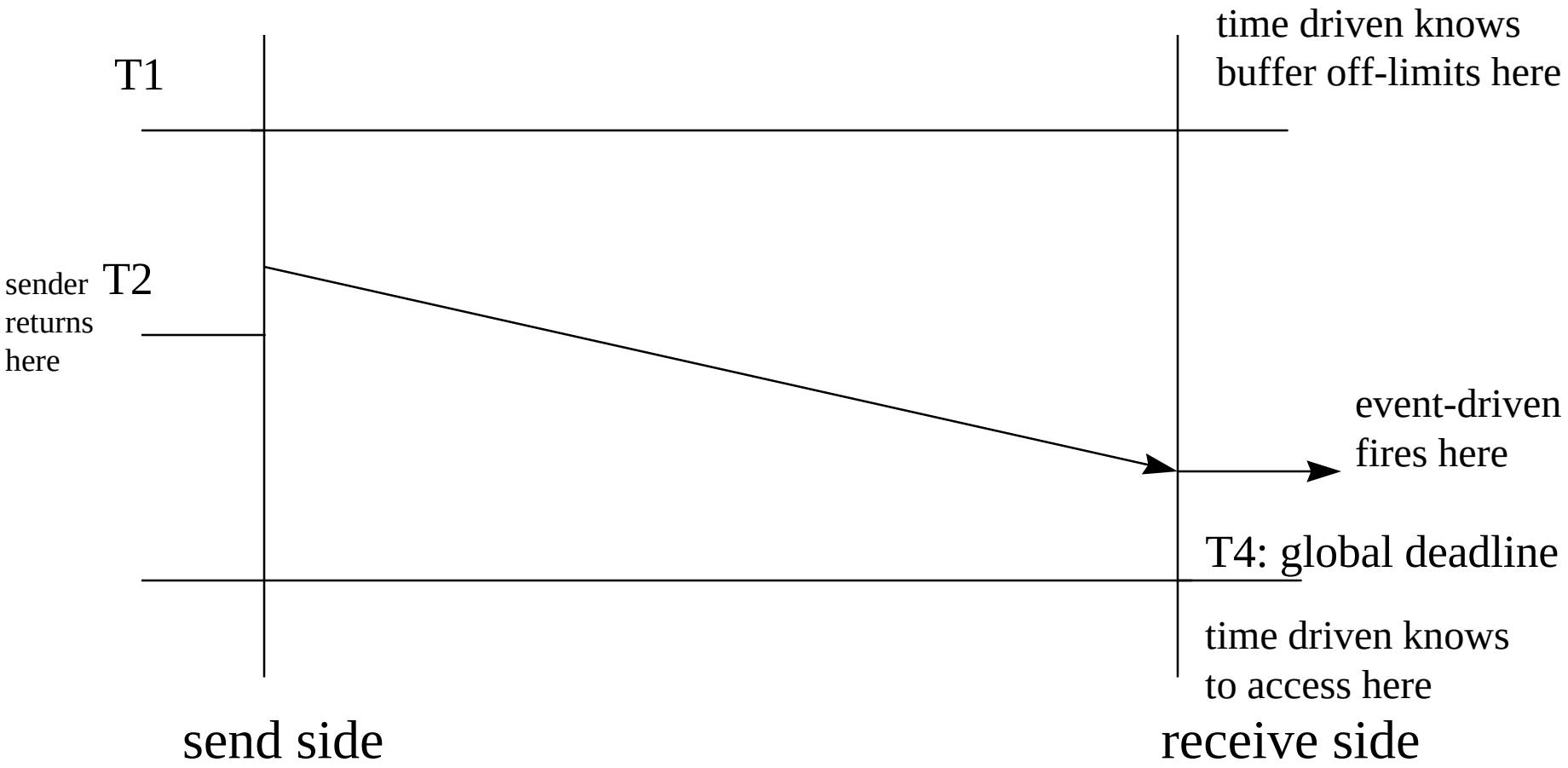
Real-Time Point-to-Point Channels

- Communicators are used to describe groups and set up real-time communication
- Sets of channels, each with QoS are requested, and collective admission tests are performed by the implementation to define channels
- Point-to-point channels are uni-directional, reliable, packet passing mechanisms with QoS, buffering schemes, and fixed end-points

Communication “Sidedness”

- MPI-1 supports 2-sided point-to-point.
- MPI/RT supports 2-sided, non-realtime modes of MPI-1.
- MPI/RT supports 1-sided point-to-point puts (and gets) on channels.
- MPI/RT supports 0-sided point-to-point transfers on time-based channels with periodic transfer.
- MPI-2 supports 1-sided window-oriented DSM. MPI/RT has not embraced this.

Comparing Time- and Event-Driven Transfers



Real-Time Collective Channels

- Collective channels use a fixed group of participants to define a collective operation as a single channel abstraction, offering the same type of quality of service as point-to-point channels
- For each collective operation, a collective channel with quality of service can be established. The MPI-type collective operation is executed on demand (or as scheduled) with the particular QOS.
- Collective channels are persistent and use “early binding” to negotiate once and use often a pattern of communication that is not binary, but rather involves the whole group of processes.

Buffers and Buffer Pools

- Buffer pools support abstraction that manages multiple buffers for an endpoint of a channel
- API specifies implementation buffer management strategy and application buffer management strategy
  - implementation: e.g., overwrite/non-overwrite of buffers when out of space
  - application: e.g., random access to buffers (not necessarily FIFO or LIFO)
- Concepts such as double buffering are supported
- Buffer sharing between end points is also allowed, with some caveats, supporting “data fusion”

Admission Tests

- Admission tests over sets of channels is supported as a collective construction process. This is asserted to be a new way to present the real-time requirements to the message passing subsystem.
- A collective operation makes a set of point-to-point channels
- A collective operation defines a single collective channel, with a specific operation.
- Systems like Tenet and U. of Michigan group use admission tests on single channels, with hop-by-hop tests. MPI/RT specifies the admission test at a higher level.

About Collective Ops., III.

- MPI/RT supports:
  - quality of service for clique and bi-partite collectives, in form of collective channels
  - early-binding semantics allows for use of poly-algorithms, and reduces per-use overheads
  - since existing MPI collectives (even MPI-2 extensions) have wrong abstraction level for transpose, we are developing specific transpose for RT, to be more easily used and optimized.

Syntactical Example, part I.

```
int MPIRT_Channels_init(
  MPIRT_Bufpool bufpools[], /* [IN] buffer pools for channels */
  int nchannels, /* [IN] number of channels to be defined locally */
  int flags[],  /* [IN] defines “send or receive” for each channel */
  int ranks[], /* [IN] defines “other end” of each channel */
  MPIRT_QOS qos[], /* [INOUT] specs QOS of each channel */
  MPIRT_QOS_Error_fn fns[], /* [IN] QOS violation error handlers */
  MPI_Comm comm, /* [IN] Communicator for group of processes */
  MPI_Request requests[], /* [OUT] handles for channels */
  int errors[], /* [OUT] error codes for each channel */
); /* C binding */
```

Syntactical Example, part II.

MPIRT_Bufpool bufpools[], /* [IN] buffer pools for channels */
int nchannels, /* [IN] number of channels to be defined locally */

- An array of length **nchannels** is specified, each with a Buffer pool. Each end of a channel has a buffer pool associated with it. Each buffer pool has at least one buffer.
- MPIRT_Bufpool is an *opaque object.*
- [IN] parameters are user-specified and are not changed by MPI/RT
- All arrays (xxx[]) in this call, are of dimension **nchannels**

  nchannels is different, in general, in each process involved, and depends on the channel topology being formed.

Syntactical Example, part III.

int flags[], /* [IN] defines “send or receive” for each channel */
int ranks[], /* [IN] defines “other end” of each channel */

- flags[] is an array of length **nchannels** of enumerated integer types, indicating endedness of channels
  - MPIRT_CHANNEL_SEND, or
  - MPIRT_CHANNEL_RECEIVE
- ranks[] specifies the MPI ranks of the corresponding process for each channel. This may be the rank of the process itself. (Ranks define names within a communicator in MPI, not Unix PIDs or other hardware names).

Syntactical Example, part IV.

MPIRT_QOS qos[], /* [INOUT] specs QOS of each channel */
MPIRT_QOS_Error_fn fns[], /* [IN] QOS violation error handlers */

- Each of these arrays corresponds entry for entry with the channel to be set up. There are **nchannels** such entries in a specific process.
- MPIRT_QOS qos[] is the array of quality of service requests, which may be subject to modification by the system, hence the [INOUT] designation.
- MPIRT_QOS_Error_fn fns[] are error handlers to be called if QOS is violated during channel use. It is possible to ignore such errors, using a special value for the function name.

Syntactical Example, part V.

MPI_Comm comm, /* [IN] Communicator for group of processes */

- The MPI communicator defines several key properties of the underlying communication
  - the number of process participants (“size” of group)
  - the actual participants, and their unique ranks (0…size-1)
  - the safe communication space for non-real-time messaging, including both point-to-point and collective
  - a container for holding attributes associated with the group of communicating processes, including callbacks for deletion and copying
- In normal MPI, a communicator appears in each communication call. In MPI/RT, these are supplemented with channels, which are represented by polymorphic request objects.

Syntactical Example, part VI.

MPI_Request requests[], /* [OUT] handles for channels */

- An array of length **nchannels** is specified, each with an opaque object (MPI_Request) handle on output. These handles are generic communication-specifiers in MPI and MPI/RT, and in this case denote the channels being managed.
- The scope of the created requests is dynamic, and they persist until explicitly destroyed singly or collectively
- Requests from this array are used to describe operations over specific point-to-point channels.

Syntactical Example, part VII.

int errors[], /* [OUT] error codes for each channel */

int error = MPI_Channels_init(…);

- An array of length **nchannels** is specified, each returning MPI_Success or an error code, depending on the individual success of creating the given channel
- The function also returns an error code, which indicates if there were problems with any of the channels. This is first consulted, and if an error has occurred, then the individual error codes can be queried for specifics.

MPI/RT Clock - Rationale

- Provide precise timing correctness
  - Synchronized clocks to support quality of service (QoS) timing requirements
  - Platform portability achieved - performance tuning using QoS parameters.
- Support fine grained instrumentation
  - delicate tuning for optimal performance
- Predictable timing behavior

What is an MPI/RT Clock?

- The MPI/RT clock is a synchronized clock.
  - The design of the synchronized clock is left to the implementor. MPI/RT Clocks may be local or global.
- Assume clocks are monotonically non-decreasing.
- MPI clock attributes (MPI_WTIME and MPI_TICK) use synchronized clocks

MPI/RT Clock Definition Supports

- Application specification of resource usage
  - Apply a timing specification for communication fabric and computation
  - Predictable behavior is achieved by managing these communication resources.

MPI/RT Clock Parameter Definitions

- Drift - for each synchronized clock a guaranteed upper bound on the error in the rate of clock. Drift is dimensionless.
- Skew - maximum bound on absolute value of difference between simultaneous values of synchronized clock in distinct nodes

MPI/RT Clock Parameters

- Accuracy - maximum bound on absolute value of the difference between simultaneous values of synchronized clock and ideal clock started at the synchronized clock's hypothetical start time.
- Access Time - maximum bound on time to execute a call of MPI_WTIME.

Clock Parameters are Attributes in MPI/RT

- Formats of MPI/RT clock attributes are POSIX 1.b compliant.
  - MPIRT_WTIME_DRIFT (DOUBLE)
  - MPIRT_WTIME_SKEW (DOUBLE)
  - MPIRT_WTIME_ACCURACY (DOUBLE)
  - MPIRT_WTIME_ACCESS_TIME (DOUBLE)

MPI/RT Clock Attributes Available to All Communicators

- MPI/RT clock attributes are cached to the MPI communicator **(MPI_COMM_WORLD).**
  - This MPI caching policy allows an application to attach arbitrary pieces of information to both intra and inter communicators
  - When MPI is initialized, the MPI/RT clock attributes are attached to **MPI_COMM_WORLD.**

MPI/RT Advice to Implementors for MPI Clock Attributes

- MPI_WTIME
  - MPI/RT always takes its value from the designated synchronized clock.
- MPI_WTICK
  - MPI/RT defines Resolution (Tick) as the time between two successive clock ticks and proposes a subtle modification to MPI_WTICK to support resolution of synchronized clocks.

MPI/RT Clocks are the Backbone of QoS

- QoS parameters use clock attributes to accurately represent:
  - activation time for event handler (event-driven paradigm)
  - time intervals, timeouts, periods, start time (time driven paradigm)

BREAK

15 minute break

Time-Driven Real-Time Paradigm

- User activities are done within the requested time interval
- MPI/RT introduces two new parameters for data transfer operations:
  - Starting time of the data transfer
  - Timeout of the data transfer
- By specifying time interval and application guarantees that data is ready in the send “message buffer” and receive “message buffer” is also ready
- Application guarantees that not system resource should be used on behalf of the time-driven message transfer outside the specified time interval

Time-Driven Real-Time Paradigm (continued)

- One-sided and Zero-sided data transfer modes
- Implicit MPI “ready” mode data transfer
- Eliminates the need for queues and system buffers
- No handshaking is required on behalf of the application
- Each end of the persistent channel specifies QoS error handler that is automatically invoked by the MPI/RT implementation of QoS of data transfer is not met
- Application does not need to know that message is transmitted successfully
  - By the end of the data transfer time interval either the message was transferred successfully or the error handler was invoked

Time-Driven Data Transfer Operations

MPIRT_START_TIME (request, index, start, timeout,period,fn)

IN	request	persistent channel request object (persistent request)
IN	index	index of the buffer for data transfer from the buffer pool associated with the channel or ignore for a single buffer version (integer)
IN	start	optional message start timing parameter (MPIRT_TIME_OBJECT)
IN	timeout	optional message stop timing parameter (MPIRT_TIME_OBJECT)
IN	period	optional periodic reinvocation (MPIRT_TIME_OBJECT)
IN	fn	optional function to call on QoS failure (MPIRT_QOS_ERROR_FN)

Applying MPI/RT Time-Driven Semantics

1. Establish buffer pools each with a single buffer for each channel
2. Establish time-driven pipeline using point-to-point channels according to the user timeline
3. Establish collective channels of the pipeline for corner-turn data transfer according to the user timeline
4. Start the pipeline
   (a) Create data in the send buffers prior to the channel start time
   (b) Use data in the receive buffer after channel timeout

Event-Driven Real-Time Paradigm

- Low Level - a mechanism for scheduling with QoS an application handler upon completion of a data transfer operations
  - Analogous to `hrecv,hsend` operations
- High level - a mechanism for scheduling with QoS any application activity:
  - Application activity:
    - MPI or MPI/RT data transfer
    - Application function
  - Triggering events:
    - System event
    - Communication event
    - Application event

Low Level Event-Driven Paradigm

- MPI/RT defines two events for completion of data transfers:
  - Local completion of data transfer (message buffer can be reused) `MPIRT_REQUEST_COMPLETE`
  - Global completion of data transfer (channel resources can be reused) `MPIRT_REQUEST_RELEASED`

Low Level Event-Driven Paradigm Request

MPIRT_REQUEST_POST_HANDLER (request, request_cond, cond_handler_fn, failure_fn, extra_state, qos)

INOUT	request	MPI request (handle) for channel
IN	request_cond	request condition (integer)
IN	cond_handler_fn	request condition handler (user defined function `MPIRT_function`)
IN	failure_fn	QoS failure handler (user defined function `MPIRT_QOS_ERROR_FN`)
IN	extra_state	user supplied state (choice)
IN	qos	event-driven QoS (handle)

Low Level Event-Driven Paradigm Request

- An application using event-driven MPI/RT will be able to specify intervals `guarded` by the specified events in order to bound the resource usage of communication and computation activities

High Level Event-Driven Paradigm - Rationale

- Current Application Control
  - Application waits on system events or user control messages
  - Application schedules a handler that is turn schedules application activities:
    - Functions
    - Processes
    - Threads/tasks
    - Data transfers
- MPI provides an interface for data flow control
- High level event-driven MPI/RT provides an interface for control flow

High Level Event -Driven Paradigm - Events

- Events can be both persistent and one time only
- Events are not necessarily local to a process or a node
- Event types:
  - System Events - MPI/RT implementation is aware of them
  - Communication Events - MPI (MPI/RT) events. Implementation is aware of them. For now , only two events associated with local and global data transfer completions are allowed, as defined in low level event-driven paradigm.
  - Application Events - MPI is not aware of them
- With each event class MPI/RT associates a name to handle periodic events. A name for a channel for communication events.

High Level Event-Driven Paradigm - Intervals

- Application events - has meaning only to the application
- MPI/RT is just a mechanism to match user activities with user requests to monitor events; a mechanism for the event delivery; and a mechanism to trigger/timeout activities
- Application specifies events that guard start and timeout of an activity
- Time interval for time-driven paradigm is a subset of the high-level event driven real-time paradigm interval guarded by events
  - Critical difference in the application's ability to schedule non-MPI activities and synchronization

High Level Event-Driven Paradigm - Activities

- An activity can use resources only within the event guarded interval
- Two activity types:
  - Functions (computation)
  - Data transfers over a persistent channel (communication)
- Event registration:
  - For function (computation)
  - For channel (communication)
  - Application generated events

High Level Event-Driven Paradigm - Operations

- `MPIRT_GUARDED_FUNCTION` and `MPIRT_GUARDED_CHANNEL_USE` that provide computation and communication activities bounded by intervals specified by events,
- `MPIRT_REGISTER_EVENT_NAMES` and `MPIRT_REGISTER_GENERATED_EVENT NAMES` that allow creation of event-response lists for processes,
- `MPIRT_GENERATE_EVENT` that allows an application to generate user events

MPI/RT Priority-Driven Paradigm - Rationale

- Only message priorities have a portable interface.
- Process priorities are discussed in Advice to Implementors.

MPI/RT Message Priorities

- The QoS message priority parameter is an integer priority of a persistent channel for message transmission.
  - Message priorities impose an ordering on data transfer operations.

MPI/RT Message Priority Future Discussions

- Should message communication have two priority classes (high and low) ?
- Should preemption granularity be defined in the QoS message priority parameters?
- Should a second level priority for the whole channel be defined in the QoS message priority parameters.

MPI/RT Priorities - Advice to Implementors

- QoS guarantees are defined at the application layer. Prioritized messages add a level of complexity to any communication layers managing data transfers.
- Process priorities are left to the implementor.

MPI/RT Instrumentation

- MPI/RT instruments provide monitoring and the ability to output metrics, Metrics contain information about MPI/RT communication and related resources.
- MPI/RT allows monitoring of user-specified information. Monitoring may occur over channels (single or collective), over a communicator or layered libraries.

How MPI/RT Manages Monitoring

- Create - creates a monitor handle, initializes parameters and returns the monitor handle
- Delete - destroys the monitor handle
- Reset - resets parameters
- MPI/RT also provides an interface to external monitoring tools.

How MPI/RT Controls Monitoring

- A monitor is started and stopped using the monitor handle.
  - For layered libraries, an additional level of granularity may be required to control monitoring over an interval of statistics.

Monitoring Provides Useful Information

- Metrics that help the user determine QoS parameters for an application request.
  - number of timeouts (collective operations, channel operations)
  - frequency and period of timeouts
- Metrics for performance analysis:
  - overhead from the operating system, overhead from MPI/RT and overhead from other services

MPI/RT Fault Tolerance Rationale

- Early detection of MPI/RT faults
  - Identify fault, and handle fault before significant communication degradation occurs.
- Late response indicators of MPI/RT faults
  - Fault is reported as an error. Communication has degraded. Fault recovery utilizes redundant resources and/or reconfigures.
  - Future functionality in MPI/RT

MPI/RT Fault Tolerance Semantics

- Timeouts added to select MPI-1.0 and MPI-2.0 functions.
- MPI/RT provides user specified handler.
- When QoS fails, user specified handlers can be invoked.
- Channels can have specified early binded error handlers.

MPI/RT Errors

- If a hard QoS guarantee is not met, then an error is reported. Other QoS errors need clarification.
  - How is an error defined for a prioritized operation?
  - How is an error defined for non-hard real-time QoS?

MPI/RT Fault Tolerance Future Discussions

- Fault Tolerance API Design Issues
  - How to provide redundancy and replications?
    - Can existing MPI and MPI/RT APIs provide some of this functionality?
- MPI/RT Design
  - Provide extensions to MPI/RT opaque objects for fault tolerance.

MPI/RT: Summary

- MPI/RT is a maturing specification
- MPI/RT will have a prototype/ref. impl.
- Specification finalizing by early 1997
- Implementation underway in early 1997
- Early implementations to get experience, generate feedback for RT
- MPI/RT to offer a comprehensive program notation for real-time messaging and parallel programming with QoS specs.

DARPA-sponsored Prototyping Efforts at MSU

- MSU/MPIRT to be full MPI-1 plus RT, with QoS and porting based on middleware architecture
- 1Q 1997 - early prototypes (on Mercury, Avalon); single QoS, simple demonstrations
- 2Q 1997 - unified design for MSU/MPIRT
- 2Q 1997 - implementation of prototype in earnest
- 3Q 1997 - full prototype (Mercury, Avalon, plus any other early-adopter ports)
- 4Q 1997 - demonstrations using MPI/RT by MSU and early adopters
- 1Q 1998 - revisions to architecture and implementation given feedback and RT evolution
- 2Q 1998 - addition of “resource constrained” MPI-2 features.

DARPA-sponsored Prototyping Efforts at MSU, II.

- Additional system ports could be accomplished (Mercury/Avalon in house)
- Need access to systems that have real-time middleware from vendors
- We will be merging MPI/RT and Maruti (time-driven RTOS) in 1997, as a further example of an experiment (MSU/UMD)
- Early adopters could work with MSU prototype as it progresses, and offer input / feedback

Conclusions, I.

- MPI-1 was a success in SPMD technical computing
- MPI/RT builds on that success, while adopting ideas from MPI-2, and incorporating ideas not finally accepted to MPI-2
- MPI/RT is not tied to all the potential “dark corners” of MPI-2
- Profiling rules allow for incremental support of MPI-2 within MPI/RT context.

Conclusions, II.

- MPI/RT supports three models of real-time as of now:
  - time-driven
  - event-driven
  - priority-driven
- Key unifying concepts drive MPI/RT:
  - channels
  - buffer pools
  - synchronized clocks
  - early binding

Conclusions, III.

- Specific syntax, semantics, and abstractions are offered to provide these individual (and ultimately joint) RT programming models.
- An open process defines, refines, and extends MPI/RT
  - attend MPI Forum
  - attend our extra meetings (movable, but often at MITRE)
  - contribute to the mail reflectors
  - offer examples to be converted to MPI/RT
  - try out forthcoming implementations
  - try implementing MPI/RT on your own specialized system