Introspection for Exascale Communication: Needs, Opportunities, Tools and Interfaces

Martin Schulz
Livermore National Laboratory

ExaComm Workshop @ ISC 2015 • Frankfurt / Main, July, 2015

http://scalability.llnl.gov/

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
Networks are Getting more Complex

- More complex network topologies
  - From 3D torus to 5D torus
  - Complex fat trees, dragonfly networks
    - Hard to visualize

- Adaptive Routing
  - Based on network conditions
    - Impact of (performance) reproducibility

- Consequences for Users
  - Task placements are becoming more important
    - Even if the vendor tells you otherwise
  - Higher variations in execution time make performance analysis hard
  - Pinpointing contention in the network is almost impossible
**Example: Task Placement**

- **Especially for more complex topologies**
  - Interactions with communication topology non-trivial
  - Node placement can have huge impact on performance

- **Example: FPMD Code at LLNL**
  - Target machine: BG/L (3D torus)
  - Full system run at 64K nodes / 128K cores

![Diagram showing task placement and performance metrics:]
- xyz (default) 39.5 TF
- quadpartite 64.7 TF
- 8x8x8 38.2 TF
- 64% speedup!
Plasma/Laser Interaction Code

- **3D Communication structure**
  - Stencil communication for Z-coupling
  - Pencil shaped FFTs in X and Y

- Need for tools to enable such optimizations
  - Scripting tool to describe task placements in tori: RUBIK
  - Network monitoring to detect link contention

Communication Improvement: 50 -> 201 MB/s
Example: “Cross-run” of Variability

- Average messaging rates for batch jobs running the same laser-plasma interaction code
  - Mira: BG/Q (5D torus), Intrepid: BG/P (3D torus)
  - Hopper: Cray XE6 (3D torus)

Total number of bytes sent on the network
Time spent sending the messages
“In Job” Variability

- Hopper system @ NERSC
  - Cray XE6 (3D torus)
- Four long running jobs of the same application
  - Different dates
Sources of Variability

- **Operating system noise (OS jitter)**
  - OS daemons running on some cores of each node

- **Placement/location of the allocated nodes for the job (Allocation shape)**
  - Especially problematic if …
    - … not in the hand of the user
    - … partition shapes are not known ahead of time

- **Contention for shared resources (Inter-job contention)**
  - Sharing network links with other jobs
4x8x8-shaped pF3D job (blue)

April: 11th: Green = MILC job

April: 16th (a): 25% higher message rate

April: 16th (b): red = LSMS job
27.8% higher message rate
Effect of MILC on Laser Code

There goes the neighborhood: performance degradation due to nearby jobs. Bhatele et al., SC13

avg = 58 MB/s
σ = 9.12 MB/s

avg = 66 MB/s
σ = 8.69 MB/s
Need for Introspection

- **Users need feedback on performance**
  - How are their applications performing?
  - Why are we seeing particular behaviors?
    - Is it caused by the application or the system?

- **We need efficient / intuitive ways to present this information**
  - Aggregated profiles often don’t tell the whole story
    - Need for performance visualization

- **Questions**
  - What can we measure today?
  - How can we present it to the user?
  - What would we like to measure?
  - How can we access measured data?
What we Can Measure/Analyze Now?

- Many networks have performance counters
  - Packet counts, token counts, some access to switch counters
  - But:
    - Often hard to get to, especially from user level
    - Only accumulative (not per message)
    - Can be expensive (network perturbation)

- Application level information
  - MPI interceptors and profilers
  - Useful for correlation to source code

- Boxfish: A Tool For Network Visualization
  - Flexible plugins for different topologies
  - Mapping of measurements to topology data
The Boxfish GUI

- Input data from multiple measurements
- Drag selection to map data to visualization
- Available Visualization domains

Selected visualization: 3D Torus

Choice of mappings: Present data on nodes or links?

Available Visualization domains:
- 3D Torus - 3D View
- 3D Torus - 2D View
- 3D Patch View
- Table

Input data from multiple measurements

Available Visualization domains:
- 3D Torus - 3D View
- 3D Torus - 2D View
- 3D Patch View
- Table

Choice of mappings: Present data on nodes or links?

http://scalability.llnl.gov/
- **Visual study and contention in networks**
  - Map link counters to 3D torus on BG/P
  - Example: AMG2000

- **Processor Neighborhoods**
  - Visualize job allocations around own job
  - Understand performance impact

- **Visualization of fat tree networks**
  - Dominant topology in cluster systems
  - Study impact of routing algorithms

- **Power optimization for networks**
  - Turn off unused links in torus networks
  - Boxfish used to show simulation data
A First Look at a Dragonfly Network
Dragonfly Performance Prediction Model

- **Input:**
  - Network graph of dragonfly routers
  - Application communication graph
  - Job placement
  - Routing strategy

- **Output:**
  - The steady-state traffic distribution on all network links, which is representative of the network throughput

- **Routing options**
  - Static Direct (SD): even split among shortest paths
  - Static Indirect (SI): each packet sent on random long path
  - Adaptive Direct (AD): pick least congested shortest path
  - Adaptive Indirect (AI): pick least congested long path
  - Adaptive Hybrid (AH): adaptively pick between AD and AH

- **Different job placement options**
Unstructured Mesh Benchmark

- Adaptive similar to static
- Node placement matters
- Indirect methods slightly worse
  - But less spread
- Hybrid method slightly worse
Structured Grid Benchmark

- Direct methods more variable
  - Indirect methods take skew away
- Indirect methods placement insensitive
- Adaptive routing slightly better
- Hybrid routing slightly worse
What is Missing? – A Wishlist

- **Dragonfly data so far from simulation**
  - Would need explicit counters for measurements
  - Need access for whole machine counters
    - Links are shared resources across the whole system
    - Influence by system parts on which a job doesn’t execute
  - Need data along the message path

- **Need to distinguish traffic by job**
  - Coloring of packets
  - Separate counters

- **Ideally: message tracking**
  - Color individual messages
  - Count impact of other messages on a colored messages
  - Would allow …
    - … precise location of contention spots
    - … clean correlation with actual application messages
How to Get to The Information?

- **State of the Art**
  - Many vendor specific solutions
    - In rare cases through PAPI, but even then hardware specific
    - Prohibits portable tools
  - Only light at the end of the tunnel: PMPI
    - MPI Profiling Interface for application level interception
    - Part of MPI since MPI 1.0

- **Need standard interfaces to enable portable tools**
  - Should be part of an existing (pseudo) standard
    - In a language standard would be best
    - PAPI would also be helpful
The MPI Tool Information Interface

- **Short name: the MPI_T interface**
  - Provide introspection into the MPI layer
  - Export internal performance information
  - Enable standardized access for tools
  - Allow for implementation specific information

- **Included in MPI 3.0 as part of a new tools chapter**
  - Replaces the existing MPI profiling interface chapter
  - PMPI included as a new subchapter (unchanged)

- **API is a set of routines with the prefix MPI_T**
  - Mostly encapsulated in the new chapter
  - Same general restrictions/requirements as any MPI routine
  - Ability to use MPI_T before MPI_Init / after MPI_Finalize
General Approach

- **Basic concept: a set of named variables**
  - Set of variables and naming left to the MPI implementation
  - MPI_T provides query functions to detect variables
  - Semantics provided as clear text
  - Routines to read and write values of these variables

- **Split into performance and control variables**
  - Performance: internal performance data
    - “Software counters for MPI”
  - Control: Configuration information / environment variables

**Performance Variables**
- Number of packets sent
- Time spent blocking
- Memory allocated

**Control Variables**
- Parameters like Eager Limit
- Startup control
- Buffer sizes and management
Basic Functionality for Variables

**Performance Variables**

- Allocate Session
- Allocate Handle
- Reset/Write Variable
- Start Variable
- Stop Variable
- Read/Readreset Variable
- Free Handle
- Free Session

**Control Variables**

- Allocate Handle
- **Read/Write Variable**
  - Scoping to define to which ranks a configuration change must be applied to
- Free Handle
MPI_T Query Approach

- Which variables exist can differ between ...
  - ... MPI implementations
  - ... compilations of the MPI library (debug vs. production version)
  - ... executions of the same application/MPI library
  - Libraries can decide not to provide any variables

- Example: Performance Variables:
  - Users need to query existing variables
  - MPI provides names, descriptions, and metadata

User Requesting a Performance Variable from MPI_T

MPI Implementation with MPI_T
Making Use of the MPI_T Interface

- **Typical Chicken & Egg problem**
  - Tools need the interface to be available
  - Library developers need to see the need
  - Test cases during development

- **Two initial tools for MPI_T**
  - Varlist: list available options (lists existing variables)
  - Gyan: profiling using MPI_T Variables (basic end-to-end profiler)

http://scalability.llnl.gov/

- **Other early approaches**
  - Use of MPI_T control variables for auto-tuning (part of Periscope)
  - Discussions for a PAPI integration
Varlist: Querying Existing Variables

- **Simple tool to read all variables offered**
  - Extract descriptions and metadata
  - Read default values

<table>
<thead>
<tr>
<th>OPTIONS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>List only Control Variables</td>
</tr>
<tr>
<td>-p</td>
<td>List only Performance Variables</td>
</tr>
<tr>
<td>-v &lt;VL&gt;</td>
<td>List up to verbosity level=1-9</td>
</tr>
<tr>
<td>-l</td>
<td>Long list with all information, including descriptions</td>
</tr>
<tr>
<td>-m</td>
<td>Do not call MPI_Init before listing variables</td>
</tr>
</tbody>
</table>

- **Use cases**
  - Gather information about which variables are available
  - Documentation of runtime environment
### Example: Control Variables on Open MPI

Control Variables

---

Found 1026 control variables
Found 1026 control variables with verbosity <= D/A-9

<table>
<thead>
<tr>
<th>Variable</th>
<th>VRB</th>
<th>Type</th>
<th>Bind</th>
<th>Scope</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi_dtt_unpack_debug</td>
<td>U/A-3</td>
<td>INT</td>
<td>n/a</td>
<td>LOCAL</td>
<td>false</td>
</tr>
<tr>
<td>mpi_dtt_pack_debug</td>
<td>U/A-3</td>
<td>INT</td>
<td>n/a</td>
<td>LOCAL</td>
<td>false</td>
</tr>
<tr>
<td>mpi_dtt_position_debug</td>
<td>U/A-3</td>
<td>INT</td>
<td>n/a</td>
<td>LOCAL</td>
<td>false</td>
</tr>
<tr>
<td>mpi_dtt_copy_debug</td>
<td>U/A-3</td>
<td>INT</td>
<td>n/a</td>
<td>LOCAL</td>
<td>false</td>
</tr>
<tr>
<td>dss_buffer_type</td>
<td>D/D-8</td>
<td>INT</td>
<td>n/a</td>
<td>ALL</td>
<td>described</td>
</tr>
<tr>
<td>dss_buffer_initial_size</td>
<td>D/D-8</td>
<td>INT</td>
<td>n/a</td>
<td>ALL</td>
<td>128</td>
</tr>
<tr>
<td>dss_buffer_threshold_size</td>
<td>D/D-8</td>
<td>INT</td>
<td>n/a</td>
<td>ALL</td>
<td>1024</td>
</tr>
<tr>
<td>event</td>
<td>U/D-2</td>
<td>CHAR</td>
<td>n/a</td>
<td>ALL</td>
<td></td>
</tr>
<tr>
<td>event_base_verbose</td>
<td>D/D-8</td>
<td>INT</td>
<td>n/a</td>
<td>LOCAL</td>
<td>0</td>
</tr>
<tr>
<td>event_libevent2021_event_include</td>
<td>U/A-3</td>
<td>CHAR</td>
<td>n/a</td>
<td>LOCAL</td>
<td>poll</td>
</tr>
<tr>
<td>opal_event_include</td>
<td>U/A-3</td>
<td>CHAR</td>
<td>n/a</td>
<td>LOCAL</td>
<td>poll</td>
</tr>
<tr>
<td>event_libevent2021_major_version</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>UNKNOWN</td>
<td>1</td>
</tr>
<tr>
<td>event_libevent2021_minor_version</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>UNKNOWN</td>
<td>9</td>
</tr>
<tr>
<td>event_libevent2021_release_version</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>UNKNOWN</td>
<td>0</td>
</tr>
<tr>
<td>mpi_param_check</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>true</td>
</tr>
<tr>
<td>mpi_yield_when_idle</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>false</td>
</tr>
<tr>
<td>mpi_event_tick_rate</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>-1</td>
</tr>
<tr>
<td>mpi_show_handle_leaks</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>true</td>
</tr>
<tr>
<td>mpi_no_free_handles</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>false</td>
</tr>
<tr>
<td>mpi_show_mpi_alloc_mem_leaks</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>0</td>
</tr>
<tr>
<td>mpi_show_mca_params</td>
<td>D/A-9</td>
<td>CHAR</td>
<td>n/a</td>
<td>READONLY</td>
<td></td>
</tr>
<tr>
<td>mpi_show_mca_params_file</td>
<td>D/A-9</td>
<td>CHAR</td>
<td>n/a</td>
<td>READONLY</td>
<td></td>
</tr>
<tr>
<td>mpi_abort_delay</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>0</td>
</tr>
<tr>
<td>mpi_abort_print_stack</td>
<td>D/A-9</td>
<td>INT</td>
<td>n/a</td>
<td>READONLY</td>
<td>true</td>
</tr>
</tbody>
</table>

...
### Example: Performance Variables in MVAPICH

<table>
<thead>
<tr>
<th>Variable</th>
<th>VRB</th>
<th>Class</th>
<th>Type</th>
<th>Bind</th>
<th>R/O</th>
<th>CNT</th>
<th>ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>posted_recvq_length</td>
<td>U/D-2</td>
<td>LEVEL</td>
<td>UINT</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>unexpected_recvq_length</td>
<td>U/D-2</td>
<td>LEVEL</td>
<td>UINT</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>posted_recvq_match_attempts</td>
<td>U/D-2</td>
<td>COUNTER</td>
<td>UNKNOW</td>
<td>n/a</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>unexpected_recvq_match_attempts</td>
<td>U/D-2</td>
<td>COUNTER</td>
<td>UNKNOW</td>
<td>n/a</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>time_failed_matching_postedq</td>
<td>U/D-2</td>
<td>TIMER</td>
<td>DOUBLE</td>
<td>n/a</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>time_matching_unexpecteddq</td>
<td>U/D-2</td>
<td>TIMER</td>
<td>DOUBLE</td>
<td>n/a</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>unexpected_recvq_buffer_size</td>
<td>U/D-2</td>
<td>LEVEL</td>
<td>UNKNOW</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mem_allocated</td>
<td>U/B-1</td>
<td>LEVEL</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mem_allocated</td>
<td>U/B-1</td>
<td>HIGHWAT</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_progress_poll_count</td>
<td>D/B-7</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>coll_bcast_binomial</td>
<td>U/B-1</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>coll_bcast_scatter_doubling_allgather</td>
<td>U/B-1</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>coll_bcast_scatter_ring_allgather</td>
<td>U/B-1</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_num_2level_comm_requests</td>
<td>U/D-2</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_num_2level_comm_success</td>
<td>U/D-2</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_num_shmem_coll_calls</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_binomial</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_scatter_doubling_allgather</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_scatter_ring_allgather</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_scatter_ring_allgather_shm</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_shmem</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_knomial_internode</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_knomial_intranode</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_mcast_internode</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>mv2_coll_bcast_pipeline</td>
<td>T/B-4</td>
<td>COUNTER</td>
<td>ULLONG</td>
<td>n/a</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>
Basic tool to profile MPI_T information
- Calipers for whole program execution
- Predefined counters defined through environment variable
  - Identify with Varlist
- Alternatively: monitor all available variables

Implemented as a PMPI tool
- Transparent preloading
- Data collected and printed at the end of execution

Following experiments
- LLNL TLCC cluster (Dual socket Intel Sandybridge nodes and IB)
- MVAPICH2-2.0a
### Performance profiling for the complete MPI job:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem_allocated</td>
<td>LEVEL</td>
<td>2119601</td>
<td>2119601</td>
<td>2119601.00</td>
</tr>
<tr>
<td>mem_allocated</td>
<td>HIGHWAT</td>
<td>17488028</td>
<td>17488028</td>
<td>17488028.00</td>
</tr>
<tr>
<td>mv2_reg_cache_hits</td>
<td>COUNTER</td>
<td>1205</td>
<td>1205</td>
<td>1205.00</td>
</tr>
<tr>
<td>mv2_reg_cache_misses</td>
<td>COUNTER</td>
<td>5</td>
<td>5</td>
<td>5.00</td>
</tr>
<tr>
<td>mv2_vbuf_allocated</td>
<td>COUNTER</td>
<td>384</td>
<td>384</td>
<td>384.00</td>
</tr>
<tr>
<td>mv2_vbuf_freed</td>
<td>COUNTER</td>
<td>160085</td>
<td>160085</td>
<td>160085.00</td>
</tr>
<tr>
<td>mv2_vbuf_available</td>
<td>COUNTER</td>
<td>283</td>
<td>283</td>
<td>283.00</td>
</tr>
<tr>
<td>mv2_ud_vbuf_n_allocated</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_ud_vuf_freed</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_ud_vbuf_available</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_progress_poll_count</td>
<td>COUNTER</td>
<td>753207</td>
<td>753207</td>
<td>753207.00</td>
</tr>
<tr>
<td>mv2_rdma_ud_retransmit_count</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>coll_bcast_binomial</td>
<td>COUNTER</td>
<td>462</td>
<td>462</td>
<td>462.00</td>
</tr>
<tr>
<td>coll_bcast_scatter_doubling_allgather</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>coll_bcast_scatter_ring_allgather</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_num_2level_comm_requests</td>
<td>COUNTER</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>mv2_num_2level_comm_success</td>
<td>COUNTER</td>
<td>1</td>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>mv2_num_shmem_coll_calls</td>
<td>COUNTER</td>
<td>21276</td>
<td>21276</td>
<td>21276.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_binomial</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_scatter_doubling_allgather</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_scatter_ring_allgather</td>
<td>COUNTER</td>
<td>220</td>
<td>220</td>
<td>220.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_scatter_ring_allgather_shm</td>
<td>COUNTER</td>
<td>110</td>
<td>110</td>
<td>110.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_shmem</td>
<td>COUNTER</td>
<td>3520</td>
<td>3520</td>
<td>3520.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_knomial_internode</td>
<td>COUNTER</td>
<td>1760</td>
<td>1760</td>
<td>1760.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_knomial_intranode</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_mcast_internode</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_coll_bcast_pipeline</td>
<td>COUNTER</td>
<td>110</td>
<td>110</td>
<td>110.00</td>
</tr>
<tr>
<td>mv2_ivb_channel_ctrl_packet_count</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_ivb_channel_out_of_order_packet_count</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_rdmaf_ctrl_packet_count</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_rdmafp_out_of_order_packet_count</td>
<td>COUNTER</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>mv2_rdmafp_exact_recv_count</td>
<td>COUNTER</td>
<td>73245</td>
<td>73245</td>
<td>73245.00</td>
</tr>
<tr>
<td>mv2_rdmafp_sendconn_accepted</td>
<td>COUNTER</td>
<td>4</td>
<td>4</td>
<td>4.00</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>posted_recvq_match</td>
<td>Counts how many times the queue for receiving expected messages is read.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unexpected_recvq_match</td>
<td>Counts how many times the queue for receiving unexpected messages is read.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>progress_poll_count</td>
<td>Counts how many times the application polls the progress of a communication. The higher the value, the more CPU time is spent in polling.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mem_allocated_level</td>
<td>Gives the instantaneous memory usage by the library in bytes.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mem_allocated_highwater</td>
<td>Gives the maximum number of bytes ever allocated by the MPI library at a given process for the duration of the application.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coll_bcast_binom</td>
<td>Counts how many of the MPI broadcast collective calls use the Binomial algorithm during an application run.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>num_shmem_coll</td>
<td>Counts how many of the collective communication calls are using shared memory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coll_bcast_shmem</td>
<td>Counts how many of the MPI broadcast communication calls are shared memory based collectives.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case Study: Receive Queues for NAS BT

![Bar chart showing number of operations in hundreds for different queue sizes and types.](image-url)
Case Study: Receive Queues for NEK5000
Case Study: Memory Consumptions for NEK5000
What Has to Happen Next?

- **Adoption across MPI implementation**
  - Availability of wide range of counters
  - Support of advanced mechanisms like configuration settings
  - Very good start by MPICH, MVAPICH and Open MPI
  - Integration of low level / device specific information

- **Tool support for counters**
  - Gyan is only a first step
    - Need more fine grained measurements
    - Need to look into correlations
  - Integration into other tools could be accelerated using a PAPI module

- **Common terminology for counters**
  - We won’t get standardization, but we may get close
  - Would help consistency across tools
Other Tool Interfaces for Communication Analysis

- **More News from the MPI Tools Group ... (planned)**
  - Tracing/Notification extensions to MPI_T
    - Ability to react to individual events
  - Attempt to revamp the PMPI interface
    - Callbacks instead of linker tricks
    - Ability to stack tools with clean interoperability

- **Need similar interfaces for other programming models**
  - Low level communication libraries
  - Higher level models with own communication (Charm++, Legion)
  - Low level communication libraries
  - Hardware support (?)
    - Must be baked in from the beginning
Conclusions

- **Modern networks pose challenges for the end user**
  - Node placement, variability, need to pinpoint contention
  - Will be even worse at exascale
  - Need introspection into network performance

- **Initial approaches**
  - Topology aware visualization of existing counters
  - Simulation infrastructures

- **Need more introspection into actual networks and their performance**

- **Standardized interfaces are key**
  - Enable tool portability
  - Enable clean correlation
  - Should be part of any message-based library or programming model
Main topics

- Performance analysis tools and optimization
- Correctness and debugging (incl. STAT, AutomaDeD, MUST)
- Tool infrastructures (incl. PnMPI, GREMLINs)
- Power-aware and power-limited computing (incl. Adagio)
- Resilience and Checkpoint/Restart (incl. SCR)

Funding sources involved in presented work:
Conclusions

- **Modern networks pose challenges for the end user**
  - Node placement, variability, need to pinpoint contention
  - Will be even worse at exascale
  - Need introspection into network performance

- **Initial approaches**
  - Topology aware visualization of existing counters
  - Simulation infrastructures

- **Need more introspection into actual networks and their performance**

- **Standardized interfaces are key**
  - Enable tool portability
  - Enable clean correlation
  - Should be part of any message-based library or programming model

http://scalability.llnl.gov/