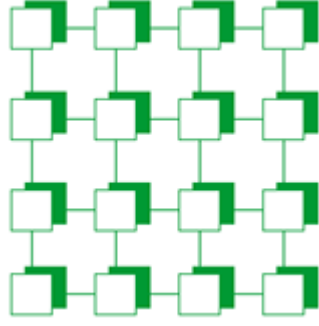


The MPI/SX Collectives Verification Library

Joachim Worringen, Jesper Larsson Träff
C&C Research Labs, NEC Europe Ltd
St. Augustin, Germany
`mpi@ccrl-necce.de`



ParCo 2005
Malaga, 12-16 September 2005
Mini-Symposium N4

NEC

Empowered by Innovation

Preface

- For nearly 10 years, **NEC C&C Research Labs** develop and maintain the MPI libraries for all NEC HPC platforms
- Full MPI-2 for 5 years, running on
 - SX series of parallel vector machines
 - Earth Simulator
 - Asama IA-64 32-way SMP machines
 - IA-32/Opteron/EMT64 based clusters
- Supported interconnects for non-SX platforms are Ethernet, Myrinet, Quadrics, Infiniband, ...
- Single source tree for all platforms and interconnects
 - Features like verification capabilities apply to **all** variants of the library

Motivation

- Parameter checking and related error handling in MPI often gets neglected by an implementation:
 - For *good* reasons: avoid useless overhead of validating a correct MPI program
 - For *dubious* reasons: „to be implemented“
 - We perform local, low-cost error checks by default
 - However, a lot of additional things *can* be done to detect incorrect usage of MPI operations
 - Here, we concentrate on *online-analysis of collective operations*
 - *Collective*: dependencies for correctness across process boundaries
 - *Online*: make usage as simple as possible
- “Analyse *where* you run, *when* you run.”

Example: Messing up MPI_Alltoallv

- **Task:**
 - Each process p should exchange individual data with all other processes q .
 - Amount of data proportional to p

A case for MPI_Alltoallv!

```
for (j = 0; j < size; j++) {  
    sendcounts[j] = p;  
    recvcounts[j] = j; No!  
}  
MPI_Alltoallv(sendbuf, sendcounts, ...,  
              recvbuf, recvcounts, ..., comm);
```

Correct?

Example: Fool yourself with MPI_Bcast

- **Task** (actual real-world case):
 - Broadcast a dynamic amount of data

A case for two-step *MPI_Bcast()*!

- **Rank 0:**

```
CALL MPI_BCAST(values, size0, ..., 0, comm, ierr)
ALLOCATE(buffer(values(3)))
CALL MPI_BCAST(buffer, values(3), ..., 0, comm, ierr)
```

- **Other ranks:**

```
CALL MPI_BCAST(values, size1, ..., 0, comm, ierr)
ALLOCATE(buffer(values(3)))
CALL MPI_BCAST(buffer, values(3), ..., 0, comm, ierr)
```

Strange crash *later* in the program..for `size1 < size0`!

Example: Funny Files with Shared Filepointer

- Access a file using the shared file pointer:

```
MPI_File_{read|write}_shared(...)
```

- Requires all processes in group to use the same file view
- Otherwise, inconsistent handling of file pointer will occur, leading to corrupted files
- File view has to be exchanged when it is established (MPI/SX does this anyway)
- **Non-local verification in non-collective operation**

The MPI/SX Approach

- Additional MPI library to be used *like* a profiling library:

```
mpicc -mpiverify test.c -o test
```

(does `cc test.c -o test ... -lvmpi -lmpi`)
- In fact, this is *not* a profiling library, and a real profiling library can still be used
- *Not* a portable approach
 - Portable solution has problems with certain collectives:
 - How to do collective communication (over `MPI_Win`) when checking `MPI_Win_fence(win, assert)`?
 - How to exchange the `MPI_Group` argument when checking `MPI_Comm_create(old_comm, group, &new_comm)`?
 - Portable solution has more overhead in run time, code and memory size
 - Occupies profiling interface (can not verify profiling libraries)

Scope of Verification

All MPI operations with non-local completion semantics:

- **Collective operations** (`MPI_Bcast()`, ...)
 - **Communicator manipulating functions** (`MPI_Comm_create()`, ...)
 - **Dynamic processes** (`MPI_Comm_spawn()`, ...)
 - **Process topologies** (`MPI_Graph_create()`, ...)
 - **One-sided communication** (`MPI_Win_fence()`, ...)
 - **Parallel I/O** (`MPI_File_open()`, ...)
- Checks are performed for Intra- and Intercommunicators as well

What is checked?

Generally:

- Call consistency:

`MPI_Bcast()`

`MPI_Reduce()`

- Root consistency

`MPI_Bcast(..., 0, ...)`

`MPI_Bcast(..., 1, ...)`

- Pairwise matching buffer size (count and datatype)

- Consistent use of other parameters (`MPI_IN_PLACE`,

`MPI_Op`, file size, offset, ...)

More specific:

- Graphs and dimensions in topology functions
- Fileview datatypes in I/O
- Identical group parameter

Diagnostic Messages

- Output for the incorrect usage of `MPI_Alltoallv()` described before:

```
VERIFY MPI_ALLTOALLV(0): sendsize[1]=4 != expected recvsize(1)[0]=16
VERIFY MPI_ALLTOALLV(1): sendsize[0]=0 != expected recvsize(0)[1]=12
VERIFY MPI_ALLTOALLV(2): sendsize[0]=0 != expected recvsize(0)[2]=8
VERIFY MPI_ALLTOALLV(3): sendsize[0]=0 != expected recvsize(0)[3]=4
```

- Non-matching datasize in `MPI_Gather()`

```
VERIFY MPI_GATHER(0): datasize=4 != datasize=40 of root 3
```

- Non-matching operation in `MPI_Reduce()`

```
VERIFY MPI_REDUCE(1): MPI_op 101 differs from MPI_op 100 at 0
VERIFY MPI_REDUCE(3): MPI_op 101 differs from MPI_op 100 at 0
```

Implementation

Typical verification (here: `MPI_Alltoallv()`)

- Local verification (i.e. negative count)
- Call consistency

```
my_call = root_call = VMPI_ALLTOALLV_ID;
internal_bcast(root_call, 1, MPI_INT, 0, comm);
If (my_call != root_call)
    /* print diagnostic message, remember error code */
```

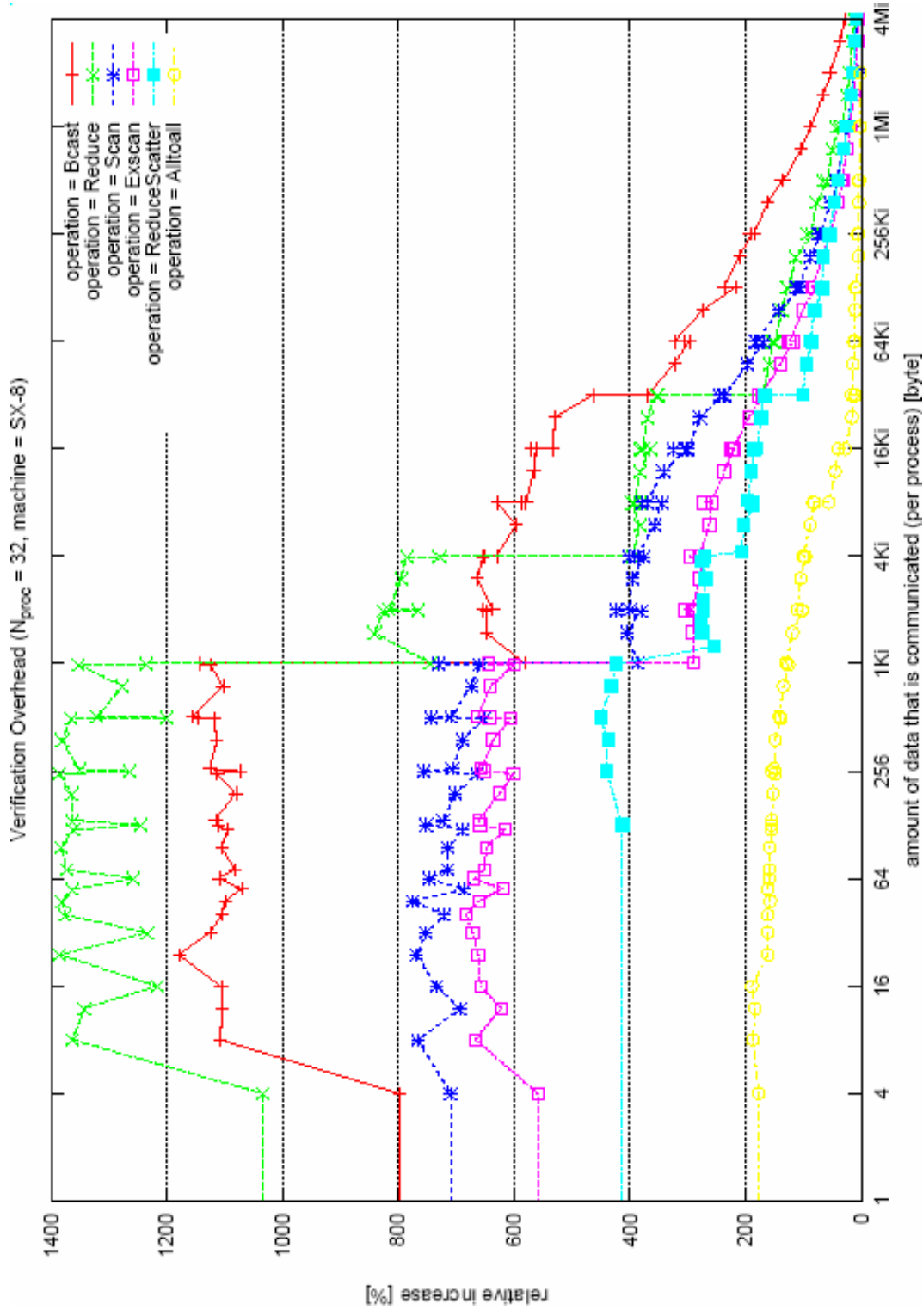
- Count consistency

```
internal_alltoall(recv_counts, size, MPI_INT,
    expected_counts, size, MPI_INT);
for (i=0; i<size; i++)
    if (send_counts[i] != expected_counts[i])
        /* print error message, remember error code */
```

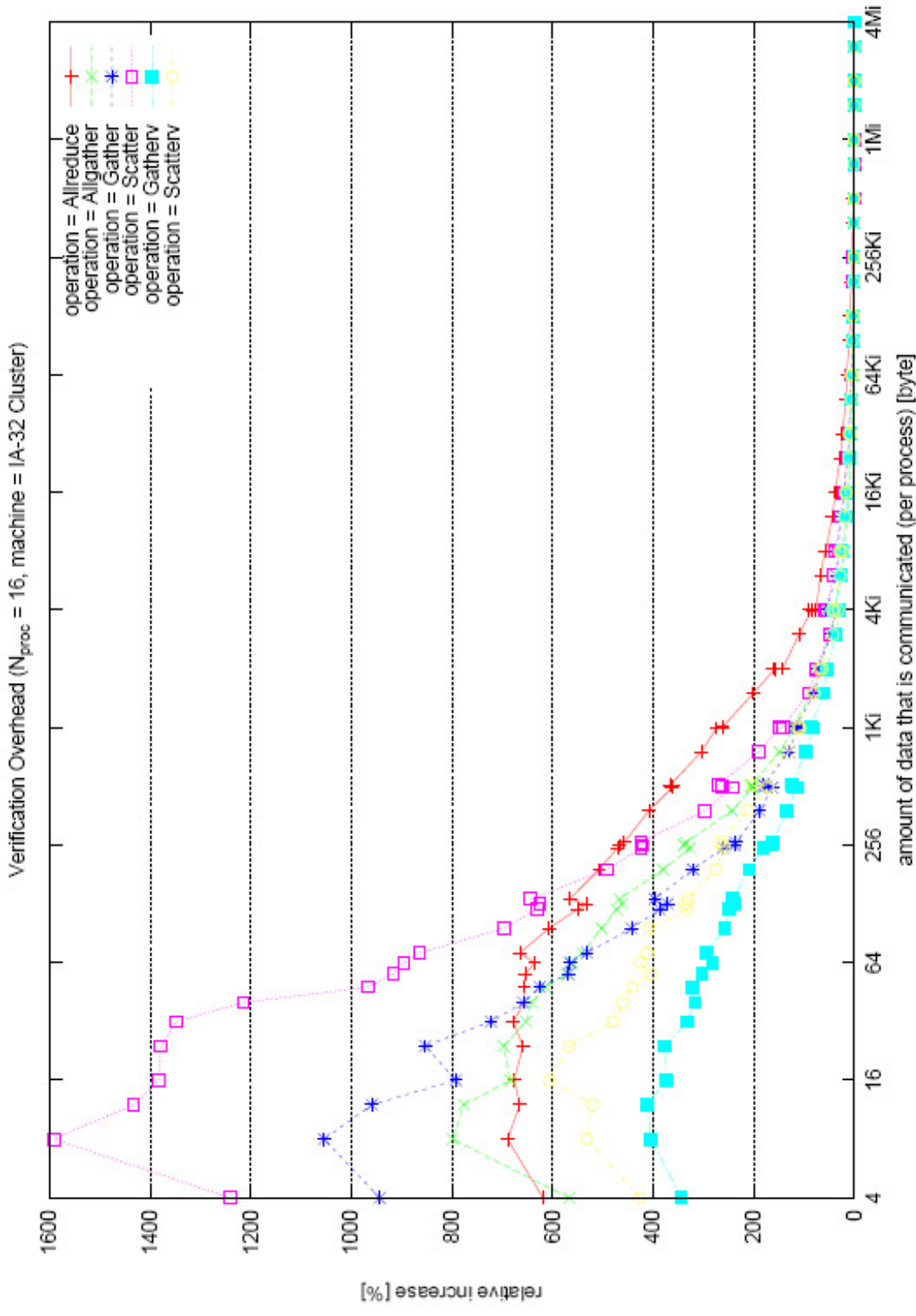
- Deadlock avoidance

```
err[0] = local_error; err[1] = remote_error;
internal_allreduce(MPI_IN_PLACE, ..., err, 2, MPI_INT, ...)
```

Synthetic Overhead (SX-8)



Synthetic Overhead (IA-32, Myrinet)



Application Overhead

- Evaluation platform:
 - IA-32 Cluster, 2xAMD Athlon 2.2GHz, 4GB, Myrinet2000
 - running 16 processes on 8 nodes
- Application
 - *hyre* library of solvers and preconditioners:
 - i,j test included with *hyre* (solver 0 ‘AMG Boomer’)
 - problem size set to 1GB per process
 - 3690 x MPI_Allreduce, 592 x MPI_Gather,
 - 640 x MPI_Allgather, 304 x MPI_Allgather
 - **Estimated overhead: ~ 2s**

Results for Application Overhead

- total wall clock time for i_j (100 samples):

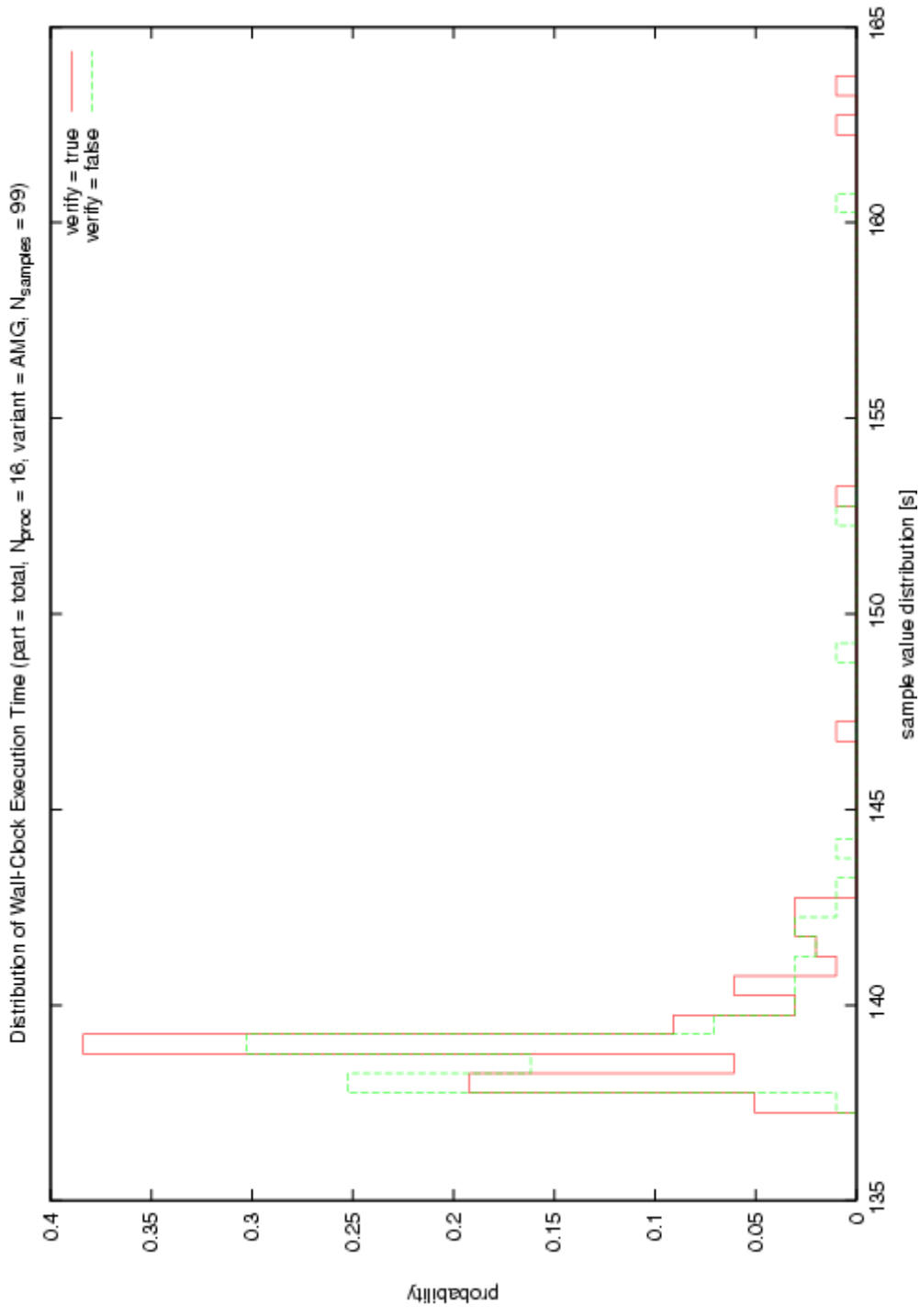
	w/o verify	w/ verify	overhead
avg. wall clock time [s]	139.78	140.10	0.32
rms [s]	2.96	3.86	n/a
90-quantile [s]	141.64	142.27	0.63

- Relative overhead: 0,2% (avg), 0,44% (90-quantile)*

Note:

- For 50 executions, the rms was about twice as high as for 100 iterations! Make sure you measure properly.

Distribution: ij Total Runtime



Summary

- **Simple-to-use extension of an MPI library**
 - Link & run
 - No messages: usage of collective operations is correct.
 - Detailed diagnostic messages for wrong usage.
- **Overhead**
 - High nominal overhead for small data size
 - Low effective overhead observed for application kernel
- **Future work**
 - Online-verification of point-to-point communication

Note

Data gathering, analysis and plotting was done using *perfbase*, an open-source toolkit developed by NEC CCRL.
[available from `perfbase.tigris.org`]

Empowered by Innovation

NEEC