

MPI Performance Tuning.



Agenda.

- MPI Implementation.
 - MPICH.
 - BG Networks.
 - MPI programming style.
- Optimisations.
 - Generally, and specifically, for the BG.
 - Point-to-point, collectives, MPI-IO.
 - Process Layout.
 - MPI engine features.
- Advanced Features/The Future.
 - Mixing openMP and MPI.
 - MPI2, MPI3.

MPICH

- Derived from MPICH2, 1.0.x
 - “MPI standard 2.0” implemented
 - Except for process spawn/connection.
 - Currently based on the 1.0.7 MPICH2 base code.
 - Working with ANL to include BGP changes into main tree.
 - Contains a number of optimisations and tuning features for specifics of the BlueGene Architecture.

BG Networks.



- Three networks.
 - A 3D torus.
 - Connects all compute nodes.
 - 3.4Gb/s all links.
 - Hardware latency $0.5\mu\text{s}$ near, $5.0\mu\text{s}$ max.
 - MPI latency $3\mu\text{s}$ near, $10.0\mu\text{s}$ max.
 - A collective network.
 - One-to-all, reductions.
 - 6.8Gb/s
 - Full tree hardware latency $2\mu\text{s}$, MPI $5\mu\text{s}$.
 - Low latency barrier and interrupt.
 - All nodes in $0.65\mu\text{s}$, MPI $1.6\mu\text{s}$.

MPI Style.

- Some things are always hard to do/scale poorly.
 - Many to one/One to many.
 - Spamming a node.e.g. IO activity.
 - Dependencies.
 - Nodes waiting for others.
 - Global tests and synchronisation.
 - Coping with poorly distributed/dynamic workloads.
- Avoid these in programming.
 - Is that barrier necessary?
 - Is many to one, or one to many, really necessary?
 - Can we find work to do while waiting?
 - Send a.s.a.p.
 - Compute items to send first.
- *Is your communication really necessary?*

BG Optimisations.

- Collectives.
 - Common tasks and synchronisation.
- Point-to-point.
 - Exchanging data.
- Process layout.
 - Mapping processes to the hardware.
- IO in MPI, and MPI-IO.
 - Getting things in and out.
- MPI Engine Tuning.
 - When this is likely to be an advantage.

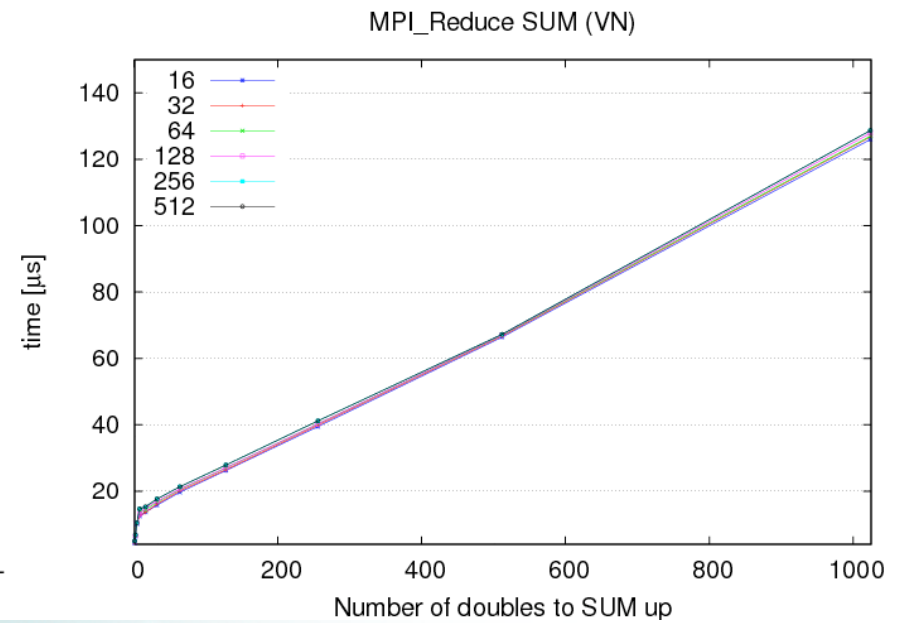
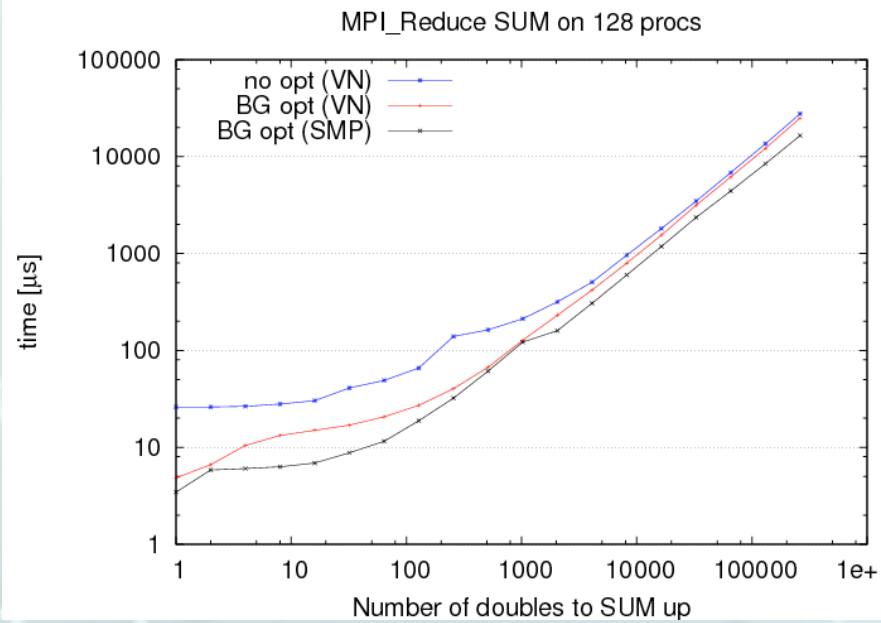
Collectives.

- Principle of having these defined in MPI is so they can be optimised by vendor to reflect system hardware.
 - So use them whenever possible!
- Some only optimised for MPI_COMM_WORLD.
- Manual tuning.
 - User can set environment variables.
 - Methods used.
 - Some have tuning for size of data, number of processors etc.
 - Choice of HOST for data.
 - Specific process, those with memory or time.
 - Round robin.

Collectives and Communicators.

- Optimised for MPI_COMM_WORLD.
 - Bcast, (All)reduce, Barrier, Gather
- Optimised for any communicators.
 - Alltoall(v)
- Mixed, methods depend on datasizes.
 - Allgather, uses reduce and bcast or alltoall.
 - Scatter, uses Bcast
 - Scatterv, uses alltoallv or bcast.
- Fine tuning is done through environment variables.
 - DCMF_COLLECTIVES=0, turns off BG optimisations.
 - Can select method each collective uses.
 - DCMF_x=MPICH, makes collective x use MPICH method.
 - DCMF_x can also set other options.
 - See REDBOOK for full details.

Proof.



Point-to-Point.

- Various SEND/RECV mechanisms.
 - Non-blocking gives best performance.
 - Avoid synchronous sends where possible.
 - Avoid buffered sends/non-contiguous data.
 - Cost of copying into memory.
- Basic principles.
 - Don't overload message system.
 - Post receives as soon as possible.
 - TEST/WAIT receives often.
 - Overlap communication and computation.
 - (Relatively) slow processors on BG, fast network.

Point-to-Point.

- Some control over internals of MPI engine.
- Three methods for message handling.
 - Short messages < 224 bytes.
 - Message is sent as single packet.
 - Medium messages, Eager.
 - Message is sent assuming space in buffers.
 - Large messages, Rendezvous >1200 bytes.
 - Checks for space, then sends, so has higher latency.
- Control using DCMF_EAGER. (Default 1200 (Bytes))
 - Decrease, if application uses many small messages in code and we are not sensitive to latency.
 - Increase, if sending mostly long messages.
 - Will use more buffer space.

Process Layout.

- Process locations determine costs of communication activity.
 - Put processors which exchange data nearest to each other.
 - Try not to create hotspots/congestion in network.
 - Map logical process topology to physical hardware.
- Two things we can do to help.
 - Two ways to route messages.
 - Deterministic, the same way every time.
 - Adaptive, depends on load.
 - Which is selected depends on size of messages which use rendezvous protocol.
 - So large DCMF_EAGER values can create hotspots.
 - Map processes to the processors to improve communications.
 - X,Y,Z for ICHEC is 8x8x16.
 - MPIX_rank2torus/MPIX_Comm_rank2torus.

Process Layout.

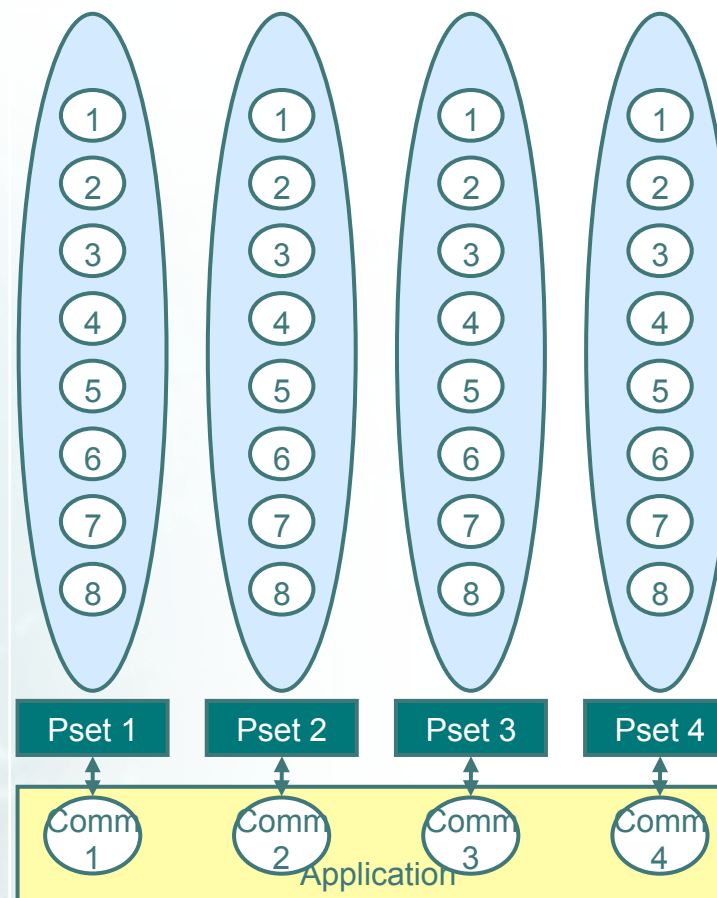
- How do the MPI ranks map to the BG hardware?
 - Consider BG to be a 4D torus, X,Y,Z and on node, T.
 - Ideally this should map naturally onto my data decomposition.
 - Can find position using, `MPIX_rank2torus/MPI_Comm_rank2torus`.
 - Various options can be specified on `mpirun` using `-mapping`
 - `-mapping XYZT` is default.
 - X, Y, and Z are torus dimensions, T is on node.
 - Fill in order specified, so on `2x2x2x2`
 - `(0,0,0,0),(1,0,0,0),(0,1,0,0),(1,1,0,0),(0,0,1,0)`
.....`(1,1,1,0)(1,1,1,1)`
 - Some mappings predefined, `T{X,Y,Z}` and `{X,Y,Z}T`.
 - Can create specific map files to match your needs.
 - More details in the REDBOOK, BlueGeneP Application Development.
 - *Quick test is to try `TXYZ`, this puts neighbouring ranks on the same node.*

IO in MPI and MPI-IO.

- Most popular IO strategy in MPI programs.
 - Collect data on one node and write!
- Some alternatives.
 - Use all nodes!
 - Write a file per process, merge later!
 - Might overload IO nodes/storage.
 - Use a few nodes instead of just one.
 - Complexity of code and porting/tuning needed.
 - Create an IO server dedicated to doing IO.
 - Complexity of coding.
 - *Use MPI-IO.*

IO in MPI.

- Extension to MPI.
- `MPIX_Pset_same_comm_create()`
 - Creates communicators which share same IO node.
 - Can then use desired rank in each communicator to write those processes out.



MPI-IO.

- BG supports full MPI-IO implementation.
- Some environment variables to define how MPI-IO works.
 - BGLMPIO_COMM, default (0) uses alltoall, set to 1 to use send/recv.
 - BGLMPIO_TUNEGATHER, aggregator I/O, default (1) uses allreduce, set to 0 to use allgather.
 - Other BGLMPIO_x.
- Soon to be a REDBOOK on BG GPFS performance tuning.

MPI Engine Tuning.

- If work isn't balanced tuning not likely to show improvement.
- Discussed tuning for message sizes.
 - Environment variables cover entire program.
- A few general tips.
 - Don't have too many outstanding messages.
 - Use non-blocking.
 - Post RECV as soon as possible.
 - TEST/WAIT often.
 - Avoid non-contiguous datatypes.
- FAST mpi.
 - MPI library without debug/error checking.
 - Use compilers from `/bgsys/drivers/ppcfloor/comm/fast/bin`.
 - *If you are confident your code works, try this to see if it is an improvement.*

Advanced Features/Future Developments.

- Mixing MPI and openMP.
- Language developments.
 - MPI2.
 - MPI3.
 - Other directions/developments.

MPI on BG/P – Run modes

Virtual Node (VN)

- Each core gets its own MPI rank, kernel image
- 4x the computing power
- Not necessarily 4x the performance
 - Each core gets $\frac{1}{4}$ memory
 - Network resources split in fourths
 - Cache split in half (L3) and 2 cores share each half
 - Memory bandwidth split in half
 - CPU does compute and communication, though DMA helps
 - Global communication can be expensive
 - No threads

SMP

- Full memory available
- All resources dedicated to single kernel image
- Can start up to 4 pthreads/OpenMP threads
 - `OMP_NUMTHREADS=x`

DUAL

- Hybrid of the two modes
 - 2x MPI ranks of SMP, each rank can start 1 additional thread
 - $\frac{1}{2}$ memory of SMP per rank

Mixing MPI and openMP.

- SMP mode.
 - Single MPI task on each node and 4 threads.
- When might this be useful.
 - Reduces MPI complexity. $\frac{1}{4}$ processes.
 - Load balance between the 4 cores can be improved.
 - Careful mixing of processes/tasks.
 - Can ease heterogeneous programming.
 - Tuning can be very specific to problem and system.
- Some challenges when mixing openMP and MPI.
 - Is MPI thread safe?
 - Best not to call MPI from parallel regions.

Future Developments for MPI.

- MPI2.
 - Almost frozen.
- MPI3.
 - Just about to start and look at new features and requirements.
- Alternatives to MPI.
 - Global Arrays.
 - Higher level languages.
 - Co-array Fortran, UPC.
 - X10, Chapel, Fortress.
 - Lower level API.

Conclusions.

- Good MPI Principles.
- BG Hardware and MPI.
- Optimisations you can do.
 - Short order environmental experiments.
- Ideas for future code developments.
 - Either apply to existing codes,
 - or when you are developing new code.
- Measure to see what happens.
 - *Report results, both good and bad.*

