IBM Advanced Clustering Technology Team

Parallel Environment MPI - Today and Tomorrow

ScicomP 12 - NCAR at Boulder, CO

July 20, 2006

Dick Treumann - MPI Team Lead

The futures material presented represents a mix of experimentation, prototyping and development.

While topics discussed may appear in some form in future IBM products there is no guarantee any particular feature will appear precisely as described.

Some work described may never go farther than prototype form.

Observations on PE 4.2

4.2.2 For AIX on System P

4.2.0 For Linux on System P

4.2.0 For Linux on System X

Fix downloads and pointers to information are now at:
http://www14.software.ibm.com/webapp/set2/sas/f/penv/home.html

Enhancements: Parallel Environment 4.2

- **MPI on AIX now supports 8196 tasks in a single job**
- **The pSeries High Performance Switch is supported on POWER 5 and Power 5+ nodes as well as POWER 4**
- **MPI and LAPI shared memory operations now support up to 128 tasks/node**
  - On a 64 CPU POWER 5 node with Simultaneous Multi Threading on, there may be reason to run 128 MPI tasks
    - Note that performance gain from using SMT with a doubled task count is application dependent. Many apps see some gain while a few see some loss.
- **Support for striping of messages over multiple adapters/networks with the pSeries HPS**
- **Linux UDP/IP MPI on both system P and system X (4/06)**

pSeries HPS provides rDMA for MPI messages

- **Larger messages use rDMA which can provide several advantages:**
  - The CPU is available to the application while data flows
  - A bit better bandwidth with a single network
  - Multiple networks provide near linear bandwidth scaling
- **No changes required in the application**
  - Making more use of computation/communication overlap should be more worthwhile with rDMA available.
  - Scheduling ISEND/IRECV with multiple partners may give better communication concurrency because CPU bottleneck is eliminated

pSeries HPS provides rDMA for MPI messages

- **rDMA works best with application buffers in large pages**
- **First use of any application buffer for an rDMA eligible message pays significant startup cost**
- **rDMA setup is semi-persistent (memory that is reused for rDMA messages is not likely to pay again)**
- **pin & translate is in aligned units of 16 MB (a 256KB rDMA can trigger a 16MB pin & xlate)**
- **a memory range that has been made rDMA ready but is not used in communication for 30 sec gets unpinned**
- **if rDMA spreads over many different 16MB chunks, some LRU unpins may occur**

Unfortunately, standard MPI has no mechanism for predeclaring buffers and giving them attributes. But applications do tend to use the same data structures over and over for communication, so most are expected to pay setup only once per 16MB.

MP_EAGER_LIMIT and MP_BUFFER_MEM

- **The MPI standard allows an eager protocol but requires that it be absolutely safe. That requires the MPI implementation refrain from eager send if it is not certain the message can be buffered.**
- **The worst case for eager protocol is when every task does aggressive sends to a single task and the target task does not post the receives until later.**
- **The sending tasks must fall back to rendezvous protocol and throttle sending before the target risks running out of early arrival buffer**

MP_EAGER_LIMIT and MP_BUFFER_MEM

- A nasty FVT test case will try the aggressive flooding but real codes seldom do anything like this
- Well designed codes often try to pre-post receives but end up buffering a few early arrivals due to load imbalance.
- Real demand for buffer memory can be hard to predict but is usually a tiny fraction of the worst case that must be covered.

MP_EAGER_LIMIT and MP_BUFFER_MEM

- Large task counts with a high eager limit either:
  - Require a very large MP_BUFFER_MEM
  - Cause premature fallback to rendezvous
- A new option for MP_BUFFER_MEM solves the problem – MP_BUFFER_MEM=P,L
  - P specifies the memory to be preallocated for early arrivals
  - L specifies the limit on memory to malloc if the application does do target flooding

Applications with modest task counts, default eager limit and reasonable working memory can usually ignore this option. The option may be most useful on Linux (AIX deferred page mapping makes large but unused preallocations less costly)

MP_EAGER_LIMIT and MP_BUFFER_MEM

Example: MP_BUFFER_MEM=4M,1G

- 4 meg is preallocated
  - Early arrivals go first to this 4 meg
  - Management is efficient
  - A halo code task that was late in posting 26 Irecvs of 64K would use less that ½ of the 4 meg
- The nasty FVT test will force malloc up to 1gig before falling back to rendezvous

To apply this to an application, use MP_STATISTICS to find the high water mark and frequency of rendezvous fallback. Pick a preallocation that covers the high plus a bit and a limit that prevents fallback.

Think twice about MP_SINGLE_THREAD

- **If your application has more than one thread calling MPI you risk race condition errors – no checking, no warnings**
- **Savings is < 1 microsec per MPI call**
  - noticable benefit on tiny message send/recv
  - imperceptable benefit with larger messages and collectives
- **MPI-IO and MPI-1sided will issue an error message**
- **The common model for OpenMP/MPI hybrid apps is safe**
  - parallel sections used for compute that do not call MPI
  - serial part of program calls MPI (on the main thread)
- **Setting MP_SINGLE_THREAD=yes has no impact on how many threads the process uses.**
  - It simply shuts off locking with no checking for whether there are MPI calls from > 1 thread.

Some simple SMT observations

- SMT pays off quite reliably as a system thru put feature. A given number of CPUs will almost always do more computations/second in SMT mode than in ST mode.
- Whether that pays off for you depends a lot on your situation and how you define the payoff.
- Many people have done evaluations of SMT and if you need detailed analysis and models for prediction, you can read as many as you like.

The next 2 slides offer a couple simple examples to hint at when SMT is more likely or less likely to pay off

Some simple SMT examples

- If your application takes 2 hours with 64 ST tasks but does not come somewhere near 1 hour with 128 ST tasks, running SMT to get 128 tasks on 64 CPUs may not pay off (and could do worse)
- When you double the task count and cut the computation per task by ½ you expect an SMT thread to accomplish this ½ portion in less time than an ST mode CPU would require for a full portion. It almost always will do that but by how much less depends on the nature of the computation.

Some simple SMT examples

- If you have 100 jobs of 32 tasks each to finish by Monday and 64 CPUs you can run 2 at a time in ST and 4 at a time in SMT. If each job on 16 CPUs with SMT takes 1.6 times as long as it would on 32 CPUs with ST – SMT gives you a better shot at having them all done by Monday
- If 2 jobs sharing the 64 CPUs would use > 50% of the system real memory then 4 jobs sharing 128 SMT threads are likely to run short.

PE MPI
Thoughts on the Future

Linux is part of the Future for Parallel Environment MPI
The first PE for Linux became available in April
Enhancements will follow

- **Many clusters, from modest to high power are being build on nodes that run Linux**
- **One enterprise may have both clusters of Linux nodes and of SMP POWER Nodes. The POWER nodes may run Linux or AIX**
- **Parallel Environment MPI and LAPI have a history of robustness and solid support for Parallel HPC**
- **The option of using a consistent environment on many different systems is attractive**
- **The optimization of the communications stack that IBM has invested in can pay off on Linux just as it does on AIX**

We understand the importance of Infiniband & Ethernet in the Future of Parallel parallel computing

- **Infiniband is growing fast in the Parallel HPC world. Demand for MPI that maximizes IB effectiveness will grow. PE intends to be there.**
- **Ethernet will continue as a cost effective mid performance interconnect. It may not be adopted where maximum performance is needed but its role will continue. PE is there today and intends to get stronger.**
- **Both IB and Ethernet networks can be coupled with adapters that include Parallel HPC enhancements. PE intends to be there.**
- **Adapters that include Parallel HPC enhancements coupled with an MPI and LAPI that exploits them could provide top flight cost/performance. PE is exploring these options.**

A better MPI 1sided implementation

- **Before PE/MPI was built on LAPI, there were serious limitations in our ability to do MPI one sided well**
  - PE already offers a standards compliant MPI-1sided but application developers have often found it too far short of pt2pt performance to seriously consider
- **Since the layering of MPI on LAPI we have available a communication lower layer that fits the needs of MPI 1sided**
- **We now have a prototype implementation of MPI 1-sided that uses LAPI directly and makes the 1-sided programming model in MPI a logical option for the application developer.**
- **We hope to have it in your hands before long so you can use the MPI 1-sided model with good performance.**

Collective Communication improvements

- **In PE/MPI we are constantly looking for way to improve collective communication perfromance**
- **Our current focus is very large task counts**
- **Small message MPI_Bcast/Reduce/Allreduce & Barrier can be improved by having more than one task per node concurrently do offnode communication.**
- **Large message MPI_Bcast can be improved by slicing up the message and establishing parallel pipelines to maximize concurrency in getting the whole message to every task**

Large Page issues will become less critical

- **Power 5+ and later system have 64K pages**
- **64K pages are pagable. Physical memory does not need to be set aside and pinned**
- **The same physical memory can be use for either 4K or 64K pages depending on the needs of the process**
- **Performance investigations are not completed but results so far suggest that communication buffers in 64K pages closely match the perfromance with buffers in pinned large pages.**

For existing systems configured for Large Page

- **When a large percentage of the nodes pages are dedicated to the large page pool and parallel jobs are run without being large page enabled - serious trouble can follow**
- **PE will provide an option to let the system admin specify either:**
  - a PE job that is not configured for large pages will be prevented from running
  - a PE job that is not configured for large pages will issue a warning
- **The option only affects Parallel Environment job tasks. Scripts and other processes can still pass up large pages**

--- Finally ---

IBM (I) would like your thoughts

please email any comments to:

treumann@us.ibm.com

Comments with rationale are very useful. A simple yes/no will be noted but does not add much to understanding the requirements

MPI_COMM_SPAWN

PE MPI does not include dynamic tasks. With a tightly managed cluster, adding real resource to a running job is problematic. With our US model, so is dynamic update of connection tables.

Questions: Is MPI_COMM_SPAWN something you need? If so, how much of your need would be satisfied by a User Space model in which a job requests a specific amount of resource at the start, owns that resource until the end and spawns/retires tasks with that allocation.

e.g. You request resource for 65 tasks but start one. That one task spawns 64 workers with executable A, after a while shuts them down and spawns 64 new tasks with executable B, then C; D etc. Because the job size stays constant, the resource is not sitting idle.

Full MPI Dynamic API

Question: Is the full MPI Dynamic API important to you? Is so, how much of the requirement would be satisfied if the API were 100% available but only in when the job was running in UDP/IP mode?

Contact Information

Richard Treumann

IBM Poughkeepsie UNIX Development Lab

treumann@us.ibm.com