



IBM Advanced Clustering Technology Team

Parallel Environment MPI - Today and Tomorrow

ScicomP 11 - Edinburgh, Scotland
June 3, 2005

Dick Treumann - MPI Development

The 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.

MPI Enhancements - PE 4.2

With help from LAPI

Enhancements: Parallel Environment 4.2

- **MPI now supports 8196 tasks in a single job**
- **The pSeries High Performance Switch is supported on 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 significant gain while a few see some loss.
- **Support for striping of messages over multiple adapters/networks with the pSeries HPS**
- **MPI collectives algorithm and optimization improvements**

pSeries HPS provides rDMA for MPI messages

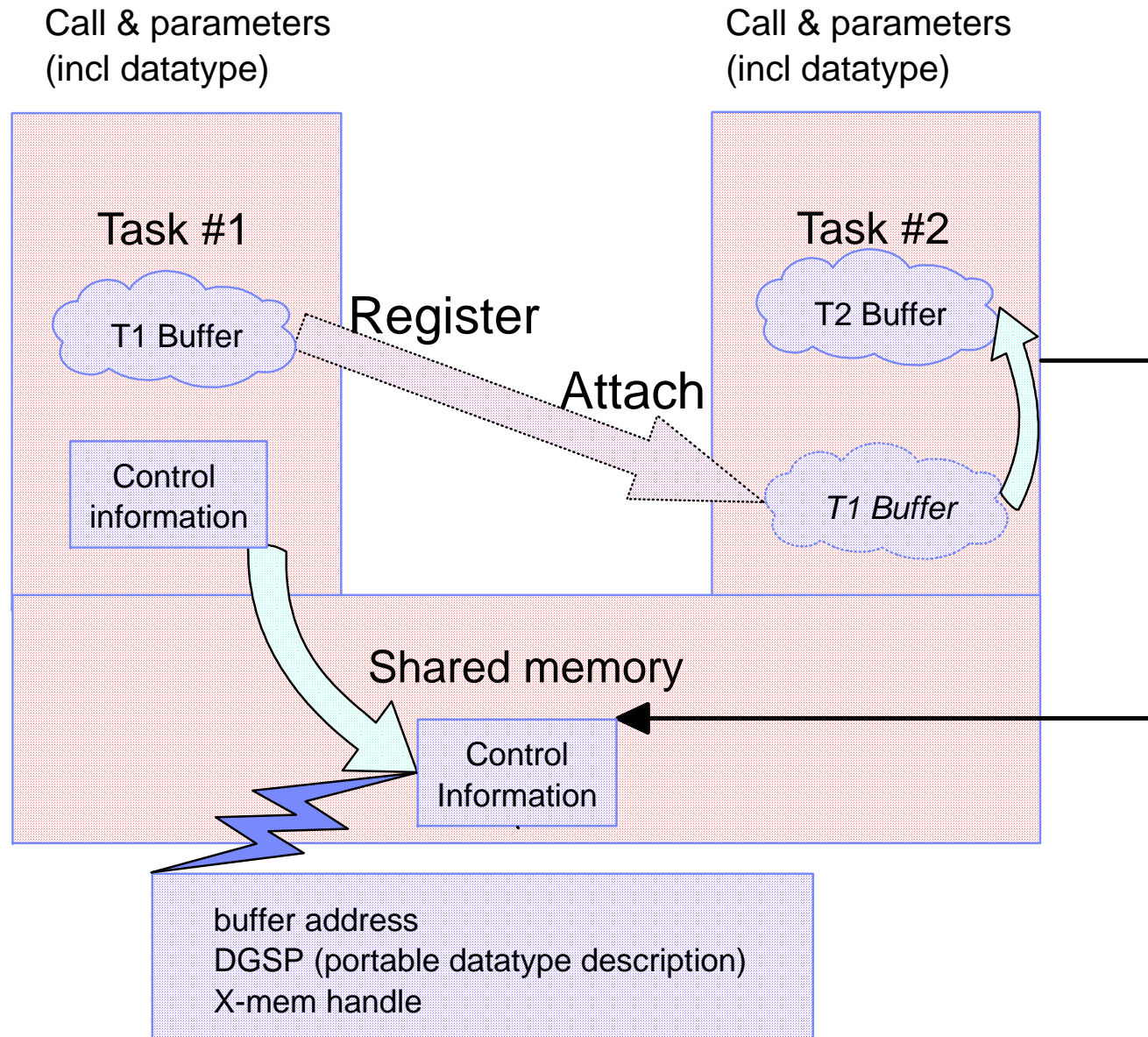
- **Off by default with AIX 5.2 - On in AIX 5.3**
- **May be switched on/off by system admin**
- **Larger messages use rDMA which can provide several advantages:**
 - The CPU is available to the application while data flows
 - A bit better bandwidth with a single network
 - Multiple networks provide 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 unused for 30 sec gets unpinned**
- **if rDMA hits many different 16MB chunks, LRU unpin may occur**

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

Improvements to Cross Memory Attach for Shared Memory MPI P2P and Collective communication



Prior Impact of Cross memory Detach costs

- **Each detach operation had OSI wide impact**
- **The time cost grew as CPU count grew**
- **Detach operations serialize in the kernel**

These costs had limited the value of using xmem attach in shared memory message passing, especially for collective operations.

In partnership with AIX, we developed a lightweight form of attach/detach on AIX 5.3 for 64 bit applications. The OSI wide impacts now happen in 1 of 1000 detach operations.

Co-Scheduler Enhancements

PE Co-Scheduler Motivation

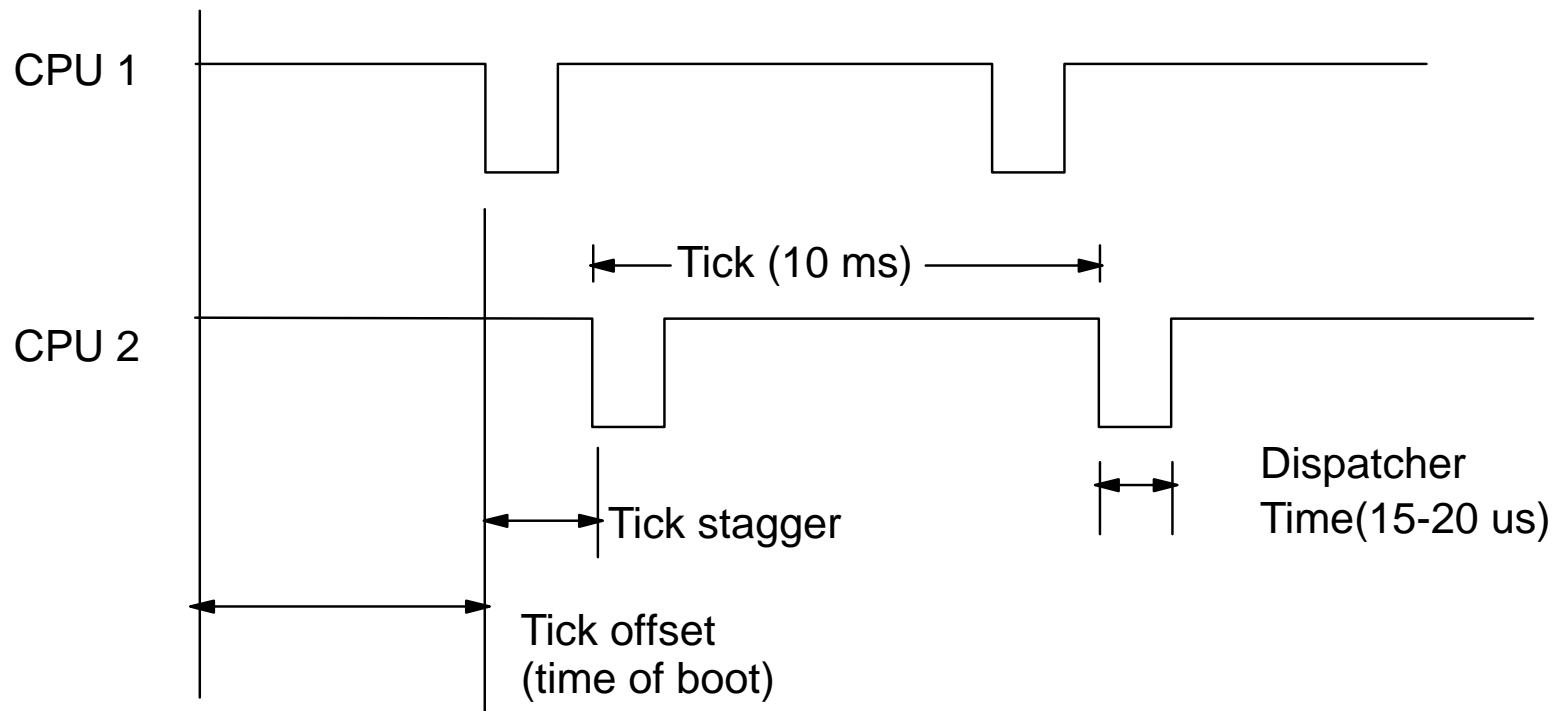
- **Several investigations and papers have looked at the impacts of random system events on synchronizing MPI operations.**
 - MPI collectives like MPI_BARRIER
 - User written synchronizations like HALO exchange
- **Three major classes of damaging events:**
 - OS needs to run its scheduler at each time slice
 - System daemons run at seemingly random times
 - Work that is not part of parallel jobs, if allowed on system has random impacts.
- **Most system administrators have long ago learned to keep asynchronous actions (interactive logins, non-vital daemons, file servers) off loaded compute nodes.**

PE Co-Scheduler

- **AIX time slice can now be increased so OS events happen less often**
- **By default, AIX does OS scheduling CPU to CPU, round robin. Good for unsynched processes but not for MPI jobs. Better every task takes the hit at the same time. New AIX option allows all CPUs to run scheduler simultaneously**
- **Daemons can be managed by moving priority of all parallel tasks up and back down in a cycle that puts the parallel tasks on/off together, letting daemons have their turn**
- **With PE Co-Scheduler, it is possible to apply both the AIX scheduler and priority cycling strategy to all nodes of a cluster and keep it in synch across the cluster.**

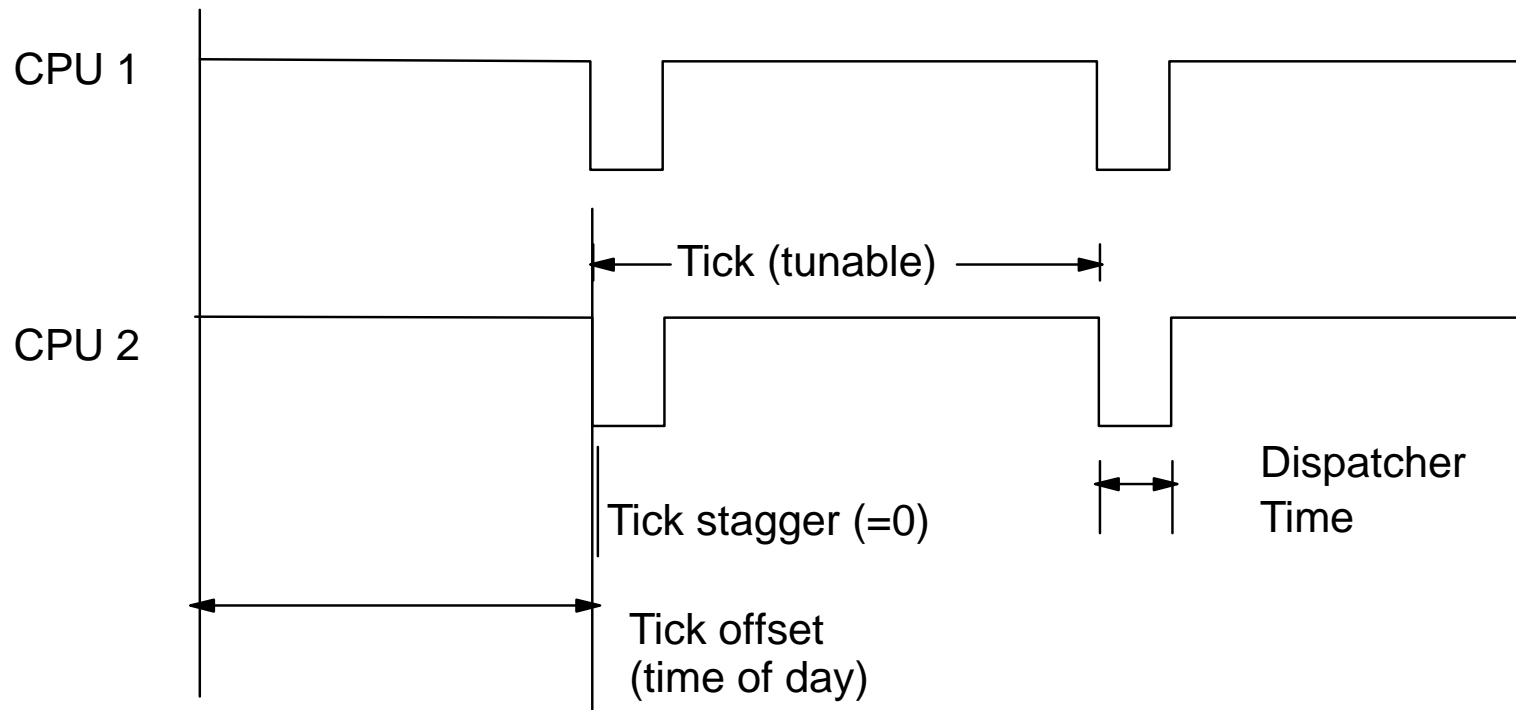
■

AIX Dispatching - Normal



- Default tick is 10 milliseconds
- Dispatcher time is typically 15 - 20 microseconds
- Stagger is $1/N_{CPUs} * \text{Tick time}$
- Tick offset, node to node, is random (based on boot time)
- Max single CPU availability is 99.8 - 99.85%
- With 64 CPUs, perhaps 20% of the time some CPU is in the dispatcher

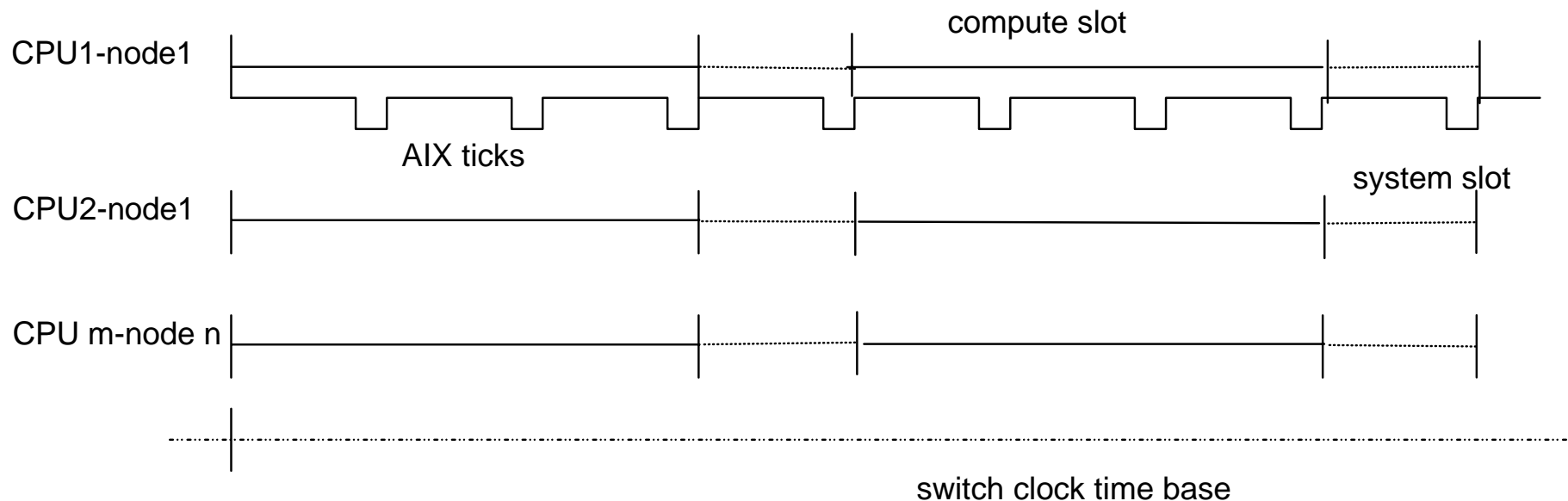
AIX Dispatching - Co-Scheduler support



- Tick is settable via `schedo -o big_tick_size=n` (AIX 5.2/5.3)
 $n \geq 1$, $n \leq 100$, n must divide 100
- Stagger is 0 (but more lock contention could increase dispatcher time a bit)
- Tick offset is based on time of day (if $n > 1$)
- "line up" the compute/communicate slots with no CPU in dispatcher
- For 20 or so microseconds per tick, all CPUs are in the dispatcher
- Run queue control via `schedo -o force_grq=1`

Co-Scheduler - objectives

- Provide "dedicated compute" slots and "system activity" slots
- Synchronize compute slots on node and across nodes
- Exploit AIX Co-Scheduler support



- setpri periodically to create favored/not-favored compute intervals
- use switch clock to synchronize AIX TOD on each node
- MPI tasks are each on a CPU associated run queue
- other processes on global run queue to effectively spread across CPUs during "system" slots

Co-Scheduler Admin File (/etc/poe.priority)

#User	Class	Hi	Lo	%Hi	Period	
bob	realhog	30	65	99.0	10.0	#Trailing comment
bob	niceguy	45	120	95.0	10.0	
*	realhog	40	100	97.0	10.0	
jim	niceguy	50	90	90.0	20.0	
*	sponge	65	70	99.9	50.0	
*	MINIMUM	40	90	90.0	5.0	
*	MAXIMUM	90	120	99.0	30.0	

MP_PRIORITY={class|(hi : lo : duty : period)}

MP_PRIORITY=realhog

or

MP_PRIORITY=35:70:98.5:15.0

Default process priority is 60

PE MPI

Thoughts on the Future

IBM Recognizes the potential of Linux in the Future of Parallel Environment MPI

- **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**

IBM Recognizes the potential of Infiniband & Ethernet in the Future of Parallel Environment MPI

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

--- 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

The US connection model uses a job key to ensure stray or malicious packets from outside the job are dropped. Two different MPI jobs have distinct keys and cannot be put into US communication. Connection tables cannot be grown dynamically

Question: Is full MPI Dynamic important to you? Is so, how much of the requirement would be satisfied if the API were 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