

Case studies and MPI lessons learned

Mark R. Fahey (& Trey White)

faheymr@ornl.gov

**Center for Computational Sciences
and
Joint Institute for Comp. Sciences**

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

University of Tennessee

1

Outline

- **Case Studies**
- **Linpack**
 - Top 500 tests
 - Characterizing computation to communication
- **MPI Benchmarks**

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

University of Tennessee

2

Cases Studies

- **Periodic Anderson Model (PAM)**
 - Performance debugging
- **Crystallography and NMR System (CNS)**
 - Optimization/parallelization of serial code
- **Hartree-Fock Calculations using THO states (Ca)**
 - Using the right compiler/linker

PAM

- **Simulates the 2-dimensional Periodic Anderson model on a square lattice**
- **Implemented with MPI**
 - Mostly barrier, broadcast, and reduce
- **Some OpenMP throughout**
 - Not used during this study
- **BLAS calls (not many)**
 - Mostly `sger` and `cgemm`

PAM (cont.)

- **4 by 8-way LPAR run 5 times faster than 1 by 32-way non-LPAR run!!!**
 - Can't be communication
 - What can it be?
 - Memory contention, cache contention,...?
- **Usual suspects don't pan out**
 - No memory swapping
 - No (auto) threading
 - Linked to ESSL (not ESSL SMP)

PAM (cont.)

- **What next?**
 - Use HPM to get idea of cache usage
 - Use `-pg` to get profile
 - Evaluate stats to determine next step
- **HPM: use `poe hpmcount executable`**
 - To use a particular group, use `-g #`
 - Used default group
 - Showed code does more work as the number of tasks per node increases
 - Different stats for # of instructions, cache misses, etc.
 - But get correct results on every run

PAM (cont.)

- Profile will show where time is spent
- Compiled code with `-pg`
- Upon execution, produces `gmon.out.#` files
 - Use `gprof` to create one summary file:
`gprof executable gmon.out.* > gmon.sum`
- `gmon.sum` file pointed to `get_rnflt` routine which is part of the SPRNG library
 - Almost 40x longer when using 32 tasks on a 32-way node

PAM (cont.)

- Investigated user's SPRNG library
 - Most likely compiled incorrectly (did not compile for MPI usage)
 - Recompiled library
 - Some sort of locking problem/cache invalidation when using the non-MPI SPRNG library, worse as more tasks are used on a node
- Result
 - Decreased wall-time in all cases
 - One 32-way node is slightly faster than four 8-way LPAR nodes (as expected)

CNS

- **Crystallography and NMR System**
 - Serial code, many paths through it
- **Particular run on “small” problem takes**
 - ~2.5 days on 375MHz Power 3
 - 22 hours on 1.3 GHz Power 4
- **How to significantly decrease wall time?**
 - Produce a profile, tackle “hot” spots

CNS (cont.)

- **Recompiled code with `-pg`**
- **Reran code**
- **View results (from `gmon.out`) with `xprofiler`**

CNS (cont.)

69200 seconds

File	Code Display	Utility	Help			
%time	cumulative seconds	self seconds	calls	self ns/call	total ns/call	name
20.0	13812.69	13812.69	106895	129.22	129.22	.ffftab [14]
12.9	22733.89	8921.20	1777957074	0.01	0.01	._exp [17]
10.6	30057.55	7323.66	106895	68.51	70.54	.ffftes2 [23]
10.4	37228.20	7170.65				._account [26]
8.5	43112.45	5884.25	109407	53.78	136.19	.ffftc [13]
7.0	47936.86	4824.41	972045041	0.00	0.00	._cos [33]
6.4	52383.32	4446.46	972045041	0.00	0.00	._sin [34]
5.0	55873.49	3490.17	2129275244	0.00	0.00	.dpassb4 [35]
2.3	57442.77	1569.28	443329244	0.00	0.00	.dpassb3 [40]
2.1	58907.34	1464.57	1031317080	0.00	0.00	.dpassb5 [41]
1.5	59958.54	1051.20	-1	-1051200.00	-8495270.00	.dcffftb1 [19]
1.3	60891.87	933.33	54710	17.06	153.26	.ffft3c2 [20]
1.3	61811.92	920.95	-1	-920050.00	-920050.00	.dpassb2 [45]
1.3	62696.48	884.36	52195	16.95	153.15	.ffft3c2 [21]
1.1	63438.16	741.68	2086671971	0.00	0.00	.cheval [46]
0.9	64092.03	653.87	500	1307.74	12881.56	.xdoft3 [32]
0.9	64695.47	603.44	54700	11.03	11.03	.rhoini [52]
0.5	65056.78	361.31	2664948160	0.00	0.00	._log [68]
0.5	65404.16	347.38	352970	0.98	1.18	.embrep [65]
0.4	65705.16	301.00				.qincrement [72]
0.4	65952.60	247.44				._stack_pointer [76]
0.4	66196.86	244.26				.qincrement1 [77]

Search Engine: (regular expressions supported)

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

University of Tennessee

CNS (cont.)

File	Utility	Help	
%total	calls	function	
13.10%	2129275244	calls from .dcffftb1 [19] to .dpassb4 [35]	
9.84%	1600323875	calls from .ffftes2 [23] to ._log [68]	
9.40%	1527532349	calls from .xftup [8] to ._exp [17]	
8.25%	1341726387	calls from .xdomf2 [50] to .cheval [46]	
6.34%	1031317080	calls from .dcffftb1 [19] to .dpassb5 [41]	
5.90%	958987844	calls from .ffftc [13] to .dcfftb [18]	
4.58%	745146259	calls from .xdomf2 [54] to ._log [68]	
4.58%	744945584	calls from .xdomf2 [54] to .cheval [46]	
3.51%	571055500	calls from .xdoft3 [32] to ._cos [33]	
3.51%	571055500	calls from .xdoft3 [32] to ._sin [34]	
2.73%	443329244	calls from .dcffftb1 [19] to .dpassb3 [40]	
2.45%	397640762	calls from ._atan2 [97] to .scalb [117]	
1.93%	313162039	calls from ._pxldatn2 [95] to ._atan2 [97]	
1.42%	230803173	calls from .getnb [148] to .copyis [165]	
1.41%	228897135	calls from .embrep [65] to .getnb [148]	
0.92%	149200419	calls from .eangle2 [106] to ._acos [129]	
0.71%	115915348	calls from .etor [47] to ._acos [129]	
0.70%	114341658	calls from ._xlzppow [39] to ._log [68]	
0.70%	114341658	calls from ._xlzppow [39] to ._sin [34]	
0.70%	114341658	calls from ._xlzppow [39] to ._cos [33]	
0.70%	114341658	calls from ._xlzppow [39] to ._exp [17]	

Search Engine: (regular expressions supported)

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

University of Tennessee

CNS (cont.)

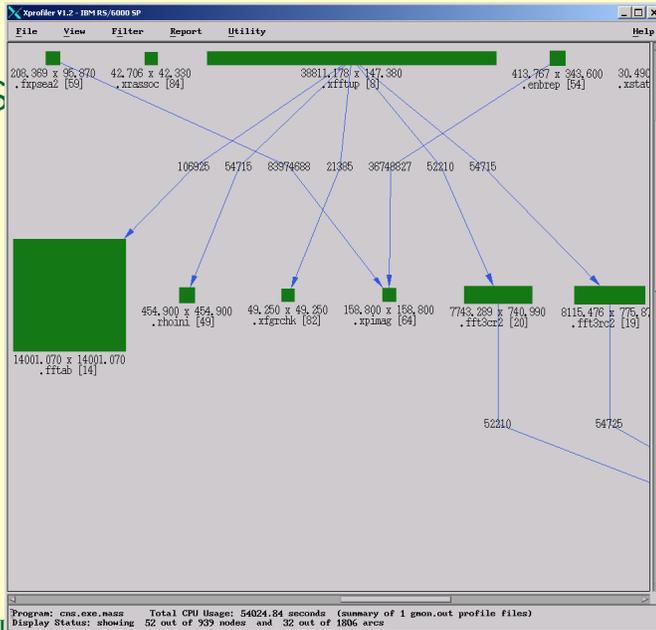
- The trig functions are used extensively
- Thus
 - re-link code with MASS library
 - re-run
- Result of linking to MASS library
 - Have cut run time by 25%
 - 19+ hours down to 15 hours

CNS (cont.)

54000 seconds

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
25.9	14001.07	14001.07	106925	130.94	130.94	.ffftab [14]
15.4	22302.22	8301.15	106925	77.64	77.64	.ffftes2 [18]
10.7	28062.12	5759.90	109437	52.63	134.12	.ffft3c [13]
6.8	31730.37	3668.25	2130997484	0.00	0.00	.dpassb4 [25]
6.5	35219.50	3489.13				.exp [26]
6.0	38441.40	3221.90				.cos [27]
3.4	40297.09	1855.69				.sincos [30]
3.3	42052.96	1755.87				._mcount [31]
3.1	43745.88	1692.92	444625244	0.00	0.00	.dpassb3 [32]
2.7	45216.71	1470.83	1031593560	0.00	0.00	.dpassb5 [34]
2.0	46288.87	1072.16	-1	-1072160.00	-8808300.00	.dcfft1 [16]
1.7	47193.01	904.14	-1	-904140.00	-904140.00	.dpassb2 [43]
1.4	47968.88	775.87	54725	14.18	148.30	.ffft3rc2 [19]
1.4	48714.57	745.69	2087271662	0.00	0.00	.cherval [45]
1.4	49455.56	740.99	52210	14.19	148.31	.ffft3c2 [20]
1.2	50084.82	629.26	500	1258.52	1929.11	.xdoft3 [40]
0.8	50539.72	454.90	54715	8.31	8.31	.rhoini [49]
0.6	50883.32	343.60	353000	0.97	1.17	.enbrep [54]
0.4	51089.44	206.12				.log [60]
0.3	51267.50	178.06	2662710	0.07	0.25	.xdodm1f2 [47]

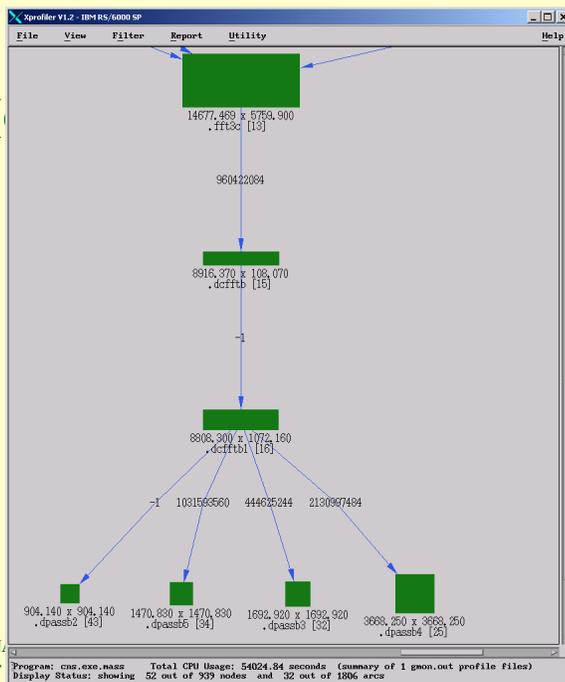
CNS



OAK RIDGE NATIONAL
U. S. DEPARTMENT OF ENERGY

University of Tennessee

CNS (



OAK RIDGE NATIONAL
U. S. DEPARTMENT OF ENERGY

University of Tennessee

CNS (cont.)

- **FFTs are called many times**
 - Number depends on grid
 - Called over 109k times in this example with
 - grid 192x200x250
 - Uses FFTPACK
 - The ESSL FFT does not “like” $200=2^3 \times 5^2$
 - Change how CNS sets up grid to be ESSL friendly

CNS (cont.)

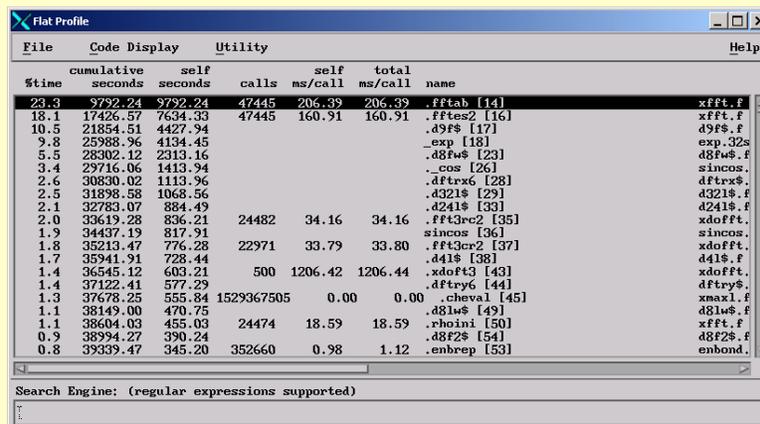
- **Grid size of 256x256x256**
 - Nearly doubles the # of entries
 - Approximately 4X the work
 - Only 64k FFT calls
 - Result is about the same wall clock time
 - 15 hours

CNS (cont.)

- **Grid size of 192x256x216**
 - Grid is only 15% larger
 - Only 49k FFT calls
 - Result is 2.5 hours less wall clock time
 - 12.5 hours
 - If use ESSL SMP with 2 threads, then
 - 10.5 hours runtime

CNS (cont.)

38040 seconds



File	Code	Display	Utility						
%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name			
23.3	9792.24	9792.24	47445	206.39	206.39	.fft4b [14]	xfft.f		
18.1	17426.57	7634.33	47445	160.91	160.91	.fft5e2 [16]	xfft.f		
10.5	21854.51	4427.94				.d9f\$ [17]	d9f\$.f		
9.8	25988.96	4134.45				.exp [18]	exp_32s		
5.5	28302.12	2313.16				.d8fw\$ [23]	d8fw\$.f		
3.4	29716.06	1413.94				.cos [26]	sincos.		
2.6	30830.02	1113.96				.dftax6 [28]	dftax\$.f		
2.5	31898.58	1068.56				.d321\$ [29]	d321\$.f		
2.1	32783.07	884.49				.d241\$ [33]	d241\$.f		
2.0	33619.28	836.21	24482	34.16	34.16	.fft3rc2 [35]	xdofft.		
1.9	34437.19	817.91				sincos [36]	sincos.		
1.8	35213.47	776.28	22971	33.79	33.80	.fft3e2 [37]	xdofft.		
1.7	35941.91	728.44				.d41\$ [38]	d41\$.f		
1.4	36545.12	603.21	500	1206.42	1206.44	.xdoft3 [43]	xdofft.		
1.4	37122.41	577.29				.dftxy6 [44]	dftxy\$.f		
1.3	37678.25	555.84	1529367505	0.00	0.00	.cheval [45]	xmax1.f		
1.1	38149.00	470.75				.d81w\$ [49]	d81w\$.f		
1.1	38604.03	455.03	24474	18.59	18.59	.rhoini [50]	xfft.f		
0.9	38994.27	390.24				.d8f2\$ [54]	d8f2\$.f		
0.8	39339.47	345.20	352660	0.98	1.12	.enbrep [53]	enbond.		

Search Engine: (regular expressions supported)

CNS (cont.)

- **Grid size of 192x224x216**
 - Grid is only 1% larger
 - Only 49k FFT calls
 - Result is 4.5 hours less wall clock time
 - 10.3 hours
 - If use ESSLSMP with 2 threads, then
 - ?

CNS (cont.)

- **Things yet to try:**
 - Vectorizing trig operations to use `massv`
 - Parallelize sections of code
 - OpenMP in `fftab` or `fftes2`?
 - Parallelize code with MPI
 - Nearly independent tasks
 - Large task of retrofitting code with MPI

Ca

- **Hartree-Fock-Bogoliubov calculations using Transformed HO (THO) states**
- **Boss-worker model**
- **Uses LAPI**
 - **Seems to hang**
 - **Job terminates when time runs out**
 - **What could be the problem? What to look for?**

Ca (cont.)

- **Problem:**
 - **Task 0 waiting for message from any other task**
 - **All other tasks except one are “done”**
 - **One task early on in run tested positive for a error condition**
 - **Called STOP, thus never entered section of code where task 0 needs one more task to send it a message**
- **Why do the other LAPI tasks continue until a hang?**
 - **Shouldn't all LAPI tasks exit when one does?**

Ca (cont.)

- **Culprit: signal-handling library**
 - Compiled with `mpx1f`
 - Should be compiled with `mpx1f_r`
- **Why?**
 - If not, then when one task terminates with `STOP`, the others continue
- **Result**
 - All LAPI tasks terminate when one calls `STOP`

Linpack

- **Observations from Top 500 tests**
- **Use of Linpack to characterize when one should use US or IP**

Linpack (cont.)

- **A benchmark to measure a computer's floating-point rate of execution**
 - Does not reflect the systems overall performance, no number can
 - Does reflect the performance of a dedicated system for solving a dense system of equations
 - Can be regarded as a correction of peak performance

Linpack (cont.)

- **Linpack has 3 forms**
 - N=100 test (serial, no modifications)
 - N=1000 test (serial, can use any math library)
 - **Cheetah gets 3.3 GF on one CPU**
 - 63.4% of peak
 - **Highly Parallel Linpack (HPL)**
 - Many configuration parameters to tune performance in input file
 - Used to determine Top 500 list
 - **Cheetah is #8 at 2.312 TF, 51.4% of peak**
 - Obtained with 216 MPI tasks each spawning 4 threads

HPL

- **HPL tuning parameters**
 - **Block size**
 - NB = 200
 - **Process grid**
 - 8x27
 - **Factorization algorithm**
 - Panel factorization occurs in one process column
 - **Panel broadcast algorithm**
 - Modified Increasing-Ring (MIR)

HPL Factorization algorithm

- **At a given iteration,**
 - **Do a panel factorization within a process column**
 - **Then a panel broadcast to other process columns**
 - Each column process is broadcasting to its process row
 - **Then update the trailing submatrix**
 - apply pivots and broadcast to each row
 - replicate U solve

HPL: undocumented tuning

- **Hardwired configuration parameter**
 - Column or row ordering of process grid
 - Must edit `HPL_pddriver.c`
 - Changing to row made a huge impact
 - Not documented on HPL website
 - Passed on from one tester to another

Linpack results (months ago)

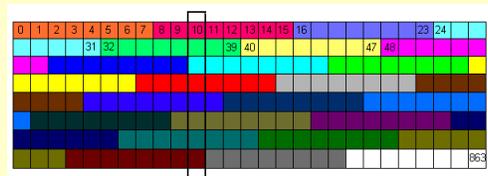
Device	Protocol	Ordering	Grid	Time	TFlops
csss	US	Row	8x27	6003	2.312
csss	US	Row	9x24	6198	2.237
css0	US	Row	8x27	6712	2.066
csss	US	Column	8x27	7893	1.757
csss	IP	Row	8x27	6192	2.239
en3	IP	Row	8x27	7483	1.853

HPL: Observations

- **Optimal process grid was 8x27**
 - 9x24 was almost as good
 - Column length is short (8 or 9)
 - Each column spread over nodes
 - Column panel factorization communicates over these nodes
 - Row operations across few nodes
 - Most communication via shared memory
 - 4 or 5 nodes in 8x27 case
 - 3 in 9x24 case

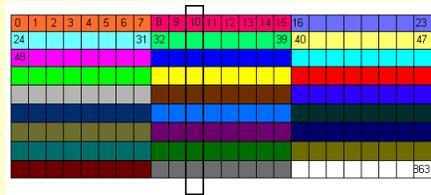
HPL: Observations (cont.)

- **Why is 8x27 best?**
 - Panel broadcasts via increasing-ring algorithm do not make heavy use of switch at any given time
 - Staggered assignment of virtual process grid to physical processor grid
 - Even load of messages over switch for duration
 - Column broadcasts have no need to be in sync



HPL: Observations (cont.)

- **Why is 9x24 not as good?**
 - Column broadcasts implicitly in lock-step
 - But no benefit because no barriers
 - Irregular use of switch
 - Floods switch every so often, then nothing



HPL: Conclusion

- **Short-wide process grid is best**
 - column communication is over a few nodes
- **8x27 grid compared to 9x24 leads to more balanced network traffic**

Linpack: communication vs. computation

- **Try to characterize what is best to use (US or IP) for relative computation and communication loads with HPL**

Motivation

- **Evidence of extremely poor performance across multiple 32-way nodes**
 - IP faster than US
 - Use fewer processors/node and observe shorter wall-clock time
 - Cyclic ordering of tasks better than block

This is expected right?

- **The current hardware supports**
 - 2 adapters per node
 - Does not matter if 4-way or 32-way
 - More bandwidth per processors for LPARs
- **Therefore one expects**
 - Better multi-node performance if using LPARs compared to 32-way nodes

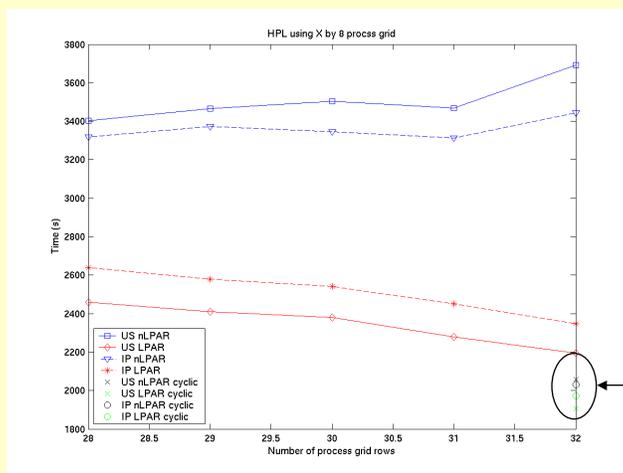
But...

- **For certain simple examples**
 - Using LPAR in US protocol with default ordering of tasks does not yield best performance
- **Furthermore, for apps that define a process grid (e.g., ScaLAPACK codes)**
 - Shape of process grid is of vital importance

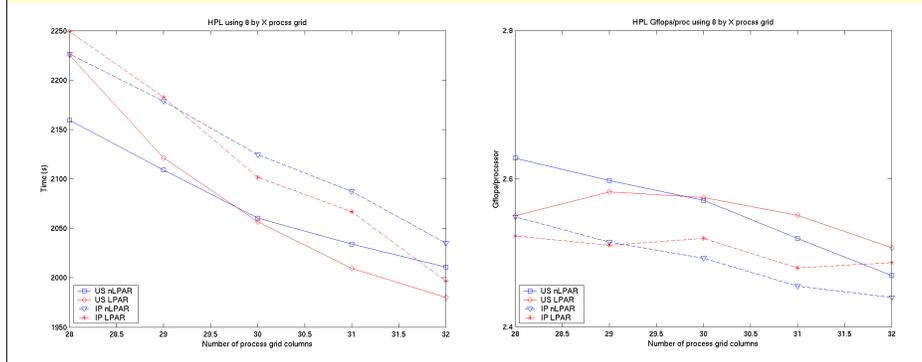
Example: HPL with 256 MPI tasks

- On 32-way nodes in IP mode with 32x8 grid
 - IP yields consistently faster timings than US by 7%
 - Cyclic ordering yields dramatically faster timings than block ordering (40%)
 - Accomplished via a script that creates a `task_geometry` for LoadL
- Although would typically use 8x32 grid, this shows, for apps with high communication volume
 - IP may be better than US
 - Cyclic ordering may be better than default block

HPL: #x8 process grid



HPL: 8x# process grid



OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

University of Tennessee

43

Observations

- **What does it mean?**
 - For apps with high-volume fixed communication patterns, IP and/or cyclic ordering may yield better performance
- **What is high volume?**
 - Communication accounts for 40% or more of runtime
- **For**
 - Medium-volume ~ 25 to 33%; use LPARs and US
 - Low-volume < 20%; use US
- **Overall IP is not that bad, and in some cases better**

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

University of Tennessee

44

Observations (cont.)

- **Comparisons weren't completely fair**
 - Problem size was fixed, thus not optimal for all tests
 - Each LPAR was completely used
 - Non-LPARs were only fully used for 32-way tests
- **Yet,**
 - LPARs will probably always be fully used for multi-node jobs where as non-LPARs may not be
 - **So tests are realistic**
 - There is an efficiency cross-over from many fully used LPARs to several mostly used non-LPARs
 - **Unexpected!**

MPI Benchmarks

- **PALLAS MPI Benchmark Suite V2.2**
- **256 processes**

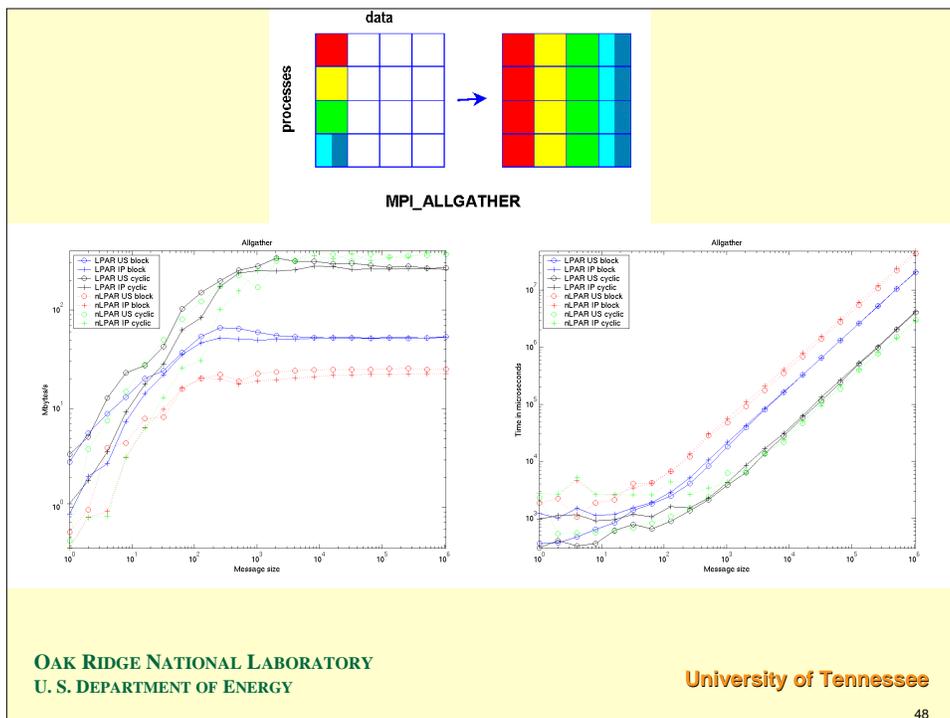
MPI routines tested

- Allgather
- Allgatherv
- Allreduce
- Alltoall
- Bcast
- Exchange
- Reduce
- Reduce_Scatter

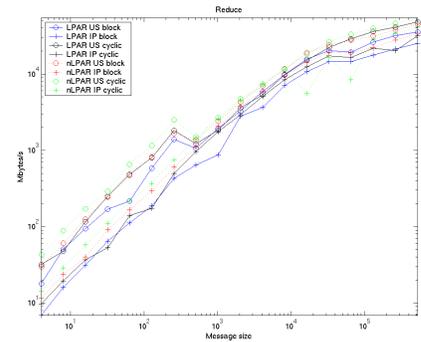
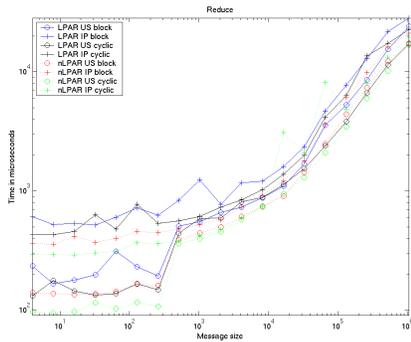
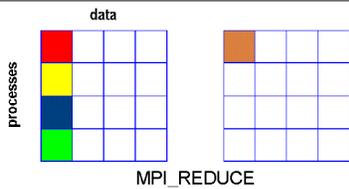
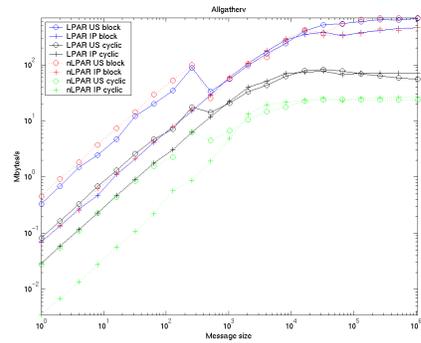
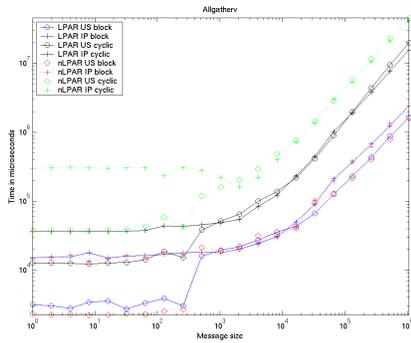
OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

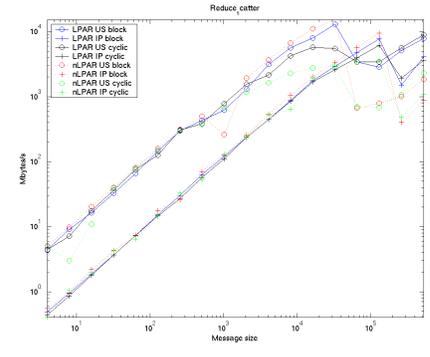
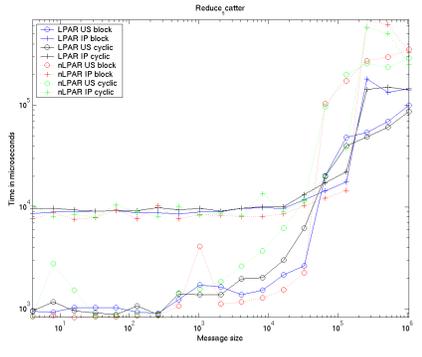
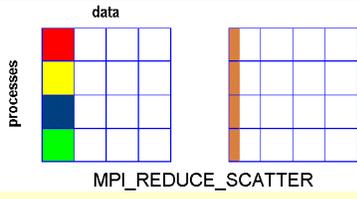
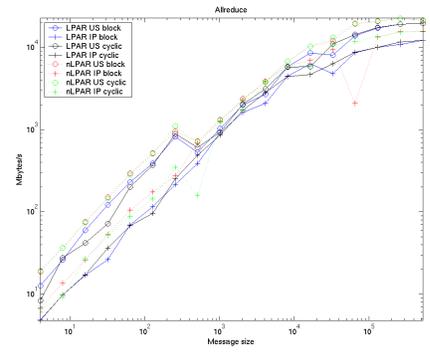
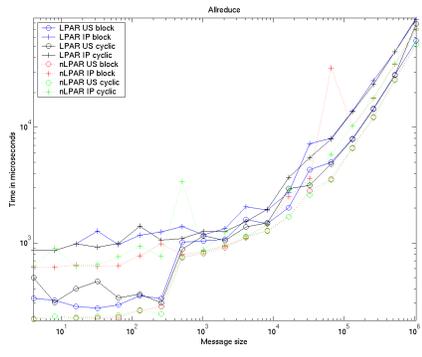
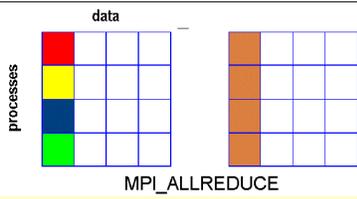
University of Tennessee

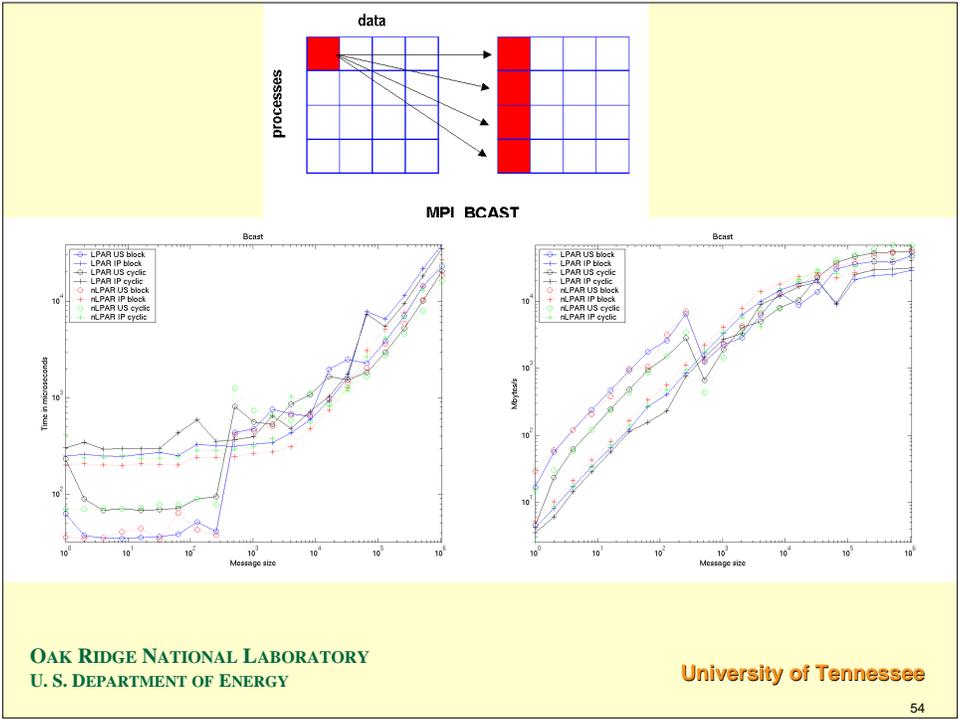
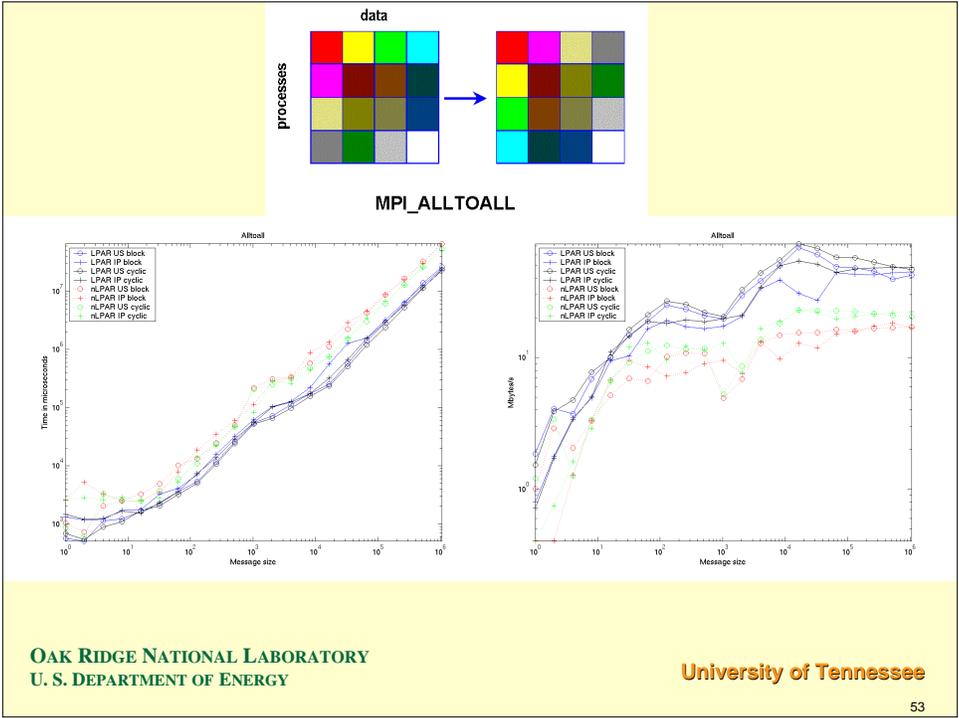
47



Allgather







Recommendations from IBM

- **US for short messages when low-latency is needed**
 - True for MPI Benchmarks
- **IP adequate for large message**
- **IP for large #s of tasks that**
 - communicate simultaneously, and
 - send large messages
- **IP worse for Barriers or Allreduce with small data**
 - Allreduce benchmark verifies this
- **Utilizing fewer switch windows is a benefit**
 - Use threads within a node
 - Linpack Top500 run is an example

Additional recommendations

- **IP for high-volume communication**
 - Also consider cyclic ordering
 - high volume = communication accounts for 40% or more of runtime
- **Process grid size can be very important (short & wide is better)**