

Porting, Scaling, and Optimization on Cray XT Systems

Jeff Larkin
larkin@cray.com

These slides are available at
<http://www.slideshare.net/jefflarkin/may2010-hexcoreopt>

COMPILING

Choosing a Compiler

- Try every compiler available to you, there's no "best"
 - ✿ PGI, Cray, Pathscale, Intel, GCC are all available, but not necessarily on all machines
- Each compiler favors certain optimizations, which may benefit applications differently
- Test your answers carefully
 - ✿ Order of operation may not be the same between compilers or even compiler versions
 - ✿ You may have to decide whether speed or precision is more important to you

Choosing Compiler Flags

■ PGI

- ✿ -fast –Mipa=fast
- ✿ man pgf90; man pgcc; man pgCC

■ Cray

- ✿ <none, turned on by default>
- ✿ man crayftn; man craycc ; man crayCC

■ Pathscale

- ✿ -Ofast
- ✿ man eko (“Every Known Optimization”)

■ GNU

- ✿ -O2 / -O3
- ✿ man gfortran; man gcc; man g++

■ Intel

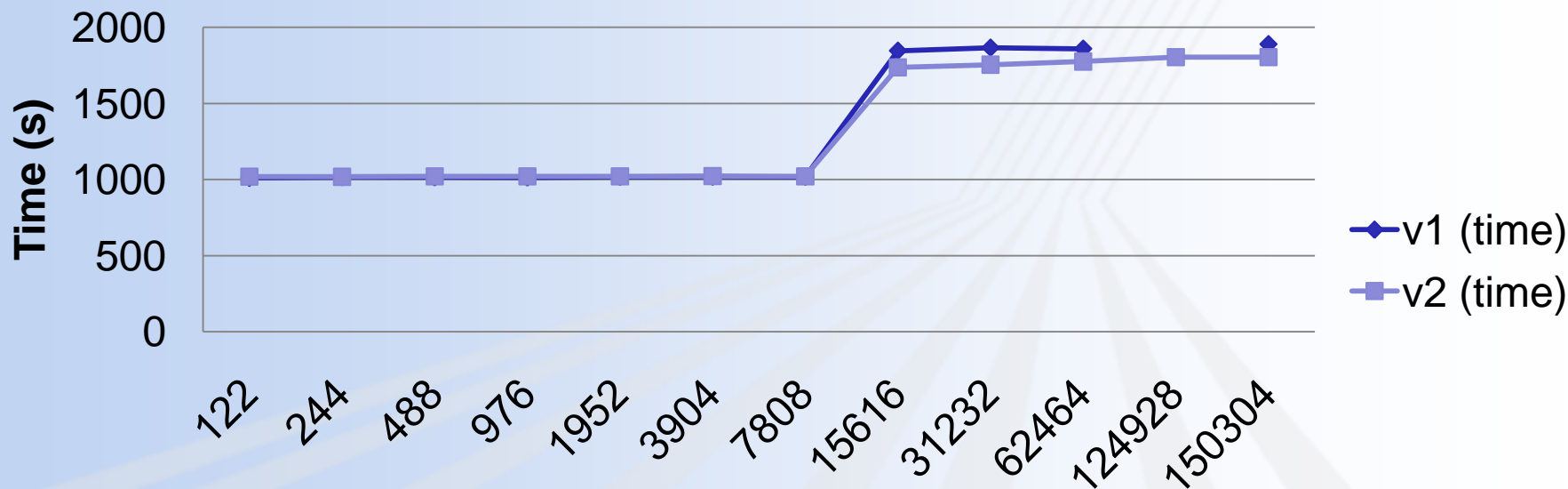
- ✿ -fast
- ✿ man ifort; man icc; man iCC

PROFILING AND DEBUGGING

Profile and Debug “at Scale”

- Codes don't run the same at thousands of cores as hundreds, 10s of thousands as thousands, or 100s of thousands as 10s
- Determine how many nodes you wish to run on and test at that size, don't test for throughput

Scaling (Time)



Profile and Debug with Real Science

- Choose a real problem, not a toy
- What do you want to achieve with the machine?
- How will you really run your code?
- Will you really run without I/O?

What Data Should We Collect?

■ Compiler Feedback

- ✿ Most compilers can tell you a lot about what they do to your code
 - ▶ Did your important loop vectorize?
 - ▶ Was a routine inlined?
 - ▶ Was this loop unrolled?
- ✿ It's just as important to know what the compiler didn't do
 - ▶ Some C/C++ loops won't vectorize without some massaging
 - ▶ Some loop counts aren't known at runtime, so optimization is limited
- ✿ Flags to know:
 - ▶ PGI: -Minfo=all -Mneginfo=all
 - ▶ Cray: -rm (Fortran) (or also -O[neg]msgs), -hlist=m (C) (or also -h[neg]msgs)
 - ▶ Pathscale: -LNO:simd_verbose=ON
 - ▶ Intel: -vec-report1

Compiler Feedback Examples: PGI

```
! Matrix Multiply
do k = 1, N
  do j = 1, N
    do i = 1, N
      c(i,j) = c(i,j) + &
        a(i,k) * b(k,j)
    end do
  end do
end do
```

24, Loop interchange
produces reordered loop
nest: 25,24,26

26, Generated an alternate
loop for the loop

Generated vector
sse code for the loop

Generated 2
prefetch instructions
for the loop

Compiler Feedback Examples: Cray

```

23.                                     ! Matrix Multiply
24.  ib-----<                         do k = 1, N
25.  ib ibr4-----<                     do j = 1, N
26.  ib ibr4 Vbr4--<                     do i = 1, N
27.  ib ibr4 Vbr4                         c(i,j) = c(i,j) + a(i,k) * b(k,j)
28.  ib ibr4 Vbr4-->                     end do
29.  ib ibr4----->                     end do
30.  ib----->                         end do

```

i - interchanged

b - blocked

r - unrolled

V - Vectorized

Compiler Feedback Examples: Intel

`mm.F90(14): (col. 3) remark: PERMUTED LOOP WAS VECTORIZED.`

`mm.F90(25): (col. 7) remark: LOOP WAS VECTORIZED.`

What Data Should We Collect?

■ Hardware Performance Counters

- ✿ Using tools like CrayPAT it's possible to know what the processor is doing
- ✿ We can get counts for
 - ▶ FLOPS
 - ▶ Cache Hits/Misses
 - ▶ TLB Hits/Misses
 - ▶ Stalls
 - ▶ ...
- ✿ We can derive
 - ▶ FLOP Rate
 - ▶ Cache Hit/Miss Ratio
 - ▶ Computational Intensity
 - ▶ ...

Gathering Performance Data

■ Use Craypat's APA

- ✿ First gather sampling for line number profile
 - ▶ Light-weight
 - ▶ Guides what should and should not be instrumented
- ✿ Second gather instrumentation (-g mpi,io,blas,lapack,math ...)
 - ▶ Hardware counters
 - ▶ MPI message passing information
 - ▶ I/O information
 - ▶ Math Libraries
 - ▶ ...

```
load module
make
pat_build -O apa a.out
Execute
pat_report *.xf
Examine *.apa
pat_build -O *.apa
Execute
```

```
Execute
pat_build -O *.apa
Examine *.apa
```

Sample APA File (Significantly Trimmed)

```
# You can edit this file, if desired, and use it
# -----
#       HWPC group to collect by default.
# -Drtenv=PAT_RT_HWPC=1 # Summary with TLB metrics.
# -----
#       Libraries to trace.
# -g mpi,math
# -----
# -w # Enable tracing of user-defined functions.
#     # Note: -u should NOT be specified as an additional option.
# 9.08% 188347 bytes
#       -T ratt_i_
# 6.71% 177904 bytes
#       -T rhsf_
# 5.61% 205682 bytes
#       -T ratx_i_
# 5.59% 22005 bytes
#       -T transport_m_computespeciesdiffflux_
```

CrayPAT Groups

- **biolib** Cray Bioinformatics library routines
- **blacs** Basic Linear Algebra communication subprograms
- **blas** Basic Linear Algebra subprograms
- **caf** Co-Array Fortran (Cray X2 systems only)
- **fftw** Fast Fourier Transform library (64-bit only)
- **hdf5** manages extremely large and complex data collections
- **heap** dynamic heap
- **io** includes stdio and sysio groups
- **lapack** Linear Algebra Package
- **lustre** Lustre File System
- **math** ANSI math
- **mpi** MPI
- **netcdf** network common data form (manages array-oriented scientific data)
- **omp** OpenMP API (not supported on Catamount)
- **omp-rtl** OpenMP runtime library (not supported on Catamount)
- **portals** Lightweight message passing API
- **pthread** POSIX threads (not supported on Catamount)
- **scalapack** Scalable LAPACK
- **shmem** SHMEM
- **stdio** all library functions that accept or return the FILE* construct
- **sysio** I/O system calls
- **system** system calls
- **upc** Unified Parallel C (Cray X2 systems only)

Useful API Calls, for the control-freaks

- Regions, useful to break up long routines
 - ✿ int PAT_region_begin (int id, const char *label)
 - ✿ int PAT_region_end (int id)
- Disable/Enable Profiling, useful for excluding initialization
 - ✿ int PAT_record (int state)
- Flush buffer, useful when program isn't exiting cleanly
 - ✿ int PAT_flush_buffer (void)

Sample CrayPAT HWPC Data

USER / rhsf_

Time%			74.6%	
Time			556.742885 secs	
Imb.Time			14.817686 secs	
Imb.Time%			2.6%	
Calls	2.3 /sec		1200.0 calls	
PAPI_L1_DCM	14.406M/sec		7569486532 misses	
PAPI_TLB_DM	0.225M/sec		117992047 misses	
PAPI_L1_DCA	921.729M/sec		484310815400 refs	
PAPI_FP_OPS	871.740M/sec		458044890200 ops	
User time (approx)	525.438 secs	1103418882813 cycles		94.4%Time
Average Time per Call			0.463952 sec	
CrayPat Overhead : Time	0.0%			
HW FP Ops / User time	871.740M/sec	458044890200 ops		10.4%peak (DP)
HW FP Ops / WCT	822.722M/sec			
Computational intensity	0.42 ops/cycle		0.95 ops/ref	
MFLOPS (aggregate)	1785323.32M/sec			
TLB utilization	4104.61 refs/miss		8.017 avg uses	
D1 cache hit,miss ratios	98.4% hits		1.6% misses	
D1 cache utilization (M)	63.98 refs/miss		7.998 avg uses	

CrayPAT HWPC Groups

- 0 Summary with instruction metrics
- 1 Summary with TLB metrics
- 2 L1 and L2 metrics
- 3 Bandwidth information
- 4 Hypertransport information
- 5 Floating point mix
- 6 Cycles stalled, resources idle
- 7 Cycles stalled, resources full
- 8 Instructions and branches
- 9 Instruction cache
- 10 Cache hierarchy
- 11 Floating point operations mix (2)
- 12 Floating point operations mix (vectorization)
- 13 Floating point operations mix (SP)
- 14 Floating point operations mix (DP)
- 15 L3 (socket-level)
- 16 L3 (core-level reads)
- 17 L3 (core-level misses)
- 18 L3 (core-level fills caused by L2 evictions)
- 19 Prefetches

MPI Statistics

- How many times are MPI routines called and with how much data?
- Do I have a load imbalance?
- Are my processors waiting for data?
- Could I perform better by adjusting MPI environment variables?

Sample CrayPAT MPI Statistics

MPI Msg Bytes	MPI Msg Count	MsgSz <16B	16B<= MsgSz <256B	256B<= MsgSz <4KB	4KB<= MsgSz <64KB	Experiment=1 Function Caller
3062457144.0	144952.0	15022.0	39.0	64522.0	65369.0	Total

3059984152.0	129926.0	--	36.0	64522.0	65368.0	mpi_isend_

1727628971.0	63645.1	--	4.0	31817.1	31824.0	MPP_DO_UPDATE_R8_3DV.in.MPP_DOMAINS_MOD
3						MPP_UPDATE_DOMAIN2D_R8_3DV.in.MPP_DOMAINS_MOD

4	1680716892.0	61909.4	--	--	30949.4	DYN_CORE.in.DYN_CORE_MOD
5						FV_DYNAMICS.in.FV_DYNAMICS_MOD
6						ATMOSPHERE.in.ATMOSPHERE_MOD
7						MAIN__
8						main

9	1680756480.0	61920.0	--	--	30960.0	pe.13666
9	1680756480.0	61920.0	--	--	30960.0	pe.8949
9	1651777920.0	54180.0	--	--	23220.0	pe.12549
=====						

I/O Statistics

- How much time is spent in I/O?
- How much data am I writing or reading?
- How many processes are performing I/O?

Understanding Cray Message Passing Toolkit (mpt)

COMMUNICATION

Understanding CPU Affinity

■ MPICH_CPUMASK_DISPLAY env variable

- Displays MPI process CPU affinity mask
- `export MPICH_CPUMASK_DISPLAY=1`

```
aprun -n 8 -N 8 -cc cpu ./mpi_exe  
[PE_0]: cpumask set to 1 cpu on nid00036, cpumask = 00000001  
[PE_1]: cpumask set to 1 cpu on nid00036, cpumask = 00000010  
[PE_2]: cpumask set to 1 cpu on nid00036, cpumask = 00000100  
[PE_3]: cpumask set to 1 cpu on nid00036, cpumask = 00001000  
[PE_4]: cpumask set to 1 cpu on nid00036, cpumask = 00010000  
[PE_5]: cpumask set to 1 cpu on nid00036, cpumask = 00100000  
[PE_6]: cpumask set to 1 cpu on nid00036, cpumask = 01000000  
[PE_7]: cpumask set to 1 cpu on nid00036, cpumask = 10000000
```

```
aprun -n 8 -N 8 -cc numa_node ./mpi_exe  
[PE_0]: cpumask set to 4 cpus on nid00036, cpumask = 00001111  
[PE_1]: cpumask set to 4 cpus on nid00036, cpumask = 00001111  
[PE_2]: cpumask set to 4 cpus on nid00036, cpumask = 00001111  
[PE_3]: cpumask set to 4 cpus on nid00036, cpumask = 00001111  
[PE_4]: cpumask set to 4 cpus on nid00036, cpumask = 11110000  
[PE_5]: cpumask set to 4 cpus on nid00036, cpumask = 11110000  
[PE_6]: cpumask set to 4 cpus on nid00036, cpumask = 11110000  
[PE_7]: cpumask set to 4 cpus on nid00036, cpumask = 11110000
```

MPT 4.0 (Released Q4 2009)

- Merge to ANL MPICH2 1.1
 - ✿ Support for the MPI 2.1 Standard (except dynamic processes)
- Binary-compatible with Intel MPI
- Additional MPI-IO Optimizations
- Improvements in Allgatherv at > 2K processors
 - ✿ 10X – 2000x improvement
- Improvements in Scatterv at > 2KB message size
 - ✿ 20-80% improvement
- Initial Gemini Support

Auto-Scaling MPI Environment Variables

- Key MPI variables that ***change*** their default values dependent on job size

MPICH_MAX_SHORT_MSG_SIZE	MPICH_PTL_UNEX_EVENTS
MPICH_UNEX_BUFFER_SIZE	MPICH_PTL_OTHER_EVENTS

- ✿ Higher scaling of MPT jobs with fewer tweaks to env variables
 - ✿ “Default” values are based on total number of ranks in job
 - ✿ See MPI man page for specific formulas used
- We don't always get it right
 - ✿ Adjusted defaults aren't perfect for all applications
 - ✿ Assumes a somewhat communication-balanced application
 - ✿ Users can always override the new defaults
 - ✿ Understanding and fine-tuning these variables may help performance

Cray MPI XT Portals Communications

■ Short Message **Eager** Protocol

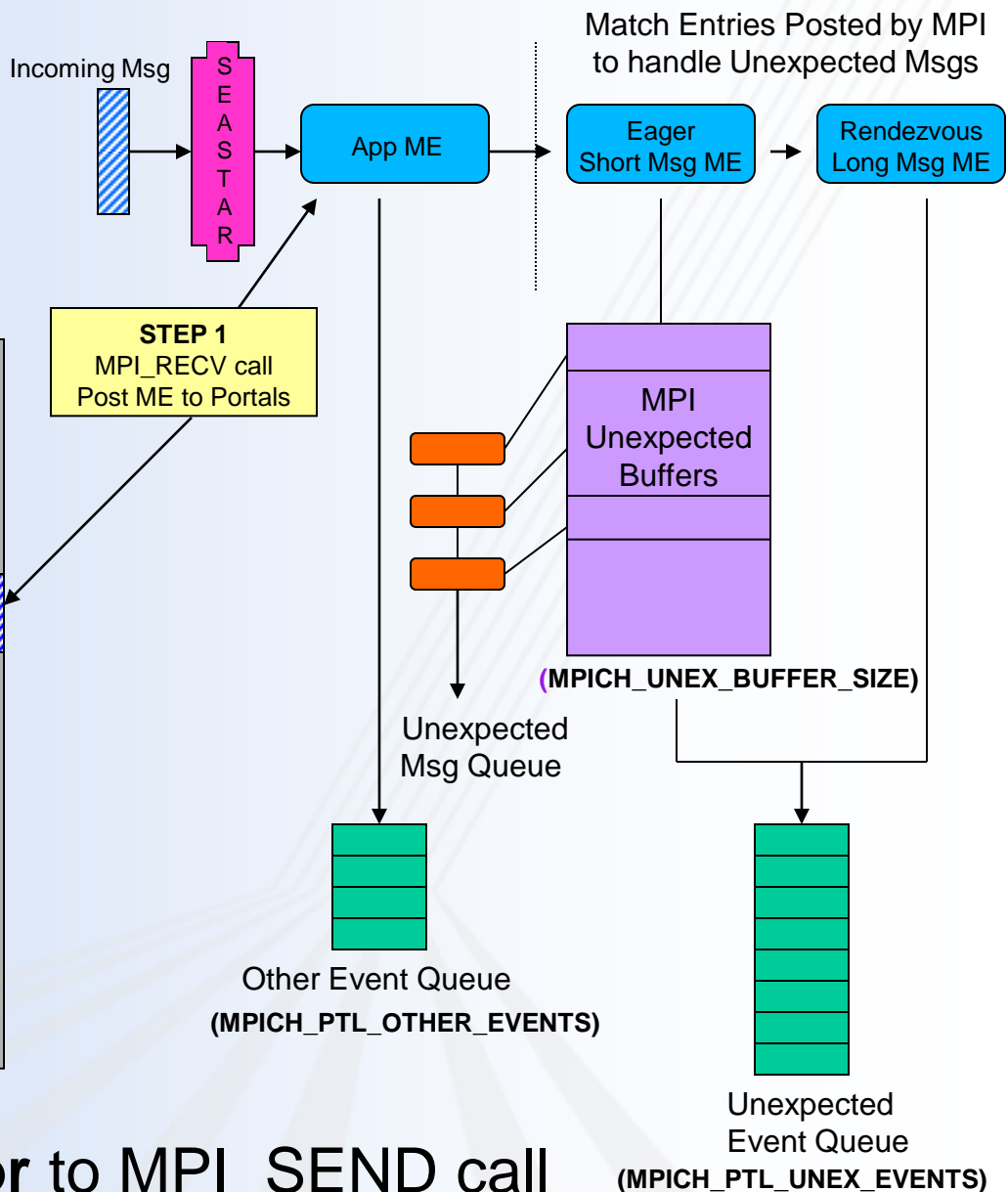
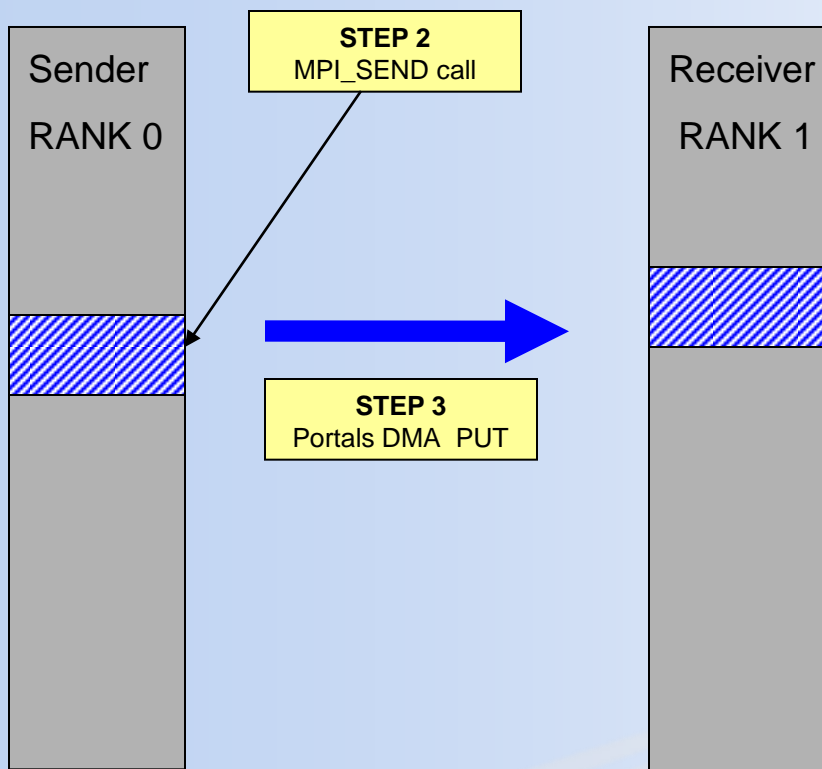
- ✿ The sending rank “pushes” the message to the receiving rank
- ✿ Used for messages **MPICH_MAX_SHORT_MSG_SIZE** bytes or less
- ✿ Sender assumes that receiver can handle the message
 - ▶ Matching receive is posted - or -
 - ▶ Has available event queue entries (**MPICH_PTL_UNEX_EVENTS**) and buffer space (**MPICH_UNEX_BUFFER_SIZE**) to store the message

■ Long Message **Rendezvous** Protocol

- ✿ Messages are “pulled” by the receiving rank
- ✿ Used for messages greater than **MPICH_MAX_SHORT_MSG_SIZE** bytes
- ✿ Sender sends small header packet with information for the receiver to pull over the data
- ✿ Data is sent only after matching receive is posted by receiving rank

MPT Eager Protocol

Data "pushed" to the receiver
(MPICH_MAX_SHORT_MSG_SIZE bytes or less)

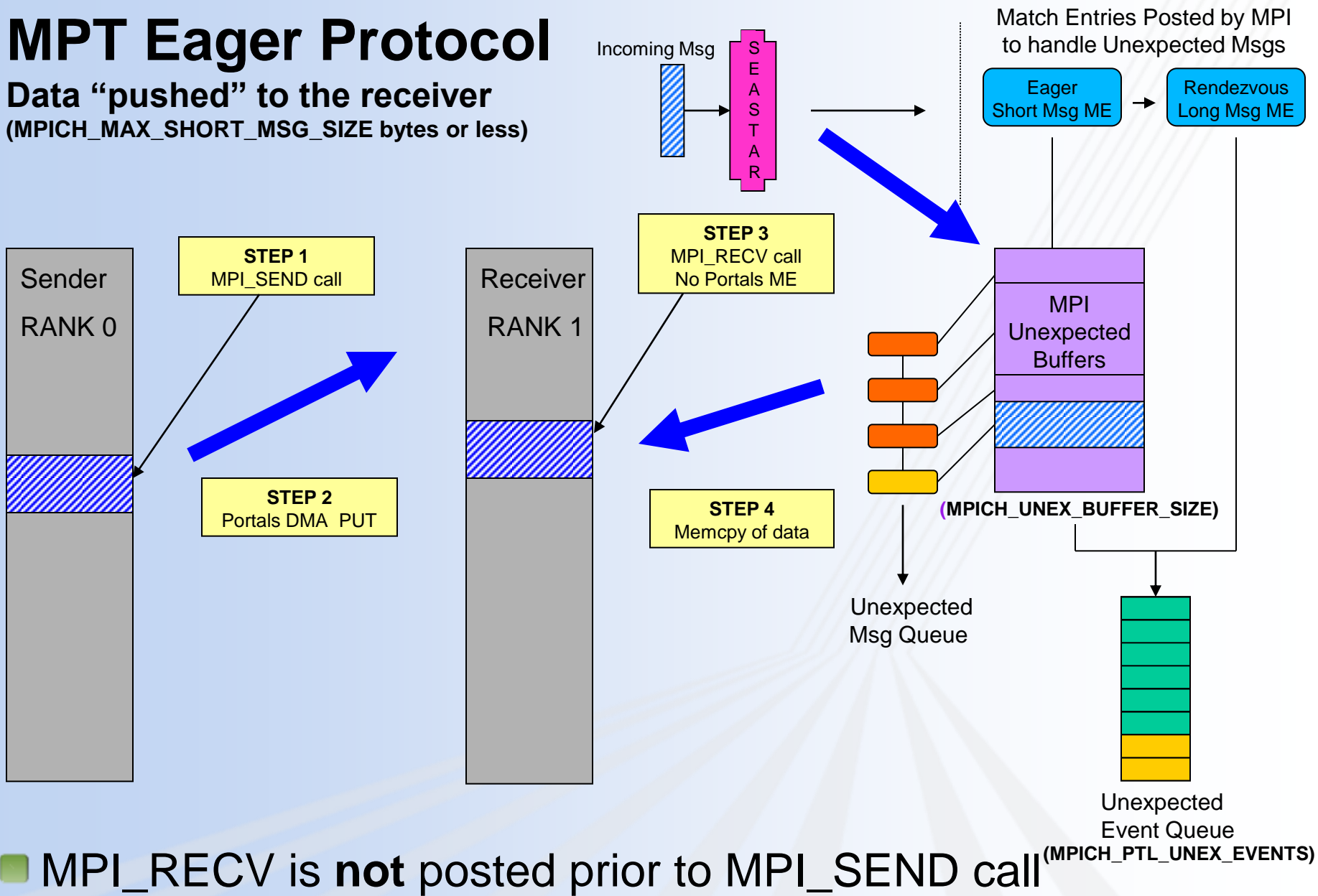


■ MPI_RECV is posted **prior** to MPI_SEND call

(MPICH_PTL_UNEX_EVENTS)

MPT Eager Protocol

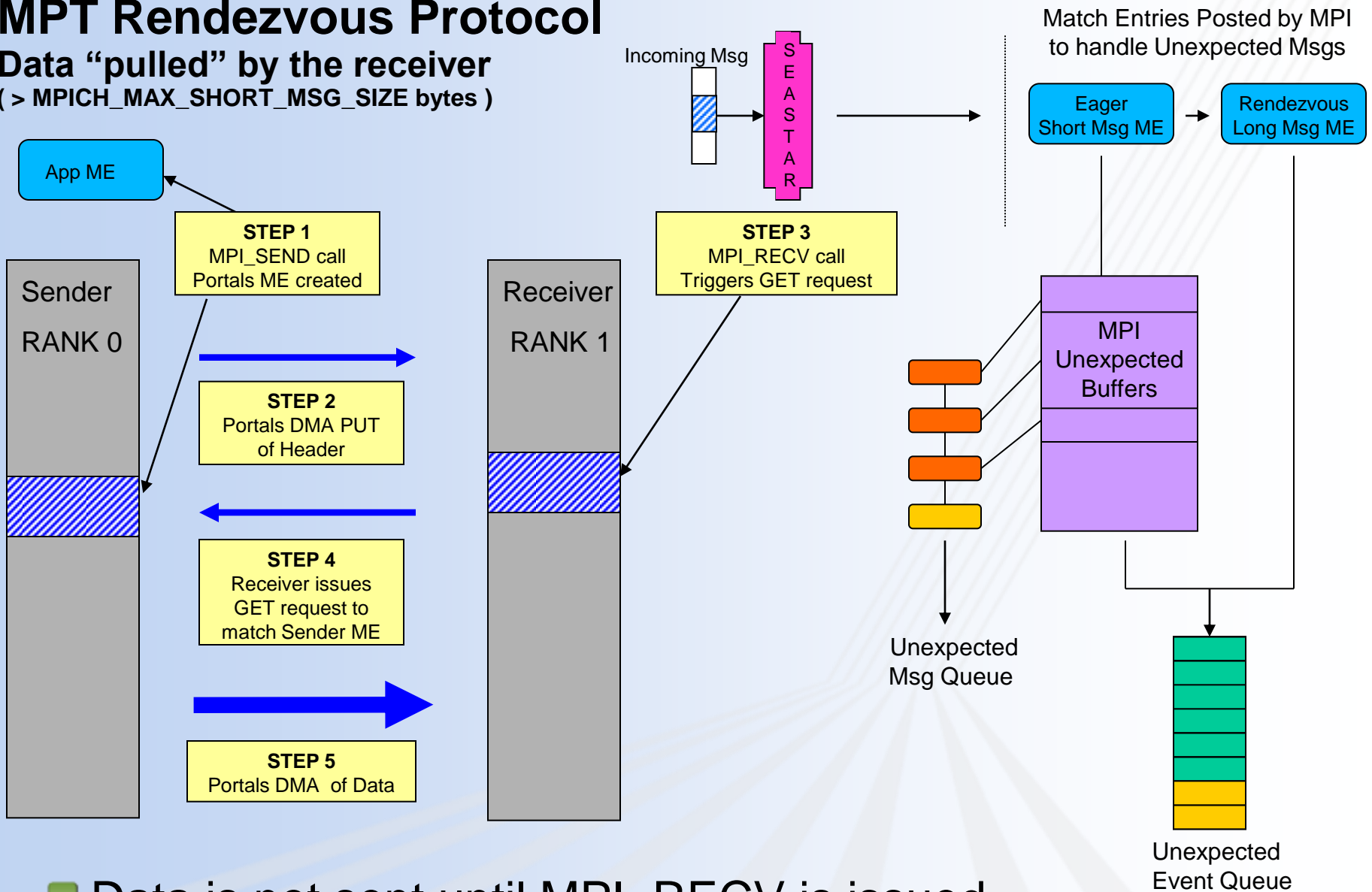
Data "pushed" to the receiver
(MPICH_MAX_SHORT_MSG_SIZE bytes or less)



■ MPI_RECV is **not** posted prior to MPI_SEND call

MPT Rendezvous Protocol

Data "pulled" by the receiver
 (> MPICH_MAX_SHORT_MSG_SIZE bytes)



■ Data is not sent until MPI_RECV is issued

Auto-Scaling MPI Environment Variables

- Default values for various MPI jobs sizes

MPI Environment Variable Name	1,000 PEs	10,000 PEs	50,000 PEs	100,000 PEs
MPICH_MAX_SHORT_MSG_SIZE (This size determines whether the message uses the Eager or Rendezvous protocol)	128,000 bytes	20,480	4096	2048
MPICH_UNEX_BUFFER_SIZE (The buffer allocated to hold the unexpected Eager data)	60 MB	60 MB	150 MB	260 MB
MPICH_PTL_UNEX_EVENTS (Portals generates <u>two</u> events for each unexpected message received)	20,480 events	22,000	110,000	220,000
MPICH_PTL_OTHER_EVENTS (Portals send-side and expected events)	2048 events	2500	12,500	25,000

Cray MPI Collectives

■ Our Collectives Strategy

- ✿ Improve performance over standard ANL MPICH2 algorithms
 - ▶ Tune for our interconnect(s)
- ✿ Work for any intra-communicator (not just MPI_COMM_WORLD)
- ✿ Enabled by default
- ✿ Can be selectively disabled via `MPICH_COLL_OPT_OFF`
 - ▶ `export MPICH_COLL_OPT_OFF=mpi_bcast,mpi_allreduce`
- ✿ Many have user-adjustable cross-over points (see man page)

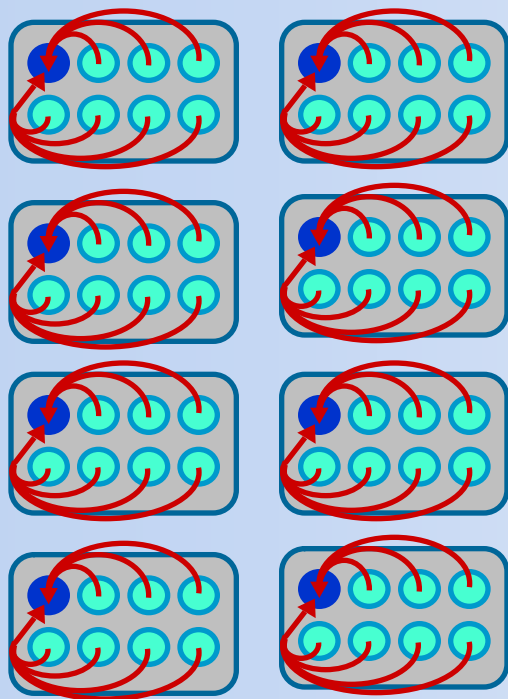
■ Cray Optimized Collectives

- ✿ `MPI_Allgather` (for small messages)
- ✿ `MPI_Alltoall` (changes to the order of exchanges)
- ✿ `MPI_Alltoallv` / `MPI_Alltoallw` (windowing algorithm)

■ Cray Optimized SMP-aware Collectives

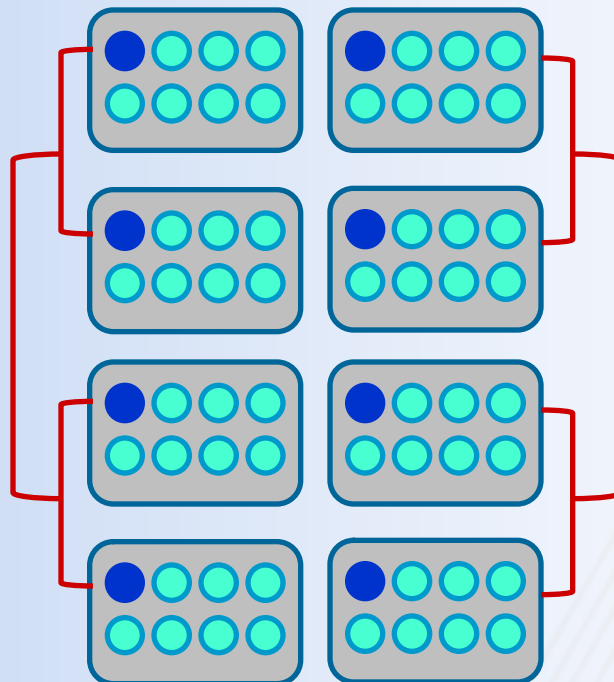
- ✿ `MPI_Allreduce`
- ✿ `MPI_Barrier`
- ✿ `MPI_Bcast` (new in MPT 3.1.1)
- ✿ `MPI_Reduce` (new in MPT 3.1.2)

SMP-aware Collectives – Allreduce Example



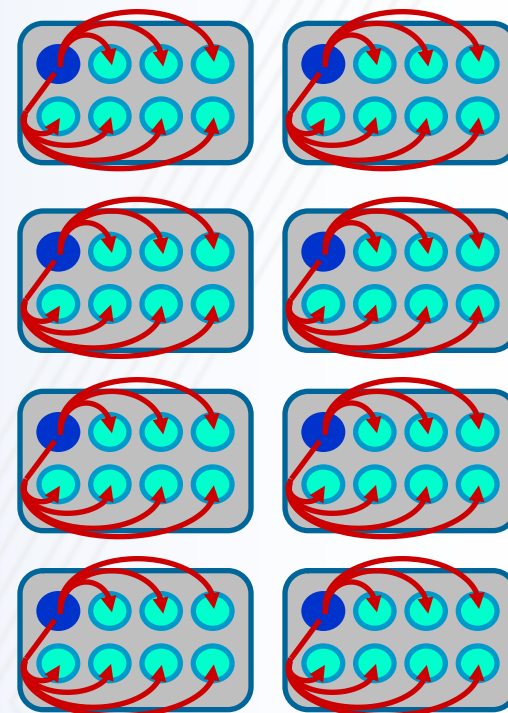
STEP 1

Identify Node-Captain rank.
Perform a local on-node
reduction to node-captain.
NO network traffic.



STEP 2

Perform an Allreduce with node-
captains only. This reduces the
process count by a factor of 8 on
XT5.



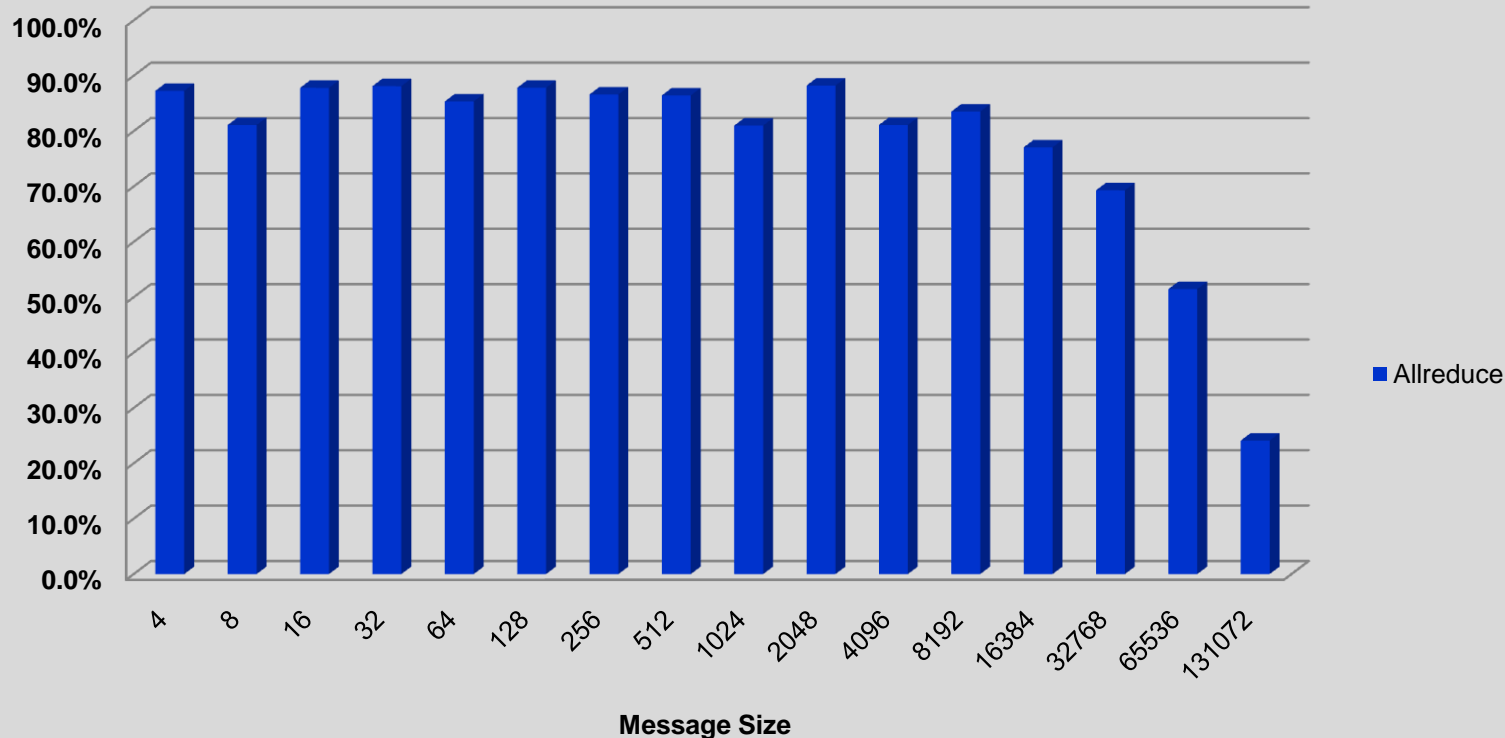
STEP 3

Perform a local on-node
bcast. NO network traffic.

Performance Comparison of MPI_Allreduce

Default vs MPICH_COLL_OPT_OFF=MPI_Allreduce

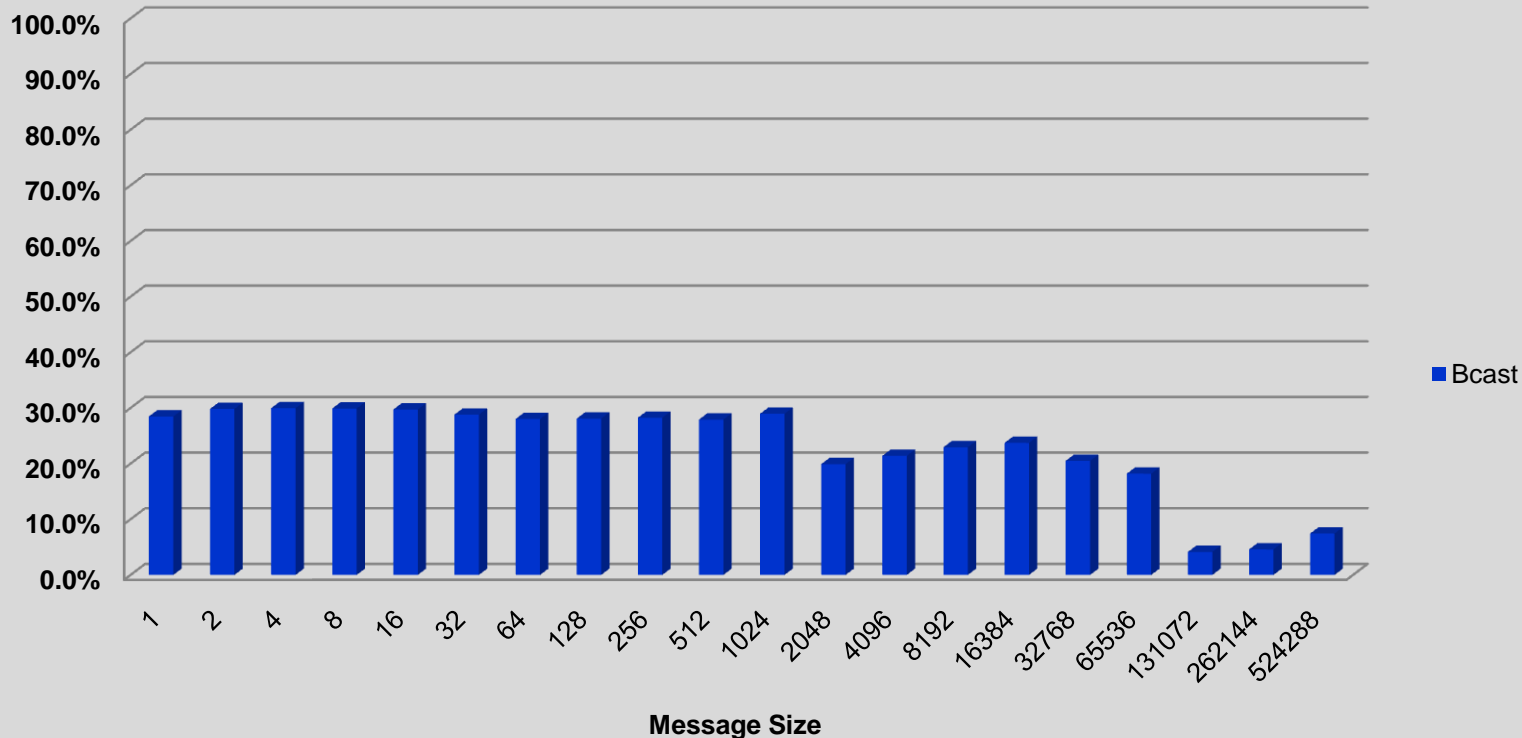
Percent Improvement of SMP-aware MPI_Allreduce (compared to MPICH2 algorithm) 1024 PEs on an Istanbul System



Performance Comparison of MPI_Bcast

Default vs MPICH_COLL_OPT_OFF=MPI_Bcast

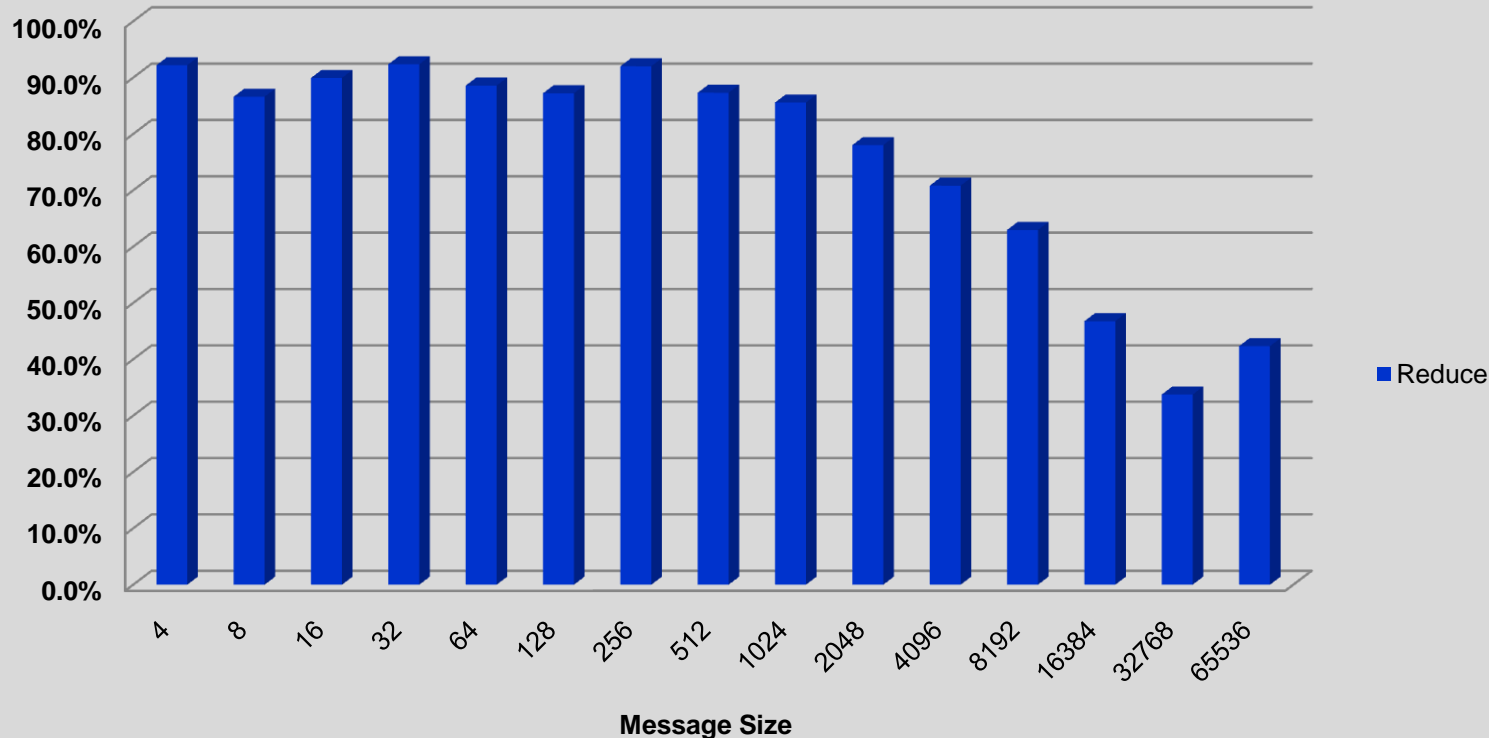
Percent Improvement of SMP-aware MPI_Bcast (compared to MPICH2 algorithm) 1024 PEs on an Istanbul System



Performance Comparison of MPI_Reduce

Default vs MPICH_COLL_OPT_OFF=MPI_Reduce

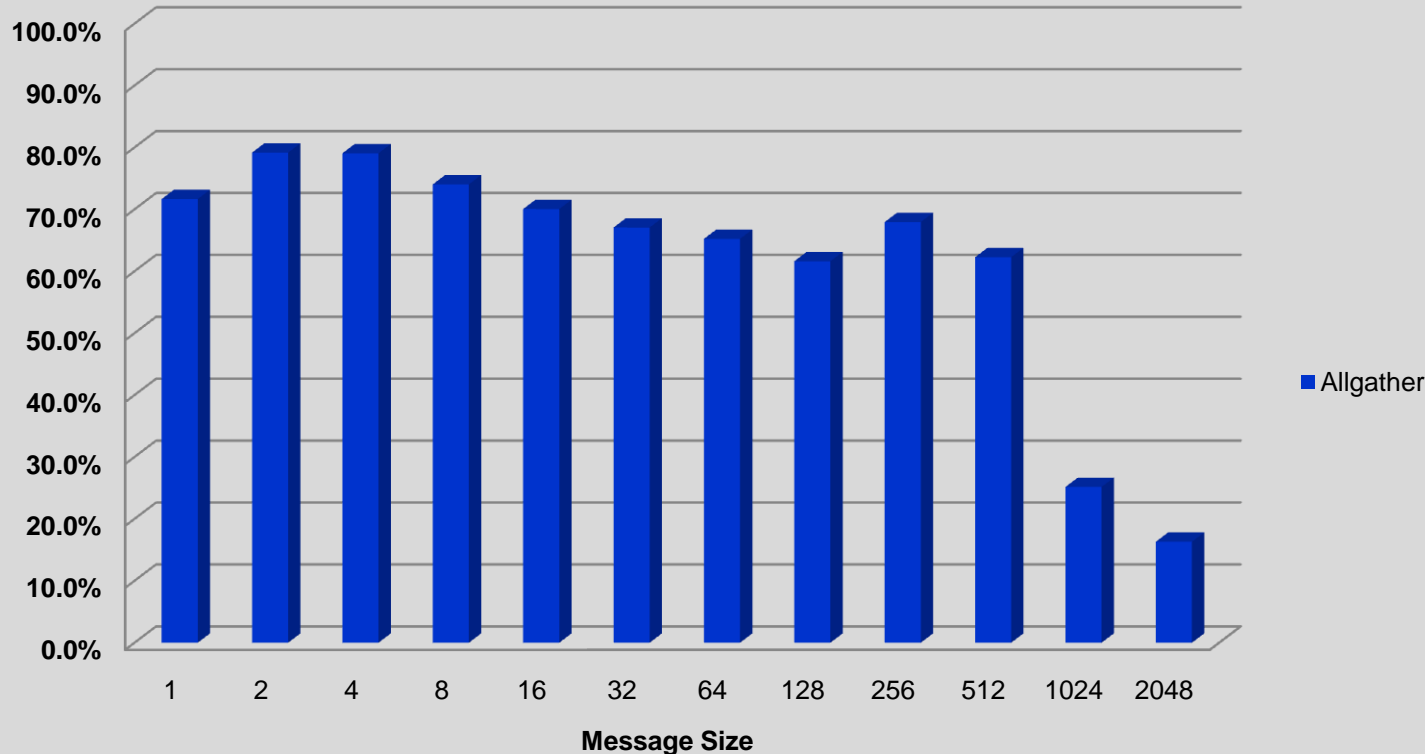
Percent Improvement of SMP-aware MPI_Reduce (compared to MPICH2 algorithm) 1024 PEs on an Istanbul System



Performance Comparison of MPI_Allgather

Default vs MPICH_COLL_OPT_OFF=MPI_Allgather

**Percent Improvement of Optimized MPI_Allgather
(compared to MPICH2 algorithm)
1024 PEs on an Istanbul System**



MPI-IO Improvements

■ MPI-IO collective buffering

⚙️ **MPICH_MPIIO_CB_ALIGN=0**

- ▶ Divides the I/O workload equally among all aggregators
- ▶ Inefficient if multiple aggregators reference the same physical I/O block
- ▶ Default setting in MPT 3.2 and prior versions

⚙️ **MPICH_MPIIO_CB_ALIGN=1**

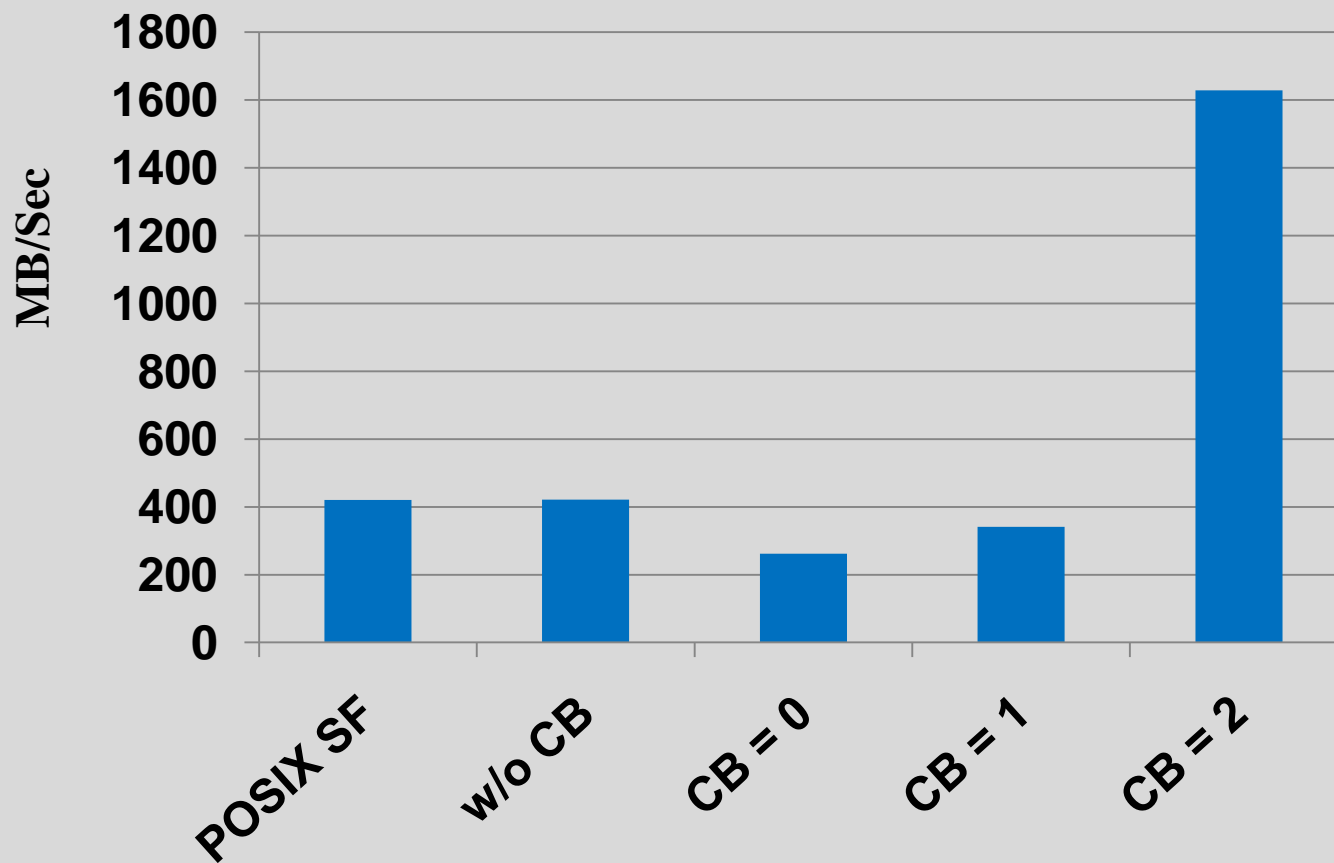
- ▶ Divides the I/O workload up among the aggregators based on physical I/O boundaries and the size of the I/O request
- ▶ Allows only one aggregator access to any stripe on a single I/O call
- ▶ Available in MPT 3.1

⚙️ **MPICH_MPIIO_CB_ALIGN=2**

- ▶ Divides the I/O workload into Lustre stripe-sized groups and assigns them to aggregators
- ▶ Persistent across multiple I/O calls, so each aggregator always accesses the same set of stripes and no other aggregator accesses those stripes
- ▶ Minimizes Lustre file system lock contention
- ▶ Default setting in MPT 3.3

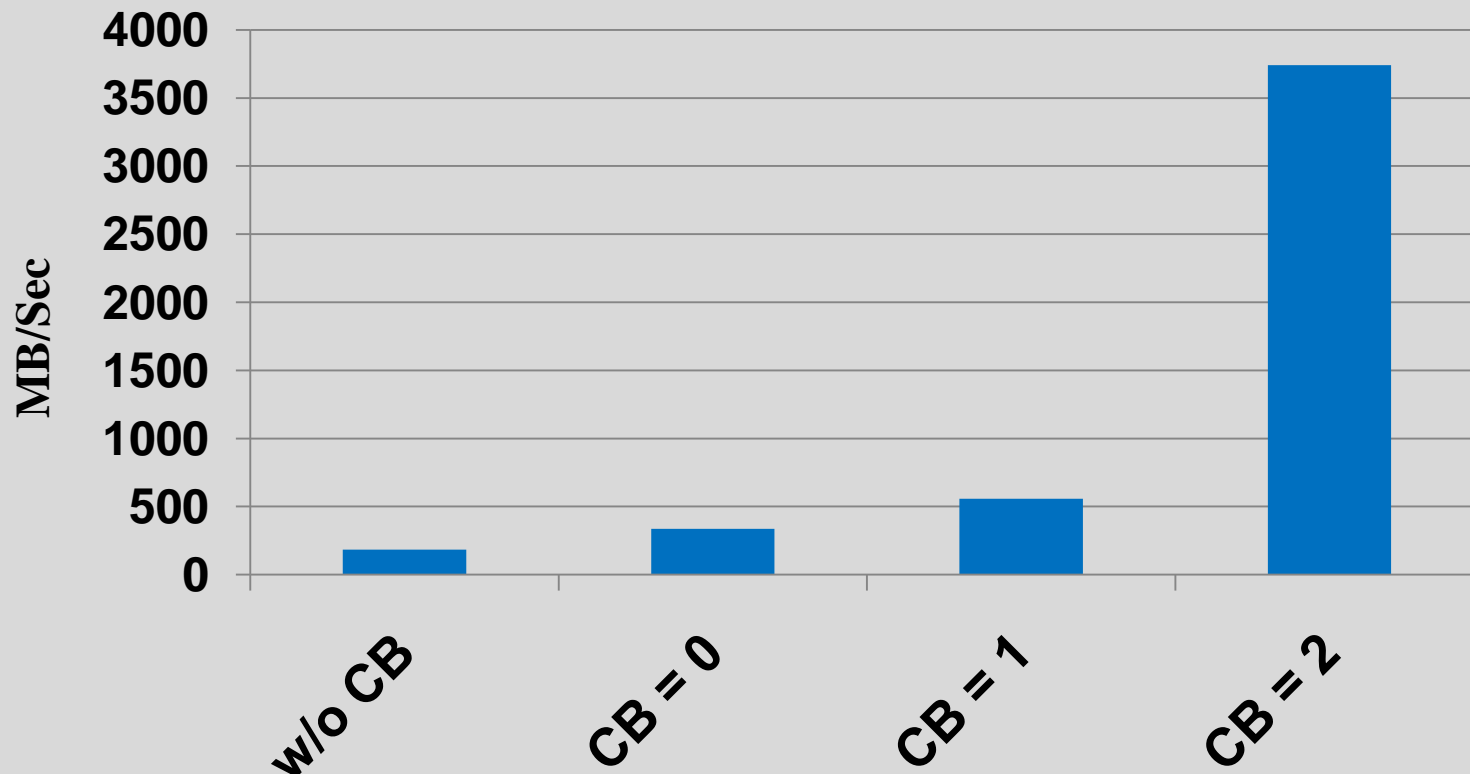
IOR benchmark 1,000,000 bytes

MPI-IO API , non-power-of-2 blocks and transfers, in this case blocks and transfers both of 1M bytes and a strided access pattern. Tested on an XT5 with 32 PEs, 8 cores/node, 16 stripes, 16 aggregators, 3220 segments, 96 GB file



HYCOM MPI-2 I/O

On 5107 PEs, and by application design, a subset of the PEs(88), do the writes. With collective buffering, this is further reduced to 22 aggregators (cb_nodes) writing to 22 stripes. Tested on an XT5 with 5107 PEs, 8 cores/node



OPTIMIZING YOUR CODE

Step by Step

1. Fix any load imbalance
2. Fix your hotspots
 1. Communication
 - Pre-post receives
 - Overlap computation and communication
 - Reduce collectives
 - Adjust MPI environment variables
 - Use rank reordering
 2. Computation
 - Examine the hardware counters and compiler feedback
 - Adjust the compiler flags, directives, or code structure to improve performance
 3. I/O
 - Stripe files/directories appropriately
 - Use methods that scale
 - MPI-IO or Subsetting

At each step, check your *answers* **and** *performance*.

Between each step, gather your data again.

Detecting a load imbalance

COMMUNICATION

Craypat load-imbalance data

Table 1: Profile by Function Group and Function

Time %	Time	Imb. Time	Imb. Time %	Calls	Experiment=1 Group Function PE='HIDE'
100.0%	1061.141647	--	--	3454195.8	Total
70.7%	750.564025	--	--	280169.0	MPI_SYNC
45.3%	480.828018	163.575446	25.4%	14653.0	mpi_barrier_(sync)
18.4%	195.548030	33.071062	14.5%	257546.0	mpi_allreduce_(sync)
7.0%	74.187977	5.261545	6.6%	7970.0	mpi_bcast_(sync)
15.2%	161.166842	--	--	3174022.8	MPI
10.1%	106.808182	8.237162	7.2%	257546.0	mpi_allreduce_
3.2%	33.841961	342.085777	91.0%	755495.8	mpi_waitall_
14.1%	149.410781	--	--	4.0	USER
14.0%	148.048597	446.124165	75.1%	1.0	main

Fixing a Load Imbalance

■ What is causing the load imbalance

✿ Computation

- ▶ Is decomposition appropriate?
- ▶ Would RANK_REORDER help?

✿ Communication

- ▶ Is decomposition appropriate?
- ▶ Would RANK_REORDER help?
- ▶ Are receives pre-posted?

■ OpenMP may help

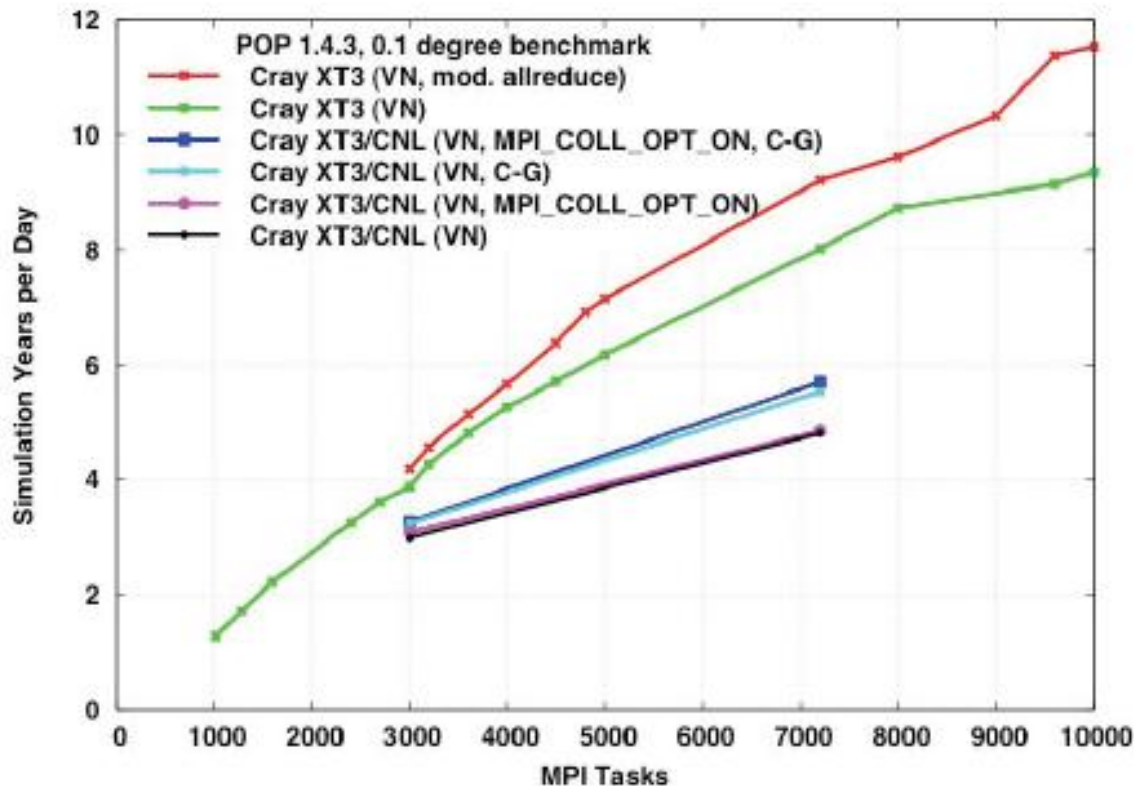
✿ Able to spread workload with less overhead

- ▶ Large amount of work to go from all-MPI to Hybrid
 - Must accept challenge to OpenMP-ize large amount of code

Pre-Post your Recvs

COMMUNICATION

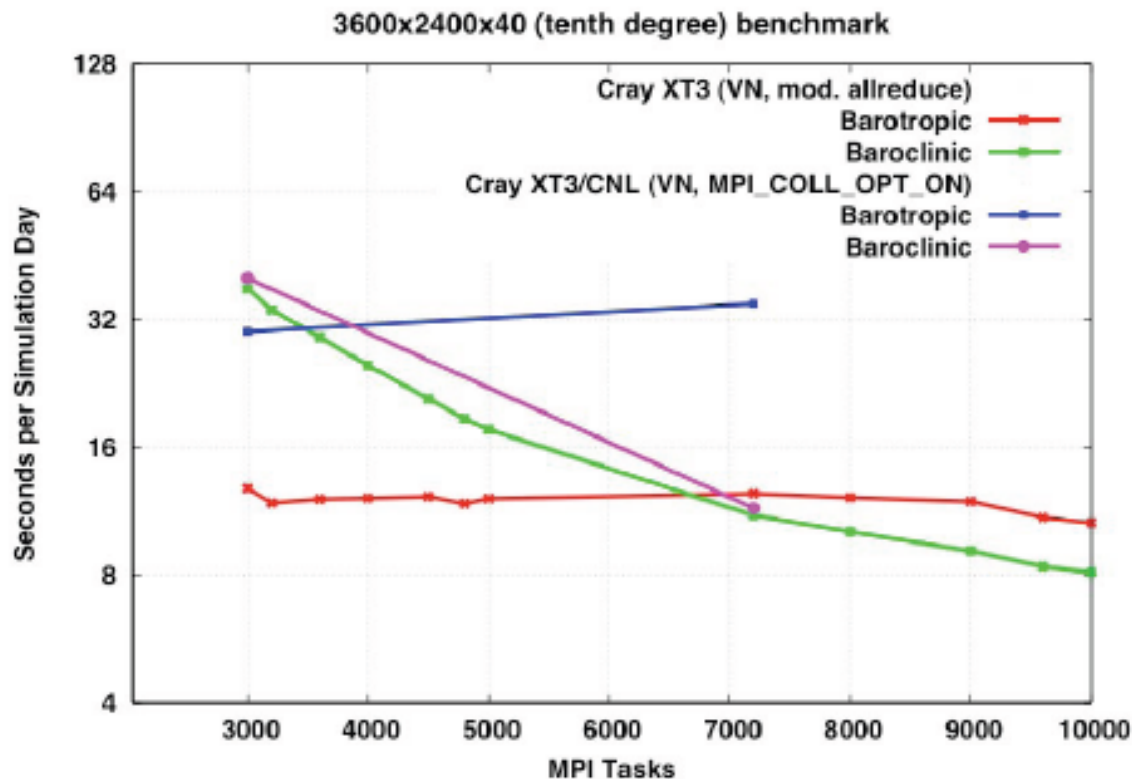
Suddenly last November ...



POP performance was not scaling well to large processor counts on Cray XT3 (using CNL). Even comparing Catamount results without C-G to CNL performance with C-G, CNL performance was much worse.



Catamount vs. CNL Phase Analysis



Performance difference is clearly in the Barotropic phase.

Hypotheses

1. First thought was that the `MPI_COLL_OPT_ON` version of `MPI_ALLREDUCE` was broken under CNL. Experiments with the “dual-core-aware” point-to-point implementation of allreduce, which was still in the code, did not improve performance. Further investigation showed that performance degradation was occurring when running in SN mode as well.
2. Second thought was that this was the dreaded OS jitter problem (for which allreduce is an excellent diagnostic tool). Added additional barriers and timers and found ...

Barotropic PCG Solver Logic

```

! calculate (PC)r
WORK1 = R*AOR ! use diag. precondition.

! update conjugate direction vector s
WORK0 = R*WORK1
  call t_startf("pcg_global_sum_c0")
eta1 = global_sum(WORK0,RCALCT)
  call t_stopf("pcg_global_sum_c0")

S = WORK1 + S*(eta1/eta0)

! compute As
  call t_startf("pcg_ninept_4_c")
call ninept_4(Q,A0,AN,AE,ANE,S)
  call t_stopf("pcg_ninept_4_c")

```

```

! compute next solution and residual
eta0 = eta1
WORK0 = Q*S
  call t_barrierf("sync_pcg_glb_sum_c1")
  call t_startf("pcg_global_sum_c1")
eta1 = eta0/global_sum(WORK0,RCALCT)
  call t_stopf("pcg_global_sum_c1")

X = X + eta1*S
R = R - eta1*Q

! test for convergence
...

```

Barotropic PCG Solver Performance

7200 MPI tasks, without timing barriers (process 0)

	Called	Wallclock	max	min
BAROTROPIC	1133	167.164764	1.315988	0.070891
pcg_global_sum_c0	139720	30.399334	1.170748	0.000111
pcg_ninept_4_c	139720	9.045540	0.002591	0.000033
pcg_global_sum_c1	139720	112.140366	0.006126	0.000062

7200 MPI tasks, with timing barriers (process 0)

	Called	Wallclock	max	min
BAROTROPIC	1133	188.724838	0.342617	0.089528
pcg_global_sum_c0	139720	26.993826	0.002908	0.000098
pcg_ninept_4_c	139720	8.519553	0.001952	0.000037
sync_pcg_glb_sum_c1	139720	111.815437	0.003291	0.000075
pcg_global_sum_c1	139720	26.179775	0.002964	0.000063

Hypotheses II

- The timing barrier is capturing all of the performance “degradation”, after which the allreduce behaves normally. If the problem were “OS jitter” occurring within the allreduce, then both the barrier and the allreduce would be impacted equally.
- The routine `ninept_4` is primarily a halo update, and the new hypothesis is that “ragged release” from the halo update is the source of the problems attributed to the allreduce. Next I tried alternate implementations of the halo update.

ninept_4 original

```
do j=jphys_b,jphys_e
  do i=iphys_b,iphys_e
    XOUT(i,j) = (9 pt. weighted sum)
  end do
end do
```

```
! fill buffers and send east-west
! boundary info
```

```
do n=1,num_ghost_cells
  do j=jphys_b,jphys_e
    buffer_east_snd(i)= ...
    buffer_west_snd(i)= ...
  end do
end do
```

```
call MPI_ISEND(buffer_east_snd, ...
call MPI_ISEND(buffer_west_snd, ...
```

```
! receive east-west boundary info and
! copy buffers into ghost cells
call MPI_RECV(buffer_west_rcv, ...
call MPI_RECV(buffer_east_rcv, ...
```

```
call MPI_WAITALL(2, ...
```

```
do n=1,num_ghost_cells
  do j=jphys_b,jphys_e
    XOUT(n,j) = ...
    XOUT(iphys_e+n,j) = ...
  end do
end do
```

```
! send north-south boundary info
call MPI_ISEND(XOUT(...
call MPI_ISEND(XOUT(...
```

```
! receive north-south boundary info
call MPI_RECV(XOUT(...
call MPI_RECV(XOUT(...
call MPI_WAITALL(2, ...
```

ninept_4 modified

```

! Prepost receive requests
call MPI_Irecv(buffer_west_rcv, ...
call MPI_Irecv(buffer_east_rcv, ...
call MPI_Irecv(XOUT(...
call MPI_Irecv(XOUT(...

do j=jphys_b,jphys_e
  do i=iphys_b,iphys_e
    XOUT(i,j) = (9 pt. weighted sum)
  end do
end do

! fill buffers and send east-west
! boundary info
do n=1,num_ghost_cells
  do j=jphys_b,jphys_e
    buffer_east_snd(i)= ...
    buffer_west_snd(i)= ...
  end do
end do

```

```

call MPI_ISEND(buffer_east_snd, ...
call MPI_ISEND(buffer_west_snd, ...

! receive east-west boundary info and
! copy buffers into ghost cells
call MPI_WAITALL(2, ...

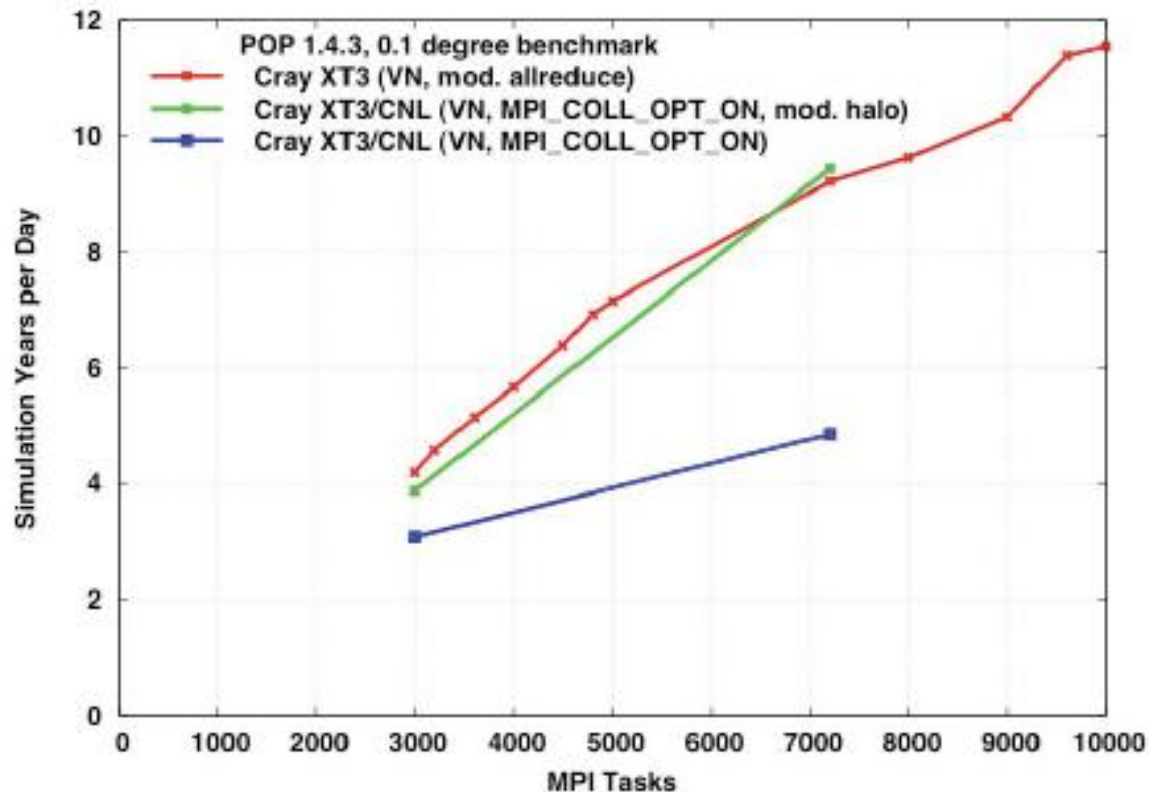
do n=1,num_ghost_cells
  do j=jphys_b,jphys_e
    XOUT(n,j) = ...
    XOUT(iphys_etn,j) = ...
  end do
end do

! send north-south boundary info
call MPI_ISEND(XOUT(...
call MPI_ISEND(XOUT(...

! receive north-south bddy info
call MPI_WAITALL(6, ...

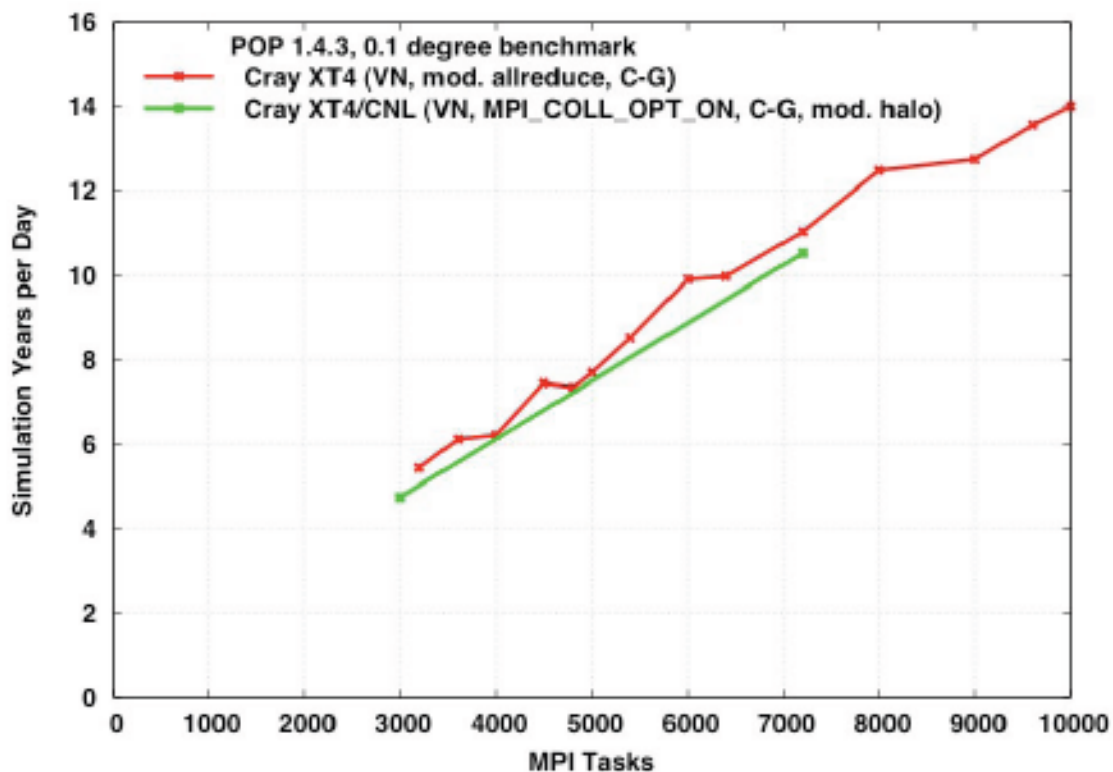
```

Modified Halo Update on XT3



Changing implementation of halo update, preposting receive requests, appears to eliminate performance degradation.

Modified Halo Update on XT4



Similar results hold with C-G algorithm and on Cray XT4.

Tweak the Library

COMMUNICATION

MPICH Performance Variable

MPICH_MAX_SHORT_MSG_SIZE

- Controls message sending protocol (Default: 128000 byte)
 - ✱ Message sizes \leq MSG_SIZE: Use EAGER
 - ✱ Message sizes $>$ MSG_SIZE: Use RENDEZVOUS
 - ✱ Increasing this variable may require that MPICH_UNEX_BUFFER_SIZE be increased
- Increase MPICH_MAX_SHORT_MSG_SIZE if App sends large msgs ($>128K$) and receives are pre-posted
 - ✱ Can reduce messaging overhead via EAGER protocol
 - ✱ Can reduce network contention
- Decrease MPICH_MAX_SHORT_MSG_SIZE if:
 - ✱ App sends msgs in 32k-128k range and receives not pre-posted

MPICH Performance Variable: MPICH_PTL_MATCH_OFF

- If set => Disables Portals matching
 - ✿ Matching happens on the Opteron
 - ✿ Requires extra copy for EAGER protocol
- Reduces MPI_Recv Overhead
 - ✿ Helpful for latency-sensitive application
 - ▶ Large # of small messages
 - ▶ Small message collectives (<1024 bytes)
- When can this be slower?
 - ✿ Pre-posted Receives can slow it down
 - ✿ When extra copy time longer than post-to-Portals time
 - ✿ For medium to larger messages (16k-128k range)

Custom Rank Ordering

- CrayPAT can provide suggestions based on metrics from a profiled run.
- If you understand your decomposition well enough, you may be able to map it to the network
 - ✦ We can help you find a better order based on your grid
- Is your “nearest neighbor” really your “nearest neighbor?”

COMPUTATION

Vectorization

- Stride one memory accesses
 - ✿ Loop on the “fast” indices
- No IF tests
- No subroutine calls
 - ✿ Inline
 - ✿ Module Functions
 - ✿ Statement Functions
- What is size of loop?
 - ✿ Vectorizing might not help with very small loops
- Unroll small loops
- Loop nest
 - ✿ Stride one on inside
 - ✿ Longest on the inside
- Increase computational intensity
 - ✿ $CI = (\text{vector flops}/\text{number of memory accesses})$

Bad Striding (64)

```
( 5)          COMMON A(8,8,IIDIM,8),B(8,8,iidim,8)

( 59)          DO 41090 K = KA, KE, -1
( 60)              DO 41090 J = JA, JE
( 61)                  DO 41090 I = IA, IE
( 62)                      A(K,L,I,J) = A(K,L,I,J) - B(J,1,i,k)*A(K+1,L,I,1)
( 63)          *      - B(J,2,i,k)*A(K+1,L,I,2) - B(J,3,i,k)*A(K+1,L,I,3)
( 64)          *      - B(J,4,i,k)*A(K+1,L,I,4) - B(J,5,i,k)*A(K+1,L,I,5)
( 65) 41090 CONTINUE
( 66)
```

PGI

59, Loop not vectorized: loop count too small

60, Interchange produces reordered loop nest: 61, 60

Loop unrolled 5 times (completely unrolled)

61, Generated vector sse code for inner loop

Pathscale

(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

(lp41090.f:62) Non-contiguous array "A(_BLNK__.0.0)" reference exists. Loop was not vectorized.

```
( 6)          COMMON AA (IIDIM, 8, 8, 8), BB (IIDIM, 8, 8, 8)

( 95)          DO 41091 K = KA, KE, -1
( 96)            DO 41091 J = JA, JE
( 97)              DO 41091 I = IA, IE
( 98)                AA (I, K, L, J) = AA (I, K, L, J) - BB (I, J, 1, K) * AA (I, K+1, L, 1)
( 99)            *   - BB (I, J, 2, K) * AA (I, K+1, L, 2) - BB (I, J, 3, K) * AA (I, K+1, L, 3)
(100)          *   - BB (I, J, 4, K) * AA (I, K+1, L, 4) - BB (I, J, 5, K) * AA (I, K+1, L, 5)
(101) 41091 CONTINUE
```

PGI

95, Loop not vectorized: loop count too small

96, Outer loop unrolled 5 times (completely unrolled)

97, Generated 3 alternate loops for the inner loop

Generated vector sse code for inner loop

Generated 8 prefetch instructions for this loop

Generated vector sse code for inner loop

Generated 8 prefetch instructions for this loop

Generated vector sse code for inner loop

Generated 8 prefetch instructions for this loop

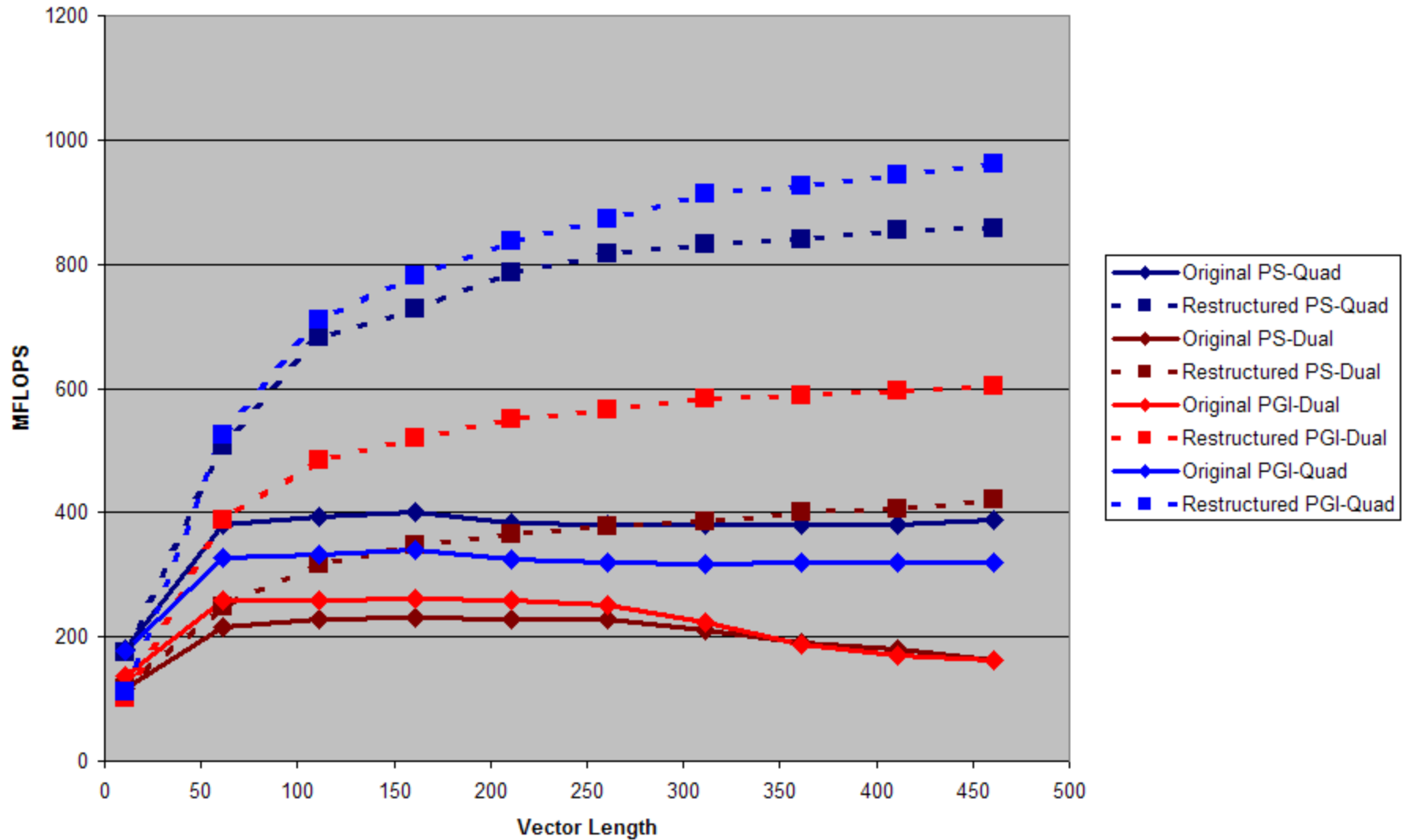
Generated vector sse code for inner loop

Generated 8 prefetch instructions for this loop

Pathscale

(lp41090.f:99) LOOP WAS VECTORIZED.

LP41090



C pointers

```
( 53) void mat_mul_daxpy(double *a, double *b, double *c, int rowa, int cola, int colb)
( 54) {
( 55)   int i, j, k;           /* loop counters */
( 56)   int rowc, colc, rowb; /* sizes not passed as arguments */
( 57)   double con;          /* constant value */
( 58)
( 59)   rowb = cola;
( 60)   rowc = rowa;
( 61)   colc = colb;
( 62)
( 63)   for(i=0;i<rowc;i++) {
( 64)     for(k=0;k<cola;k++) {
( 65)       con = *(a + i*cola + k);
( 66)       for(j=0;j<colc;j++) {
( 67)         *(c + i*colc + j) += con * *(b + k*colb + j);
( 68)       }
( 69)     }
( 70)   }
( 71) }
```

mat_mul_daxpy:

66, Loop not vectorized: data dependency

Loop not vectorized: data dependency

Loop unrolled 4 times

C pointers, rewrite

```
( 53) void mat_mul_daxpy(double* restrict a, double* restrict b, double*  
restrict c, int rowa, int cola, int colb)  
( 54) {  
( 55)   int i, j, k;           /* loop counters */  
( 56)   int rowc, colc, rowb; /* sizes not passed as arguments */  
( 57)   double con;           /* constant value */  
( 58)  
( 59)   rowb = cola;  
( 60)   rowc = rowa;  
( 61)   colc = colb;  
( 62)  
( 63)   for(i=0;i<rowc;i++) {  
( 64)     for(k=0;k<cola;k++) {  
( 65)       con = *(a + i*cola + k);  
( 66)       for(j=0;j<colc;j++) {  
( 67)         *(c + i*colc + j) += con * *(b + k*colb + j);  
( 68)       }  
( 69)     }  
( 70)   }  
( 71) }
```

C pointers, rewrite

66, Generated alternate loop with no peeling - executed if loop count ≤ 24

Generated vector sse code for inner loop

Generated 2 prefetch instructions for this loop

Generated vector sse code for inner loop

Generated 2 prefetch instructions for this loop

Generated alternate loop with no peeling and more aligned moves -
executed if loop count ≤ 24 and alignment test is passed

Generated vector sse code for inner loop

Generated 2 prefetch instructions for this loop

Generated alternate loop with more aligned moves - executed if loop
count ≥ 25 and alignment test is passed

Generated vector sse code for inner loop

Generated 2 prefetch instructions for this loop

- This can also be achieved with the PGI safe pragma and `-Msafept` compiler option or Pathscale `-OPT:alias` option

Nested Loops

```
( 47)          DO 45020 I = 1, N
( 48)            F(I) = A(I) + .5
( 49)            DO 45020 J = 1, 10
( 50)              D(I,J) = B(J) * F(I)
( 51)              DO 45020 K = 1, 5
( 52)                C(K,I,J) = D(I,J) * E(K)
( 53) 45020 CONTINUE
```

PGI

49, Generated vector sse code for inner loop

Generated 1 prefetch instructions for this loop

Loop unrolled 2 times (completely unrolled)

Pathscale

(lp45020.f:48) LOOP WAS VECTORIZED.

(lp45020.f:48) Non-contiguous array "C(_BLNK__.0.0)"
reference exists. Loop was not vectorized.

Rewrite

```
( 71)    DO 45021 I = 1,N
( 72)      F(I) = A(I) + .5
( 73) 45021 CONTINUE
( 74)
( 75)    DO 45022 J = 1, 10
( 76)      DO 45022 I = 1, N
( 77)        D(I,J) = B(J) * F(I)
( 78) 45022 CONTINUE
( 79)
( 80)    DO 45023 K = 1, 5
( 81)      DO 45023 J = 1, 10
( 82)        DO 45023 I = 1, N
( 83)          C(K,I,J) = D(I,J) * E(K)
( 84) 45023 CONTINUE
```

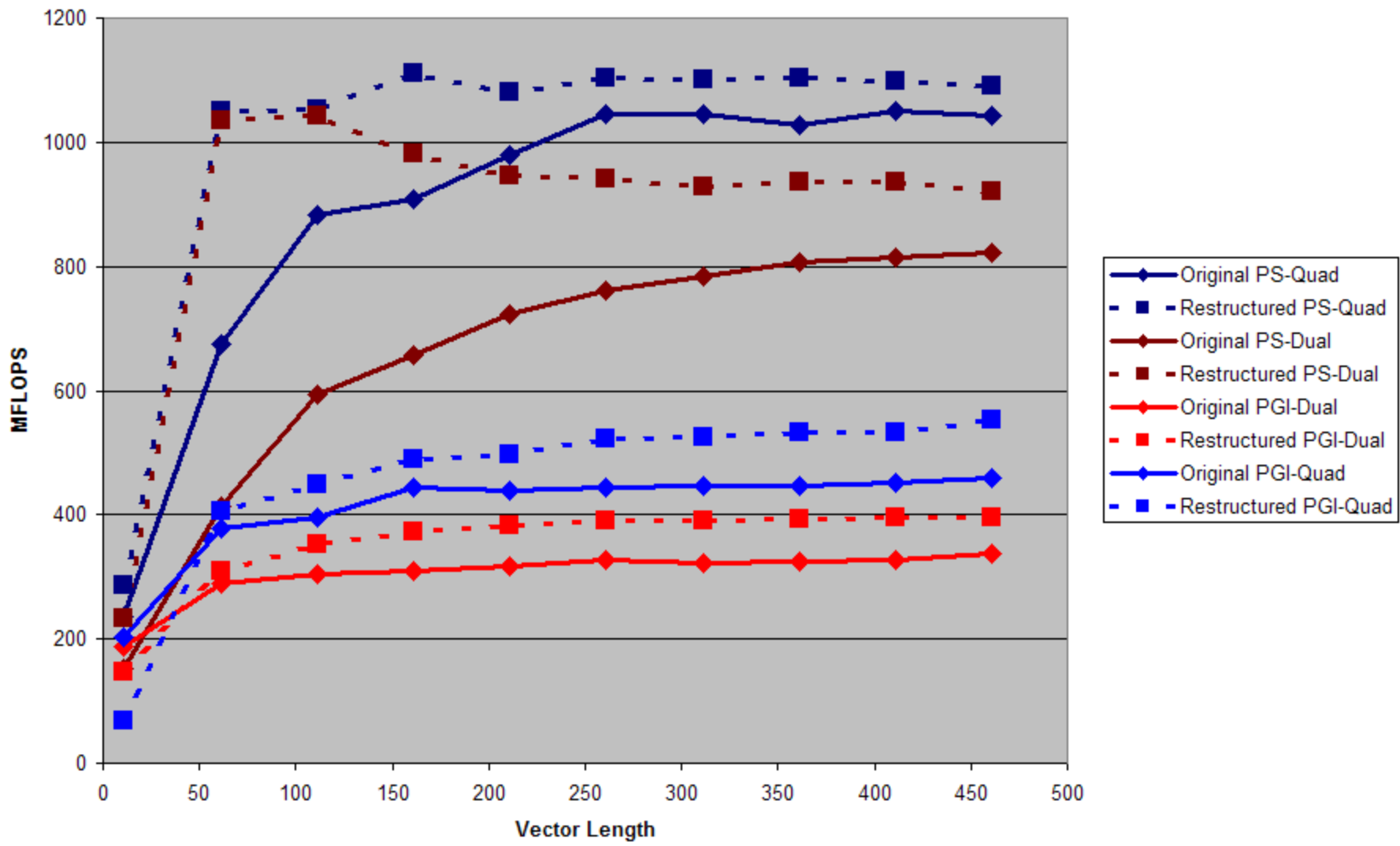
PGI

- 73, Generated an alternate loop for the inner loop
 - Generated vector sse code for inner loop
 - Generated 1 prefetch instructions for this loop
 - Generated vector sse code for inner loop
 - Generated 1 prefetch instructions for this loop
- 78, Generated 2 alternate loops for the inner loop
 - Generated vector sse code for inner loop
 - Generated 1 prefetch instructions for this loop
 - Generated vector sse code for inner loop
 - Generated 1 prefetch instructions for this loop
 - Generated vector sse code for inner loop
 - Generated 1 prefetch instructions for this loop
- 82, Interchange produces reordered loop nest: 83, 84, 82
 - Loop unrolled 5 times (completely unrolled)
- 84, Generated vector sse code for inner loop
 - Generated 1 prefetch instructions for this loop

Pathscale

- (lp45020.f:73) LOOP WAS VECTORIZED.
- (lp45020.f:78) LOOP WAS VECTORIZED.
- (lp45020.f:78) LOOP WAS VECTORIZED.
- (lp45020.f:84) Non-contiguous array "C(_BLNK__.0.0)" reference exists.
Loop was not vectorized.
- (lp45020.f:84) Non-contiguous array "C(_BLNK__.0.0)" reference exists.
Loop was not vectorized.

LP45020



Big Loop

```

( 52) C      THE ORIGINAL
( 53)
( 54)      DO 47020  J = 1, JMAX
( 55)      DO 47020  K = 1, KMAX
( 56)      DO 47020  I = 1, IMAX
( 57)      JP          = J + 1
( 58)      JR          = J - 1
( 59)      KP          = K + 1
( 60)      KR          = K - 1
( 61)      IP          = I + 1
( 62)      IR          = I - 1
( 63)      IF ( J .EQ. 1)      GO TO 50
( 64)      IF( J .EQ. JMAX) GO TO 51
( 65)      XJ = ( A(I,JP,K) - A(I,JR,K) ) * DA2
( 66)      YJ = ( B(I,JP,K) - B(I,JR,K) ) * DA2
( 67)      ZJ = ( C(I,JP,K) - C(I,JR,K) ) * DA2
( 68)      GO TO 70
( 69) 50      J1 = J + 1
( 70)      J2 = J + 2
( 71)      XJ = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
( 72)      YJ = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
( 73)      ZJ = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
( 74)      GO TO 70
( 75) 51      J1 = J - 1
( 76)      J2 = J - 2
( 77)      XJ = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
( 78)      YJ = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
( 79)      ZJ = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
( 80) 70      CONTINUE
( 81)      IF ( K .EQ. 1)      GO TO 52
( 82)      IF ( K .EQ. KMAX) GO TO 53
( 83)      XK = ( A(I,J,KP) - A(I,J,KR) ) * DB2
( 84)      YK = ( B(I,J,KP) - B(I,J,KR) ) * DB2
( 85)      ZK = ( C(I,J,KP) - C(I,J,KR) ) * DB2
( 86)      GO TO 71

```

Big Loop

```

( 87)   52   K1 = K + 1
( 88)           K2 = K + 2
( 89)           XK = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
( 90)           YK = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
( 91)           ZK = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
( 92)           GO TO 71
( 93)   53   K1 = K - 1
( 94)           K2 = K - 2
( 95)           XK = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
( 96)           YK = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
( 97)           ZK = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
( 98)   71   CONTINUE
( 99)           IF ( I .EQ. 1)      GO TO 54
(100)           IF ( I .EQ. IMAX) GO TO 55
(101)           XI = ( A(IP,J,K) - A(IR,J,K) ) * DC2
(102)           YI = ( B(IP,J,K) - B(IR,J,K) ) * DC2
(103)           ZI = ( C(IP,J,K) - C(IR,J,K) ) * DC2
(104)           GO TO 60
(105)   54   I1 = I + 1
(106)           I2 = I + 2
(107)           XI = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
(108)           YI = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
(109)           ZI = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
(110)           GO TO 60
(111)   55   I1 = I - 1
(112)           I2 = I - 2
(113)           XI = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
(114)           YI = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
(115)           ZI = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
(116)   60   CONTINUE
(117)           DINV      = XJ * YK * ZI + YJ * ZK * XI + ZJ * XK * YI
(118)           *          - XJ * ZK * YI - YJ * XK * ZI - ZJ * YK * XI
(119)           D(I,J,K) = 1. / (DINV + 1.E-20)
(120) 47020 CONTINUE
(121)

```

PGI

55, Invariant if transformation

**Loop not vectorized: loop count too
small**

56, Invariant if transformation

Pathscale

Nothing

Re-Write

```

( 141) C      THE RESTRUCTURED
( 142)
( 143)      DO 47029 J = 1, JMAX
( 144)      DO 47029 K = 1, KMAX
( 145)
( 146)      IF(J.EQ.1) THEN
( 147)
( 148)      J1          = 2
( 149)      J2          = 3
( 150)      DO 47021 I = 1, IMAX
( 151)      VAJ(I) = (-3. * A(I,J,K) + 4. * A(I,J1,K) - A(I,J2,K) ) * DA2
( 152)      VBJ(I) = (-3. * B(I,J,K) + 4. * B(I,J1,K) - B(I,J2,K) ) * DA2
( 153)      VCJ(I) = (-3. * C(I,J,K) + 4. * C(I,J1,K) - C(I,J2,K) ) * DA2
( 154) 47021 CONTINUE
( 155)
( 156)      ELSE IF(J.NE.JMAX) THEN
( 157)
( 158)      JP          = J+1
( 159)      JR          = J-1
( 160)      DO 47022 I = 1, IMAX
( 161)      VAJ(I) = ( A(I,JP,K) - A(I,JR,K) ) * DA2
( 162)      VBJ(I) = ( B(I,JP,K) - B(I,JR,K) ) * DA2
( 163)      VCJ(I) = ( C(I,JP,K) - C(I,JR,K) ) * DA2
( 164) 47022 CONTINUE
( 165)
( 166)      ELSE
( 167)
( 168)      J1          = JMAX-1
( 169)      J2          = JMAX-2
( 170)      DO 47023 I = 1, IMAX
( 171)      VAJ(I) = ( 3. * A(I,J,K) - 4. * A(I,J1,K) + A(I,J2,K) ) * DA2
( 172)      VBJ(I) = ( 3. * B(I,J,K) - 4. * B(I,J1,K) + B(I,J2,K) ) * DA2
( 173)      VCJ(I) = ( 3. * C(I,J,K) - 4. * C(I,J1,K) + C(I,J2,K) ) * DA2
( 174) 47023 CONTINUE
( 175)
( 176)      ENDIF

```

Re-Write

```

( 178)      IF(K.EQ.1) THEN
( 179)
( 180)      K1          = 2
( 181)      K2          = 3
( 182)      DO 47024 I = 1, IMAX
( 183)          VAK(I) = (-3. * A(I,J,K) + 4. * A(I,J,K1) - A(I,J,K2) ) * DB2
( 184)          VBK(I) = (-3. * B(I,J,K) + 4. * B(I,J,K1) - B(I,J,K2) ) * DB2
( 185)          VCK(I) = (-3. * C(I,J,K) + 4. * C(I,J,K1) - C(I,J,K2) ) * DB2
( 186) 47024 CONTINUE
( 187)
( 188)      ELSE IF(K.NE.KMAX) THEN
( 189)
( 190)      KP          = K + 1
( 191)      KR          = K - 1
( 192)      DO 47025 I = 1, IMAX
( 193)          VAK(I) = ( A(I,J,KP) - A(I,J,KR) ) * DB2
( 194)          VBK(I) = ( B(I,J,KP) - B(I,J,KR) ) * DB2
( 195)          VCK(I) = ( C(I,J,KP) - C(I,J,KR) ) * DB2
( 196) 47025 CONTINUE
( 197)
( 198)      ELSE
( 199)
( 200)      K1          = KMAX - 1
( 201)      K2          = KMAX - 2
( 202)      DO 47026 I = 1, IMAX
( 203)          VAK(I) = ( 3. * A(I,J,K) - 4. * A(I,J,K1) + A(I,J,K2) ) * DB2
( 204)          VBK(I) = ( 3. * B(I,J,K) - 4. * B(I,J,K1) + B(I,J,K2) ) * DB2
( 205)          VCK(I) = ( 3. * C(I,J,K) - 4. * C(I,J,K1) + C(I,J,K2) ) * DB2
( 206) 47026 CONTINUE
( 207)      ENDIF
( 208)

```

Re-Write

```

( 209)      I = 1
( 210)      I1      = 2
( 211)      I2      = 3
( 212)      VAI(I) = (-3. * A(I,J,K) + 4. * A(I1,J,K) - A(I2,J,K) ) * DC2
( 213)      VBI(I) = (-3. * B(I,J,K) + 4. * B(I1,J,K) - B(I2,J,K) ) * DC2
( 214)      VCI(I) = (-3. * C(I,J,K) + 4. * C(I1,J,K) - C(I2,J,K) ) * DC2
( 215)
( 216)      DO 47027 I = 2, IMAX-1
( 217)          IP      = I + 1
( 218)          IR      = I - 1
( 219)              VAI(I) = ( A(IP,J,K) - A(IR,J,K) ) * DC2
( 220)              VBI(I) = ( B(IP,J,K) - B(IR,J,K) ) * DC2
( 221)              VCI(I) = ( C(IP,J,K) - C(IR,J,K) ) * DC2
( 222) 47027  CONTINUE
( 223)
( 224)      I = IMAX
( 225)      I1      = IMAX - 1
( 226)      I2      = IMAX - 2
( 227)      VAI(I) = ( 3. * A(I,J,K) - 4. * A(I1,J,K) + A(I2,J,K) ) * DC2
( 228)      VBI(I) = ( 3. * B(I,J,K) - 4. * B(I1,J,K) + B(I2,J,K) ) * DC2
( 229)      VCI(I) = ( 3. * C(I,J,K) - 4. * C(I1,J,K) + C(I2,J,K) ) * DC2
( 230)
( 231)      DO 47028 I = 1, IMAX
( 232)          DINV = VAJ(I) * VBK(I) * VCI(I) + VBJ(I) * VCK(I) * VAI(I)
( 233)      1      + VCJ(I) * VAK(I) * VBI(I) - VAJ(I) * VCK(I) * VBI(I)
( 234)      2      - VBJ(I) * VAK(I) * VCI(I) - VCJ(I) * VBK(I) * VAI(I)
( 235)          D(I,J,K) = 1. / (DINV + 1.E-20)
( 236) 47028  CONTINUE
( 237) 47029  CONTINUE

( 238)

```

PGI**144, Invariant if transformation****Loop not vectorized: loop count too small****150, Generated 3 alternate loops for the inner loop****Generated vector sse code for inner loop****Generated 8 prefetch instructions for this loop****Generated vector sse code for inner loop****Generated 8 prefetch instructions for this loop****Generated vector sse code for inner loop****Generated 8 prefetch instructions for this loop****Generated vector sse code for inner loop****Generated 8 prefetch instructions for this loop****160, Generated 4 alternate loops for the inner loop****Generated vector sse code for inner loop****Generated 6 prefetch instructions for this loop****Generated vector sse code for inner loop****o o o**

Pathscale

(lp47020.f:132) LOOP WAS VECTORIZED.

(lp47020.f:150) LOOP WAS VECTORIZED.

(lp47020.f:160) LOOP WAS VECTORIZED.

(lp47020.f:170) LOOP WAS VECTORIZED.

(lp47020.f:182) LOOP WAS VECTORIZED.

(lp47020.f:192) LOOP WAS VECTORIZED.

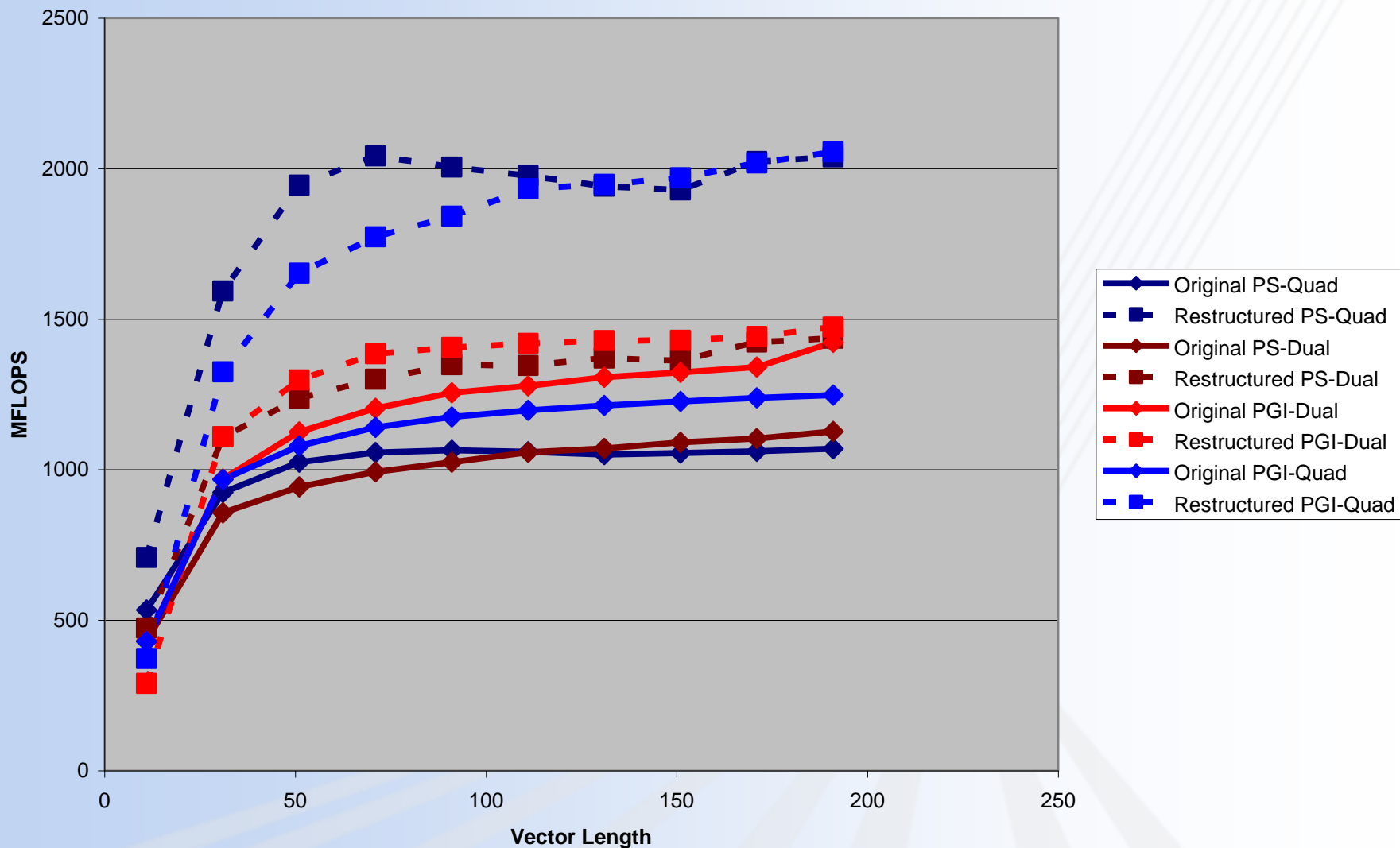
(lp47020.f:202) LOOP WAS VECTORIZED.

(lp47020.f:216) LOOP WAS VECTORIZED.

(lp47020.f:231) LOOP WAS VECTORIZED.

(lp47020.f:248) LOOP WAS VECTORIZED.

LP47020



Original

```
( 42) C      THE ORIGINAL
( 43)
( 44)      DO 48070 I = 1, N
( 45)      A(I) = (B(I)**2 + C(I)**2)
( 46)      CT  = PI * A(I) + (A(I))**2
( 47)      CALL SSUB (A(I), CT, D(I), E(I))
( 48)      F(I) = (ABS (E(I)))
( 49) 48070 CONTINUE
( 50)
```

PGI

44, Loop not vectorized: contains call

Pathscale

Nothing

Restructured

```

( 69) C      THE RESTRUCTURED
( 70)
( 71)      DO 48071 I = 1, N
( 72)      A(I) = (B(I)**2 + C(I)**2)
( 73)      CT  = PI * A(I) + (A(I))**2
( 74)      E(I) = A(I)**2 + (ABS (A(I) + CT)) * (CT * ABS (A(I) - CT))
( 75)      D(I) = A(I) + CT
( 76)      F(I) = (ABS (E(I)))
( 77) 48071 CONTINUE
( 78)

```

PGI

71, Generated an alternate loop for the inner loop

Unrolled inner loop 4 times

Used combined stores for 2 stores

Generated 2 prefetch instructions for this loop

Unrolled inner loop 4 times

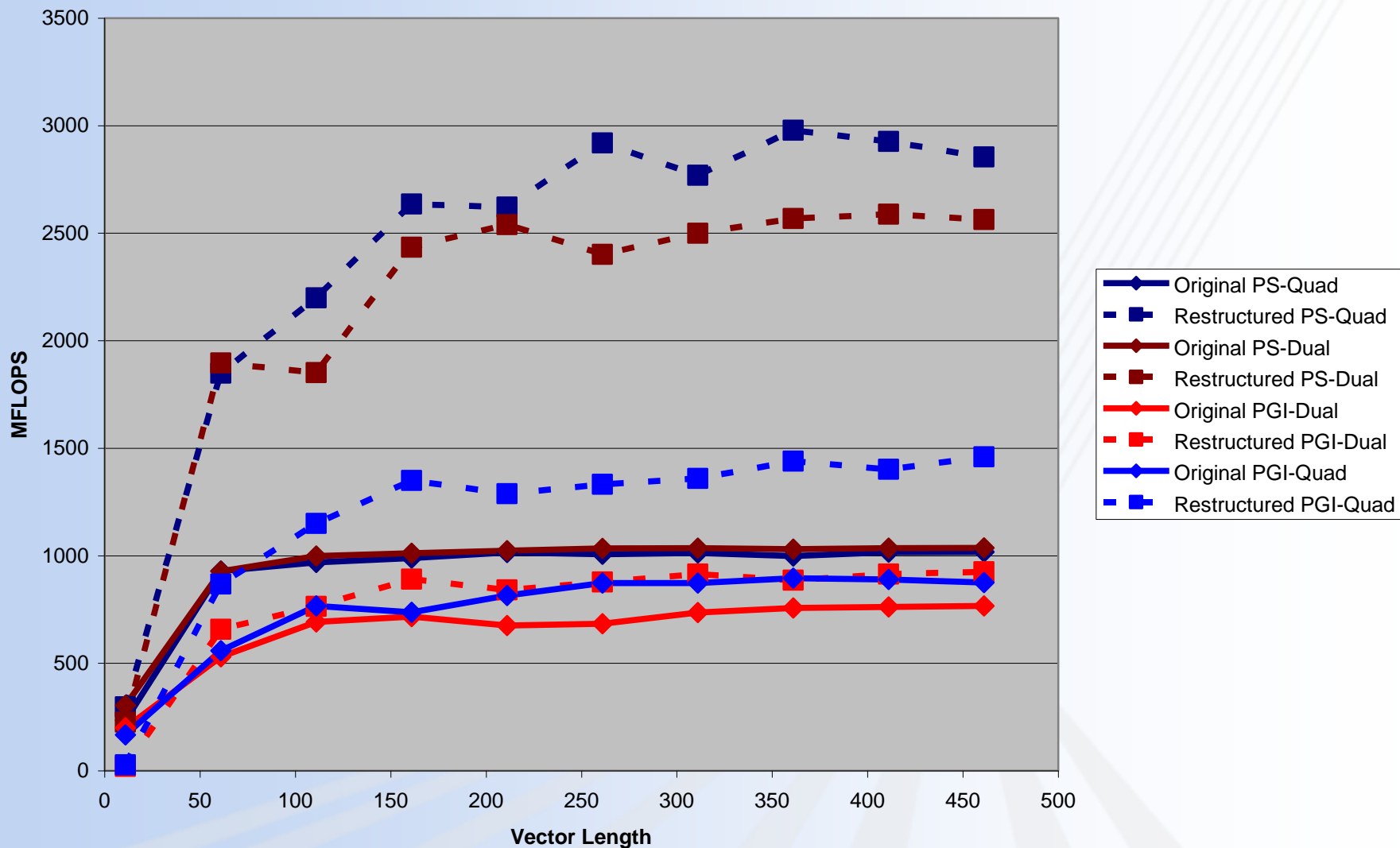
Used combined stores for 2 stores

Generated 2 prefetch instructions for this loop

Pathscale

(lp48070.f:71) LOOP WAS VECTORIZED.

LP48070



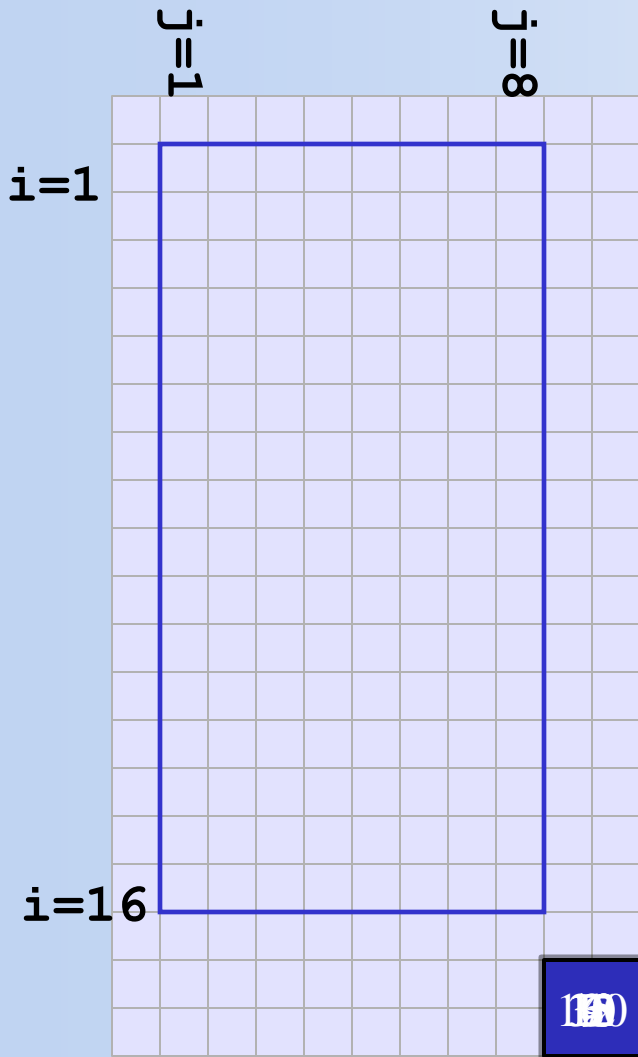
Cache-Blocking

OPTIMIZING YOUR CODE

What is Cache Blocking?

- **Cache blocking** is a combination of strip mining and loop interchange, designed to increase data reuse.
 - ✿ Takes advantage of temporal reuse: re-reference array elements already referenced
 - ✿ Good blocking will take advantage of spatial reuse: work with the cache lines!
- Many ways to block any given loop nest
 - ✿ Which loops get blocked?
 - ✿ What block size(s) to use?
- Analysis can reveal which ways are beneficial
- But trial-and-error is probably faster

Cache Use in Stencil Computations

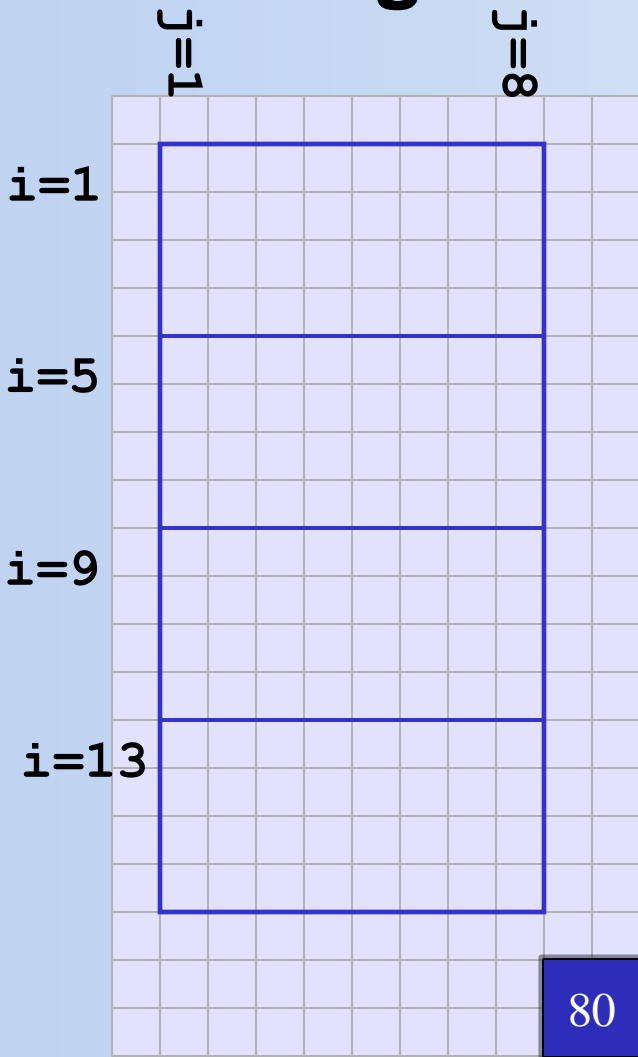


2D Laplacian

```
do j = 1, 8
  do i = 1, 16
    a = u(i-1,j) + u(i+1,j) &
      - 4*u(i,j) &
      + u(i,j-1) + u(i,j+1)
  end do
end do
```

- Cache structure for this example:
 - ✿ Each line holds 4 array elements
 - ✿ Cache can hold 12 lines of u data
- No cache reuse between outer loop iterations

Blocking to Increase Reuse

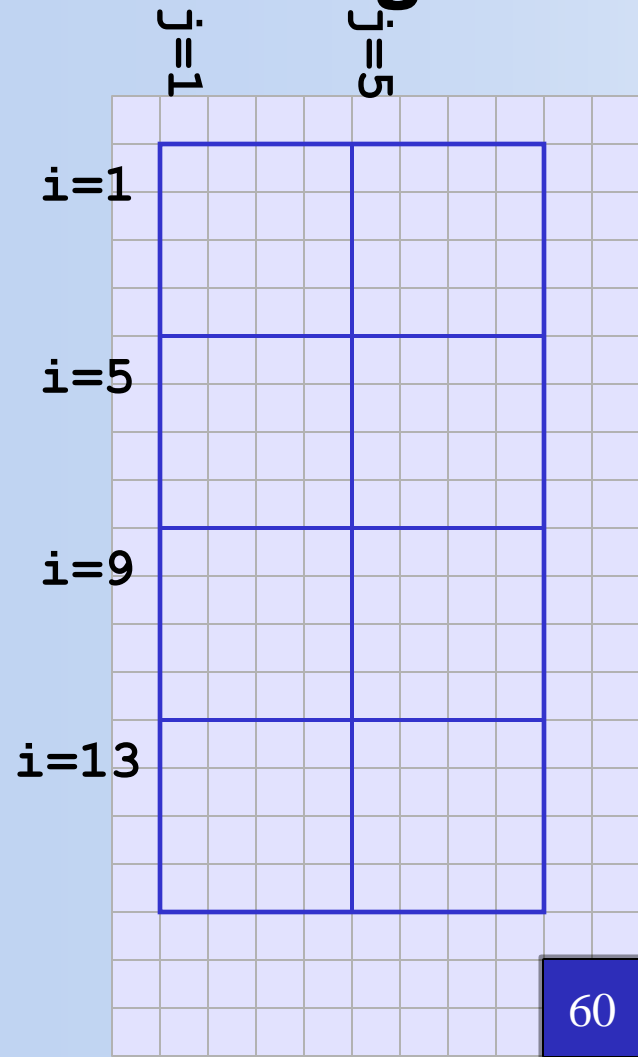


- Unblocked loop: 120 cache misses
- Block the inner loop

```
do IBLOCK = 1, 16, 4
  do j = 1, 8
    do i = IBLOCK, IBLOCK + 3
      a(i,j) = u(i-1,j) + u(i+1,j) &
              - 2*u(i,j)           &
              + u(i,j-1) + u(i,j+1)
    end do
  end do
end do
```

- Now we have reuse of the “j+1” data

Blocking to Increase Reuse



- One-dimensional blocking reduced misses from 120 to 80
- Iterate over 4 4 blocks

```

do JBLOCK = 1, 8, 4
  do IBLOCK = 1, 16, 4
    do j = JBLOCK, JBLOCK + 3
      do i = IBLOCK, IBLOCK + 3
        a(i,j) = u(i-1,j) + u(i+1,j) &
                - 2*u(i,j) &
                + u(i,j-1) + u(i,j+1)
      end do
    end do
  end do
end do

```

- Better use of spatial locality (cache lines)

Obligatory GEMM discussion

- Matrix-matrix multiply (GEMM) is the canonical cache-blocking example
- Operations can be arranged to create multiple levels of blocking
 - ✿ Block for register
 - ✿ Block for cache (L1, L2, L3)
 - ✿ Block for TLB
- No further discussion here. Interested readers can see
 - ✿ Any book on code optimization
 - ▶ Sun's *Techniques for Optimizing Applications: High Performance Computing* contains a decent introductory discussion in Chapter 8
 - ▶ Insert your favorite book here
 - ✿ Gunnel, Henry, and van de Geijn. June 2001. *High-performance matrix multiplication algorithms for architectures with hierarchical memories*. FLAME Working Note #4 TR-2001-22, The University of Texas at Austin, Department of Computer Sciences
 - ▶ Develops algorithms and cost models for GEMM in hierarchical memories
 - ✿ Goto and van de Geijn. 2008. *Anatomy of high-performance matrix multiplication*. *ACM Transactions on Mathematical Software* 34, 3 (May), 1-25
 - ▶ Description of GotoBLAS DGEMM

What Could Go Wrong?

“I tried cache-blocking my code, but it didn't help”

- You're doing it wrong.
 - ✿ Your block size is too small (too much loop overhead).
 - ✿ Your block size is too big (data is falling out of cache).
 - ✿ You're targeting the wrong cache level (?)
 - ✿ You haven't selected the correct subset of loops to block.
- The compiler is already blocking that loop.
- Prefetching is acting to minimize cache misses.
- Computational intensity within the loop nest is very large, making blocking less important.

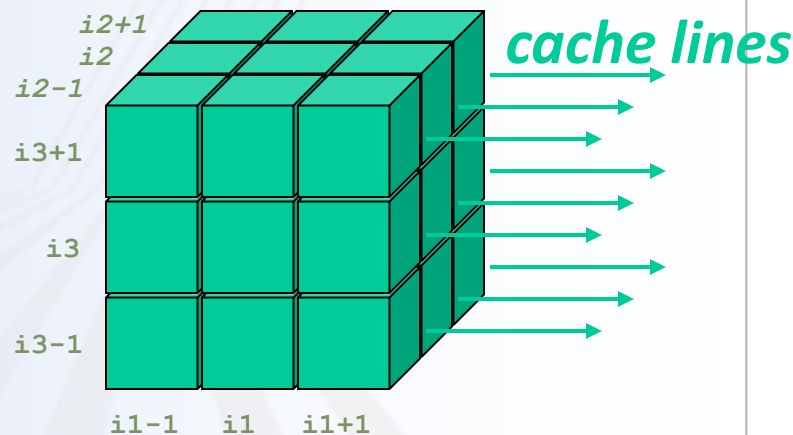
A Real-Life Example: NPB MG

- Multigrid PDE solver
- Class D, 64 MPI ranks
 - ✿ Global grid is 1024 1024 1024
 - ✿ Local grid is 258 258 258
- Two similar loop nests account for >50% of run time
- 27-point 3D stencil
 - ✿ There is good data reuse along leading dimension, even without blocking

```

do i3 = 2, 257
  do i2 = 2, 257
    do i1 = 2, 257
      !       update u(i1,i2,i3)
      !       using 27-point stencil
    end do
  end do
end do

```



I'm Doing It Wrong

- Block the inner two loops
- Creates blocks extending along *i3* direction

```

do I2BLOCK = 2, 257, BS2
  do I1BLOCK = 2, 257, BS1
    do i3 = 2, 257
      do i2 = I2BLOCK,                &
          min(I2BLOCK+BS2-1, 257)
        do i1 = I1BLOCK,                &
            min(I1BLOCK+BS1-1, 257)
!           update u(i1,i2,i3)
!           using 27-point stencil
        end do
      end do
    end do
  end do
end do

```

Block size	Mop/s/proces s
unblocked	531.50
16 16	279.89
22 22	321.26
28 28	358.96
34 34	385.33
40 40	408.53
46 46	443.94
52 52	468.58
58 58	470.32
64 64	512.03
70 70	506.92

That's Better

- Block the outer two loops
- Preserves spatial locality along *i1* direction

```

do I3BLOCK = 2, 257, BS3
  do I2BLOCK = 2, 257, BS2
    do i3 = I3BLOCK,                &
      min(I3BLOCK+BS3-1, 257)
      do i2 = I2BLOCK,                &
        min(I2BLOCK+BS2-1, 257)
        do i1 = 2, 257
!           update u(i1,i2,i3)
!           using 27-point stencil
          end do
        end do
      end do
    end do
  end do
end do

```

Block size	Mop/s/proces s
unblocked	531.50
16 16	674.76
22 22	680.16
28 28	688.64
34 34	683.84
40 40	698.47
46 46	689.14
52 52	706.62
58 58	692.57
64 64	703.40
70 70	693.87

Tuning Malloc

OPTIMIZING YOUR CODE

GNU Malloc

- GNU malloc library
 - ⚙ malloc, calloc, realloc, free calls
 - ▶ Fortran dynamic variables
- Malloc library system calls
 - ⚙ Mmap, munmap => for larger allocations
 - ⚙ Brk, sbrk => increase/decrease heap
- Malloc library optimized for low system memory use
 - ⚙ Can result in system calls/minor page faults

Improving GNU Malloc

- Detecting “bad” malloc behavior
 - ✱ Profile data => “excessive system time”
- Correcting “bad” malloc behavior
 - ✱ Eliminate mmap use by malloc
 - ✱ Increase threshold to release heap memory
- Use environment variables to alter malloc
 - ✱ `MALLOC_MMAP_MAX_ = 0`
 - ✱ `MALLOC_TRIM_THRESHOLD_ = 536870912`
- Possible downsides
 - ✱ Heap fragmentation
 - ✱ User process may call mmap directly
 - ✱ User process may launch other processes
- PGI’s `-Msmartalloc` does something similar for you at compile time

Google TCMalloc

- Google created a replacement “malloc” library
 - ✿ “Minimal” TCMalloc replaces GNU malloc
- Limited testing indicates TCMalloc as good or better than GNU malloc
 - ✿ Environment variables not required
 - ✿ TCMalloc almost certainly better for allocations in OpenMP parallel regions
- There’s currently no pre-built tcmalloc for Cray XT, but some users have successfully built it.

Memory Allocation: Make it local

- Linux has a “first touch policy” for memory allocation
 - ✿ *alloc functions don’t actually allocate your memory
 - ✿ Memory gets allocated when “touched”
- Problem: A code can allocate more memory than available
 - ✿ Linux assumed “swap space,” we don’t have any
 - ✿ Applications won’t fail from over-allocation until the memory is finally touched
- Problem: Memory will be put on the core of the “touching” thread
 - ✿ Only a problem if thread 0 allocates all memory for a node
- Solution: Always initialize your memory immediately after allocating it
 - ✿ If you over-allocate, it will fail immediately, rather than a strange place in your code
 - ✿ If every thread touches its own memory, it will be allocated on the proper socket

Using Libraries

OPTIMIZING YOUR CODE

Using Libraries

- Cray's Scientific Libraries team has worked to optimize common library calls for each architecture
 - ✿ There are more library routines than developers, so tell us what's important
- Let the wrappers choose the right library for you
 - ✿ As long as you have `xtpc-<arch>` loaded, the wrappers will pick the best library for you
 - ✿ Linking against the wrong library can dramatically reduce performance
- Library calls are tuned for general cases, if you have a particular size, they may be able to do better
 - ✿ GEMMs are tuned for square matrices, if yours aren't square, they may be able to help you do better.

NCCS: help@nccs.gov
NICS: help@teragrid.org
Contact Your Liaison

Jeff Larkin: larkin@cray.com

The Best Optimization Technique:

SEEK HELP