

Impact of Communication Protocol on Performance

Patrick H. Worley

Oak Ridge National Laboratory

P.O. Box 2008

Oak Ridge, Tennessee 37831-6367

email: worleyph@ornl.gov

phone: +1 423 574-3128

Fax: +1 423 574-0680

1. Introduction. On previous generation MPP systems, interprocessor communication often represented a significant fraction of the runtime of production parallel codes, and the choice of communication transport layer and communication protocol were important steps in porting and tuning application codes. Processor, network, and transport layer performance continue to improve, and the sensitivity of performance to these implementation issues needs to be reexamined.

In this paper we use the PSTSWM parallel application code to examine

- 1) single processor performance,
- 2) peak achievable point-to-point communication performance,
- 3) performance variation of kernels as a function of communication protocols,
- 4) performance of vendor-supplied collective communication routines, and
- 5) performance sensitivity of full code to choice of parallel implementation

for the SGI/Cray Research T3E and Origin 2000, using both the MPI [2] and SHMEM libraries to implement interprocessor communication. While other researchers have looked at communication performance on these machines (e.g., [1]), this study differs in that we examine the effect on performance of an application code.

2. PSTSWM. The Parallel Spectral Transform Shallow Water Model (PSTSWM) is a message-passing parallel benchmark code and parallel algorithm testbed that solves the nonlinear shallow water equations on a rotating sphere using the spectral transform method. PSTSWM was developed by the author and by I. T. Foster at Argonne National Laboratory from the serial code STSWM, written by J. J. Hack and R. Jakob at the National Center for Atmospheric Research. PSTSWM was used to evaluate parallel algorithms for the spectral transform method as it is used in global atmospheric circulation models.

PSTSWM has characteristics that make it useful for performance studies. It makes interesting and varied demands on the communication subsystem, multiple parallel algorithms are embedded in the code, and multiple message-passing transport layers are supported. See <http://www.epm.ornl.gov/champp/pstswm/index.html> for a

Computer Science and Mathematics Division

Oak Ridge National Laboratory is managed by Lockheed Martin Energy Research Corporation for the United States Department of Energy under Contract No. DE-AC05-96OR22464.

"This submitted manuscript has been authored by a contractor of the U.S. Government under Contract No. DE-AC05-96OR22464. Accordingly, the U.S. Government retains a nonexclusive royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes."

partial bibliography of the performance studies utilizing PSTSWM.

3. Platforms. We focus on the T3E and Origin2000 in these studies, but include measurements from the following platforms to aid in the understanding of the results.

Intel Paragon XP/S 150 at Oak Ridge National Laboratory.

This machine has 1024 MP nodes (3 50-MHz iPSC/860 processors per node). Measurements were taken in January, 1998. Only one processor per node was used for computation. KAI math routines were used.

CRI T3D at Cray Research in Eagan, MN.

This machine had 128 150-MHz DEC Alpha EV4 processors. Measurements were taken in August, 1996.

IBM SP2 at NASA Ames Research Center.

This machine had 160 RS6000/590 nodes ("wide", 66.7 MHz POWER2). Measurements were taken in August, 1996. ESSL math routines were used.

Convex SPP-1200 at the National Center for Supercomputer Applications.

This machine has 64 120-MHz HP PA-RISC 7200 processors (8 Hypernodes). Measurements were taken in September, 1996.

SGI/CR T3E-900 at the National Energy Research Scientific Computing Center.

This machine has 532 450-MHz DEC Alpha EV5 RISC processors. Measurements were taken in January, 1998.

HP/CONVEX SPP-2000 at the National Center for Supercomputer Applications.

This machine has 64 180-MHz HP PA-RISC 8000 processors (4 Hypernodes). Measurements were taken in April, 1998. VECLIB math routines were used.

Intel PII-266 cluster at Oak Ridge National Laboratory.

This machine has 10 266-MHz dual Pentium II nodes. Measurements were taken in February, 1998. LINUX and Portland Group f77 compiler were used.

SGI/CR Origin2000 at Los Alamos National Laboratory.

This machine has 128 195-MHz MIPS R10000 processors. Measurements were last taken in April, 1998. SCSL math routines were used.

4. Serial Performance. Table 1 contains the MFlop/sec rates for one processor runs of the code PSTSWM for a number of different problem sizes. PSTSWM computes the solution by timestepping, advancing the approximation to a new timelevel (in simulation time) by using the approximations at the two previous timelevels. The computational complexity and code executed for a timestep in PSTSWM are identical for all timesteps.

We use the standard benchmark problem for the shallow water equations, global steady state nonlinear zonal geostrophic flow [3], and two problem size classes: T42

and T85, characterized by the following computational grids and complexity.

	physical grid	Fourier grid	spectral coefficients	flops per timestep
T42	64 × 128	64 × 64	946	4129859
T85	128 × 256	128 × 128	3741	24235477

There is also a vertical component to the problem size. For example, T42L16 is a T42 horizontal grid with 16 vertical levels. The complexity of solving the problem is linear in the number of vertical levels.

64-bit precision floating point computation is used in all experiments. Math library routines are used for the Fourier transforms where available, as indicated in the description of the platforms in the previous section.

	T42L1	T42L3	T42L16	T85L1	T85L3
Intel Paragon	13.9	14.0	13.9	13.1	13.1
CRI T3D	25.2	25.7	23.3	24.9	24.4
IBM SP2	98.3	98.3	91.0	107.7	102.4
Convex SPP-1200	24.9	23.2	22.9	24.2	24.0
SGI/CR T3E-900	79.4	70.0	64.1	84.7	70.7
HP/Convex SPP-2000	138.8	107.3	83.5	117.5	114.2
Intel PII-266 cluster	45.4	37.2	30.3	38.9	33.7
SGI/CR Origin2000	153.0	140.8	92.5	131.7	130.1

TABLE 1
Serial MFlop/sec rates.

From this data it is clear that the serial performance of MPP processors has generally improved over the past few years, and that optimized math libraries are important performance enhancers. Also note that some effects of the memory hierarchy on performance can be observed from the variation in MFlop/sec rate as the problem size varies.

5. Point-to-Point Communication Performance. Communication overhead is best measured in the context of the full code, but it is useful to establish a performance baseline by determining the “peak achievable” point-to-point interprocessor communication performance. Performance-critical interprocessor communication in PSTSWM is implemented using two basic types of commands: SWAP and SENDRECV. The message-passing transport layer used to implement these commands is specified at compile time, while the protocol used is specified at runtime.

To characterize the basic communication capabilities in terms relevant to PSTSWM, we use the PSTSWM SWAP command. We measure the time required to exchange 262144 64-bit floating point numbers between two neighboring processors, varying the protocol used for the exchange to find the minimum. We refer to these as the 2MB experiments. We also measure the time to swap 1024 and 16384 64-bit values, referring to these as the 8KB and 128KB experiments, respectively.

Two general classes of protocols are used: unordered (ping-ping) and ordered (ping-pong). While not all protocols are available for all message-passing transport layers, they are drawn from those described in Table 2. Examples are given using MPI commands. Note that the examples have been simplified (to save room) and do not accurately represent the MPI implementations.

Table 3 contains the maximum observed bandwidth and typical SWAP overhead ("latency") for the corresponding communication protocol. (Note that this protocol does not necessarily have the smallest latency.) The following observations on communication performance on the T3E and the Origin2000 can be drawn from this summary data:

- The T3E and the Origin2000 demonstrate significant performance improvement over previous generation MPPs of like architecture. (Note however that the SPP-2000 performance is better than both, for these particular tests.)
- SHMEM achieves considerably higher bandwidth and lower latency than MPI, but MPI performance is still an improvement over what was achievable on earlier systems.

Looking at the raw timing data, we can also determine the sensitivity of performance to the choice of communication protocol. On the T3E the achievable bandwidth shows little sensitivity to the communication protocol when using MPI, and the simple protocols are generally slightly better. On the Origin2000, MPI performance is somewhat more sensitive to the communication protocol, but the communication protocol is still not too important. This is a significant difference from earlier results on the Intel Paragon and the IBM SP2, but is similar to the T3D results, and appears to reflect the SGI/CR implementation of MPI. When using SHMEM, the variability is higher (for both systems).

6. Parallel Algorithm Performance. Some indication of the impact of communication protocol on performance can be seen from the point-to-point communication tests, but it is difficult to use these results to predict the effect on application code performance. Here we examine this issue in more detail, looking at the effect on the performance of specific parallel algorithm options in PSTSWM.

Unordered	Ordered
(0,0): <u>simple</u> Processors 1 and 2 MPLBSEND MPLRECV	(1,0): <u>simple</u> Processor 1 Processor 2 MPLSEND MPLRECV MPLRECV MPLSEND
(0,1): <u>nonblocking send</u> Processors 1 and 2 MPLISEND MPLRECV	(1,1): <u>nonblocking send</u> Processor 1 Processor 2 MPLISEND MPLRECV MPLRECV MPLISEND
(0,2): <u>nonblocking receive</u> Processors 1 and 2 MPLIRECV MPLSEND	(1,2): <u>nonblocking receive</u> Processor 1 Processor 2 MPLIRECV MPLRECV MPLSEND MPLSEND
(0,3): <u>nonblocking send & receive</u> Processors 1 and 2 MPLIRECV MPLISEND	(1,3): <u>nonblocking send & receive</u> Processor 1 Processor 2 MPLIRECV MPLRECV MPLISEND MPLSEND
(0,4): <u>ready send</u> Processors 1 and 2 MPLIRECV MPLRSEND	(1,4): <u>ready send</u> Processor 1 Processor 2 MPLIRECV MPLRECV MPLRSEND MPLRSEND
(0,5): <u>nonblocking ready send</u> Processors 1 and 2 MPLIRECV MPLRSEND	(1,5): <u>nonblocking ready send</u> Processor 1 Processor 2 MPLIRECV MPLRECV MPLRSEND MPLRSEND
(0,6): <u>native sendrecv</u> Processors 1 and 2 MPLSENDRECV	(1,4): <u>synchronous</u> Processor 1 Processor 2 — MPLRECV MPLSEND — — MPLSEND MPLRECV —

TABLE 2
SWAP protocols (simplified).

	Unordered								
	2MB			128KB			8KB		
	BW	lat.	prot.	BW	lat.	prot.	BW	lat.	prot.
Paragon									
: MPI	73	82	(0,6)	70	136	(0,3)	48	139	(0,3)
: NX	76	32	(0,3)	71	83	(0,3)	57	74	(0,3)
: SUNMOS	293	63	(0,3)	—	—	—	—	—	—
T3D									
: SHMEM	183	19	(0,2)	—	—	—	—	—	—
SP2									
: MPI	96	136	(0,4)	—	—	—	—	—	—
SPP-1200									
: MPI	45	104	(0,4)	—	—	—	—	—	—
T3E-900									
: MPI	286	30	(0,2)	245	24	(0,0)	66	25	(0,2)
: SHMEM	543	7	(0,1)	494	7	(0,1)	258	7	(0,1)
SPP-2000									
: MPI	654	39	(0,6)	629	15	(0,6)	145	23	(0,2)
Origin2000									
: MPI	142	39	(0,1)	128	29	(0,6)	57	30	(0,1)
: SHMEM	287	15	(0,1)	222	14	(0,1)	114	12	(0,2)
	Ordered								
	2MB			128KB			8KB		
	BW	lat.	prot.	BW	lat.	prot.	BW	lat.	prot.
Paragon									
: MPI	118	75	(1,0)	107	105	(1,3)	52	82	(1,0)
: NX	118	50	(1,0)	114	62	(1,0)	75	52	(1,0)
: SUNMOS	154	35	(1,0)	—	—	—	—	—	—
T3D									
: SHMEM	126	12	(1,2)	—	—	—	—	—	—
SP2									
: MPI	71	74	(1,1)	—	—	—	—	—	—
SPP-1200									
: MPI	29	27	(1,3)	—	—	—	—	—	—
T3E-900									
: MPI	163	29	(1,6)	134	20	(1,0)	47	21	(1,0)
: SHMEM	336	9	(1,2)	340	5	(1,2)	210	5	(1,2)
SPP-2000									
: MPI	541	20	(1,0)	547	9	(1,0)	158	5	(1,0)
Origin2000									
: MPI	126	33	(1,5)	98	17	(1,3)	40	17	(1,0)
: SHMEM	166	8	(1,1)	140	8	(1,1)	71	10	(1,2)

TABLE 3

Peak observed bandwidth (MBytes/sec) and latency (microseconds) for optimal protocol.

In the spectral transform method used in PSTSWM, fields are transformed at each timestep between the physical (longitude-latitude-vertical) domain and the Fourier (wavenumber-latitude-vertical) domain using Fourier transforms in the longitude direction, and between the Fourier and spectral (spectral coefficients - vertical) domains using a Legendre transform in the latitude direction. All parallel algorithms in PSTSWM are based on decomposing the different computational domains onto a logical two-dimensional grid of processors, $PX \times PY$. In each domain, two of the domain dimensions are decomposed across the processor grid, for example, assigning longitude-latitude patches of the physical domain to individual processors, but leaving one domain dimension undecomposed.

Two general types of parallel algorithms are used in PSTSWM: transpose and distributed. In a transpose algorithm, the decomposition is “rotated” before a transform begins, to ensure that all data needed to compute a particular transform is local to a single processor. In a distributed algorithm the original decomposition of the domain is retained, and communication is performed to allow the processors to cooperate in the calculation of a transform.

Three transpose algorithms are examined, each of which is functionally equivalent to `MPI_ALLTOALLV`:

- **srtrans**: sends $P-1$ messages using `SENDRECV` to transpose across P processors;
- **swtrans**: sends $P-1$ messages using `SWAP` to transpose across P processors;
- **logtrans**: sends $\Theta(\log P)$ messages using `SWAP` to transpose across P processors.

Each of these are options for both the parallel Fourier and parallel Legendre transform algorithms. Here we restrict our study to transpose-based parallel Fast Fourier transform algorithms. One distributed Fast Fourier transform is also examined:

- **dfft**: sends $\Theta(\log P)$ messages using `SWAP` to calculate Fourier transform distributed across P processors.

The distributed Legendre transform algorithms in PSTSWM are based on the evaluation of distributed vector sums. Four distributed vector sum algorithms are examined, the first three of which are functionally equivalent to `MPI_ALLREDUCE`:

- **exchsum**: an exchange-based algorithm implemented using `SWAP`;
- **halfsum**: a recursive halving-based algorithm implemented using `SWAP`;
- **ringsum**: a ring-based algorithm implemented using `SENDRECV`;
- **ringpipe**: a pipeline-based algorithm implemented using `SENDRECV`.

Each of these algorithms can be implemented using the protocols described in Table 2. Two different types of implementations are also supported. The first uses the basic `SWAP` and `SENDRECV` commands to exchange the data. The second reorders

the elements of the SWAP or SENDRECV protocol in an attempt to overlap communication with computation and to hide communication latency. These algorithms and protocols are described in more detail in [5]. The overlap algorithms using unordered and ordered communication protocols will be designated by $(2, x)$ and $(3, x)$ respectively, where $x \in \{1, 2, 3, 4, 5, 6\}$.

To examine the performance issues in these different implementation options, we run the following experiments. We use one-dimensional decompositions of the form 8×1 or 1×8 and 32×1 or 1×32 , where the first decomposition in each pair is for examining parallel Fourier transform algorithms, and the second is for examining parallel Legendre transform algorithms. The problem sizes are based on T42L16 and T85L32 as they would appear on a two-dimensional processor grid of size 8×8 , 16×32 , or 32×16 . This is accomplished by modifying the problem size to achieve the desired granularity (problem size per processor), and allows us to examine the performance for problem granularities that are typical of what would be seen in practice.

Results are presented in Table 4. The first column is the overall best protocol for each parallel algorithm. Multiple protocols are given when no single protocol is good for all problem sizes and numbers of processors. The other columns indicate how much performance is lost by using the MPI.SENDRECV-based protocol instead of the optimal MPI protocol and by using the optimal MPI protocol instead of the optimal SHMEM protocol. Note that these are total runtimes, and that the indicated performance loss is a function of both the size of the messages and the communication/computation ratio for a given experiment.

From this data it is clear that there is no reason to use anything but $(0,6)$ on the T3E if using MPI, but that significant performance gains are possible if the SHMEM library is used instead. Note that, unlike with MPI, the overlap algorithms are optimal for some of the SHMEM experiments, indicating that overlap logic can be useful with this architecture if the message-passing library supports it.

On the Origin2000, the conclusions are less clear. While $(0,6)$ is rarely optimal, it is a good choice for all but a few cases. For those few cases, however, it should not be used. Similarly, MPI is competitive with (or better than) SHMEM in most cases, but MPI performs much worse than SHMEM for some of the smaller granularity cases.

7. Full Simulation Performance. Efficient parallelizations of PSTSWM exploit two-dimensional decompositions of the domain, parallelizing both the Fourier and Legendre transforms. Here we consider two classes of parallel algorithms.

- **DTH:** double transpose for the Fourier transform and **halfsum** for the Legendre

		T3E							
opt. protocols		$\frac{t_{(0,6),mpi} - t_{opt,mpi}}{t_{opt,mpi}}$				$\frac{t_{opt,mpi} - t_{opt,shmem}}{t_{opt,shmem}}$			
		$P = 8$		$P = 32$		$P = 8$		$P = 32$	
		T42	T85	T42	T85	T42	T85	T42	T85
dfft	(0,6),(2,2)	0%	8%	0%	0%	18%	6%	54%	18%
exchsum	(0,6)	0%	0%	0%	0%	7%	3%	30%	16%
halfsum	(0,6)	0%	0%	0%	0%	7%	2%	39%	11%
logtrans	(0,6)	0%	0%	1%	0%	16%	7%	82%	20%
ringpipe	(0,6)	0%	0%	0%	0%	14%	4%	75%	39%
ringsum	(0,6)	0%	0%	0%	0%	11%	3%	87%	24%
srtrans	(0,6)	2%	0%	0%	0%	17%	5%	121%	27%
swtrans	(0,6)	1%	0%	0%	0%	17%	5%	143%	29%

		Origin2000							
opt. protocols		$\frac{t_{(0,6),mpi} - t_{opt,mpi}}{t_{opt,mpi}}$				$\frac{t_{opt,mpi} - t_{opt,shmem}}{t_{opt,shmem}}$			
		$P = 8$		$P = 32$		$P = 8$		$P = 32$	
		T42	T85	T42	T85	T42	T85	T42	T85
dfft	(0,1),(3,3)	1%	22%	0%	10%	7%	7%	-4%	-10%
exchsum	(0,4),(0,6)	0%	20%	0%	48%	-4%	0%	-11%	32%
halfsum	(0,1),(0,5)	1%	0%	5%	27%	3%	2%	5%	-5%
logtrans	(0,6),(1,1)	0%	9%	0%	3%	6%	14%	59%	23%
ringpipe	(2,1),(2,2)	6%	4%	1%	3%	0%	-5%	78%	3%
ringsum	(0,1)	2%	1%	1%	4%	0%	-9%	93%	-7%
srtrans	(0,1)	0%	0%	0%	1%	-1%	-3%	115%	35%
swtrans	(0,1)	1%	0%	0%	1%	0%	-3%	116%	37%

TABLE 4

Effect of protocol on performance of parallel algorithms.

transform. The double transpose algorithm uses a transpose to serialize the Fourier transforms, then another transpose to return to a domain decomposition analogous to the original. This approach has the best load balance among the parallel algorithm options. **halfsum** is the best MPI_ALLREDUCE-equivalent algorithm on the T3E and the Origin2000.

- **DR: dfft/ringpipe.** This parallel algorithm combination has good load balance, requires the minimum storage, and has the maximum potential for communication/computation overlap.

DTH and **DR** stress the underlying transport mechanisms in significantly different ways, and represent different tests of the communication protocol sensitivity. Due to their good load balances, the performance differences between them reflect primarily the differences in communication costs.

For each platform we measure the runtimes when solving T42L16 and T85L16 using

- *opt*: the best transpose algorithms (for **DTH**) and the best communication protocols for each parallel algorithm, determined empirically,
- *gen*: **srtrans** (for **DTH**) and (0,6)-based parallel implementations, and
- *coll*: MPI collective communication routines **MPI_ALLTOALLV** and **MPI_REDUCEALL** (for **DTH**),

for logical processor meshes of sizes: 4×4 , 4×8 , 8×8 , 8×16 , 16×16 , and 16×32 . Algorithms *gen* and *coll* represent the typical algorithm choices if nothing is known about the communication protocol sensitivities. Measurements are also taken using 8×14 for **DR** and 14×8 for **DTH**, since the 128 processor experiments do not run efficiently on a 128 processor Origin2000 (due to competition with system processes).

Results are presented in Table 5. The optimal times are given for both MPI and SHMEM implementations. Additionally, the performance degradation (if any) is given for using the *gen* and *coll* implementations instead of the optimal MPI implementation.

- *T3E results*. For **DTH**, *gen* is the best MPI implementation except for the smallest granularity cases. In those two cases *coll* is the best, but *coll* is an erratic performer in general. For **DR**, *gen* is never the best, and it is worthwhile searching for the optimal MPI protocol. But the optimal SHMEM implementations are faster than the optimal MPI implementations in all cases, and often significantly so.
- *Origin2000 results*. For **DTH**, *gen* is a reasonable choice for the T42L16 cases, but the optimal MPI protocols are worth identifying for the T85L16 cases. *coll* is never a good choice. For **DR**, *gen* is a poor choice, and it is worthwhile searching for the optimal MPI protocols. The optimal SHMEM implementations are faster than the optimal MPI implementations only for the largest granularity cases. In the other cases, the optimal MPI implementations are consistently better.

8. Summary. Both the T3E and the Origin2000 results indicate the importance of considering the interprocessor communication protocols when tuning performance, but the similarity in the results ends there. On the T3E, performance is optimized by using the SHMEM communication library. On the Origin2000, optimization should include both the communication library (MPI or SHMEM) and the particular protocol used in the implementation. Disappointingly, the collective communication-based implementation *coll* is not competitive on either platform, which is consistent with earlier evaluations on other parallel platforms [4].

			T3E						
alg.	prot.	library	4 × 4	4 × 8	8 × 8	8 × 14	8 × 16	16 × 16	16 × 32
			T42L16 runtimes						
DTH	opt	MPI	30.9	14.8	7.4	4.5	4.1	2.5	2.0
	opt	SHMEM	29.6	13.9	6.5	3.5	3.2	2.0	1.2
DR	opt	MPI	23.6	12.5	7.3	6.9	5.7	4.4	-
	opt	SHMEM	22.2	11.2	6.0	5.2	4.0	2.8	-
			T85L16 runtimes						
DTH	opt	MPI	311.3	149.8	69.8	38.6	36.2	20.6	12.4
	opt	SHMEM	304.6	144.3	65.6	36.4	33.4	17.0	9.4
DR	opt	MPI	228.9	116.3	61.7	39.1	37.4	21.5	18.9
	opt	SHMEM	221.1	110.8	57.0	35.7	29.5	16.9	12.6
			T42L16 MPI performance sensitivity						
DTH	gen	MPI	0%	0%	3%	0%	0%	17%	13%
	coll	MPI	1%	5%	0%	7%	7%	0%	0%
DR	gen	MPI	5%	8%	3%	2%	4%	2%	-
			T85L16 MPI performance sensitivity						
DTH	gen	MPI	0%	0%	0%	0%	0%	0%	0%
	coll	MPI	1%	6%	5%	17%	16%	2%	23%
DR	gen	MPI	12%	7%	4%	4%	7%	8%	7%

			Origin2000						
alg.	prot.	library	4 × 4	4 × 8	8 × 8	8 × 14	8 × 16	16 × 16	16 × 32
			T42L16 runtimes						
DTH	opt	MPI	17.5	9.9	5.9	4.9	-	-	-
	opt	SHMEM	18.7	10.4	6.0	-	-	-	-
DR	opt	MPI	18.3	10.3	6.8	6.2	-	-	-
	opt	SHMEM	18.6	10.6	7.2	7.6	-	-	-
			T85L16 runtimes						
DTH	opt	MPI	250.9	126.4	52.5	42.6	-	-	-
	opt	SHMEM	244.1	112.7	54.8	-	-	-	-
DR	opt	MPI	250.6	122.5	56.2	40.1	-	-	-
	opt	SHMEM	234.0	104.8	56.4	54.5	-	-	-
			T42L16 MPI performance sensitivity						
DTH	gen	MPI	0%	1%	3%	4%	-	-	-
	coll	MPI	28%	37%	14%	16%	-	-	-
DR	gen	MPI	36%	18%	3%	5%	-	-	-
			T85L16 MPI performance sensitivity						
DTH	gen	MPI	2%	5%	13%	17%	-	-	-
	coll	MPI	11%	30%	120%	111%	-	-	-
DR	gen	MPI	14%	35%	92%	24%	-	-	-

TABLE 5

Runtimes of 5 day simulations of PSTSWM (seconds) and performance degradation from using gen and coll algorithms: $(t_{\text{gen,mpi}} - t_{\text{opt,mpi}})/t_{\text{opt,mpi}}$ and $(t_{\text{coll,mpi}} - t_{\text{opt,mpi}})/t_{\text{opt,mpi}}$.

9. Acknowledgements. This research was supported by the U.S. Department of Energy under Contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Inc. We thank NASA-Ames for access to their SP2 system, and Cray Research for access to a T3D system. We thank the Advanced Computing Laboratory at Los Alamos National Laboratory for access to the SGI/Cray Research Origin2000. The Intel XP/S 150 MP Paragon operated by the Center for Computational Science at ORNL is funded by the Department of Energy's Mathematical, Information and Computational Sciences Division of the Office of Computational and Technology Research. Access to the CONVEX Exemplar SPP-1200 and the HP/CONVEX Exemplar SPP-2000 was supported by the National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign under grant number ASC960028N. Access to the SGI/Cray Research T3E-900 at the National Energy Research Scientific Computing Center was supported by the Environmental Sciences Division, U.S. Department of Energy.

REFERENCES

- [1] G. R. LUECKE, J. J. COYLE, AND W. UL HAQUE, *Comparing communication performance of MPI on the Cray Research T3E-600 and IBM SP-2*, Performance Evaluation and Modelling of Computer Systems, (1997). <http://hpc-journals.ecs.soton.ac.uk/PEMCS/>.
- [2] MPI COMMITTEE, *MPI: a message-passing interface standard*, Internat. J. Supercomputer Applications, 8 (1994), pp. 165-416.
- [3] D. L. WILLIAMSON, J. B. DRAKE, J. J. HACK, R. JAKOB, AND P. N. SWARZTRAUBER, *A standard test set for numerical approximations to the shallow water equations on the sphere*, Tech. Report ORNL/TM-11773, Oak Ridge National Laboratory, Oak Ridge, TN, 1991.
- [4] P. H. WORLEY, *MPI performance evaluation and characterization using a compact application benchmark code*, in Proceedings of the Second MPI Developers Conference and Users' Meeting, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 170-177.
- [5] P. H. WORLEY AND B. TOONEN, *A users' guide to PSTSWM*, Tech. Report ORNL/TM-12779, Oak Ridge National Laboratory, Oak Ridge, TN, July 1995.