

# ESMF Array Bundle Sparse Matrix Multiplication Benchmark

Peggy Li

November 4, 2008

## Objective

In December 2007, we conducted a performance evaluation on the ESMF Array Sparse Matrix Multiplication functions and published the results on the ESMF website ([http://www.esmf.ucar.edu/metrics/performance/timing\\_0712\\_asmm-1.pdf](http://www.esmf.ucar.edu/metrics/performance/timing_0712_asmm-1.pdf)). In the report, we identified several scalability problems in ESMF 3.1.0:

- The ESMF VM memory usage increases more than linearly when using 512 and more PEs.
- The ASMMStore time increases when using 256 and more PEs on the Cray XT4.
- The ASMM run time does not scale well on 512 and more PE counts on the Cray XT4.

The VM Memory has been optimized in the ESMF public release 3.1.0r. An ESMF memory scalability benchmark was conducted and the result was published in July 2008 at [http://www.esmf.ucar.edu/metrics/performance/timing\\_0807\\_memory\\_scalability-1.pdf](http://www.esmf.ucar.edu/metrics/performance/timing_0807_memory_scalability-1.pdf). In this report, we are going to address the other two performance findings, i.e., the scalability issues of the ASMM init and run routines. In the previous benchmark we used an array with only one variable, whereas in the real world applications, we usually have multiple fields associated with one distgrid and we bundle them together when we redistribute them among components. Therefore, the ASMM performance based on one variable may not accurately reflect its performance in an application using multiple variables. In this benchmark, we modified the program to measure the ASMM performance with multiple variables and we compare the results with its MCT counterpart.

The performance evaluation was conducted on the IBM Blue Gene/L at UCAR (frost) using 4 to 512 processors and on the Cray XT4 at Oak Ridge National Laboratory (jaguar) using 4 to 16,384 processors. The IBM Blue Gene/L is running Linux and IBM XL 10.1 compilers. The Cray XT4 is running CNL (Computer Node Linux) OS with PGI 7.2.4 compilers. We used ESMF 3.1.0 beta snapshot #81 and MCT 2.5.1 to run this benchmark on both machines.

## Benchmark Program

Both the ESMF and MCT benchmark programs consist of three components, a source Grid Component, a destination Grid Component, and a Coupler component, and a driver program responsible for the creation, initialization, execution and termination of the components. Note there is no component concept in MCT. We organize the MCT code into init, run, and finalize functions mimicking the ESMF component structure. Both the source and the destination arrays are based on a 2D distgrid. The source distgrid is a 720x360 grid distributed regularly in a row-column block fashion and the destination distgrid is a 1080x640 grid distributed in a slightly different row-column blocks. Table 1

describes the data decomposition and the block size for each configuration.

#processors	Source Decomp	Block Size	Destination Decomp	Block Size
4	4x1	180x360	2x2	540x320
8	2x4	360x90	4x2	270x320
16	8x2	90x180	4x4	270x160
32	4x8	180x45	8x4	135x160
64	16x4	45x90	8x8	135x80
128	8x16	90x23	16x8	68x80
256	32x8	23x45	16x16	68x40
512	16x32	45x12	32x16	34x40
1024	64x16	12x23	32x32	34x20
2048	32x64	23x6	64x32	17x20
4096	128x32	6x12	64x64	17x10
8192	64x128	12x3	128x64	9x10
16384	256x64	3x6	128x128	9x5

**Table 1: Data Decomposition and the block size**

Both source and destination arrays have 10 single-precision floating point fields. In the ESMF benchmark, we implemented the multiple-field arrays in two ways: a 3D ESMF\_Array and ESMF\_ArrayBundle. When using the 3D ESMF\_Array, we made the undistributed dimension the first dimension of the array, thus the multiple fields for a given grid point are stored contiguously in memory. In MCT, we use a AttributeVector with 10 fields of type REAL. For the SMM initialization time, we measured *ESMF\_ArraySMMStore()* and *ESMF\_BundleSMMStore()* once in ESMF, and *m\_SparseMatrixPlus::init()* in MCT. For the SMM run time, we called *ESMF\_ArraySMM()* and *ESMF\_BundleSMM()* five times, reported the maximal time among all the processors for each time it is called, then chose the smallest time of the five runs. We did the same for MCT except that we measured the time for *m\_MatAttrVecMul::sMatAvMult()* instead.

## The Results

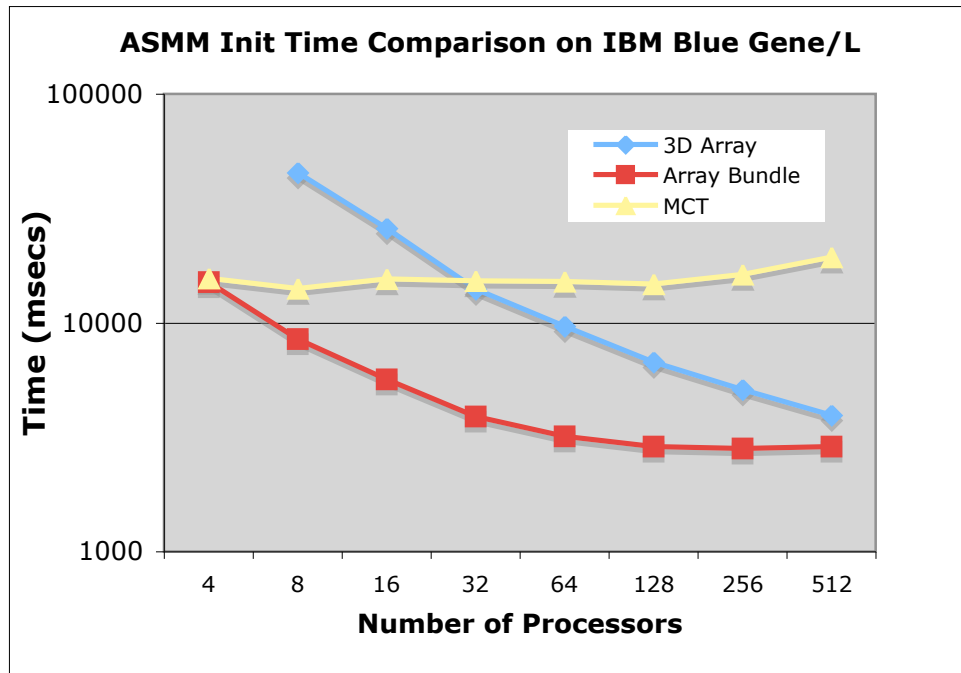
### ESMF and MCT Comparison on the IBM Blue Gene/L

We ran the ESMF and MCT benchmark programs on the IBM Blue Gene/L at UCAR (frost) using 4 to 512 processors. We compare the two different implementations in ESMF, 3D ESMF\_Array and ESMF\_ArrayBundle with the MCT AttributeVector. Table 2 shows the ASMM initialization and run time results of the three different implementations. Figure 1 shows the comparison of the ASMM init time. Figure 2 depicts the comparison of the ASMM run time. MCT's ASMM Init time remains constant regardless number of processors used. On the contrary, the ESMF ASMM init time decreases almost linearly with increasing number of processors. The initialization time for the 3D Array is almost 5 times more comparing to the ArrayBundle with small PE counts. The time for the ArrayBundle flattened out with 64 PEs and above and the time gap between the 3D Array and ArrayBundle are getting smaller with large PE

counts. The ESMF 3D Array benchmark ran out of memory on 4PEs, thus no timing results were reported.

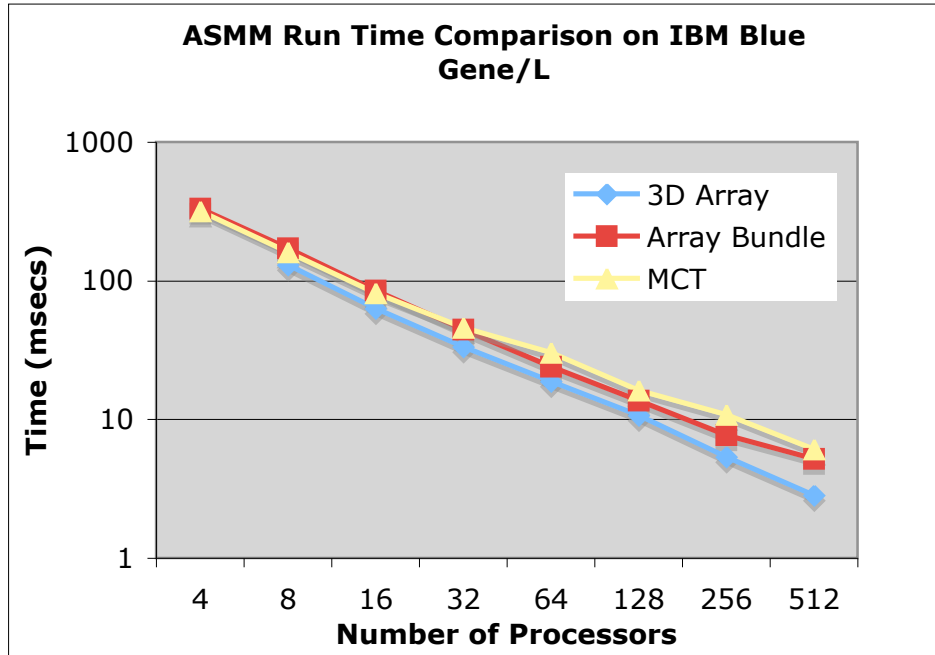
# nodes	ASMM Initialization (msecs)			ESMF_ASMM Run (msecs)		
	3D Array	Array Bundle	MCT	3D Array	Array Bundle	MCT
4		15012.50	15689.2		332.334	317.216
8	45343.54	8502.00	14142.14	129.438	170.666	161.419
16	25841.14	5649.95	15502.18	63.127	84.675	81.446
32	14019.44	3894.92	15204.15	33.475	44.619	46.024
64	9660.01	3185.62	15107.27	18.872	23.922	30.264
128	6707.92	2870.22	14759.39	10.678	13.637	16.19
256	5102.50	2818.87	16241.22	5.387	7.692	10.805
512	3938.10	2883.12	19307.57	2.836	5.192	6.112

*Table 2 The ASMM Timing on IBM Blue Gene/L*



*Figure 1 ASMM Init Time Comparison between ESMF and MCT*

The ASMM run time for the MCT and ESMF ArrayBundle are pretty close. The 3D ESMF\_Array implementation outperformed the ESMF\_ArrayBundle by 30% consistently for all the configurations. The longer initialization time seems to pay off here. All three implementations scaled down linearly with increasing number of processors.



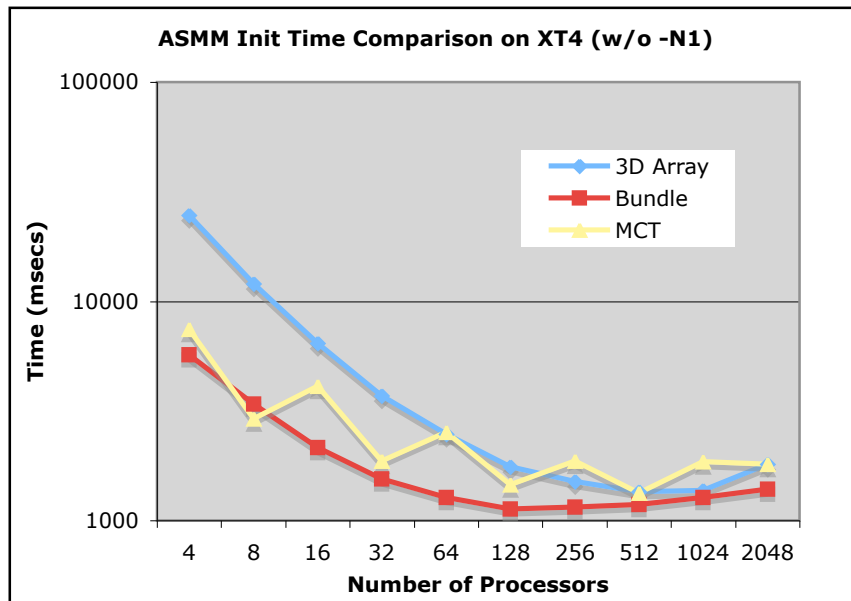
*Figure 2 ASMM Run Time Comparison between ESMF and MCT*

### ESMF and MCT Comparison on Cray XT4

We ran the ESMF and MCT benchmark programs on the Cray XT4 at ORL (jaguar) using 4 to 16,384 processors. Each Jaguar node is a AMD Opteron quad-core processor. We ran the benchmarks on all four cores per node. MCT failed to run on 4K, 8K, and 16K processors, thus we only report the results from 4 to 2048 processors here. The ESMF results on large PE counts will be reported in the next section. We compared the two different implementations in ESMF, 3D ESMF\_Array and ESMF\_ArrayBundle with the MCT AttributeVector. Table 3 shows the ASMM initialization and run time results of the three different implementations. Figure 3 shows the comparison of the ASMM init time. Figure 4 depicts the comparison of the ASMM run time. MCT's ASMM Init time is very different from what we observed on the Blue Gene. It decreases with increasing number of processors and it is very sensitive to the way data is distributed. When the source block is more row-dominant, it is faster to initialize the sparse matrix (for example, the init time for 8 PEs with a block size 360x90 is faster than the init time for 16 PEs with a block size 90x180). On the contrary, the ESMF benchmark behaves similarly as the runs on Blue Gene – the ASMM init time decreases almost linearly with increasing number of processors. The initialization time for the 3D Array is almost 4 times more comparing to the ArrayBundle with small PE counts. The time for the ArrayBundle flattened out with 64 PEs and above and the time gap between the 3D Array and ArrayBundle are getting smaller with large PE counts. The Cray XT4 outperforms IBM Blue Gene by 3 to 4 times. It is expected because the Blue Gene processor (740MHz PowerPC-440) is 3 times slower than the XT4's processor (2.1 GHz AMD Opteron).

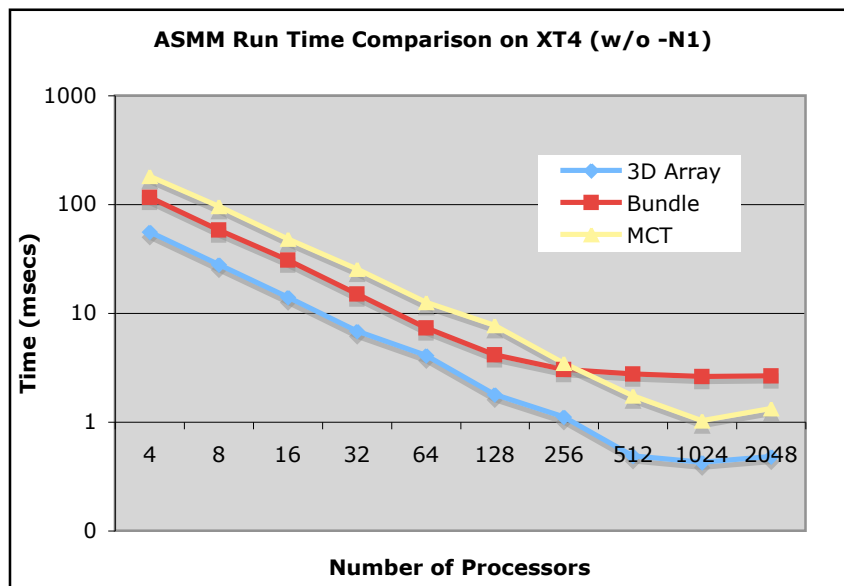
# nodes	ESMF_ASMMStore (msecs)			ESMF_ASMM (msecs)		
	3D Array	Bundle	MCT	3D Array	Bundle	MCT
4	24744.15	5715.08	7454.14	55.958	115.567	182.361
8	12011.04	3403.37	2915.97	28.155	58.312	95.638
16	6451.06	2153.69	4099.33	13.949	30.933	48.221
32	3704.76	1552.04	1863.35	6.846	14.948	25.301
64	2484.15	1272.43	2535.57	4.086	7.294	12.649
128	1760.98	1132.15	1458.51	1.779	4.172	7.768
256	1503.74	1149.65	1866.19	1.107	3.024	3.499
512	1350.50	1180.82	1337.89	0.487	2.745	1.737
1024	1373.28	1270.90	1852.74	0.424	2.600	1.025
2048	1809.87	1390.10	1805.59	0.481	2.651	1.325

*Table 3 ASMM Timing Comparison on XT4*



*Figure 3 ASMM Init Time Comparison between ESMF and MCT on XT4*

The ASMM run times for all three approaches scale down linearly up to 256 processors. The 3D ESMF\_Array approach performs the best among the three, which is consistent with the results on the Blue Gene. However, the performance difference on XT4 is much bigger than that on Blue Gene. The ESMF\_ArrayBundle approach stopped to scale at 256 PEs and the other two approaches up to 1024PEs.



*Figure 4 ASMM Run Time Comparison between ESMF and MCT on XT4*

### The ESMF ASMM Scalability Results on Cray XT4

The jaguar system contains 7,832 quad-core AMD Opteron processors and 62TB of memory (2GB memory per core). When we schedule a job, it runs on all the four cores. We can choose to use any number of cores per node when running the job using *aprun* with a switch `-N #`. By using fewer cores per node, we can reduce the socket contention due to the MPI communication calls, thus improving the overall performance. For the scalability benchmark, we ran the ESMF benchmark program using all four cores (without `-N1`) and one core per node (with `-N1`). Since we use more nodes with `-N1`, we can only run up to 4096 PEs. In Table 4, we report the results from 4PEs up to 16384PEs. In Figure 5 and 6, we compare the timings of the ASMM Init and ASMM run routines using the 3D Array and ArrayBundle and running with `-N1` and without `-N1`.

Consistent with the results from the IBM Blue Gene/L, the 3D Array SMM initialization is much slower than the ArrayBundle. The SMM run time for the two different approaches are pretty comparable with smaller processor count (256 or below). The 3D Array SMM run routine scales better than the ArrayBundle in large PE counts, as a result, it performs better than the ArrayBundle for processors 512 and above.

We observed about 20% performance improvement for the ASMM Initialization time when using only one core per node. The improvement for the ASMM run time is much bigger – 2-3 times faster for all the configurations. From Figure 5 and Figure 6, we see both the SMM init and run routines scale down linearly with increasing number of processors initially and start to flatten out at certain point. In general, The ESMF\_ArrayBundle approach scales worse than the 3D ESMF\_Array approach. It is worth noting that the problem size in the benchmark is too small to run on 4K or more processors (see the block size in Table 1). With only a few hundred elements per block,

the latency dominates the communication time thus the program stops to scale. We should either increase the problem size or scale the problem size according to the number of processors used in order to get a more accurate scalability results.

We also ran the MCT benchmark using one core per node configuration. The timing difference between one core per node and 4 core per node runs are very small in both the SMM init and run time. We did not report the MCT timing results using one core per node in this report.

# nodes	ESMF_ASMMStore (msecs)				ESMF_ASMM (msecs)			
	3D Array	Bundle	3D Array (-N1)	Bundle (-N1)	3D Array	Bundle	3D Array (-N1)	Bundle (-N1)
4	24744.15	5715.08	20112.52	4664.88	55.958	115.567	32.216	60.672
8	12011.04	3403.37	9724.91	2842.59	28.155	58.312	16.541	32.388
16	6451.06	2153.69	5174.64	1822.00	13.949	30.933	8.497	17.101
32	3704.76	1552.04	3036.23	1342.62	6.846	14.948	4.489	6.595
64	2484.15	1272.43	1997.16	1143.10	4.086	7.294	2.319	3.442
128	1760.98	1132.15	1502.47	1074.69	1.779	4.172	0.874	2.143
256	1503.74	1149.65	1367.88	1115.18	1.107	3.024	0.406	1.281
512	1350.50	1180.82	1271.39	1132.12	0.487	2.745	0.309	1.063
1024	1373.28	1270.90	1287.27	1205.00	0.424	2.600	0.193	1.010
2048	1809.87	1390.10	1400.44	1325.49	0.481	2.651	0.144	0.883
4096	1674.40	1532.28	1500.11	1483.19	0.385	2.660	0.122	0.866
8192	1932.57	1816.13			0.372	2.800		
16384	2507.81	2506.96			0.299	2.840		

*Table 4 The Timing Result form the Cray XT4*

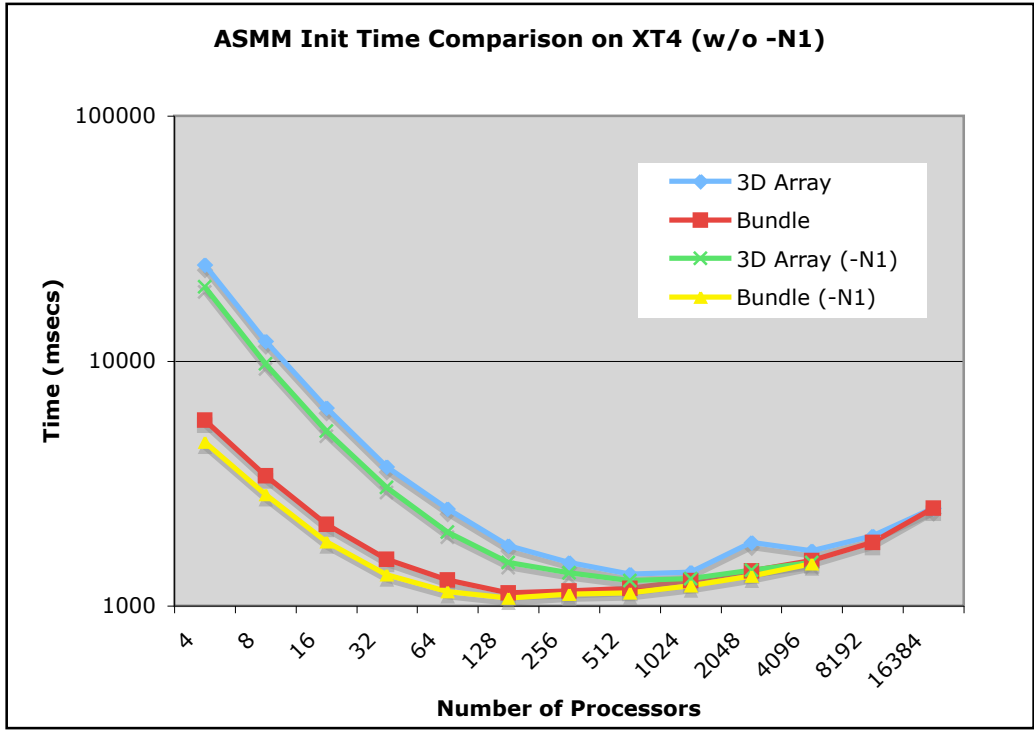


Figure 5 ESMF ASMM Init Time Scalability on Cray XT4

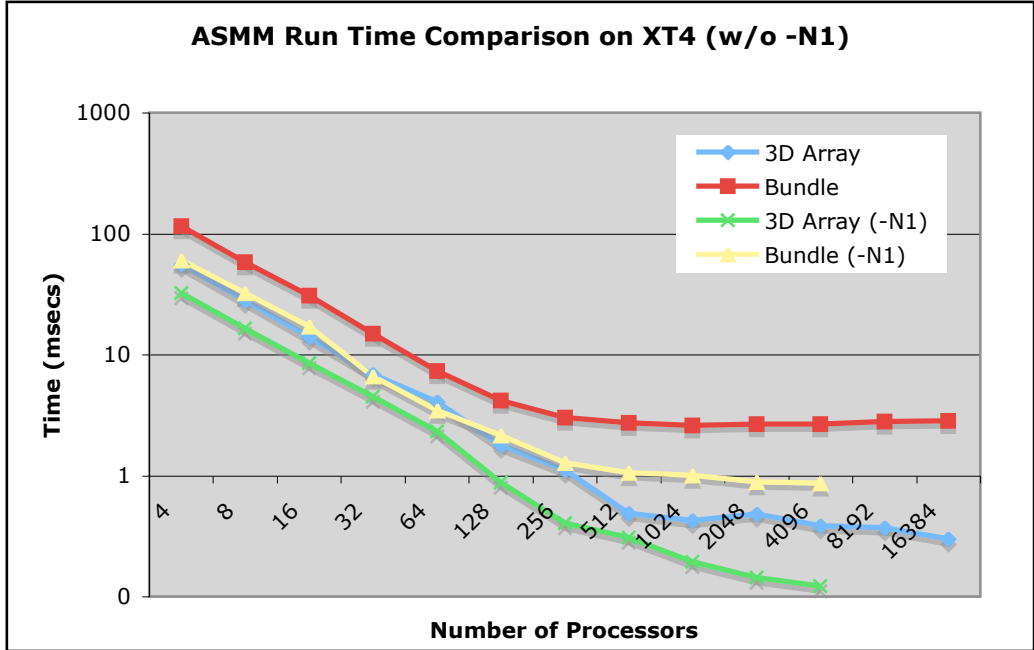


Figure 6 ESMF ASMM Run Time Scalability on Cray XT4

## Conclusion

With the recent optimization in the ESMF implementation, we see ESMF SMM run routine performs equally or better than MCT on the IBM Blue Gene/L and the Cray XT4 for all the configurations. The 3D ESMF\_Array approach (with the first dimension of the 3D array representing the fields) performs better than the ESMF\_ArrayBundle approach. The 3D ESMF\_Array approach also scales better with large PE counts. In other words, there is still rooms for more optimization for the ESMF\_ArrayBundle implementation.

The source code of the benchmark program can be downloaded at the SourceForge ESMF Contributions site:

```
cvs -z3 -d:pserver:anonymous@esmfc.contrib.sourceforge.net:/cvsroot/esmfcontrib co  
-r V_01 -P performance_tests/ArrayBundleSMM
```

*Note #1: The source code was tagged as V\_01 on January 18, 2012.*

*Note #2: The MCT represents “real” attributes internally as double-precision floating point numbers. In this benchmark, we used single precision float point arrays in the ESMF benchmark. An updated report using double-precision floating point numbers for both ESMF and MCT can be found in the ESMF website with URL [http://earthsystemmodeling.org/metrics/performance/timing\\_1012\\_asmm.pdf](http://earthsystemmodeling.org/metrics/performance/timing_1012_asmm.pdf). The ESMF ASMM Init time is about the same and the ASMM run time is about twice slower comparing to the results using single-precision numbers. (updated on April 16, 2012)*