

# ESMF Bundle Redistribution Scalability Benchmark

Peggy Li  
July 1, 2007

## Objective

The objective of this task is to evaluate the performance of the ESMF bundle redistribution functions on large number of processors, i.e., over 1000 processors. The benchmark measures both the redistribution initialization and run routines, namely, ESMF\_BundleRedistStore() and ESMF\_BundleRedist(). The benchmark is limited to block distributed grids.

We conducted the performance evaluation on the Cray XT3/XT4 at Oak Ridge National Laboratory and the SGI Altix Superclusters at NASA Ames. The Cray XT3/XT4 is running UNICOS/lc 1.4.35 operating system with PGI 6.1.6 compilers and the SGI Altix is running Linux 2.6.5-7.276-sn2 with Intel compilers version 9.1.039 and SGI Message Passing Toolkit (MPT) version 1.12.0. The ESMF version used to run the benchmark is ESMF 3.0.1 snapshot #06.

## Benchmark Program

The benchmark program consists of two files: the main program ESMF\_BundleRedistPerfTest.F90 and the helper program ESMF\_BundleRedistHelpers.F90. The program creates a 3D source grid and an identical destination grid. The source grid and the destination grid are partitioned into local processors in different ways. The program then creates the source bundle and the destination bundle each containing 8 double precision floating point fields. The source bundle fields are assigned with values calculated using a linear combination of sin waves with different frequencies and amplitudes. The formula will be used to validate the correctness of the redistribution. We measure the time to run ESMF\_BundleRedistStore() and the average time of 50 ESMF\_BundleRedist() calls. We used ESMF\_VMWTime() to get the elapse time. We then run the program on each configuration 10 times and report the minimal time from the 10 runs.

We ran the benchmark with two different grid size. The first grid is a middle-size grid with 512x1024x16, or 8 Mega, sample points. The total memory for an 8 field bundle is 512 Mbytes (536,870,912 bytes). The second grid is a bigger grid with size 2048x4096x8, or 256 Mega, sample points. The total memory for an 8 field bundle is 16 Gbytes. The source and the destination grid distribution are chosen to minimize the overlap of the grid points in each local processor thus maximizing the number of bytes transferred during the redistribution. Table 1 shows the distribution (DELayout) of the source grid and the destination grid and the number of total bytes transferred in each redistribution step for the 512x1024x16 grid.

**Table 1 The source and destination grid distribution**

# processors	src grid partition	dst grid partition	Total bytes redistributed (bytes)
4	4x1	1x4	402653184
8	2x4	4x2	402653184
16	2x8	8x2	469762048
32	4x8	16x2	503316480
64	4x16	16x4	503316480
128	8x16	32x4	520093696
256	16x16	32x8	520093696
512	32x16	16x32	520093696
1024	32x32	64x16	528482304
2048	32x64	64x32	528482304

## The Results

ESMF provides several routing options for user to specify when ESMF\_BundleRedistStore() is called. The routing options determine how the ESMF Route object executes communication internally. We measured the times with different routing options:

### Synchronous vs Asynchronous

ESMF\_ROUTE\_OPTION\_ASYNC and ESMF\_ROUTE\_OPTION\_SYNC are two mutually exclusive options. ESMF uses asynchronous communication routines (i.e., MPI\_Isend and MPI\_Irecv) if ESMF\_ROUTE\_OPTION\_ASYNC is set. Otherwise, it uses synchronous communication routines (i.e. MPI\_Sendrecv). In addition to the communication mode, there are three packing options user can set, namely, ESMF\_ROUTE\_OPTION\_PACK\_PET, ESMF\_ROUTE\_OPTION\_PACK\_XP, ESMF\_ROUTE\_OPTION\_PACK\_NOPACK. These three packing options are also mutually exclusive. When we measure the performance for the sync and async communication options, we use ESMF\_ROUTE\_OPTION\_PACK\_PET.

The timing difference was measured on Cray XT3 using a 1024x512x16 grid. As shown in Figure 1 and Figure 2, the time for ESMF\_BundleRedistStore is about the same for both cases since the communication mode does not affect the pre-calculation of the route. The run time for the synchronous mode is slightly slower than that of asynchronous mode.

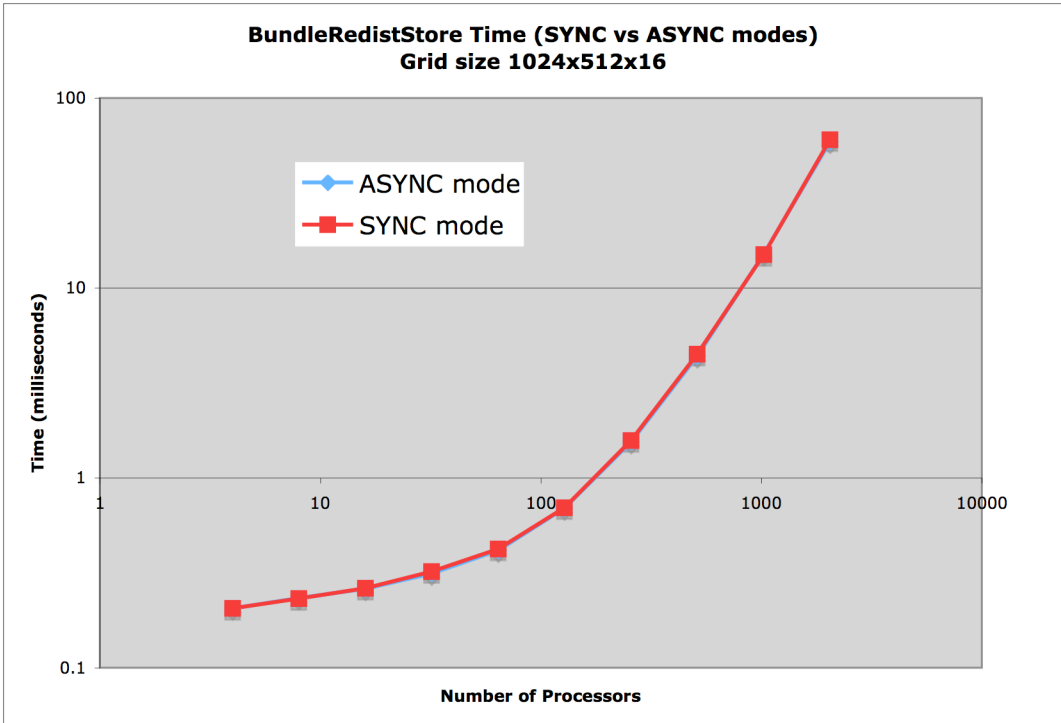


Figure 1 BundleRedistStore timing on Cray XT3

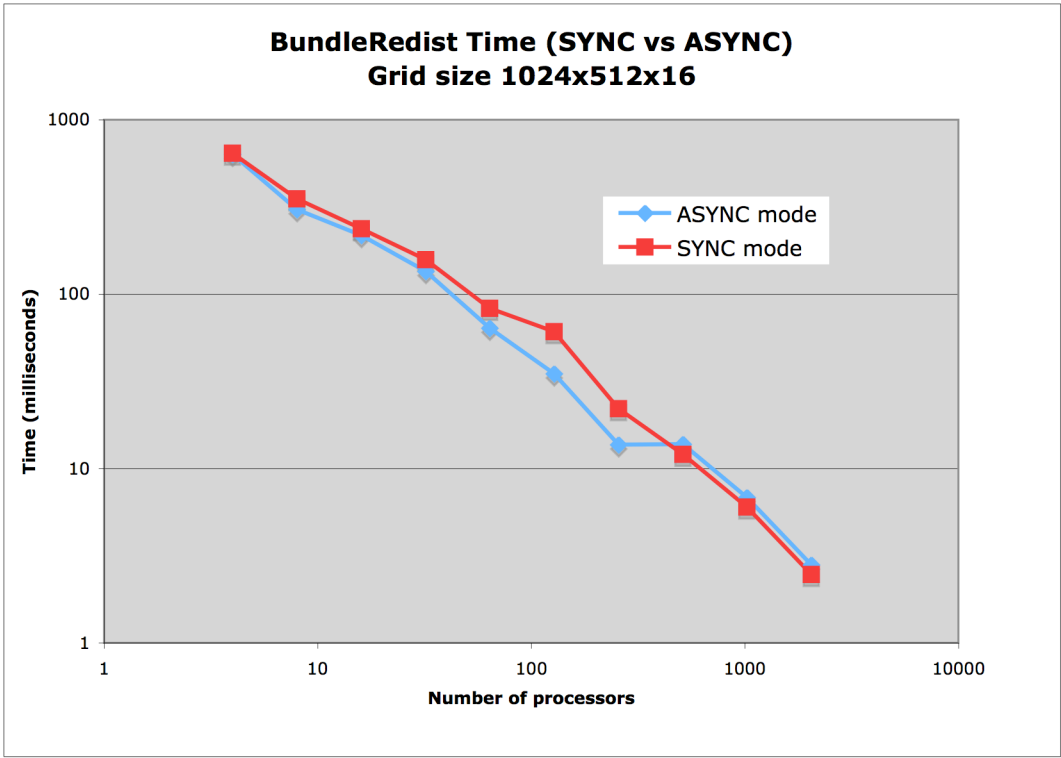


Figure 2 BundleResitRun time on Cray XT3

## Comparison of three different packing modes

There are three packing options for ESMF communication functions: ESMF\_ROUTE\_OPTION\_PACK\_NOPACK, ESMF\_ROUTE\_OPTION\_PACK\_PET and ESMF\_ROUTE\_OPTION\_PACK\_XP. With ESMF\_ROUTE\_OPTION\_PACK\_PET, the data sent to each destination PET will be packed into one single message. Therefore, there is one message for each pair of source and destination PETs. With ESMF\_ROUTE\_OPTION\_PACK\_XP, the non-contiguous data stored in one XPacket will be packed into one single message before it is sent. In the block distributed grid, the intersection of the local blocks for any pair of source and destination PETs is represented by one XPacket. therefore ESMF\_ROUTE\_OPTION\_PACK\_XP is the same as ESMF\_ROUTE\_OPTION\_PACK\_PET. It is not the case in the arbitrarily distributed grid though -- since one XPacket only contains one grid point, ESMF\_ROUTE\_OPTION\_PACK\_XP will be equivalent to ESMF\_ROUTE\_OPTION\_PACK\_NOPACK. Figure 3 shows the timing for ESMF\_BundleRedist() for three different packing options. ESMF\_ROUTE\_OPTION\_PACK\_NOPACK is much slower than the other two options since there are many smaller messages sent between any pair of source and destination PETs and the latency of the messages is much higher than the cost of packing and unpacking buffers. The timings for the PACK\_PET and PACK\_XP cases are almost identical as expected. The timings for ESMF\_BundleRedistStore() are indistinguishable for three cases thus the result is not shown here.

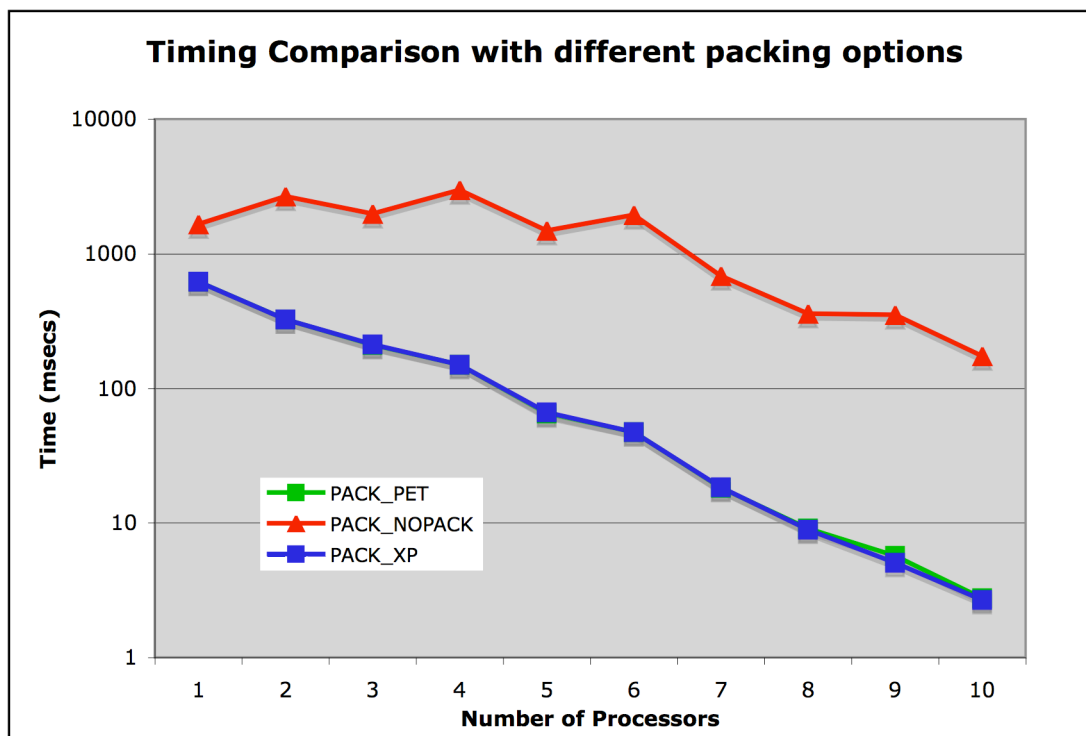


Figure 3 ESMF\_BundleRedist() Time for three different packing options

In addition to the five routing options described above, there are two more routing options user can set, i.e., `ESMF_ROUTE_OPTION_PACK_VECTOR` and `ESMF_ROUTE_OPTION_PACK_BUFFER`. `ESMF_ROUTE_OPTION_PACK_VECTOR` is not implemented and you get an error message if you set it. `ESMF_ROUTE_OPTION_PACK_BUFFER` is quite different from the other three packing options. If not set, the redistribution route of each field in the bundle will be pre-calculated separately and the redistribution will be done one field at a time. If it is set, the redistribution route will be calculated once and all the fields will be packed into one message for redistribution. It is not clear to me why we ever need to unset `ESMF_ROUTE_OPTION_PACK_BUFFER` or use `ESMF_ROUTE_OPTION_PACK_NOPACK` since they will always make the program run slower.

In conclusion, the best routing option for bundle redistribution is the default routing option, i.e. the combination of `ESMF_ROUTE_OPTION_ASYNC`, `ESMF_ROUTE_OPTION_PACK_PET` and `ESMF_ROUTE_OPTION_PACK_BUFFER`. It is recommended not to change the route options unless the user has a clear understanding how the system performs.

#### **SGI Altix vs Cray XT3/XT4**

We did the benchmark on both the SGI Altix Supercluster, Columbia, at NASA AMES and the Cray XT3/XT4, jaguar, at Oak Ridge. We compare the performance difference of the two machines using two different sizes of grids. Figure 4 and Figure 5 are the timing results for the 1024x512x16 grid. Figure 6 and 7 show the results for the 2048x4096x8 grid. We ran the configuration up to 2048 processors on jaguar and up to 1024 processors on Columbia's 2048 system. Since the maximal number of compute processors we can use on Columbia is 2032, we are not able to run more than 1024 processors. In the large grid case, we ran out of memory with 4 processors on Columbia and with both 4 and 8 processors on Jaguar. On Jaguar, we are using both processors in each node, thus each processor has less memory to use.

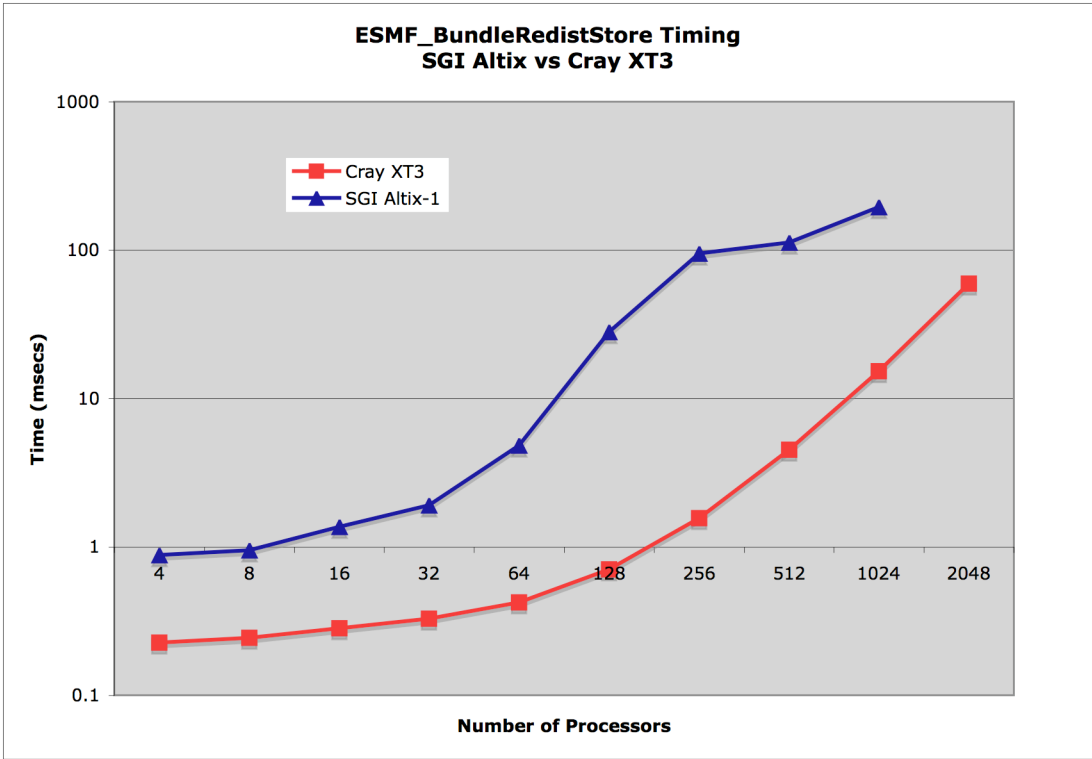


Figure 4 ESMF\_BundleRedistStore timing for 1024x512x16 grid

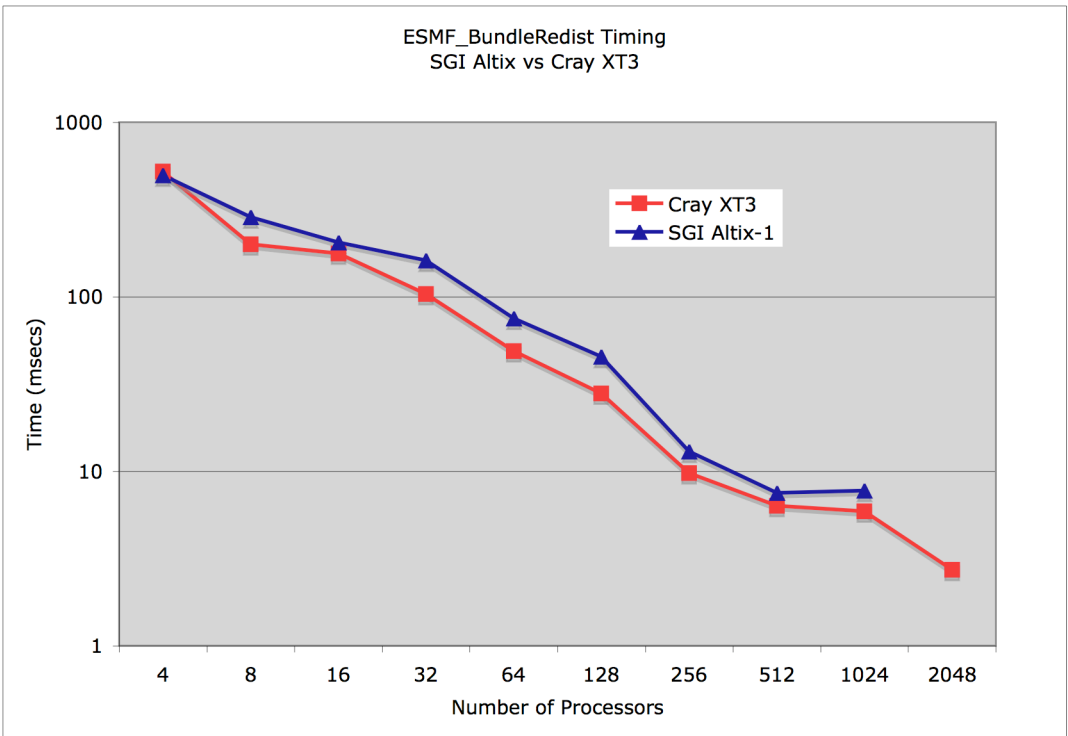


Figure 5 ESMF\_BundleRedist Timing for 1024x512x16 grid

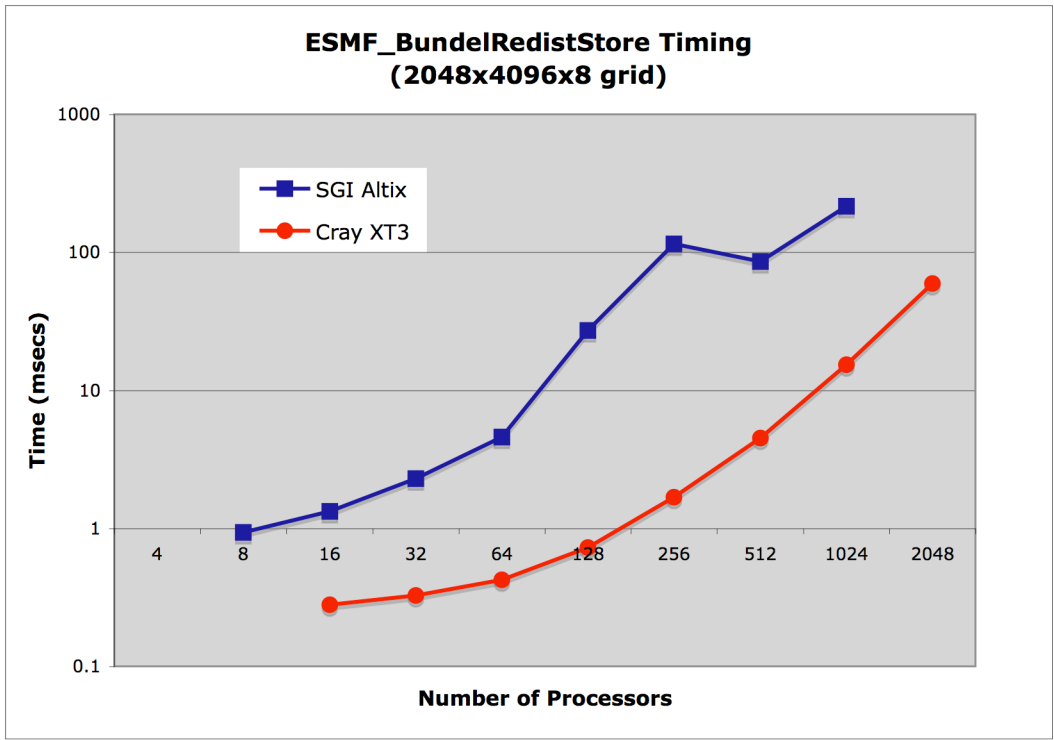


Figure 6 ESMF\_BundleRedistStore Timing for 2048x4096x8 grid

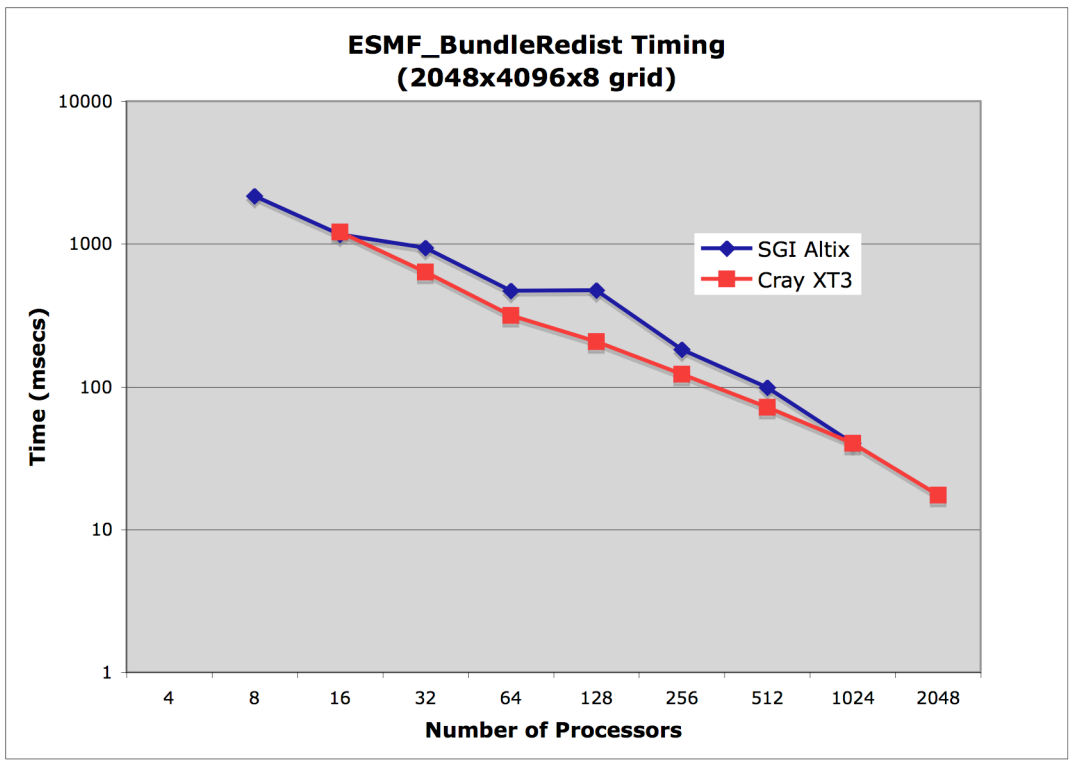


Figure 7 ESMF\_BundleRedist Timing for 2048x4096x8 grid

The Cray XT3/XT4 consistently outperforms the SGI Altix for both grid sizes and in both ESMF\_BundleRedistStore() and ESMF\_BundleRedist(). However the timing curves on both machines seem to agree with each other except for a discontinuity from 256 processors to 512 processors on the SGI Altix, Columbia. Note that, in Columbia, we ran the benchmark program on a 512-processor cluster for up to 256 processors and on a 2048-PE system for 512 and 1024 processors. The processors in the 2048-PE system is a little bit faster than the ones in a regular cluster (1.6GHz vs 1.5 GHz) and a processor's level 3 cache in the 2048 system is 50% more than the cache of the processor in the 512 clusters (9Mbytes vs 6 Mbytes). That explains why the performance of the 512 processor configuration is better than the performance of the 256 processors.

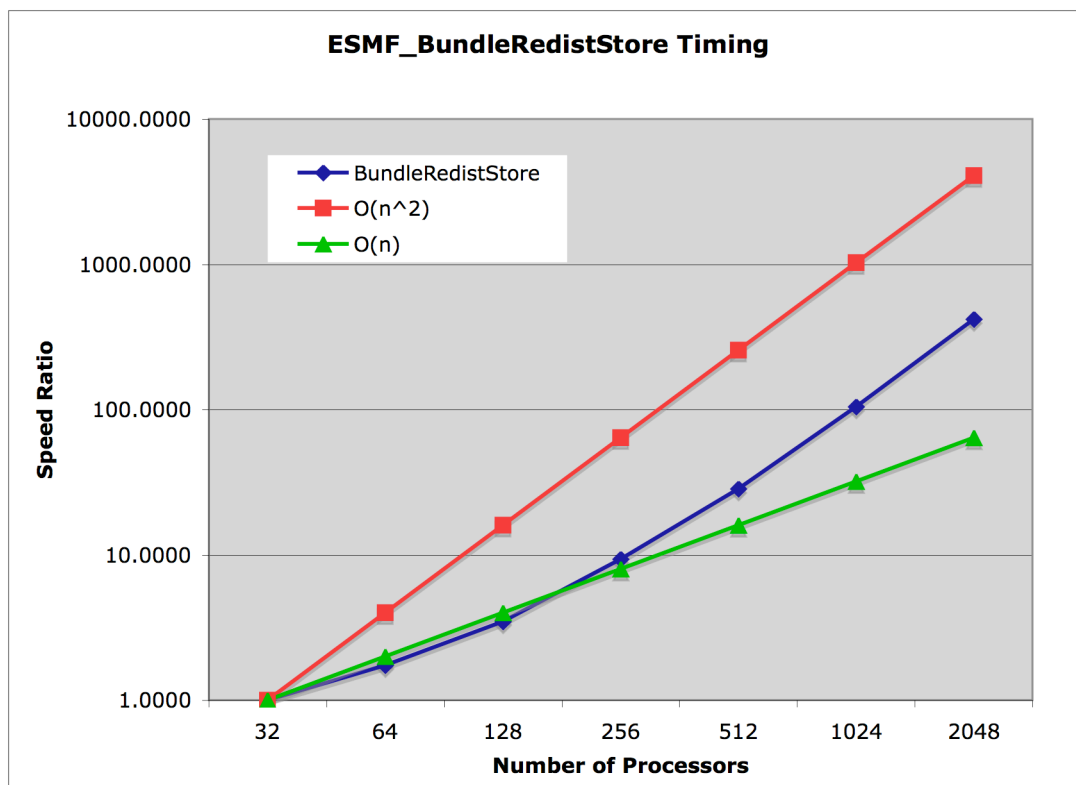
### Scalability

The ESMF\_BundleRedistStore() timing is independent of the size of the grid. It is a function of the number of processors used. On XT3/XT4, ESMF\_BundleRedistStore() time increases linearly with number of processors when  $n$  is smaller than or equal to 128, and it demonstrates a  $O(n^2)$  timing increase when  $n$  is greater than 128 processors. Further analyzing the ESMF\_BundleRedistStore() timing, we found the following two functions, ESMF\_RouteCreate() and ESMF\_RoutePrecomputeRedist(), take more than 90% of the total time. In Table 2, the timing for these two functions from 32 processors to 2048 processors is listed. The last column of the table shows the ratio of total time for  $n$  processors vs the total time of 32 processors for  $n=32$  to 2048. In Figure 8, we plot the ratio versus the number of processors together with two reference curves - an  $O(n)$  curve and an  $O(n^2)$  curve. Both ESMF\_RouteCreate() and ESMF\_RoutePrecomputeRedist() code contains loops over all the processors, therefore, theoretically they are  $O(n)$  algorithms. It is not clear why a  $O(n^2)$  performance degradation was measured when  $n > 128$ .

ESMF\_BundleRedist() time depends on the amount of data transferred by each processor. Therefore, the time to redistribute the large grid is 6 to 12 times slower than the time to redistribute the smaller grid (The large grid has 16 times more grid points than the small grid). Given a fixed size grid, double the number of processors will reduce the number of local grid points by half. Thus the total message size communicated in each processor during redistribution is also reduced linearly, so is the amount of data to be packed or unpacked. That is why we see a linear speedup in the ESMF\_BundleRedist() timing with increasing number of processors.

**Table 2 The critical function timing in ESMF BundleRedistStore()**

# Processors	ESMC_RoutePrecompute Redist (msecs)	ESMF_RouteCreate (msecs)	Total (msecs)	Ratio T(n)/T(128)
32	0.117064	0.01502	0.132084	1.0000
64	0.194073	0.036	0.230073	1.7419
128	0.358105	0.1001	0.458205	3.4690
256	0.872135	0.36	1.232135	9.3284
512	2.5389	1.2409	3.7798	28.6166
1024	8.876	4.986	13.862	104.9484
2048	32.5729	22.762	55.3349	418.9372



**Figure 8 Scalability Study of ESMF\_BundleRedistStore**

### Conclusion

The important findings from this benchmark study can be summarized as follows:

- The default routing option for ESMF\_BundleRedistStore() yields the best performance. Do not change it if you don't have clear understanding what each different route option means and how it performs in the target architecture.
- We observed an  $O(n)$  performance when  $n \leq 128$  and an  $O(n^2)$  performance when  $n > 128$  in ESMF\_BundleRedistStore(). The two main functions called in ESMF\_BundleRedistStore(), namely, ESMF\_RouteCreate() and

ESMF\_RoutePrecomputeRedist() have the same performance pattern. Since both functions have a complexity of  $O(n)$ , it is not clear what causes the  $O(n^2)$  performance degradation with large number of processors.

- ESMF\_BundleRedist() scales linearly with number of processors used.