

ESMF Memory Scalability Benchmark

Peggy Li
July 1, 2008

Objective

In the ESMF Array Sparse Matrix Multiplication Benchmark report, we observed super linear increase of memory usage in the ESMF component APIs while using 512 or more processors. Figure 1 is the chart taken from the ASMM Benchmark. The memory usage was measured on the Cray XT3 at Oak Ridge National Lab using heap_info() system call.

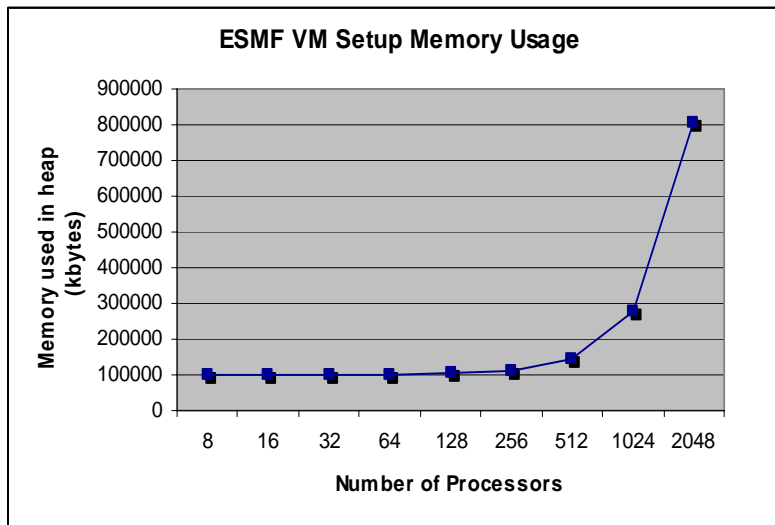


Figure 1 ESMF VM/Component Memory Usage before Optimization

The memory inefficiency was identified as a problem in the implementation of the ESMF VM module. The VM implementation was optimized in the public release of ESMF 3.1.0r and the memory usage before and after the optimization was benchmarked using the ESMF Superstructure benchmark program used in the “Component Overhead for Large Processor Counts” report dated 11/3/2006.

The benchmark was conducted on the Cray XT4 system, jaguar, at Oak Ridge National Lab. Jaguar has recently been upgraded to quad-core 2.1 GHz AMD Opteron processors and it has a total of 7,832 compute nodes. We ran the benchmark using 4 to 16384 processors. Two snapshots of ESMF were used for the benchmark: ESMF 3.1.0 beta snapshot #51 (bs51) (without memory optimization) and ESMF 3.1.0 beta snapshot #53 (bs53) (with optimization). Jaguar’s compute nodes are running the Compute Node Linux (CLN) Operating System and heap_info() is no longer available on CLN. Mallinfo() was thus used to measure the dynamic memory allocation in the benchmark.

Benchmark Program

The benchmark program is adapted from the program ESMF_SuperPerfTest.F90 in the esmfcontrib repository. The test program tests basic ESMF superstructure functions, such as ESMF_Initialize(), ESMF_Finalize(), ESMF_GridCompCreate(), ESMF_CompSetServices(), ESMF_GridCompInitialize(), ESMF_GridCompRun(), and ESMF_GridCompFinalize(). This benchmark program also includes a simple grid component user_model.F90. It defines an init subroutine, a run subroutine and a finalize subroutine. In order to measure the ESMF overhead only, the user defined routines are all empty routines. In the program, only one ESMF_GridComp is created.

The Results

Table 1 shows the total dynamic memory usage for bs51 and bs53 and the ratio of the memory usage before and after the optimization. The bs51 benchmark core dumps on 8192 and 16384 processors because it exceeds the physical memory on the compute node. Before the optimization, there is quadruple memory increase while using 256 or more processors. The optimized code has less than linear memory increase all the way to 16384 processors. More than two order of magnitude improvement is observed on 2048 processors and above.

# Procs	Total Memory (Kbytes)		
	bs51	bs53	bs51/bs53
4	333	332	1.00
8	338	335	1.01
16	351	340	1.03
32	396	349	1.13
64	557	368	1.51
128	1168	406	2.88
256	3542	482	7.35
512	12897	633	20.37
1024	50039	935	53.50
2048	197811	1299	152.24
4096	788508	2268	347.67
8192		3053	
16384		4238	

Table 1 Memory Usage before and after optimization

Figure 1 shows the memory usage comparison of the two ESMF snapshots. The memory usage curve for VM is the dynamic memory allocated by ESMF_Initialize(). The memory curve for component is the dynamic memory allocated by creating and initializing a Gridded Component. Both functions create an ESMF_VM object, thus the memory usage is similar.

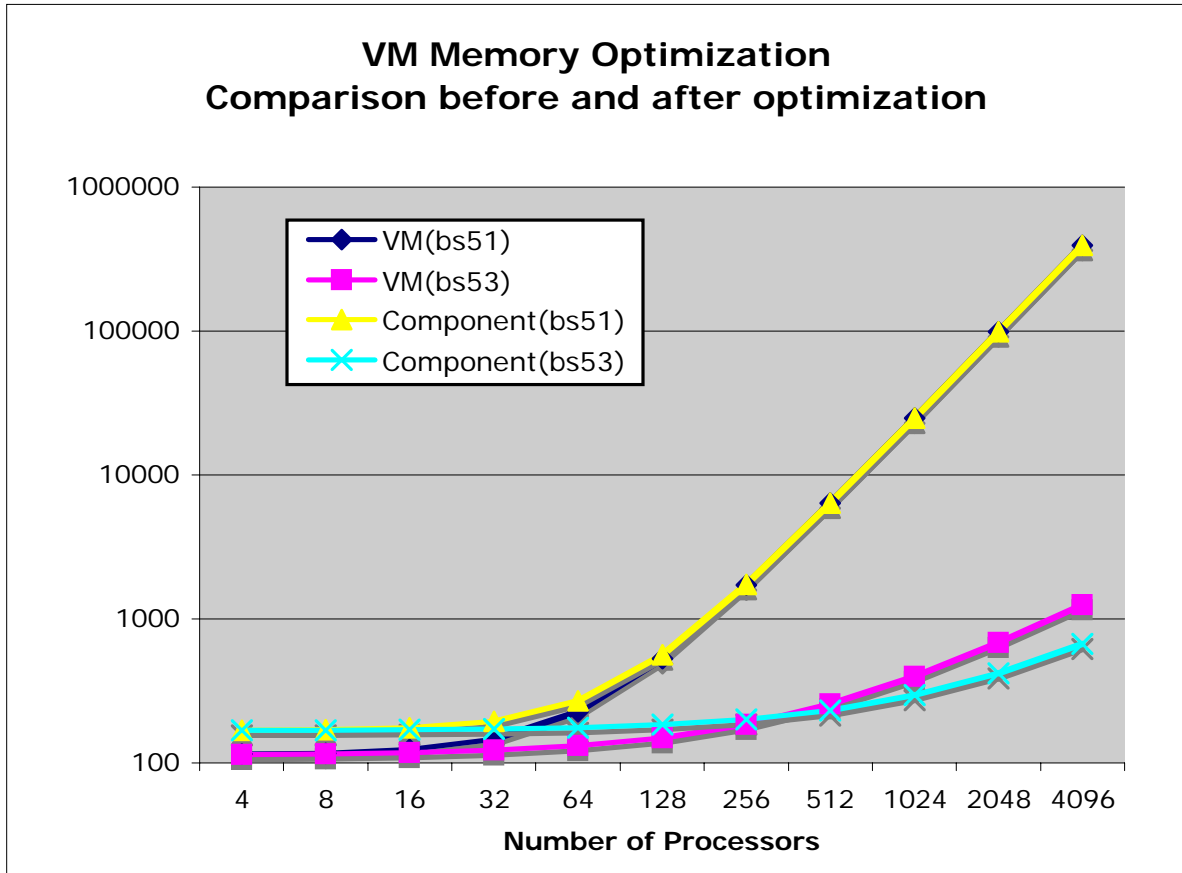


Figure 2 Memory Usage Comparison before and after Optimization

Conclusion

Based on the benchmark results, we conclude that the ESMF 3.1.0r (which includes the memory optimization) VM and component interface introduces little memory overhead and is scalable to tens of thousands of processors.