Open access • Proceedings Article • DOI:10.1109/HPCC.2010.55

Performance Analysis of Scientific and Engineering Applications Using MPInside and TAU — Source link

Subhash Saini, Piyush Mehrotra, Kenichi Taylor, Sameer Shende ...+1 more authors

Institutions: Ames Research Center

Published on: 01 Sep 2010 - High Performance Computing and Communications

Performance Analysis of Scientific and Engineering Applications Using MPInside and TAU

Subhash Saini[1], Piyush Mehrotra[1], Kenichi Taylor[2], Sameer Shende[3], Rupak Biswas[1]

[1] NASA Advanced Supercomputing
NASA Ames Research Center
Moffett Field, CA 94035 USA
{subhash.saini, piyush.mehrotra, rupak.biswas}@nasa.gov

[2] Silicon Graphics International (SGI)
46600 Landing Pkwy
Fremont, CA 94538 USA
kenichi@sgi.com

[3] ParaTools, Inc.
2836 Kincaid Street
Eugene, OR 97405 USA
sameer@paratools.com

ABSTRACT

In this paper, we present performance analysis of two NASA applications using performance tools like Tuning and Analysis Utilities (TAU) and SGI MPInside. MITgcmUV and OVERFLOW are two production-quality applications used extensively by scientists and engineers at NASA. MITgcmUV is a global ocean simulation model, developed by the Estimating the Circulation and Climate of the Ocean (ECCO) Consortium, for solving the fluid equations of motion using the hydrostatic approximation. OVERFLOW is a general-purpose Navier-Stokes solver for computational fluid dynamics (CFD) problems. Using these tools, we analyze the MPI functions (*MPI_Sendrecv, MPI_Bcast, MPI_Reduce, MPI_Allreduce, MPI_Barrier, etc.*) with respect to message size of each rank, time consumed by each function, and how ranks communicate. MPI communication is further analyzed by studying the performance of MPI functions used in these two applications as a function of message size and number of cores. Finally, we present the compute time, communication time, and I/O time as a function of the number of cores.

1. INTRODUCTION

Developing or porting codes on new computing architectures to achieve good performance is a challenging and daunting task for application scientists and engineers. Performance of most of the real-world applications is less than 10% of the peak performance on these computing systems. Low performance is due to a number of challenges facing the high-performance scientific community, including increasing levels of parallelism (threads, multi- and many-cores, nodes), deeper and more complex memory hierarchies (register, multiple levels of cache, on node NUMA memory, disk, network), and hybrid hardware (processors and GPGPUs). In many cases, factors such as runtime variation due to system noise, traditional computer benchmarking is not sufficient to understand the performance of large-scale applications. In such cases, simple inspection of the profile (the timing breakdown) is not adequate to analyze performance of particularly MPI applications. One needs to know what is happening "inside" both the application and the MPI library and along with the interaction of the two.

The present study uses two performance tools (SGI's MPInside and TAU from University of Oregon) to profile two production-quality applications (OVERFLOW-2 and MITgcmUV, hereafter OVERFLOW-2 will be referred as OVERFLOW). This study also uses the low-level MPI function benchmarks to measure their performance as a function of message size. The study was carried out on an SGI Altix ICE 8200EX cluster, Pleiades, located at NASA Ames Research Center. Pleiades consists of two sub-clusters: one part based uses the Xeon 5472 Harpertown processor [1–2] (hereafter called "Pleiades-HT"), and the second uses Xeon 5570 Nehalem processor, the first server implementation of a new 64-bit micro-architecture (henceforth called "Pleiades-NH") [3-4]. All the nodes employ the Linux operating system and SGI MPT library and are connected in a hypercube topology using InfiniBand [5-6].

In this paper we have conducted the performance profiling of OVERFLOW and MITgcmUV using the two performance tools, MPInside and TAU, on Pleiades-HT and Pleiades-NH. We have also evaluated and compared the performance of MPI functions as a function of message size on Pleiades-HT and Pleiades-NH.

The remainder of this paper is organized as follows: Section 2 describes the two performance tools, SGI's MPInside and TAU from University of Oregon, used in the study. Section 3 gives the overview of the applications and MPI function benchmarks. Section 4 presents and analyzes results from running these benchmarks and applications on the two clusters. Section 5 contains a summary and conclusions of the study.

2. Performance Tools Used

Based on an initial survey and looking into pros and cons of each performance tool, we decided to use two tools, MPInside and TAU, to conduct in-depth performance analysis of two real-world applications used extensively by scientists and engineers at NASA [7-22]. MPInside is a profiling and diagnostic tool developed by SGI to analyze and predict the performance of MPI applications [15].

Tuning and Analysis Utilities (TAU) developed by University of Oregon, and supported by ParaTools, Inc., is a portable profiling and tracing toolkit for performance analysis of parallel programs [14].

3. Applications and Benchmarks

3.1 Science and Engineering Applications

For this study, we used two production applications, taken from NASA's workload. OVERFLOW, developed at NASA's Langley Research Center, is a general-purpose Navier-Stokes solver for CFD problems [23]. MITgcmUV, developed by the Estimating the Circulation and Climate of the Ocean (ECCO) Consortium, is a global ocean simulation model for solving the fluid equations of motion using the hydrostatic approximation [24].

3.2 Intel MPI Benchmarks (IMB)

The performance of real-world applications that use MPI as the programming model depends significantly on the MPI library and the performance of various point-to-point and collective message exchange operations supported by the MPI implementations. Intel MPI Benchmarks (IMB), (formerly, the Pallas MPI Benchmarks) is a commonly used benchmark suite to evaluate and compare the performance of different MPI implementations [25].

The MPI standard defines several collective operations, which can be broadly classified into three major categories based on the message exchange pattern: *OnetoAll, AlltoOne*, and *Alltoall*. We have evaluated the performance of *MPI_Bcast, MPI_Reduce, MPI_Alltoall*, *and MPI_Allreduce* collective operations on both clusters.

4. Results

In this section, we present the results of our study.

4.1 Scientific and Engineering Applications

4.1.1 MITgcmUV

Figure 1 shows the sustained performance of MITgcmUV using TAU and MPInside.

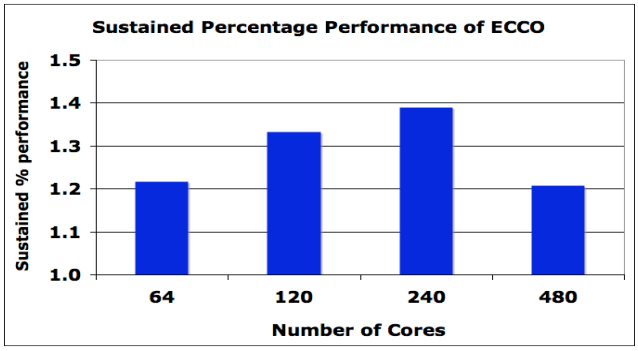

Figure 1: Sustained performance of MITgcmUV on Pleiades-HT

Sustained performance of MITgcmUV is about 1.2–1.4% of the peak, which is relatively low, as most of the applications have sustained performance around 3–8% of peak. We have not looked into the cause of this low performance from processor and memory subsystem perspective here but have only investigated the role of various MPI functions for the application.

In Figure 2, we show the percentage of time for total, compute, communication, and I/O times on Pleiades-HT and Pleiades-NH. As expected, percentage of compute time decreases and communication time increases for increasing numbers of cores for both systems. Percentage contribution of I/O time increases for large number of cores. On 64 cores of Pleiades-NH compute is 93%, communication 3.5%, I/O 3.5%. Corresponding numbers for 480 cores are: compute 59.1%, communication 23.4%, and I/O 17.5%.

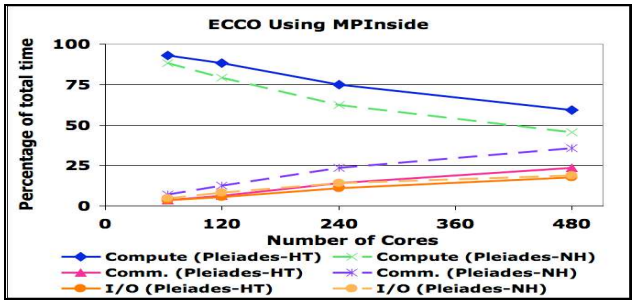

Figure 2: Time percentage of MITgcmUV using MPInside on two systems.

Figure 3 shows the read, write, and (read+write) times for MITgcmUV on two systems. Read time is almost the same on both systems; however, write time on Pleiades-HT is much higher than on Pleiades-NH—it writes 8 GB of data. This is due to the fact that Pleiades-NH has three times more memory than Pleiades-HT so one is measuring writes to buffer cache in memory. On the other hand, on Pleiades-HT one is measuring "write time" to disk because there is not enough memory to hold all 8 GBs of output data.

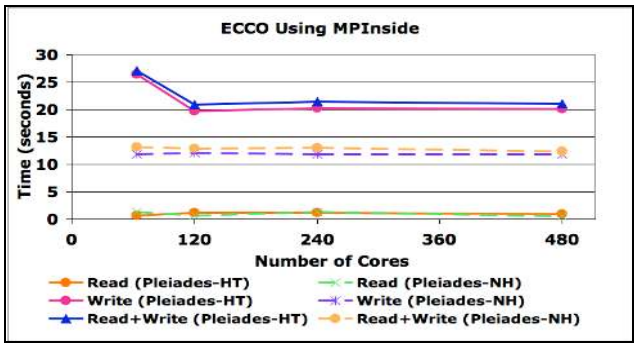

Figure 3: I/O times for MITgcmUV using MPInside on two systems.

In Figure 4, we plot the write bandwidth on the two systems. Write bandwidth on Pleiades-NH is about 55% higher than on Pleiades-HT. As mentioned in the previous paragraph, this is due to the fact that there is three times more memory per core in Pleiades-NH than Pleiades-HT—write is done using the memory buffer as opposed to disk write in Pleiades-HT.

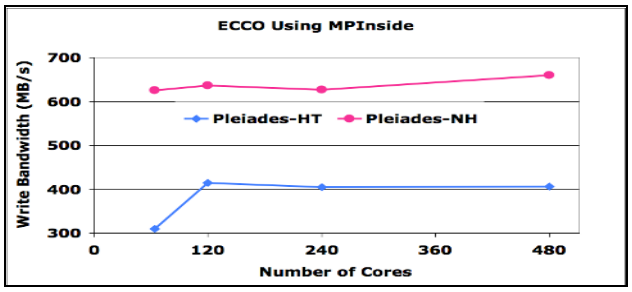

Figure 4: Write bandwidth of MITgcmUV on two systems.

In Figure 5, we plot the percentage of time spent in each of the MPI functions in MITgcmUV. Percentage of communication time spent is 60, 30, and 5% in *MPI_Recv, MPI_Allreduce*, and *MPI_Waitall*, respectively. Only 5% of the time is spent in *MPI_Send, MPI_Isend, MPI_Bcast*, and *MPI_Barrier*.

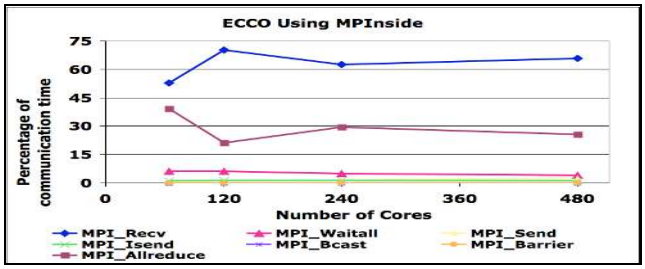

Figure 5: Percentage of time spent in MPI functions for MITgcmUV.

In Figure 6, we plot the minimum, average, and maximum message size of *MPI_Recv* in MITgcmUV using MPInside. The average message size varies from 3–9 KB.

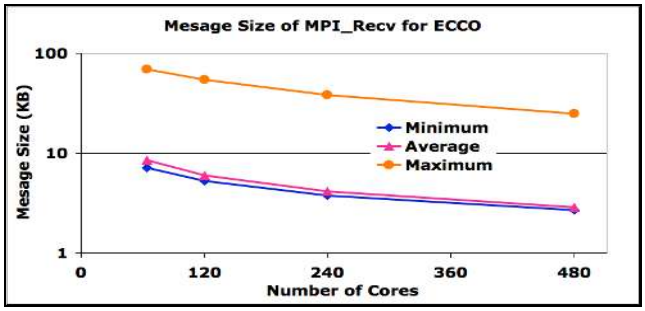

Figure 6: Message size of *MPI_Recv* in MITgcmUV using MPInside.

With both the tools the message size in *MPI_Allreduce* is 8 bytes for cores ranging from 60 to 480. Since data size is only 8 bytes, *MPI_Allreduce* is network latency-bound in MITgcmUV. A message size of 225 KB is broadcast to all cores. Message sizes for all MPI functions in MITgcmUV including *MPI_Recv*, *MPI_Allreduce*, and *MPI_Bcast* were the same, as measured by TAU and MPInside.

4.1.2 OVERFLOW

In this subsection, we present results for OVERFLOW using the performance tools MPInside and TAU. Only the results obtained using MPInside are shown, as they are same as those obtained by using TAU.

Figure 7 shows the sustained performance of OVERFLOW. Sustained performance is about 2.5% of peak. Performance of OVERFLOW is slightly better than MITgcmUV. We notice that even for 16 cores (2 nodes), performance is low. We did not investigate the cause of this low sustained performance but believe it is related to processor and memory subsystem.

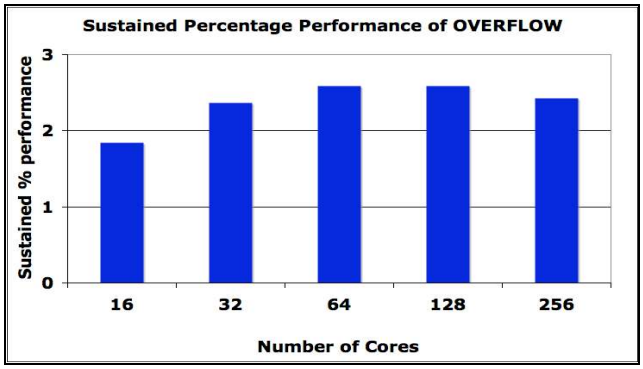

Figure 7: Sustained performance of OVERFOW.

Figure 8 shows the percentage of computation, communication, I/O, and total time on both systems. On both systems, percentage of computation time decreases as the number of cores increase from 32 to 128 and then increases for 256 cores. In addition, percentage of communication time increases as the number of cores increases from 32 to 128, and then decreases for 256 cores. For 256 cores on Pleiades-HT: computation 62%, communication 25%, and I/O 13%; Pleiades-NH: computation 52%, communication 33%, and I/O 15 %.

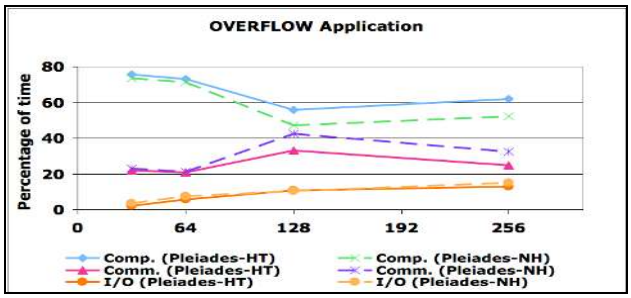

Figure 8: Percentage of computation, communication, I/O, and total time in OVERFLOW using MPInside.

Figure 9 shows the I/O time for OVERFLOW on the two

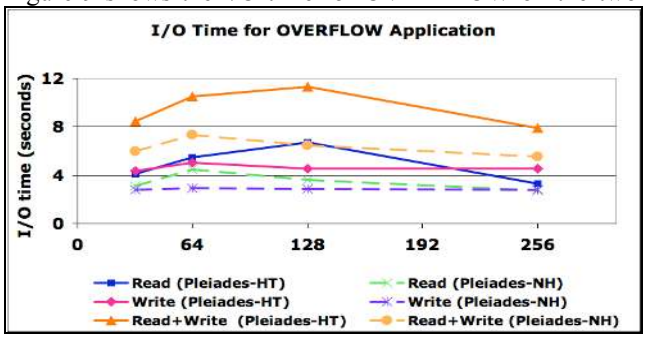

Figure 9: I/O time in OVERFLOW for Pleiades-HT and Pleiades-NH.

systems. I/O times are better on Pleiades-NH than on Pleiades-HT. Performance of (read+write) is better on Pleiades-NH than Pleiades-HT by a factor of 1.4 for all core counts except at 128 cores where it is a factor of 1.7.

Figure 10 shows the read and write bandwidth in OVERFLOW for the two systems. Size of input data file read is 1.6 GB, and size of the solution file written is 2 GB. Both read and write bandwidths are higher on Pleiades-NH than on Pleiades-HT. The reason for this is that memory per node is three times higher on Pleiades-NH than on Pleiades-HT (24 vs. 8 GB), so size of memory buffers is higher in the former. Performance of the write bandwidth in OVERFLOW is almost the same as in MITgcmUV, although data written is four times larger in MITgcmUV (2 vs. 8 GB).

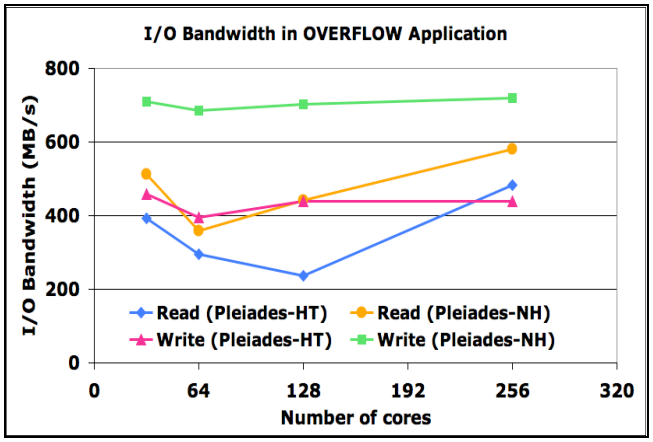

Figure 10: Read and write bandwidth in OVERFLOW for two systems.

In Figure 11, we show times for the top five MPI functions. Most of the time is consumed by the two functions *MPI_Waitall* and *MPI_Gatherv*, followed by *MPI_Recv* and *MPI_Send* and the lowest time by *MPI_Bacst*. For 128–256 cores, time for *MPI_Waitall* and *MPI_GatherV* decreases, whereas time for *MPI_Recv*, *MPI_Send*, and *MPI_Bcast* remains almost constant.

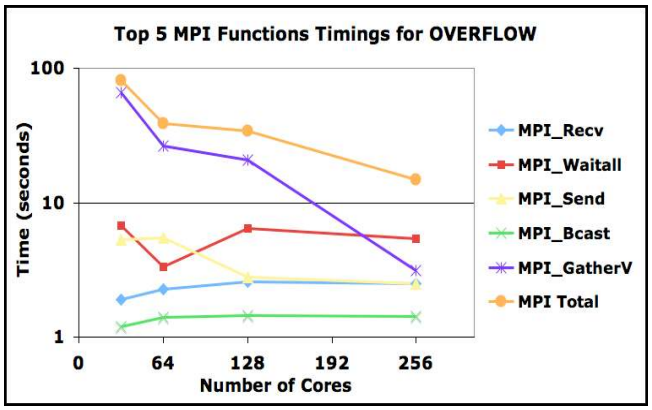

Figure 11: Timings for the top 5 MPI functions in OVERFLOW.

Figure 12 shows percentage time for the top 5 MPI functions. Percentage of time taken by *MPI_Recv* and *MPI_Send* increases as the number of cores increases. Up to 64 cores, percentage of time taken by *MPI_Send* is more than *MPI_Recv* and then it becomes the same for 128 and 256 cores. For higher numbers of cores, percentage time consumed by all MPI functions increases, except for *MPI_GatherV*. At 256 cores, percentage of time consumed by *MPI_Waitall* is the highest. The function *MPI_Waitall* waits for all communications to complete. At 256 cores, percentage of time contributions are *MPI_Waitall* 36%, *MPI_GatherV* 21%, *MPI_Recv* 17%, *MPI_Send* 17%, and *MPI_Bcast* 9%.

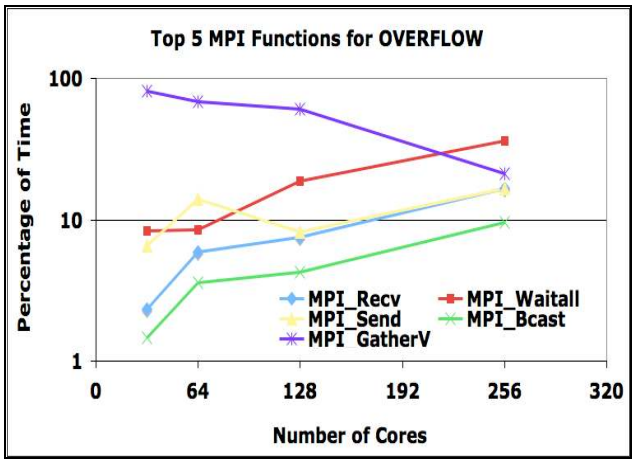

Figure 12: Percentage time for the top 5 MPI functions in OVERFLOW using MPInside.

Figure 13 shows the minimum, average, and maximum message size of *MPI_Send* in the OVERFLOW application. Message size decreases as the number of cores increases. The average message size for *MPI_Send* is 348, 129, 80, and 54 KB for 16, 128, 256, and 512 cores, respectively.

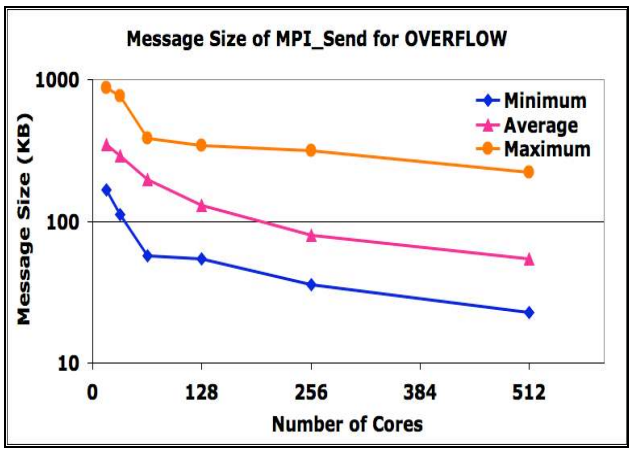

Figure 13: Message size for the *MPI_Send* function in OVERFLOW using MPInside.

Figure 14 shows the minimum, average, and maximum message size for *MPI_Recv*. For all three cases, size first increases up to 32/64 cores, and then decreases up to 512 cores. Average message size for *MPI_Recv* is 53, 104, 144, and 219 KB for 16, 128, 256, and 512 cores respectively.

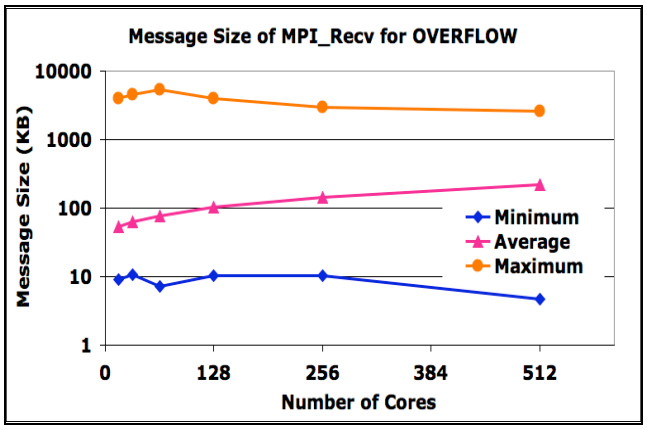

Figure 14: Message size for the *MPI_Recv* function in OVERFLOW using MPInside.

Figure 15 shows the message size for *MPI_Bcast*. Message size for *MPI_Bcast* is 1.29 MB in the OVERFLOW application from 16 to 512 cores.

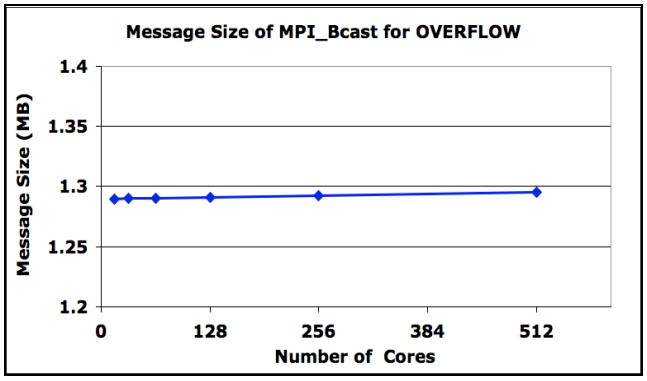

Figure 15: Message size for the *MPI_Bcast* function in OVERFOW using MPInside.

Figure 16 shows the minimum, average, and maximum message size for *MPI_Gatherv*. Average size of the message gathered by *MPI_Gatherv* is 270 bytes for 16 to 512 cores. Since the size of the message is very small, performance of *MPI_Gatherv* depends on network latency and not on network bandwidth.

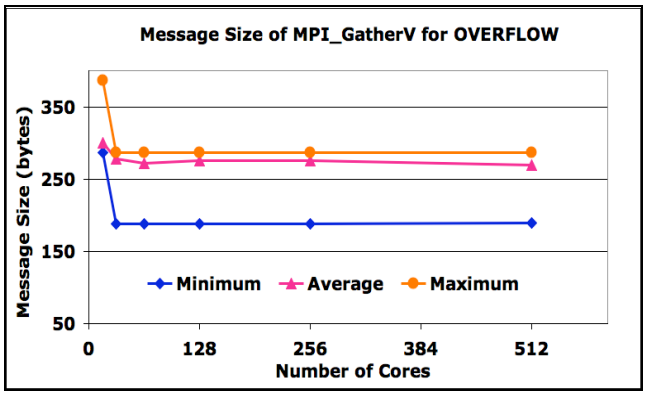

Figure 16: Message size for the *MPI_Gatherv* function in OVERFLOW using MPInside.

4.2 Intel MPI Benchmarks (IMB)

In this section, we describe the performance of various MPI functions relevant to the two applications (MITgcmUV and OVERFLOW) used in this paper.

4.2.1 MPI_Sendrecv & MPI_Exchange

In Figure 17, we plot the performance of the *MPI_Sendrecv* and *MPI_Exchange* benchmarks for small messages on both systems. This plot provides insights into the relationship between the message exchange pattern, point-to-point message exchange algorithms, and overall performance. On both systems, performance of the *MPI_Sendrecv* benchmark is better than *MPI_Exchange*. In the *MPI_Exchange* benchmark, each process exchanges messages with both its left and right neighbors simultaneously, whereas in the *MPI_Sendrecv* benchmark, each process receives from its left neighbor and sends to its right neighbor at any instant. Since the *MPI_Sendrecv* benchmark involves a lesser volume of messages exchanged in comparison with *MPI_Exchange*, it is natural to expect better throughput. We see a change in slope for both benchmarks on the two systems around a message size of 1 KB, which is due to a change of algorithm.

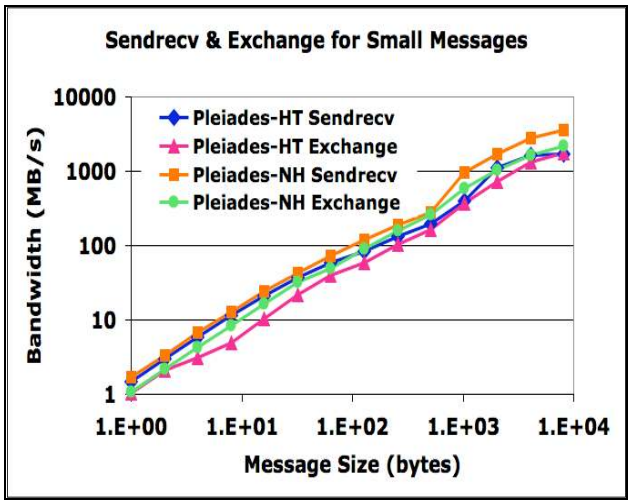

Figure 17: Performance of the *MPI_Sendrecv* and *MPI_Exchange* functions on two systems for small messages.

In Figure 18, we plot the performance of the *MPI_Sendrecv* and *MPI_Exchange* benchmarks for large messages on both systems. We see a peak bandwidth with a 16 KB message (3.6 GB/s for Pleiades-NH vs. 2.6 GB/s for Pleiades-HT), which falls drastically for larger messages and stabilizes at 2.3 GB/s for Pleiades-NH and 2.9 GB/s for Pleiades-HT. We believe this could be due to cache effects as large message intra-node exchanges usually involve making a copy from the user buffer to the shared-memory buffers. As size of the data in the user buffer grows, we may not be able to fit it in the cache, leading to cache misses.

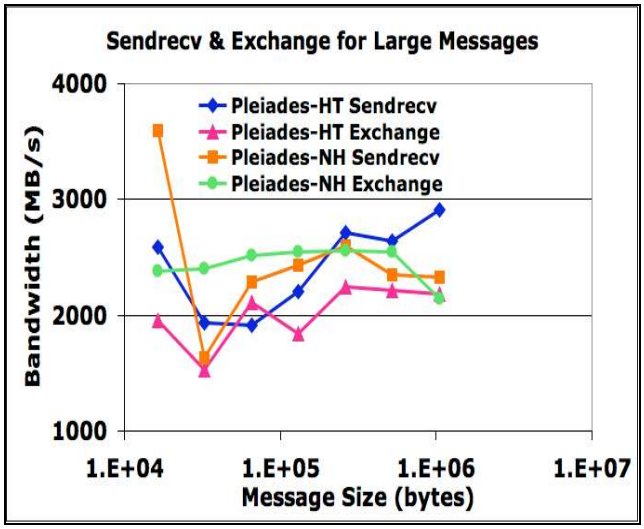

Figure 18: Performance of the *MPI_Sendrecv* and *MPI_Exchange* functions on two systems for large messages.

Figure 19 shows the bandwidth of the *MPI_Sendrecv* benchmark for a message size of 262 KB, which is the average size used in MITgcmUV for the cores ranging from 2 to 512. Bandwidth within a node (8 cores) is higher on Pleiades-NH than on Pleiades-HT as the former uses faster intra-node communication via QPI. Beyond 8 cores (a node), the bandwidth on both systems is almost same except at 512 cores where Pleiades-NH has higher bandwidth.

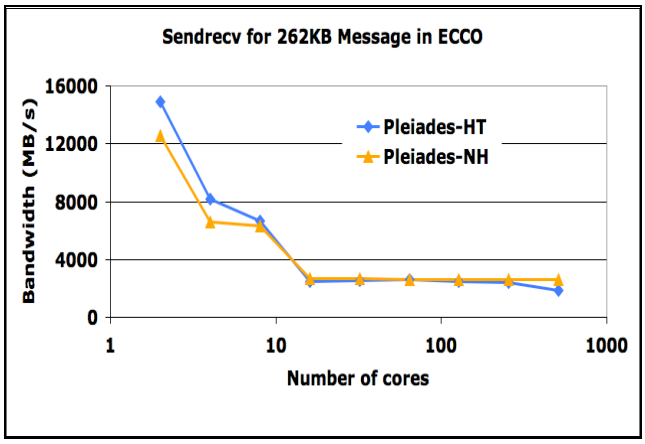

Figure 19: Performance of the *MPI_Sendrecv* function on two systems for a message size of 262 KB.

4.2.2 MPI_Bcast

Figure 20 shows the performance of *MPI_Bcast* for small messages on the two systems. Up to a 1 KB message size, performance on both systems is almost the same. However, beyond that we notice there is a drastic change of slope on both systems due to transition of algorithms used in its implementation. In addition, performance is better on Pleiades-NH than on Pleiades-HT.

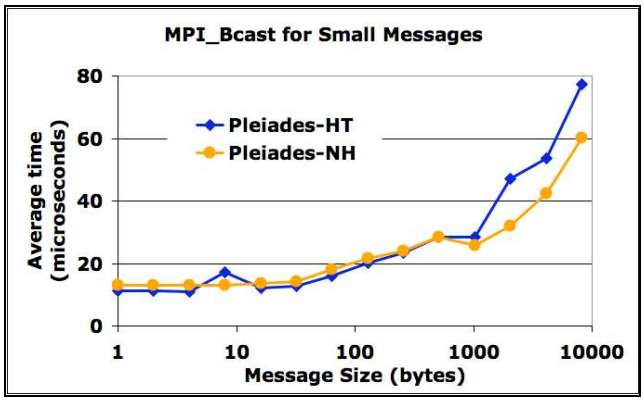

Figure 20: Performance of *MPI_Bcast* on two systems for small messages.

Figure 21 shows the performance of *MPI_Bcast* for large messages on the two systems. Performance difference between the two systems is small for 16 to 64 KB, and then the performance gap increases as the message size increases. Timings are (a) 64 KB: 289 vs. 481 µs, and (b) 1 MB: 5,398 vs. 8,038 µs on two systems.

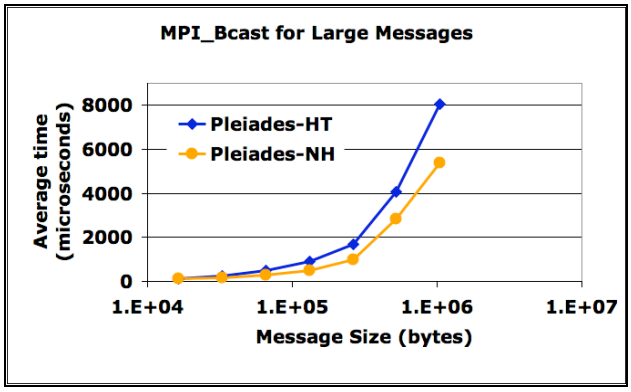

Figure 21: Performance of *MPI_Bcast* on two systems for large messages.

Figure 22 shows the performance of *MPI_Bcast* for a 1 MB message size used in OVERFLOW. We see that performance on Pleiades-NH is higher than Pleiades-HT for both intra- and inter-node communication.

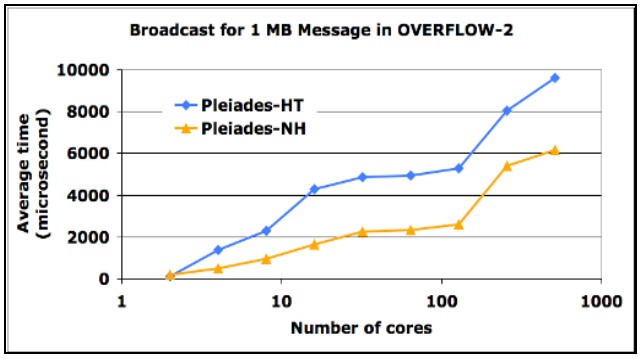

Figure 22: Performance of *MPI_Bcast* on two systems for a 1 MB message.

4.2.3 *MPI_Allreduce*

In Figure 23, we plot average time for the *MPI_Allreduce* benchmark for small messages for both systems. Up to 64 bytes, performance is higher on Pleiades-HT and then from 128 bytes to 1 KB, performance is the same. From 2 KB onwards, the performance gap continues to widen and at 8 KB, it is 40% higher (151 vs. 211 µs).

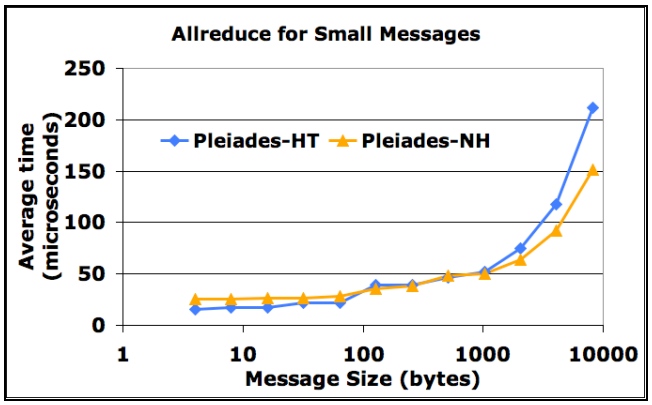

Figure 23: Performance of *MPI_Allreduce* on two systems for small messages.

In Figure 24, we plot the average time for the *MPI_Allreduce* benchmark for large messages for both systems. Throughout all cores, performance on Pleiades-NH is higher than on Pleiades-HT, and the performance gap increases as the number of cores increases. At 16 KB, times are 261 and 392 µs, and at 1 MB, they are 7,958 and 10,897 µs for Pleiades-NH and Pleiades-HT, respectively.

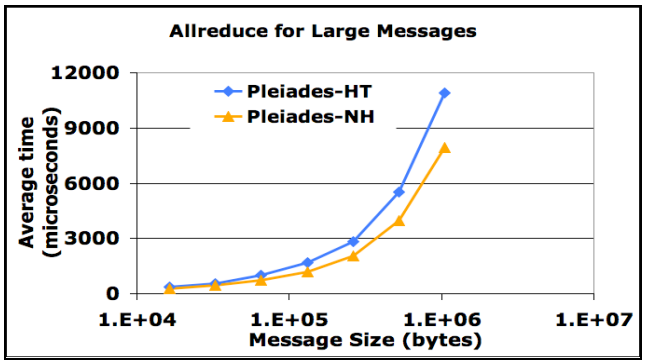

Figure 24: Performance of *MPI_Allreduce* on two systems for small messages.

Figure 25 shows the performance of *MPI_Allreduce* on two systems for a message size of 8 bytes used in MITgcmUV. On both systems up to 64 cores, performance of *MPI_Allreduce* is same and degrades slowly as the number of cores increase. It may be recalled that in MITgcmUV the average size of message broadcast is 8 bytes. Since the message size is very small the performance of *MPI_Allreduce* in MITgcmUV depends on the network latency of the system. Network latency of both systems increases with increasing number of cores especially beyond 128 cores (1 IRU) and therefore degrades rapidly.

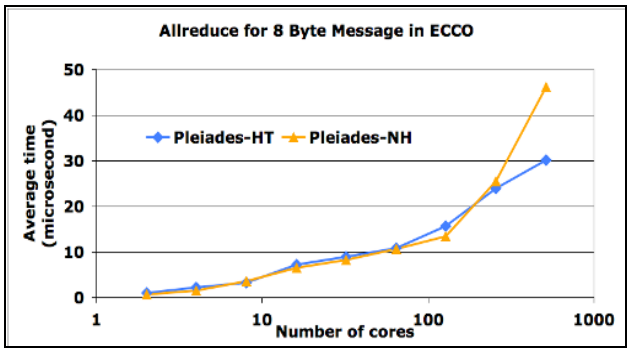

Figure 25: Performance of *MPI_Allreduce* on two systems for an 8-byte message.

4.2.4 *MPI_Gatherv*

Figure 26 shows the performance of *MPI_Gatherv* on two systems for small messages. Up to a message size 4 KB, performance of Pleiades-HT is much better than Pleiades-NH, however for 8 KB message performance of Pleiades-NH is better. The reason for this is the change in algorithm for the implementation of *MPI_Gatherv* on MPT.

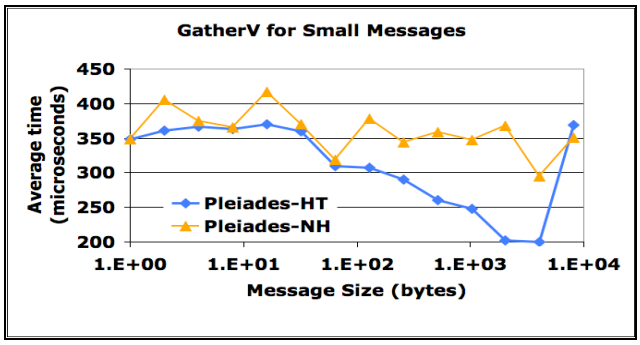

Figure 26: Performance of *MPI_Gatherv* on two systems for small messages.

Figure 27 shows the performance of *MPI_Gatherv* on two systems for large messages. Up to 64 KB, performance of Pleiades-HT is better than Pleiades-NH, however for 128 KB to 1 MB performance of Pleiades-NH is better.

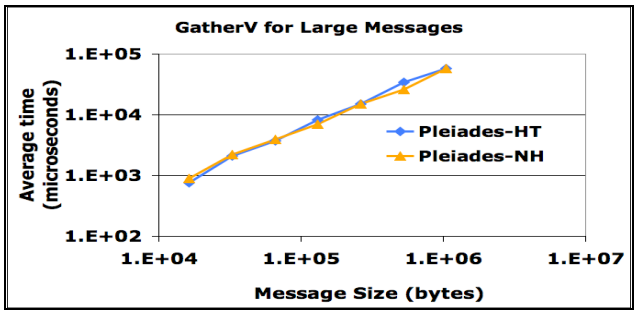

Figure 27: Performance of *MPI_Gatherv* on two systems for large messages.

Figure 28 shows the performance of *MPI_Gatherv* on two systems for a message 262 KB. It is worth mentioning that

average message in *MPI_Gatherv* is 270 KB. Within a node (8 cores), performance of Pleiades-NH is better than Pleiades-HT—the former's inter-socket communication is faster due to QPI. Performance of both systems is same for 16 to 64 cores. Beyond 64 cores, performance of Pleiades-NH is better than Pleiades-HT.

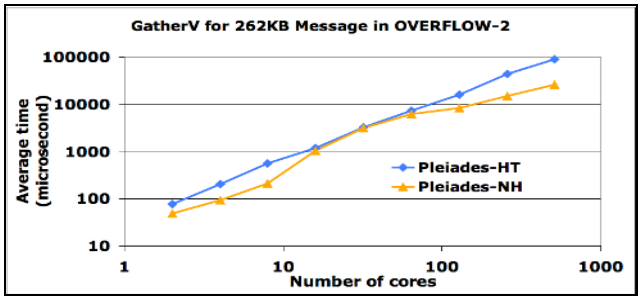

Figure 28: Performance of *MPI_Gatherv* on two systems for a 262 KB message.

5. Summary and Conclusions

In this paper, we study the performance of two NASA applications using two different analysis tools, TAU from University of Oregon and SGI's MPInside. We focus particularly on the communication times analyzing the performance of various MPI functions used in these applications. One of the most interesting results reached by our analysis is that relatively few functions in the MPI library are used in the MITgcmUV and OVERFLOW applications. The other conclusion is that write data (solution file) is relatively small, namely 2 GB and 8 GB for OVERFLOW and MITgcmUV, respectively, and is performed sequentially.

There was wide variation in message lengths—the shortest is 8-byte messages in *MPI_Allreduce* in MITgcmUV, and the largest message length is 1.3 MB for *MPI_Bcast* in OVERFLOW. Message length for *MPI_Gatherv* and *MPI_Recv* used in OVERFLOW is 270 bytes and 100 KB, respectively. Average message length for *MPI_Recv* and *MPI_Bcast* used in MITgcmUV is 6 KB (actually 3 to 9 KB) and 225 KB. Overall, the conclusion that can be drawn is that inter-core communication for hardware and software must be optimized for both short and long messages. This paper shows that a large percentage of messages, for these applications, are not extremely long.

We used two different tools for analyzing the performance of the MPI benchmarks and the two applications: SGI's MPInside and TAU from University of Oregon. TAU has more extensive, sophisticated features and a nice visual interface. However, it does have a steep learning curve and to use it effectively, it is helpful to have support and training. On the other hand, MPInside is easy to use for the basic MPI functions but needs experience and training for collectives. Also, MPInside needs a better user interface and more features such as support to calculate the average message sizes.

REFERENCES

[1] Intel 5400 Chipset - Technical Documents, www.intel.com/Products/Server/Chipsets/5400/5400-technicaldocuments.htm

[2] SGI Altix ICE, http://www.sgi.com/products/servers/altix/ice/configs.html

[3] Intel Microarchitecture (Nehalem), http://www.intel.com/technology/architecture-silicon/next-gen/

[4] "An Introduction to the Intel® QuickPath Interconnect," Document Number: 320412, January 2009.

[5] InfiniBand Trade Assoc., www.infinibandta.org/home

[6] Message Passing Toolkit (MPT) User's Guide, http://techpubs.sgi.com/library/manuals/3000/007-3773-003/pdf/007-3773-003.pdf

[7] HPCToolkit, http://www.hipersoft.rice.edu/hpctoolkit/

[8] IMP, http://ipm-hpc.sourceforge.net/

[9] KOJAK, http://icl.cs.utk.edu/kojak/

[10] mpiP, http://sourceforge.net/projects/mpip

[11] PAPI, http://icl.cs.utk.edu/papi/

[12] PDT (Program Database Toolkit), http://www.cs.uoregon.edu/research/pdt

[13] SvPablo, http://www.renci.org/projects/pablo.php

[14] TAU Performance System®, http://tau.uoregon.edu

[15] MPInside Reference Manual 3.01, SGI, 2010.

[16] Scalasca (Scalable Performance Analysis of Large-Scale Applications), http://www.fz-juelich.de/jsc/scalasca/

[17] OpenSpeedShop: http://techpubs.sgi.com/library/tpl/cgi-bin/browse.cgi?coll=0650&db=bks&cmd=toc&pth=/SGI_Developer/SShop_UG

[18] Vampir - Performance Optimization, http://www.vampir.eu

[19] Paraver, http://www.bsc.es/plantillaA.php?cat_id=485

[20] Jumpshot, http://www-unix.mcs.anl.gov/perfvis/software/viewers/

[21] Vtune, http://www.intel.com/software/products/vtune/vlin/index.htm

[22] PerfSuite, http://perfsuite.ncsa.uiuc.edu/

[23] OVERFLOW-2, http://aaac.larc.nasa.gov/~buning/

[24] ECCO: Estimating the Circulation and Climate of the Ocean, www.ecco-group.org/

[25] Intel MPI Benchmarks: Users Guide and Methodology Description, Intel GmbH, Germany.