Abstract: In this paper, we describe software tools for measuring MPI process-to-process communication latency on computational clusters and a visualization technique for displaying the communication latencies. Our tools include a C-language MPI program that measures process-to-process, round-trip, blocking, communication latency between MPI processes, a Python script that formats the resulting latency measurements for visualization, and a MATLAB script for creating plots of core-to-core latencies. Our latency plots show how MPI communication latencies vary by process rank within a single MPI job. We ran our MPI measurement software on several TeraGrid sites and we present the resulting plots. Along with communication latencies, the plots also show how system schedulers distribute MPI processes across cores, processors, and nodes. These tools and techniques provide a simple and rapid method for evaluation of cluster communication performance.

Introduction: The MPI programming model hides the physical connectivity between communicating processes from the programmer and therefore contains an unstated assumption that the characteristics of the MPI communication between all processes in a job are equivalent. However, as the communication hierarchies of modern computing clusters grow, we expect to see differences in communication performance between MPI processes based on the physical connectivity between communication processes [1]. MPI communication performance has multiple measurable aspects and MPI performance is often characterized [2] by both bandwidth and latency. Here we limit our discussion of MPI communication performance to measures of communication latency between processes. For modern, many-core, multi-processor, multi-node clusters, our expectation is that the lowest communication latencies will between processes running on different cores within the same processor. We expect communication latencies between processes running on different processors within the same node to be somewhat higher. Then, we expect highest latency between processes running on separate nodes to be the highest.

In order to examine communication latencies for multi-core, multi-processor, multi-node systems, we perform a two step process. First, we measure the communication latencies between processes in an MPI job. Next, if we measure difference in latencies between processes, we try to correlate the latencies and the physical connectivity between processes. Correlation of the physical connectivity between processes may be difficult to determine. However, information regarding the mapping of MPI processes to system cores can be obtained while the MPI program is executing.

Method: The primary goal of this effort was to evaluate how point-to-point MPI communication latencies vary on modern multi-level hierarchical computing clusters. We identified three key system requirements for this study. First, we must start up a multi-process (task) MPI job, and measure the point-to-point, round trip, blocking communications latencies between each pair of processes. The code must run on multi-core, multi-processor, multi-node systems. Second, the code must provide a way for us to correlate differences in latencies between processes to physical connectivity between communication processes. Third the summary must be presented in an easy to understand manner.

We examined existing MPI communication performance measurement tools, such as MPPTest [3] that are widely available. In addition, excellent guidance on how to produce meaningful performance measurements has been published by the authors of existing tools [4]. We examined existing performance measuring tools and evaluating their capabilities for our purposes. These tools are useful and we may adapt our technique to using the output of these tools. However, as an initial implementation, we developed our own simple MPI measurement code, following the guidelines provided by the authors for producing meaningful measurements.

We developed a simple MPI program that measures the round-trip, blocking, MPI latency between all MPI processes running in as a single job on a cluster. As this program executes on a cluster, it builds an MPI communication performance log that contains physical node ID, the average communication latency between every process pair, and a measure of the variation in communications latency for each measurement. We then plot these performance measures in a format that we call surface plots. These plots are a graphical representation of the round trip MPI communication latency between processes ordered by rank ID.

Measure-MPI (mmpi.c) Implementation: The C language MPI codes is designed to start up an MPI job, collect communication measurements, output communication latency information and then exit. The processing occurs in two stages: 1) collect process to processor information, and 2) measure process to process communication latency. The code is written so the all information is returned to the rank 0 process, and the rank 0 process outputs the performance log.

To collect the process to processor information, at start-up each process calls gethostname(). This returns a string with host information. The host information varies by system. As an example, here is what is returned on TACC Ranger: i182-102.ranger.tacc.utexas.edu. Each process then sends these results back to the rank 0 process.

The communication pattern that is used to collect the MPI communication latency proceeds as follows. The code loops through each pair of processes and sends a message using MPI_Send() from lowest rank to a higher rank. This originating process then moves to a blocking MPI_Recv(). The higher ranking process is sitting on an MPI_Recv. Once the message is received, the higher rank process immediately sends a return message of the same length. Communications between processes of equal rank are not measured. Using this scheme, the total number of comm. paths is defined by TP = (N*N – N)/2. For a 4 process job, the comm. order will be in the order shown in Table 1.

Measurement Order	Originating Rank	Responding Rank
1	0	1
2	0	2
3	0	3
4	1	2
5	1	3
6	2	4

Table 1: Communication Pattern in the mmpi.c test program showing the order in which rank-to-rank round-trip communication paths are measured in a 4 process MPI job.

MPI_Barrier() calls are used to synchronize processes and to ensure that receiving processes are waiting when the sends are sent. MPI_Wtime() is called prior to the send and it is called once the returning message is received. The difference between the receive time and the send time is the round trip comm. latency. We use a fixed message size of 64 byte in order to minimize the impact of buffering on the communication measurements. When a communication latency values has been determined, the results are send back to the rank 0 node for collection.

Once the communication latency between each process pair has been measured, the measurements are repeated. The measurements are repeated a configurable number of times. The standard deviation of the variation in latencies is calculated.

The mmpi.c program determines the number of process to process measurements to make by calling MPI_Comm_size(). Two key initialization values Message Length and Repeat Number are specified in the mmpi.h file. Default values are MsgLen = 64Bytes and repeatNum=100.

Performance Log: Once all measurements have been made, the mmpi.c program outputs a communication latency log that reports the system name, the number of processes measured, and the round trip MPI communication latencies for a small message. The message size we used in all the results we show in this report currently set at 64 bytes, although this is configurable. An example of a 4 process log is shown below. Measurements between equal ranks are defined to be 0.0 and are not measured.

```
# Testing Hostname: tg-login4
# Processors: 4 MsgLen 64 Repeats: 100
# ProcName1:rank   ProcName2:rank   AvgCommTime(usecs) StdDev
nid00670 000  nid00670 000       0.00    0.00
nid00670 000  nid00670 001      10.44    0.12
nid00670 000  nid00671 002      20.07    0.09
nid00670 000  nid00671 003      24.34    0.14
nid00670 001  nid00670 001       0.00    0.00
nid00670 001  nid00671 002      23.20    0.09
nid00670 001  nid00671 003      22.72    0.23
nid00671 002  nid00671 002       0.00    0.00
nid00671 002  nid00671 003      11.44    0.14
nid00671 003  nid00671 003       0.00    0.00
```

Measure MPI to MATLAB (mmpi2mat.py): For visualization, a Python script is used to post-process and reformat the mmpi.c performance logs. This script converts the performance measures into two matrices, each of size Number_of_MPI_Processes x Number_of_MPI_Processes. In one matrix, the data values are the average communication latency between the two processes in microseconds. In the second matrix, the data values are the standard deviation of the communication latency between processes over the number of repeated measurements. The number of repeated measurements in the examples we present is 100. The matrices are symmetric around the diagonal which represents duplication of information. Rank to rank latencies can be read as either columns or rows.

Interpretation: We have run our MPI measurement codes on several TeraGrid machines. The communication latency plots produced by our system are shown in figures 1 through figures 8.

All the attached plots show average, point-to-point, round-trip, MPI communication latency for a 64 byte message over 100 measurements for 32 process MPI jobs on TeraGrid systems. Each mesh point represents a communication path between processes. Communication paths between matching ranks (shown on a diagonal in the images) are not measured and are specified as 0.0.

The communication latency surface plots are typically read starting at the lower left. The left most column represents the communication latencies between the rank 0 process, and the other processes. The values at the origin in the bottom left represents the latency measurement between rank 0 and rank 1 processes, and the top left corner represents the latency measurement between the rank 0 and the rank 63 processes. As we move up the diagonal in the plot, we are seeing the measurement of the latency between the other processes in the job. Each node in the surface represents a round trip communication time measurement for a process to process. The lowest latency communication paths are shown in blue. High latency paths are shown as yellow and highest latencies are shown in red. With interpretation, these plots also provide additional information about the clusters under analysis. Through the distribution of fast communication paths, the plots show how the schedule distributes ranks across cores, processors, and nodes. They also indicate a communication hierarchy on the system, often showing core, processor, node, and switch distributions in their simple color coded diagrams.

Application to TeraGrid Sites: We have applied this technique to several of the TeraGrid sites. Our plots and interpretations of these systems are shown below.

SDSC IA-64: Figure 1 shows the communication latencies for a 32 core processor job on NCSA and SDSC IA-64 system.

NCSA IA-64:

TACC Lonestar:

NCSA Abe:

SDSC Blue Gene:

TACC Ranger:

PSC Cray XT3:

SDSC DataStar P655:

Related Work: Several MPI test programs are available notably MPPTest [4] available from Argonne National Laboratory. This full featured program provides many MPI measurements including both latency and bandwidth. We believe that MPPTest performance logs could probably be adapted for use in this system. Our mmpi.c program has some current advantages for our purpose. Our program returns all measurements back to the rank 0 process, avoiding the need to collect multiple log files for analysis. Also, we provide simple post-processing codes to help with the specific type of analysis in which we are interested. MPI Broadcast communication performance is discussed in [5]. Our method does not address MPI broadcast communications and is therefore complementary to this work.

Discussion:
The measurements and plots we produce provide a simple and easy to use technique for measuring and analyzing the communication latency performance of a clusters as well as the behavior of a the job schedulers. The codes we have developed are open source and freely available at http://scecdata.usc.edu/petasha/software. The results of our initial trials with the codes indicate substantial differences in MPI communication latencies on several of the TeraGrid system. Any important aspect of our method is its simplicity. Our program easily ported to each of the system and it should be useful on nearly any MPI-based system with a C compiler. This simple technique provides insight into two difficult to access characteristics of a cluster. One, it shows the communication latency and variability in latency for a system. Second, it shows characteristics of the scheduler. At this point, the MPI programming model does not provide easy remedies to modify codes based on these characteristics. However, we believe that tools such as the Numactl [6] command that provides precise control of mapping of MPI task onto cores will provide remedies that allow the programmer to improve the performance of some MPI applications by exploiting the communication characteristics shown with this diagnostic method.

References:
[1] Grama, A., A. Gupta, G. Karypis, V. Kumar [2004], Introduction to Parallel Computing, Second Edition
[2] Doerfler, Douglas, Ron Brightwell [2005] Measuring MPI Send and Receive Overhead Application Availability in High Performance Network Interface, CCIM Sandi National Labs
[3] Gropp, William, Ewing Lusk [2000] Reproducible Measurements of MPI Performance Characteristics, Argonne National Laboratory

[4] MPPTest – Measure MPI Performance: http://www-unix.mcs.anl.gov/mpi/mpptest/
[5] Supinski, B (2000), Accurately Measuring MPI Broadcasts in a Computational Grid, Lawrence Livermore National Lab.
[6] TACC Ranger User Guide:
http://www.tacc.utexas.edu/services/userguides/ranger/#numactl

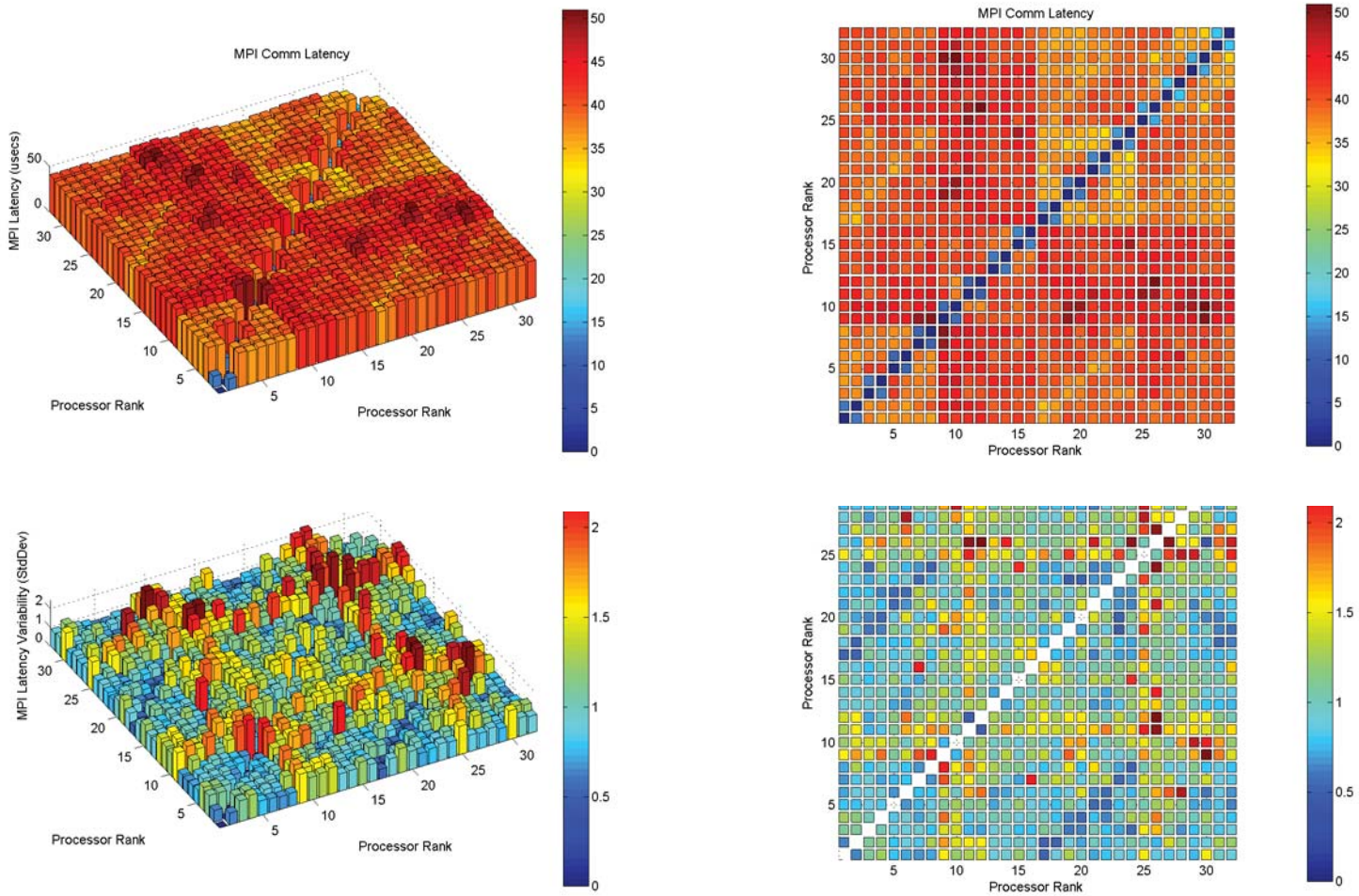

Figure 1: IBM IA64 Dual Processors (SDSC IA64)

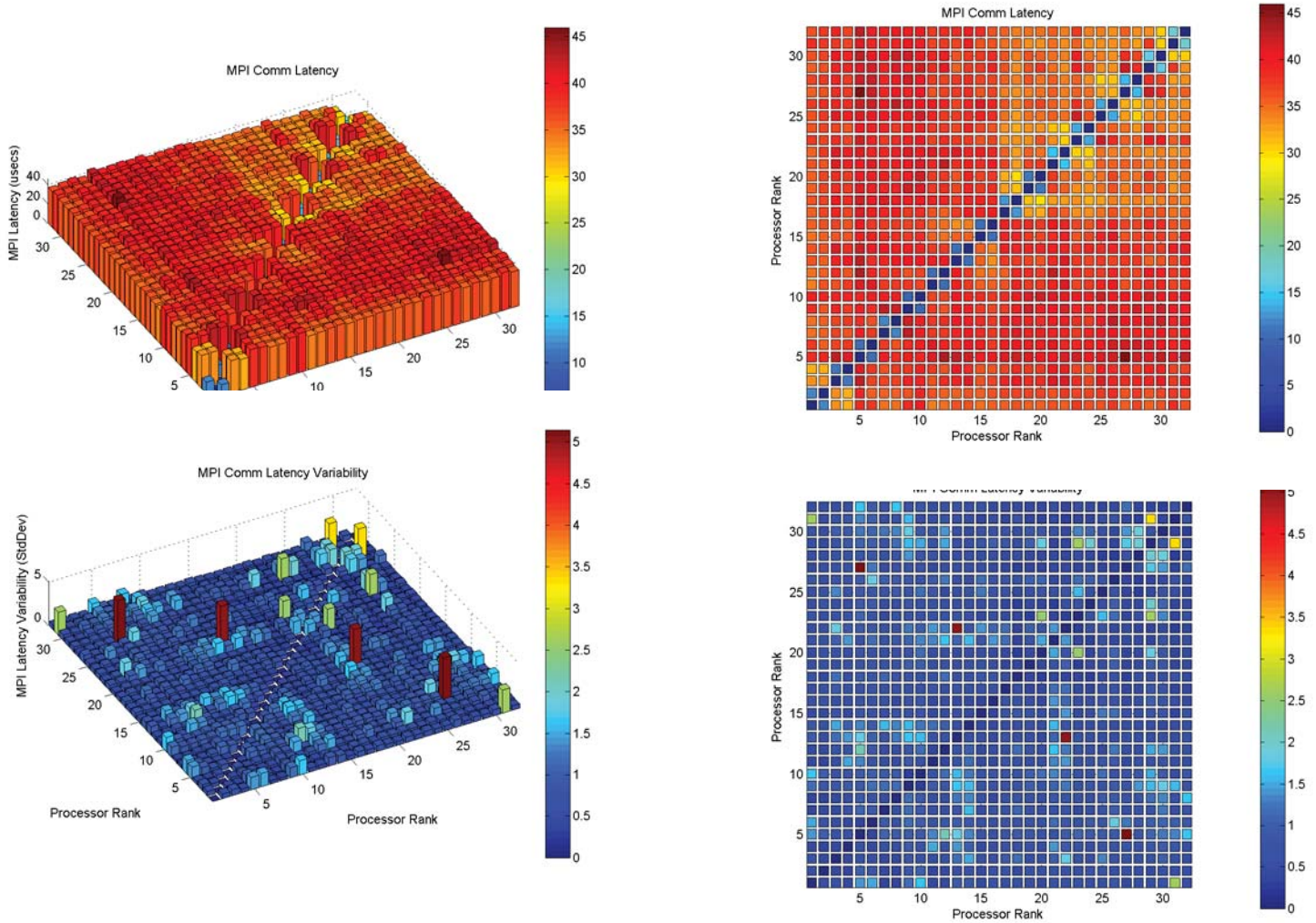

Figure 2: IBM IA64 Dual Processor (NCSA IA64)

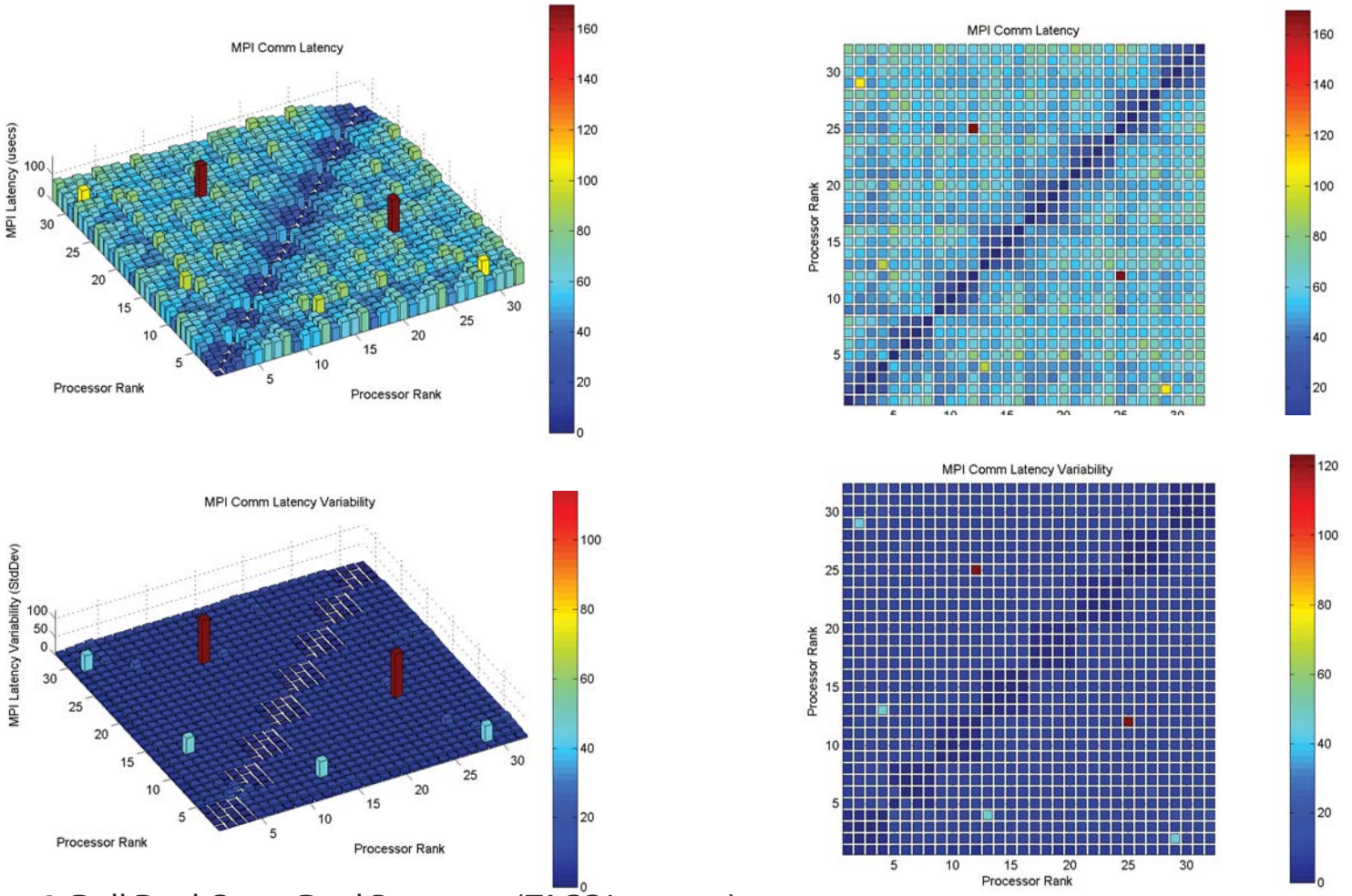

Figure 3: Dell Dual Core - Dual Processor (TACC Lonestar)

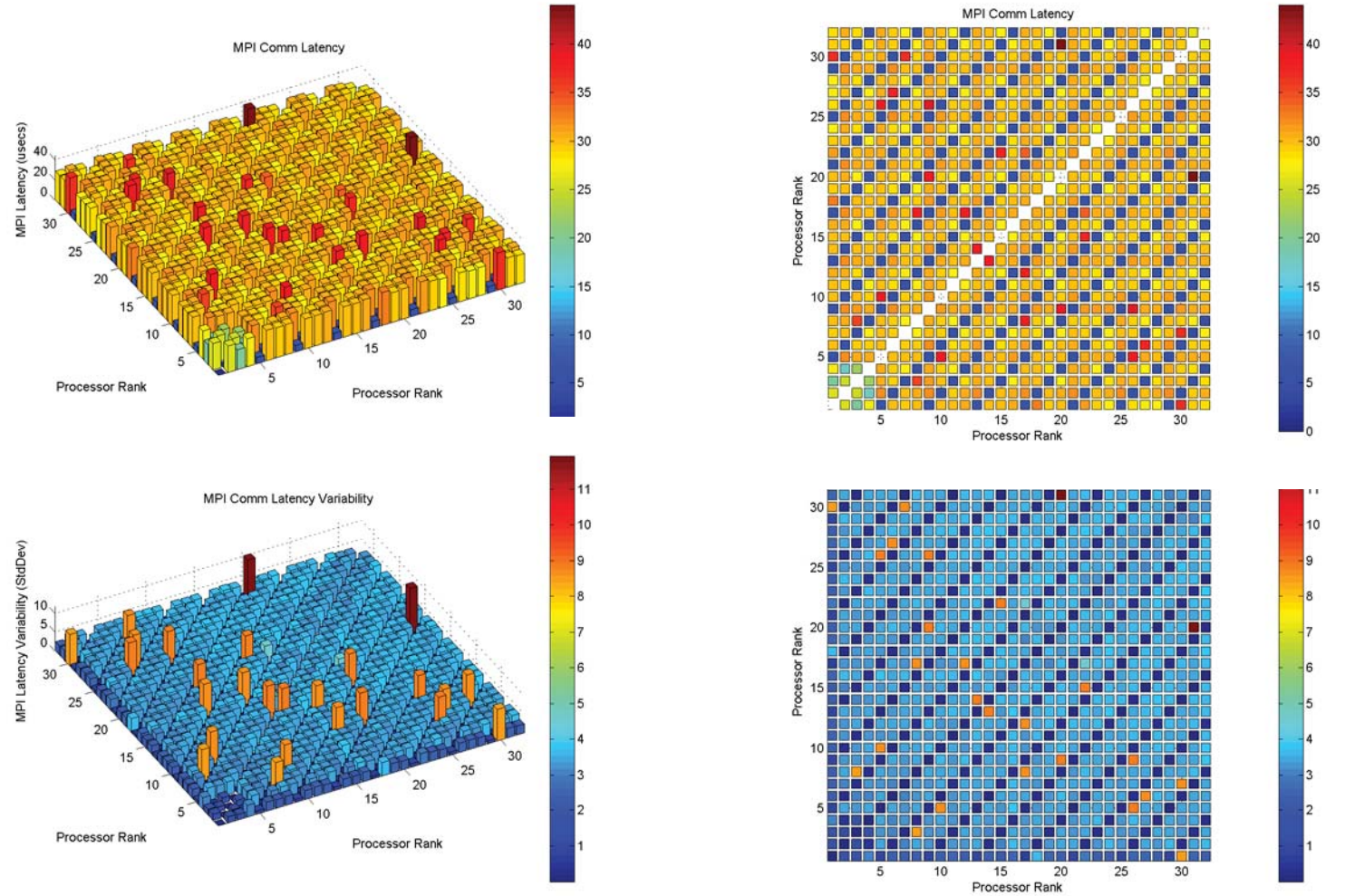

Figure 4: Intel 64 Quad Core - DualProcessor (NCSA Abe)

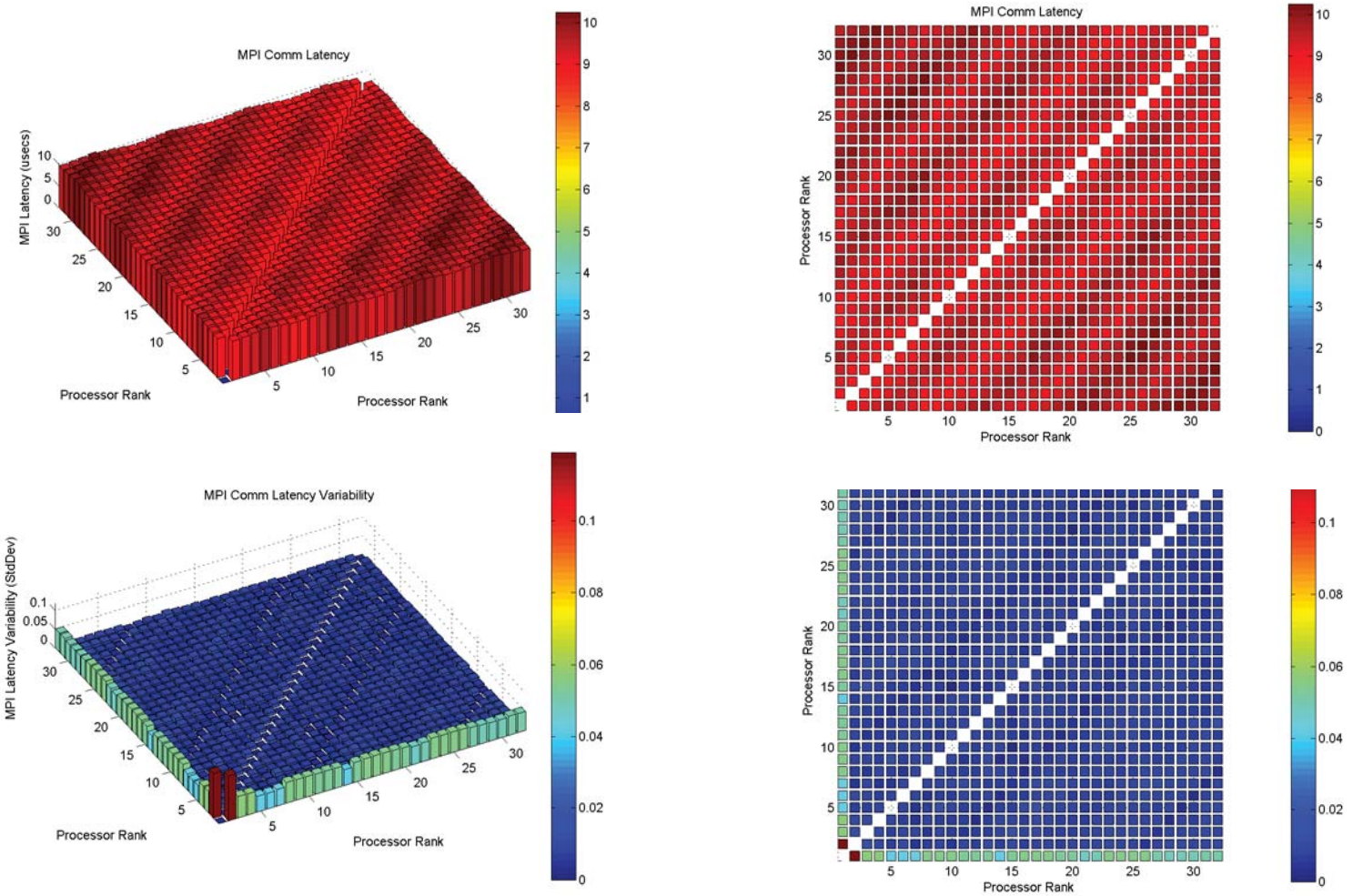

Figure 5: IBM BlueGene/L (SDSC BlueGene)

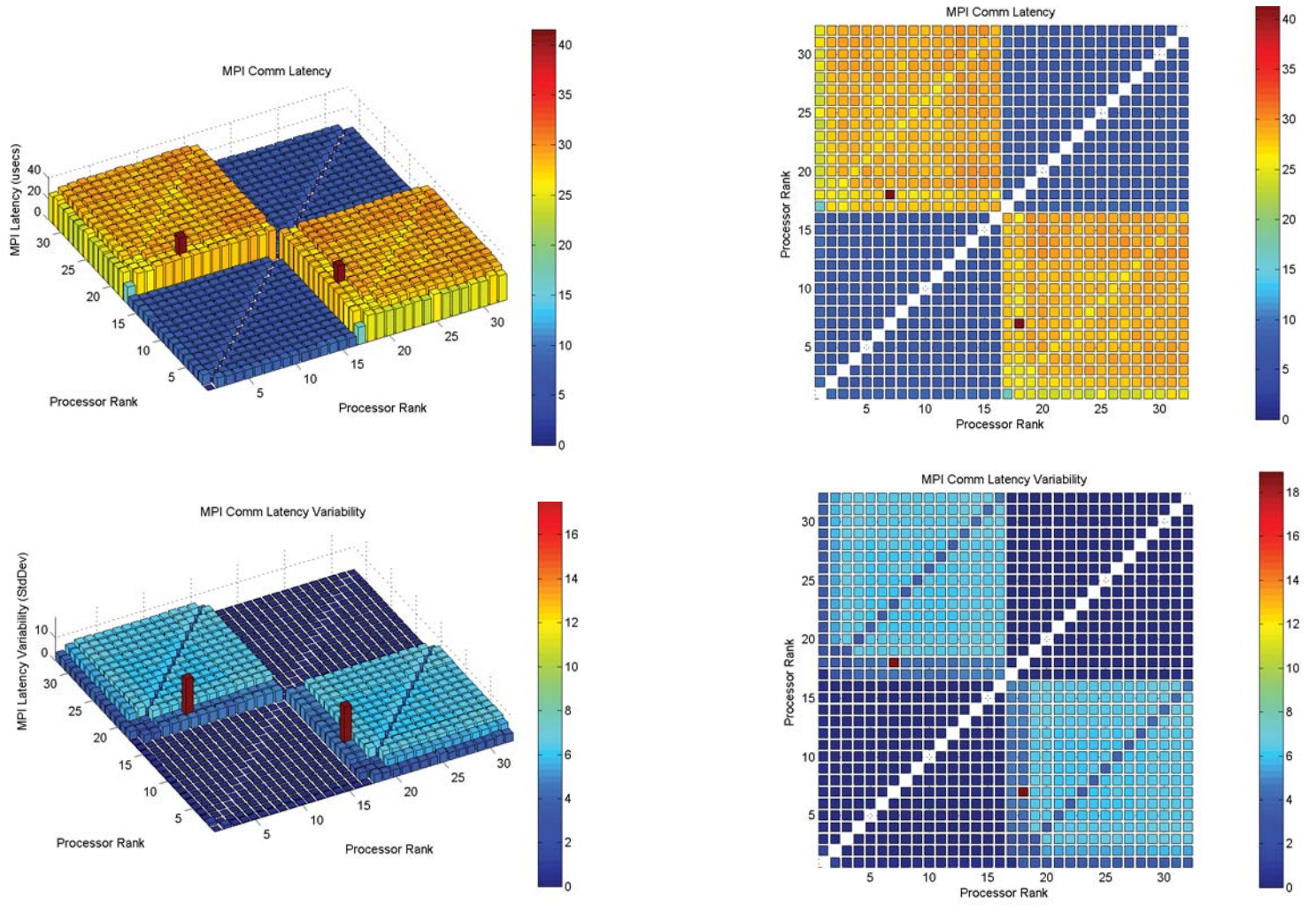

Figure 6: Sun Constellation Quad Core - Quad Processor (TACC Ranger)

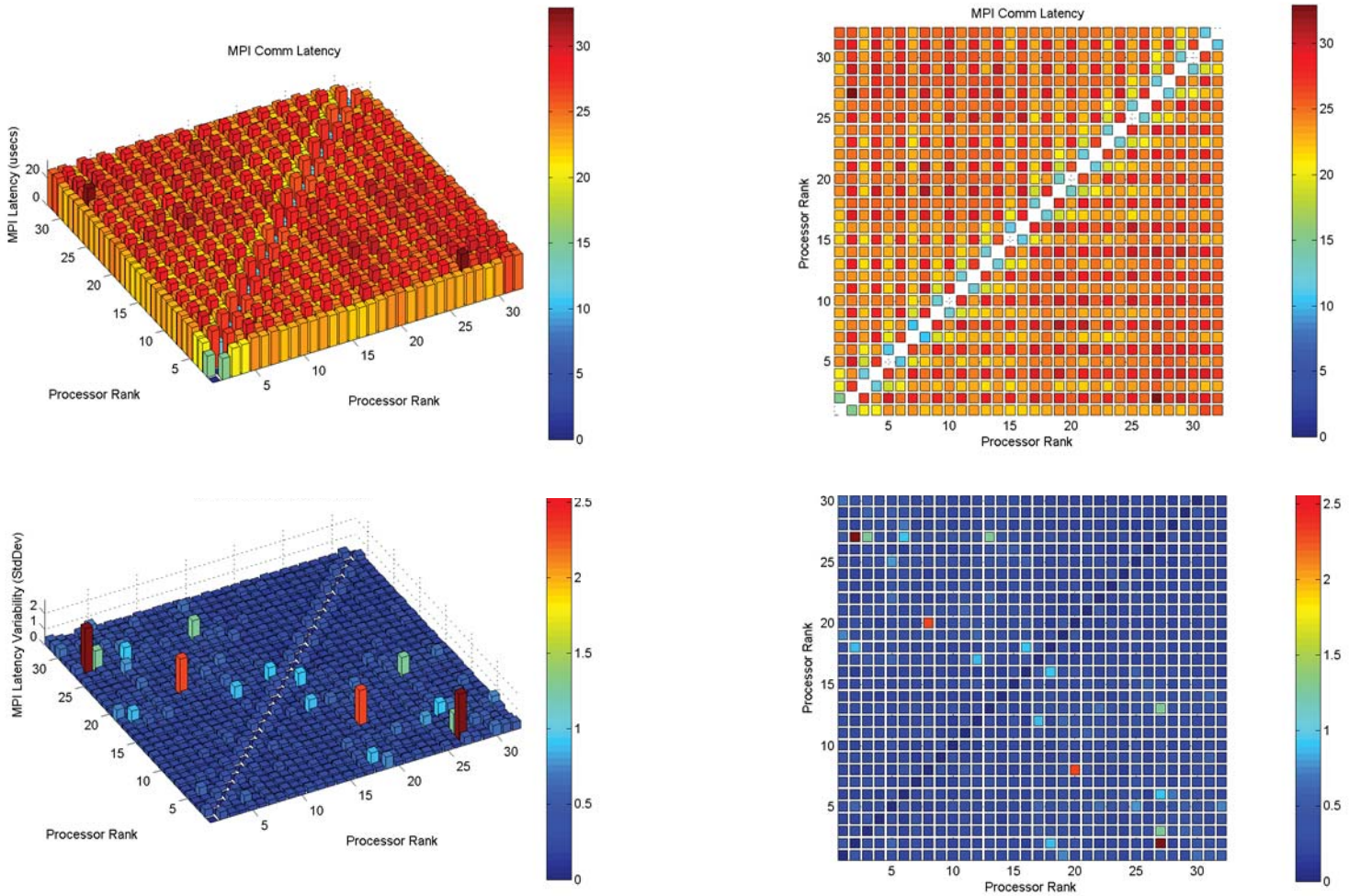

Figure 7: Cray XT3 MPP Dual Processor Nodes (PSC Big Ben)

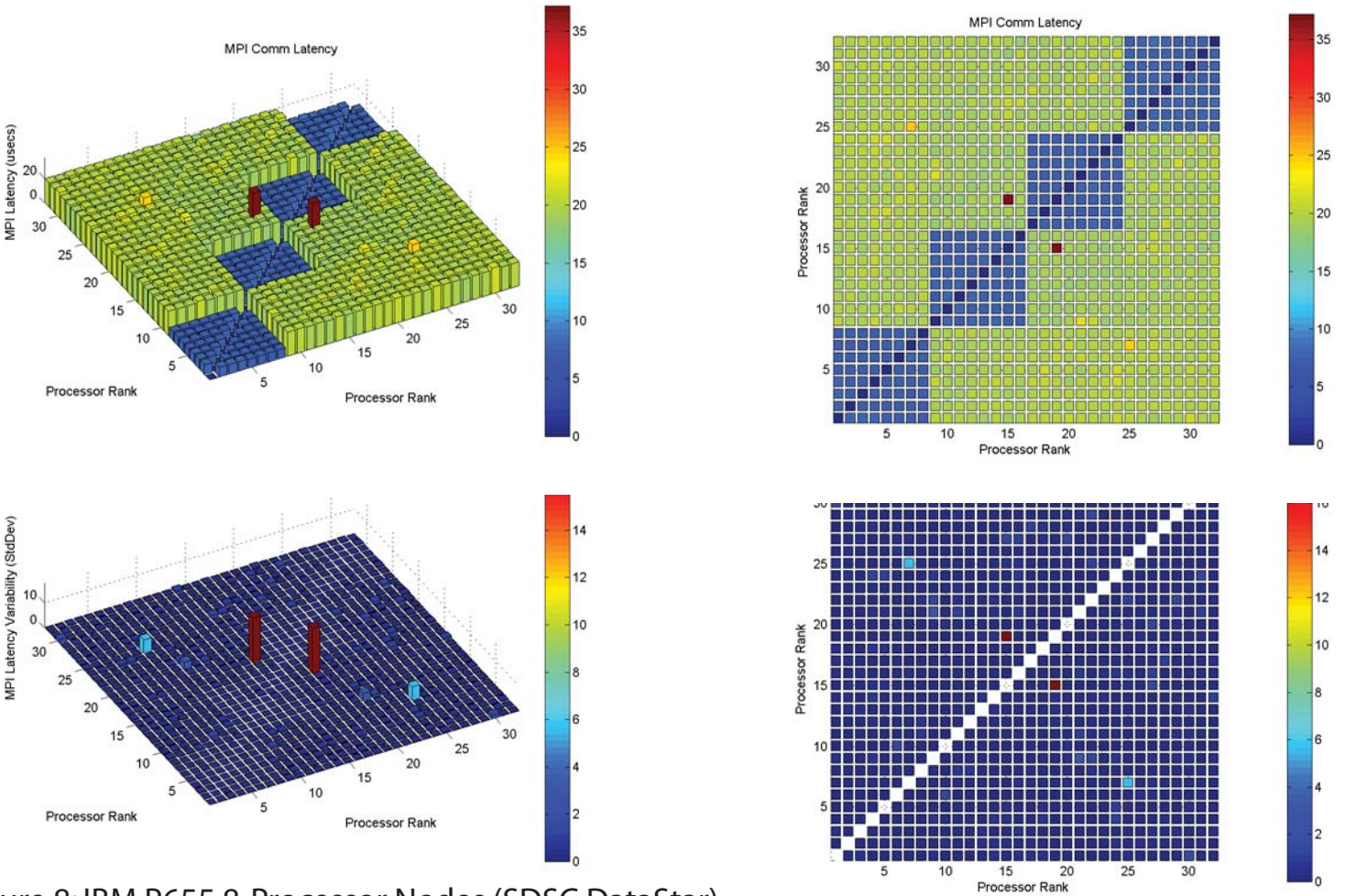

Figure 8: IBM P655 8-Processor Nodes (SDSC DataStar)