Spring 2016

HPC Made Easy: Using Docker to Distribute and Test Trilinos

Sean J. Deal
College of Saint Benedict/Saint John's University, sjdeal@csbsju.edu

HPC Made Easy: Using Docker to Distribute and Test Trilinos

An All-College Thesis

College of Saint Benedict/Saint John's University

In Partial Fulfillment of the Requirements

for Distinction in the Department of Computer Science

by Sean Deal

April 2016

Project Title: HPC Made Easy: Using Docker to Distribute and Test Trilinos

Approved by:

Mike Heroux

Scientist-in-Residence, Department of Computer Science

Imad Rahal

Associate Professor and Chair, Department of Computer Science

---

Noreen Herzfeld

Professor, Department of Computer Science

Emily Esch

Director, All-College Thesis Program

Abstract

Virtualization is an enticing option for computer science research given its ability to provide repeatble, standardized environments, but traditional virtual machines have too much overhead cost to be practical. Docker, a Linux-based tool for operating-system level virtualization, has been quickly gaining popularity throughout the computer science field by touting a virtualization solution that is easily distributable and more lightweight than virtual machines. This thesis aims to explore if Docker is a viable option for conducting virtualized research by evaluating the results of parallel performance tests using the Trilinos project.

Contents

1 Introduction

Virtualization - the isomorphic mapping of a virtual guest system to a real host system - is an enticing option for many computing tasks. In particular, virtual machines (VMs) are commonly used for several purposes. VMs can be summarized as virtual systems that replicate an entire machine - for instance, VM software such as Oracle VM VirtualBox can be used to run a virtual instance of Ubuntu Linux on a Windows machine or vice versa. This level of virtualization is achieved by placing virtualizing software between the software and hardware of the host machine. When creating an instance of a system virtual machine, virtualizing software provides a guest operating system and facilitates access to the native hardware of the host machine. In the case of differing system architectures between the guest and host machines, the virtualization software also translates machine instructions [24].

Virtual machines are sufficient for most casual computing needs, such as wanting to run Windows software on a Mac. Due to their standardized environments, VMs have also been used for more serious computing endeavors, such as software development and research. However, VMs tend to be too impractical for repeatable and reliable research, as they are not easily scalable and cam complicate the pipeline of studies using different combinations of tools [3].

Docker is a new technology that has been making waves in the computer science field, touting a method of virtualization that is both easily distributable and more lightweight than virtual machines. As Docker has been rising in popularity, it has caught the attention of researchers as a possible method of simplifying repeatable research [3]. This thesis aims to test the performance of Docker containers compared to native hardware. By running tests from the Trilinos project in parallel, we will evaluate various performance metrics in both environments. We will then attempt to conclude whether or not Docker truly can be of use to computer science researchers and provide an easier way to conduct repeatable research without sacrificing performance.

2 Background

2.1 Docker

Docker is a tool for operating system-level virtualization, which is a server virtualization method where the operating system's kernel allows for multiple isolated userspace instances, referred to as containers. This allows multiple users to run operations as if they are working on their own dedicated server, while these containers are being run off of a single server. In addition, the server administrator has power to regulate workloads across these isolated containers. Because these containers are completely isolated, operations executed in one container will not affect other containers, even if they are running simultaneously [17].

Docker started its life as a component of the 'Platform as a Service' provider dotCloud. In March 2013, dotCloud released Docker as an open source project. Docker was originally built using Linux Containers (LXC), a userspace interface for the Linux kernel that allows users to create and manage Linux containers. LXC's primary goal "is to create an environment as close as possible to a standard Linux installation but without the need for a separate kernel" [5].

Docker was touted as a repeatable and lightweight virtualization solution due to its heavy focus on isolation of both resources and file systems [1]. The benefits of Docker were embraced immediately by developers. One of the most welcomed benefits was the use of Docker for environment standardization in development. Previously, testing environments varied at each step along the development cycle, but by using Docker, developers could ensure that the environments used to develop and test the software would be consistent [16].

Just about one year later, in March 2014, Docker was updated to version 0.9, which included a major change to Docker's infrastructure. Instead of exclusively using LXC to access Linux container functionalities, the Docker group developed their own execution environment called libcontainer [15]. This environment allows Docker to have direct access to container APIs instead of relying on outside technology, though Docker still supports LXC as well as other execution environments (Figure 1). This meant that Docker was now one complete package and also opened the door for Docker to run on non-Linux platforms [25]. By allowing Docker to become a self-contained complete package, libcontainer was monumental in Docker's rise to the top of the Linux container community.

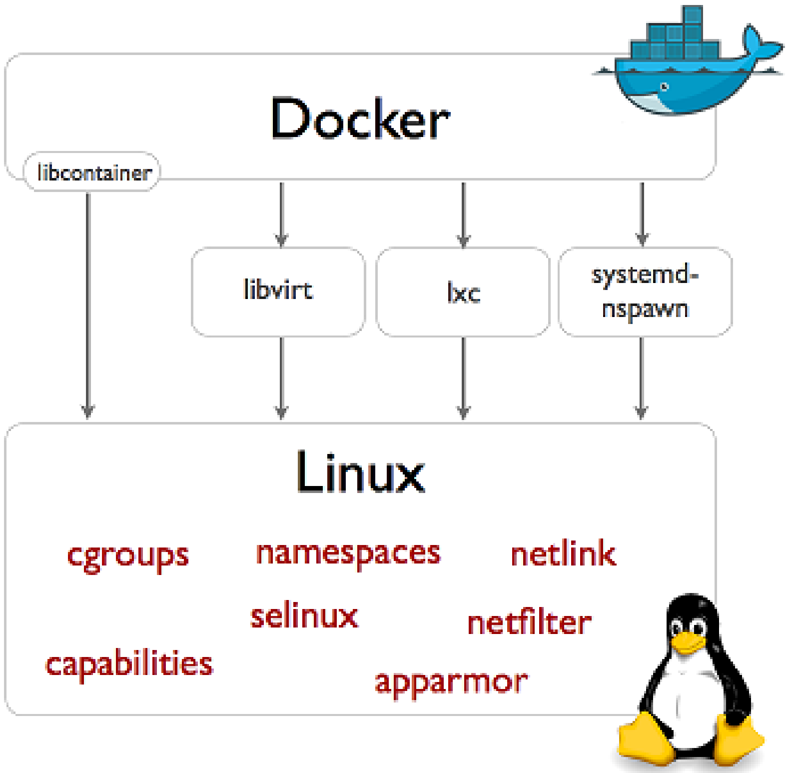

Figure 1: A diagram showing the various execution environments compatible with Docker as of Docker 0.9. libvirt, lxc, and systemdnspawn are separate from the Docker engine, while libcontainer is part of Docker. This diagram also shows Linux container APIs used by Docker [25].

Docker has quickly become the de facto standard for operating system-level virtualization. dotCloud, Inc. officially changed its name to Docker, Inc. in October 2013 to reflect its change in focus from the dotCloud service to Docker [10]. Docker, Inc. proceeded to sell dotCloud to the German company cloudControl in 2014; cloudControl filed for bankruptcy in February 2016, shutting down the dotCloud service that originally spawned Docker [20]. Meanwhile, Docker has teamed up with high-profile companies such as Google, Microsoft, Amazon, and IBM to create the Open Container Initiative.

This project is an effort to make Docker the true standard for Linux containers. Ideally, the Open Container Initiative will make the Docker container format and runtime the basis of this new standard, meaning that developers will be able to run their containerized applications in any runtime [18].

2.2 Trilinos

Trilinos is an open-source project consisting of several packages used for scalable science and engineering applications. The initial goals of the Trilinos project regarded the development of production-quality mathematical solvers. The project garnered success and recognition early, receiving an R&D 100 award in 2004 [19]. Presently, there are more than fifty packages in Trilinos covering a broad range of algorithms in the areas of computational science and engineering (Figure 2).

Trilinos strategic capability areas:

- User experience
- Parallel programming environments
- Framework and tools
- Software engineering technologies and integration
- I/O support
- Meshes, geometry, and load balancing
- Discretizations
- Scalable linear algebra
- Linear and eigensolvers
- Embedded nonlinear analysis tools

Figure 2: Trilinos strategic capability areas. The primary package used in this thesis, Epetra, falls under the area of scalable linear algebra [26].

Trilinos packages are self-contained software components, each with their own requirements and dependencies. Trilinos is predominantly a community-driven project, so keeping packages mostly isolated from each other allows Trilinos developers to focus mainly on their own package. However, packages can also be built in combination with each other. Many packages are built in close relation with others, providing expanded functionality. In addition, Trilinos has support for more than eighty third-party libraries which can be used in tandem with packages in Trilinos [13].

2.2.1 Epetra

Epetra is a package that implements serial and parallel linear algebra and provides the foundation for Trilinos solvers. Epetra's uses include construction and use of sparse graphs, sparse matrices, and dense vectors. The package also includes wrappers that provide simplified interaction with BLAS and LAPACK, two common linear algebra packages outside of Trilinos [8]. Epetra was the primary package used for performance testing in this thesis, the details of which will be explained later.

3 Benefits of Docker

3.1 Development

Because Docker containers are isolated userspace instances, they provide standardized environments that could be beneficial for both development and bug reproduction. Instead of several developers working on the same project from different machines, creating one or more standardized Docker images would provide a standard environment for all the developers to work from. This would help fix errors during development that may arise due to different developers having different versions of tools used to build and run Trilinos. In addition, standardized Docker containers can reduce costs needed for developers to maintain their own development environments.

Any image can be run on any operating system that supports Docker. For instance, a developer running Ubuntu can pull and work from a container based on Fedora. This allows developers to test their software in several environments and also allows several developers to work in the same environment regardless of their host machines' operating system (Figure 3). By having standardized images for issue handling, bugs can be reproduced in several different environments regardless of the host operating system of the issue handler.

One of the key goals of Trilinos is universal interoperability, meaning that any combination of packages and third-party libraries that makes algorithmic sense can be built into a specific installation of Trilinos [13]. However, a problem arises when attempting to use an installation of Trilinos that was built for a different purpose. If the current installation does not include necessary packages or third-party libraries, Trilinos must be completely re-built and re-installed with the new packages included. Sometimes this re-building process can be as simple as changing the configuration file, but considering the large number of packages and third-party libraries compatible with Trilinos, this is not always an easy process [2].

One of the most intriguing areas of potential benefit is the use of Dockerfiles for creating new builds of Trilinos. Dockerfiles are short scripts that are used to automatically create containers and run specified commands in them. This means that a simple Dockerfile can be used to configure, build, and install a new installation of Trilinos and provide an image with this new installation included. By providing different configuration files to the same Dockerfile, it is possible to create many different images containing different builds of Trilinos.

3.2 Distribution

A key function of Docker is the use of images in conjunction with containers. Docker images are essentially snapshots of containers that are used as bases from which other containers are created. At any time, a container can be committed to either the host image or a new image, functionally saving the changes made inside the container. These images can be shared through the Docker Hub Registry, a hosting service integrated into Docker which acts as a repository for Docker images (Figure 3). Any user can pull any public images and, if they are registered with Docker Hub, push their own images to the Registry through simple Docker commands.

As mentioned previously, standardized environments are a benefit to develop-

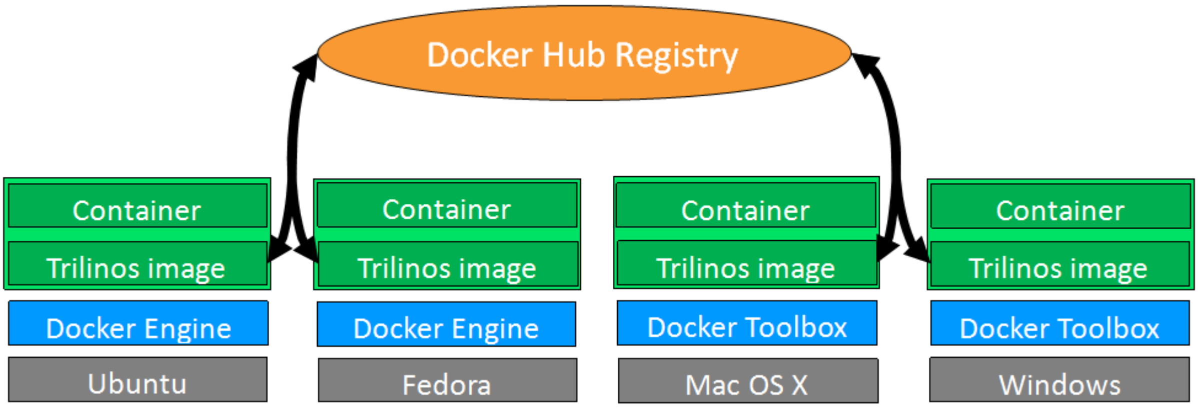

Figure 3: A visualization of multiple Trilinos developers with different host operating systems working from the same image. Changes made by any developer can be pushed to the Docker Hub and then pulled by other developers.

ers, but this quality of Docker extends to users as well. Docker images contain not only the software component of an application but also all of the application's dependencies, including binaries, libraries, scripts, and other tools [1]. This allows developers to distribute not only their software but also the entire environment. Often, software issues arise because the environment that a user runs the application in is different than that of the developer, or the user is missing a certain tool that the application requires to run. Docker images ensure that these environments are the same, effectively eliminating this issue and resolving the problem of "dependency hell" [3].

Docker can provide an easy pathway to distributing Trilinos to end users. Currently, users of Trilinos have to download the Trilinos source, configure it for their purposes, build it, and install it before they are able to start using it for their own applications. By providing images that have Trilinos already installed, users can start building their own applications right away without having to go through the Trilinos build process. These users could then create an image of their own application that uses the Trilinos image as a base and distribute that image to users of their application [9].

3.3 Comparison to Virtual Machines

Docker containers are commonly compared to virtual machines. Both containers and VMs are isolated instances, and both are built and run from a base image. The main difference is that virtual machine instances include the entire guest operating system, whereas Docker containers are run using the Linux kernel directly through the Docker engine. By using built-in Linux functions such as `cgroups` and `namespaces`, Docker containers create an isolated workspace on the same kernel that is significantly more lightweight than a virtual machine instance (Figure 4). In addition, Docker images are much smaller in size when compared to VM images; for instance, an Ubuntu VM image is roughly 943 MB [21], while an Ubuntu Docker image is only about 188 MB [23].

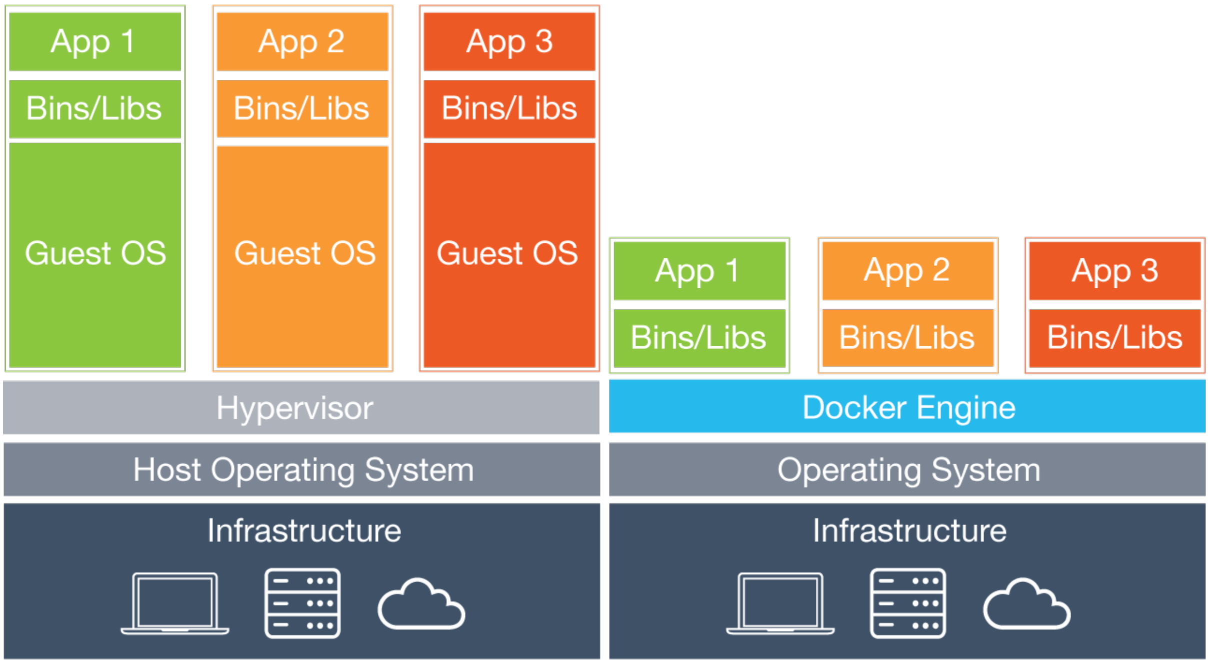

Figure 4: A visualization comparing virtual machines and Docker containers. VMs include the entire guest OS, whereas Docker builds containers directly from the operating system through the Docker Engine [7].

One drawback to Docker's approach to virtualization is that it is entirely Linux-based, meaning Docker does not run natively on Mac or Windows machines. Instead, Mac and Windows users must run a custom VM through VirtualBox that allows access to all the same Docker functionalities. For these users, Docker provides Docker Toolbox, which includes everything needed to run the VM and start using Docker [6]. However, not all Windows and Mac machines are capable of virtualization, and even if they are, enabling virtualization can be a tedious process.

On March 24, 2016, Docker announced a new beta for Docker for Mac and Windows which eliminates the need to run a VM through VirtualBox. This new beta directly utilizes xhyve and Hyper-V, the built-in virtualization tools on Mac and Windows respectively, to run an Alpine Linux distribution which in turn runs the Docker application. Instead of running a VM, users on Windows and Mac now simply have to run the Docker application [4]. This is an intriguing development that will likely make Docker much easier to use on non-Linux platforms. It does not remove all problems, though, as users still need to enable virtualization on their machines before being able to use this new Docker application.

4 Performance Testing

With Docker providing a simplified way to distribute applications, its appeal has spread to the area of computational research, including the field of high-performance computing (HPC). Trilinos itself is not an application, rather a collection of libraries, but its packages can be used for a wide range of algorithms and technologies in the areas of computational science and engineering [13]. If Docker containers allow for performance at a similar level as a native installation, the process of conducting repeatable computing research could be greatly simplified.

4.1 Methods

For this thesis, performance testing was conducted in two environments. The first was an eight node cluster named Melchior at CSB/SJU. Each node of Melchior uses an Intel Xeon processor with 12 cores (Appendix 6.1). The second environment was a series of Docker containers running on each node of Melchior (Appendix 6.2), with each container being built off of the same base image. The installations of Trilinos were identical in both environments (Appendix 6.3).

The Message Passing Interface (MPI) was used to conduct performance tests in parallel; specifically, OpenMPI was used. MPI is a realization of the message-passing model of parallel computation, which consists of a set of processes that only have local memory but can communicate with other processes by sending and receiving messages [11]. By programming with MPI, programs are able to split the workload between a number of separate processes. Understandably, MPI is used frequently by computer researchers, as it allows complex or computation-intensive tasks to be done much quicker.

As previously mentioned, Epetra is a package within Trilinos that implements serial and parallel linear algebra. While Trilinos does not provide standalone software, its packages have executable tests that can be used to evaluate performance. One of these is the Epetra BasicPerfTest. This test takes parameters for the size of a mesh grid and, if running in parallel, a matrix of processors. It then sets up a grid of the type Epetra_Map of the specified size on each processor and performs the following operations for each element of each matrix:

- MatVec - A simple solve of the equation $y = Ax$. The MatVec is performed with new and old implementations, with and without optimized storage, and with a Trans variable set to 0 and 1, indicating whether to solve for the transpose of A. All combinations are performed ten times each, resulting in eighty operations total.
- Lower/Upper Solve - An LU factorization. Both lower and upper triangular solves are performed, varying optimized storage and transpose similarly to the MatVec for a total of eighty operations.

The test then creates a vector of the type Epetra_MultiVector of the same length as the matrix used above and performs these operations:

- Norm2 - The Euclidean norm of the vector. This operation is performed ten times.
- Dot - The dot product of the vector with itself. This operation is performed ten times.
- Update - A linear combination of the vector with itself, following the equation $w = \alpha x + \beta y$ with $\alpha = 1.0$ and $\beta = 1.0$. This operation is performed ten times.

For all operations, the BasicPerfTest returns a result in millions of floating-point operations per second (Mega FLOPs, or MFLOPs), defined as

$$\frac{Number\,of\,floating\!-\!point\,operations\,in\,a\,program}{Execution\,time \times 10^6}$$

This serves as a more reliable indicator of performance than simply recording the time spent to complete an operation, as MFLOPs values are solely dependent upon the machine and the program [22].

The test was run several times, varying both the number of processes and the problem size. Grid sizes of 1000, 2000, and 4000 square were used, and each grid size was tested using 1, 8, 16, and 48 processes. The total number of equations evaluated for a given test is equal to $g^2 * p$ where g is the grid size and p is the number of processes. For each case, the test was performed five times, and results were recorded for the new MatVec with optimized storage and Trans=0, the lower triangular solve with optimized storage and Trans=0, the 10 Norm2's, the 10 Dot products, and the 10 Updates. The harmonic mean and median of each operation were then calculated [14]. This was repeated for both the native Trilinos installation on the Melchior cluster and the installation in Docker containers. For the Docker installations, an equal number of containers was used on each node of Melchior to match the number of processes; for example, with 16 processes, 2 Docker containers were used on each of the 8 nodes of Melchior.

4.2 Results

4.2.1 Problem Size - 1000x1000

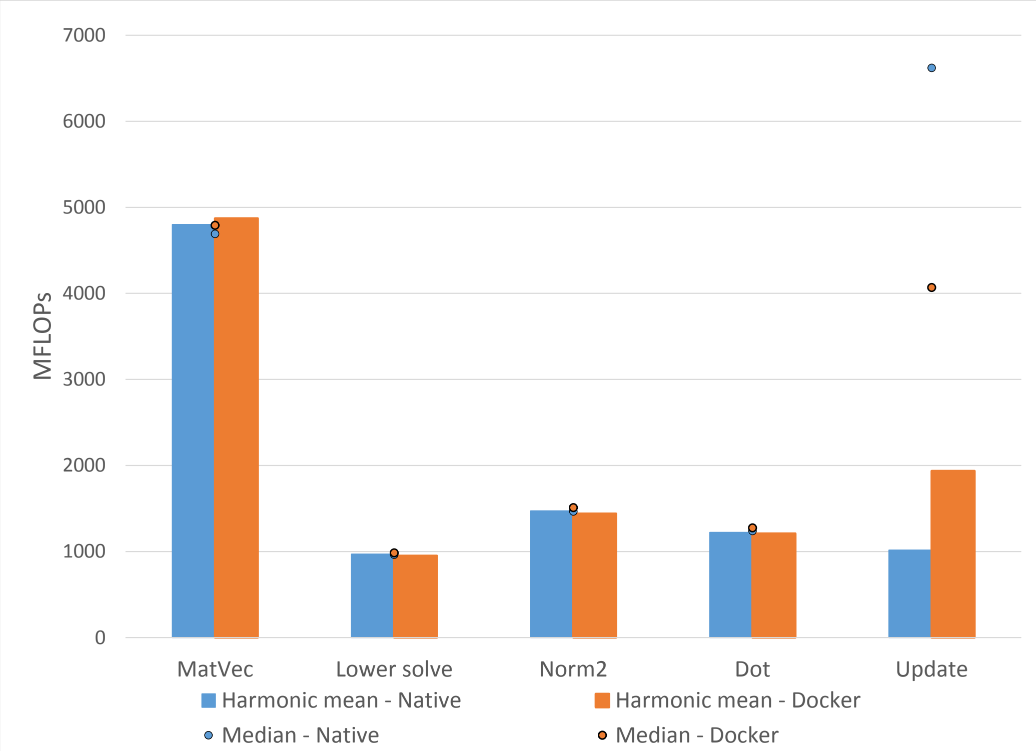

Figure 5: Performance results for the Epetra BasicPerfTest with a problem size of 1000x1000 and 1 process.

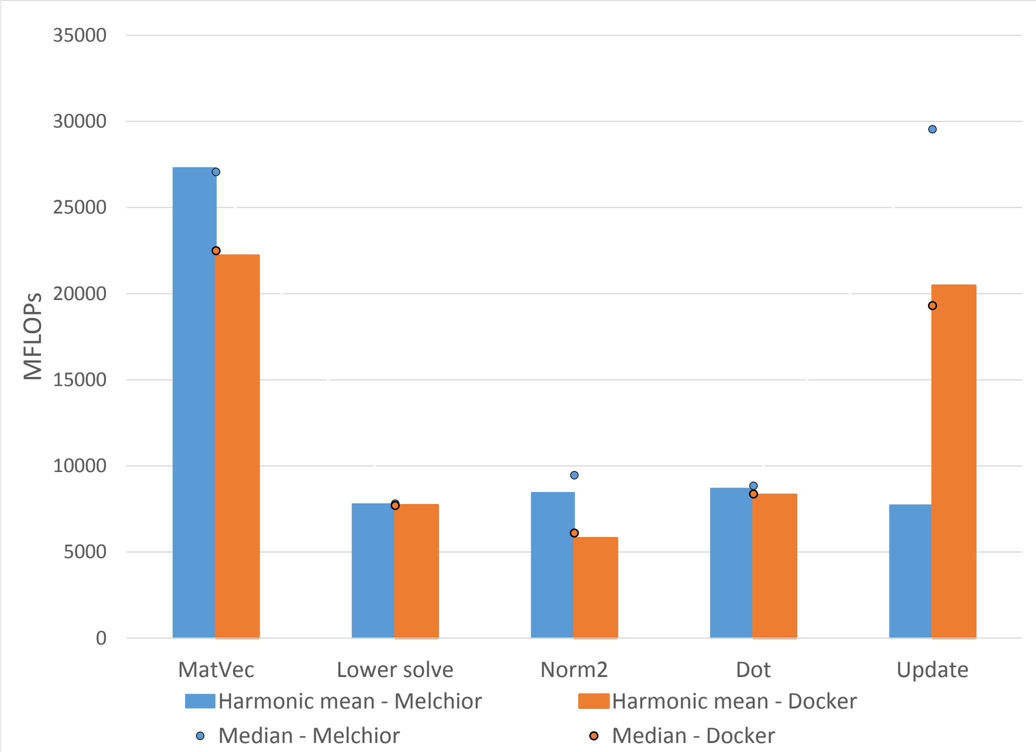

Figure 6: Performance results for the Epetra BasicPerfTest with a problem size of 1000x1000 and 8 processes. One Docker container was used on each node of Melchior.

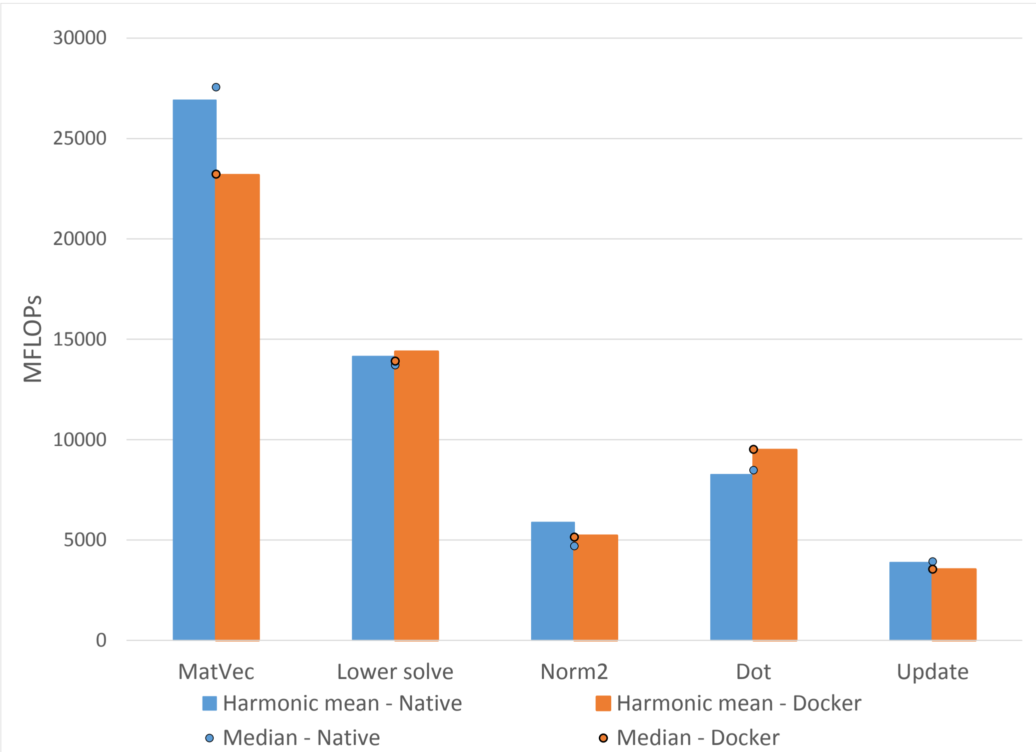

Figure 7: Performance results for the Epetra BasicPerfTest with a problem size of 1000x1000 and 16 processes. Two Docker containers were used on each node of Melchior.

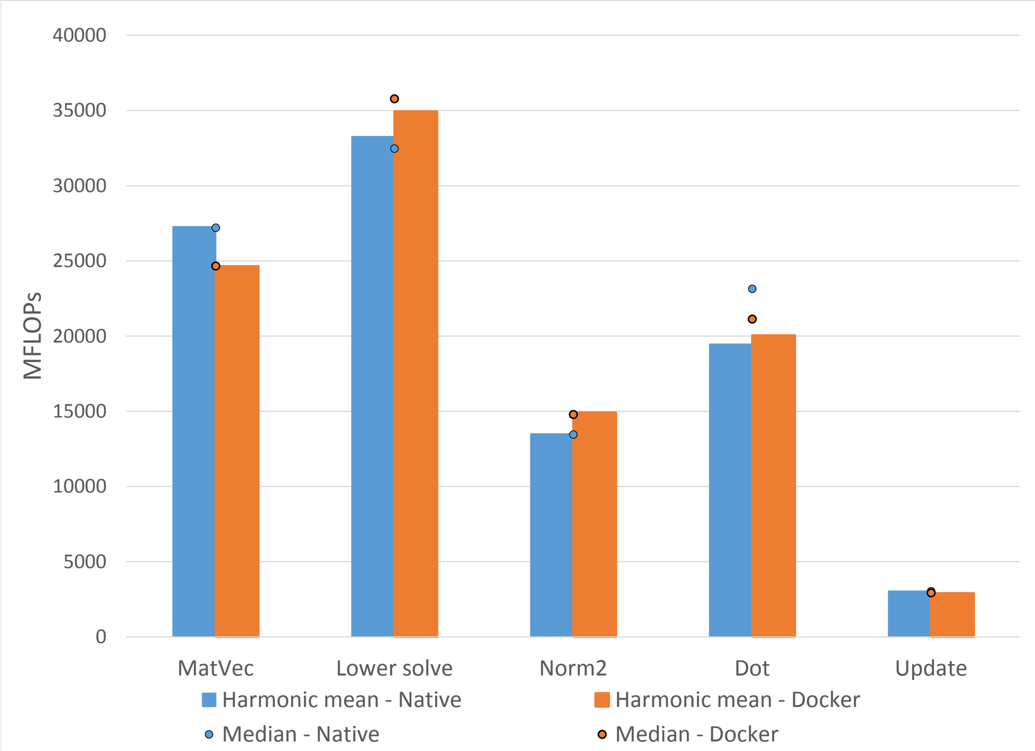

Figure 8: Performance results for the Epetra BasicPerfTest with a problem size of 1000x1000 and 48 processes. Six Docker containers were used on each node of Melchior.

4.2.2 Problem Size - 2000x2000

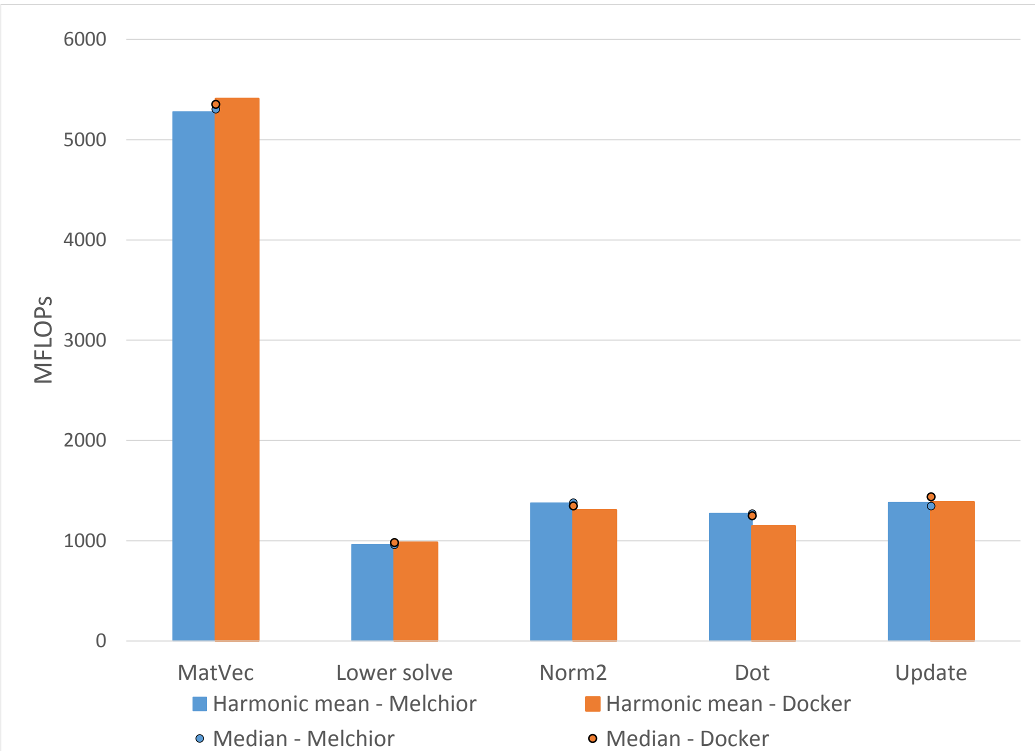

Figure 9: Performance results for the Epetra BasicPerfTest with a problem size of 2000x2000 and 1 process.

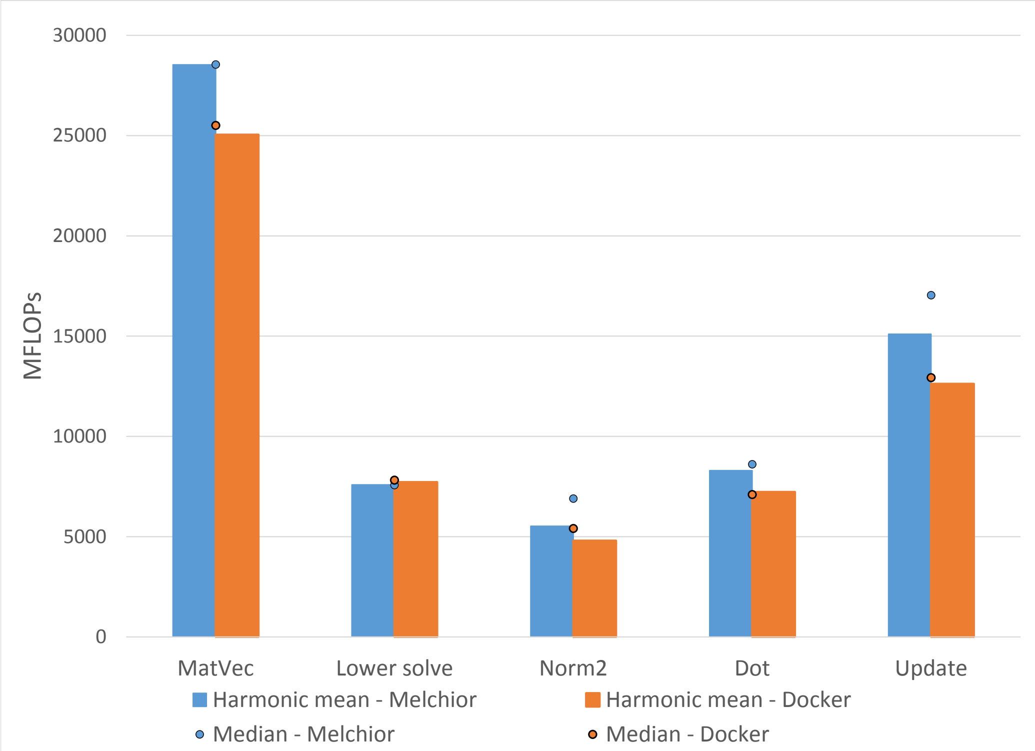

Figure 10: Performance results for the Epetra BasicPerfTest with a problem size of 2000x2000 and 8 processes. One Docker container was used on each node of Melchior.

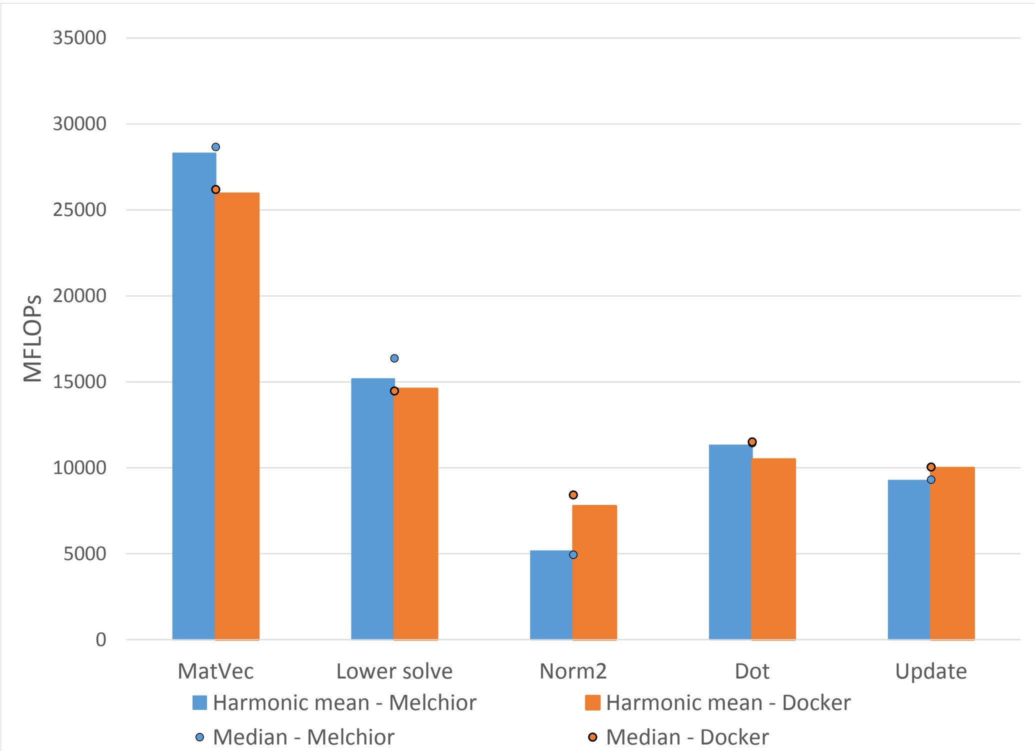

Figure 11: Performance results for the Epetra BasicPerfTest with a problem size of 2000x2000 and 16 processes. Two Docker containers were used on each node of Melchior.

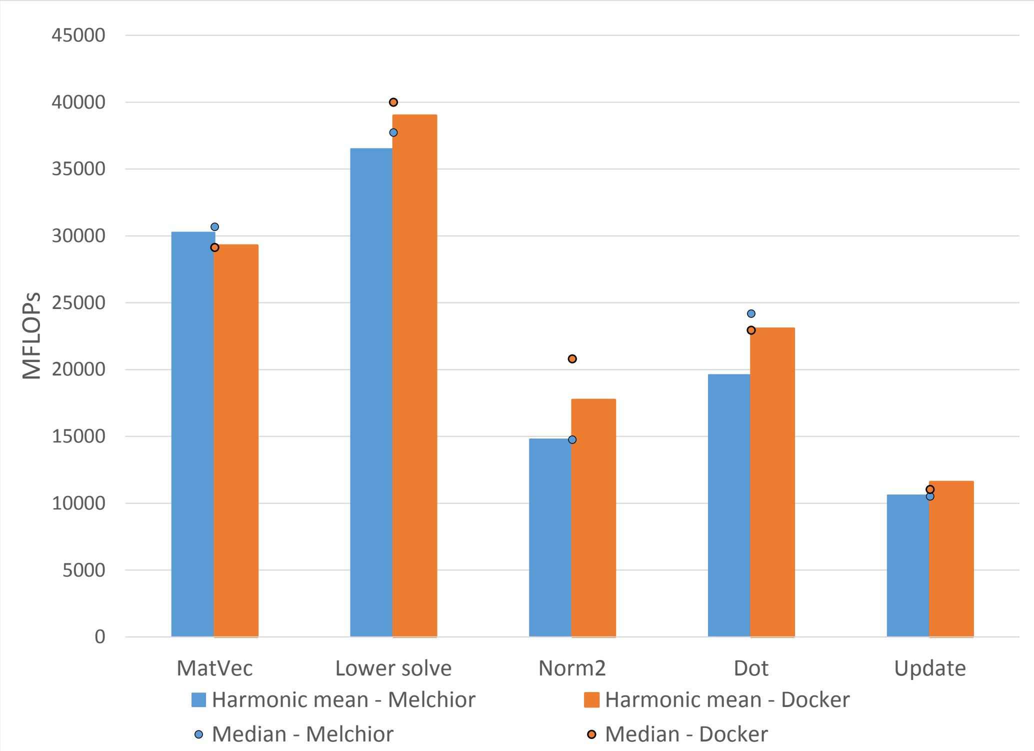

Figure 12: Performance results for the Epetra BasicPerfTest with a problem size of 2000x2000 and 48 processes. Six Docker containers were used on each node of Melchior.

4.2.3 Problem Size - 4000x4000

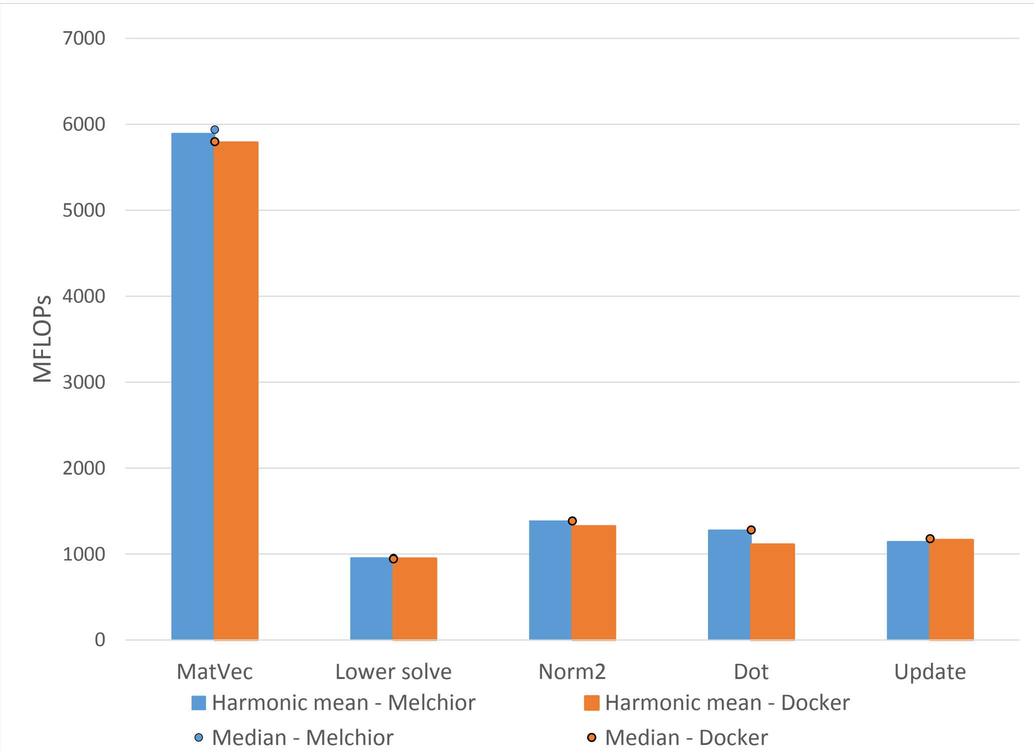

Figure 13: Performance results for the Epetra BasicPerfTest with a problem size of 4000x4000 and 1 process.

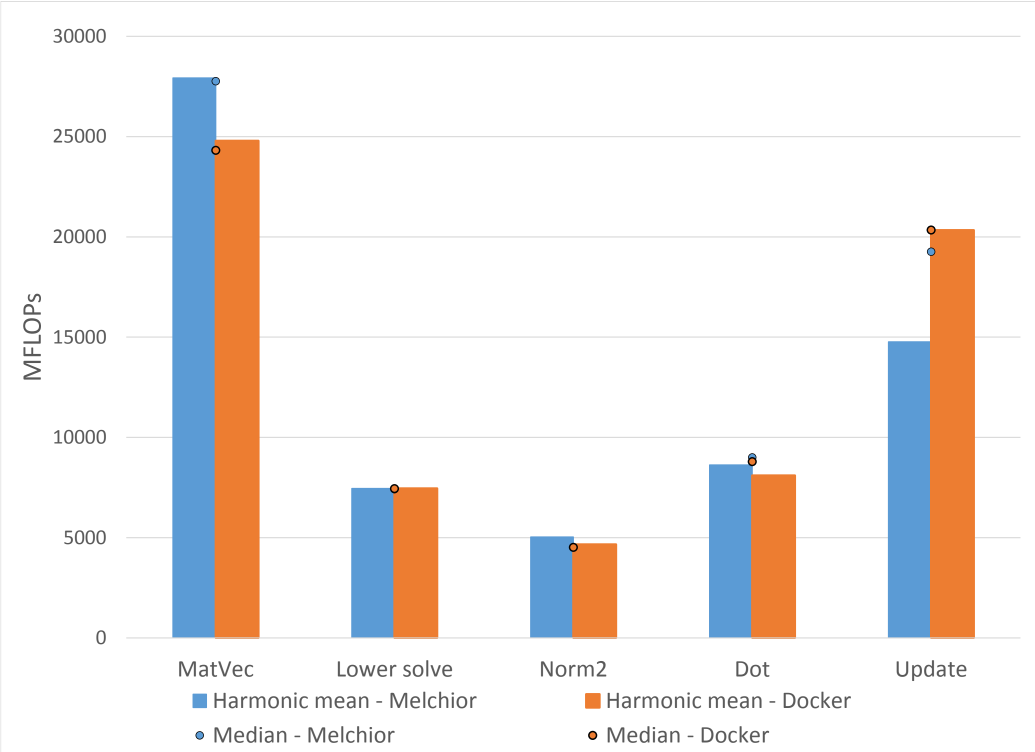

Figure 14: Performance results for the Epetra BasicPerfTest with a problem size of 4000x4000 and 8 processes. One Docker container was used on each node of Melchior.

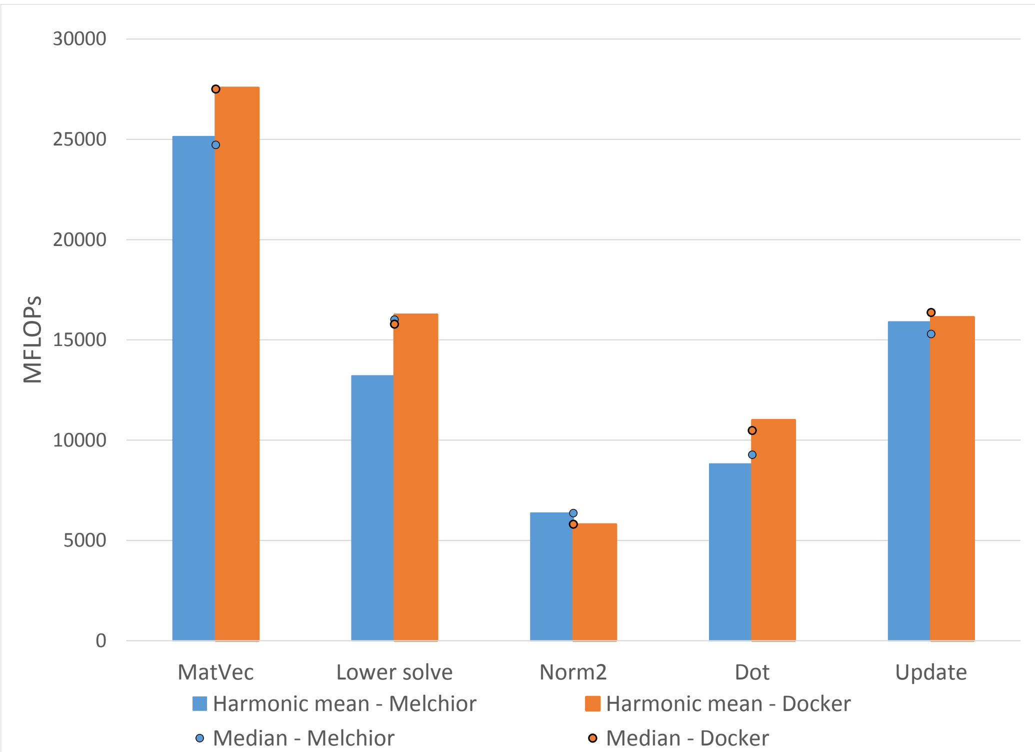

Figure 15: Performance results for the Epetra BasicPerfTest with a problem size of 4000x4000 and 16 processes. Two Docker containers were used on each node of Melchior.

When attempting to test a problem size of 4000x4000 with 48 processes on both Melchior and Docker, the test process was killed with a signal indicating that the process ran out of memory.

5 Conclusion

The results of running the Epetra BasicPerfTest both on the Melchior cluster and in Docker containers have shown that there is little to no drop in performance when performing tests using Docker containers. On serial performance tests, Melchior and Docker performed virtually the same regardless of problem size (Figures 5, 9, 13). When running in parallel with MPI, the results became slightly more erratic, though Docker consistently performed very similarly to the native installation. Larger problem sizes seemed to be more conclusive, as the results were more consistent with fewer outliers. Interestingly, Docker seemed to perform best with a larger number of processes, as the Docker containers outperformed the native installation more consistently on tests with 48 processes (Figures 8, 12). One possible explanation is the use of multiple containers on each node of Melchior for these tests. Using separate containers may have allowed each process to remain tied to a single processor, cutting down on performance lost by switching processors in the middle of the test. This certainly suggests that Docker handles scalable applications very well, though further testing may be required to definitively conclude if Docker actually improves scalability.

These results are very exciting, especially when combined with the other benefits of Docker explained earlier. Since Docker provides an easy way to distribute applications, the prospect of distributing research-related programs or packages is very promising. Specifically with Trilinos, this means that a pre-installed copy of Trilinos could be distributed through the Docker Hub, and users could begin developing and distributing their applications much quicker, and with no drop in performance.

Docker does not seem to be a complete silver bullet to virtual machines, though. The lack of native Docker support on Windows and Mac is notable and possibly unavoidable due to Docker's reliance on native Linux commands to create containers. Still, Docker has shown that they are doing all they can to improve the experience of Windows and Mac users through the Docker Toolbox and the recently announced Docker for Mac and Windows beta, which eliminates the need for VirtualBox. For Linux users, Docker indeed appears to be a favorable alternative to virtual machines, and these results show that the HPC community can also find benefit in using Docker for scalable applications.

5.1 Future Research

Moving forward, similar performance testing will be done using different Trilinos packages, such as AztecOO, which works closely with Epetra to provide an object-oriented interface to the Aztec linear solver library [12]. This will serve to further explore the performance capabilities of Docker containers. It would also be useful to vary not only the number of processes but also the number of Docker containers set up on each node of Melchior. Doing so would demonstrate whether or not the number of containers has an effect on performance and would shed more light on the scalability of Docker containers.

In addition, we will likely move to create official Trilinos images in the Docker Hub Registry. This will realize the prospect of providing Trilinos users with an installed version of Trilinos that they can develop their applications against and distribute their applications on top of to their users. It may also lead to the use of a consistent development environment for Trilinos developers. In addition, with the recent announcement of Docker for Mac and Windows, the pathway to using Docker on non-Linux platforms is becoming easier. It would be beneficial to do more in-depth exploration into using Docker on these platforms.

References

[1] Abel Avram. Docker: Automated and consistent software deployments, 2013.

[2] Roscoe Bartlett. Trilinos configure, build, test, and install quick reference guide.

[3] Carl Boettiger. An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79, 2015.

[4] Patrick Chanezon. Docker for mac and windows beta: the simplest way to use docker on your laptop, 2016.

[5] Linux Containers. What's lxc?

[6] Docker.com. Docker toolbox.

[7] Docker.com. What is docker?

[8] Epetra Doxygen. Trilinos/epetra: Linear algebra services package, 2015.

[9] John Foster. Run peridigm (and other scientific hpc codes) without building via docker, 2015.

[10] Ben Golub. dotcloud, inc. is becoming docker, inc., 2013.

[11] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, 1999.

[12] M. A. Heroux. Aztecoo users guide, 2007.

[13] M. A. Heroux and J. M. Willenbring. A new overview of the trilinos project. *Scientific Programming*, 20(2):83–88, 2012.

[14] Torsten Hoefler and Roberto Belli. Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results, 2015.

[15] Solomon Hykes. Docker 0.9: introducing execution drivers and libcontainer, 2014.

[16] Mike Kavis. Docker is open source!, 2013.

[17] Bill Kleyman. Understanding application containers and os-level virtualization, 2015.

[18] Frederic Lardinois. Docker, coreos, google, microsoft, amazon and others come together to develop common container standard, 2015.

[19] R&D Magazine. 2004 r&d 100 winner: This pearl is a real gem, 2004.

[20] Jordan Novet. Dotcloud, the cloud service that gave birth to docker, is shutting down february 29, 2016.

[21] OSBoxes. Ubuntu.

[22] Karkal Prabhu. Using mips and mflops as performance metrics, 2008.

[23] Docker Hub Registry. Ubuntu official repository.

[24] James Smith and Ravi Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufman Publishers, 2005.

[25] Chris Swan. Docker drops lxc as default execution environment, 2014.

[26] Trilinos.org. Capabilities.

6 Appendix

6.1 Computing Architecture

The computing architecture of the Melchior cluster is as follows:

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 45
model name      : Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
stepping        : 7
microcode       : 0x710
cpu MHz         : 1199.968
cache size      : 15360 KB
physical id     : 0
siblings        : 12
core id         : 0
cpu cores       : 6
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
   mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
   syscall nx pdpe1gb rdt scp lm constant_tsc arch_perfmon pebs bts
   rep_good nopl xtopology nonstop_tsc ap erfmperf eagerfpu pni
   pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm
    pcid dca sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave
   avx lahf_lm ida arat epb pln pts dtherm tpr_shadow vnmi flexpriority
    ept vpid xsave opt
bogomips        : 3790.61
clflush size    : 64
```

```
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

6.2 Docker Setup Notes

The following notes come from CSB/SJU Linux administrator Josh Trutwin, who was invaluable in helping to get Docker working properly on Melchior. IP addresses and MAC addresses have been obscured for security.

Firstly install docker, which requires a custom CentOS repository to run on our Red Hat Enterprise Linux 7 Workstation environment:

```
# cat /etc/yum.repos.d/docker.repo
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/7
enabled=0
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
```

Install docker:

```
#  yum -y --disablerepo="*" --enablerepo=dockerrepo install docker-
   engine
```

Setup a private network on 10.0.x.y on the second NIC on each node of the cluster:

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
TYPE=Ethernet
HWADDR=--:--:--:--:--:--
BOOTPROTO=none
ONBOOT=yes
BRIDGE=br0

```

```
# cat /etc/sysconfig/network-scripts/ifcfg-br0
DEVICE=br0
TYPE=Bridge
IPADDR=10.0.0.4      <----  This is different for each HPC, node 0 is 10.0.0.1, 2 is 10.0.0.2, etc
NETMASK=255.255.0.0
BOOTPROTO=none
ONBOOT=yes
DELAY=0

# /sbin/sysctl -w net.ipv4.ip_forward=1
# service network restart
```

Edit the docker service ExecStart configuration to assign a portion of the 10.1.x.y network to each docker instance - for example, hpc3 below:

```
# cat /usr/lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network.target docker.socket
Requires=docker.socket

[Service]
Type=notify
ExecStart=/usr/bin/docker daemon --bridge=br0 --fixed-cidr=10.1.4.0/24 -H fd://     <---  HPC0 is 10.1.1.0, HPC1 is 10.1.2.0 etc
MountFlags=slave
LimitNOFILE=1048576
LimitNPROC=1048576
LimitCORE=infinity

[Install]
WantedBy=multi-user.target
```

Start Docker, set to run on boot:

```
# systemctl enable docker.service
# systemctl start docker.service
```

Verify:

```
[root@hpc3 ~]# docker run -it centos /bin/bash

[root@e11e04cb0b6b /]# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
link/loopback :00:00:00:00:00 brd 00:00:00:00:00:00
inet ---.---.---.---/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
35: eth0@if36: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    state UP
link/ether --:--:--:--:--:-- brd ff:ff:ff:ff:ff:ff link-netnsid 0
inet 10.1.4.12/16 scope global eth0
valid_lft forever preferred_lft forever
inet6 ----::--:---:----:---/64 scope link
valid_lft forever preferred_lft forever
```

6.3 Trilinos Configuration Script

The same configuration script was used for the Trilinos installation on Melchior and the installation in Docker.

```
rm -rf CMakeCache.txt CMakeFiles/

EXTRA_ARGS=$@

cmake \
-D CMAKE_BUILD_TYPE:STRING=RELEASE \
-D CMAKE_INSTALL_PREFIX=../install \
```

```
\
-DTPL_ENABLE_MPI:BOOL=ON \
-DMPI_BASE_DIR:PATH=/usr/lib64/openmpi \
\
-DTrilinos_ENABLE_OpenMP:BOOL=ON \
-D Trilinos_ENABLE_TESTS:BOOL=ON \
-D Trilinos_ENABLE_ALL_PACKAGES:BOOL=OFF \
-D Trilinos_ENABLE_Epetra:BOOL=ON \
-DTrilinos_ENABLE_CXX11=ON \
-DTrilinos_ASSERT_MISSING_PACKAGES=OFF \
-DBUILD_SHARED_LIBS:BOOL=OFF \
\
-D CMAKE_VERBOSE_MAKEFILE:BOOL=OFF \
-D Trilinos_VERBOSE_CONFIGURE:BOOL=OFF \
$EXTRA_ARGS \
../publicTrilinos
```

6.4 Epetra BasicPerfTest Source Code

```
//@HEADER
// *********************************************************************
//
//                Epetra: Linear Algebra Services Package
//                  Copyright 2011 Sandia Corporation
//
// Under the terms of Contract DE-AC04-94AL85000 with Sandia Corporation,
// the U.S. Government retains certain rights in this software.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are
```

```
// met:
//
// 1. Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
//
// 2. Redistributions in binary form must reproduce the above copyright
// notice, this list of conditions and the following disclaimer in the
// documentation and/or other materials provided with the distribution.
//
// 3. Neither the name of the Corporation nor the names of the
// contributors may be used to endorse or promote products derived from
// this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY SANDIA CORPORATION "AS IS" AND ANY
// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
// IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
// PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SANDIA CORPORATION OR THE
// CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
// EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
// PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
// PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
// LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
// NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
// SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//
// Questions? Contact Michael A. Heroux (maherou@sandia.gov)
//
// *********************************************************************
//@HEADER

```

```
#include "Epetra_Map.h"
#include "Epetra_LocalMap.h"
#include "Epetra_BlockMap.h"
#include "Epetra_Time.h"
#include "Epetra_CrsMatrix.h"
#include "Epetra_VbrMatrix.h"
#include "Epetra_Vector.h"
#include "Epetra_IntVector.h"
#include "Epetra_MultiVector.h"
#include "Epetra_IntSerialDenseVector.h"
#include "Epetra_SerialDenseVector.h"
#include "Epetra_Flops.h"
#ifdef EPETRA_MPI
#include "Epetra_MpiComm.h"
#include "mpi.h"
#else
#include "Epetra_SerialComm.h"
#endif
#include "../epetra_test_err.h"
#include "Epetra_Version.h"

// prototypes

void GenerateCrsProblem(int numNodesX, int numNodesY, int numProcsX,
    int numProcsY, int numPoints,
                        int * xoff, int * yoff,
                        const Epetra_Comm  &comm, bool verbose, bool
                            summary,
                        Epetra_Map *& map,
                        Epetra_CrsMatrix *& A,
                        Epetra_Vector *& b,
                        Epetra_Vector *& bt,
```

```
                              Epetra_Vector *&xexact, bool StaticProfile, bool MakeLocalOnly);

void GenerateCrsProblem(int numNodesX, int numNodesY, int numProcsX, int numProcsY, int numPoints,
                              int * xoff, int * yoff, int nrhs,
                              const Epetra_Comm  &comm, bool verbose, bool summary,
                              Epetra_Map *& map,
                              Epetra_CrsMatrix *& A,
                              Epetra_MultiVector *& b,
                              Epetra_MultiVector *& bt,
                              Epetra_MultiVector *&xexact, bool StaticProfile, bool MakeLocalOnly);

void GenerateVbrProblem(int numNodesX, int numNodesY, int numProcsX, int numProcsY, int numPoints,
                              int * xoff, int * yoff,
                              int nsizes, int * sizes,
                              const Epetra_Comm  &comm, bool verbose, bool summary,
                              Epetra_BlockMap *& map,
                              Epetra_VbrMatrix *& A,
                              Epetra_Vector *& b,
                              Epetra_Vector *& bt,
                              Epetra_Vector *&xexact, bool StaticProfile, bool MakeLocalOnly);

void GenerateVbrProblem(int numNodesX, int numNodesY, int numProcsX, int numProcsY, int numPoints,
                              int * xoff, int * yoff,
                              int nsizes, int * sizes, int nrhs,
                              const Epetra_Comm  &comm, bool verbose, bool summary,
```

```
                        Epetra_BlockMap *& map,
                        Epetra_VbrMatrix *& A,
                        Epetra_MultiVector *& b,
                        Epetra_MultiVector *& bt,
                        Epetra_MultiVector *&xexact, bool StaticProfile
                            , bool MakeLocalOnly);

void GenerateMyGlobalElements(int numNodesX, int numNodesY, int
    numProcsX, int numProcs,
                              int myPID, int * & myGlobalElements);

void runMatrixTests(Epetra_CrsMatrix * A,  Epetra_MultiVector * b,
    Epetra_MultiVector * bt,
                    Epetra_MultiVector * xexact, bool StaticProfile,
                       bool verbose, bool summary);
void runLUMatrixTests(Epetra_CrsMatrix * L,  Epetra_MultiVector * bL,
    Epetra_MultiVector * btL, Epetra_MultiVector * xexactL,
                      Epetra_CrsMatrix * U,  Epetra_MultiVector * bU,
                         Epetra_MultiVector * btU, Epetra_MultiVector *
                          xexactU,
                      bool StaticProfile, bool verbose, bool summary);
int main(int argc, char *argv[])
{
  int ierr = 0;
  double elapsed_time;
  double total_flops;
  double MFLOPs;


#ifdef EPETRA_MPI

  // Initialize MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm comm( MPI_COMM_WORLD );
```

```cpp
#else
  Epetra_SerialComm comm;
#endif

  bool verbose = false;
  bool summary = false;

  // Check if we should print verbose results to standard out
  if (argc>6) if (argv[6][0]=='-' && argv[6][1]=='v') verbose = true;

  // Check if we should print verbose results to standard out
  if (argc>6) if (argv[6][0]=='-' && argv[6][1]=='s') summary = true;

  if(argc < 6) {
    cerr << "Usage: " << argv[0]
         << " NumNodesX NumNodesY NumProcX NumProcY NumPoints [-v|-s]" << endl
         << "where:" << endl
         << "NumNodesX         - Number of mesh nodes in X direction per processor" << endl
         << "NumNodesY         - Number of mesh nodes in Y direction per processor" << endl
         << "NumProcX          - Number of processors to use in X direction" << endl
         << "NumProcY          - Number of processors to use in Y direction" << endl
         << "NumPoints         - Number of points to use in stencil (5, 9 or 25 only)" << endl
         << "-v|-s             - (Optional) Run in verbose mode if -v present or summary mode if -s present" << endl
         << " NOTES: NumProcX*NumProcY must equal the number of processors used to run the problem." << endl << endl
         << " Serial example:" << endl
         << argv[0] << " 16 12 1 1 25 -v" << endl
```

```
        << " Run this program in verbose mode on 1 processor using a 16 X 12 grid with a 25 point stencil."<< endl <<endl
        << " MPI example:" << endl
        << "mpirun -np 32 " << argv[0] << " 10 12 4 8 9 -v" << endl
        << " Run this program in verbose mode on 32 processors putting a 10 X 12 subgrid on each processor using 4 processors "<< endl
        << " in the X direction and 8 in the Y direction.  Total grid size is 40 points in X and 96 in Y with a 9 point stencil." << endl
        << endl;
    return(1);

  }
    //char tmp;
    //if (comm.MyPID()==0) cout << "Press any key to continue..."<< endl;
    //if (comm.MyPID()==0) cin >> tmp;
    //comm.Barrier();

  comm.SetTracebackMode(0); // This should shut down any error traceback reporting
  if (verbose && comm.MyPID()==0)
    cout << Epetra_Version() << endl << endl;
  if (summary && comm.MyPID()==0) {
    if (comm.NumProc()==1)
      cout << Epetra_Version() << endl << endl;
    else
      cout << endl << endl; // Print two blank line to keep output columns lined up
  }

  if (verbose) cout << comm <<endl;

```

```
// Redefine verbose to only print on PE 0

if (verbose && comm.MyPID()!=0) verbose = false;
if (summary && comm.MyPID()!=0) summary = false;

int numNodesX = atoi(argv[1]);
int numNodesY = atoi(argv[2]);
int numProcsX = atoi(argv[3]);
int numProcsY = atoi(argv[4]);
int numPoints = atoi(argv[5]);

if (verbose || (summary && comm.NumProc()==1)) {
  cout << "␣Number␣of␣local␣nodes␣in␣X␣direction␣␣=␣" << numNodesX << endl
       << "␣Number␣of␣local␣nodes␣in␣Y␣direction␣␣=␣" << numNodesY << endl
       << "␣Number␣of␣global␣nodes␣in␣X␣direction␣=␣" << numNodesX*numProcsX << endl
       << "␣Number␣of␣global␣nodes␣in␣Y␣direction␣=␣" << numNodesY*numProcsY << endl
       << "␣Number␣of␣local␣nonzero␣entries␣␣␣␣␣␣␣=␣" << numNodesX*numNodesY*numPoints << endl
       << "␣Number␣of␣global␣nonzero␣entries␣␣␣␣␣␣=␣" << numNodesX*numNodesY*numPoints*numProcsX*numProcsY << endl
       << "␣Number␣of␣Processors␣in␣X␣direction␣␣␣=␣" << numProcsX << endl
       << "␣Number␣of␣Processors␣in␣Y␣direction␣␣␣=␣" << numProcsY << endl
       << "␣Number␣of␣Points␣in␣stencil␣␣␣␣␣␣␣␣␣␣␣=␣" << numPoints << endl << endl;
}
// Print blank line to keep output columns lined up
if (summary && comm.NumProc()>1)
```

```
    cout << endl << endl << endl << endl << endl << endl << endl << endl<< endl << endl;

  if (numProcsX*numProcsY!=comm.NumProc()) {
    cerr << "Number of processors = " << comm.NumProc() << endl
         << " is not the product of " << numProcsX << " and " << numProcsY << endl << endl;
    return(1);
  }

  if (numPoints!=5 && numPoints!=9 && numPoints!=25) {
    cerr << "Number of points specified = " << numPoints << endl
         << " is not 5, 9, 25" << endl << endl;
    return(1);
  }

  if (numNodesX*numNodesY<=0) {
    cerr << "Product of number of nodes is <= zero" << endl << endl;
    return(1);
  }

  Epetra_IntSerialDenseVector Xoff, XLoff, XUoff;
  Epetra_IntSerialDenseVector Yoff, YLoff, YUoff;
  if (numPoints==5) {

     // Generate a 5-point 2D Finite Difference matrix
    Xoff.Size(5);
    Yoff.Size(5);
    Xoff[0] = -1; Xoff[1] = 1; Xoff[2] = 0; Xoff[3] = 0;  Xoff[4] = 0;
    Yoff[0] = 0;  Yoff[1] = 0; Yoff[2] = 0; Yoff[3] = -1; Yoff[4] = 1;

     // Generate a 2-point 2D Lower triangular Finite Difference matrix
    XLoff.Size(2);
    YLoff.Size(2);
```

```
    XLoff[0] = -1; XLoff[1] =  0;
    YLoff[0] =  0; YLoff[1] = -1;

     // Generate a 3-point 2D upper triangular Finite Difference matrix
    XUoff.Size(3);
    YUoff.Size(3);
    XUoff[0] =  0; XUoff[1] =  1; XUoff[2] = 0;
    YUoff[0] =  0; YUoff[1] =  0; YUoff[2] = 1;
  }
  else if (numPoints==9) {
    // Generate a 9-point 2D Finite Difference matrix
    Xoff.Size(9);
    Yoff.Size(9);
    Xoff[0] = -1;  Xoff[1] =  0; Xoff[2] =  1;
    Yoff[0] = -1;  Yoff[1] = -1; Yoff[2] = -1;
    Xoff[3] = -1;  Xoff[4] =  0; Xoff[5] =  1;
    Yoff[3] =  0;  Yoff[4] =  0; Yoff[5] =  0;
    Xoff[6] = -1;  Xoff[7] =  0; Xoff[8] =  1;
    Yoff[6] =  1;  Yoff[7] =  1; Yoff[8] =  1;

    // Generate a 5-point lower triangular 2D Finite Difference matrix
    XLoff.Size(5);
    YLoff.Size(5);
    XLoff[0] = -1;  XLoff[1] =  0; Xoff[2] =  1;
    YLoff[0] = -1;  YLoff[1] = -1; Yoff[2] = -1;
    XLoff[3] = -1;  XLoff[4] =  0;
    YLoff[3] =  0;  YLoff[4] =  0;

    // Generate a 4-point upper triangular 2D Finite Difference matrix
    XUoff.Size(4);
    YUoff.Size(4);
    XUoff[0] =  1;
    YUoff[0] =  0;
    XUoff[1] = -1;  XUoff[2] =  0; XUoff[3] =  1;
```

```
    YUoff[1] =  1;  YUoff[2] =  1; YUoff[3] =  1;

  }
  else {
    // Generate a 25-point 2D Finite Difference matrix
    Xoff.Size(25);
    Yoff.Size(25);
    int xi = 0, yi = 0;
    int xo = -2, yo = -2;
    Xoff[xi++] = xo++;  Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++;
    Yoff[yi++] = yo  ;  Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ;
    xo = -2, yo++;
    Xoff[xi++] = xo++;  Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++;
    Yoff[yi++] = yo  ;  Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ;
    xo = -2, yo++;
    Xoff[xi++] = xo++;  Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++;
    Yoff[yi++] = yo  ;  Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ;
    xo = -2, yo++;
    Xoff[xi++] = xo++;  Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++;
    Yoff[yi++] = yo  ;  Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ;
    xo = -2, yo++;
    Xoff[xi++] = xo++;  Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++; Xoff[xi++] = xo++;
    Yoff[yi++] = yo  ;  Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ; Yoff[yi++] = yo  ;

```

```
// Generate a 13-point lower triangular 2D Finite Difference matrix
XLoff.Size(13);
YLoff.Size(13);
xi = 0, yi = 0;
xo = -2, yo = -2;
XLoff[xi++] = xo++;  XLoff[xi++] = xo++; XLoff[xi++] = xo++; XLoff[xi++] = xo++; XLoff[xi++] = xo++;
YLoff[yi++] = yo  ;  YLoff[yi++] = yo  ; YLoff[yi++] = yo  ; YLoff[yi++] = yo  ; YLoff[yi++] = yo  ;
xo = -2, yo++;
XLoff[xi++] = xo++;  XLoff[xi++] = xo++; XLoff[xi++] = xo++; XLoff[xi++] = xo++; XLoff[xi++] = xo++;
YLoff[yi++] = yo  ;  YLoff[yi++] = yo  ; YLoff[yi++] = yo  ; YLoff[yi++] = yo  ; YLoff[yi++] = yo  ;
xo = -2, yo++;
XLoff[xi++] = xo++;  XLoff[xi++] = xo++; XLoff[xi++] = xo++;
YLoff[yi++] = yo  ;  YLoff[yi++] = yo  ; YLoff[yi++] = yo  ;

// Generate a 13-point upper triangular 2D Finite Difference matrix
XUoff.Size(13);
YUoff.Size(13);
xi = 0, yi = 0;
xo = 0, yo = 0;
XUoff[xi++] = xo++;  XUoff[xi++] = xo++; XUoff[xi++] = xo++;
YUoff[yi++] = yo  ;  YUoff[yi++] = yo  ; YUoff[yi++] = yo  ;
xo = -2, yo++;
XUoff[xi++] = xo++;  XUoff[xi++] = xo++; XUoff[xi++] = xo++; XUoff[xi++] = xo++; XUoff[xi++] = xo++;
YUoff[yi++] = yo  ;  YUoff[yi++] = yo  ; YUoff[yi++] = yo  ; YUoff[yi++] = yo  ; YUoff[yi++] = yo  ;
xo = -2, yo++;
XUoff[xi++] = xo++;  XUoff[xi++] = xo++; XUoff[xi++] = xo++; XUoff[xi++] = xo++; XUoff[xi++] = xo++;
```

```
    YUoff[yi++] = yo  ;  YUoff[yi++] = yo  ; YUoff[yi++] = yo  ; YUoff[yi++] = yo  ; YUoff[yi++] = yo  ;

  }

  Epetra_Map * map;
  Epetra_Map * mapL;
  Epetra_Map * mapU;
  Epetra_CrsMatrix * A;
  Epetra_CrsMatrix * L;
  Epetra_CrsMatrix * U;
  Epetra_MultiVector * b;
  Epetra_MultiVector * bt;
  Epetra_MultiVector * xexact;
  Epetra_MultiVector * bL;
  Epetra_MultiVector * btL;
  Epetra_MultiVector * xexactL;
  Epetra_MultiVector * bU;
  Epetra_MultiVector * btU;
  Epetra_MultiVector * xexactU;
  Epetra_SerialDenseVector resvec(0);

  //Timings
  Epetra_Flops flopcounter;
  Epetra_Time timer(comm);

  int jstop = 1;
  for (int j=0; j<jstop; j++) {
    for (int k=1; k<2; k++) {
      int nrhs=k;
      if (verbose) cout << "\n**************␣Results␣for␣" << nrhs << "␣RHS␣with␣";

      bool StaticProfile = (j!=0);
```

```
    if (verbose) {
      if (StaticProfile) cout << " static profile\n";
      else cout << " dynamic profile\n";
    }

    GenerateCrsProblem(numNodesX, numNodesY, numProcsX, numProcsY, numPoints,
                       Xoff.Values(), Yoff.Values(), nrhs, comm, verbose, summary,
                       map, A, b, bt, xexact, StaticProfile, false);


    runMatrixTests(A, b, bt, xexact, StaticProfile, verbose, summary);

    delete A;
    delete b;
    delete bt;
    delete xexact;

    GenerateCrsProblem(numNodesX, numNodesY, numProcsX, numProcsY, XLoff.Length(),
                       XLoff.Values(), YLoff.Values(), nrhs, comm, verbose, summary,
                       mapL, L, bL, btL, xexactL, StaticProfile, true);


    GenerateCrsProblem(numNodesX, numNodesY, numProcsX, numProcsY, XUoff.Length(),
                       XUoff.Values(), YUoff.Values(), nrhs, comm, verbose, summary,
                       mapU, U, bU, btU, xexactU, StaticProfile, true);
```

```
runLUMatrixTests(L, bL, btL, xexactL, U, bU, btU, xexactU,
    StaticProfile, verbose, summary);

delete L;
delete bL;
delete btL;
delete xexactL;
delete mapL;

delete U;
delete bU;
delete btU;
delete xexactU;
delete mapU;

Epetra_MultiVector q(*map, nrhs);
Epetra_MultiVector z(q);
Epetra_MultiVector r(q);

delete map;
q.SetFlopCounter(flopcounter);
z.SetFlopCounter(q);
r.SetFlopCounter(q);

resvec.Resize(nrhs);


flopcounter.ResetFlops();
timer.ResetStartTime();

//10 norms
for( int i = 0; i < 10; ++i )
```

```
    q.Norm2( resvec.Values() );

  elapsed_time = timer.ElapsedTime();
  total_flops = q.Flops();
  MFLOPs = total_flops/elapsed_time/1000000.0;
  if (verbose) cout << "\nTotal MFLOPs for 10 Norm2's= " << MFLOPs << endl;

  if (summary) {
    if (comm.NumProc()==1) cout << "Norm2" << '\t';
    cout << MFLOPs << endl;
  }

  flopcounter.ResetFlops();
  timer.ResetStartTime();

  //10 dot's
  for( int i = 0; i < 10; ++i )
    q.Dot(z, resvec.Values());

  elapsed_time = timer.ElapsedTime();
  total_flops = q.Flops();
  MFLOPs = total_flops/elapsed_time/1000000.0;
  if (verbose) cout << "Total MFLOPs for 10 Dot's  = " << MFLOPs << endl;

  if (summary) {
    if (comm.NumProc()==1) cout << "DotProd" << '\t';
    cout << MFLOPs << endl;
  }

  flopcounter.ResetFlops();
  timer.ResetStartTime();

```

```
      //10 dot's
      for( int i = 0; i < 10; ++i )
        q.Update(1.0, z, 1.0, r, 0.0);

      elapsed_time = timer.ElapsedTime();
      total_flops = q.Flops();
      MFLOPs = total_flops/elapsed_time/1000000.0;
      if (verbose) cout << "Total MFLOPs for 10 Updates= " << MFLOPs << endl;

      if (summary) {
        if (comm.NumProc()==1) cout << "Update" << '\t';
        cout << MFLOPs << endl;
      }
    }
  }
#ifdef EPETRA_MPI
  MPI_Finalize() ;
#endif

return ierr ;
}

// Constructs a 2D PDE finite difference matrix using the list of x and y offsets.
//
// nx      (In) - number of grid points in x direction
// ny      (In) - number of grid points in y direction
//   The total number of equations will be nx*ny ordered such that the x direction changes
//   most rapidly:
//      First equation is at point (0,0)
//      Second at                  (1,0)
//       ...
```

```
//      nx equation at             (nx-1,0)
//      nx+1st equation at         (0,1)

// numPoints (In) - number of points in finite difference stencil
// xoff    (In) - stencil offsets in x direction (of length numPoints)
// yoff    (In) - stencil offsets in y direction (of length numPoints)
//   A standard 5-point finite difference stencil would be described as:
//     numPoints = 5
//     xoff = [-1, 1, 0,  0, 0]
//     yoff = [ 0, 0, 0, -1, 1]

// nrhs - Number of rhs to generate. (First interface produces vectors, so nrhs is not needed

// comm    (In) - an Epetra_Comm object describing the parallel machine (numProcs and my proc ID)
// map    (Out) - Epetra_Map describing distribution of matrix and vectors/multivectors
// A      (Out) - Epetra_CrsMatrix constructed for nx by ny grid using prescribed stencil
//                Off-diagonal values are random between 0 and 1.  If diagonal is part of stencil,
//                diagonal will be slightly diag dominant.
// b      (Out) - Generated RHS.  Values satisfy b = A*xexact
// bt     (Out) - Generated RHS.  Values satisfy b = A'*xexact
// xexact (Out) - Generated exact solution to Ax = b and b' = A'xexact

// Note: Caller of this function is responsible for deleting all output objects.

void GenerateCrsProblem(int numNodesX, int numNodesY, int numProcsX, int numProcsY, int numPoints,
                        int * xoff, int * yoff,
```

```
                        const Epetra_Comm  &comm, bool verbose, bool summary,
                        Epetra_Map *& map,
                        Epetra_CrsMatrix *& A,
                        Epetra_Vector *& b,
                        Epetra_Vector *& bt,
                        Epetra_Vector *&xexact, bool StaticProfile, bool MakeLocalOnly) {

  Epetra_MultiVector * b1, * bt1, * xexact1;

  GenerateCrsProblem(numNodesX, numNodesY, numProcsX, numProcsY, numPoints,
                     xoff, yoff, 1, comm, verbose, summary,
                     map, A, b1, bt1, xexact1, StaticProfile, MakeLocalOnly);

  b = dynamic_cast<Epetra_Vector *>(b1);
  bt = dynamic_cast<Epetra_Vector *>(bt1);
  xexact = dynamic_cast<Epetra_Vector *>(xexact1);

  return;
}

void GenerateCrsProblem(int numNodesX, int numNodesY, int numProcsX, int numProcsY, int numPoints,
                        int * xoff, int * yoff, int nrhs,
                        const Epetra_Comm  &comm, bool verbose, bool summary,
                        Epetra_Map *& map,
                        Epetra_CrsMatrix *& A,
                        Epetra_MultiVector *& b,
                        Epetra_MultiVector *& bt,
```

```
                                Epetra_MultiVector *&xexact, bool StaticProfile
                                    , bool MakeLocalOnly) {

  Epetra_Time timer(comm);
  // Determine my global IDs
  int * myGlobalElements;
  GenerateMyGlobalElements(numNodesX, numNodesY, numProcsX, numProcsY,
      comm.MyPID(), myGlobalElements);

  int numMyEquations = numNodesX*numNodesY;

  map = new Epetra_Map(-1, numMyEquations, myGlobalElements, 0, comm);
      // Create map with 2D block partitioning.
  delete [] myGlobalElements;

  int numGlobalEquations = map->NumGlobalElements();

  int profile = 0; if (StaticProfile) profile = numPoints;

  if (MakeLocalOnly)
    A = new Epetra_CrsMatrix(Copy, *map, *map, profile, StaticProfile);
        // Construct matrix with rowmap=colmap
  else
    A = new Epetra_CrsMatrix(Copy, *map, profile, StaticProfile); //
        Construct matrix

  int * indices = new int[numPoints];
  double * values = new double[numPoints];

  double dnumPoints = (double) numPoints;
  int nx = numNodesX*numProcsX;

  for (int i=0; i<numMyEquations; i++) {

```

```
    int rowID = map->GID(i);
    int numIndices = 0;

    for (int j=0; j<numPoints; j++) {
      int colID = rowID + xoff[j] + nx*yoff[j]; // Compute column ID based on stencil offsets
      if (colID>-1 && colID<numGlobalEquations) {
        indices[numIndices] = colID;
        double value = - ((double) rand())/ ((double) RAND_MAX);
        if (colID==rowID)
          values[numIndices++] = dnumPoints - value; // Make diagonal dominant
        else
          values[numIndices++] = value;
      }
    }
    //cout << "Building row " << rowID << endl;
    A->InsertGlobalValues(rowID, numIndices, values, indices);
  }

  delete [] indices;
  delete [] values;
  double insertTime = timer.ElapsedTime();
  timer.ResetStartTime();
  A->FillComplete(false);
  double fillCompleteTime = timer.ElapsedTime();

  if (verbose)
    cout << "Time to insert matrix values = " << insertTime << endl
         << "Time to complete fill        = " << fillCompleteTime << endl;
  if (summary) {
    if (comm.NumProc()==1) cout << "InsertTime" << '\t';
    cout << insertTime << endl;
```

```
    if (comm.NumProc()==1) cout << "FillCompleteTime" << '\t';
    cout << fillCompleteTime << endl;
  }

  if (nrhs<=1) {
    b = new Epetra_Vector(*map);
    bt = new Epetra_Vector(*map);
    xexact = new Epetra_Vector(*map);
  }
  else {
    b = new Epetra_MultiVector(*map, nrhs);
    bt = new Epetra_MultiVector(*map, nrhs);
    xexact = new Epetra_MultiVector(*map, nrhs);
  }

  xexact->Random(); // Fill xexact with random values

  A->Multiply(false, *xexact, *b);
  A->Multiply(true, *xexact, *bt);

  return;
}


// Constructs a 2D PDE finite difference matrix using the list of x and y offsets.
//
// nx       (In) - number of grid points in x direction
// ny       (In) - number of grid points in y direction
//   The total number of equations will be nx*ny ordered such that the x direction changes
//   most rapidly:
//      First equation is at point (0,0)
//      Second at                  (1,0)
```

```
//        ...
//       nx equation at              (nx-1,0)
//       nx+1st equation at          (0,1)

// numPoints (In) - number of points in finite difference stencil
// xoff     (In) - stencil offsets in x direction (of length numPoints)
// yoff     (In) - stencil offsets in y direction (of length numPoints)
//    A standard 5-point finite difference stencil would be described as:
//      numPoints = 5
//      xoff = [-1, 1, 0,  0, 0]
//      yoff = [ 0, 0, 0, -1, 1]

// nsizes  (In) - Length of element size list used to create variable block map and matrix
// sizes   (In) - integer list of element sizes of length nsizes
//     The map associated with this VbrMatrix will be created by cycling through the sizes list.
//     For example, if nsize = 3 and sizes = [ 2, 4, 3], the block map will have elementsizes
//     of 2, 4, 3, 2, 4, 3, ...

// nrhs - Number of rhs to generate. (First interface produces vectors, so nrhs is not needed

// comm    (In) - an Epetra_Comm object describing the parallel machine (numProcs and my proc ID)
// map    (Out) - Epetra_Map describing distribution of matrix and vectors/multivectors
// A      (Out) - Epetra_VbrMatrix constructed for nx by ny grid using prescribed stencil
//                Off-diagonal values are random between 0 and 1.  If diagonal is part of stencil,
//                diagonal will be slightly diag dominant.
```

```
// b      (Out) - Generated RHS.  Values satisfy b = A*xexact
// bt     (Out) - Generated RHS.  Values satisfy b = A'*xexact
// xexact (Out) - Generated exact solution to Ax = b and b' = A'xexact

// Note: Caller of this function is responsible for deleting all output objects.

void GenerateVbrProblem(int numNodesX, int numNodesY, int numProcsX, int numProcsY, int numPoints,
                        int * xoff, int * yoff,
                        int nsizes, int * sizes,
                        const Epetra_Comm  &comm, bool verbose, bool summary,
                        Epetra_BlockMap *& map,
                        Epetra_VbrMatrix *& A,
                        Epetra_Vector *& b,
                        Epetra_Vector *& bt,
                        Epetra_Vector *&xexact, bool StaticProfile, bool MakeLocalOnly) {

  Epetra_MultiVector * b1, * bt1, * xexact1;

  GenerateVbrProblem(numNodesX, numNodesY, numProcsX, numProcsY, numPoints,
                     xoff, yoff, nsizes, sizes,
                     1, comm, verbose, summary, map, A, b1, bt1, xexact1, StaticProfile, MakeLocalOnly);

  b = dynamic_cast<Epetra_Vector *>(b1);
  bt = dynamic_cast<Epetra_Vector *>(bt1);
  xexact = dynamic_cast<Epetra_Vector *>(xexact1);

  return;
}
```

```

void GenerateVbrProblem(int numNodesX, int numNodesY, int numProcsX,
    int numProcsY, int numPoints,
                        int * xoff, int * yoff,
                        int nsizes, int * sizes, int nrhs,
                        const Epetra_Comm  &comm, bool verbose, bool
                           summary,
                        Epetra_BlockMap *& map,
                        Epetra_VbrMatrix *& A,
                        Epetra_MultiVector *& b,
                        Epetra_MultiVector *& bt,
                        Epetra_MultiVector *&xexact, bool StaticProfile
                           , bool MakeLocalOnly) {

  int i, j;

  // Determine my global IDs
  int * myGlobalElements;
  GenerateMyGlobalElements(numNodesX, numNodesY, numProcsX, numProcsY,
     comm.MyPID(), myGlobalElements);

  int numMyElements = numNodesX*numNodesY;

  Epetra_Map ptMap(-1, numMyElements, myGlobalElements, 0, comm); //
     Create map with 2D block partitioning.
  delete [] myGlobalElements;

  int numGlobalEquations = ptMap.NumGlobalElements();

  Epetra_IntVector elementSizes(ptMap); // This vector will have the
     list of element sizes
  for (i=0; i<numMyElements; i++)
    elementSizes[i] = sizes[ptMap.GID(i)%nsizes]; // cycle through
       sizes array
```

```
map = new Epetra_BlockMap(-1, numMyElements, ptMap.MyGlobalElements(), elementSizes.Values(),
                          ptMap.IndexBase(), ptMap.Comm());

int profile = 0; if (StaticProfile) profile = numPoints;

if (MakeLocalOnly)
  A = new Epetra_VbrMatrix(Copy, *map, *map, profile); // Construct matrix rowmap=colmap
else
  A = new Epetra_VbrMatrix(Copy, *map, profile); // Construct matrix

int * indices = new int[numPoints];

// This section of code creates a vector of random values that will be used to create
// light-weight dense matrices to pass into the VbrMatrix construction process.

int maxElementSize = 0;
for (i=0; i< nsizes; i++) maxElementSize = EPETRA_MAX(maxElementSize, sizes[i]);

Epetra_LocalMap lmap(maxElementSize*maxElementSize, ptMap.IndexBase(), ptMap.Comm());
Epetra_Vector randvec(lmap);
randvec.Random();
randvec.Scale(-1.0); // Make value negative
int nx = numNodesX*numProcsX;


for (i=0; i<numMyElements; i++) {
  int rowID = map->GID(i);
```

```
    int numIndices = 0;
    int rowDim = sizes[rowID%nsizes];
    for (j=0; j<numPoints; j++) {
      int colID = rowID + xoff[j] + nx*yoff[j]; // Compute column ID based on stencil offsets
      if (colID>-1 && colID<numGlobalEquations)
        indices[numIndices++] = colID;
    }

    A->BeginInsertGlobalValues(rowID, numIndices, indices);

    for (j=0; j < numIndices; j++) {
      int colDim = sizes[indices[j]%nsizes];
      A->SubmitBlockEntry(&(randvec[0]), rowDim, rowDim, colDim);
    }
    A->EndSubmitEntries();
  }

  delete [] indices;

  A->FillComplete();

  // Compute the InvRowSums of the matrix rows
  Epetra_Vector invRowSums(A->RowMap());
  Epetra_Vector rowSums(A->RowMap());
  A->InvRowSums(invRowSums);
  rowSums.Reciprocal(invRowSums);

  // Jam the row sum values into the diagonal of the Vbr matrix (to make it diag dominant)
  int numBlockDiagonalEntries;
  int * rowColDims;
  int * diagoffsets = map->FirstPointInElementList();
```

```
  A->BeginExtractBlockDiagonalView(numBlockDiagonalEntries, rowColDims)
      ;
  for (i=0; i< numBlockDiagonalEntries; i++) {
    double * diagVals;
    int diagLDA;
    A->ExtractBlockDiagonalEntryView(diagVals, diagLDA);
    int rowDim = map->ElementSize(i);
    for (j=0; j<rowDim; j++) diagVals[j+j*diagLDA] = rowSums[
        diagoffsets[i]+j];
  }

  if (nrhs<=1) {
    b = new Epetra_Vector(*map);
    bt = new Epetra_Vector(*map);
    xexact = new Epetra_Vector(*map);
  }
  else {
    b = new Epetra_MultiVector(*map, nrhs);
    bt = new Epetra_MultiVector(*map, nrhs);
    xexact = new Epetra_MultiVector(*map, nrhs);
  }

  xexact->Random(); // Fill xexact with random values

  A->Multiply(false, *xexact, *b);
  A->Multiply(true, *xexact, *bt);

  return;
}

void GenerateMyGlobalElements(int numNodesX, int numNodesY, int
    numProcsX, int numProcs,
                              int myPID, int * & myGlobalElements) {

```

```
  myGlobalElements = new int[numNodesX*numNodesY];
  int myProcX = myPID%numProcsX;
  int myProcY = myPID/numProcsX;
  int curGID = myProcY*(numProcsX*numNodesX)*numNodesY+myProcX*numNodesX;
  for (int j=0; j<numNodesY; j++) {
    for (int i=0; i<numNodesX; i++) {
      myGlobalElements[j*numNodesX+i] = curGID+i;
    }
    curGID+=numNodesX*numProcsX;
  }
  //for (int i=0; i<numNodesX*numNodesY; i++) cout << "MYPID " << myPID <<" GID "<< myGlobalElements[i] << endl;

  return;
}

void runMatrixTests(Epetra_CrsMatrix * A,  Epetra_MultiVector * b, Epetra_MultiVector * bt,
                    Epetra_MultiVector * xexact, bool StaticProfile, bool verbose, bool summary) {

  Epetra_MultiVector z(*b);
  Epetra_MultiVector r(*b);
  Epetra_SerialDenseVector resvec(b->NumVectors());

  //Timings
  Epetra_Flops flopcounter;
  A->SetFlopCounter(flopcounter);
  Epetra_Time timer(A->Comm());
  std::string statdyn =        "dynamic";
  if (StaticProfile) statdyn = "static ";

  for (int j=0; j<4; j++) { // j = 0/2 is notrans, j = 1/3 is trans
```

```
    bool TransA = (j==1 || j==3);
    std::string contig = "without";
    if (j>1) contig =    "with   ";

#ifdef EPETRA_SHORT_PERFTEST
    int kstart = 1;
#else
    int kstart = 0;
#endif
    for (int k=kstart; k<2; k++) { // Loop over old multiply vs. new multiply

      std::string oldnew = "old";
      if (k>0) oldnew =    "new";

      if (j==2) A->OptimizeStorage();

      flopcounter.ResetFlops();
      timer.ResetStartTime();

      if (k==0) {
        //10 matvecs
#ifndef EPETRA_SHORT_PERFTEST
        for( int i = 0; i < 10; ++i )
          A->Multiply1(TransA, *xexact, z); // Compute z = A*xexact or z = A'*xexact using old Multiply method
#endif
      }
      else {
        //10 matvecs
        for( int i = 0; i < 10; ++i )
          A->Multiply(TransA, *xexact, z); // Compute z = A*xexact or z = A'*xexact
```

```
      }

      double elapsed_time = timer.ElapsedTime();
      double total_flops = A->Flops();

      // Compute residual
      if (TransA)
        r.Update(-1.0, z, 1.0, *bt, 0.0); // r = bt - z
      else
        r.Update(-1.0, z, 1.0, *b, 0.0); // r = b - z

      r.Norm2(resvec.Values());

      if (verbose) cout << "ResNorm = " << resvec.NormInf() << ": ";
      double MFLOPs = total_flops/elapsed_time/1000000.0;
      if (verbose) cout << "Total MFLOPs for 10 " << oldnew << " MatVec's with " << statdyn << " Profile (Trans = " << TransA
                        << ")  and " << contig << " optimized storage = " << MFLOPs << " (" << elapsed_time << " s)" <<endl;
      if (summary) {
        if (A->Comm().NumProc()==1) {
          if (TransA) cout << "TransMv" << statdyn<< "Prof" << contig << "OptStor" << '\t';
          else cout << "NoTransMv" << statdyn << "Prof" << contig << "OptStor" << '\t';
        }
        cout << MFLOPs << endl;
      }
    }
  }
  return;
}
```

```
//==========================================================================
void runLUMatrixTests(Epetra_CrsMatrix * L,  Epetra_MultiVector * bL,  Epetra_MultiVector * btL, Epetra_MultiVector * xexactL,
                      Epetra_CrsMatrix * U,  Epetra_MultiVector * bU,  Epetra_MultiVector * btU, Epetra_MultiVector * xexactU,
                      bool StaticProfile, bool verbose, bool summary) {

  if (L->NoDiagonal()) {
    bL->Update(1.0, *xexactL, 1.0); // Add contribution of a unit diagonal to bL
    btL->Update(1.0, *xexactL, 1.0); // Add contribution of a unit diagonal to btL
  }
  if (U->NoDiagonal()) {
    bU->Update(1.0, *xexactU, 1.0); // Add contribution of a unit diagonal to bU
    btU->Update(1.0, *xexactU, 1.0); // Add contribution of a unit diagonal to btU
  }

  Epetra_MultiVector z(*bL);
  Epetra_MultiVector r(*bL);
  Epetra_SerialDenseVector resvec(bL->NumVectors());

  //Timings
  Epetra_Flops flopcounter;
  L->SetFlopCounter(flopcounter);
  U->SetFlopCounter(flopcounter);
  Epetra_Time timer(L->Comm());
  std::string statdyn =        "dynamic";
  if (StaticProfile) statdyn = "static ";
```

```
  for (int j=0; j<4; j++) { // j = 0/2 is notrans, j = 1/3 is trans

    bool TransA = (j==1 || j==3);
    std::string contig = "without";
    if (j>1) contig =    "with   ";

    if (j==2) {
      L->OptimizeStorage();
      U->OptimizeStorage();
    }

    flopcounter.ResetFlops();
    timer.ResetStartTime();

    //10 lower solves
    bool Upper = false;
    bool UnitDiagonal = L->NoDiagonal();  // If no diagonal, then unit must be used
    Epetra_MultiVector * b = TransA ? btL : bL;  // solve with the appropriate b vector
    for( int i = 0; i < 10; ++i )
      L->Solve(Upper, TransA, UnitDiagonal, *b, z); // Solve Lz = bL or L'z = bLt

    double elapsed_time = timer.ElapsedTime();
    double total_flops = L->Flops();

    // Compute residual
    r.Update(-1.0, z, 1.0, *xexactL, 0.0); // r = bt - z
    r.Norm2(resvec.Values());

    if (resvec.NormInf()>0.000001) {
      cout << "resvec = " << resvec << endl;
```

```
      cout << "z = " << z << endl;
      cout << "xexactL = " << *xexactL << endl;
      cout << "r = " << r << endl;
    }

    if (verbose) cout << "ResNorm = " << resvec.NormInf() << ": ";
    double MFLOPs = total_flops/elapsed_time/1000000.0;
    if (verbose) cout << "Total MFLOPs for 10 " << " Lower solves " << statdyn << " Profile (Trans = " << TransA
                        << ")  and " << contig << " opt storage = " << MFLOPs << " (" << elapsed_time << " s)" <<endl;
    if (summary) {
      if (L->Comm().NumProc()==1) {
        if (TransA) cout << "TransLSv" << statdyn<< "Prof" << contig << "OptStor" << '\t';
        else cout << "NoTransLSv" << statdyn << "Prof" << contig << "OptStor" << '\t';
      }
      cout << MFLOPs << endl;
    }
    flopcounter.ResetFlops();
    timer.ResetStartTime();

    //10 upper solves
    Upper = true;
    UnitDiagonal = U->NoDiagonal();  // If no diagonal, then unit must be used
    b = TransA ? btU : bU;  // solve with the appropriate b vector
    for( int i = 0; i < 10; ++i )
      U->Solve(Upper, TransA, UnitDiagonal, *b, z); // Solve Lz = bL or L'z = bLt

    elapsed_time = timer.ElapsedTime();
```

```
    total_flops = U->Flops();

    // Compute residual
    r.Update(-1.0, z, 1.0, *xexactU, 0.0); // r = bt - z
    r.Norm2(resvec.Values());

    if (resvec.NormInf()>0.001) {
      cout << "U = " << *U << endl;
      //cout << "resvec = " << resvec << endl;
      cout << "z = " << z << endl;
      cout << "xexactU = " << *xexactU << endl;
      //cout << "r = " << r << endl;
      cout << "b = " << *b << endl;
    }


    if (verbose) cout << "ResNorm = " << resvec.NormInf() << ": ";
    MFLOPs = total_flops/elapsed_time/1000000.0;
    if (verbose) cout << "Total MFLOPs for 10 " << " Upper solves " << statdyn << " Profile (Trans = " << TransA
                        << ")  and " << contig << " opt storage = " << MFLOPs <<endl;
    if (summary) {
      if (L->Comm().NumProc()==1) {
        if (TransA) cout << "TransUSv" << statdyn<< "Prof" << contig << "OptStor" << '\t';
        else cout << "NoTransUSv" << statdyn << "Prof" << contig << "OptStor" << '\t';
      }
      cout << MFLOPs << endl;
    }
  }
  return;
}
```