Running Docker* Containers on Intel® Xeon Phi™ Processors

White Paper

March 2017

Revision 001

Document Number: 335644-001US

Contents

Tables

Revision History

Document Number	Revision Number	Description	Date
335644	001	• Initial release of the document.	March 2017

1 Introduction

1.1 Content and Scope

This document intends to present Intel's experiences with running Docker* containers on Intel® Xeon Phi™ processors. This paper will not describe all the details about Docker container building and execution. Readers can find this information in the Docker documentation [1].

Containers are becoming ever more popular as a mechanism of running applications and services on Linux*, and Docker remains among the most popular solutions. Its containers are pretty much independent from the underlying hardware. For HPC systems, they offer separation of their virtual environment, where an application is executed, from the native OS services (bare metal) in a way similar to the classical virtual machines but with less overhead and complexity.

In order to evaluate Docker containers on Intel Xeon Phi processors, Intel containerized and ran the *micperf* application, which is a set of benchmarks created to demonstrate the performance of Intel Xeon Phi processors and coprocessors. Section 2 describes *micperf* as well as the process of containerization and container execution, Section 3 contains performance results, Section 4 contains the summary, and Section 5 shows the configuration files.

1.2 References

[1] Docker Inc., "Docker Documentation," 25 January 2017. [Online]. Available: https://docs.docker.com/.

[2] Intel Corporation, "Intel® Xeon Phi™ Processor Software," [Online]. Available: https://software.intel.com/en-us/articles/xeon-phi-software. [Accessed 26 01 2017].

[3] Docker Inc., "Install Docker on Linux distributions," [Online]. Available: https://docs.docker.com/engine/installation/linux/.

[4] Intel Corporation, "Intel Parallel Studio XE 2017," [Online]. Available: https://software.intel.com/en-us/intel-parallel-studio-xe. [Accessed 26 01 2017].

[5] Intel Corporation, "Intel Math Kernel Library (Intel MKL)," [Online]. Available: https://software.intel.com/en-us/intel-mkl.

[6] Intel Corporation, "Intel MPI Library," [Online]. Available: https://software.intel.com/en-us/intel-mpi-library.

[7] Intel Corporation, "Intel® Xeon Phi™ Processor Software User's Guide," [Online]. Available: https://software.intel.com/en-us/articles/xeon-phi-software.

2 Running *micperf* as a Container

2.1 About *micperf*

There are many benchmarks which can be used to measure the performance of Intel Xeon Phi processors. *Micperf* is a software package designed to incorporate a variety of benchmarks into a simple user experience with a single interface for execution and a unified means of data inspection. It can be downloaded as a part of the Intel Xeon Phi Processor Software [2].

The *micperf* package targets a range of users including engineers interested in performance regression testing while implementing modifications to hardware, firmware, drivers, or operating system software. In addition, *micperf* also has other applications that can be employed by users and hardware manufacturers for demonstration and system verification purposes.

The *micprun* executable, the primary application in the *micperf* package, executes the following benchmarks on the Intel Xeon Phi Processor:

- *Linpack*
- *HPLinpack*
- *HPCG*
- *SGEMM*
- *DGEMM*
- *STREAM*

These benchmarks were carefully chosen to demonstrate performance in all of the major bottlenecks of the system. Their implementations use the Intel Xeon Phi processor's features, like its support for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) vector instructions or MCDRAM memory, to fully utilize the processor's capabilities and demonstrate its effectiveness. The processor excels at dense level basic linear algebra calculations, and the *Linpack*, *HPLinpack*, *HPCG*, *DGEMM*, and *SGEMM* benchmarks demonstrate these capabilities. All of these benchmarks compute with double precision floating point numbers, except *SGEMM*, which computes in single precision. The *STREAM* benchmark measures the bus bandwidth between the processor's main memory and the computational registers.

Micperf depends on several runtime libraries, and the results heavily depend on their exact version. To simplify the execution of *micperf* benchmarks and to ensure that the right dependencies are used, for the purpose of this white paper, a containerized version of this package was created that can be executed on any Linux distribution supported by Intel Xeon Phi processors without the need to install extra packages.

2.2 Prerequisites

2.2.1 Docker Runtime

To create and run containers, Docker must be installed. Some Linux distributions already include the Docker runtime software – for example Red Hat* Enterprise Linux* 7.2 used for experiments described here includes Docker version 1.9.1. The latest version of the software can be downloaded from the Docker servers. During the Intel experiments, Docker* version 1.12.5 was used. For instructions on how to update Docker, refer to its documentation [3].

2.2.2 Runtime Libraries

Micperf uses existing benchmark implementations and various runtime libraries optimized for best performance on Intel Xeon Phi processors. They are delivered as part of the Intel® Parallel Studio product [4]. However, installing the entire Intel Parallel Studio inside the *micperf* container is not required. Instead, only the following runtime libraries are needed:

- Intel® Math Kernel Library (Intel® MKL) L runtime (see [5] for downloading instruction)
- Intel Parallel Studio runtime (see [6] for installation)

2.2.3 Intel Xeon Phi Processor Software

Micperf is a part of Intel Xeon Phi Processor Software. This software can be downloaded from reference [2]. Separate tar archives are available for various OS distributions. Once the right tar file is downloaded, it should be unpacked. It contains multiple RPM files. All of the files are not needed in the container. For *micperf*, only *memkind* and *micperf* RPM files are needed: xppsl-memkind-<version-and-build>,x86_64.rpm xppsl-micperf-<version-and-build>,x86_64.rpm. Those files should be copied to the directory where the container is built.

2.3 Building *micperf* Container

Building a *micperf* container can be performed on any Linux-based node, not necessarily equipped with an Intel Xeon Phi processor.

To create a *micperf* container, a new directory was created where the Intel MKL runtime distribution file was copied, the Intel® MPI distribution file, and the OpenMP* library. Then, a typical Dockerfile* configuration file was created. The content of this file is described below, while the complete file is shown in Section 5.1.

The Dockerfile starts with information about the source container (in the Intel case, it is the CentOS* 7.2 container):

```
FROM centos:7.2.1511
```

An optional instruction informs about the container's maintainer:

```
MAINTAINER <maintainer>@<domain>
```

If the container is built in a corporate network environment, proxy addresses to the access public repositories may be required:

```
ENV http_proxy="http://<proxy.address:port>/"
ENV https_proxy="https:// <proxy.address:port>/"
```

Some environment variables used within the container are also specified. *MKLROOT* indicates where the Intel MKL runtime is installed. *LD_LIBRARY_PATH* is modified to include the OpenMP runtime.

```
ENV MKLROOT="/opt/mkl"
ENV
LD_LIBRARY_PATH="/opt/intel/psxe_runtime/linux/compiler/lib/intel64
_lin:${LD_LIBRARY_PATH}"
```

Next, specify the filenames for the installation versions of the Intel MKL runtime. The exact names should be changed when new versions of those libraries are used.

```
# MKL runtime filename
ARG MKL_RUNTIME_FILENAME="l_mklb_p_2017.1.013"
```

The filenames of the latest versions of the *micperf* RPMs must be specified as well:

```
# Memkind and micperf RPM names
ARG MEMKIND="xppsl-memkind-1.5.0-4036.x86_64.rpm"
ARG MICPERF="xppsl-micperf-1.5.0-4036.x86_64.rpm"
```

The Intel MKL runtime is installed by unpacking the distribution tarball to the desired location within the container image.

```
# install mkl runtime
ADD ${MKL_RUNTIME_FILENAME}.tgz ${MKLROOT}
```

Once all dependencies are installed, install *micperf* into the newly created container image.

```
# Install memkind from local RPM
ADD ${MEMKIND} .
RUN yum install -y ./${MEMKIND} && \
     rm -f ${MEMKIND}
```

```
# Install micperf from the local RPM
ADD ${MICPERF} .
RUN  yum install -y ./${MICPERF} && \
      rm -f  ./${MICPERF}
```

Finally, the Intel Parallel Studio script is run to configure environmental variables and to run the command shell:

```
# Set Intel MPI enviroment variables and run command shell
CMD source /opt/intel/psxe_runtime/linux/bin/psxevars.sh &&
/bin/bash
```

Once the Dockerfile is defined, build the container. The container creation requires root privileges.

```
$ sudo docker build -t <user>/<project>:micper-mkl-1-132 .
```

As a result of the above operation, a container is labeled as *micperf:01*. That container can be pushed to the container repository:

```
$ sudo docker push <user>/<project>:micper-mkl-1-132
```

2.4 Running *micperf* Container

To obtain meaningful performance results, the *micperf* container should be executed on the Intel Xeon Phi processor. Details on how to run the *micperf* container are presented below.

2.4.1 Security Configuration

Docker version 1.9.x, delivered with Red Hat Enterprise Linux 7.3, does not have any security restrictions that affects the *micperf* container execution. However, extra security features were added starting from Docker version 1.10. Therefore, on the newest versions of Docker, execution of certain system calls within the container is blocked.

Micperf uses two system calls, *mbind()* and *set_mempolicy()*, that are blocked by the default configuration of Docker 1.12.x. Those system calls are needed to enforce the memory allocation from MCDRAM instead of DDR.

To change the security configuration, the default security configuration file was downloaded from Docker github: raw.githubusercontent.com/docker/docker/v1.12.3/profiles/seccomp/default.json.

Based on that file, a new custom security configuration file was created called *micperf.json* and added the following:

```
{
        "name": "mbind",
        "action": "SCMP_ACT_ALLOW",
        "args": []
},
{
        "name": "set_mempolicy",
        "action": "SCMP_ACT_ALLOW",
        "args": []
},
```

This new policy allows the use of *mbind()* and *set_mempolicy()* system calls from the container.

2.4.2 Running the Container

So far the *micperf* container was prepared and the optional security policy configuration. Now the *micperf* container can be executed on the Intel Xeon Phi processor. The latest version of the *micperf* container may be uploaded first. This is optional, since the run command will upload the container if a specified name and version is not available.

```
$ docker pull <user>/<project>:micper-mkl-1-132
```

Now the container can be run. The simplest way is to run it in the interactive mode and run the *micprun* command within the container. First, the iterative container is started:

```
$ docker run -it <user>/<project>:micper-mkl-1-132
```

If Docker version 1.10 or newer is installed, a custom security policy (described in the previous section) must be used:

```
$ docker run --security-opt seccomp=micperf.json -it \
<user>/<project>:micper-mkl-1-132
```

Once the container prompt is shown, an entire set of all available benchmarks can be run:

```
root@98df59cd3def# micprun
```

Optionally, you can select a benchmark, for example:

```
root@98df59cd3def# micprun -k dgemm
```

Micperf tries to detect coprocessors using *dmidecode* and *lspci* commands that are intentionally not included in the container. Warning messages are generated, but they can be ignored.

```
root@98df59cd3def# micprun -k linpack
    WARNING: Could not find dmidecode.
```

```
WARNING: Could not find lspci in path, and it is not located in
/sbin/lspci.
```

3 Performance in the Container

3.1 Configuration

Performance tests were executed on a system with one Intel Xeon Phi processor 7250 at 1.40 GHz (68 cores) with hyper threading enabled, 48 GB DDR and 16 GB MCDRAM (flat mode, quadrant).

The installed OS (bare metal) software: Red Hat Enterprise Linux release 7.2 (Maipo), XPPSL 1.5.0, kernel 3.10.0-327.36.3.el7.xppsl_1.5.0.3509.x86_64.

3.2 Performance Results

The *micperf* benchmarks were executed in the container and on the bare metal system. The goal of the performance test was to compare the results on both configurations. The performance results of benchmarks such as *SGEMM* and *DGEMM* may differ slightly between runs. As presented in Table 1, the difference may be greater than 1%.

Table 1 Difference Between Multiple Runs on Bare Metal

Benchmark	Diff Between Runs (Gflops)	Relative Difference %
sgemm	45	1.03%
dgemm	158	8.01%

Table 2 compares the average benchmark results obtained in the bare metal (Native OS) and inside the container. It shows that the performance impact of the containers can be ignored, as the difference between bare metal and container average results are smaller than between multiple executions of the same benchmark on the bare metal node.

Table 2 Difference Between Bare Metal and Container Results

Benchmark	Difference Between Bare Metal and Container	Difference in %
sgemm [GFlops]	-5	-0.11%
dgemm [GFlops]	4	0.21%
stream [GB/s]	0	0.00%
HPLinpack [GFlops]	0	0.00%
linpack [GFlops]	4	0.21%
HPCG [GFlops]	0.11	0.23%

4 Summary

This paper presented how to create and run a Docker container with the *micperf* benchmark on the Intel Xeon Phi processor. The performance results obtained within the container are comparable to results obtained on the native operating system (bare metal). It proves that important hardware features specific for Intel Xeon Phi processor, like new Intel AVX-512 instructions or MCDRAM memory running in the flat mode, are available in the container. The modifications to the default security policy, which were required to use MCDRAM, were very limited and did not affect overall system security.

5 Sources

5.1 Dockerfile* Configuration File

```
FROM centos:7.2.1511
MAINTAINER <maintainer>@<domain>

ENV http_proxy="http://<your_proxy>:<your_proxy_port>/"
ENV https_proxy=<your_proxy>:<your_proxy_port>/
ENV MKLROOT="/opt/mkl"
ENV
LD_LIBRARY_PATH="/opt/intel/psxe_runtime/linux/compiler/lib/intel64
_lin:${LD_LIBRARY_PATH}"

# MKL runtime filename
ARG MKL_RUNTIME_FILENAME="l_mklb_p_2017.1.132"

# Memkind and micperf RPM names
ARG MEMKIND="xppsl-memkind-1.5.0-4036.x86_64.rpm"
ARG MICPERF="xppsl-micperf-1.5.0-4036.x86_64.rpm"

# install mkl runtime + benchmarks
ADD ${MKL_RUNTIME_FILENAME}.tgz ${MKLROOT}

# Install Intel Paralell Studio runtime from a repository
# This will install MPI and OpenMP runtimes
RUN rpm --import http://yum.repos.intel.com/2017/setup/RPM-GPG-KEY-
intel-psxe-runtime-2017 && \
    rpm -Uhv http://yum.repos.intel.com/2017/setup/intel-psxe-
runtime-2017-reposetup-1-0.noarch.rpm && \
    yum -y install intel-mpi-runtime-64 && \
    yum -y install intel-openmp-runtime-64


# Install memkind from local RPM
ADD ${MEMKIND} .
RUN yum install -y ./${MEMKIND} && \
     rm -f ${MEMKIND}

# Install micperf from the local RPM
ADD ${MICPERF} .
RUN  yum install -y ./${MICPERF} && \
      rm -f  ./${MICPERF}

# Set  Intel MPI enviroment variables and run command shell
CMD source /opt/intel/psxe_runtime/linux/bin/psxevars.sh && /bin/sh
```