



Intel® Xeon Phi™ Product Family

Unified Interface for Benchmark Tools (micperf)

User's Guide

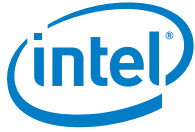
August 2017

Copyright © 2017 Intel Corporation

All Rights Reserved

Revision: 1.17

World Wide Web: <http://www.intel.com>



Legal Disclaimer

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: <http://www.intel.com/design/literature.htm>

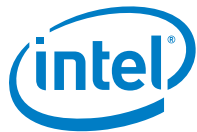
*Other names and brands may be claimed as the property of others.

Copyright © 2017, Intel Corporation. All rights reserved.



Table of contents

1	Introduction	6
1.1	Organization	7
1.2	Conventions and symbols	7
2	Novice user: benchmark execution	8
2.1	Benchmark selection	9
2.2	Memory selection	10
2.3	Parameter category selection.....	10
2.4	Explicit benchmark parameter specification	11
2.5	Verbosity selection	12
3	Expert user: generating collateral	14
3.1	Creating a <i>micprun_stats</i> file	14
3.2	Data inspection with micpinfo	14
3.3	Data inspection with micpprint.....	14
3.4	Data inspection with micpcsv.....	15
	3.4.1 Summary format	15
	3.4.2 Long format.....	15
	3.4.3 Short format	16
3.5	Data inspection with micpplot.....	16
4	Hardware tester: executing regression tests	18
5	Systems engineer: creating reference data	20
6	Tinkerer: benchmark modifications	21
7	Test developer: integrating new benchmarks	23
8	References	24

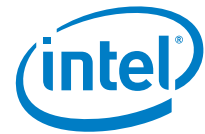


Figures

Figure 1 Example Output from micpplot	17
---	----

Tables

Table 1 Benchmarks	9
Table 2 Parameter Categories	11
Table 3 Verbosity Output.....	12
Table 4 Output from micpcsv as displayed by Microsoft* Excel*	15
Table 5 micprun error codes	18



Revision History

Date	Revision	Description
August 2017	1.17	Fixed multiple sections to reflect current state of software.
July 2017	1.16	Remove obsolete contents and unify sections to be applicable for all existing platforms supported by the the micperf.
April 2016	1.15	Added information about the Fio benchmark.
March 2016	1.14	Added information about the Deepbench suite.
September 2016	1.13	Added information on the micperf source RPM and how to rebuild it, Sections 2 and 6.
June 2016	1.12	Added new error code in Table 5 Added note to Section 2.2 Memory Selection
June 2016	1.11	Addressed internal feedback
May 2016	1.1	Added HPCG documentation
December 2015	1.0	Addressed internal feedback Added HPLinpack documentation.
November 2015	0.51	Adding 2.2 Memory Selection
June 2015	0.5	Initial release.



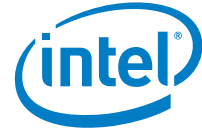
1 Introduction

Numerous benchmarking application exist which can be used to measure performance of systems with Intel® Xeon Phi™ Processors. These benchmarks are developed by various teams, and consequently have diverse user interfaces, methods of execution and output. The *micperf* package is 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. The user interface to *micperf* consists of five executable files: one for execution of benchmarks (*micprun*), and four that interpret its output. These executables are all well documented with standard UNIX* style command line interfaces. The results can be displayed as professional quality plots, human readable text or comma separated values that can be easily imported into a variety of other tools. Results of different runs can be easily combined and compared. To support tracking of the performance impact of changes to the system configuration, *micperf* stores detailed hardware and software configuration information along with the performance data. *Micperf* also serves as a harness to integrate a variety of benchmarks into automated testing for performance regressions.

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 to these highly technical customers, there are also application users and hardware manufacturers who may choose to use the *micperf* software for demonstration and system verification purposes. This paper provides a description of the *micperf* user experience for a range of use cases. The simplest use cases are outlined in [Chapter 2](#). Further sections identify more sophisticated use cases for collateral generation, regression testing, benchmark modification and extending the *micperf* package to include new benchmarks.

The *micprun* executable, the primary application in the *micperf* package, executes several benchmarks: *MKL* [12] *SMP Linpack* [2], *MKL HPLinpack* [2], *MKL* [12] *HPCG*, *MKL SGEMM*, *MKL* [12] *DGEMM*, *MKL* [12] *IGEMM*, *STREAM* [10] [11], *Deepbench convolutions* (*libxsmm_layer*, *std_conv_bench*) [15] and *fio* [16]. These benchmarks were carefully chosen to demonstrate performance in all of the major bottlenecks in the system. The Vector and Floating Point Unit (VFU) in the processor excel at dense level three basic linear algebra [9] calculations, and the *Linpack*, *HPLinpack*, *HPCG*, *DGEMM*, and *SGEMM* benchmarks demonstrate these capabilities, *Deepbench* and *MKL IGEMM* benchmarks exercise operations that are used in deep neural networks. The *fio* benchmark assess performance of I/O operations.

SMP Linpack, *HPLinpack*, *HPCG* and *DGEMM* compute with double precision floating point numbers, *SGEMM* computes in single precision and *IGEMM* focuses on integer matrix matrix multiplication. The *STREAM* benchmark measures the bus bandwidth between the processor's main memory and the computational registers. *Deepbench* tests performance of convolution operation performed using the Intel® *MKL-DNN* and *libxsmm* libraries. *Fio* checks the performance of drive read operations.



1.1 Organization

This document is organized into chapters that describe a different user's experience with the *micperf* package.

Chapter 2: Focuses on the end user of the processor and shows *micperf's* basic features.

Chapter 3: Discusses the experience of the advanced user who generates professional quality content regarding the performance of a computing platform.

Chapter 4: Describes the use case of the hardware validator performing performance regression tests against Intel® Validation measurements.

Chapter 5: Details how a systems engineer may use *micperf* to run performance regression tests after introducing modifications to their system.

Chapter 6: Describes how one may inspect or introduce modifications to benchmarks' source code.

Chapter 7: Describes a benchmark developer's experience integrating new benchmarks into the *micperf* infrastructure.

1.2 Conventions and symbols

<code>This type style</code>	Indicates an element of syntax, reserved word, keyword, filename, computer output, command, or part of a program example. The text appears in lowercase unless uppercase is significant.
<code>[host]#</code>	Indicates that a command requires root privileges.
<code>[host]\$</code>	Indicates that a command can be executed without root privileges.
This type style	Used to highlight the elements of a graphical user interface such as buttons and menu names.
<i>This type style</i>	Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder. Also used to indicate new terms, URLs, email addresses, filenames, and file extensions.
<code>[items]</code>	Indicates that the items enclosed in brackets are optional.
<code>{ item item }</code>	Indicates to select only one of the items listed between braces. A vertical bar () separates the items.
<code>... (ellipses)</code>	Indicates that you can repeat the preceding item.



2 Novice user: benchmark execution

A user who has just installed an Intel® Xeon Phi™ Processor in their system, or has just obtained access to a system that already has the processor installed, may be interested in running *micperf* to see the performance boost the processor can provide, or they would like to demonstrate the capabilities of this new piece of hardware. *Micperf* can be used to demonstrate the processor's functionality and performance numbers which may serve as a motivation to learn more on how to use the product.

The *micperf* package is distributed as an *RPM* [3] or *DEB* file and is part of the Intel® Xeon Phi™ Processor Software. The *micperf* executables are distributed as a binary *RPM* or *DEB* files, whereas source code is distributed in source *RPM* files.

After installation all documentation can be found under a standard documentation path specific to target distribution. Usually documentation is stored under one of these directories:

```
/usr/share/doc/micperf-<version>/  
/usr/share/doc/packages/micperf/  
/usr/share/doc/
```

Documentation consists of the following files:

- The *README.txt* file describes what was installed, as well as basic usage instructions and refers other sources of documentation.
- The *CHANGES.txt* file contains the change log.
- The *INSTALL.txt* file contains installation instructions.
- The *micperf_faqs.txt* file contains answers to questions frequently asked by users

The *micperf* installation procedure conforms to Linux* conventions. For the purpose of transparency the source code is distributed in source *RPM* files and can be inspected, modified and re-build. Refer to [Chapter 6](#) for more information.

The first step to using *micperf* is running the *micprun* executable file and reading the help documentation that the application provides. Use the command below to access the help documentation.

```
[host]$ micprun --help
```

The most basic execution command of *micperf* is:

```
[host]$ micprun -k <kernel_name> -c <category>
```

where:

<kernel_name> - name of the kernel to be executed, refer to section **Error! Reference source not found.** for more information



<category> - category of parameters, refer to section **Error! Reference source not found.** for more information

When executed, above command will cause *micperf* to run selected benchmark with optimal set of parameter.

By default *micperf* output has following structure:

```
***** RUN *****
Running <kernel_name> <execution_paramteres>
Please be patient, this may take a few minutes...

[ ENVIRONMENT ] <environmental variable>=<value> ...
[ CMD LINE    ] <command line used for execution>
[ DESCRIPTION ] <human readable description of run>
[ PERFORMANCE] <score of kernel>

[ INFO        ] Kernels output has been saved to file
                  <output_file_name.log>
```

Micperf prints only short human readable summary about kernel execution and its final score. Whole outputs generated by kernels are stored in file which name can be found in section INFO of *micperf* output.

The amount of generated output can be controlled with the verbosity option, -v. For more fine-grained control of the output see [Chapter 3](#).

2.1 Benchmark selection

One of the simplest modifications to the standard *micprun* execution is to select the benchmarks that will be run (see [Table 1](#) to review available benchmarks). Benchmark selection is done with the -k command line option. If the -k option is not specified, all benchmarks are executed. To list the names of all available benchmarks, execute the *micprun -k help* command. More than one benchmark can be selected using a colon-separated list, for example:

```
[host]# micprun -k linpack:dgemm:stream
```

The default option to run all workloads can be explicitly specified with:

```
[host]# micprun -k all
```

Table 1 Benchmarks

Benchmark	CLI Name	Target Operations	Component
MKL DGEMM ¹	dgemm	Double precision floating point	VFU
MKL SGEMM ¹	sgemm	Single precision floating point	VFU
MKL IgemM ¹	igemmm	Integer	ALU



MKL SMP Linpack ^{2,3}	linpack	Double Precision Floating Point	VFU
STREAM	stream	Round-trip memory to registers	MCDRAM, DDR and caches
HPLinpack ^{2,3}	hpllinpack	Double precision floating point	VFU
HPCG ^{2,3}	hpcg	Double precision floating point	VFU
Deepbench convolutions	libxsmm_conv mkl_conv	Single precision floating point	VFU
Fio ³	fio	Drive to CPU data transfer	SATA bus

- 1 For the Intel® Xeon Phi™ processor *micperf* provides an MCDRAM and a DDR version of this workload. See [Section 2.2](#) for further details on memory selection.
- 2 Requires Intel® MPI Libraries, please refer to the *INSTALL.txt* file for details.
- 3 Binary is not provided in *micperf* package, please refer to the *INSTALL.txt* file for details.

2.2 Memory selection

Integrated MCDRAM memory is one of the key features of the Intel® Xeon Phi™ Processor. Depending on the application, the use of MCDRAM memory can help improve performance.

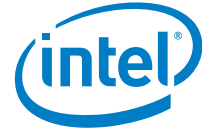
By default, where possible kernels are being executed in MCDRAM memory by utilizing memkind library or numactl tool. However it is possible to force usage of DDR memory by adding -D option. In this case MCDRAM memory availability check is not performed. Please note that this option only works in flat memory mode. In cache mode MCDRAM is used transparently.

To learn more about the memkind library install the xppsl-memkind-devel package and run the command below.

```
[host]$ man memkind
```

2.3 Parameter category selection

Another basic modification to the standard *micprun* execution is to select the category of parameters that will be passed to the benchmark executables by using the -c option. The names of categories are purposefully abstract, as they are intended to apply to all benchmarks included in the *micperf* package and also to any benchmarks that are added as extensions to *micperf*. The three categories that all benchmarks, including extensions, implement in some form are *optimal*, *scaling*, and *test*. Each benchmark interprets the meaning of these categories differently, but typically *optimal* sets the options that give nearly optimal performance, *scaling* runs at least one parameter through a range of values, and



test performs a self-check test where *micprun* gives a non-zero return code if the test fails.

For some benchmarks, the exact parameters to achieve optimal performance depend opaquely on the hardware or software configuration. For these benchmarks, several parameter sets may be executed, but if a category begins with the string *optimal* then only the best performing parameters in the set executed are included in *micperf*'s summary reporting. This feature allows a benchmark to define a parameter space to search for optimal performance.

The *scaling* category is especially useful for plotting performance as a function of an input parameter to the benchmark. There are some benchmarks that can be scaled in more than one parameter, and for these there are multiple scaling categories. For example, the *DGEMM* benchmark's *scaling* category runs through matrix sizes to show data scaling while using all available cores, but strong core scaling [8] is also implemented as the *scaling_core* category which runs through a range of core counts while keeping the matrix size constant. If a category is selected along with a benchmark that does not implement the category then an error is raised. All of the benchmarks included in the *micperf* package implement the categories defined in Table 2.

Table 2 Parameter Categories

Category	Definition
optimal	Parameters that yield nearly optimal performance.
scaling	Run one parameter, typically data size, through a range of values.
test	Executes a self-check.
optimal_quick	Parameters that yield nearly optimal performance, but with a short run time.
scaling_quick	A subset of the scaling category: excludes long run time parameters.

2.4 Explicit benchmark parameter specification

The parameter categories serve two functions: one is to abstract the specifics of the benchmark parameters from the user, and the other is to define a common execution purpose for all benchmarks so that they can be run together. There are times, however, when the user wants to have fine-grained control over the parameters that are passed to a particular benchmark. This can be achieved using the *-p* option to *micprun*. Note that the *-p* and *-c* option cannot be used together.

The benchmarks applications themselves have a number of different ways of obtaining parameters from the user: long form command line options, short form command line options, positionally defined command line arguments, environment variables, or a parameter file. To distill these into a uniform interface, *micprun* takes only long form command line options and maps this specification to the method that each particular benchmark application uses for parameter input.



Novice user: benchmark execution

To print the long form command line options for a benchmark and the default values run: `micprun -k <benchmark_name> -p help`. Afterwards, pass the obtained parameters and their values to the `-p` option.

In the example below possible parameters of the `fio` benchmark are obtained and then the benchmark with custom values is executed:

```
[host]$ micprun -k fio -p help
Parameter help for kernel fio <param:default>:
<size:10MB> <numjobs:10>
[host]$ micprun -k fio -p "--size 20MB --numjobs 12"
```

Note: The default values are the parameters for the optimal category for all of the benchmarks included in the *micperf* package.

2.5 Verbosity selection

By default, the verbosity option (the `-v` flag to *micprun*) gives coarse control of the output. This behavior can be modified by providing the option with an integer from 0 (least verbose, default value) to 3 (most verbose). The details about the output associated with each level can be found in the *micprun --help* documentation and in Table 4 below.

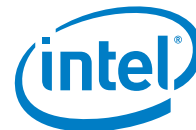
In [Table 3](#), the **Summary** column refers to a concluding section in the standard output that reprints the performance data collected as it is displayed by *micpprint*. The **Joint Plot** column refers to an attempt to create an additional plot that combines the data from all of the benchmarks. This aggregated plot will only be created in the case where all the selected benchmarks are plotted with the same X and Y axes. The infrastructure supports benchmarks that collect ancillary data that is not displayed by default, but can be displayed at the user's requests. The primary data is considered *rolled up*, and the ancillary data is included when all data is requested.

Table 3 Verbosity Output

CLI Options	Summary	Plots	Joint Plot	CSV Rolled	CSV All
-v0	No	No	No	No	No
-v0 -o	No	No	No	No	No
-v1	Yes	No	No	No	No
-v1 -o	Yes	No	No	Yes	No
-v2	Yes	Yes	No	No	No
-v2 -o	Yes	Yes	No	Yes	Yes
-v3	Yes	Yes	Yes	No	No
-v3 -o	Yes	Yes	Yes	Yes	Yes

None of the benchmarks included in the *micperf* package create ancillary data, and consequently *CSV Rolled* and *CSV All* refer to CSV files that have the same content for all included benchmarks.

Novice user: benchmark execution



§



3 Expert user: generating collateral

Some data inspection beyond the benchmark log is available by setting the verbosity level (the `-v` option) of *micprun* above zero. For advanced usage it is recommended that the `-v` option is avoided. More fine-grained control over the output is obtained by using the file created with the `-o` option to *micprun* in conjunction with the *micperf* helper applications: *micpprint*, *micplot*, *micpcsv*, and *micpinfo*.

3.1 Creating a *micprun_stats* file

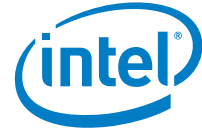
Understanding how to use the `-o` and `-t` options to *micprun* is the first step to data inspection with the helper applications. The `-o` option specifies an output directory where output files from *micprun* are created. Setting the verbosity to zero (the default when `-m` is not given) results in only one file being created in the output directory. It will be named *micp_run_stats_TAG.pkl* where TAG is the tag associated with the run. This tag has a default value that describes some of the characteristics of the system under test, but it can be set explicitly with the `-t` option. The tag will be displayed in a number of places in the output, and should be chosen to be descriptive, as well as different from any tags given to runs that are to be compared.

3.2 Data inspection with *micpinfo*

A wealth of system information is included in a *micprun_stats* file, and *micpinfo* is the helper application that is used for inspection of this data. The bundling of system information with the performance data allows for the correlation of performance with system configuration and provides a mechanism for comparison of the configurations of two systems that were benchmarked. The system configuration data is collected by running a set of system commands and the standard output of these commands is recorded. The *micpinfo* application allows the user to inspect the log of any subset of commands using the `--app` option. By default, the application runs the command set on the current system, and if a *micprun_stats* file is passed on the command line then the log produced reflects what was recorded on the system just prior to the execution of the benchmarks when the file was created.

3.3 Data inspection with *micpprint*

The benchmark data recorded in a *micprun_stats* file can be displayed in human readable form with the *micpprint* helper application. The output is organized by the benchmark and offload method. Multiple *micprun_stats* files can be passed to the command line of *micpprint*, and the output will be interleaved allowing for easy comparison of the performance recorded in different *micprun_stats* files. Each section of output is preceded by the tag associated with the file.



Note: The first file listed on the *micpprint* command line determines the set of benchmark and offload methods that will be displayed. For this reason, when comparing a targeted test to a comprehensive reference file, the targeted test file should be listed first. This is also true for the other helper applications that display performance data: *micpcsv* and *micplot*.

3.4 Data inspection with micpcsv

The *micpcsv* helper application is used to create data that can be easily parsed by a variety of other applications such as spreadsheets and databases. Since comma separated value output is an unstructured format, there are several styles of output formatting from *micpcsv* which are controlled with command line options.

3.4.1 Summary format

Running *micpcsv* with no arguments produces a single summary table. The data in this table is derived by selecting the highest performance runs for each benchmark. Obtained data is then displayed in a summary table that includes performance information and the value of the independent variable in the scaling parameter category. If the *-o <output directory>* option is given, *micpcsv* creates an output file named *summary.csv* in the output directory. An example of a summary table is presented below.

Table 4 Output from micpcsv as displayed by Microsoft* Excel*

	SGEMM pragma/native (GF/s)	DGEMM pragma/native (GF/s)	SMP Linpack (GF/s)	STREAM (Triad) (GB/s)
B1QS-5510P	1448.24 / 1668.51	652.43 / 794.54	721.03	171.41
Parameters	15360 / 15360	10240 / 7680	26624	58
B1QS-7110P	1550.54 / 1782.89	693.53 / 842.13	755.17	175.22
Parameters	15360	10240 / 7680	26624	60

3.4.2 Long format

If a *micprun.stats* file is passed to *micpcsv* but the *-o* flag is not provided, then the standard output is composed of blocks of tables which are separated by one or two empty lines. Two tables separated by just one empty line are related: the first is the identifying table, and the other is the run record table. The identifying table has a header row containing the *KERNEL*, *OFFLOAD* and *TAG* cells and an additional single row that describes the benchmark, offload method and tag. These three identifiers apply to all of the run records in the table that follow the identifying table. The run record table has a first column of run descriptions. This is followed by columns containing values of all passed parameters (one column per parameter). The last column in these tables provides the performance metric.

For the optimal parameter category, the summary table for a given kernel will only have two rows: one for the header and one for the values. When specifying the optimal parameter category to *micprun*, the short format output (see the section below) may be more useful. When specifying the scaling run parameter



category to *micprun*, the tables will have multiple rows of values: one for each execution of the benchmark.

If a *micprun_stats* file is passed to *micpcsv* and the *-o* flag is passed then each of the paired tables described above are written to a separate file. The name of each file is derived from the identifying table, and the entirety of the contents of each file is the run record table. This is much more useful than the dump to standard output since each file has a fixed column width.

3.4.3 Short format

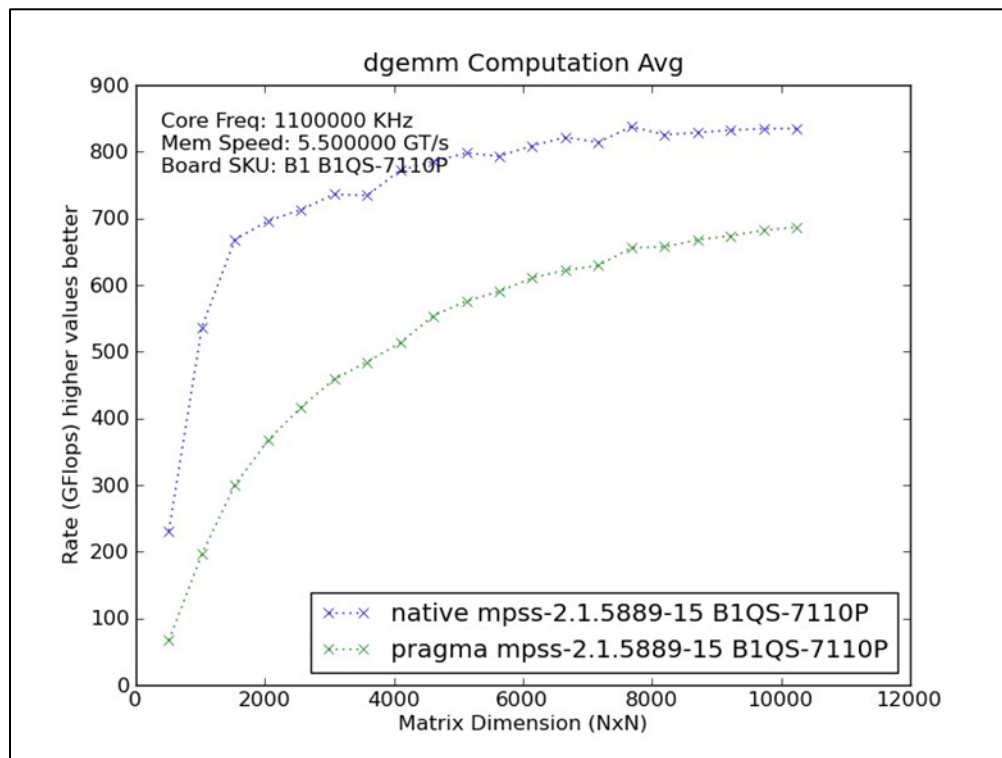
The short format is selected by passing the *-s* flag to *micpcsv*. This flag has no impact on the summary format (for instance, when no *micprun_stats* files are given). The short format is advantageous over the long form in that a single table with a fixed column width is produced, however, all of the parameters are stored in a single column and the tags are not included. If the *-o* flag is not given then the table is written to standard output, and if it is given then a single file named *short_form.csv* is produced in the output directory.

3.5 Data inspection with micpplot

The *micperf* package makes use of the *matplotlib* [6] Python* visualization library if it is installed. The *matplotlib* package is not required to be installed on the system being benchmarked, and the *micprun_stats* file can be transferred to another machine for visualization purposes. The *micpplot* application is used to visualize the data from runs using scaling parameter categories, and especially for comparison of different *micprun_stats* files. The application will plot the results from each benchmark on a different graph, and will combine different methods of offload of the same benchmark onto a single graph. As with the other helper applications that take multiple *micprun_stats* files, the first file listed determines the benchmarks and offload methods that will be plotted. In the case where all of the benchmarks that are plotted have the same x and y axis then *micpplot* will produce a final image that plots all of the benchmarks onto a single graph. By default the *micpplot* application generates interactive plots that the user can re-size and save in the Portable Network Graphics (PNG) [5] format with a user specified file name. As the user closes each plotting window, the next one in the sequence appears. The *micpplot* application also accepts the *-o* option which creates PNG files in the specified directory in a non-interactive mode. Figure 2 shows an example of the output from *micpplot*.



Figure 1 Example Output from micpplot



§



4 Hardware tester: executing regression tests

The *micperf* package is distributed with a set of reference *micprun_stats* files. These files are accessible with the *-R* command line option with *micprun* and all of the *micperf* helper applications. When an application is called with *-R help* it will print the list of tags that are available in the installed location. The user can specify one of the tags from the list as the argument to the *-R* option. This feature is especially useful for performing regression tests with *micprun*. The data stored in the distributed reference files was measured by Intel® Validation and can be used as a reference mark to determine if the Intel® Xeon Phi™ processor is performing up to specification. To execute this test regression the *-m* option can be used to specify the acceptable margin allowed from the reference measurement given as a percentage. If the performance measured by this run is lower than the reference by more than the fractional margin, an error message is printed and the return code from *micprun* is 88, return codes are documented in the *micprun --help* output and in [Table 5](#) below.

Table 5 micprun error codes

Error Code	Meaning
0	No error
1	General exception
2	Command line parse error
3	I/O error
87	Permission error
88	Performance regression error
90	Key/index/name lookup error
91	Missing dependencies error
126	Missing executable error
127	Missing shared object libraries error

The distributed data uses the default tag which contains the product stock keeping unit (SKU), the Intel® Xeon Phi™ processor software version and the parameter category.

To check available reference files execute below command:

```
[host]$ micprun -R help
```

Then basic regression test can be performed by executing following command:

```
[host]$ micprun -k <kernel_name> -t <output_files_postfix> -o  
<output_directory> -m <regression_margin_in_percent> -R  
<reference_file_name>
```



Above command produces *micperf* stats file *micp_run_stats_<memory>_<postfix>.pkl* and its csv formatted duplicate *micp_run_stats_<memory>_<postfix>.csv*.

Examples below show how a *micperf* stats file can be used to determine possible problems in system configuration. Generated *micperf* stat file contains detailed information about the execution environment. The *micpinfo* application extracts this information which then can be compared with the *diff* tool:

```
[host]$ micpinfo -R <reference_file_name> > ref_info.log
[host]$ micpinfo <micperf_stat_file> > test_info.log
[host]$ diff ref_info.log test_info.log > diff_info.log
```

This will highlight any differences between the test environment (using which *micperf_stat_file* has been generated) and reference configuration (using which *micperf* provided reference files has been generated)

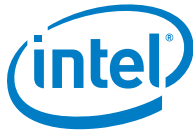
Analogically, the *micpprint* utility can be used to examine the performance data and compare it to the reference results:

```
[host]$ micpprint -R <reference_file_name> > ref_perf.log
[host]$ micpprint <micperf_stat_file> > test_perf.log
[host]$ diff ref_perf.log test_perf.log > diff_perf.log
```

The *micppplot* application can be used to inspect the regression. For example, the command below can be issued to plot the test lines against reference lines.

```
[host]$ micppplot -R <reference_file_name> <micperf_stat_file>
```

§



5 Systems engineer: creating reference data

[Chapter 4](#) showed how to use the reference data shipped with *micperf* to perform regression tests. The way the user accesses the reference data in this case is through selecting a tag with the *-R* flag, but all of the examples in Chapter 4 can be executed with user created reference files as well. User-created reference files can be used to determine how performance changes as an element of the system under test, as opposed to complete system comparison against Intel validation measurements. A simple example of this would be BIOS settings, and this example can be easily extended to apply to changes to the operating system software.

For example, the user would like to determine if changing a power management BIOS setting on the host system impacts the performance. To test this, the user generates a *micprun* stats file on the system with the original BIOS setting using the *-o* option to *micprun*. They then reboot the system, change the BIOS setting and run benchmarks against the *micprun* stats file they have created using the *-r* and *-m* options to *micprun*. In this way, the user can isolate how specific changes in a system impact performance. The BIOS setting is just one example of a change to a platform. This change could be a software modification to the kernel or driver, or any change in the design of a platform that includes a processor or a coprocessor. Any engineer involved in the design of the platform that wants to elucidate the performance impact of a design change can use the tool in this way.

§



6 *Tinkerer: benchmark modifications*

Transparency is an important consideration in the distribution of benchmarking software and one of the most important aspects of this transparency is using open source benchmarks that are standards in the industry. All benchmarks included in the *micperf* package are distributed with source code and make files for compilation. Note that some benchmarks are supported but the binaries have to be provided by external package. In those cases source can, but does not have to be provided by the author and the usage model depends on external license.

The source files are included in the corresponding source RPMs that can be found in Intel® Xeon Phi™ Processor Software package. Source RPMs allow users to inspect and modify source code they contain and re-build them to generate a new binary RPM. An example of how to recompile the *micperf* source RPM is presented below. For further details on how to apply a patch and re-build the source RPM files please refer to [14] and [15].

To inspect the source code install the source RPM and change to the appropriate directory:

```
[host]$ rpm --ihv <source_package_name>.src.rpm
[host]$ cd ~/rpmbuild/SOURCES/
[host]$ tar -xf <product>-micperf-<version>.tar.gz
[host]$ cd <product>-micperf-<version>
```

In order to re-build the source RPM, the Intel® Composer XE package must be installed, and the *compilervars* script must be sourced using the following bash command:

```
[host]$ source \
/PATH/TO/COMPOSER_XE_INSTALL_DIR/bin/compilervars.sh intel64
```

Use the command below if you use C Shell:

```
[host]$ source \
/PATH/TO/COMPOSER_XE_INSTALL_DIR/bin/compilervars.csh intel64
```

Run the command below once the above requirements are met.

```
[host]$ rpmbuild --rebuild <source_package_name>.src.rpm
```

Please note that the commands above are executed as non-root user, the binary RPM produced by *rpmbuild* is typically stored in *<HOME>/rpmbuild/RPMS/x86_64*, the exact location and name of the binary RPM is reported by *rpmbuild* in its output.

Note: When using *rpmbuild* and the spec file to re-build the source RPM, three variables should be defined in the command line:



Tinkerer: benchmark modifications

```
[host]$ rpmbuild --bb <SPEC file> --define 'name <NAME>' \  
--define 'version <VERSION>' --define 'release <RELEASE>'
```

Where:

<NAME> is *<product>-micperf* depending on the software stack

<VERSION> for instance 1.5.0 or 4.4.3

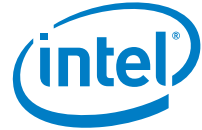
<RELEASE> a single digit indicating the release number e.g. 1

For example:

```
[host]$ rpmbuild --bb xppsl-micperf-1.5.0.spec --define \  
'name xppsx-micperf' --define 'version 1.5.0' \  
--define 'release 1'
```

Note: *Micperf* make files respect the GNU standard *DESTDIR* and *prefix* variables that can be used to relocate the installation path, however, if the install path is relocated, *micprun* will not find the new executable at run time.

§



7 Test developer: integrating new benchmarks

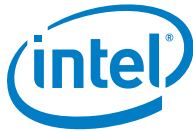
The *micperf* package is designed to be able to incorporate a wide range of benchmarks, and can be used to wrap small computational kernels that are not fully featured benchmarks. The *micperf* infrastructure is written in Python and the class that is used for abstracting the requirements of a benchmark or computational kernel is called *micp.kernel.Kernel*. If a user wants to extend the set of benchmarks used by *micprun* they simply need to add a package to the Python environment (typically with the *PYTHONPATH* environment variable, or by installing into the *site-packages*) where this package has one module for each add-on benchmark, and each module has a class that inherits from *micp.kernel.Kernel*. Each user defined class derived from the *micp.kernel.Kernel* class must have the same name as the module that includes it. To access a user defined add on package with *micprun*, the package name is passed with the *-e* option. The *kernels* sub-directory of the *micp* package serves as an example of how the add-on package should be structured including the *__init__.py* package file.

The *micp.kernel.Kernel* class is an abstract base class with a supporting factory class [4]. Each method of the base class has a doc string that defines the requirements of the method, and each method has a default implementation with the exception of *__init__()*. The default implementations are designed to support simple computational kernel function calls that were wrapped with an executable that uses positional command line arguments and prints performance data with a format styled after the Google Test framework. To the extent that a workload deviates from this simple case the base class methods must be overridden.

The default implementations provided by the Kernel base class should not be used when adapting an existing benchmark if doing so would require alteration of the benchmark; rather, the method implementations of the derived Kernel class should be adapted to the behavior of the existing benchmark. The requirements and definitions of the class methods can be obtained by looking at the help for this class. This can be done by invoking a Python interpreter that has *micp* included in the *PYTHONPATH* and run:

```
>>> import micp.kernel
>>> help(micp.kernel.Kernel)
```

The user can also refer to the *kernels* sub-directory of *micp* which contains examples of how the class was adapted to run the benchmarks included in *micp*.



8 References

- [1] Anthony Danalis, Gabriel Marin, Collin McCurdy, Jeremy S. Meredith, Philip C. Roth, Kyle Spafford, Vinod Tipparaju, and Jeffrey S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, GPGPU '10*, pages 63–74, New York, NY, USA, 2010. ACM.
- [2] Jack Dongarra, Piotr Luszczek, and Antoine Petit. The linpack benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003.
- [3] E. Foster-Johnson. *Red Hat RPM Guide*. Linux solutions from the experts at Red Hat. Wiley, 2003.
- [4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [5] Michael J. Hammel. Png: The definitive guide. *Linux J.*, 2000(69es), January 2000.
- [6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.
- [7] IEEE. *IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Shell and Utilities, Issue 6*. 2001. Revision of IEEE Std 1003.11996 and IEEE Std 1003.2-1992) Open Group Technical Standard Base Specifications, Issue 6.
- [8] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1994.
- [9] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3):308–323, September 1979.
- [10] John D. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. Technical report, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. <http://www.cs.virginia.edu/stream/>.
- [11] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, December 1995.
- [12] Intel® Math Kernel Library (Intel® MKL). <http://software.intel.com/en-us/intel-mkl>.
- [13] <https://software.intel.com/en-us/articles/intel-optimized-technology-preview-for-high-performance-conjugate-gradient-benchmark>
<https://wiki.centos.org/HowTos/RebuildSRPM>
- [14] https://fedoraproject.org/wiki/Packaging:Guidelines?rd=Packaging/Guidelines#Patch_Guidelines
- [15] <https://github.com/baidu-research/DeepBench>
- [16] <https://github.com/axboe/fio>