Micperf: A Unified Interface for Benchmark Tools on the Intel ® Xeon Phi ™Coprocessor

Christopher Cantalupo

September 18, 2014

Abstract

There are many benchmarks that can be used with the Intel ® Xeon Phi ™Coprocessor to measure performance. These have different developers, and, consequently, different user interfaces, different methods of execution and different output. Micperf 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 executables: one for execution of benchmarks (`micprun`), and four that interpret the output of the first. 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 value output that can be easily imported into a variety of other applications. 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.

1 Introduction

The micperf package targets a range of users including engineers interested in performance regression testing while implementing modifications to hardware, firmware, driver software or operating system software. In addition to these highly technical customers, there are also application users and hardware manufacturers who use the micperf software for demonstration and system verification purposes. This paper serves as a user guide by providing a description of the micperf user experience for a range of use cases. The simplest use cases are identified in Section 2. The sections that follow 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 six benchmarks: MKL [13] SMP Linpack [2], MKL SGEMM, MKL DGEMM, SHOC [1] download, SHOC readback, and STREAM [11, 10]. These benchmarks were carefully chosen to demonstrate performance in all of the major bottlenecks in the system. The Intel ® Xeon Phi ™Coprocessor floating point unit (FPU) and vector processing unit (VPU) excel at dense level three basic linear algebra [9] calculations, and the Linpack, DGEMM, and SGEMM benchmarks demonstrate these capabilities. Linpack and DGEMM compute with double precision floating point numbers and SGEMM computes in single precision. The SHOC download and SHOC readback benchmarks test the performance of the PCIe bus in transferring data between the host and coprocessor. The STREAM

benchmark measures the coprocessor memory bus bandwidth between the GDDR memory and the computational registers.

This paper is organized into sections that each describe a different user's experience with the micperf package. Section 2 focuses on the final consumer of the Intel ® Xeon Phi ™Coprocessor product who uses basic features of micperf. Section 3 discusses the experience of the advanced user who generates professional quality content about the performance of the Intel ® Xeon Phi ™Coprocessor for presentation. Section 4 describes the use case of the hardware validator performing performance regression tests against Intel ® Validation measurements. Section 5 details how a systems engineer uses micperf to run performance regression tests on their system modifications. Section 6 describes how a curious user can inspect or make modifications to benchmark source code. Section 7 describes a benchmark developer's experience integrating new benchmarks into the micperf infrastructure.

2 Novice User: Benchmark Execution

An important use case, and category which includes many micperf users, is the customer who has just installed the Intel ® Xeon Phi ™Coprocessor on their own system, or has just obtained access to a system that has the coprocessor installed. This customer is interested in the product because of the performance boost it can provide, and they would like to demonstrate the capabilities of this new piece of hardware. Micperf provides a demonstration of functionality, and performance numbers that serve as a motivation to learn how to use the product.

The micperf package is distributed as a Red Hat Package Manager (RPM) [3] file which is an optional package in the `perf` sub-directory within the Intel ® Manycore Platform Software Stack (Intel ® MPSS) [12] below the primary packages. The micperf package is distributed as a binary RPM, but this archive also installs source code. The Intel ® MPSS general readme file includes instructions regarding micperf's software dependencies and installation instructions. Once installed there is a readme file specific to micperf installed to

```
/usr/share/doc/micperf-MPSSVERSION/README.txt
```

This file describes what has been installed, basic use instructions and references to other sources of documentation. In the same directory, a change log (`CHANGES.txt`) is posted, and the instructions that were given in the Intel ® MPSS general readme are repeated in a file called `INSTALL.txt`. The micperf install procedure conforms to Linux conventions with the exception that it includes source code in a binary RPM. The reason for this choice was to avoid having a dependency on the Intel ® Composer XE package and because packaging the source code along with binaries serves the purpose of transparency as described in Section 6.

The first step to using the micperf package is running `micprun` and reading the help documentation that the application provides. This is accessed using the POSIX standard [7] getopt style command line arguments in either long form: `micprun --help`, or short form: `micprun -h`. The Unix convention for printing the version number is also respected: `micprun --version`. The command line arguments that give basic control over execution are `-k`, `-c` and `-p`. The amount of output created can be controlled by the verbosity option, `-v`. For more fine grained control of the output see Section 3.

2.1 Benchmark Selection

One of the simplest modifications to the standard `micprun` execution is to select the benchmarks that will be run. Benchmark selection is done with the `-k` command line option and by default when `-k` is not specified, all benchmarks are executed. To list the names of all available benchmarks execute the command `micprun -k help`. More than one benchmark can be selected by a colon separated list, *e.g.* `micprun -k linpack:dgemm:stream`, and the default option to run all workloads can be explicitly specified by `micprun -k all`.

Table 1: Benchmarks

Benchmark	CLI Name	Target Operations	Component
MKL DGEMM	dgemm	Double precision floating point	FPU and VPU
MKL SGEMM	sgemm	Single precision floating point	FPU and VPU
MKL SMP Linpack	linpack	Double Precision Floating Point	FPU and VPU
SHOC Download	shoc_download	Bus transfer host to device	PCIe bus
SHOC Readback	shoc_readback	Bus transfer device to host	PCIe bus
STREAM	stream	Round-trip memory to registers	GDDR and caches

2.2 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 category names 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	execute a self-check
optimal_quick	parameters that yield good performance, but with a short run time
scaling_quick	a subset of the scaling category: excludes long run time parameters
scaling_core	strong core scaling test if possible otherwise run scaling category

2.3 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 functionality is accessed with the `-p` option to `micprun`, and note that the `-p` option cannot be used on the command line with the `-c` option. 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.

To print the long form command line options for a benchmark and the default values run `micprun -k <benchName> -p help`. Note that the default values are the parameters for the optimal category for all of the benchmarks included in the micperf package. In the case where the optimal parameter category searches a list of parameters, the default values correspond to the last parameter set in the search list. It is possible to substitute short form command line options by using the first character of the long form option, but this feature can only be used when each of the long form parameters identified with the benchmark starts with a unique character.

2.4 Offload Selection

Benchmarking the Intel ® Xeon Phi ™Coprocessor differs from benchmarking on other platforms due to the fact that the coprocessor runs an independent Linux OS communicating with the host system across the PCIe bus. As the name "coprocessor" implies, the use model for the Intel ® Xeon Phi ™Coprocessor is to offload work from the host system. There are several methods for doing this and one of the important features of micperf is the ability to compare performance characteristics of a variety of offload methods on the same benchmark.

These offload options include running natively on the coprocessor without any interaction with the host, using the Intel ® Symmetric Communications Interface (SCIF) low level API to connect a host and coprocessor application, and compiler assisted offload where snippets of code from within a host application are offloaded to the coprocessor during run-time in an automated way. These options can be selected with the `-x` option to `micprun`, and more than one method can be given in a colon separated list.

Table 3: Offload Methods

Method	Host Process	Device Process	Communication
native	No	Yes	None
scif	Yes	Yes	Intel ® Symmetric Communications Interface
pragma	Yes	No	Intel ® Composer XE Compiler Assisted Offload

2.5 Coprocessor Selection

Many systems have more than one Intel ® Xeon Phi ™Coprocessor installed. A single instance of `micprun` is able to execute benchmarks on only one coprocessor at a time. The coprocessor index can be selected with the `-d` option. The details about each of the coprocessors and how they map to the coprocessor index can be seen with the Intel ® MPSS tool `micinfo`.

2.6 Verbosity Selection

The verbosity option to `micprun` gives coarse control of the output. The `-v` option takes 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 are outlined in Table 4.

Table 4: 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

Some of the column labels in Table 4 require clarification. “Summary” refers to a concluding section in the standard output that reprints the performance data collected as it is displayed by `micpprint`. “Joint Plot” 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 the same X and Y axes. The infrastructure supports benchmarks that collect ancillary data that is not displayed by default, but can be displayed if the user requests. The primary data is considered “rolled up”, and the ancillary data is included when “all” data is requested. 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.

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`, `micpplot`, `micpcsv`, and `micpinfo`.

3.1 Creating a micp_run_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 whem `-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 micp_run_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 have been 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 the commands using the `--app` option. By default, the application runs the command set on the current system, and if a micp_run_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 micp_run_stats file can be displayed in human readable form with the `micpprint` helper application. The output is organized by benchmark and offload method. Multiple micp_run_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 micp_run_stats files. Each section of output is preceded by the tag that is associated with the file. Note that 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 true for the other helper applications that display performance data as well: `micpcsv` and `micpplot`.

3.4 Data Inspection with micpcsv

The `micpcsv` helper application is used to create data that can be easily parsed by applications external to micperf. In particular it produces comma separated value data which is readily imported into a wide variety of applications such as spreadsheets and databases. Since comma separated value output is a very unstructured format, there are several styles of output formatting from `micpcsv` which are chosen by passing flags to the command line.

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, offload method and coprocessor SKU from the scaling reference data included in the micperf package. These data are displayed in a summary table that includes performance information and the value of the independent variable in the scaling parameter category. If the `-o` flag is given then the output file named "summary.csv" is created in the output directory. See Figure 1 for an example of the summary table.

	SGEMM pragma/native (GF/s)	DGEMM pragma/native (GF/s)	SMP Linpack (GF/s)	PCIeDownload pragma/scif (GB/s)	PCIeReadback pragma/scif (GB/s)	STREAM (Triad) (GB/s)
B1QS-5110P	1448.24 / 1668.51	652.43 / 794.54	721.03	6.93 / 6.94	6.98 / 6.98	171.41
Parameters	15360 / 15360	10240 / 7680	26624	1024MB / 1024MB	1024MB / 128MB	58
B1QS-7110P	1550.54 / 1782.89	693.53 / 842.13	755.17	6.99 / 6.95	7.06 / 7.06	175.22
Parameters	15360 / 15360	10240 / 7680	26624	1024MB / 1024MB	1024MB / 1024MB	60

Figure 1: Example summary output from micpcsv as displayed by Microsoft Excel

3.4.2 Long Format

If a micp_run_stats file is passed to `micpcsv` but the `-o` flag is not given then the standard output is comprised 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 second is the run record table. The identifying table has just a header row of the form "KERNEL, OFFLOAD, TAG" and a single row that describes the benchmark, offload method and tag. These three identifiers apply to all of the run records in the table that follows the identifying table. The run record table has a first column of run descriptions. This is followed by columns giving the values of all parameters passed (one column per parameter). The last column in these tables provides the performance metric. Note that for optimal parameter category runs each of the run record tables has a header row, and only one row of values. When specifying the optimal parameter category to `micprun`, the short format output (Section 3.4.3) can 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 micp_run_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 (*i.e.* when no micp_run_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 and disadvantageous in that 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 micp_run_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 micp_run_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 micp_run_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. An example of the output from micpplot is given in Figure 2.

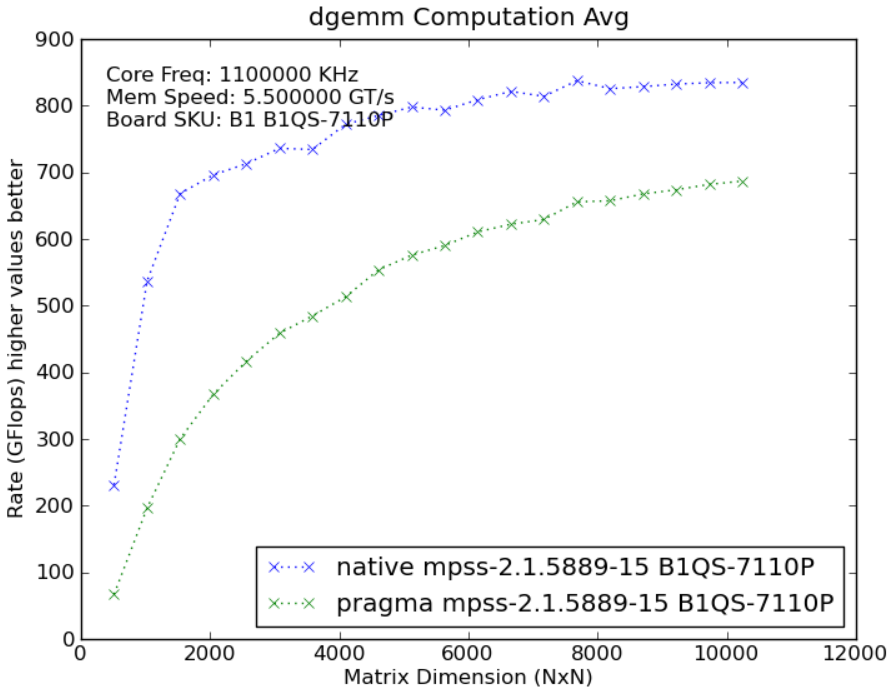

Figure 2: Example output from micpplot

4 Hardware Tester: Executing Regression Tests

The micperf package is distributed with a set of reference micp_run_stats files. These files are accessible by using 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 ™Coprocessor 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 `--help` output).

The data that is distributed uses the default tag which contains the product stock keeping unit (SKU), the Intel ® MPSS version, the offload methods and the parameter category. The Unix grep utility can be used to select the tag appropriate for the regression test to be performed. For instance, to run a regression test on a 5110 SKU part using the optimal parameter category the user would run the following command:

```
refTag=‘micprun -R help | grep 5110 | grep optimal‘
micprun -t test -o . -m 0.04 -R $refTag
```

Note that the `-o` and `-t` flags were given above which produces a micp_run_stats file. It can be very useful to inspect this output if a performance regression is detected. Also note that it is possible to run a subset of the benchmarks included in the reference file by specifying the `-k` and/or `-x` flags (in fact all flags can be overridden).

To show how the micp_run_stats file from the regression test can be used with the helper applications we will give some examples here. The `micpinfo` application can be used to compare the difference between the two system configurations using the Unix diff tool:

```
micpinfo -R $refTag > ref_info.log
micpinfo micp_run_stats_test.pkl > test_info.log
diff ref_info.log test_info.log > diff_info.log
```

This will highlight any differences between the entireties of the `micpinfo` logs and put this comparison into the file `diff_info.log`. This comparison can be narrowed down by passing the `--app` flag to `micpinfo`: *e.g.* passing `--app conf` in the calls to micpinfo above will show the differences in the mic configuration files. The `micpprint` utility can be used to examine the performance data and compare with the reference:

```
micpprint -R $refTag micp_run_stats_test.pkl
```

Similar results can be produced in a form more easily parsed by a computer program using `micpcsv`. Note that the tags are not displayed in the CSV short form output making the `-s` flag less useful when comparing multiple files. The `micpplot` application can be used to inspect the regression; for example, the command:

```
micpplot -R $refTag micp_run_stats_test.pkl
```

will plot the test lines against the reference lines. Note that the identifying tag is displayed in the plot legend.

5 Systems Engineer: Creating Reference Data

Section 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 Section 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 is changed, 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.

Let's say the user wants to determine if changing a power management BIOS setting on the host system will have an impact on performance. To test this, the user generates a micp_run_stats file on the system with the original BIOS setting using the `-o` option to `micprun`. The user then reboots the system, changes the BIOS setting and then runs against the micp_run_stats file just 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 an Intel ® Xeon Phi ™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. With the exception of Intel ® MKL SMP Linpack, all of the benchmarks that are included in the micperf package are distributed with source code and make files for compilation. The included source files are installed to `/usr/src/micperf`. Intel ® MKL SMP Linpack can be used with micperf; however, the binary is distributed by Intel ® MKL, and neither the source nor the binary are distributed with the micperf package.

In order to build the benchmark source Intel ® Composer XE must be installed, and the "compilervars" script must be sourced. If these requirements are met then the user can simply run `make; sudo make install` in the directory containing the source code to replace a benchmark executable with one built by the user. This gives the user the option of changing the source code to meet the specific needs of a test they would like to execute. Note that the micperf make files respect the GNU standard "DESTDIR" and "prefix" variables that can be used to relocate the install path; however, if the install path is relocated `micprun` will not find the new executable at run time.

An example of a user modification to benchmark source code is changing the GEMM benchmark so that it uses `malloc` rather than `mmap` to allocate the memory used for the computation. The user could replace the `mmap` call in the `utils.c` file in the GEMM source directory recompile following the steps above and run `micprun`. This could even be done as a regression test following the steps outlined in Section 5, where the benchmark is modified rather than an element of the platform.

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 has a default implementation with the exception of `__init__()`. The default implementations are designed to support simple computational kernel function calls that have been 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 Summary

The micperf package provides benchmarking solutions for the users and developers of the Intel ® Xeon Phi ™Coprocessor product. The material presented here gives a holistic view of the users and use cases, but is not a substitute for the specific detailed documentation provided by the `--help` message from the micperf executables or the other documentation included in the package. In this paper the reader has learned how to run benchmarks and create presentation material from those runs. The paper has covered how to do regression testing against Intel ® Validation performance measurements, and how to do regression testing against the user's own measurements. Some of the details on how to modify the source code for the included benchmarks and how to extend micperf to include new benchmarks have been discussed.

The micperf package appeals to community standards for benchmark inclusion, open source distribution, GNU standard build, Unix command line interfaces and established object oriented

design patterns. The use of standards allows the micperf package to be intuitive for users who are familiar with the Unix environment and industry benchmarks. Appealing to standards also improves transparency while lending credibility to the results. The micperf package allows for modular extension by using the object oriented design offered by Python while decoupling the Python infrastructure from the build of the benchmark executables. The users and use cases discussed cover a wide range of benchmarking needs, and this paper will facilitate wider and more effective use of the tool.

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 Petitet. 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 1003.1-1996 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 ®. Intel ®Manycore Platform Software Stack (MPSS). http://software.intel.com/en-us/articles/intel-manycore-platform-softwa%re-stack-mpss.

[13] Intel ®. Intel ®Math Kernel Library (Intel ®MKL) 11.0. http://software.intel.com/en-us/intel-mkl.