Supplementary Materials: A Novel Statistical Learning-Based Methodology for Measuring the Goodness of Energy Profiles of Applications Executing on Multicore Computing Platforms

Muhammad Fahad *, Arsalan Shahid, Ravi Reddy Manumachu and Alexey Lastovetsky

The supporting materials for the main manuscript, "A Novel Statistical Learning-Based Methodology for Measuring the Goodness of Energy Profiles of Applications Executing on Multicore Computing Platforms" are:

- Pattern Matching Approaches in Other Fields for Closely Related Problems
- Experimental Setup and Applications Suite Used for Construction of The Energy Profiles
- Similarity Results of Group A
- Similarity Results of Group B

1. Pattern Matching Approaches in Other Fields for Closely Related Problems

The similarity measure problem, in general, can also be classified as *pattern matching*. A plethora of different methods and approaches have been proposed to solve this problem in different fields such as data mining, time series similarity analysis, and graph (matching) theory. The proposed solutions can be categorized into a non-exhaustive list of categories such as lock-step, pattern-based, model-based, elastic, feature-based et cetera.

However, the main difference of similarity matching in energy modeling with time-series analysis is that the energy profiles are (usually) a function of application configuration parameters (such as problem sizes, CPU threads, CPU cores, etc.) whereas the time series data is uni-variate (only one variable varies over time). Furthermore, the energy profiles are of the same cardinality as of the ground truth (i.e., they have equal lengths) whereas it is not necessary in case of graph matching, time series, and data mining. Therefore, we do not cover the similarity measures commonly known as elastic similarity measures (such as dynamic time warping (DTW) [1]) where a data point of a time series is compared with many data points of other time series and vice-versa i.e., one-to-many mapping and many-to-one mapping. Now we present some of the popular similarity measures and distance metrics used to determine the pattern matching.

Similarity Measures: Similarity measures establish the resemblance between the ground truth and a profile as an absolute value usually within the interval of [0,1] or [−1,1] where 0 or −1 indicates an absolute dissimilarity and 1 denotes a maximum similarity. Apart from the Pearson correlation coefficient, Cosine similarity is another popular similarity measure that measures the orientation of two non-zero n-dimensional vectors irrespective of their magnitude. It measures the cosine of the angle between both vectors and is mainly used in text mining problems such as text classification, text summarization, information retrieval, question answering, etc. [2]. It is calculated by the dot product of the vectors and normalized by the product of their lengths. Nakamura et al. [3] propose a shape-based similarity measure called angular metric for shape similarity (AMSS) for time series data. It adopts cosine similarity as a principle measure which disregards exact points or vector length.

Distance Metrics: Another way to compare a profile against the ground truth is the use of a distance metric which establishes an absolute value of how far two objects are. Euclidean distance is one of the most popular distance metrics in data mining and time series similarity comparison. It is based on Pythagoras' theorem and measures the distance between two vectors or two points in a Euclidean space [4]. It is also used for the clustering of time series data [5,6] because of its indexing

capabilities and simple computation. The generalized form of Euclidean distance is the Minkowski distance [7] which is also called L_p norm. For Manhattan distance, p is equal to 1, and for Euclidean distance, p is equal to 2. However, Minkowski distance variants are blind to capture the data correlation.

Another distance metric widely used in pattern recognition or graph matching in graph theory is graph edit distance (GED) [8]. It is related to string-edit-distance between two strings. It computes the cost of recognition of nodes and the minimum number of modifications (such as deletion, insertion, substitution) required to transform the input graph into the referenced one. However, the complexity of computing GED is non-polynomial [9].

Auto-regressive (AR) modeling is a model-based approach that extracts the features from time-series to use their underlying models to determine the similarity between them. AR modeling specifies that the current value in a data-set (time-series) depends linearly on its preceding value(s). The autoregressive integrated moving average (ARIMA) also called the Box–Jenkins [10] method, is a popular approach used in time-series analysis and for anomaly detection in data [11]. The main idea of using AR modelling to measure the goodness is to learn the models of time-series and then compute the goodness using the model parameters. Several approaches can be found in the literature which use AR to find the similarity between two time-series such as [12–14]. However, the basic assumption of AR modelling is that the data (time-series) is uni-variate and the future value depends upon the past value(s). This is not the case with energy modeling because application energy profiles are (usually) a function of application parameters, and each data-point in an energy profile is distinct and independent of all other data-points in that profile. Furthermore, AR modelling assumes that the data is stationary which means that the statistical properties such as mean and variance of the time-series do not change over time. On the contrary, the energy profiles are not stationary and there also exists an energy consumption trend. Therefore, AR modelling approach is not applicable straightforwardly for determining the goodness of energy profiles.

2. Experimental Setup and Applications Suite Used for Construction of The Energy Profiles

The input to the proposed solution method trend-based Similarity measure (TSM) is a set of energy profiles (of an application) constructed with different energy measurement approaches (such as integrated on-chip power sensors, external power meters, energy predictive models employing PMCs as predictor variables) and the precision settings used for constructing the profiles. The output of TSM is the ranking of energy profiles based on their goodness.

The dataset used in this article comprises of 235 energy profiles of different application configurations on multicore heterogeneous hybrid computing platforms with on-chip sensors, power meters, or PMCs as predictor variables. The application configuration parameters are (a) problem size, (b) number of CPU threads, or the number of CPU cores. Table S2 in Appendix 2.5 presents the list of applications employed to construct the energy profiles. Our application suite contains highly optimized memory bound and compute-bound scientific routines such as OpenBLAS Double-precision General Matrix Multiplication (DGEMM), Fastest Fourier Transform in the West (FFTW) 2D, DGEMM and Fast Fourier Transform (FFT) from Intel Math Kernel Library (MKL), benchmarks from NASA Application Suite (NAS), Intel High Performance Conjugate Gradients (HPCG), stress, naive matrix-matrix multiplication, and naive matrix-vector multiplication.

The profiles are constructed as the results of experiments conducted for different research works already published in open access peer-reviewed journals such as *IEEE Access* and *Energies*. The experimental set-up, platforms, application suite, configuration parameters, and the boundary conditions all are presented in details in their corresponding publications [15,16] which are open access. We briefly present the details of experimental methodology and experiment configurations in following sections used for these experiments for the convenience of reader and to ensure the reproducibility.

2.1. Technical Description of Experimental Platforms

We employ three nodes for our comparative study: (a) HCLServer01 has an Intel Haswell multicore CPU having 24 physical cores with 64 GB main memory and integrated with two accelerators: one Nvidia K40c GPU and one Intel Xeon Phi 3120P, (b) HCLServer02 has an Intel Skylake multicore CPU consisting of 22 cores and 96 GB main memory and integrated with one Nvidia P100 GPU and (c) HCLServer03 has an Intel Skylake multicore CPU having 56 cores with 187 GB main memory. Technical description of each node is provided in table S1. These nodes are representative of computers used in cloud infrastructures, supercomputers, and heterogeneous computing clusters.

Each node has a power meter installed between its input power sockets and the wall A/C outlets. HCLServer01 and HCLServer02 are connected with a Watts Up Pro power meter; HCLServer03 is connected with a Yokogawa WT310 power meter. Watts Up Pro power meters are periodically calibrated using the ANSI C12.20 revenue-grade power meter, Yokogawa WT310.

The maximum sampling speed of Watts Up Pro power meters is one sample every second. The accuracy specified in the data-sheets is $\pm 3\%$. The minimum measurable power is 0.5 watts. The accuracy at 0.5 watts is ± 0.3 watts. The accuracy of Yokogawa WT310 is 0.1% and the sampling rate is 100 k samples per second.

HCLServer01, HCLServer02, and HCLServer03 have a dedicated power meter installed between their input power sockets and wall A/C outlets. The power meter captures the total power consumption of the node. It has a data cable connected to the USB port of the node. A perl script collects the data from the power meter using the serial USB interface. The execution of this script is non-intrusive and consumes insignificant power.

Table S1. Technical Specifications of HCLServers.

Technical Specifications	**HCLServer01**	**HCLServer02**	**HCLServer03**
Processor	Intel E5-2670 v3 @2.30GHz	Intel Xeon Gold 6152	Intel Xeon Platinum 8180
Micro-architecture	Haswell	Skylake	Skylake
OS	CentOS 7	CentOS 7	CentOS 7
Thread(s) per core	2	2	2
Cores per socket	12	22	28
Socket(s)	2	1	2
NUMA node(s)	2	1	2
L1d cache / L11 cache	32 KB / 32 KB	32 KB / 32 KB	32 KB / 32 KB
L2 cache	256 KB	1024 KB	1024 KB
L3 cache	30,720 KB	30,976 KB	30,976 KB
	NVIDIA K40c	**NVIDIA P100 PCIe**	
No. of processor cores	3584	2880	
Total board memory	12 GB CoWoS HBM2	12 GB GDDR5	
Memory bandwidth	549 GB/sec	288 GB/s	
	Intel Xeon Phi 3120P		
No. of processor cores	57		
Total main memory	6 GB GDDR5		
Memory bandwidth	240 GB/sec		

We use Running Average Power Limit (RAPL) [17], NVIDIA Management Library (NVML) [18] and an Intel System Management Controller chip (SMC) [19] (using Intel manycore platform software stack (MPSS) [20]) to determine the energy consumed by the application kernels executing on Intel CPUs, Nvidia GPUs, and Intel Xeon Phi respectively. HCLWattsUp interface [21] is used to obtain the power measurements from the WattsUp Pro power meters. The Intel MKL on all HCLServers is 2017.0.2, whereas the CUDA versions used on HCLServer01 is 7.5 and 9.2.148 on HCLServer02.

The remaining section is organized as follows: The details on the HCLWattsUp interface is presented in Section 2.2 followed by the methodology used to obtain a reliable data point using different tools (on-chip power sensors and power meters) in Section 2.3. After that the details about

the methodology to obtain the dynamic energy consumption using on-chip power sensors is presented in Section 2.4. Finally, the application suite used for construction of the dynamic energy profiles is presented Section 2.5.

2.2. Application Programming Interface (API) for Measurements Using External Power Meter Interfaces (HCLWattsUp)

We use HCLWattsUp API function, which gathers the readings from the power meters to determine the average power and energy consumption during the execution of an application on a given platform. HCLWattsUp API can provide the following four types of measures during the execution of an application:

- *TIME*—Execution time (seconds).
- *DPOWER*—Average dynamic power (watts).
- *TENERGY*—Total energy consumption (joules).
- *DENERGY*—Dynamic energy consumption (joules).

We confirm that the overhead due to the API is very minimal and does not have any noticeable influence on the main measurements. It is important to note that the power meter readings are only processed if the measure is not *hcl::TIME*. Therefore, for each measurement, we have two runs. One run for measuring the execution time and the other for energy consumption. The example provided in figure S1 illustrates the use of statistical methods to measure the dynamic energy consumption during the execution of an application.

```
#include <wattsup.hpp>
int main(int argc, char** argv)
{
std::string pathsToMeters[2] = {
"/opt/powertools/bin/wattsup1",
"/opt/powertools/bin/wattsup2"};
std::string argsToMeters[2] = {
"--interval=1",
"--interval=1"};
hcl::Wattsup wattsup(
2, pathsToMeters, argsToMeters
);
hcl::Precision pIn = {
maxRepeats, cl, maxElapsedTime, maxStdError
};
hcl::Precision pOut;
double sampleMean, sd;
int rc = wattsup.execute(
hcl::DENERGY, executablePath,
executableArgs, &pIn, &pOut,
&sampleMean, &sd
);
if (rc == 0)
std::cerr << "Precision NOT achieved.\n";
else
std::cout << "Precision achieved.\n";
std::cout << "Max repetitions "
<< pOut.reps_max
<< ", Elasped time "
<< pOut.time_max_rep
<< ", Relative error "
<< pOut.eps
<< ", Mean energy "
<< sampleMean
<< ", Standard Deviation "
<< sd
<< std::endl;
exit(EXIT_SUCCESS);
}
```

Figure S1. Example illustrating the use of HCLWattsUp Application Programming Interface (API) for measuring the dynamic energy consumption.

The API is confined in the *hcl* namespace. Lines 10–12 construct the Wattsup object. The inputs to the constructor are the paths to the scripts and their arguments that read the USB serial devices containing the readings of the power meters.

The principal method of *Wattsup* class is *execute*. The inputs to this method are the type of measure, the path to the executable *executablePath*, the arguments to the executable *executableArgs* and the statistical thresholds (*pIn*) The outputs are the achieved statistical confidence *pOut*, the estimators, the sample mean (*sampleMean*) and the standard deviation (*sd*) calculated during the execution of the executable.

The *execute* method repeatedly invokes the executable until one of the following conditions is satisfied:

- The maximum number of repetitions specified in $maxRepeats$ is exceeded.
- The sample mean is within $maxStdError$ percent of the confidence interval cl. The confidence interval of the mean is estimated using Student's t-distribution.
- The maximum allowed time $maxElapsedTime$ specified in seconds has elapsed.

If any one of the conditions are not satisfied, then a return code of 0 is output suggesting that statistical confidence has not been achieved. If statistical confidence has been achieved, then the number of repetitions performed, time elapsed and the final relative standard error is returned in the output argument $pOut$. At the same time, the sample mean and standard deviation are returned. For our experiments, we use values of (1000, 95%, 2.5%, 3600) for the parameters ($maxRepeats, cl, maxStdError, maxElapsedTime$). Since we use Student's t-distribution for the calculation of the confidence interval of the mean, we confirm specifically that the observations follow normal distribution by plotting the density of the observations using *R* tool.

2.3. Methodology to Obtain a Reliable Data Point

We follow the following strict methodology described below to make sure the experimental results are reliable:

- The server is fully reserved and dedicated to these experiments during their execution. We also made certain that there are no drastic fluctuations in the load due to abnormal events in the server by monitoring its load continuously for a week using the tool *sar*. Insignificant variation in the load was observed during this monitoring period suggesting normal and clean behaviour of the server.
- We set the application kernel's CPU affinity mask using SCHED API's system call SCHED_SETAFFINITY() Consider for example mkl-DGEMM application kernel running on HCLServer01. To bind this application kernel, we set its CPU affinity mask to 12 physical CPU cores of Socket 1 and 12 physical CPU cores of Socket 2.
- To make sure that pipelining, cache effects, and so forth, do not happen, the experiments are not executed in a loop and sufficient time (120 s) is allowed to elapse between successive runs. This time is based on observations of the times taken for the memory utilization to revert to base utilization and processor (core) frequencies to come back to the base frequencies.
- To obtain a data point, the application is repeatedly executed until the sample mean lies in the 95% confidence interval and a precision of 0.025 (2.5%) has been achieved. For this purpose, Student's t-test is used assuming that the individual observations are independent and their population follows the normal distribution. We verify the validity of these assumptions by plotting the distributions of observations.

The function $MeanUsingTtest$, shown in Algorithm 1, describes this step. The inputs to the function $MeanUsingTtest$ are:

- The application to execute, app

- The minimum number of repetitions, $minReps \in \mathbb{Z}_{>0}$
- The maximum number of repetitions, $maxReps \in \mathbb{Z}_{>0}$
- The maximum time allowed for the application to run, $maxT \in \mathbb{R}_{>0}$
- The required confidence level, $cl \in \mathbb{R}_{>0}$
- The required accuracy, $eps \in \mathbb{R}_{>0}$

The outputs by the function $MeanUsingTtest$ are:

- The number of experimental runs actually made, $repsOut \in \mathbb{Z}_{>0}$
- The confidence level achieved, $clOut \in \mathbb{R}_{>0}$
- The accuracy achieved, $epsOut \in \mathbb{R}_{>0}$
- The elapsed time, $etimeOut \in \mathbb{R}_{>0}$
- The mean, $mean \in \mathbb{R}_{>0}$

For each data point, the function is invoked, which repeatedly executes the application app until one of the following three conditions is satisfied:

1. The maximum number of repetitions ($maxReps$) have been exceeded (Line 3).
2. The sample mean falls in the confidence interval (or the precision of measurement eps has been achieved) (Lines 15–17).
3. The elapsed time of the repetitions of application execution has exceeded the maximum time allowed ($maxT$ in seconds) (Lines 18–20).

So, for each data point, the function $MeanUsingTtest$ is invoked and the sample mean $mean$ is returned at the end of invocation. The function $Measure$ measures the execution time or the dynamic energy consumption using the HCL's WattsUp library [21] based on the input, $TIME$ or $ENERGY$. The input minimum and maximum number of repetitions, $minReps$ and $maxReps$, differ based on the problem size solved. For small problem sizes ($32 \leq n \leq 1024$), these values are set to 10,000 and 100,000. For medium problem sizes ($1024 < n \leq 5120$), these values are set to 100 and 1000. For large problem sizes ($n > 5120$), these values are set to 5 and 50. The values of $maxT$, cl and eps are set to 3600, 0.95, and 0.025. If the precision of measurement is not achieved before the maximum number of repeats has been completed, we increase the number of repetitions and also the maximum elapsed time allowed. However, we observed that condition (2) is always satisfied before the other two in our experiments. The complexity of the function $MeanUsingTtest$ is O(N).

2.4. Methodology to Compare Measurements Using On-Chip Sensors and HCLWattsUp

To analyze the dynamic energy consumption by a given component when running an application, we need to build application profiles on them. HCLWattsUp API provides the dynamic energy consumption of application instead of the component. It, therefore, contains the contributions by other components including CPU host-core and DRAM. Built-in sensors, on the other hand, only provide the power consumption of GPU or Xeon Phi only (we offload the applications to run on Intel Xeon Phi, so it includes the CPU host core, DRAM and PCIe to copy and migrate the data between CPU host core and Xeon Phi). Therefore, to compare both methodologies in a most fairly equitable way and to obtain the dynamic energy profiles of applications, we use RAPL as an aide to sensors for determining the application energy because RAPL determines the power consumption of CPU and DRAM using an on-chip voltage regulator and current sensor [22].

Algorithm 1 Function determining the sample mean using Student's *t*-test.

1: **procedure** MEANUSINGTTEST(
app,minReps,maxReps,
maxT, cl, accuracy,
repsOut, clOut, etimeOut, epsOut,mean)

Input:

The application to execute, app
The minimum number of repetitions, $minReps \in \mathbb{Z}_{>0}$
The maximum number of repetitions, $maxReps \in \mathbb{Z}_{>0}$
The maximum time allowed for the application to run, $maxT \in \mathbb{R}_{>0}$
The required confidence level, $cl \in \mathbb{R}_{>0}$
The required accuracy, $eps \in \mathbb{R}_{>0}$

Output:

The number of experimental runs actually made, $repsOut \in \mathbb{Z}_{>0}$
The confidence level achieved, $clOut \in \mathbb{R}_{>0}$
The accuracy achieved, $epsOut \in \mathbb{R}_{>0}$
The elapsed time, $etimeOut \in \mathbb{R}_{>0}$
The mean, $mean \in \mathbb{R}_{>0}$

$reps \leftarrow 0;\ stop \leftarrow 0;\ sum \leftarrow 0;\ etime \leftarrow 0$
while $(reps < maxReps)\ and\ (!\,stop)$ **do**
$st \leftarrow MEASURE(TIME)$
$EXECUTE(app)$
$et \leftarrow MEASURE(TIME)$
$reps \leftarrow reps + 1$
$etime \leftarrow etime + et - st$
$ObjArray[reps] \leftarrow et - st$
$sum \leftarrow sum + ObjArray[reps]$
if $reps > minReps$ **then**
$clOut \leftarrow$ fabs(gsl_cdf_tdist_Pinv($cl,\ reps - 1$))
$\times$ gsl_stats_sd($ObjArray, 1,\ reps$)
/ sqrt($reps$)
if $clOut \times \frac{reps}{sum} < eps$ **then**
$stop \leftarrow 1$
end if
if $etime > maxT$ **then**
$stop \leftarrow 1$
end if
end if
end while
$repsOut \leftarrow reps;\ epsOut \leftarrow clOut \times \frac{reps}{sum}$
$etimeOut \leftarrow etime;\ mean \leftarrow \frac{sum}{reps}$
end procedure

Now, we present the work-flow of experiments that we follow to determine the dynamic energy consumption of the application. To obtain the CPU host-core and DRAM contribution in dynamic energy consumption of the application, we use RAPL in following way:

1. Using Intel PCM/PAPI, we obtain the *base power* of CPU and DRAM (when the given application is not running).
2. Using HCLWattsUp API, we obtain the *execution time* of the given application.

3. Using Intel PCM/PAPI, we obtain the *total energy* consumption of the CPU host-core (because all other cores are idle) and DRAM, during the execution of the given application.
4. Finally, we calculate the *dynamic energy* consumption (of CPU and DRAM) by subtracting the *base energy* from *total energy* consumed during the execution of the given application.

To obtain the GPU/Xeon Phi contribution, we use NVML/Intel SMC in following way:

1. Using NVML/Intel SMC, we obtain the base power of GPU/Xeon Phi (when the given application is not running).
2. Using HCLWattsUp API, we obtain the execution time of the given application.
3. Using NVML/Intel SMC, we obtain the total energy consumption of GPU/Xeon Phi during the execution of the given application.
4. Finally, we calculate the dynamic energy consumption GPU/Xeon Phi by subtracting the base energy from total energy consumed during the execution of the given application.

Now, we present the workflow of the experiments to determine the dynamic energy consumption by the given application kernel, using HCLWattsUp:

1. Using HCLWattsUp API, we obtain the base power of the server (when the given application is not running).
2. Using HCLWattsUp API, we obtain the execution time of the application.
3. Using HCLWattsUp API, we obtain the total energy consumption of the server, during the execution of the given application.
4. Finally, we calculate the dynamic energy consumption by subtracting the base power from total energy consumed during the execution of the given application.

2.5. Application Suite Used for the Experiments

We use following applications to construct the energy profiles.

Table S2. Applications suite for constructing the energy profiles.

Application	**Description**
OpenBLAS DGEMM	OpenBLAS library routine to compute the matrix product of two dense matrices
FFTW 2D	Two-dimensional FFT routine to compute the discrete Fourier transform of a complex signal
CuBLAS DGEMM	NVIDIA library routine which is optimized for NVIDIA GPUs to compute the matrix product of two dense matrices
CuFFT	NVIDIA library routine which is optimized for NVIDIA GPUs to compute the discrete Fourier transform of a complex signal
MKL FFT	MKL routine for fast Fourier transform
MKL DGEMM	MKL routine for dense matrix multiplication
HPCG	High performance conjugate gradient
NPB IS	Integer sort, kernel for random memory access
NPB LU	Lower-upper Gauss–Seidel solver
NPB EP	Embarrassingly parallel, kernel
NPB BT	Block tri-diagonal solver
NPB MG	Multi-grid on a sequence of meshes
NPB FT	Discrete 3D fast Fourier transform
NPB DC	Data cube

NPB UA	Unstructured adaptive mesh, dynamic memory access
NPB CG	Conjugate gradient
NPB SP	Scalar penta-diagonal solver
NPB DT	Data traffic
stress	CPU, disk, and I/O stress
Naive MM	Naive matrix-matrix multiplication
Naive MV	Naive matrix-vector multiplication

3. Similarity Results of Group A

Group A is comprised of the EPSs where there is more than one energy profile of the same application constructed with different approaches such as on-chip power sensors, system-level power measurements provided by power meters, etc.

Table S3. Similarity results for Group A.

Correlation	Average Error [%]	Euclidean Distance Between Profiles	Similarity Class	TSM Rank
DGEMM_DiffLoad				
0.9992	1.2646	1975.9796	same	**1**
0.9668	63.5344	90,472.0176	opposite	-
0.9682	64.9294	92,104.4449	opposite	-
DGEMM_EqualLoad				
0.9995	4.5639	13623.2651	similar	3
0.9993	21.2295	8270.0584	similar	2
0.9993	16.1469	7625.6795	similar	**1**
FFT_Different Load				
0.9933	3.7506	483.9803	same	**1**
0.8983	75.6650	8394.0909	similar	3
0.9002	65.9681	7318.8504	similar	2
FFT(socket1)_DGEMM(socket2)				
0.9960	1.4562	441.0981	same	**1**
0.8596	29.9055	8014.0644	similar	2
0.8900	34.5172	9025.7489	similar	3
FFT(HCLServer01)-Sensors				
0.9785	4.3116	3806.0259	similar	**1**
0.9105	15.0796	12,702.5932	similar	2
DGEMM(HCLServer01)- Sensors				
0.9383	3.0720	3803.1372	similar	**1**
0.9037	19.0641	19,732.2589	opposite	-
DGEMM_AnMoHA				
0.9762	2.0190	2257.6144	same	**1**
0.8641	7.7355	8794.7977	similar	3
0.5741	6.5949	8420.5404	opposite	-
0.6741	5.9963	7522.9645	opposite	-
0.8945	4.0381	4515.2345	similar	2
FFT_Predictive Models				
0.9887	37.2600	2414.0882	similar	5
0.9924	91.8419	3321.1547	similar	6
0.9994	7.8324	472.2268	similar	2
0.9991	11.8382	593.4430	similar	4
0.9998	3.8569	293.9054	similar	**1**
0.9997	5.1557	334.3252	similar	3
DGEMM_Predictive Models				
0.9993	27.3336	6217.9387	similar	5
0.9973	39.0893	10,576.7382	similar	6

0.9999	8.5995	2076.8247	similar	**1**
0.9985	13.1182	6574.3586	similar	3
0.9999	3.0024	1276.4288	similar	2
0.9986	6.2412	6556.0057	similar	4

4. Similarity Results of Group B

An EPS is a constituent of application, configuration parameters, profile, platform, problem size and step size.

Table S4. Similarity results of Group B. Here, problem size is $(M \times N)$ where $0 \geq M \leq N$.

Application, Configuration Parameter, Profile, Platform	Problem Size, Step Size(SS)	Correlation	Average_ Error[%]	Euclidean Distance Between Profiles	TSM Similarity
DGEMM, Problem Size, RAPL, HCLServer01	M = 12,800–20,480, N = 20,480, SS = 256	0.9319	10.2457	5161.1469	opposite
FFT, Problem Size, RAPL, HCLServer01	M = 15,104–18,688, N = 23,552, SS = 64	0.8315	9.2421	92.3809	similar
DGEMM, Problem Size, Sensors, HCLServer01	M =1 0,752–21,504, N = 21,504, SS = 256	0.7945	53.7876	3841.1001	opposite
FFT, Problem Size, Sensors, HCLServer01	M = 15,104–18,688, N = 23,552, SS = 64	0.7779	11.2001	152.0388	opposite
DGEMM, Problem Size, Sensors, HCLServer02	M = 18,176–22,528, N = 22528 SS = 128	0.5959	13.1062	1745.9799	opposite
FFT, Problem Size, Sensors, HCLServer02	M = 21,504–25,600, N = 25,600, SS = 128	0.6419	73.3393	258.5867	opposite
DGEMM, Problem Size, Sensors, HCLServer01	M = 12,800–20,480, N = 20,480, SS = 256	0.8791	37.0786	14,275.7493	similar
FFT, Problem Size, Sensors, HCLServer01	M = 15,104–18,688, N = 23,552, SS = 64	0.8534	40.8715	2715.9380	similar
DGEMM, Problem Size, RAPL, HCLServer01	M = 512–16,384, N = 16,384, SS = 512	0.9694	62.4245	4014.0942	similar
FFT, Problem Size, RAPL, HCLServer01	M = 16,256–22,528, N = 22,528, SS = 128	0.9964	16.0109	92.5880	similar
DGEMM, Problem Size, RAPL, HCLServer02	N = 6400–29,504, SS = 64	0.9857	36.1267	10,085.1871	similar
FFT, Problem Size, RAPL, HCLServer02	N = 22,400–41,536, SS = 64	0.9928	28.6694	2557.1955	opposite
IntelMKLFFT, CPU Cores, RAPL, HCLServer03	N = 43,328	0.9976	13.0454	6699.9534	similar
FFTW, CPU Threads, RAPL, HCLServer03	N = 32,768	0.9999	7.4316	5485.4512	similar
FFTW, Problem Size, RAPL, HCLServer03	N = 32,768	0.9836	10.4683	1134.8875	similar
OpenBlas, Problem Size, RAPL, HCLServer03	N = 10,240–26,112, SS = 512, G = 2, T = 56	0.9999	23.3422	15,628.3773	similar
	N=10,240–26,112, SS = 512, G = 4, T = 28	0.9998	20.8883	31,537.2291	similar
	N = 10240–26112, SS = 512, G = 7, T = 16	0.9998	22.7078	48,717.0221	similar
	N = 10240–26112, SS = 512, G = 8, T = 14	0.9998	21.5933	58,193.8600	similar
	N = 10,240–26,112, SS = 512, G = 14, T = 8	0.9997	15.8470	95,839.8982	similar
	N=10,240–26,112, SS = 512, G = 16, T = 7	0.9997	15.8663	143,722.2497	similar
	N=10,240–26,112, SS = 512, G = 28, T = 4	0.9998	15.4408	258,623.5761	similar
	N = 10,240–26,112, SS = 512, G = 56, T = 2	0.9994	16.1594	31431.2584	similar

FFTW, Problem Size, RAPL, HCLServer03	M = 512– 10,240, N = 20,480, SS = 512	0.9977	6.5117	937.3472	similar
	M = 544–10,272, N = 20,544, SS=512	0.9674	11.1643	360.9246	similar
	M = 576–10,304, N = 20,608, SS = 512	0.8846	7.0593	114.4169	same
	M = 308–10,336, N = 20,672, SS = 512	0.4054	8.0483	218.7999	similar
	M = 128–10,368, N = 20,736, SS = 512	0.5962	5.9544	83.1244	similar
	M = 108–10,400, N = 20,800, SS = 512	0.7604	6.7705	100.8612	similar
	M = 192–10,432, N = 20,864, SS = 512	0.8973	10.5817	326.0428	similar
	M = 224–10,464, N = 20,992, SS = 512	0.6522	9.9545	285.7523	similar
	M = 256–10,496, N = 20,992, SS = 512	0.5333	9.1475	274.5072	similar
	M = 288–10,528, N = 21,056, SS = 512	0.9991	5.9861	819.9989	similar
	M = 320–10560, N = 21,120, SS = 512	0.1895	5.6268	81.5864	similar
	M = 352–10,592, N = 21,184, SS = 512	0.9967	6.3244	928.4300	similar
	M = 384–10,624, N = 21,248, SS = 512	0.9480	12.5488	471.9435	similar
	M = 416–10,656, N = 21,312, SS = 512	0.9964	6.4423	900.8481	similar
	M = 448–10,688, N = 21,376, SS = 512	0.9461	10.0455	333.6082	similar
	M = 480–10,720, N = 21,440, SS = 512	0.9487	10.2235	281.0485	similar
	M = 512–10,752, N = 2154, SS = 512	0.8474	6.1786	97.7390	similar
MKL, Problem Size, RAPL, HCLServer03	N = 25,600–46,080, SS = 512, G = 1, T = 56	0.9998	13.5649	1489.3704	opposite
	N = 25,600–46,080, SS = 512, G = 2, T = 28	0.9999	12.3451	1375.7246	similar
	N = 25,600–46,080, SS = 512, G = 4, T = 14	0.9999	11.9745	1398.6977	similar
	N = 25,600–46,080, SS = 512, G = 7, T = 8	0.9999	12.3395	1603.8175	similar
	N = 25,600–46,080, SS = 512, G = 8, T = 7	0.9999	12.4057	1695.1576	similar
	N = 25,600–46,080, SS = 512, G = 14, T = 4	0.9998	12.9987	2153.5891	similar
	N = 25,600–46,080, SS = 512, G = 28, T = 2	0.9997	14.4593	3179.7384	similar
MKL, Problem Size, RAPL, HCLServer03	N = 32,768–43,456, SS = 64, G = 1, T = 56	0.9999	14.1218	7423.8074	same
	N = 32,768–43,456, SS = 64, G = 2, T = 28	0.9998	13.0212	7452.8136	similar
	N = 32,768–43,456, SS = 64, G = 4, T=14	0.9998	13.0325	7923.3728	similar
	N = 32,768–43,456, SS = 64, G = 7, T=8	0.9999	13.2150	8457.6352	same
	N = 32,768–43,456, SS = 64, G = 8, T = 7	0.9999	13.2373	8563.3480	same
	N = 32,768–43,456, SS = 64, G = 14, T = 4	1.0000	13.6266	9325.5557	similar
	N = 32,768–43,456, SS = 64, G = 28, T = 2	0.9999	15.0864	11,717.0005	same

FFTW, Problem Size, RAPL, HCLServer03	N = 35,480–41,920, SS = 64, G = 1, T = 112	0.9978	24.6166	78,669.0124	same
	N = 35,480–41,920, SS = 64, G = 2, T = 56	0.9995	12.2964	5993.3535	same
	N = 35,480–41,920, SS = 64, G = 4, T = 28	0.9976	13.7285	6227.9184	similar
	N = 35,480–41,920, SS = 64, G = 7, T = 16	0.9966	14.5904	5915.2791	similar
	N = 35,480–41,920, SS = 64, G = 8, T = 14	0.9970	13.6615	5569.3958	similar
	N = 35,480–41,920, SS = 64, G = 14, T = 8	0.9946	13.1908	5102.1465	similar
	N = 35,480–41,920, SS = 64, G = 16, T = 7	0.9947	12.3994	4850.3314	similar
FFTW, Problem Size, RAPL, HCLServer03	N = 30,720–34,816, SS = 64, G = 1, T = 112	0.9986	25.0459	81473.9811	similar
	N = 30,720–34,816, SS = 64, G = 2, T = 56	0.9984	10.9418	3060.2949	similar
	N = 30,720–34,816, SS = 64, G = 4, T = 28	0.9840	9.3161	3639.7016	similar
	N = 30,720–34,816, SS = 64, G = 7, T = 16	0.9945	14.8833	3013.0125	similar
	N = 30,720–34,816, SS = 64, G = 8, T = 14	0.9942	16.1527	2955.8465	similar
	N = 30,720–34,816, SS = 64, G = 14, T = 8	0.9912	16.3173	2595.7632	similar
	N = 30,720–34,816, SS = 64, G = 16, T = 7	0.9917	42.8061	2416.1999	similar
FFTW, Problem Size, RAPL, HCLServer03	N = 20,480–26,560, SS = 64, G = 1, T = 112	0.9996	25.6272	86,911.0622	similar
	N = 20,480–26,560, SS = 64, G = 2, T = 56	0.9994	9.7569	1502.1569	similar
	N = 20,480–26,560, SS = 64, G = 4, T = 28	0.9985	7.5984	1324.9766	similar
	N = 20,480–26,560, SS = 64, G = 7, T = 16	0.9549	17.7605	1611.0385	opposite
	N = 20,480–26,560, SS = 64, G = 8, T = 14	0.9427	21.5876	1806.2769	similar
	N = 20,480–26,560, SS = 64, G = 14, T = 8	0.6497	28.1641	2012.2371	similar
	N = 20,480–26,560, SS = 64, G = 16, T = 7	0.6010	30.9980	2163.2482	opposite
AnMoHA [2.5% precision]					
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01	M = 12,800–20,224, N = 20,224, SS = 128	0.9750	2.2414	3398.2869	same
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01	M = 12,800–20,480, N = 20,480, SS = 256	0.9383	3.0720	3803.1504	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01	M = 12,800–20,736, N = 20,224, SS = 256	0.9739	3.8751	4349.8794	same
FFT, Problem Size, HCLWattsUp_Combined, HCLServer01	M = 15,104–18,688, N = 23,552, SS = 64	0.9785	4.3116	3806.0146	same
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer02	M = 16,384–20,096, N = 22,528, SS = 128	0.9504	2.1708	956.8094	similar
FFT, Problem Size, HCLWattsUp_Combined, HCLServer02	M = 21,504–25,600, N = 25,600, SS = 64	0.9387	4.8698	2252.6855	similar
AnMoHA Less Accurate (10%rercision)					

DGEMM, Problem Size, HCLWattsUp_Parallel, HCLServer01	M = 12,800–20,224, N = 20,224, SS = 128	0.9134	6.8688	9937.8680	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01		0.9216	7.7965	10,996.5086	similar
DGEMM, Problem Size, HCLWattsUp_Parallel, HCLServer01	M = 12,800–20,480, N = 20,480, SS = 256	0.9094	5.9996	6914.4728	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01		0.9023	8.9318	9266.7659	similar
DGEMM, Problem Size, HCLWattsUp_Parallel, HCLServer01	M = 12,800–20,736, N = 20,736	0.9110	8.1647	8258.8659	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01		0.9456	7.5815	7766.5003	similar
FFT, Problem Size, HCLWattsUp_Parallel, HCLServer01	M = 15,104–18,688, N = 23,552, SS = 64	0.7930	4.4804	2080.0339	similar
FFT, Problem Size, HCLWattsUp_Combined, HCLServer01		0.8625	3.0740	1516.4442	similar
DGEMM, Problem Size, HCLWattsUp_Parallel, HCLServer01	M = 16,384–20,096, N = 22,528, SS = 128	0.9308	8.4808	7966.4267	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01		0.9010	8.1606	7241.1619	similar
FFT, Problem Size, HCLWattsUp_Parallel, HCLServer01	M = 21,504–25,600, N = 25,600, SS = 64	0.8667	13.4232	7170.1068	similar
FFT, Problem Size, HCLWattsUp_Combined, HCLServer01		0.9010	8.1606	7241.1619	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01	M = 64–20,288, N = 10,112, SS = 64	0.9963	8.0723	7060.6467	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer02		0.9994	5.0213	1184.0506	similar
DGEMM, Problem Size, HCLWattsUp_Combined, HCLServer01, HCLServer02		0.9972	7.6614	7598.0246	similar
FFT, Problem Size, HCLWattsUp_Combined, HCLServer01	M = 1024–10,160, N = 51,200, SS = 16	0.9991	14.5338	6279.0562	similar
FFT, Problem Size, HCLWattsUp_Combined, HCLServer02		0.9971	5.7143	147.5044	similar
FFT, Problem Size, HCLWattsUp_Combined, HCLServer01, HCLServer02		0.9992	14.0973	6383.0253	similar

References

1. Berndt, D.J.; Clifford, J. Using Dynamic Time Warping to Find Patterns in Time Series. In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining. Seattle, WA, US 1994; p. 359–370.

2. Li, B.; Han, L. *Distance Weighted Cosine Similarity Measure for Text Classification. Intelligent Data Engineering and Automated Learning – IDEAL 2013*; Yin, H.; Tang, K.; Gao, Y.; Klawonn, F.; Lee, M.;Weise, T.; Li, B.; Yao, X., Eds.; Springer: Berlin, Heidelberg, Germany, 2013; pp. 611–618.
3. Nakamura, T.; Taki, K.; Nomiya, H.; Seki, K.; Uehara, K. A Shape-Based Similarity Measure for Time Series Data with Ensemble Learning. *Pattern Anal. Appl.* **2013**, *16*, 535–548.
4. Anton, H.; Rorres, C. *Elementary Linear Algebra: Applications Version*; 11th ed.; John Wiley & Sons, Inc.: Hoboken, New Jersey, US, 2013.
5. Warren Liao, T. Clustering of Time Series Data-A Survey. *Pattern Recogn.* **2005**, *38*, 18571874.
6. Iglesias, F.; Kastner, W. Analysis of Similarity Measures in Times Series Clustering for the Discovery of Building Energy Patterns. *Energies* **2013**, *6*, 579–597.
7. SALKIND, N.J. *Encyclopedia of Measurement and Statistics Spurious Correlation*; Vol. 1, SAGE Publications, Inc.: Thousand Oaks, California, US, 2007.
8. Sanfeliu, A.; Fu, K. A Distance Measure between Attributed Relational Graphs for Pattern Recognition. IEEE Trans. Syst. Man Cybern. **1983**, *SMC-13*, 353–362.
9. Lin, C.L. Hardness of Approximating Graph Transformation Problem. In *Algorithms and Computation*; Du, D.Z.; Zhang, X.S., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, Germany, 1994; pp. 74–82.
10. Box, G.E.P.; Jenkins, G. *Time Series Analysis, Forecasting and Control*; Holden-Day, Inc.: San Francisco, CA, US, 1976.
11. Yu, Q.; Jibin, L.; Jiang, L. An Improved ARIMA-Based Traffic Anomaly Detection Algorithm for Wireless Sensor Networks. *Int. J. Distrib. Sens. Netw.* **2016**, *12*, 9653230.
12. Piccolo, D. A Distance Measure for Classifying Arima Models. J. Time Ser. Anal. **1990**, *11*, 153–164.
13. Ramoni, M.; Sebastiani, P.; Cohen, P. Bayesian Clustering by Dynamics. Mach. Learn. **2002**, *47*, 91–121.
14. Maharaj, E.A. Cluster of Time Series. Journal of Classification 2000, 17, 297–314.
15. Fahad, M.; Shahid, A.; Manumachu, R.R.; Lastovetsky, A. Supplementary Materials: A Novel Statistical Learning-Based Methodology for Measuring the Goodness of Energy Profiles of Applications Executing on Multicore Comptuing Platforms. Available online: https://csgitlab.ucd.ie/Muhammad_Fahad/supplementals-to-the-publications/-/tree/master/2020%2FMDPI_goodness (accessed on 25 July 2020).
16. Fahad, M.; Shahid, A.; Manumachu, R.R.; Lastovetsky, A. Supplementary Materials: A Novel Statistical Learning-Based Methodology for Measuring the Goodness of Energy Profiles of Applications Executing on Multicore Comptuing Platforms. Available online: https://csgitlab.ucd.ie/Muhammad_Fahad/supplementals-to-the-publications/-/tree/master/2020%2FMDPI_goodness (accessed on 25 July 2020).
17. Rotem, E.; Naveh, A.; Ananthakrishnan, A.; Weissmann, E.; Rajwan, D. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro* **2012**, *32*, 20–27.
18. Nvidia. NVML Reference Manual. Available online: https://docs.nvidia.com/pdf/NVML_API_Reference_Guide.pdf (accessed on 16 June 2020).
19. Corporation, I. Intel Xeon Phi Coprocessor SystemSoftware Developers Guide. Available online: https://software.intel.com/sites/default/files/managed/09/07/xeon-phi-coprocessor-system-software-developers-guide.pdf (accessed on 16 June 2020).
20. Corporation, I. Intelo Manycore Platform Software Stack (Intel MPSS). Available online: https://software.intel.com/ en-us/articles/intel-manycore-platform-software-stack-mpss (accessed on 16 June 2020).
21. Heterogeneous Computing Laboratory, University College Dublin. HCLWattsUp: API for Power and Energy Measurements Using WattsUp Pro Meter. Available online: https://csgitlab.ucd.ie/ucd-hcl/hclwattsup (accessed on 23 May 2020).
22. Gough, C.; Steiner, I.; Saunders,W. *Energy Efficient Servers: Blueprints for Data Center Optimization*; Apress: New York, NY, US, 2015.