Determination of OpenMP Runtime Profiler Parameters for Optimal Number of Threads

Tamara Dancheva, Marjan Gusev, Vladimir Zdravevski, Sashko Ristov
Ss. Cyril and Methodius University, Faculty of Computer Science and Engineering
1000 Skopje, Macedonia
Email: tamaradanceva19933@gmail.com, {marjan.gushev, vladimir.zdraveski}@finki.ukim.mk

***Abstract*—An optimization approach is used for dynamic configuration of the number of threads in an OpenMP environment. The approach analyzes the current runtime state and predicts the number of threads by use of a mix of profiling and machine learning techniques based on a random forest tool.**

In this paper we analyze the behavior of the tool, its performance and effectiveness in case of two algorithms, one that represents a highly parallel program and the other a program with a lot of data flow dependencies which prevent a high degree of parallelism. The experiments are conducted in two environments that determine the workload that concurrently is executed while testing, by determining a stress and no stress environment.

The outcome of the realized research includes a detailed analysis of datasets used in the optimization approach. The conclusions are that for a set of certain dataset attributes the random forest optimization tool does not degrade performance relative to the performance achieved by assigning the maximum number of cores as a number of threads. In addition, it provides better trade off performance for a set of certain dataset attributes.

I. Introduction

TBD

Additionally, the state of the runtime during the process cannot be known in advance to be taken in account when predicting the optimal number of threads. The dynamic nature of the runtime has a great impact on the performance, since the operating system has a considerable non-negligible autonomous role in making decisions for the management of low-level resources. Another advantage of using the tool proposed in this paper is that it is able to dynamically keep track of the runtime state and make real time decision on the number of threads based on it. It uses these additional runtime parameters to make decisions on the optimal number of threads.

TBD

The following assumptions are made concerning the hardware and the OpenMP algorithm we are optimizing:

A1 The algorithm is run on a multiprocessor machine.

A2 The cost of parallelism is amortized by the OpenMP algorithm for at least num_threads=2.

TBD

TBD The paper is organized according the following structure. Section II describes the optimization approach and Section III describes the methodology used for the experiments. The performance analysis of the measured results is presented in Section III-E. Section VI evaluates the results and gives detailed explanations. The related work is elaborated in Section VII. Section VIII contains the conclusions and future work.

II. Optimization Approach

The optimization approach is based on construction of forest trees with proper profiling.

A. Interposition

Each time omp_set_num_threads is called in the user program, the wrapper for this method contained in a preloaded shared library is invoked. The wrapper returns a new handle created in order to resume execution of the previous program. The parameter passed is the predicted value for the optimal number of threads.

To keep the simplicity of the approach we use the GNU compiler.

B. Profiling

TBD

TBD

C. Benchmarks and PThreads

TBD

OpenCV is an open source cross-platform library that exposes optimized parallel implementations (OpenMP for C++) of many algorithms including which is the random forest algorithm referenced as random trees [1]. The input dataset is a file on disk in the form of a comma separated table. The records contained in it are generated by the previous profiling. The dataset is split into a training set and a test set. After training the set, the recognition rate and the importance of the predictor variables is measured and the forest is saved as a file with a .forest extension. The generated forest is kept to be further used in predicting the number of threads for an input unknown runtime state.

D. OpenMP algorithm

TBD

III. Experimental Methodology

This section describes the methodology used in the experiments.

TABLE I
LIST OF ATTRIBUTES USED FOR PROFILING

num_threads	number of threads for one run
cpu_usage	cpu usage for all processes
pid_cpu_usage	cpu usage for the current process
ram_usage	ram usage for all processes
pid_ram_usage	ram usage for the current process
pid_page_faults	page faults for the current process
vm_usage	virtual memory usage for all processes
pid_vm_usage	virtual memory usage for the current process
pid_vm_read_usage	virtual memory reads percentage for the current process
pid_vm_write_usage	virtual memory writes percentage for the current process
processes	# of processes and threads created
procs_running	# of running processes and threads
procs_blocked	# of blocked processes and threads
user (current cpu and all cpus)	time(USER_HZ) spent on user space processes
sys (current cpu and all cpus)	time(USER_HZ) spent on kernel processes
nice (current cpu and all cpus)	time(USER_HZ) spent on user processes with priority other than default
idle (current cpu and all cpus)	idle percentage
wait (current cpu and all cpus)	waiting for an I/O
irq (current cpu and all cpus)	time spent servicing interrupts
soft_irq (current cpu and all cpus)	time spent servicing software interrupts
total_cpu_usage (current cpu and all cpus)	total cpu usage

A. Test goal

TBD

B. Dataset record constraints

The dataset attributes should have significant variance. The high variance is an indication of no information or in terms of information retrieval, high entropy, which ultimately indicates a disorder or a big number of possible states. For example, if we take one record, and start the classification with a parameter that has high entropy, there is a low probability the right child (leading to the optimal number of threads) of that node will be traversed in the next step. Therefore, this parameter will be at the bottom of the tree if it is included. However with a negligible variance across the column, it is unnecessary and it might affect the other variables by taking away some of their importance.

Another key constraint is that the variables/ columns taken in the dataset record should not be linearly dependent or approximately linearly dependent since it will affect the predictions negatively. Therefore ideally only a subset is chosen from Table I.

This machine learning approach most importantly enables us to generalize the concept and speculate about the model, aiming at getting good predictions by dynamic formula determination, being one of the key benefits from using this approach over statistical modeling. It requires no prior guesswork or intel on the specific algorithm that is to be run. It is essential though that the assumptions made in the introduction hold for any OpenMP algorithm we wish to optimize using this tool.

TABLE II
LIST OF OPTIONS AND ARGUMENTS FOR THE STRESS INVOCATIONS

# Invocation	**–cpu**	**–io**	**–vm-bytes**	**–vm-keep**	**–t**
# Stress 1	2	2	1260315000	-	1h
# Stress 2	1	1	1260315000	-	1h

C. Boosting and random forest

TBD

D. Test procedure

TBD

Testing is performed by executing 50-100 runs of each of the the selected benchmarks. Each test run consists of an execution for a specific number of threads in the range [1, max_num_cores]. The state and number of threads associated with the best performance is written in the dataset used to train and create the forest.

TBD

TBD

E. Performance analysis

The overall performance is analyzed in environments with and without stress. The stress environment is simulated using the stress program package for POSIX systems that imposes stress on the system (CPU, memory, I/O, disk stress). Table II presents the options and the corresponding arguments used in the experiments' stress invocations.

The main challenge consists of verifying that the tool is able to correct the number of threads based on the dynamic runtime state yielding better performance. To improve this functionality and we realize a thorough evaluation and analysis of two algorithms that exhibit significantly different resource utilization patterns. Executions of these algorithms result in different runtime states caused by the stress and no stress environments.

The results from the conducted experiments will prove the validity of both the H1 and H2 hypothesis. Additionally, the characteristics of a good choice of attributes will be explored by analyzing the attribute subsets that yield performance worse than the optimal one.

Three experiments are conducted as OpenMP default schemes and four more experiments using the tool with different dataset. To evaluate the speedup Sp of using the tool and the default configuration we compare the elapsed times with corresponding indexes according to (1).

$$Sp(\#\ threads, Dataset\ \#) = \frac{Time_{\#\ threads}}{Time_{Dataset\ \#}} \quad (1)$$

For example, the speedup obtained of using the dataset 2 over the default OpenMP scheme using 4 threads is denoted by $Sp(4T, D2)$.

The experiments will be realized with two different algorithms Algorithms I and II (Alg1 and Alg2). The *algorithm ratio* R_a will be used to compare their performances by comparing their speedups according to (2), using the speedup determined by (1).

$$R_a(\#\ threads, Dataset\ \#) = \frac{Sp_{Alg1}}{Sp_{Alg2}} \tag{2}$$

The *environment ratio* R_e will be used to compare the speedups obtained for experiments in no stress and stress environments by calculating (3).

$$R_e(\#\ threads, Dataset\ \#) = \frac{Sp_{No\ Stress}}{Sp_{Stress}} \tag{3}$$

Datasets have different attributes that can be characterized by a negligible variability, calculated as relative standard deviation RSD using (4), besides the mean values and standard deviation SD. It is the ratio of the corrected sample standard deviation to the mean of the sample. The smaller this ratio is, the closer the attribute values are clustered around the mean.

$$RSD = \frac{\sqrt{\sum_1^n \frac{(x-\bar{x})^2}{n-1}} * 100}{\bar{x}},\ \bar{x} > 0 \tag{4}$$

IV. Performance evaluation in the No Stress environment

The environment is classified as a *no stress environment*, representing an environment with no additional workloads running except the benchmarking workload that generates the random forest and then the OpenMP algorithm using the random forest from the previous step to dynamically optimize the number of threads. The same forests are used both to test the performances of two algorithms.

A. Analysis of the generated random forests

Tables III - VI present the attributes with highest importance using 4 different datasets to capture the state of the runtime at a given moment (given in Table **??**). These tables will later be used in the analysis of the performance of the algorithms and the quality of the predictions made by the tool.

B. Algorithm I

Algorithm I is an easily parallelized algorithm, that conducts exhaustive search to find a combination of input gates to produce logical circuit with output 1 [2]. Therefore, a logical conclusion is that using the maximum number of threads yields better performance.

TBD

The performance is obtained using the tool to dynamically adjust the number of threads across the first 20 test runs of Algorithm I.

TBD

TBD

TBD

Table VII presents the average values of elapsed times for executing the Algorithm I.

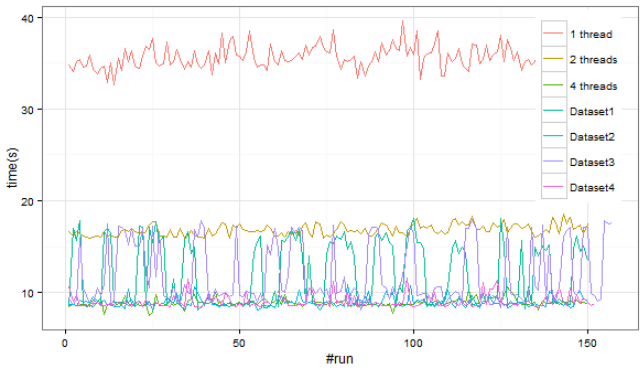

Fig. 1. Performance of executing the Algorithm I in a no stress environment.

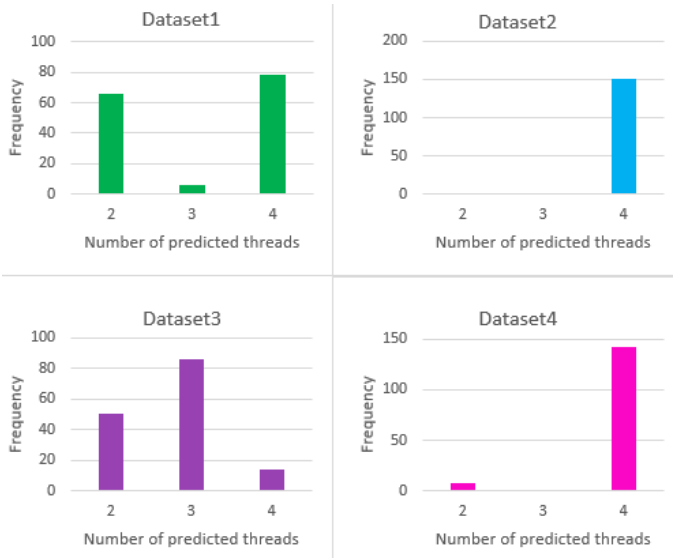

Fig. 2. Histograms for the 4 datasets for executing Algorithm I in a no stress environment.

Due to different implementations of OpenMP policies when adjusting the number of threads automatically, for a machine with 2 physical cores and 4 threads, there are two common default values for the OpenMP num_threads control variable, 2 or 4.

TBD

TBD

Dataset 2 and Dataset 4 differ in pid_ram_usage, nice, wait and soft_irq (Table IX). These attributes are present in Dataset 4 and eliminated in Dataset 2, characterized by either negligible variability RSD, by pid_ram_usage and soft_irq or by a small number of entries that contain information resulting in a very high RSD. Note that the > 0 column represents the number of entries with values greater than 0.

The no stress environment means that while the measurements were conducted there were no other user processes running. Therefore, the percentage of processes running with changed priority is almost always zero and unlikely to vary during the profiling. The data needed for execution

TABLE III
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A NO STRESS ENVIRONMENT USING DATASET 1 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
lst_cpu_usage	83.05311	28.12959	33.8694	310	[0, 99.96459]
ram_usage	0.06854966	0.00213278	3.111292	310	[0.066359, 0.078121]
pid_ram_usage	97127183	2889136	2.97459	310	[85148548, 100241835]
proc_blocked	0.9977936	0.00738197	0.739829	310	[0.95092, 1]
user	0.743736	0.3547	47.69164	294	[0, 1]
sys	0.069683	0.111046	159.359	147	[0, 0.522727]

TABLE IV
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A NO STRESS ENVIRONMENT USING DATASET 2 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
pid_cpu_usage	22.9753	7.799845	33.94883	306	[0,34.56392]
ram_usage	0.8537368	0.002282	0.2672901	310	[0.844875, 0.862207]
user	0.960583	0.083141	0.086553	310	[0.45, 1]
sys	0.036072	0.082209	227.9018	106	[0, 0.55]
proc_blocked	0.937261	0.007322	0.2672901	310	[0.903955, 0.94186]

TABLE V
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A NO STRESS ENVIRONMENT USING DATASET 3 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
pid_cpu_usage	29.63281	9.439129	31.85364	309	[0, 40.25845]
lst_cpu_usage	98.80933	1.658517	1.678502	310	[76.84211, 99.9871]

TABLE VI
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A NO STRESS ENVIRONMENT USING DATASET 4 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
cpu_usage	69459731	1262968	1.818274	310	[63580746, 71519130]
pid_cpu_usage	0.095976	0.001802	1.877526	310	[0.093181, 0.104815]
lst_cpu_usage	36.80855	9.706447	26.37009	310	[4.656863, 50.24752]
ram_usage	0.675817	0.260094	0.739829	308	[0, 1]
user	0.323522	0.260420	80.49529	218	[0, 1]
sys	0.518930	0.314664	60.63711	293	[0, 1]
idle	0.154639	0.152510	98.62325	249	[0, 0.880952]

TABLE VII
AVERAGE ELAPSED TIMES FOR ALGORITHM I IN A NO STRESS ENVIRONMENT (FIG. 1)

	Mean elapsed time	SD	RSD
1 Thread	35.65783	1.324917	03.71564
2 Threads	16.79859	0.600862	03.57686
4 Threads	08.89994	0.432442	04.85893
Dataset 1	11.95526	3.462988	28.96622
Dataset 2	08.95332	0.538926	06.01928
Dataset 3	12.16092	3.295266	27.55028
Dataset 4	09.00934	0.668523	07.42034

TABLE VIII
SPEEDUPS FOR EXECUTING THE ALGORITHM I IN NO STRESS ENVIRONMENT

	Dataset1	Dataset2	Dataset3	Dataset4
1 Thread	2.982606	3.982636	2.932165	3.957874
2 Thread	1.405121	1.876241	1.381358	1.864575
4 Thread	0.744437	0.994037	0.731847	0.987857

of Algorithm I does not put too much stress on the Ram (pid_ram_usage has low RSD) and is not I/O intensive. Since no other processes that are I/O intensive are running, the wait and soft_irq (I/O requests) are also unnecessary.

Therefore, the elimination results in truncating unnecessary trees or branching of the logical decision making tree. Finally, it leads to better performance results.

This confirms that the variance is an important factor and it needs to be further utilized in developing an algorithm for automatic configuration of the parameters used for the profiling.

Dataset 1 consists of all attributes and contains variables that are linearly dependent such as the following:

- pid_vm_usage is approximately equal to the sum of the pid_vm_read_usage and the pid_vm_write_usage
- user, sys, nice, idle, wait, irq, soft_irq are used for calculating the current total term in the cpu usage calculation
- processes ˜procs_running and procs_blocked

Similarly, the attributes in Dataset 3 are mostly not significant as confirmed in Table V presenting the statistics of the more significant attributes included. Consequently, these datasets produce ambiguous results in this simple base case scenario. The values for the number of threads prevalent in the predictions for Dataset1 and Dataset3 in Fig. 2 do not

TABLE IX
RELATIVE STANDARD DEVIATION OF PROBLEMATIC ATTRIBUTES IN DATASET 4 FOR ALGORITHM I IN A NO STRESS ENVIRONMENT (TOTAL OF 500 ENTRIES)

Attribute	Mean	SD	RSD	> 0	Range
pid_ram_usage	69259670	1297396	1.87324	500	[63274627, 71519130]
nice	0.00174	0.01372	787.649	11	[0, 0.17073]
wait	0.00020	0.00447	2238.303	1	[0,0.1]
soft_irq	0	0	NaN	0	[0,0]

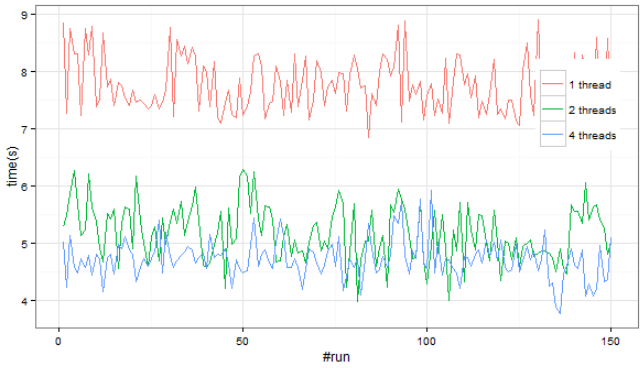

Fig. 3. Performance of executing the Algorithm II in a no stress environment with default OpenMP schemes

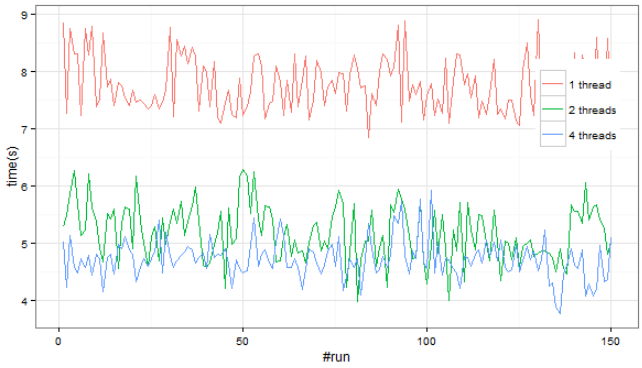

Fig. 4. Performance of executing the Algorithm II in a no stress environment using the tool with Datasets 1 - 4

correspond well with the reference optimal number of threads in this setting, that is 4, confirming this conclusion.

To make a proper conclusion about situations to be avoided and develop an intuition which subsets make up a good choice of parameters we will also experiment with another algorithm that has different parallelization properties.

C. Algorithm II

Algorithm II is an OpenMP implementation of the Fast Fourier transform, which exploits hyperthreading considerably less than Algorithm I.

The results obtained with the Algorithm II using the OpenMP default scheme are presented in Fig. 3 and for executing the tool with Datasets 1 - 4 in Fig. 4

TBD

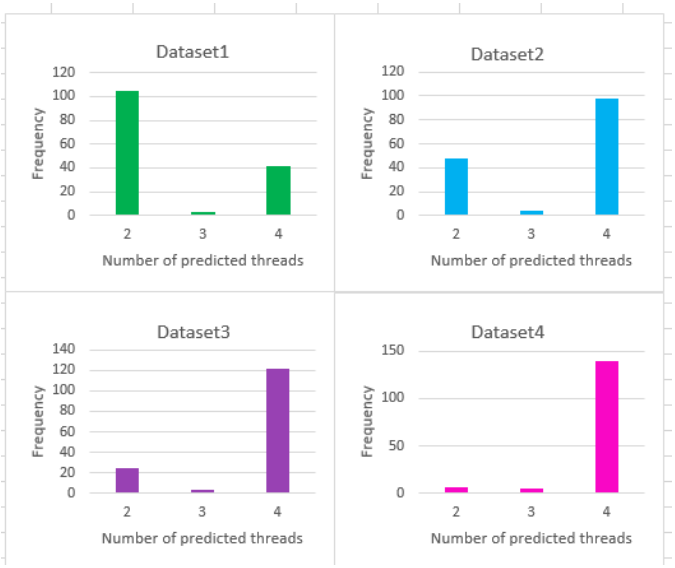

Fig. 5. Histograms of predicted threads for executing Algorithm II in a no stress environment.

TABLE X
AVERAGE ELAPSED TIMES FOR ALGORITHM II IN A NO STRESS ENVIRONMENT

	Mean elapsed time	SD	RSD
1 Thread	7.752695	0.479097	6.179750
2 Threads	5.192298	0.487749	9.393697
4 Threads	4.724949	0.334973	7.089452
Dataset 1	4.831201	0.330802	6.847199
Dataset 2	4.770104	0.372229	7.803371
Dataset 3	4.640983	0.385242	8.300873
Dataset 4	4.556192	0.408120	8.957487

TABLE XI
SPEEDUPS FOR EXECUTING THE ALGORITHM II IN NO STRESS ENVIRONMENT

	Dataset1	Dataset2	Dataset3	Dataset4
1 Thread	1.604713	1.625267	1.670486	1.701573
2 Threads	1.074743	1.088508	1.118793	1.139614
4 Threads	0.978007	0.990534	1.018092	1.037039

Table X presents the average elapsed times of executing the Algorithm II in both the OpenMP default scheme and tool experiments.

The obtained values of speedups for executing the Algorithm II in the no stress environment are displayed in Table XI, along with corresponding values of SD and RSD.

A comparison of the OpenMP default scheme with the tools

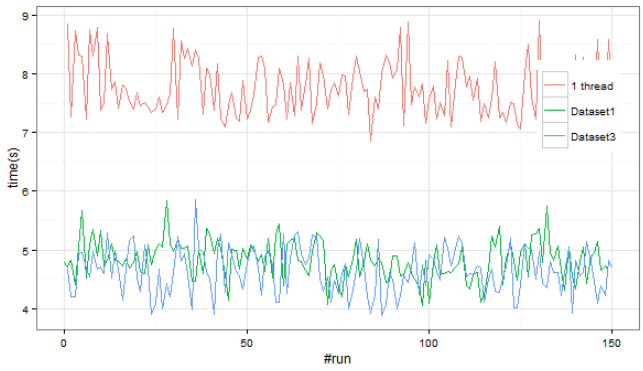

Fig. 6. Performance of executing the Algorithm II in a no stress environment - comparison: Dataset 1 and 3 vs OpenMP 1 thread

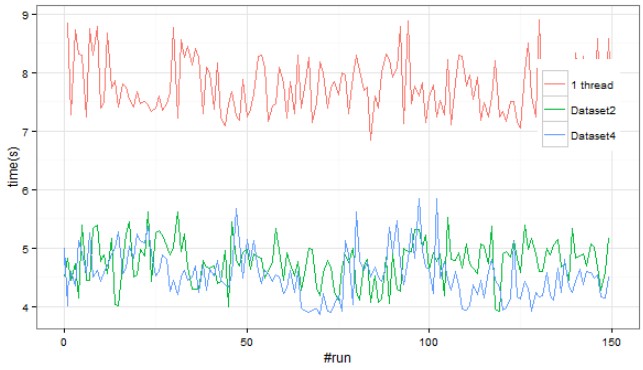

Fig. 7. Performance of executing the Algorithm II in a no stress environment - comparison: Dataset 2 and 4 vs OpenMP 1 thread

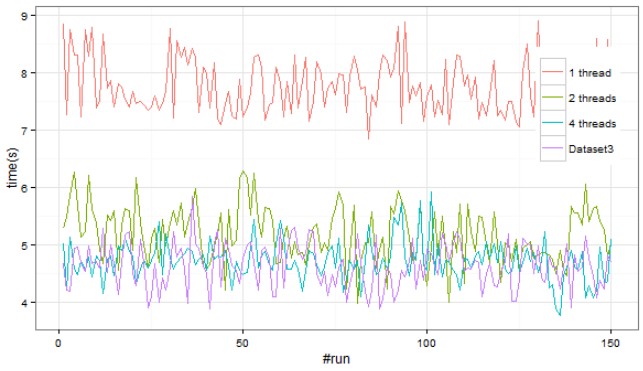

Fig. 8. Performance of executing the Algorithm II in a no stress environment - comparison: Dataset 3 vs OpenMP schemes

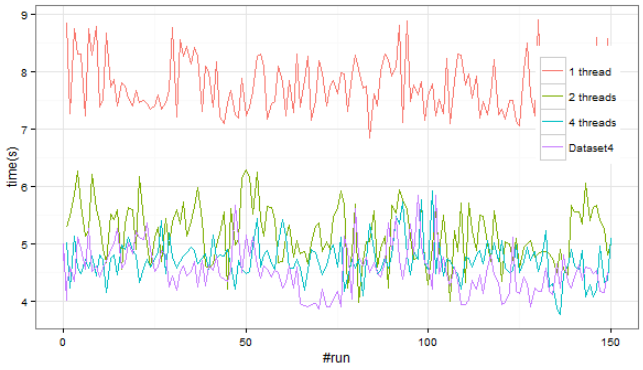

Fig. 9. Performance of executing the Algorithm II in a no stress environment - comparison: Dataset 4 vs OpenMP schemes

described by datasets in Table **??** are presented in Fig. 6 and Fig. 7.

Fig. 6 and Fig. 7 present a lot of oscillations in the prediction results generally in all the datasets. The difference between them can be analyzed more thoroughly by examining the histograms showing the number of predicted threads for each of the four datasets, as shown in Fig. 5.

One can conclude that Dataset 1 exhibits preference for 2 threads, while the rest of the datasets exhibit preference for 4 threads. From the results with 1, 2 and 4 threads, the scheme with 4 threads produces the best performance even though with slight difference compared to the execution times when using 2 threads Fig. 9. If we again consider 4 as the reference optimal number of threads for this algorithm in no stress environment, then Dataset1 produces poor predictions while Dataset 4 produces the most precise and accurate predictions, configuring the optimal number of threads (Fig. 9 where it can be seen that Dataset4 mostly lies in the lower range covered by the OpenMP default scheme for 2 and 4 threads).

TBD

However, the average elapsed time for this algorithm is generally not a good indicator of the whether the predictions optimize the performance since the intervals [mean-sd, mean +sd] of the measurements listed in Table VII all overlap with each other except for 1 thread. Therefore the graphs along with the significant attributes should be analyzed in order to determine which dataset results in the best optimization of the algorithm.

One can observe that the Dataset 3 curve lies mostly in the region below the Dataset 1 curve, as presented in Fig. 6. This leads to a conclusion that Dataset 3 yields better performance than Dataset 1, which corresponds to the average in this case. Additionally, Dataset 3 results in a slightly better performance than the best performing OpenMP scheme for a significant number of points (Fig. 8) and in comparison with the other datasets.

Therefore, Dataset 3 and Dataset 4 produce the best results which happen to coincide with the minimum average elapsed times (Table VII).

TBD

Tables III-VI show that Dataset 1 and Dataset 4 contain the largest number of attributes with good results for the RSD, meaning that a considerable portion of the input possible values for those attributes has been covered by the forest training set. Since this fact does not result in Dataset 1 yielding the optimal number of threads, it is shown that the relation and

dependencies between the attributes have a great impact on the output number of threads as well.

Since Dataset 4 maps to the optimal number of threads more precisely, we can assume that the resource utilization patterns exhibited by running the Algorithm II in a no stress environment, is correctly mapped because the attributes in Dataset 4 and their coverage correspond with the key attributes that determine the state of the runtime.

From the analysis of the Algorithm II for a higher number of threads, the idle time, ram and cpu usage are negatively impacted. Also another very important factor is the sys attribute, which is caused by the kswapd process responsible for the swap space which is used to alleviate the ram usage. This also applies to Dataset 2 (Table IV) and Dataset 3 (Table V). During the training (running the benchmarks concurrently), it covers all possible values VI. In this scenario the sys attribute featured in the states for which a prediction is made, is not significant, however it will be later observed that in a stress environment this attribute plays a significant part in determining the optimal number of threads.

The results conducting the experiment using the Algorithm II (Fig. 3) suggest that using 4 threads in comparison with 2 threads does not yield a considerable speedup, concluded by the corresponding R_a calculated by (2). The main reason for this behavior is the flow dependency introduced by the shared variables in each step of the algorithm and the resource limitations.

Slight optimization to alleviate the problem with the flow dependency is achieved using variable renaming and loop fusion. The resource sharing patterns when increasing the number of threads more than the number of physical cores either aggravates the state impeding the already limited potential performance boost (can be spotted on the graph in the points where 4 threads overlaps 2 threads), results in similar performance as in the case of using 2 threads or slightly boosts performance.

V. Performance evaluation in the stress environment

The stress environment is characterized with coexistence of other workloads in the operating environment. This is actually an environment where the largest number of mismatches arise between the optimal number of threads and the OpenMP default scheme number of threads, because of the deviation of the optimal number of threads. Our task will be to distinguish the most extreme scenario and evaluate the side effects.

A. Analysis of the generated random forests

Tables XII - XV present the attributes with the highest importance using 4 different datasets of attributes to capture the runtime state according to the definitions in Table **??**. The experiments are realized in a stress environment with 2 additional workloads running concurrently to the benchmarking to generate the random forest and the OpenMP algorithm using the random forest from the previous step to dynamically optimize the number of threads. The same forests are used both to test the performance of Algorithms I and II.

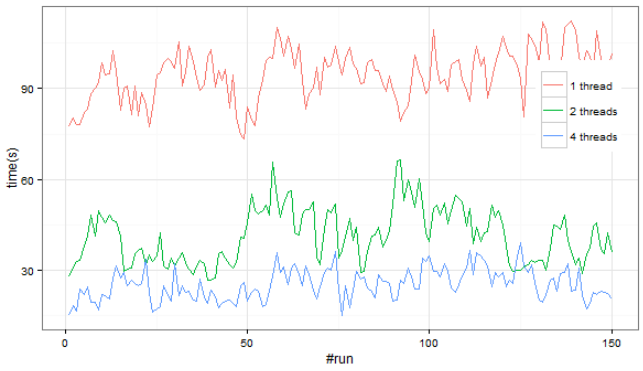

Fig. 10. Performance of executing the Algorithm I in a stress environment with OpenMP default schemes

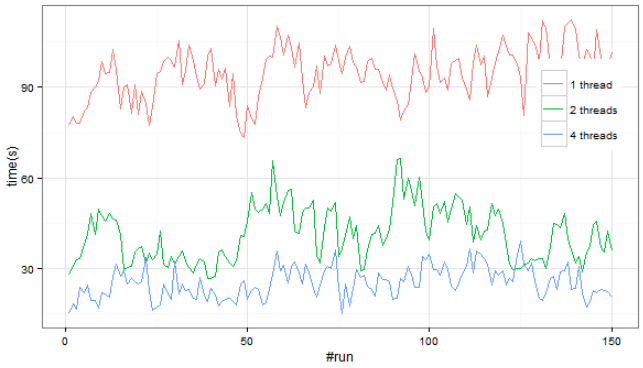

Fig. 11. Performance of executing the Algorithm I in a stress environment with Datasets 1 -4

These tables will later be used in the analysis of the performance of the algorithms and the quality of the predictions made by the tool.

A general observation that can be made is that compared with the statistics of the forests generated in the no stress environment, the attributes' range increases. This is a positive change because more input states are being covered during the training, meaning that the chances of obtaining good predictions are increased for an unknown input i.e runtime state provided that the attributes satisfy the requirement discussed in the dataset constraints subsection.

B. Algorithm I

TBD

TBD

TBD

The average values of elapsed times for executing the Algorithm I in a stress environment is presented in Table XVI.

The corresponding speedups calculated for executing the Algorithm I in a stress environment are presented in Table XVII.

TABLE XII
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A STRESS ENVIRONMENT USING DATASET 1 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
cpu_usage	0.9984582	0.024412	2.445028	310	[0.571429,1]
lst_cpu_usage	11.12539	4.672085	41.9948	306	[0, 31.43638]
pid_cpu_usage	98.99668	1.432277	1.446793	310	[82.11806, 99.99292]
pid_ram_usage	0.4153416	0.035384	8.51931	310	[0.061474, 0.485284]
proc_blocked	0.9927251	0.012257	1.234737	310	[0.952096, 1]
user	0.948838	0.117084	12.3397	310	[0.219512, 1]
sys	0.044814	0.114837	256.2495	119	[0, 0.780488]

TABLE XIII
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A STRESS ENVIRONMENT USING DATASET 2 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
cpu_usage	0.995936	0.044914	4.509797	310	[0.058824,1]
pid_cpu_usage	26.2557	9.829089	37.43601	308	[0,49.95054]
lst_cpu_usage	98.95641	4.493973	4.541366	310	[14.64247,99.98321]
ram_usage	0.7930204	0.1831631	23.09689	310	[0.084097, 0.857526]
procs_blocked	0.949784	0.017286	1.820039	310	[0.872093,1]
user	0.931213	0.096705	10.38487	310	[0.044118, 1]
sys	0.261951	0.84265	227.9018	31.086572	[0, 0.512195]
idle	0.002848	0.044648	1567.642	5	[0, 0.941176]
irq	0.002848	0.044648	1567.642	8	[0, 0.941176]

TABLE XIV
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A STRESS ENVIRONMENT USING DATASET 3 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
pid_cpu_usage	39.30662	17.0454	43.3652	304	[0, 51.5144]
lst_cpu_usage	99.9467	30.28789	97.41707	310	[76.84211, 99.9752]
idle	0.2148448	0.356523	165.9447	146	[0,1]

TABLE XV
CHARACTERISTIC VALUES OF SELECTED ATTRIBUTES IN A STRESS ENVIRONMENT USING DATASET 4 (TOTAL OF 310 ENTRIES)

Attribute	Mean	SD	RSD	>0	Range
pid_ram_usage	0.042112	0.070324	166.9899	310	[0.02472882, 0.3583614]
ram_usage	0.799170	0.184730	23.1153	310	[0.05959, 0.863556]
pid_cpu_usage	23.71295	9.680669	40.8244	307	[0, 49.9007
cpu_usage	0.996638	0.030904	3.100786	310	[0.547619, 1]
lst_cpu_usage	36.80855	9.706447	26.37009	310	[4.656863, 50.24752]
procs_blocked	0.941616	0.014994	1.592347	310	[0.910112, 1]
user	0.9740171	0.058782	6.035018	310	[0.47619, 1]
sys	0.02062797	0.4782458	228.5833	109	[0, 0.4]

TABLE XVI
AVERAGE ELAPSED TIMES FOR ALGORITHM I IN A STRESS ENVIRONMENT
-

	Mean elapsed time	SD	RSD
1 Thread	94.67928	8.482138	8.958811
2 Threads	41.3563	8.996143	21.75277
4 Threads	25.33419	5.090215	20.09228
Dataset 1	39.43302	11.41166	28.93936
Dataset 2	25.10371	3.880403	15.45748
Dataset 3	40.83514	11.68102	28.60531
Dataset 4	31.2954	5.083857	16.24474

TABLE XVII
SPEEDUPS FOR EXECUTING THE ALGORITHM I IN THE STRESS ENVIRONMENT

	Dataset1	Dataset2	Dataset3	Dataset4
1 Thread	2.401015	3.771525	2.318574	3.025342
2 Threads	1.048773	1.647418	1.012763	1.321481
4 Threads	0.541023	0.849842	0.522447	0.681704

TBD However due to the relatively high RSD expected in a stress environment (Table XVI), this reference optimal value is an issue to be more carefully examined.

The comparison of results obtained with the execution of Algorithm I using the tool to optimize the number of threads are presented in Fig. 13 - Fig. 16.

The predictions for Dataset 1 vary mostly between 2 and 4 threads (Fig. 12), generating oscillating performance in the range covered by the default scheme for 2 and 4 threads. There are slight exceptions corresponding to the overlapping errors derived from the standard deviation of the sample (Fig. 13).

Comparing the predictions with the optimal number of threads, i.e the maximum number of threads, Dataset 1 does

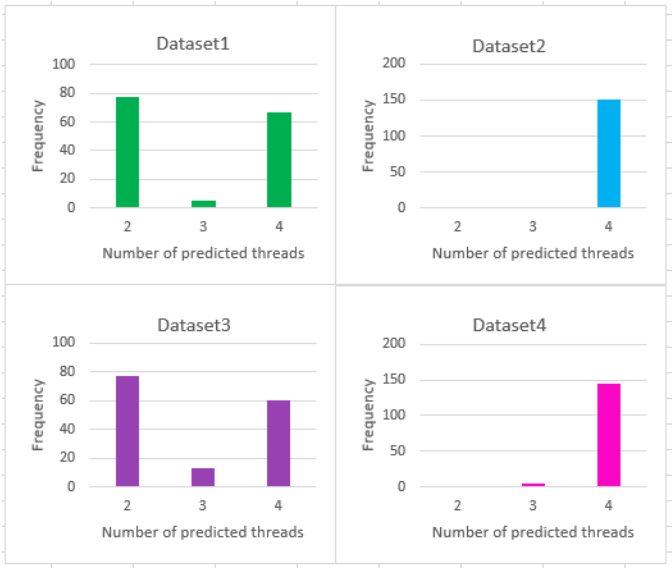

Fig. 12. Histogram of predicted threads for executing the Algorithm I in a stress environment

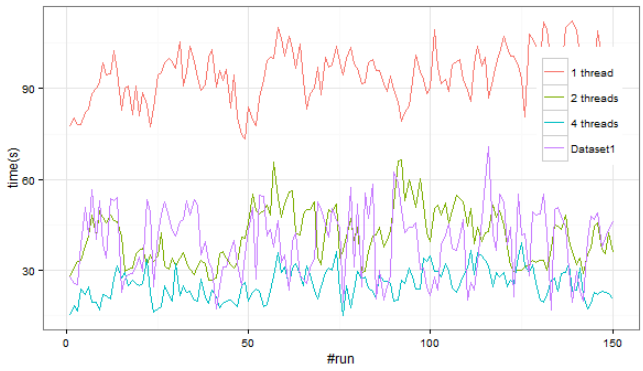

Fig. 13. Performance of executing the Algorithm I in a stress environment - Comparison OpenMP schemes vs Dataset 1

not generate good quality results for this algorithm in this setting. The statistics for the attributes included in Dataset 1 (Table XII) and Fig.. **??** show that even though the attributes span a significant range of input values with a high coefficient of variation or RSD, there are linear dependencies between the attributes. They often manifest their negative influence in generating predictions inconsistent with the optimal number of threads in this case.

Dataset 2 produces optimal results, always assigning the maximum number of threads. It results in a performance similar to the one with the OpenMP default scheme with 4 threads (Fig. 12). The success can be justified firstly by the independence of the attributes in correlation with each other and their wide coverage of input values. Additionally this dataset produces better results in a stress environment than in a no stress environment for the same algorithm.

In the stress environment, procs_blocked, sys gain importance due to the inflicted stress and the intensive processing

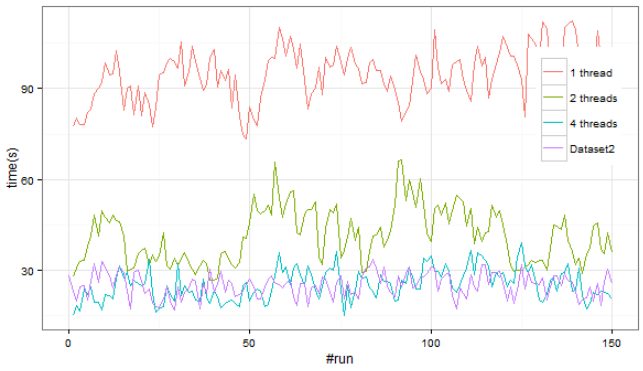

Fig. 14. Performance of executing the Algorithm I in a stress environment - Comparison OpenMP schemes vs Dataset 2

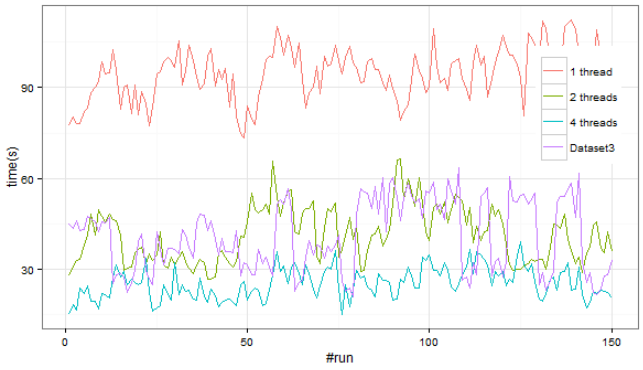

Fig. 15. Performance of executing the Algorithm I in a stress environment - Comparison OpenMP schemes vs Dataset 3

that is required of the system. The idle and irq attributes also gain importance although to a smaller degree. Consequently, this results in a random forest consisting of better decision trees, that result in more accurate optimization of the number of threads.

The performance of Dataset 2 is concentrated in the lower regions mainly within the range of the performance achieved with OpenMP default scheme with 4 threads (Fig. 14).

The frequency of the predicted number of threads for Dataset 3 closely resembles the frequency of the predicted number of threads for Dataset 1 Fig. 12. The inaccuracy of the predictions in this case comes from the fact that very few of the attributes are important enough in the decision making. The variation of the execution time spanning the whole range of 2 threads and four threads combined in Figure. 15 confirms this conclusion, which is further quantified in Table XVI with high SD and RSD values for Dataset 3.

The performances of executing the Dataset 4 are compared with OpenMP schemes in Fig. 16/

Similarly to Dataset 2, the Dataset 4 produces optimal predictions for this algorithm in the majority of cases, reverting to 3 threads in 4 cases. The problematic values for Dataset 4 listed in Table IX in a stress environment take on non-

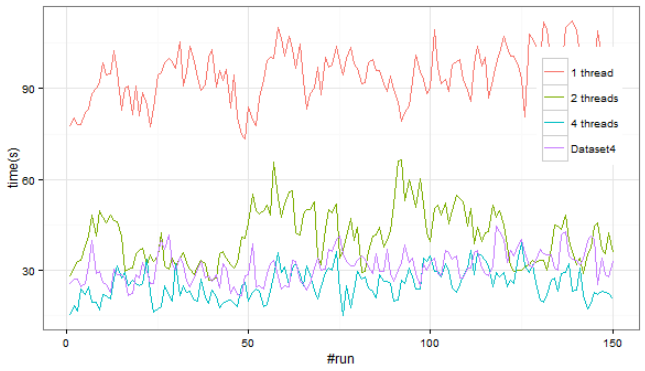

Fig. 16. Performance of executing the Algorithm I in a stress environment - Comparison OpenMP schemes vs Dataset 4

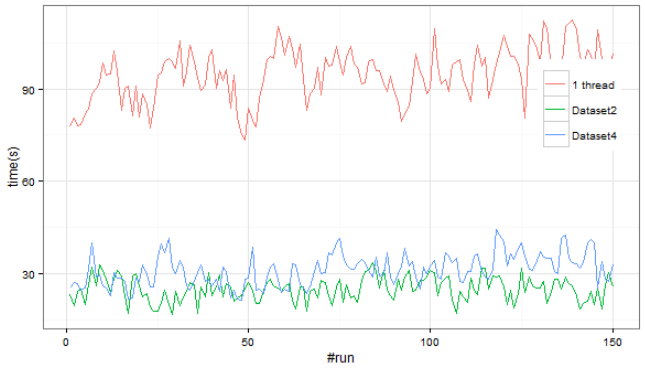

Fig. 17. Performance of executing the Algorithm I in a stress environment - Comparison of Dataset 2 and 4 vs 1 thread

negligible RSD, and gain importance.

This results in a slightly larger speedup than using Dataset 4 in a no stress environment for the same algorithm.

The difference between the performance achieved by using Dataset 2 versus Dataset 4 can be seen in Fig. 12 and graphically in Fig. 17. Dataset 2 outperforms Dataset 4 in this setting for most of the input states, which is additionally reflected in the average given in Table XVI.

One can conclude that Dataset 1 and Dataset 3 are relatively more uniformly distributed than Dataset 3 and Dataset 4 from the histograms representing the execution time for each of the datasets in Fig. 18. One possible explanation for this is that the regular oscillations between 2 and 4 threads result in execution times distributed in a way that spans the range of the two of them as regularly as the stress environment allows this (hence the uneven variations from interval to interval, closely clustered nevertheless).

The distributions of Dataset 2 and Dataset 4 resemble normal distribution, having a local maximum as a result of the predictions favoring one class, that is 4 threads in this case for both Dataset 2 and Dataset 4. The low SD and RSD values for Dataset 2 and Dataset 4 in Table XVI correspond with the prevalence of 4 threads in the predictions.

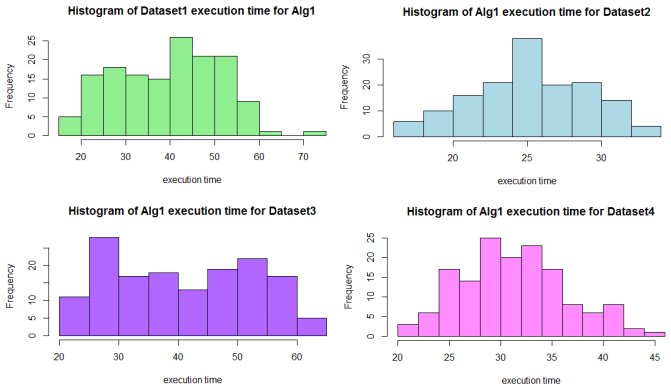

Fig. 18. Histogram of execution times for executing the Algorithm I in a stress environment for Datasets 1-4

C. Algorithm II

Running the Algorithm II in stress environment presents a challenging optimization problem for the tool since all of the following three scenarios are likely to figure during testing:

- The runtime is under a lot of stress, using hyperthreading will likely decrease performance since the performances with 2 and 4 threads in OpenMP default scheme are so close together (in contrast with Algorithm I, where the speedup obtained using hyperthreading is almost equal to the maximum theoretical speedup equal to the number of cores with 100% efficiency). In this case the tool should not use hyperthreading in order to boost performance in this situation.
- Blocking processes/threads and OS scheduling mechanisms have made space for hyperthreading to boost performance. In this case the tool should use hyperthreading in order to boost performance.
- Since there exist more overlapping regions between 2 threads and 4 threads under stress than otherwise, another possibility is that it is not possible to determine with high accuracy whether 2 or 4 threads will result in better performance. In this case any choice of the tool will be optimal.

TBD

Table XVIII presents the average elapsed times for executing the OpenMP default schemes with 1, 2 and 4 threads and the datasets defined in Table **??** in the simulated reproducible stress environment. The average values for 2 and 4 threads confirm the high likelihood of the previous scenarios. Additionally, the average values for the datasets are close to each other.

Speedup values of the average elapsed times for executing the Algorithm II in a stress environment are presented in Table XIX.

TBD

TBD

More attributes gain importance in Dataset 1 because of the spike in cpu related parameters, sys, procs_blocked, wait, and

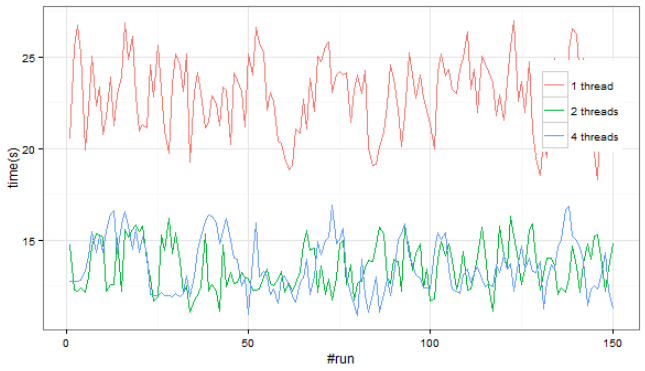

Fig. 19. Performance of executing the Algorithm II in a stress environment with default OpenMP schemes.

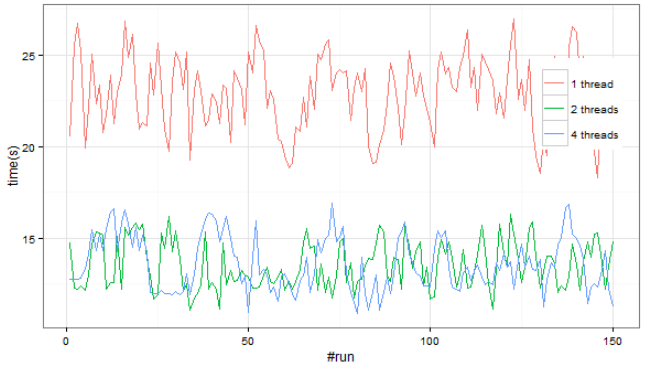

Fig. 20. Performance of executing the Algorithm II in a stress environment with Datasets 1-4.

TABLE XVIII
AVERAGE ELAPSED TIMES FOR ALGORITHM II IN A STRESS ENVIRONMENT.

	Mean elapsed time	SD	RSD
1 Thread	22.99271	2.007224	8.729827
2 Threads	13.52446	1.284616	9.49846
4 Threads	13.58029	1.473687	10.85166
Dataset 1	13.7053	1.243011	9.069563
Dataset 2	12.70467	0.8573054	6.747957
Dataset 3	13.42115	1.495956	11.14626
Dataset 4	12.54097	0.7223649	5.760038

TABLE XIX
SPEEDUPS FOR EXECUTING THE ALGORITHM II IN STRESS ENVIRONMENT

	Dataset1	Dataset2	Dataset3	Dataset4
1 Thread	1.677651	1.809784	1.713170	1.833408
2 Threads	0.986805	1.064527	1.007698	1.078422
4 Threads	0.990874	1.068921	1.011857	1.082874

ram_usage. Dataset 1 leans toward 4 threads in this setting agreeing with the conclusion deduced from the OpenMP default scheme according to Fig. 21, However, it yields the second worst average value and the worst RSD from all the datasets (Table XVIII). This is an indication that the dataset attributes do not conform to the constraints that a good dataset

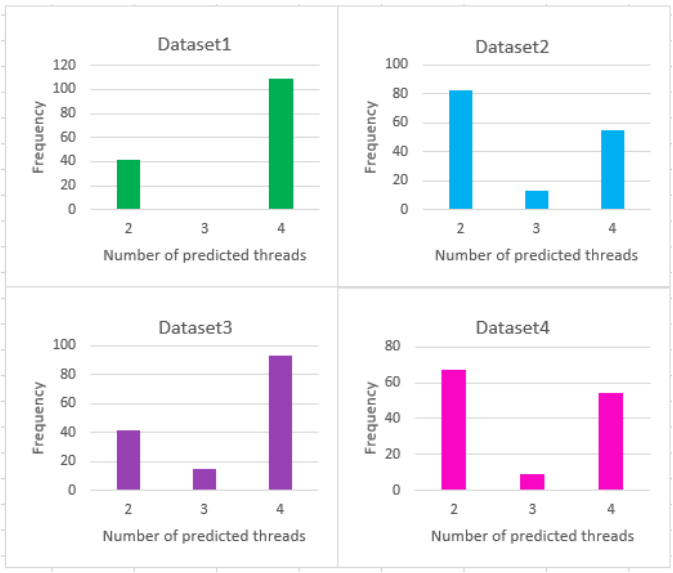

Fig. 21. Histogram of predicted threads for executing the Algorithm I in a stress environment

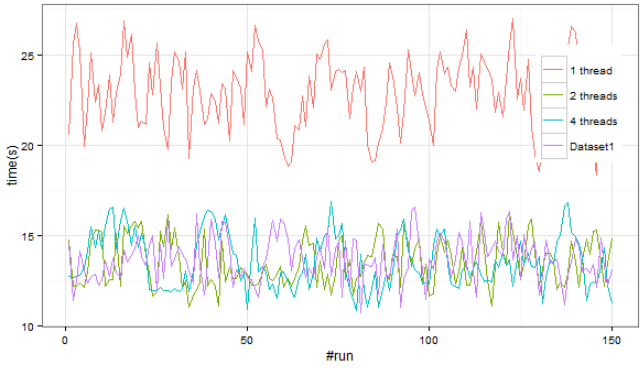

Fig. 22. Performance of executing the Algorithm II in a stress environment - Comparison OpenMP schemes vs Dataset 1.

candidate should abide in order to produce optimal results.

Dataset 2 maps to 2 threads in 70% of the 150 input cases which is opposite of Dataset 1, but results in better performance indicated by the average and depicted in comparison with the OpenMP default scheme (Fig. 23). This is caused by the variability of the optimal number of threads. The response to runtime changes of the forest generated using Dataset 2 attributes, boosts performance compared with the OpenMP default scheme.

The average elapsed time shown in Table XVIII and the numerous regions in Fig. 23 located inside the lower half of the band formed by the OpenMP default performances for 2 and 4 threads combined, indicate this successful mapping that takes advantage of hyperthreading wherever possible. The fact that the predictions weight toward 2 threads despite 4 being established as the optimal number of threads in the majority of the cases in the OpenMP default scheme measurements, indicates that the tool responds to the change

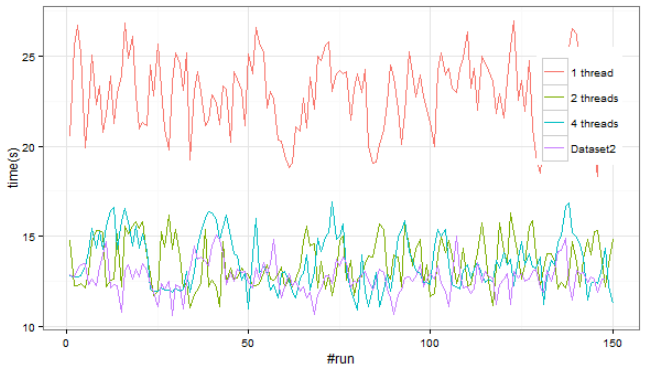

Fig. 23. Performance of executing the Algorithm II in a stress environment - Comparison OpenMP schemes vs Dataset 2.

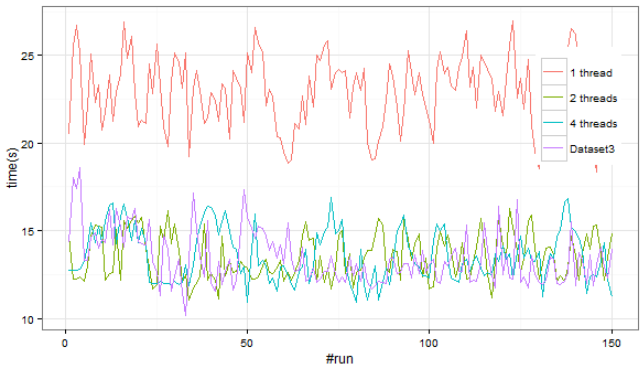

Fig. 24. Performance of executing the Algorithm II in a stress environment - Comparison OpenMP schemes vs Dataset 3.

of the runtime due to the stress imposed on it. The desired speedup is produced in this scenario by not setting a favor to the maximum number of threads. This is a confirmation that the tool achieves the goal set, confirming H2.

Additionally, Dataset 2 has more informative attributes/columns compared with the no stress environment for this same algorithm: cpu_usage and lst_cpu_usage with wide coverage of input values, and idle and irq with less coverage. The irq attribute is a consequence of the stress tool io parameter. The pid_cpu_usage attribute on the other hand gets more coverage. These improvements also affect the quality of the predictions positively.

Moreover, another indication of the quality of the predictions is that even 2 and 4 threads both figure in Dataset 2 predictions with a high frequency, as seen in the histogram Fig. 21. This does not affect negatively the SD and RSD, that is the precision of Dataset 2 in Table XVIII. Different number of threads results in an optimal performance for a certain input. This is closely clustered to the mean value, which is the second best from all four dataset i.e confirming the accuracy of the results.

The average performance and standard deviation achieved by Dataset 3 is closest to the one achieved by Dataset 1 (Table XVIII). Although generally not a good one, in this case, this is an indicator of the similar performance properties exhibited by these two datasets. Also, since not many attributes in Dataset 3 in this environment gain significance such was the case with Dataset 2, the quality of the predictions compared with the one in the no stress environment is not expected to improve a lot.

Although, Dataset 3 favors 4 threads, the performance achieved does not only perform worse than 2 and 4 threads, but a number of spikes is considerably higher than the one for both the OpenMP default scheme for 2 and 4 threads (Fig. 24). This is also reflected in the highest SD and RSD out of all the datasets shown in Table XVIII. It sets additional limits on the use of the average in the comparison.

The standard deviation of the results that is significantly higher in the stress environment complicates making relevant

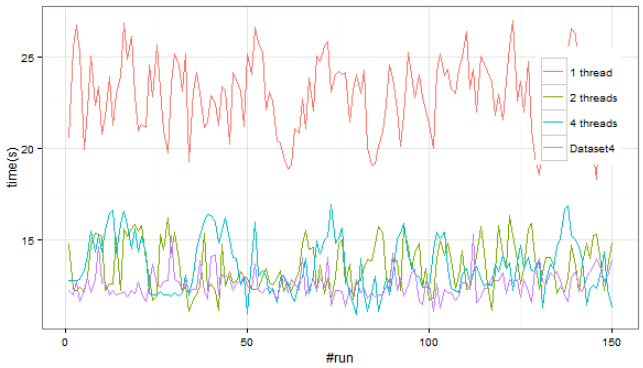

Fig. 25. Performance of executing the Algorithm II in a stress environment - Comparison OpenMP schemes vs Dataset 4.

conclusions. The situation is further augmented with Dataset 3 because the average by itself does not reflect very accurately the actual quality of the predictions compared with the other average performances.

The pid_ram_usage and idle attributes gain significance in the stress environment. Ram_usage, cpu_usage and pid_cpu_usage gain coverage. From Fig. 21, Dataset 4 weights towards 2 threads more than it does towards 4, similar to Dataset 2.

The oscillations in the performance are limited in the range covered by the OpenMP default scheme for 2 and 4 threads (Fig. 25), including minimum average elapsed time, SD and RSD (Table XVIII). These are the closest values to the average of Dataset 2 out of the four datasets. However, the predictions made using Dataset 4 optimize the number of threads successfully based on the current runtime state.

Fig. 26 and Fig. 27 present a closer view of Fig. 25 in order to confirm the previous conclusion and highlight the fact that the variation of the predictions made with Dataset 4 is also smaller despite both 2 and 4 threads being present in the results, increasing the confidence in the validity of the conclusion.

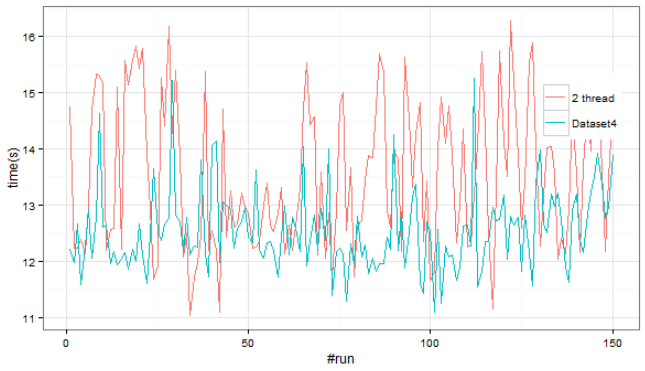

Fig. 26. Performance of executing the Algorithm II in a stress environment - Comparison OpenMP 2 threads vs Dataset 2.

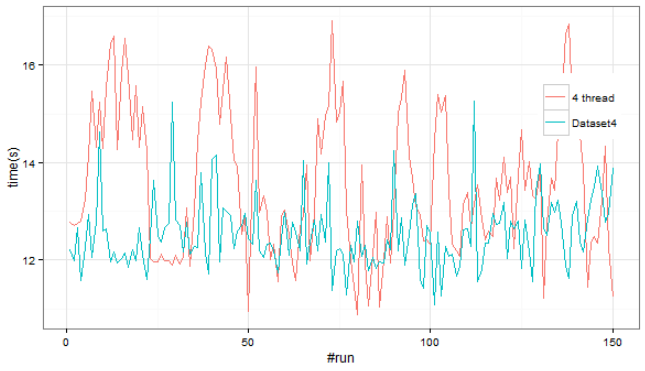

Fig. 27. Performance of executing the Algorithm II in a stress environment - Comparison OpenMP 4 threads vs Dataset 4.

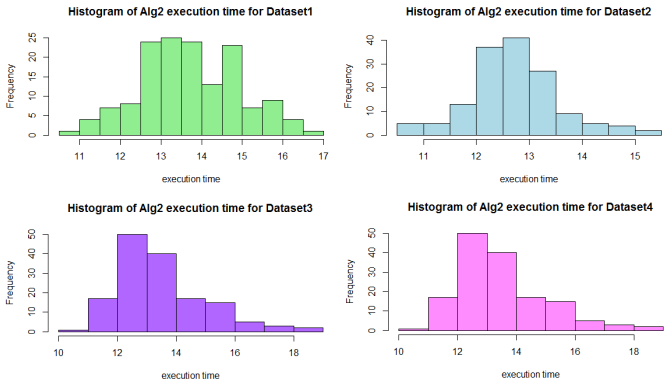

Fig. 28. Histogram of execution times for executing the Algorithm II in a stress environment for Datasets 1-4

D. Comparison

Algorithms I and II exhibit very different resource utilization patterns that result in different predictions. The purpose of running both algorithms is to check whether the tool responds to different running state states such that it boosts performance wherever possible and does not degrade performance.

Analyzing the average performance of executing the Algorithm I in both environments in the OpenMP default scheme (Table XVI and Table VII), one can conclude that as the number of threads increases the performance improves linearly. This confirms that Algorithm I is a highly vertically scalable algorithm.

In contrast to this, increasing the number of threads in the OpenMP default scheme does not necessarily result in a better performance (Table XVIII and Table X).

The environment ratio defined in **??** is useful to compare the effectiveness of the tool in a no stress to a stress environment. The no stress environment is considered as the base case scenario with a minimal amount of additional workloads for which this tool should produce satisfying results and not degrade performance.

Table XX presents all the possible environment ratios R_e defined by (3) comparing the average values of speedups achieved with execution of Algorithm I in a no stress and a stress environment.

It is necessary to consider the fact that in a no stress environment the Algorithm I has more potential speedup than in a stress environment (with an intensive competition of threads for resources such as the one in the conducted experiments). Consequently, from the environment ratios presented in Table XX, one can conclude that Algorithm I achieves better speedup in the no stress environment. Furthermore, the ratios increase as the number of threads increases for each of the datasets executing the Algorithm I. This is an indicator of the impediments posed by the stress to boost the performance.

The tool results with a peak in the base case scenario with minimal workload (no stress environment) for executing the Algorithm I. Since the pattern of resource utilization is mostly affected by the running program, the predictions made are highly effective. As the workload increases, the optimal number of threads is increasingly affected by the other workloads just as its speedup is decreasing, leading to lower speedups compared with the speedups in the no stress environment.

Table XXI presents the Environment ratio defined in 3 comparing the speedups achieved by executing the Algorithm II in a no stress and a stress environment.

TABLE XX
RATIO OF COMPARED SPEEDUPS OF ALGORITHM I IN NO STRESS AND STRESS ENVIRONMENT

	Dataset1	**Dataset2**	**Dataset3**	**Dataset4**
1 Thread	1.242209	1.055975	1.264642	1.308240
2 Threads	1.339776	1.138898	1.363949	1.410973
4 Threads	1.375980	1.169672	1.400806	1.449100

TABLE XXI
RATIO OF COMPARED SPEEDUPS OF ALGORITHM II IN NO STRESS AND STRESS ENVIRONMENT

	Dataset1	**Dataset2**	**Dataset3**	**Dataset4**
1 Thread	0.956524	0.898045	0.975084	0.928093
2 Threads	1.089114	1.022527	1.110246	1.056742
4 Threads	0.987014	0.926667	1.006162	0.957673

TABLE XXII
R_a IN NO STRESS ENVIRONMENT

	Dataset1	Dataset2	Dataset3	Dataset4
1 Thread	1.431177	2.083965	1.353382	1.650119
2 Threads	1.062796	1.547558	1.005026	1.225384
4 Threads	0.546006	0.795047	0.516325	0.629532

TABLE XXIII
R_a IN STRESS ENVIRONMENT

	Dataset1	Dataset2	Dataset3	Dataset4
1 Thread	1.858654	2.450450	1.755277	2.326009
2 Threads	1.307401	1.723681	1.234686	1.636146
4 Threads	0.761178	1.003536	0.718841	0.952575

The situation with the ratios in Table XXI is almost completely reversed compared with Table XX. Algorithm II achieves approximately the same speedups in both environments (considering the standard deviation of the results in Table X and Table XVIII) or a slightly better speedup in the stress environment.

Since Algorithm II is considerably less scalable than Algorithm I, the ratios indicate that for algorithms such as Algorithm II, with a highly variable number of optimal threads, the tool predictions are at least equally effective in the stress environment as in the no stress environment. Therefore, the dynamic optimization of the number of threads performed by the tool makes a greater difference under increased workload for algorithms that do not have an obvious or unique number of optimal threads. This is another evidence indicating that the mapping of the runtime state to a number of threads results in an optimal performance for certain datasets.

The optimal results from running the program in OpenMP default schemes depend on whether the default number of threads agrees with the optimal scheme for the algorithm that is running. With the experiments presented in this section, it has been shown that the tool responds to different running states in an optimal way and does not degrade performance for two of the datasets Dataset 2 and Dataset 4.

This indicates that finding a pattern in resource utilization is a feasible challenge. Additionally, there is a way to calculate the optimal number of threads, which is encoded in the runtime parameters and the relations between them. The Random forest algorithm limits these rules to a certain type, where all the attributes have to be linearly independent in order to produce the best results.

Additional limitation imposed by this is the that domains of input OpenMP algorithms include only algorithms that benefit from using at least 2 threads in comparison with the sequential execution. This happens because it is not possible to determine whether the overhead introduced can be amortized for the duration of the program to be optimized based on the runtime only without running the algorithm beforehand. Additionally the benchmarks that are used to produce the forest, are scaled successfully to at least 2 cores.

VI. DISCUSSION

This Section discusses the constraints and improvements that will gain more benefit of using the algorithms.

The ratio of the speedup of Algorithms I and II in both the stress and no stress environment decreases as the number of threads increases for each of the datasets (Table. XXII and Table. **??**).

The ratios for two threads in both environments are all greater than one, which indicates that for the number of physical cores, Algorithm I achieves greater speedup than Algorithm II for all Datasets 1-4. In the case of 4 threads, Algorithm II achieves greater speedup than Algorithm I for all Datasets 1-4 except in the case of $R_a(4T, DS2)$ in a stress environment, where it equals.

One can conclude that Algorithm II has a variable number of optimal threads from the variations in the performance of Algorithm II for the OpenMP static schemes in both environments, as presented in Table X and Table XVIII. However, Algorithm I performs best with 4 threads. Therefore, it follows that dynamic configuration of the number of threads would benefit this algorithm better than Algorithm I in the case where the tool results for Algorithm I are compared with the best OpenMP scheme. Additionally, this ratio confirms the effectiveness of the predictions made by the tool in optimizing the performance with greater speedup for 4 threads demonstrating the benefit of the dynamic optimization.

The difference between the ratios in a stress and no stress environment is that the ratios in the stress environment have greater values than the ratios in a no stress environment. This means that for two threads there is a wider gap between the speedups in a stress environment, and for 4 threads there is a smaller gap between the speedups. For two threads, this observation agrees with the conclusions from the performance analysis that show Algorithm I demonstrates a higher potential for speedup.

For four threads, this observation agrees with the conclusion that Algorithm II provides more opportunities for correction that Algorithm I especially in a stress environment, while Algorithm I is compared with its optimal performance yielding low speedup.

The maximum ratio for the number of physical cores and maximum threads with value of 2 is $R_a(2T, DS2)$ in both environments. Algorithm I achieves the greatest speedup (for the number of physical cores) in comparison to Algorithm II for Dataset 2. The performance analysis show that Dataset 2 produces good results for Algorithm I in both environments although it produces relatively poor results for Algorithm II in a no stress environment and second best results in a stress environment. This explains the wide gap presented in the ratios.

However, Algorithm II achieves the greatest speedup in comparison to Algorithm I for the value of $R_a(4T, DS3)$. This is explained with the evidence from the performance analysis where Dataset 3 produces good results for Algorithm II in a no stress environment while poor for Algorithm I and a relatively

worse results for Algorithm I than Algorithm II in a stress environment.

Table VIII and Table XVII present all speedups calculated over the average elapsed times of executing the Algorithm I and Table XI and Table XIX for Algorithm II. TBD

TBD

TBD

TBD

A. Profiling

Improvements in the profiling module will increase the accuracy of the dataset in capturing the state of the configuration that yields the best performance in each run of a specific benchmark. They will also improve the accuracy of capturing the state of the runtime at the time of deciding on the optimal number of threads. Here are a few propositions how to achieve this.

- Use more than two snapshots to calculate the cpu usage related attributes during profiling.
- Add additional parameters to the record for virtualization environments.
- Dynamically determine which attributes to monitor and include in the dataset, by checking if their variance and importance is significant enough using the average, the range, the standard deviation of the column, the node quality and the vote count extracted by overriding OpenCV predict method.

B. Benchmarks

The following is a list of propositions on how to improve the quality of the benchmarks and increase the range of states covered by their concurrent execution.

- Use standardized benchmarks in order to be able to compare the results with other similar works
- Include more benchmarks
- Run the same benchmark with varying parameters and determine its execution time
- Determine the number of concurrent pthreads automatically based on the number of processors on the machine
- The subset of benchmarks used should optimally encompass a significant part of the possible variations of the runtime state for the given parameters.

C. Machine learning algorithm

Random forest is a convenient algorithm to use because it does not require normalization, but the its deficiency is that its domain (for which accurate output/prediction is produced) heavily depends on the dataset. Even with the boost technique, if the values for some input we want to compute a prediction for are not within the range of the corresponding attribute constraints in the decision tree, then the negative impact on the predictions is significant.

Other parameters that can be tweaked and parameterized are the tree depth, the n-ary dimension of the tree and the max number of variables.

Among other possible approaches are the following: the purely Boost approach, k-means, bayesian, SVM and linear regression.

With linear regression the domain problem previously mentioned is less of a problem, even though the inaccuracy increases and the outlier gains power as we move away from the mean of the predictor and away from the regression line.

SVM decomposition also has the potential to produce good quality prediction since singling out the n most influential attributes with the right value for n removes much of the noise in the data. However it is highly sensitive to variation and correlation between the attributes. It also has to be normalized.

D. Prediction of the optimal number of threads

Further improvement on the prediction can be done by creating multiple forests for different subset of approximately independent variables/columns. Then we pick an odd number of forests to be able to get a majority vote and increase the likelihood of an accurate prediction of the number of threads for a specific parallel region.

In case of a predicted number of threads greater than the default number of threads (if equal to the number of physical cores), adding an extra branch condition using a higher confidence threshold.

In some cases, the overhead of using more threads than the maximum number of threads given with the internal control variable max_num_threads, leads to a performance boost, and is a worthwhile option for further exploration. Among the difficulties posed is the inevitable prolonged initial screening/profiling. This requires coming up with an optimized profiling and forest generation scheme.

Concerning the SMT optimization aspect here are two options to explore:

- addition of parameters that characterize the flow of the program are needed, some general instruction level characteristics along with program specific parameters that will enable the construction of a data dependency and dependency graph.
- corrections of the initial random forest generated done during runtime based on the performance of the running program

E. Interactivity

Since the idea behind this tool is essentially to assist developers (later referred to as users) with highly varied level of expertise in obtaining optimal performance, here are a few features that will greatly improve the usability of the tool.

Expand the tool with an option to:

- create and load forests through command line parameters
- load forests automatically based on the the current runtime state
- change parameters of the training and building process

VII. Related work

TBD

The parameters used for the training are a mix of compiler and runtime knowledge parameters. They address the programs running on instruction level (load/store, instruction count and branch count), an average number of running tasks, number of tasks, queue length and number of processors.

In comparison the parameters used by the tool in this paper, address solely an extensive list of runtime parameters Table I, seeking a generic approach rather than a program-centric one such as the one taken in [3].

TBD

TBD

TBD It proposes a two level hierarchical scheduler that handles overall performance using the first one and load imbalance using the second level determining the number of SMT threads to be used in the scope of a processor. With SIGAR, detailed information can be extracted about the each logical processor, creating the opportunity to address loop scheduling strategies in further expansion of the functionalities provided so far.

VIII. Conclusion

TBD

TBD

Even though many improvements are possible especially in the usability and user friendliness subdomain, this work is a proof of concept that can be further built upon to evolve into a tool with effortless installation and configuration. One simple solution to the benchmark configuration is using the pThreads in order to generate the dataset is to use the number of physical cores, as the number of benchmarks threads. In order to establish the correctness and accuracy of the results, especially for machines with a higher number of processors and virtual machines, one has to conduct more experiments using randomization techniques. Randomization is needed to determine the confidence of the results due to the high standard deviation of the predictions that are made in the stress environment.

Automatic configuration of the attributes to be included in the dataset requires extensive analysis of the qualities that a dataset should possess in order to optimize an unknown OpenMP program based on the runtime state at the time of the call. This qualities and properties are sought out throughout the analysis in both the no stress and stress environment.

TBD

The generic nature of the tool and its dependence on runtime parameters and not on specific program knowledge put limitations on the domain of algorithms that can be optimized with this tool. The more complex and long running the program is, assuming it can be scaled to at least 2 threads with a performance boost, the more it become cost-effective to use this tool.

TBD

Future work is based on developing a truly generic self-configuring tool using statistics to calculate the importance of the parameters, the proximity between them and the confidence interval for the predictions using the vote count.

References

[1] OpenCV dev team. (2014, Nov) OpenCV documentation, Random Trees. [Online]. Available: http://docs.opencv.org/3.0-beta/modules/ml/doc/random_trees.html

[2] John Burkardt. (2011, May) C++ Examples of Parallel Programming with OpenMP. [Online]. Available: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[3] M. K. Emani, Z. Wang, and M. F. O'Boyle, "Smart, adaptive mapping of parallelism in the presence of external workload," in *Code Generation and Optimization (CGO), 2013 IEEE/ACM International Symposium on*. IEEE, 2013, pp. 1–10.