An OpenMP runtime profiler/configuration tool for dynamic optimization of the number of threads

Tamara Dancheva - University of Cyril and Methodius, Skopje, Macedonia
Faculty of computer science and engineering
tamaradanceva19933@gmail.com

Abstract

The OpenMP API was designed with a goal to provide a simple, fast, compact and portable way of parallelizing programs [1]. The first release dates from 1998, and computer architectures have diversified and evolved a great deal in the time being requiring OpenMP API to evolve too, the latest release being in 2013. Today, considering the prevalence of SMPs and SMT on the market, the massive virtualization and ccNUMA architectures, the default schemes that has sufficed in the past are no longer producing the desired results.

This paper describes the implementation and the experimental results of an approach for dynamic configuration of the number of threads used in the OpenMP environment based on the current state of the runtime at the time of the call. For this purpose a mix of profiling and machine learning techniques are used to make the decision about the number of threads. The decision is made at the time of the call to set the number of threads. The following approach is designed to be cost effective in the scenario of a highly dynamic runtime primarily when running relatively long tasks consisting of a number of parallel constructs when it gets difficult to configure the optimal number of threads.

Introduction

Multi-cache architectures, hyperthreading and multi-core techniques helped prevent the potential stagnation in performance due to the limits imposed by physics with a cost. The intensive exploitation of virtualization techniques especially lately (of memory, computing power, application, operating systems) only enhance the complexity introduced.

Therefore, in order to get good performance, programmers as well as scientists have had to start studying the hardware characteristics, and learn
because the tree is too specific to a subset of all the possible inputs or runtime states). Further details are disclosed in the following sections.

about cache associativity, swapping, virtualization and many other low-level hardware and software characteristics. Problems with OpenMP concurrency and default schemes have been observed for years..The main problem is the diversity of resource sharing mechanisms in different architectures, most notably the SMT [3]. OpenMP has a default scheme of #cpus * SMT threads/cpu_as the the number of threads. This does exploit the hyper threading automatically, and depending on the interconnection, location and resource sharing of the processors involved it is not always a good choice because of the complexity it involves.

Having a tool that can automatically learn the formula for the optimal number of threads without previously gathering explicit user feedback can significantly automate the process of producing better performing algorithms.

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. 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. The dynamicity 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.

Profiling is used to capture the state of the system while running a suite of benchmarks concurrently using pThreads, and is used as an input for the creation of random trees (also known as the forest tree algorithm). This is a machine learning algorithm that uses a supervised approach to train a specific number of decision trees that all vote in order to decide on the classification of an runtime state designed in such a way as to avoid overfitting (meaning generate bad predictions

Benchmarks and PThreads

PThreads is a an implementation of thread-level parallelism[6] in C/C++. It is used to concurrently run the suite of OpenMP algorithms. The benchmarks are run for a range of threads [1, num_proc] which constitutes one run. This is done 10 times by default. Then the best performance time is found and the previously taken state (using SIGAR) corresponding to it is written to disk.

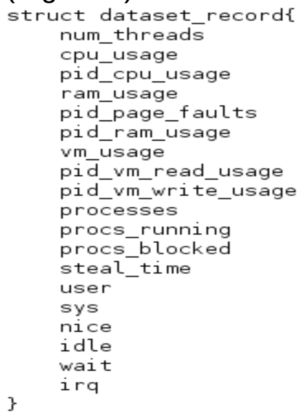
Implementation

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 to 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.
The compiler used is GNU.

Profiling

Retrieval of the runtime parameters, is done using the SIGAR API [4]. The SIGAR API is an open source cross-platform library that offers helps in parsing and locating major system parameters including cpu usage, ram usage, virtual memory io traffic (total and by process) as well as more detailed information about the processes, the hardware features and the operating system running on the machine.An indicator for its quality is that it is a framework suitable for use in cloud environments for measuring hypervisor performance [5] It offers an extensive set of utilities to monitor the state of a highly dynamic runtime which makes it a suitable tool for the kind of measurements needed for this tool. A sampling of all these parameter makes up one dataset record (FIgure 1).



```
struct dataset_record{
        num_threads
        cpu_usage
        pid_cpu_usage
        ram_usage
        pid_page_faults
        pid_ram_usage
        vm_usage
        pid_vm_read_usage
        pid_vm_write_usage
        processes
        procs_running
        procs_blocked
        steal_time
        user
        sys
        nice
        idle
        wait
        irq
}
```

Figure1.List of all possible parameters

Random forest algorithm

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[7]. The input dataset is a file on disk in the form of 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 a .forest extension The generated forest is kept to further use to predict the number of threads for an input unknown runtime state.

OpenMP algorithm

The OpenMP algorithm which utilizes the tool has to call the omp_set_num_threads function before each parallel block in order for the wrapper function to be invoked. The number of threads is predicted using the random forest and used as a parameter in the new handle which is returned.

Theoretical analysis

Benchmarks

The goal of running the benchmarks concurrently is to simulate as many states of the runtime as possible so that a representative decision trees can be constructed out of the provided dataset. This is closely related to the parameters that are used to represent the state and is essential to be carefully configured so that a performance boost can be obtained.

Dataset record constraints

The dataset record columns included as parameters should not have negligible variance or be constant across all rows. The high variance is an indication of no information or in terms of information retrieval, high entropy, which ultimately indicates 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 only a subset is chosen from Figure1.This machine learning approach most importantly conceptually enables us to generalize and speculate about the model, while getting good predictions, which is the main goal.

Boosting and random forest

The dataset is boosted by generating additional dataset records using the existing dataset records with repetition. Basically the experiment is repeated a number of times, and this helps to break the independency, or decrease its negative impact on the decision trees that are to be generated out of the dataset by adding a variance to the output that helps remove some of the initial bias. This works for small biases only. Additionally boosting enables us to get p-values and provides a good basis for running statistical tests and deriving conclusions about our data. Therefore, the parameters have to be very carefully selected. Additionally , random forest does not require any normalization of the dataset, since it considers each column independently[8].

Performance analysis

The following measurements are obtained by running 8 instances of openmp program concurrently, half of which do not use the tool.

Each forest is created the first time the call to set_num_threads is made. It is created by executing 50 runs of each of the the benchmarks. Each run consists of one execution for each number of threads in the range [1 , num_processors]).

After the initial configuration, the forest is used to predict the number of threads for the parallel region in the openmp algorithm , that is repeated 25 time.

The result is 100 values obtained (time elapsed) by using the random forest for prediction, and 100 values obtained utilizing OpenMP default schemes.

List of programs ran as benchmarks ran: Dijkstra, Multitask, Poisson and Circuit [12].

Hardware: HP Probook 650 i5-4210M “15.6” 4GB

OS: Debian, GNU/Linux 6.0

Average speedup across 10 of this experiments :

8.536212/2.64210383=3.23083896

Figures 2-6 show the speedup obtained by one exp.

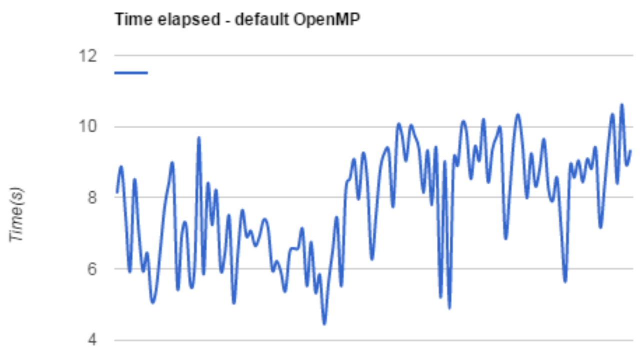


Figure2. Time elapsed(s) - default OpenMP scheme

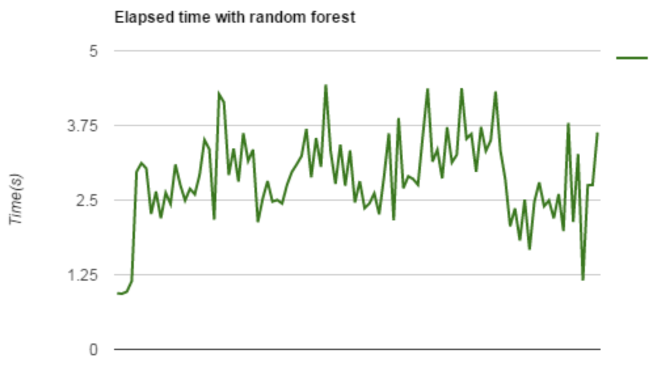


Figure3. Time elapsed(s) - random forest scheme

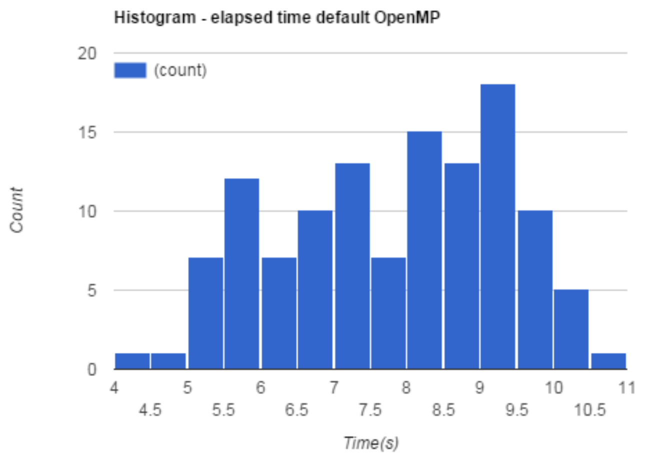


Figure4. Histogram - default OpenMP scheme

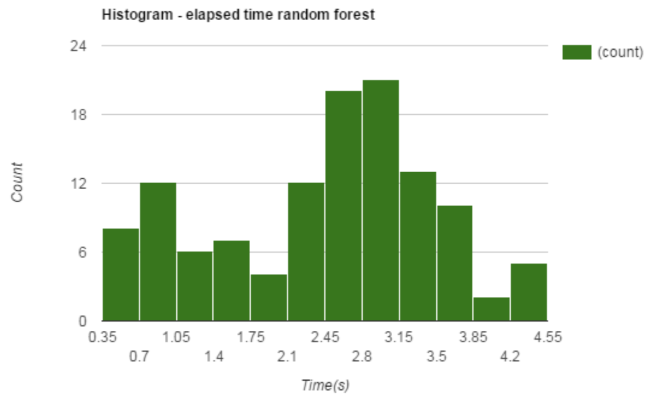


Figure5. Histogram - random forest

The results in low workload runtime coincide with the default OpenMP scheme.

Improvements

Profiling

- Use more than two snapshots to calculate the cpu usage related attributes during profiling.
- Add parameters to the record correlated with virtualization.
- Dynamically determine which attributes to monitor and include in the dataset, by

The k-means approach is problematic because the shape of the clusters might not be spherical or oval, in which case, the distance to the centroids might cause the record to be put into a neighbouring cluster. Additionally, small or overlapping boundaries between clusters will negatively impact the output. Alternatively, an approach using multiple centroids to describe the state might be used to alleviate the first problem[10].

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.

Interactivity

- Expand the tool with an option to create and load forests through command line parameter
- Expand the tool with an option to change parameters of the training and building process

Benchmarks

- Use standardized benchmarks in order to be able to compare the results with other similar works
- Include more benchmarks
- Determine the number of concurrent threads 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 checking if their variance is significant enough using the average, the range and the standard deviation of the column.

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 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 the negative impact on the predictions is significant.

Other parameters that can be tweaked and parameterized are the tree depth, the n-arity 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 attributes. It also has to be normalized.

Related work

Among the research work done in area of adaptive or dynamic runtime OpenMP configuration, one of the most recent works, is based on creating a Multilayer Perception neural network with one hidden layer with back propagation to determine the number of threads for a parallel region based on the external workload[9]. The parameters used for the training are addressing 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 more general parameters concerning not only processor and task related parameters. The main difference in the choice of parameters is that memory is included and the focus is on resource utilization capturing the state of the runtime from a different perspective. The parameters chosen for this tool include percents of utilization of current cpu, list of all cpus and memory as well as percent of processes in a certain state or certain type (blocked, running, idle, user/system).

Even though different machine algorithm, benchmarks and profiling tools are used, a parallel can be drawn between the effectiveness of these two approaches. The benefits are mainly demonstrated with increase in the workload in both cases. The speedup achieved in the cited paper is above 2.4x (up to 3.2x) over the OpenMP default scheme. The speedup achieved with the tool presented in this paper is 3.2.

Other similar work is done building up on the previous paper using other approaches other than neural networks in dynamic workload settings, such as reinforced learning and Markov decision processes for unsupervised learning [11].Random forest are mentioned as a suggestion for another machine learning algorithm that can solve the problem.

Substantial amount of work is done in creating loop scheduling policies using runtime information. One of the paper demonstrating promising results proposes an adaptive loop scheduling strategy targeting SMT machine[2]. 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.

Conclusion

There is a lot of room for improvement of this tool along with more measurement to establish the correctness and accuracy of the results. Randomization is needed and a more extensive set of measurements. The results so far are encouraging, and the tool is able to correct the default scheme and find the optimal number of threads in cases of a highly dynamic runtime when the variation of the parameters is significant enough to provide indication of it. My plan is to improve upon the automation of producing a truly generic tool that will be able to detect the right configuration using statistics to calculate the importance of the parameters, the proximity between them and the confidence interval for the predictions using the vote count as well as implement random forest to remove the dependency. Being able to load and save the forest, a user will not have to regenerate the forest each time a program is ran.

References

[1] OpenMP - goals of OpenMP , Blaise Barney, Lawrence Livermore National Laboratory https://computing.llnl.gov/tutorials/openMP/

[2] Y. Zhang, M.Burcea, V.Chang, R.Ho and M.Voss. An Adaptive OpenMP Loop Scheduler for Hyperthreaded SMPs. In Proceedings of PDCS-2004:International Conference on Parallel and Distributed Computing Systems, September 2004

[3] M. Curtis-Maury, X. Ding, C.D.Antonopoulos, D.S.Nikolopoulos. An Evaluation of OpenMP on Current and Emerging Multithreaded/Multicore Processors. In the Proceedings of the 2005 and 2006 international conference on OpenMP shared memory parallel programming, Pages 133-144

[4] Hyperic, SIGAR API, Ryan Morgan, https://support.hyperic.com/display/SIGAR/Home

[5] V.V Reddy, L.Rajamani, Evaluation of Different Hypervisors Performance in the Private Cloud with SIGAR Framework , In International Journal of Advanced Computer Science and Applications, Vol. 5, No. 2, 2014

[6] Pthreads, Blaise Barney, Lawrence Livermore National Laboratory https://computing.llnl.gov/tutorials/pthreads/

[7] OpenCV Documentation, Random Trees http://docs.opencv.org/3.0-beta/modules/ml/doc/random_trees.html

[8] L.Breiman and A.Cutler, Random Forests http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[9] M. Krishna, E. Zheng and W. M.F.P. O’Boyle. Smart, Adaptive Mapping of Parallelism in the Presence of External Workload. In "Celebrating Diversity: A Mixture of Experts Approach for Runtime Mapping in Dynamic Environments." (2015).

[10] S. Guha, R. Rastogi and K. Shim, “CURE an efficient clustering algorithm for large database”, Information Systems, vol. 26, no. 1, (2001), pp. 35-58.

[11] Emani, Murali Krishna. "Adaptive parallelism mapping in dynamic environments using machine learning." (2015).

[12] C++ Examples of Parallel Programming with OpenMP May 2011. https://people.sc.fsu.edu/~jburkardt/cpp_src/openmp/openmp.html