Development of Performance Assessment Method Based on Aspen DSL and Micro-Kernels benchmarking

Ekaterina Tyutlyaeva, Alexander Moskovsky, Igor Odintsov, Sergey Konyukhov

RSC Technologies

Introduction

Accurate assessment and predicting the applications performance are essential for the effective usage of modern multi-core computers. Performance models can allow to describe the dynamical behavior of applications on different computing platforms so they can be useful for the design of future supercomputers.

In this research, we propose the application performance assessment method based on Aspen DSL modeling complimented with direct performance measurements and experimental data analyzing.

On the poster an analytical performance model for multispectral images processing application is presented as an example of more general approach. The model construction steps include application profiling, experimental direct measurements of actual FLOPs, processor cycles and different memory usage data.

Application

The studied application is the most computationally intensive part of an automatic system for detecting fishing boat lights from nighttime images of the VIIRS multispectral radiometer [1].

The system detects isolated bright spikes that are sharply visible on the sea's night surface. In the moonlight, the interference from clouds and lunar glint are taken into account as well. In the current work, a module based on Direct Fourier Transformation in the moving window is modeled.

Hardware

In the Table 1 codenames and specifications of the studied testbeds are listed.

Table 1: Testbeds: Technical Specifications

Codename	Processors (CPU + GPU)	Frequency		Memory
		Basic	Turbo	
Broadwell	2 x Intel® Xeon® E5-2697A v4	2.60 GHz	3.60 GHz	128 GB DDR4/2400MHz
Skylake	2 x Intel® Xeon® Gold 6150	2.70 GHz	3.70 GHz	192 GB DDR4/2666MHz

Step 1: Profiling

Starting point of any application performance analysis is a profiling. The profiling allows us to understand an application dynamical structure and identify sections of program code consuming the most parts of execution time.
The Intel® VTune™ Amplifier XE [2] results have been used to visualize call graph of the application under the study.

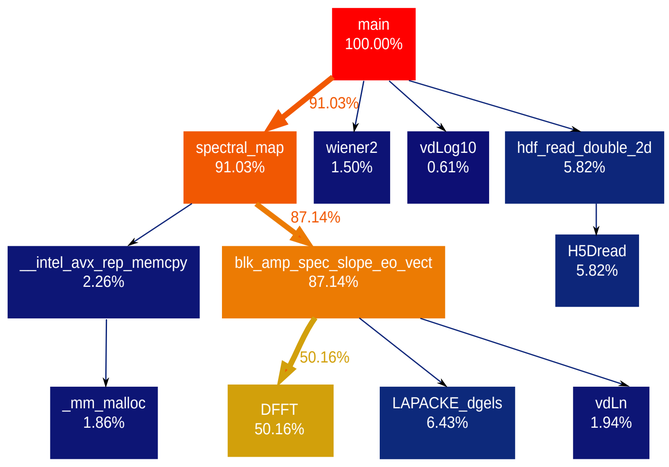

Figure 1: Visualised Call graph of the Studied Application for Skylake testbed

According to the call graph (Fig. 1), the most computationally intensive part of the application is the moving window processing stage (*spectral_map* function, Fig. 2). The detailed statistics on processing stages for all architectures is shown in Table 2.

```
for (r = 0..num_rows ; r+=blk_size/8){
    for (c = 0..num_cols; c+=blk_size/8){
        Prepare_input_block; //blk_size x blk_size
        blk_amp_spec_slope_eo_vect(input_block, …);
        Save_output;
    }
}
```

Figure 2: Processing in moving window

Table 2: Statistics of Processing Stages Times

Processing Stage	Broadwell		Skylake	
Input	1.44 sec	3.3%	1.04 sec	2.6%
Preprocessing	0.616 sec	1.4%	0.555 sec	1.4%
Moving Window	41.2 sec	94.1%	38.3 sec	95.0%
Postprocessing	0.0486 sec	0.1%	0.0341 sec	0.1%
Output	0.474 sec	1.1%	0.399 sec	0.9%
Total	43.3 sec	100%	40.3 sec	100%

Summary

The poster presents the practical approach to model application performance by the combining theoretical asymptotical bounds with the information obtained from experimental data.
This method is based on Aspen DSL representation of analytical expression for application runtime and its parameterization by the results of measurements of number of FLOPs-instructions and memory requests.
Our approach consists of the following steps:

0. Describing the abstract machine model in terms of crucial technical parameters such as CPU and DDR DRAM frequencies and so on.
1. Application profiling and call graph visualization to identify the main computational intensive kernels.
2. For each kernel from step 1 measurements of CPU cycles with high accuracy to separate its basic blocks.
3. For each BB from step 2 development of micro-benchmarks; analyzing the CPU and memory usage for various inputs; composing the analytical expression for FLOPs and memory requests as functions of input parameters.
4. Constructing the Aspen DSL model for application runtime based on data collected through steps 1-3.
5. Prediction of the application runtime using machine model (step 0) and Aspen model (step 4).
6. Application runtime adjustments using micro-benchmark that simulates the basic memory access pattern for our application.

Described approach allows creating machine-independent model that can be adjusted to assess performance for a given computing platform.

Step 2: Aspen

Aspen (Abstract Scalable Performance Engineering Notation) [3] is a domain specific language for analytical performance modeling. It includes a formal specification of an application's performance behavior and an abstract machine model; but it also includes collection of costs that can be extracted from direct performance measurements to refine the model.

In the framework of Aspen DSL paradigm, the execution time is defined as follows (see Formula 1).

$$T_{run} = T(\bar{h}, \bar{s}, \bar{n}) \quad (1)$$

$\bar{s}$ — software (algorithmic) parameters
$\bar{n}$ — input (runtime) parameters

An ideal performance approximation for computation kernel runtime depends only on the number of executed flops, number of processed bytes and the control flow composition (serial or parallel) of subkernels runtimes:

$$kernel = \{basic\_blocks\} \cup \{subkernels\} \quad (2)$$

$$T_{run}(kernel) = F(T_{run}(bb), T_{run}(sk))$$

$$F = (T_1, T_2, \ldots, T_n) = \begin{cases} \text{Serial Composing} & \Longrightarrow T_1 + T_2 + \ldots + T_n \\ \text{Parallel Composing} & \Longrightarrow max(T_1, T_2, \ldots, T_n) \end{cases}$$

$$T_{run}(bb) = T_{flops} + T_{mem}$$

where: $T_{flops} = \frac{Number\_of\_Flops}{IssueRate}$ $T_{mem} = \frac{Number\_of\_Bytes}{Bandwidth}$

So, for each basic block we need to estimate:

- $N_{flops}(\bar{s}, \bar{n})$ → number of flops
- $N_{bytes}(\bar{s}, \bar{n})$ → number of bytes

} depends on algorithm, input parameters

Step 3: Basic Blocks

The main computational kernel of the application is the double nested for-cycle, each iteration of which includes the preparation of input block of pixels (“moving window”), the block processing, and the writing results (see Fig. 3).

Each iteration of the main computational kernel was considered as subkernel splatted into 9 basic blocks (BB) consuming together more than 95% of runtime. BB were chosen on the data collected by time stamp counter (TSC) register - data highest granularity or precision timer.

The blk_amp_spec_slope_eo_vect kernel splitting results are presented on Fig. 3.

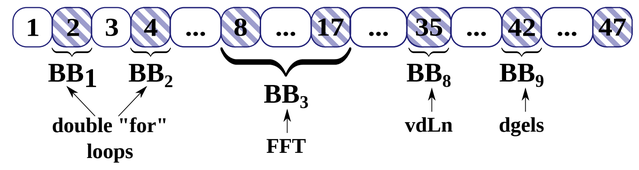

Figure 3: The blk_amp_spec_slope_eo_vect kernel

Dividing blk_amp_spec_slope_eo_vect into basic blocks

Codefragments with runtime <1% were skipped

Step 4: FLOPs Estimations

To find out constants in theoretical asymptotic expressions, we have measured number of retired FLOPs-instructions for basic blocks of moving window processing: FFT and DGELS.

We have collected performance data for all studied architectures by performance monitoring suite LIKWID [4]. The overall number of FLOPs-instruction refers to a total sum of 64-, 128-, 256- and 512-bit FLOPs-instructions multiplied by appropriate factors:

$N_{FLOPs} = N_{64bit\ flops} + 2 \cdot (N_{128\ bit\ flops}) + 4 \cdot (N_{256\ bit\ flops}) + 8 \cdot (N_{512\ bit\ flops})$

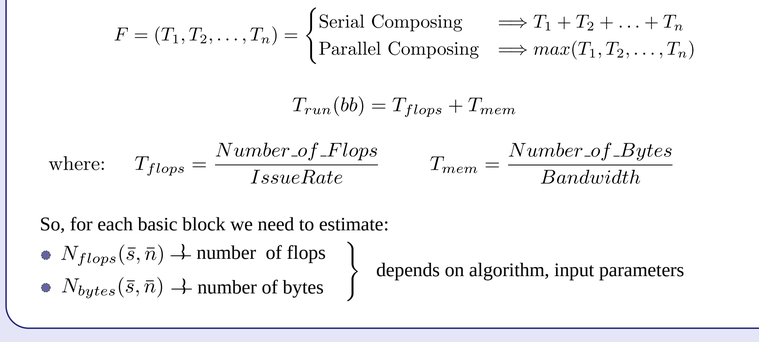

Figure 4: FFT

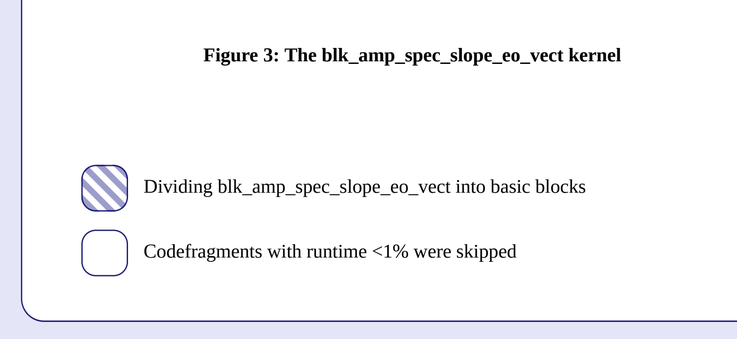

Figure 5: Dgels

Empirical relations between the linear size N of moving window and the number of retired FLOPs-instructions are presented on Fig. 4-5 and Table 4.

Table 3: Models

Codename	FFT	DGels
Broadwell	$N_{flops}(x) = c_1 x \cdot \log(x) + c_2$ where $c_1 = 3.363$, $c_2 = 2512.664$ for 1st model where $c_1 = 2.605$, $c_2 = -2014.426$ for 2nd model	$N_{flops}(x) = c_1 x + c_2$ where $c_1 = 13.96$, $c_2 = 126.43$ for 1st model where $c_1 = 10.17$, $c_2 = 144.79$ for 2nd model
Skylake	$N_{flops}(x) = c_1 x \cdot \log(x) + c_2$ where $c_1 = 4.064$ $c_2 = 2577.243$ for 1st model where $c_1 = 3.086$ $c_2 = -3312.459$ for 2nd model	$N_{flops}(x) = c_1 x + c_2$ where $c_1 = 19.17$, $c_2 = 367.82$ for 1st model where $c_1 = 11.22$, $c_2 = 309.93$ for 2nd model

Step 5: Memory Usage

Memory contention resulting can become a significant component (i.e., over 50 percent) of an application’s execution time. According to the paper [5] a model that ignores memory contention predicts an average execution time about four times smaller than the experimental value.
Although some analytical models capable of estimating with an acceptable accuracy the execution time of jobs running on multi-core machines, it requires set of highly detailed measurements of memory access and service time. It is hard to assess these parameters for single application with limited input data. Actually, according to the LIKWID data almost all memory operations for our application have been processed in cache hierarchy.

Thus, in this work we have developed the micro-benchmark that simulates the basic memory access pattern without performing actual computation for the most intensive part of the application - moving window processing stage.

The values of memory usage time are computed as follows. First, we have collected the load/evict data for all cache levels using LIKWID performance counters. Then, using the L1, L2 and L3 cache latency listed in processors detailed specification [6] we can estimate memory usage time as follows:

$$T_{mem} = \frac{L1_{Latency} \cdot (pmc_0 + \ldots + pmc_3) + L2_{Latency} \cdot (pmc_0 + pmc_1) + L3_{Latency} \cdot (pmc_2 + pmc_3)}{cpu_{frequency}} \quad (3)$$

Where:

pmc_0 - L2 to L1 load data
pmc_1 - L1 to L2 evict data
pmc_2 - L3 to L2 load data
pmc_3 - L2 to L3 evict data

$L1_{latency}$ – 5 cycles
$L2_{latency}$ – 14 cycles
$L3_{latency}$ – 50-70 cycles

According to the estimated results of the benchmark described above, we have found the empirical relations between the linear size N of moving window and the memory usage impact on processing time (see Fig. 6).

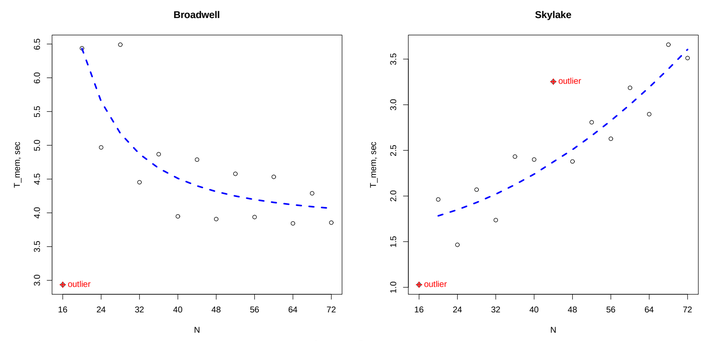

Figure 6: Memory Usage

Future Work

Work in progress involves more accurate assessment of the influence of caches and main memory bandwidth on the performance. Future work will include verifications and refinements for architectures, for example Intel Xeon Phi family. An important component is to fully extend the proposed method to multi-threaded applications. There are some challenges that has to be tackled to increase the accuracy of the multi-core performance predictions models [7].

Step 6: Modeling

Table 5 shows the measured and modeling results for Skylake and Broadwell-based platforms. It should be noted, that modeling results are calculated for a fixed processor and memory frequency, while modern architectures support dynamic frequency scaling for both CPU and DRAM. So frequency of a processor can be automatically adjusted depending on the actual needs, which affects application performance. For that reason, table 5 includes modeling results for two available processor frequencies for both architectures. Our experiments used single-threaded version of applications. We will investigate the impact of multi-threaded version of this application in future work.

In addition, the constructed analytical models can be recalculated and used for performance estimation for other hardware with similar architecture (for example, with other CPUs from the same CPU line). Table 5 (last row) shows the modeling results for Skylake using analytical model constructed for Broadwell-based platforms. This performance assessments did not require a Skylake testbed benchmarking, it only uses specification data.

Table 4: Modeling Results for Broadwell Testbed

NxN	24x24	32x32	40x40	48x48	56x56	64x64
T_{exp}	40.268	20.920	28.078	26.095	25.039	21.603
T_{basic_model} Basic Frequency	22.444	15.988	22.263	22.593	23.042	18.755
T_{basic_model} Turbo Mode	16.993	12.333	16.869	16.959	17.438	14.345
$T_{memory\_extended\_model}$ Basic Frequency	25.276	18.030	23.929	24.051	24.369	19.996
$T_{memory\_extended\_model}$ Turbo Frequency	18.255	13.022	17.282	17.37	17.599	14.442

Table 5: Modeling Results for Skylake Testbed

NxN	24x24	32x32	40x40	48x48	56x56	64x64
T_{exp}	32.275	16.737	22.128	19.435	19.469	16.343
T_{basic_model} Basic Frequency	22.856	15.210	23.249	23.702	24.325	19.016
T_{basic_model} Turbo Mode	16.635	11.07	16.921	17.251	17.705	13.84
$T_{memory\_extended\_model}$ Basic Frequency	23.014	15.533	23.782	24.496	25.428	20.480
$T_{memory\_extended\_model}$ Turbo Frequency	16.75	11.305	17.309	17.829	18.507	14.906
$T_{basic\_model\_Broadwell}$ Basic Frequency	21.766	15.534	21.592	21.911	22.345	18.207

References:

1. Elvidge, Christopher D. and Zhizhin, Mikhail and Baugh, Kimberly and Hsu, Feng-Chi: Automatic Boat Identification System for VIIRS Low Light Imaging Data. *Remote Sensing Journal*, Vol.7, Num. 3, pp. 3020 -- 3036 ISSN 2072-4292, doi: 10.3390/rs70303020 (2015)
2. Intel® VTune™ Amplifier. URL: https://software.intel.com/en-us/vtune/
3. K. L. Spafford and J. S. Vetter: Aspen: A domain spec ific language for performance modeling. *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, 2012, pp. 1 -11. doi: 10.1109/SC.2012.20
4. Treibig, G. Hager and G. Wellein : LIKWID: A lightweight performance -oriented tool suite for x86 multicore environments. Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures, San Diego CA, September 13, 2010. DOI: 10.1109/ICPPW.2010 .38 Preprint: http://arxiv.org/abs/1004.4431
5. Bardhan and D. A. Menascé: Predicting the Effect of Memory Contention in Multi-Core Computers Using Analytic Performance Models. In *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2279-2292, 1 Aug. 2015. doi: 10.1109/TC.2014.2361511
6. Skylake (server) Microarchitectures. URL: https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)
7. Markus Frank, Floriment Klinaku, and Steffen Becker. 2018. Challenges in Multicore Performance Predictions. In Companion of the 2018 ACM/SPEC International Conference on Performance Engineering (ICPE '18). ACM, New York, NY, USA, 47 -48. doi: https://doi.org/10.1145/3185768.3185773