A unified methodology for a fast benchmarking of parallel architecture

Alexandre Guerre, Jean-Thomas Acquaviva, Yves Lhuillier
CEA, LIST, Embedded Computing Laboratory, F-91191 Gif-sur-Yvette, France.
Email: name.surname@cea.fr

***Abstract*—Benchmarking of architectures is today jeopardized by the explosion of parallel architectures and the dispersion of parallel programming models. Parallel programming requires architecture dependent compilers and languages as well as high programming expertise. Thus, an objective comparison has become a harder task. This paper presents a novel methodology to evaluate and to compare parallel architectures in order to ease the programmer work. It is based on the usage of micro-benchmarks, code profiling and characterization tools. The main contribution of this methodology is a semi-automatic prediction of the performance for sequential applications on a set of parallel architectures. In addition the performance estimation is correlated with the cost of other criteria such as power or portability. Our methodology prediction was validated on an industrial application. Results are within a range of 20%.**

I. Introduction

The explosion of parallel architectures and the dispersion of the parallel programming models are questioning the traditional benchmarking techniques. Indeed parallel programming is more and more complex and requires larger and new expertise. Furthermore, the performance equation is multi-criteria: FLOPS, power, software cost. A consistent approach is to provide quantitative information on all these aspects. This allow end-users to select the best execution platform, ie. the couple architecture/programming model, in function of their set of constraints.

This paper describes a methodology which aims to quantify the relevance of an execution platform for a given application. This methodology makes two assumptions: first, it assumes that the application to analyze is in the field of image processing. Second, it assumes that a sequential version of the application is available on an x86 platform.

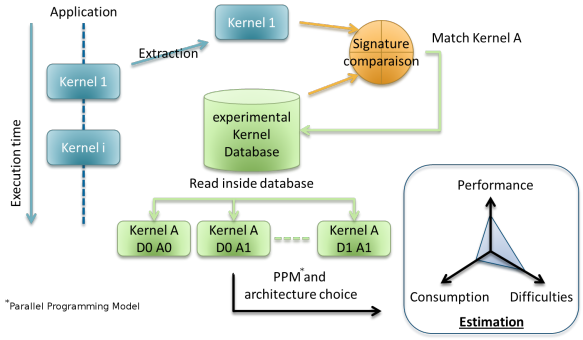


Fig. 1. Our methodology workflow to predict performance of an application.

978-3-9815370-2-4/DATE14/©2014 EDAA

Figure 1 describes the our methodology prediction chain. A preliminary and crucial point in characterization is to harness the complexity of modern architectures. Nowadays, analytical models are challenged by this complexity and simulation is not always possible due to the lack of proprietary information. To circumvent these difficulties and to allow comparison between platforms our methodology relies on micro-benchmarking. The corollary problem is the ability to establish a link between the targeted application and these micro-benchmarks. A rapid observation shows that the performance of a given algorithm differs considerably according to the architecture and the parallel programming model. To maximize the quality of the correlation between micro-benchmarks our methodology proposes to build up a knowledge database of measurements (*experimental kernel database*). Therefore, a set of relevant micro-benchmarks are manually ported to all the considered architectures with the different supported programming models. The relevance of these kernels is subject to discussion, they were selected as being representative of the workload in the image processing domain. Clearly this is not an automatic process and the selection was done based on human expertise. The second stage (*extraction*) is more automated: the application is executed within a dynamic binary instrumentation tool [1] to identify the hot spots to accelerate. A signature is calculated for each of these hot spots. With a correlation calculus, the best candidate in the micro-benchmarks database is identified and the potential on each architecture/ programming model is estimated (*signature comparison*). Using the heuristic presented in section II, the application performance is rebuilt depending on the architecture and programming model choice.

The remainder of the paper is organized as follows: the general concepts of the methodology and their implementation are detailed in section II. The result of this methodology on a real industrial application is presented in section III. Related works are addressed in section IV. At last section V concludes and presents future orientations.

II. Methodology details

A. Parallelization and tuning of a micro-benchmarks set

As explained in the former section, our methodology is based on a database composed of micro-benchmarks. Theses micro-benchmarks have to be chosen depending on their characteristics (memory access pattern, regularity, complexity). In order to limit the number of micro-benchmarks, 6 kernels are used: max 3x3, Deriche filter, FGL [2], quad-tree, summed area table and matrix multiply.

Max 3x3 is a typical filter with its 2D memory access. It realizes more memory accesses than operations. Deriche

and FGL filters are respectively an 8 times and a 4 times 1D filter. These filters have horizontal and vertical access patterns and are causal and anti-causal. The quad-tree is a recursive algorithm that calculates the variance, compares the result to a threshold and if it is greater, splits the input in 4 parts. This kernel catches the recursive pattern. The summed area table algorithm calculates for each pixel of an image the sum of all values within a rectangle between the origin and the current point. This kernel has a high dependency between each result point. The last one, matrix multiply, is well known as a simple representation of a 3D pattern.

Depending on the target architecture, all programming models are not available. Four programming models have been used: OpenMP, Farming, OpenCL and CUDA. The farming model has been developed in C using pthread library. This model creates a fix number of workers depending on the requested parallelism and creates a number of jobs greater than the number of workers. The OpenMP parallelization scheme emphazize a straightforward parallelism based on OpenMP parallel for constructs. Theses constructs are simple to implement but yet effective in the case of image processing. OpenCL and CUDA languages are used for GPU programming models. OpenCL is also used in the case of Intel processors. The last parameter to take into account in the kernel database is the size of the data being processed. Putting it all together, the database is composed of the following entries: the set of benchmarked architectures, the set of benchmarked kernels, the range of input dataset sizes, images size ranges from 256x256 to 2048x2048 pixels and the set of programming models depending on their availability on the target architecture. It is also possible for multicore architectures, to control the parallelism degree by limiting the amount of available cores used for parallel sections.

B. Application kernel extraction

To compare full applications to the kernel database, it is necessary to extract application kernels. Performing a whole application analysis would result in an averaged analysis that does not take into account the specificity of the small algorithmic sequences the application is composed of.

Defining application kernel is a hard task [3], [4], and yet a significant point in order to enable relevant analysis of the code. First in our methodology, the focus is on dynamic execution trace of the application. The contiguous aspect of kernel execution trace is especially important so that the considered kernel represents a standalone part of the initial application (not interleaved with other code). Thus, the goal is to detect dynamic instructions traces composed of a maximally limited set of static instructions. Static instructions refer to instructions of the application binary whereas dynamic instructions refer to occurrence of static instructions in an execution trace. The definition of a kernel is the minimal set of static instructions covering dynamic instructions traces of maximum length.

Our methodology includes a tool to process dynamic instructions traces in order to locate kernels. The output of this tool is a set of points representing the best application kernel candidates. For example, in the dynamic instructions sequence `a.x.y.x.y.b.x.y.x.y` of length 10, choosing 2 static instruction (`x,y`) provides a contiguous sequence `x.y.x.y` of length 4, whereas properly choosing 3 static instructions (`b,x,y`) gives a contiguous sequence `x.y.x.y.b.x.y.x.y` of length 9. Using this method, the tool provides the candidate containing a minimal set of static instructions providing a maximally long contiguous sequence.

Once the best kernel point is identified, its static instructions are extracted to find the corresponding portion of the original source code (C, C++, Fortran...). In the case, where the best kernel is not covering a major part of the whole application the method can be repeated iteratively to extract secondary kernels on the remaining part of the trace. Note that because the application kernel extraction is done on dynamic execution traces, the application should be considered along with its input dataset.

C. Kernel characterization: building a signature

Once an application kernel is identified, a second tool is used to compute its signature. The signature allows comparisons with the known kernels database. This analysis is performed on dynamic execution. The signature tool is built on an instrumented instruction set simulator that gathers dynamic information on executed instructions. The instrumentation consists in performing full data renaming (registers and memory) to locate producers and consumers of all runtime data. Because this ideal dataflow graph represents huge amount of data, online compression is performed by folding the graph relatively to application static instructions.

The signature metrics have been chosen in order to capture significant (control and dataflow) information on the kernel behavior. The metrics are the following:

Dataflow stability. Gives the average number of producer locations for each instructions. It allows to capture whether computations follow a fixed dataflow or if data are accessed using complex address computations; in the later case, streaming architectures (ex. GPUs) would not be efficient targets. Moreover, a poor dataflow stability often means poor parallelization opportunities (dependencies revealed at runtime).

Parallel aspect ratio. Computes, on the ideal dataflow graph, the ratio between ideal parallelism width and the number of executed instructions. A high value for this metric means high parallelization opportunities.

Dataflow reuse distance. Gives the average amount of time a byte of data should be stored before reuse. This metric is evaluated on the ideal dataflow graph. This allows to capture the data locality and to determine whether the kernel would privilege a high-bandwidth or a low-latency architecture.

Data volume. Evaluates the total amount of data that the code processes. This information is important since previous signature metrics are independent of data volume (computed relatively to the number of executed instructions).

These metrics are as hardware-independent as possible [5], in order to capture application information rather than architecture adequacy. The "distance" between two kernels is characterized by a correlation computation.

D. Putting it all together: performance prediction

Once the extracted application kernel is matched with a database kernel, the performance prediction can be performed. This performance prediction provides insights on the best architecture and the best programming model to use. Once a parallel programming model/architecture couple is chosen, the database is looked-up to extract a measured speedups ($m\_speedup$). Finally, to compute the final execution time on the target platform, a sequential performance ratio ($arch\_factor$) between the tested and the reference architecture is also needed. Using the extraction tools, all extracted application kernels can not overlap. Thus, it is possible to assume that these kernels will be parallelized independently. Thus, the formula (1) derived from the Amdahl Law, can be used to compute the potential execution time of the parallelized application.

$$predictedTime = seqRefTime \times archFactor + \frac{seqKernelTime \times archFactor}{mSpeedUP} \quad (1)$$

Additionally, the correlation between the extracted kernels provides a confidence coefficient that can be used to determine whether the selected database kernel is really close to the application kernel. Our methodology performs an auto-correlation of database kernels and evaluates maximum and average values for the confidence factor; minimum values are always zero and correspond to comparisons of kernel with themselves. Selected database kernels are considered good candidates when their confidence coefficient (comparing with application kernels) are below non-zero minimal confidence factors (of two distinct kernels of the database).

III. Experimental Results

A. Test platforms and measurement protocol

Currently four architectures have been characterized and fully supported in the database: Intel Xeon i7-2600, ARM Cortex A9 quadcore, Tilera TilePro64 and Nvidia Geforce GTX 580. The Intel i7-2600 is a high performance general purpose processor. It has 4 cores with hyperthreading working at 3.4 GHz for a peak power consumption of 95W. At the opposite side, the Cortex A9 of ARM is a simple quad core working at 400 MHz for a peak power consumption around 1W. It targets embedded systems and low power solutions. The TilePro64 of Tilera is a 64 cores working at 700 MHz for a 22W power envelop.

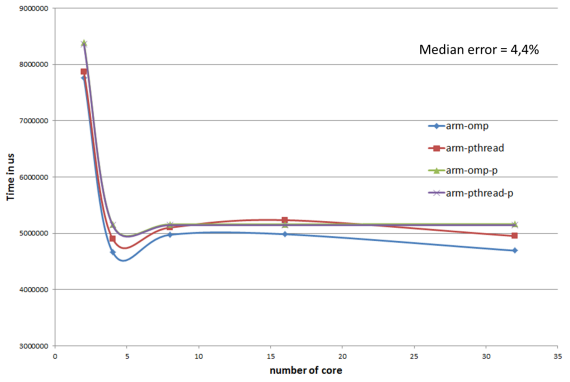
To ease measurements comparisons and to limit the experimental noise a generic API is available across the whole set of architectures/programming models. This API drives the random generation of input image and initialization of OpenCL and CUDA environments. Before measurements, a benchmarks starts by an initialization stage, which includes platform configuration when needed and input data set generation. After a warm-up stage, the time measurement of kernels[1] is realized with minimal intrusiveness using C-macros which inline assembly code probes. Measurements are repeated 30 times on the target systems (following usually recommended good practices [6]). Between two measurements a clean-up code (cache flush) is called to guarantee that kernels are always measured in the same conditions.

[1]Note that in the case of code/data offloading (GPU for example), the transfer time is also measured.

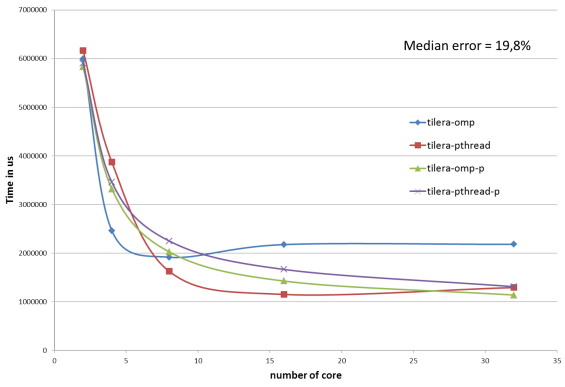
B. Case study: real application projection

The case study is an industrial application for pedestrian detection. The application core algorithm consists in searching pedestrian presence within the image. The algorithm uses a classifier [7], which decides whether a pedestrian is present at a given location and at a given scale. In order to build this classifier, an off-line statistical learning process for pedestrian appearance is conducted on many images. A supervised learning is driven using both positive (when a pedestrian is really there) and negative (extracted from the background) examples. The on-line detector is based on a preprocessing and on a classification process.

The kernel extraction automatically found 600 static binary instructions which represents 90% of the overall execution time. Using a instruction-address to source-line tool, it was possible to identify the C equivalent of the hotspot which roughly corresponds to the outermost loop of the classification. This extracted kernel was then compared to the kernel database resulting in a high correlation which identified a matrix multiplication kernel as the best candidate for performance prediction. Performance prediction was then performed using the formula described in Section II-D. Results are displayed in Figure 2. Performance prediction is performed for multiple parallelism degrees and, in the worst case (32 parallel threads), the median error is under 20%.



(a)



(b)

Fig. 2. (a) Predicted time compare to measured time for Cortex a9, (b) Predicted time compare to measured time for TilePro64.

IV. Related works

Due to the inherent difficulties of performance prediction of a given application on a platform, multiple benchmarks propose to focus on a specific domain. Such domain could be for instance image processing [8], scientific computing [9] or emerging applications [10]. To abstract issues related to programming language or execution model a drastic approach is proposed by the Berkeley dwarf [11]. In the dwarf model the implementation is variable but the important concern is the problem class (fluid dynamic, unstructured grid). In the same way, our methodology put the focus on core algorithms of an application, this is more suitable to address heterogeneity as illustrated by the Rodinia benchmarks [12]. Finer grain kernel performance prediction has been addressed by Jalby *et al.* [13], they use binary versioning to determine the performance bottlenecks and measure potential improvements. The prediction is accurate but their work is limited to x86 and mostly oriented toward architectural prospective.

Exploiting benchmark to predict performance of an application implies the ability to estimate similarity between codes. Hoste *et al* [14] use 47 parameters such as load/store ratio, arithmetic intensity, ILP or branch behavior. The distance between application and reference code is then computed using Principal Component Analysis and prediction is performed by genetic algorithms. On the particular topic of GPU performance prediction based on kernels, GROPHECY [15] proposes a similar approach. Parallel code similarity is mostly centered around communication analysis [16] or coupled with simulation [17]. Static analysis can also be combined with machine model, the paper of R. Saavedra and A.-J. Smith [18] proposes a framework where static information are injected in a machine model and cross-correlated with micro-benchmarks measurements. Despite being focused on Fortran code, this work is an important inspiration of our methodology.

V. Conclusion

This paper presents a methodology and its implementation in a tool chain for performance prediction. The main contribution of this work is the ability to produce performance prediction for a range of parallel architectures based on the characterization of a sequential version of an application. The methodology is following a kernel approach using binary instrumentation. We advocate that addressing the performance problem at the granularity of kernels is adapted to the massive trend of heterogeneous and hardware accelerated platforms. In order to deal with the complexity of modern architectures our methodology relies heavily on measurements through a database of microbenchmarks. In our methodology, the fact that performance is not limited to execution speed but also includes power or more complex metrics is a clear contribution of this work. Overall the error margin of the prediction for our industrial application test case is below 20%. Performance prediction is currently narrowed to the field of image processing but nothing prevents an extension toward other domains.

References

[1] D. I. August, J. Chang, S. Girbal, D. G. Pérez, G. Mouchard, D. A. Penry, O. Temam, and N. Vachharajani, "Unisim: An open simulation environment and library for complex architecture design and collaborative development," *Computer Architecture Letters*, vol. 6, no. 2, pp. 45–48, 2007.

[2] F. G. Lorca, L. Kessal, and D. Demigny, "Efficient ASIC and FPGA implementations of IIR filters for real time edge detection," in *Image Processing, 1997. Proceedings., International Conference on*, vol. 2. IEEE, 1997, pp. 406–409.

[3] C. G. Nevill-Manning and I. H. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *CoRR*, vol. cs.AI/9709102, 1997.

[4] H. Hayashizaki, P. Wu, H. Inoue, M. J. Serrano, and T. Nakatani, "Improving the performance of trace-based systems by false loop filtering," in *ASPLOS*, 2011, pp. 405–418.

[5] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *Micro, IEEE*, vol. 27, no. 3, pp. 63–72, 2007.

[6] A. Mazouz, S. A. A. Touati, and D. Barthou, "Performance evaluation and analysis of thread pinning strategies on multi-core platforms: Case study of spec omp applications on intel architectures," in *HPCS*, 2011, pp. 273–279.

[7] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, 2001, pp. I–511 – I–518 vol.1.

[8] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems," in *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, ser. MICRO 30. Washington, DC, USA: IEEE Computer Society, 1997, pp. 330–335.

[9] J. M. Arnold, D. A. Buell, and E. G. Davis, "Splash 2," in *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, ser. SPAA '92. New York, NY, USA: ACM, 1992, pp. 316–322.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ser. PACT '08. New York, NY, USA: ACM, 2008, pp. 72–81.

[11] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams *et al.*, "The landscape of parallel computing research: A view from berkeley," UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Tech. Rep., 2006.

[12] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, oct. 2009, pp. 44 –54.

[13] W. Jalby, D. Wong, D. Kuck, J. Acquaviva, and J. Beyler, "Measuring computer performance," *High-Performance Scientific Computing*, pp. 75–95, 2012.

[14] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. John, and K. De Bosschere, "Performance prediction based on inherent program similarity," in *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. ACM, 2006, pp. 114–122.

[15] J. Meng, V. Morozov, K. Kumaran, V. Vishwanath, and T. Uram, "Grophecy: Gpu performance projection from cpu code skeletons," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE, 2011, pp. 1–11.

[16] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of splash-2 and parsec," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 86–97.

[17] A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha, "A framework for performance modeling and prediction," in *Supercomputing, ACM/IEEE 2002 Conference*. IEEE, 2002, pp. 21–21.

[18] R. Saavedra and A. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM Transactions on Computer Systems (TOCS)*, vol. 14, no. 4, pp. 344–384, 1996.