CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

WCET Measurement-based and Extreme Value Theory Characterisation of CUDA Kernels

Kostiantyn Berezovskyi*
Luca Santinelli
Konstantinos Bletsas*
Eduardo Tovar*

* CISTER Research Center

CISTER-TR-141009

2014/10/08

WCET Measurement-based and Extreme Value Theory Characterisation of CUDA Kernels

Kostiantyn Berezovskyi*, Luca Santinelli, Konstantinos Bletsas*, Eduardo Tovar*

*CISTER Research Center

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: kosbe@isep.ipp.pt, ksbs@isep.ipp.pt, emt@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

The massive computational power of graphics processor units (GPUs), combined with novel programming models such as CUDA, makes them attractive platforms for many parallel applications. This includes embedded and real-time applications, which, however, also have temporal constraints: computations must not only be correct but also completed on time. This poses a challenge because the characterisation of the worst-case temporal behaviour of parallel applications on GPUs is still an open problem. To address this situation, this paper proposes a measurement-based and statistical approach for the probabilistic characterisation of the worst-case execution time of such an application.

WCET Measurement-based and Extreme Value Theory Characterisation of CUDA Kernels

Kostiantyn Berezovskyi[+], Luca Santinelli[*], Konstantinos Bletsas[+] and Eduardo Tovar[+]
[+]CISTER/INESC-TEC ISEP, Portugal, [*]ONERA Toulouse, France

ABSTRACT

The massive computational power of graphics processor units (GPUs), combined with novel programming models such as CUDA, makes them attractive platforms for many parallel applications. This includes embedded and real-time applications, which, however, also have temporal constraints: computations must not only be correct but also completed on time. This poses a challenge because the characterisation of the worst-case temporal behaviour of parallel applications on GPUs is still an open problem. To address this situation, this paper proposes a measurement-based and statistical approach for the probabilistic characterisation of the worst-case execution time of such an application.

1. INTRODUCTION

Graphics processor units (GPUs) offer processing capacity orders-of-magnitude greater than CPUs. Novel parallel programming models, such as Nvidia CUDA and OpenCL, brought us General-Purpose GPU (GPGPU) computing: the use of GPUs as accelerators for computationally intensive (non-graphics) functions.

But GPUs are designed for high throughput via massive parallelism; not via executing any single thread particularly fast. Therefore, the applications best-suited for GPUs: (i) are easily decomposable in thousands of parallel threads; (ii) have minimal dependency across data (no need for synchronisation; maximum parallelism); (iii) are computationally intensive, to justify the costly copying of the GPU input and output over the bus.

To provide temporal guarantees for GPU-accelerated applications, we need a technique for upper-bounding their execution time on the GPU. Traditional WCET analyses for CPUs are inapplicable because they focus on the WCET of a single thread. Yet, on GPUs the result is pieced together from thousands of threads, competing for GPU resources, and we are not interested in the WCET of any single thread in particular. Rather, we seek to bound the time, from when the earliest GPU thread starts executing until all of them have completed.

RTNS 2014 , October 8 - 10 2014, Versailles, France

http://dx.doi.org/10.1145/2659787.2659827 .

To that end, we undertake a measurement-based probabilistic approach, based on Statistic Analysis and Extreme Value Theory (EVT). This technique allows the derivation of *highly accurate* estimates on the probability that any run of the GPU application exceeds a respective time threshold, even if such high execution times are *not* observed in *any* of the measurements. It advances the state of the art because it accurately captures the overall behaviour of the memory subsystem. By comparison, our prior approaches [3, 4] for deriving WCET estimates for GPU applications were optimistic in their assumptions on cache misses and the memory subsystem in general. Extending them, as originally intended, to also consider the effects of cache and memory, was not practical for two reasons. First, due to tractability issues, inherent in those approaches, which kicked in when considering long-latency operations (e.g. hundreds of cycles for an L1 miss). Secondly, because the exact cache architectures and replacement policies for modern GPUs are trade secrets, thus not openly documented. A probabilistic measurement-based approach bypasses both hurdles.

This paper considers the NVIDIA CUDA programming model but the main idea can also be applied to other analogous technologies (such as AMD APP).

In terms of outline, the next section offers additional background and discusses related work. Section 3 elaborates on measurement collection and Section 4 offers background on the statistical analysis of the measurements and on EVT, which we use to obtain highly accurate probabilistic WCET estimates. Section 5 discusses our experiments. Section 6 concludes.

2. BACKGROUND

The *Compute Unified Device Architecture* (CUDA) programming model by NVIDIA implements the *stream processing* computational paradigm on GPUs for general-purpose computation conveniently and efficiently. In stream processing, there is a set of input data (the stream) upon all of which a series of operations (termed the *kernel function*; not to be confused with operating system kernels) is performed. In CUDA, this is implemented as numerous ultra-lightweight identical *threads* (instances of the CUDA kernel) with minimal or no data dependencies, designed for execution in parallel. Because CUDA threads have a single-GPU-cycle context switch, they can efficiently extract all the available potential for parallelism. When a thread stalls, the GPU switches to executing another one. And even if a stall takes hundreds of cycles (e.g. to access memory), the functional units of the

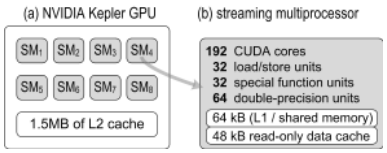


Figure 1: The NVIDIA Kepler GK104 has 8 SMs. Each SM has many CUDA cores and load/store, special function and double precision units.

GPU can be kept almost always busy this way.

Characteristically, under CUDA, at any time, groups of 32 threads (termed *warps*) execute in lockstep. Namely, the threads from the same warp (i) execute during the same cycles as each other and (ii) execute the same kernel instruction. Although there are exceptions to this (e.g. when the threads of a warp diverge in control flow), they are avoided by design, when possible, for performance reasons.

2.1 On GPU architectures and CUDA

Modern GPUs are immensely parallel architectures. A GPU (Figure 1) contains several "Streaming Multiprocessors" (SMs). Each SM is a complex manycore in itself, as it includes many (i) CUDA cores, for integer and floating-point arithmetic, (ii) "load/store" units that load data from/store data to cache or DRAM, (iii) special function units, implementing sine, cosine, square root etc in H/W and (iv) double precision (64-bit) units.

For example the Kepler GK104 [26] has 8 SMs and supports gigabytes of global GPU memory. The SMs share 1.5 MB of L2. Each SM has (i) a very fast dedicated memory (64 KB), divided into shared memory and L1 cache (configurable as 48/16, 32/32 or 16/48 KB) and (ii) a 48 KB read-only cache. The L1, shared among all threads on a given SM, has register-like latency. But many aspects (e.g. replacement policy) of this complex memory hierarchy are not publicly documented.

At run-time, warps are bundled together in groups termed *thread blocks* and each thread block is sent to one SM for execution. Each SM has a few thread blocks assigned to it at any time. Thread blocks do not migrate among SMs. The CUDA engine tries to keep the processing units of each SM busy but exactly how warps are dispatched is not publicly documented.

2.2 Work on the timing behaviour of GPUs

Characterising the timing behaviour of such complex architectures, with so little available information, is challenging. Nevertheless, serious efforts are made to either make GPU computing more time predictable or to derive appropriate WCET analysis.

Many works have attempted to make the scheduling on the GPU more predictable [2, 18] and provide multitasking [19] among different GPU contexts and efficiently manage shared resources. In [17], data transfers between CPU and GPU are made preemptable, to reduce the related blocking times. The GPU management infrastructure in [28] supports data transfer and computation overlap and multi-GPU systems. The lock-based management framework for multi-GPU systems in [11] allocates GPU resources to tasks according to an execution cost predictor that infers costs (e.g. computation time, data transfer delay) from a few runs. Mangharam et al. [24] discussed adaptive runtime scheduling of anytime algorithms. In the case-study provided, the worst-case scenarios for GPU kernels are empirically derived by running few experiments.

Other works seek to provide WCET analysis for GPUs. In [3] we presented the first (to the best of our knowledge) work aimed at computing safe upper bounds on the WCET of a CUDA kernel. That approach was ILP-based and safely extrapolated the WCET from that of a smaller problem instance (i.e. fewer threads). However, it suffered from many limitations: it only considered execution on a single SM, it was not tractable with respect to longer kernels (due to control variable explosion) and crucially, the output was only safe subject to an *optimistic* assumption regarding cache misses. Therefore, we next [4] formulated an alternative more tractable metaheuristic-based technique for estimating the WCET. That estimate, however, was no longer provably safe, hence only useful for soft (not hard) real-time systems. Moreover, optimistic assumptions regarding cache misses remained. In fact, our efforts, to extend [3] or [4] to account for the effects of the memory subsystem, were unfruitful, mainly due to (i) inherent tractability problems when modeling latencies of hundreds of cycles (as in L1 misses) and (ii) the lack of public documentation.

Betts *et al.* also presented [5] two techniques for estimating the WCET of CUDA kernel functions. They relied on the simulator GPGPU-sim [1], configured for NVIDIA Fermi. The first technique (*dynamic*) estimates from the respective high-water mark measurements the maximum "release jitter" (delay in launch, measured from the kernel launch) and WCET (including the effects of contention for shared resources, e.g., cache, GPU main memory) of the GPU warps. The second technique (*hybrid*) assumes a fixed delay for launching of each additional warp and uses static analysis based on instrumentation point graphs annotated with execution time parameters obtained from the measurements. This assumes thread blocks arriving in "waves" and processed in round-robin. However, static instrumentation point graphs tend to be pessimistic; conversely, high-water mark times may be optimistic.

This brings us to this work, which uses measurements on *real* hardware but, through Statistic Analysis and EVT, can "predict" worst-case timing behaviour even when that is not observable in the high-water mark times. In using measurements, we also largely sidestep the lack of public knowledge about the characteristics of the memory subsystem, which hampered us in previous work. Next we describe this new approach.

3. ON COLLECTING MEASUREMENTS

In the typical CUDA setup, the following sequence of actions is performed by a CUDA-C program [16].

S1: The program allocates memory on the host for the input and output of the CUDA kernel.

S2: The program allocates[1] memory on the GPU for the input and output of the CUDA kernel.

S3: The program initiates[1] the copying of the input from host memory to GPU memory. This is normally a

[1]Via the high-level CUDA Run-time API or directly the Driver.

blocking operation, unless the copied data is less than 64KB ([27], Section 3.4.5.1).

S4: The program launches the CUDA kernel. This operation is non-blocking: the driver returns control to the CPU immediately after the launch[2].

S5: The kernel executes on the GPU until completion. In parallel, the program on the host polls on the status of its completion.

S6: Upon completion of the kernel, the program copies[1] the output of the CUDA kernel from GPU memory to host memory.

S7: The program continues its execution on the host.

The execution time of the kernel corresponds to stage S5. Let that be denoted as T^{DEV}. However, the combined duration of stages S2 to S6 is also of interest, since it determines the acceleration attained via CUDA. Let us denote that by T^{HOST}. If determining T^{DEV} analytically (which our earlier work attempted) is hard, for T^{HOST} it is even more so, since it also includes the execution of the CUDA driver and the I/O latency for copy over the PCI-e bus. Therefore, we attempt to characterise both by collecting measurements over a sufficiently large number of runs and applying EVT.

To measure T^{HOST} we used standard Linux primitives for reading the system time. We placed those system calls just before S2 and at the start of S7. Accurately measuring T^{DEV} is harder because the GPU cannot be probed. Any instrumentation code added to the kernel would be executed by *all* CUDA threads so it would have to be extremely lightweight/non-intrusive for the cumulative effect on T^{DEV} to not be significant. Recall that T^{DEV} is the interval *from* when the first kernel instruction by some thread (warp) executes *until* the last kernel instruction by some thread (warp) is completed. The tricky part is that we cannot know *a priori* which warp starts to execute first and which one completes last. We deal with this as follows:

There is a special *clock-register* on each SM, which counts GPU cycles. We read/record its value via manually inserted assembly code, at the start/end of each thread. A *naive* approach would use two respective *per-thread* variables, start_cycle and end_cycle. But this would use too much shared memory (out of the 48 KBs, at most, per SM) or else thrash the L1 cache, significantly altering the timing behaviour. Hence we use a single *per-SM* pair of start_cycle and end_cycle variables (Figure 2), and leverage the fact that execution on the GPU is in-order. The first thread to execute, whichever that is, sets the start_cycle variable. All subsequent threads detect this (if-condition at line 1) and avoid overwriting its value. Upon completion, all threads write to the stop_cycle variable (line 4), which means that the last value written to it is by the latest thread to complete. Then T^{DEV} (in GPU cycles) is derived[3], with p denoting the index of the SM, as:

$$T^{\text{DEV}} = \max_p\{\text{end\_cycle}[p]\} - \min_p\{\text{start\_cycle}[p]\} \qquad (1)$$

[2]Synchronous semantics (i.e. self-suspension until the GPU-side computation completes) can still be obtained, e.g., via custom CPU-side programming.

[3]In the rare case of clock-register overflow, the above code does not work. We detect/discard such data, offline.

```
//start_cycle initialised to MAXINT
//if-condition TRUE only for the earliest thread
1. if (start_cycle > CLK_REG)
2.  start_cycle := CLK_REG;
3.  —– (The instructions of the kernel go here...) —-
4. stop_cycle := CLK_REG; //overwritten by every thread
```

Figure 2: High-level overview of the measurement-collecting assembly inserted in each GPU thread.

To apply EVT, we need such measurements from many runs of a given CUDA kernel. We therefore developed a tool that repeatedly (i) launches the same kernel and (ii) records its timing measurements. To eliminate interference from screen rendering, we switch off the windowing system entirely. To guarantee the safe application of the EVT, the number of runs must be large enough; in the order of thousands, as has been demonstrated. We conservatively opted for 10^5 runs, which, as expected, proved more than enough.

4. STATISTICAL ANALYSES OF EXECUTION TIME

Statistical estimations of worst-case execution time are becoming popular within the real-time community, [14, 8]. They lead to the notion of probabilistic WCET (pWCET), alternative to the deterministic WCET, as distributions of values C_j with an associated probability of being the WCET. C_j upper-bounds the task execution time with a probability p_j. $1 - p_j$ is the probability for a task instance having a bound on its execution time different than C_j.

DEFINITION 1 (PROBABILISTIC WCET). *Given $\mathcal{C}_i$, the distribution of execution time measured in a certain configuration/condition i, the probabilistic Worst-Case Execution Time distribution $\mathcal{C}^*$ of a task is a tight upper bound on the execution time distribution $\mathcal{C}_i$ of all possible execution conditions*[4]*. Hence, $\forall i$, $\mathcal{C}^*$ is larger than or equal to $\mathcal{C}_i$. In notation: $\mathcal{C}^* \succeq \mathcal{C}_i \ \forall i$.*

The total ordering among distributions is defined such that, a distribution $\mathcal{C}_j$ is greater than or equal to a distribution $\mathcal{C}_k$, $\mathcal{C}_j \succeq \mathcal{C}_k$, iff $P\{\mathcal{C}_j \leq d\} \leq P\{\mathcal{C}_k \leq d\}$ for any d and the two random variables are not identically distributed (two different distributions), [10]. The tightest possible pWCET distribution would be the exact pWCET, which is unknown. However, we still need to come up with a *safe* pWCET estimation, meaning a pWCET estimation $\mathcal{C}^*$ that is greater than or equal to the (unknown) exact pWCET. And the only information we can rely on, for constructing such a pWCET estimation is the set of measurements ($\{\mathcal{C}_i\}$) and the execution conditions (i) under which they were taken.

The probabilistic worst-case execution time can also be defined in terms of the exceeding thresholds and the 1-Cumulative Distribution Function (1-CDF) representation. Given a probability of exceedence p^*, C^* is the worst-case execution time such that $P\{\mathcal{C}^* \geq C^*\} \leq p^*$. Alternative to the pWCET distribution, we can call minimum probabilistic worst-case execution time the tuple $\langle C^*, p^* \rangle$. In our experiments we consider $p^* = 10^{-6}$, $p^* = 10^{-9}$, and $p^* = 10^{-12}$.

[4]We use calligraphic letters to represent probability distributions. Non calligraphic letters are for single values.

Measurements, when used in conjunction with statistical approaches such as the EVT, contribute at estimating safe pWCETs. On their own, measurements are *not* enough to obtain pWCETs since they may lack completeness: through the measurements there is no guarantee to have experienced all the execution conditions. Nonetheless, measurements are important for extracting observable features such as average behaviours and trends that can appear while executing tasks. Extreme value analysis is for the statistical inference on the tail region of a distribution function. The statistical estimation of the pWCET makes use of the EVT for exploring rare events, wherein the WCET and its probabilistic version pWCET should lie. In the following we state the basics for the EVT that we apply in our framework.

Classical EVT discusses the possible limiting laws for the maximum $M_n = max\{X_1, X_2, \ldots, X_n\}$ of n independent identically distributed (i.i.d.)[5] random variables $\{X_n\}$ as n tends to infinite[6]. [13].

THEOREM 1 (FISHER-TIPPETT-GNEDENKO EVT). *Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically-distributed random variables, and $M_n = \max\{X_1, \ldots, X_n\}$. If a sequence of pairs of real numbers a_n, b_n exists such that each $a_n > 0$ and*

$$\lim_{n\to\infty} P\left\{\frac{M_n - b_n}{a_n} \leq x\right\} = \mathcal{G}(x), \tag{2}$$

where $\mathcal{G}$ is a non degenerate distribution function, then the limit distribution $\mathcal{G}$ belongs to either the Gumbel, the Frechet or the Weibull family. These can be grouped into the generalised extreme value distribution.

Theorem 1 expresses the EVT theory in case of independence among samples: the maxima of an i.i.d. sequence converge to a Generalised Extreme Value (GEV) distribution $\mathcal{G}_\xi$, which admits the following Cumulative Distribution Function (CDF):

$$\mathcal{G}_\xi(x) = \begin{cases} \exp(-\exp(-x)), & \text{if} \quad \xi = 0 \\ \exp\left(-(1+\xi x)^{-\frac{1}{\xi}}\right), & \text{if} \quad \xi \neq 0 \end{cases}. \tag{3}$$

The GEV distribution $\mathcal{G}_\xi$ can be of three distinct types, characterised by $\xi = 0$, $\xi > 0$ and $\xi < 0$, which correspond to the Gumbel, Fréchet and Weibull distributions, respectively.

Usually, the EVT is established for i.i.d. observations, and previous works have linked the safety of EVT estimations to that hypothesis. Therein, it is claimed that if both independence and identical distribution are verified, the EVT distribution tail projection can be considered as a safe pWCET estimation, [8].

However, more recent developments showed that independence is *not* a necessary hypothesis for the EVT. Leadbetter et al. [21], Hsing [15] and Northrop [25] developed EVT for stationary weakly dependent time series. The latter two references also established statistical tools for use under that assumption.

THEOREM 2 (LONG RANGE INDEPENDENCE EVT, [20]). *Let $\{X_n\}$ be a stationary sequence such that $M_n = max\{X_1, \ldots X_n\}$ has a non-degenerate limiting distribution $\mathcal{G}$ as in*

$$P\{a_n(M_n - b_n) \leq x\} \xrightarrow{d} \mathcal{G}(x), \tag{4}$$

for some constants $a_n > 0$, b_n. Suppose that

$$D(u_n): \ |F_{i_1,\ldots,i_p,j_1,\ldots,j_q}(u_n) - F_{i_1,\ldots,i_p}(u_n) \cdot F_{j_1,\ldots,j_q}(u_n)| \leq \alpha_{n,l},$$

where $lim_{l\to\infty} lim_{n\to\infty} \alpha_{n,l} = 0$, holds for all sequences u_n given by $u_n = x/a_n + b_n$, $-\infty < x < \infty$. Then $\mathcal{G}$ is one of the three classical types: Weibull, Frechet, Gumbel.

The distributional mixing condition $D(u_n)$ alone is sufficient to guarantee that the central classical result concerning the possible extremal types (the EVT), holds also for stationary sequences. Both a_n and b_n can be computed as best-fit of the input observations. $D(u_n)$ is called long-range dependence conditions, and if satisfied it means that there is no dependence between far away observations.

In [20] it is introduced the local dependence condition $D'(u_n)$, $D'(u_n): \ lim_{n\to\infty} \sup n \cdot \sum_{j=2}^{n/k} P\{X_1 > u_n, X_j > u_n\} \to 0$, slightly more constraining than $D(u_n)$, seeking to assure the independence between close-in-time observations. If $D'(u_n)$ holds with $k \to \infty$ and for each $u_n = x/a_n + b_n$, then the particular distribution type which applies is the same as if the sequence $\{X_n\}$ were i.i.d, with the same marginal distribution function, and the same normalizing constants a_n, b_n may be used.

THEOREM 3 (EXTREMAL INDEPENDENCE EVT, [20]). *Let $\{X_n\}$ be a stationary sequence with marginal distribution function $\mathcal{F}$ such that $M_n = max\{X_1, \ldots X_n\}$, and $\{u_n\}$ a sequence of constants such that $D(u_n)$, $D'(u_n)$ hold. Let $0 \leq \tau < \infty$, then*

$$P\{M_n \leq u_n\} \xrightarrow{d} exp(-\tau) \tag{5}$$

iff

$$n \cdot [1 - F(u_n)] \to \tau. \tag{6}$$

Theorem 3 states that if both $D(u_n)$ and $D'(u_n)$ are satisfied, the resulting EVT is equal to the one obtained in case of observing independence.

Chernick [7], extending Loynes [22], showed that, if for each $\tau > 0$, $u_n = u_n(\tau)$ is defined to satisfy Equation (6), under $D(u_n)$ conditions alone, then any limit function for $P\{M_n \leq u_n(\tau)\}$ must be of the form

$$P\{M_n \leq u_n(\tau)\} \xrightarrow{d} exp(-\theta\tau), \tag{7}$$

for some θ with $0 \leq \theta \leq 1$.

The parameter θ, called the *extremal index* of the time series, is a measure of clustering at the extremes. It is useful for analysing the behaviour of the extremes in the tail; a small θ means greater clustering of the largest observations, i.e., higher dependence between observations; a value of $\theta = 1$ i.e., no extremal clustering, denotes independence.

Assuming $\overline{\mathcal{C}}$ the pWCET EVT estimation in case of stationarity, and $\widehat{\mathcal{C}}$ the pWCET EVT estimation in case of independence. Supposing that the execution time measurements

[5]Readers not already familiar with the concept of independent and identically distributed variables, may peek ahead to Sections 4.1.1 and 4.1.2, where we formally define and discuss these concepts.

[6]$\{X_n\}$ is the sequence of observations; each observation results from a distribution $\mathcal{X}_n$. The identical distribution hypothesis assumes that all the observations follow the same distribution, thus $\mathcal{X}_1 = \mathcal{X}_2 = \ldots \mathcal{X}_n = \mathcal{F}$. In our case, both observations and distributions refer to execution time, hence there is equivalence between $\{X_n\}$ and $\{C_n\}$ as well as $\mathcal{X}_n$ and $\mathcal{C}$, in terms of representation.

in the two cases follow the same marginal distribution, it is $\overline{\mathcal{C}} = \widehat{\mathcal{C}}^{\theta}$ with $\overline{\mathcal{C}} \succeq \widehat{\mathcal{C}}$, [6]. In case of independence at the extremes, $\theta = 1$, $\overline{\mathcal{C}} \approx \widehat{\mathcal{C}}$, [6]. Once one of the above hypotheses (either independence, extremal dependence, or stationarity) is satisfied the EVT provides pWCET estimations which are greater than or equal to the exact pWCET. In here, the safety of pWCET EVT estimations.

In the present paper we apply these theoretical developments to the execution time analysis and safe pWCET estimations. In doing so, we consider the Gumbel distribution for EVT pWCETs, as it has been demonstrated to be the most appropriate distribution for execution times, [8].

4.1 On the Verification of the EVT hypotheses

Hypothesis testing means to decide, from a number of observations, whether one should consider a property to be true or not. We may never know for sure, but a statistical test will give us guidance in making a decision. In statistics we can state this problem using two hypotheses: H_0 (named null hypothesis) that denotes the hypothesis that the property is true, and H_1 (namely alternative hypothesis) denoting the hypothesis that the property is false. It has to be decided whether to accept or reject the hypothesis H_0 based on a sample (set of observations). The ρ-value is the result for hypothesis testing. ρ is the probability of obtaining a test result at least as extreme as the one that was actually observed, assuming that the null hypothesis is true. Normally, $\rho > 0.05$ validates H_0; $\rho \leq 0.05$ rejects H_0, thus validates H_1. Various alternative approaches exist for calculating such an ρ-value, leading to different hypothesis tests; we discuss, later on in this section, those ones that we will be using.

4.1.1 Independence of Observations

In statistics, a collection of random variables is independent (i.) if all the random variables are mutually independent. By this, we mean whether individual observations within the same execution trace are correlated with each other or not. If knowing one observation tells you something about another, then the observations are dependent; if knowing one observation tells you nothing about another, in that case they are independent.

A test applied in [8] aims at proving that samples are independent looking for randomness. This is called *runs test*, where randomness is sought within the observed data series by examining the frequency of "runs"; a "run" is a series of similar responses.

In this paper we look to extend independence tests from runs test, since randomness is not formally sufficient to verify independence. This type of independence can not be proven or tested except for time series. Time series tests are based on autoregression and autocorrelation. In particular, we aim at verifying stationarity, which gives more information about the observation traces and applying it to characterise system execution behaviour while looking for the worst-case execution conditions.

4.1.2 Identical Distribution of Observations

In statistics, a collection of random variables is identically distributed (i.d.) if each random variable has the same probability distribution. A common test for verifying identical distribution in observations is the two-sample *Kolmogorov-Smirnov* test: The trace of observations is divided into two sets which are compared, to verify whether they represent the same distribution.

4.2 Statistical Analyses

In practical applications, the independence assumption may or may not be realistic. To test how realistic it is on a given execution time data set, the autocorrelation can be computed with *lag plots*, or a *turning point test* can be performed. These are to test the relationship that exists between measured observations. We apply them in order to extract patterns and behavioural models which could describe the observed system behaviour. In particular, we employ autocorrelation tests together with the notion of stationarity, which indirectly quantify the statistical independence between observations.

With no means of formalism, a process is stationary if its mean variance and autocovariance structure do not change over time. This is what is called weak form of stationarity, which means flat-looking observations, no trend, constant variance over time, and no periodic fluctuations or autocorrelation.

Autocorrelation, in a time series, is the similarity between observations as a function of the time lag between them. In our case, the time is given with the order of observations, thus lags are in terms of number of observations. The sample Auto Correlation Function (ACF) is one of the most important assessment tools for detecting data dependence and fitting models to data. Although the model is not faced at first, the observed data $\{X_1, \ldots, X_N\}$ are known.

An *autoregressive (AR) model* instead, is a representation of a type of random process. The AR model describes the underlying stationarity model of a trace of observations (time series): $AR(0)$, the sequence of observations has no dependence between the observations - white noise; $AR(1)$, a process where, with a positive parameter, only the previous observation in the process and the noise term contribute to the output - very very light dependence; $AR(2)$, a process where the previous two observations and the noise term contribute to the output. And so it goes on, increasing the dependence pattern between observations.

We also use the *Ljung-Box* test, which looks for any significant evidence for non-zero correlations between lags. Large ρ-values from the test suggest that the series is not stationary, thus there is no trend between consecutive observations; this supports non stationarity, and thus independence.

Valuable to time series analysis is also the test called *extremogram* [9], where the dependence at the extremes is estimated. The extremogram defines an analogue of the autocorrelation function, which depends only on the extreme values in the sequence of observations.

Finally, to compare with the independence case, there is the extremal index θ of the observations which is another tool for measuring the dependency of extreme values. We make use of the *blocks test* to compute θ, based on estimators [12]. We stress that there is equivalence between the extremogram and the extremal index, for evaluating extremal dependences, thus ultimately the EVT applicability. For completeness we apply both, although just one of the two would have been enough to verify extremal behaviour of observations.

Tests such as the above allow us to conclude about the stationarity of execution time observations and their eventual extremal dependence. As earlier argued, under those

circumstances it *is* still possible to derive safe EVT distributions, thus safe pWCET estimations. More importantly, the stationarity helps with describing the execution behaviour and points out to us which are the worst-case conditions necessary, in order to safely conclude about pWCETs.

5. EXPERIMENTS

Our testbed used a Kepler GK104 with 8 SMs (Figure 1), configured with 32KB of shared memory and 32KB of L1 each. As benchmark, we developed in CUDA a Voronoi diagram generator, according to the raster-coloring massively parallel approach [23]. Informally, a Voronoi diagram for a 2D-plane and K points on it, divides the plane into tiles, each tile consisting of the points in the plane closer to one of the K points than to any other. For a 2D-raster, this is formed by calculating, for every pixel, the distance to each of the K points. Our application uses a separate thread per pixel. Therefore, the raster size (X by Y) determines the number of threads, whereas the number of points K determines the workload of a thread. For valid comparisons (same per-thread workload) we used $K = 32$ in all setups and simply varied the number of threads. The first setup (VOR-1) used X=Y=32 which corresponds to 1024 threads (32 warps), the maximum thread block size in Kepler. The other setups involved 8, 28 and 32 thread blocks of this size. The execution times are in *nsec*.

5.1 Timing Analysis

The experiments made provide execution time measurement traces, to be statistically tested. As we will proceed to show, although the T^{DEV} and T^{HOST} traces behaved very differently, all traces, upon testing, indeed support the conditions that permit the safe application of EVT.

Table 1 groups the numerical results of the independence tests carried out, i.e. runs test (runs), Ljung-Box (LB), and autoregressive (AR). These results reveal the **independence** of the T^{DEV} case; hence realistic cases could be independent, and the EVT could be applicable with no need for artificially induced randomicity, as made in [8] with random replacement caches. Instead, the T^{HOST} traces are **not independent, but stationary**. This is due to the filtering effects that HOST exercises, which reduce variability and thus the independence of the observations. This stationarity is present at different degrees in the 4 different traces of T^{HOST}, but EVT is still applicable to all of them (Eq. 7).

The combination of the autocorrelation tests, the stationary tests and the extremogram (Figure 5) gives more accuracy and completeness to the independence/stationarity verification than just the runs test. For example, in case of VOR-32 T^{HOST}, the runs test would have concluded about the trace independence; in reality though, it exhibits stationarity – and, in particular, a strong stationary relationship ($AR(22)$).

Noticeably, for T^{DEV} the AR is at most 1 indicating very very light dependence; together with the LB test with $\rho \geq 0.1424$, thus no evidence of stationarity at all. This allows us to confirm the independence of the observations. With T^{HOST}, AR is larger than 11, revealing stronger dependence between observations in the form of stationarity; LB has small ρ. Crucially for the applicability of EVT, the stronger stationarity of the T^{HOST} cases does not reflect into dependence of extreme observations, being the exponential trend of the ACFs with respect to lags. This is also supported by the extremogram results, in Figure 5. In there, the extremogram estimation $\hat{\rho}(h)$ varying lag h is represented. Small $\hat{\rho}$-values i.e., less than 0.05 suggest that the series has no dependences at the extremes. The extremal index θ confirms that, hence the resulting EVT pWCET estimation for T^{HOST} is equal to the one in case of full independence, Theorem 3, being $\theta \approx 1$.

	T^{DEV}	T^{HOST}
VOR-1 runs (ρ)	0.3175	$5.235e-13$
VOR-8 runs (ρ)	0.7844	$< 2.2e-16$
VOR-28 runs (ρ)	0.664	$1.336-07$
VOR-32 runs (ρ)	0.5288	0.6189
VOR-1 KS (ρ)	0.9987	0.267
VOR-8 KS (ρ)	0.9601	0.532
VOR-28 KS (ρ)	0.6104	0.391
VOR-32 KS (ρ)	0.727811	0.5861
VOR-1 LB (ρ)	0.7407	$< 2.2e-16$
VOR-8 LB (ρ)	0.1424	$4.622e-07$
VOR-28 LB (ρ)	0.9205	$< 2.2e-16$
VOR-32 LB (ρ)	0.9715	$6.988e-05$
VOR-1 AR	1	26
VOR-8 AR	0	12
VOR-28 AR	0	22
VOR-32 AR	0	22
VOR-1 θ	1	1
VOR-8 θ	0.992	1
VOR-28 θ	1	1
VOR-32 θ	1	0.994

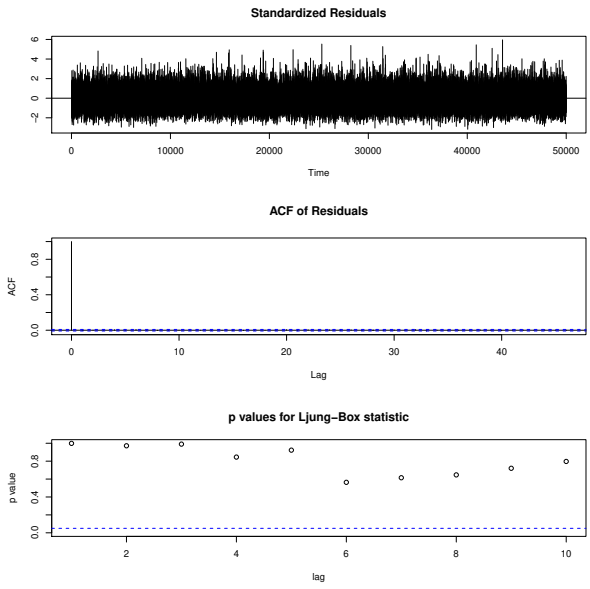
Table 1: Independence, stationarity and extremal tests.

The trends we could find in the measurement-bases distributions through the stationarity tests, therefore give us support to further statistically investigate measurements seeking the worst-case execution conditions.

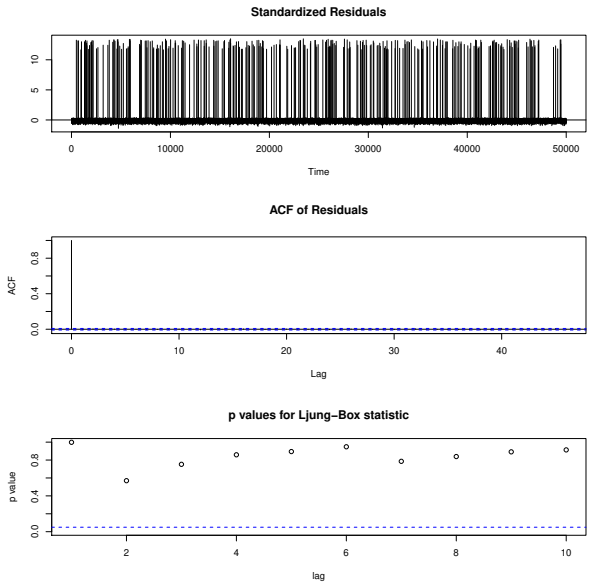
The identical distribution, Kolmogorov-Smirnov (KS) test, is verified for all traces, with $\rho > 0.05$. It suffices to check if the observations follow the same distribution: indeed, this is always the case whenever the observations are taken with the same execution conditions.

To further comment on the different behaviour of T^{DEV} and T^{HOST} cases, notice the differences in Figure 3 and Figure 4. For T^{DEV}, the non stationarity (LB test and ACF residuals) is clearly explained with the trace of the standardised residuals: there is no evident pattern, thus it resembles white noise. In case of T^{HOST}, an execution pattern appears, more evident with VOR-32 T^{HOST}. The pattern is not that strong since ACF residuals and Ljung-Box outline stationarity until leg 5, VOR-32 T^{HOST}. Hence, it is not a strong stationarity, but stationarity is present anyway. With T^{HOST} we can see that there is no randomicity anymore, except for VOR-32 T^{HOST}. Moreover, execution peaks with a certain periodicity appear. Conversely, in case of T^{DEV} the appearances of peaks do not exhibit any periodic trend.

Seeking the worst case by investigating different execution conditions, we can see how the VOR-32, unsurprisingly, represents the worst-case among the ones considered (VOR-1, VOR-8, VOR-28, and VOR-32), being the case with larger observations. In Figure 6 we have represented the measurement-based distributions as Cumulative Distribution Functions (CDFs). In there we can also see that there is no measurable difference between VOR-28 and VOR-32 at both T^{DEV} and T^{HOST} cases. In future work we will eval-

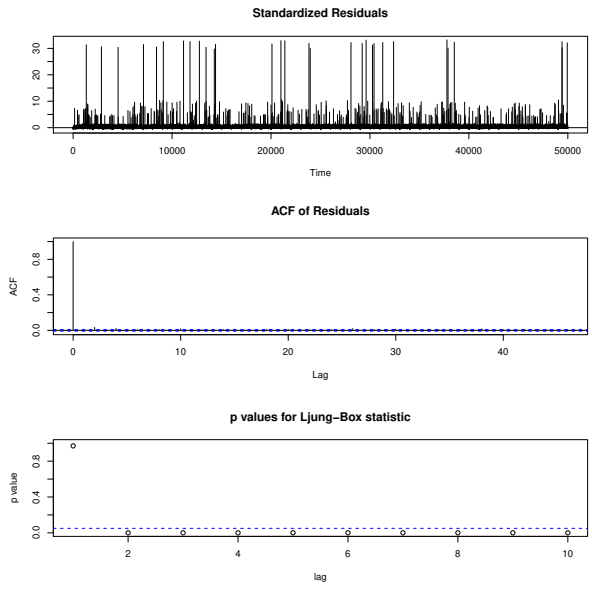


(a) VOR-1 T^{DEV}

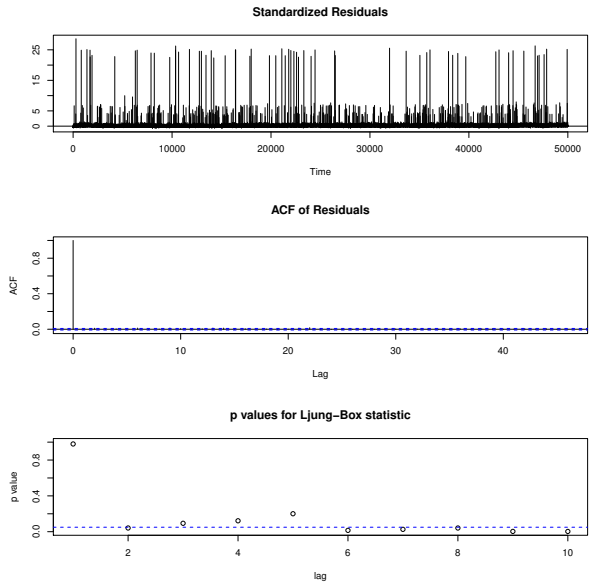


(b) VOR-32 T^{DEV}

Figure 3: Statistics from the autocorrelation function (ACF) and the Ljung-Box statistics. VOR-1 and VOR-32 T^{DEV} compared.



(a) VOR-1 T^{HOST}



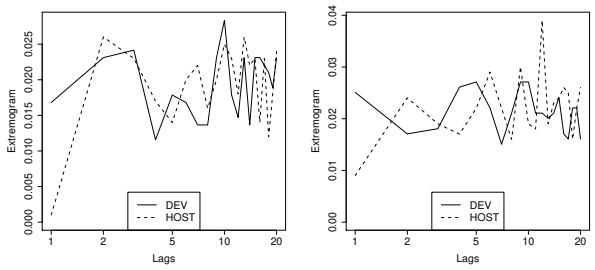
(b) VOR-32 T^{HOST}

Figure 4: Statistics from the autocorrelation function (ACF) and the Ljung-Box statistics. VOR-1 and VOR-32 T^{HOST} compared.

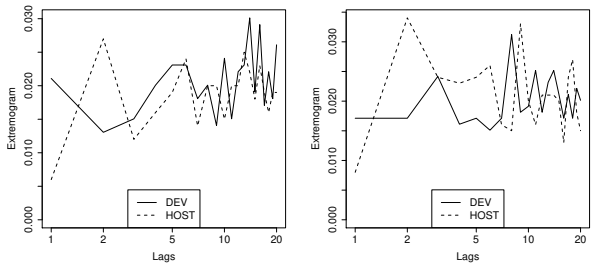
uate more execution conditions and the applicability of the statistical analysis to prove worst-cases. For example, by also considering different shared memory/L1 configurations (16/48 or 48/16 KB) rather than just different input sizes.

5.2 From the Measurements to the pWCET

Finally we apply the EVT, in particular the block maxima version of the EVT [8]. In this paper we do not give any detail about the complexity of the block maxima EVT due to parameter decision (notably the block size), and we consider a block size of 25 observations. The application of the EVT is meant to compare the pWCETs of the different execution conditions. Figures 7 and 8 illustrate the differences accuracy in between VOR-x cases. The CDF representation is applied to the EVT pWCET distribution estimations. Although the real pWCET is not known, we can still reason about the accuracy of the pWCET estimations. For VOR-1 T^{DEV} and VOR-8 T^{DEV}, the EVT is closer to the measurements while for VOR-28 T^{DEV} and VOR-32 T^{DEV} it is less close. This is due to the shape of the measurement distributions. Wider distributions (larger execution variability) means that rare events could be far away from the average



(a) VOR-1 (b) VOR-8



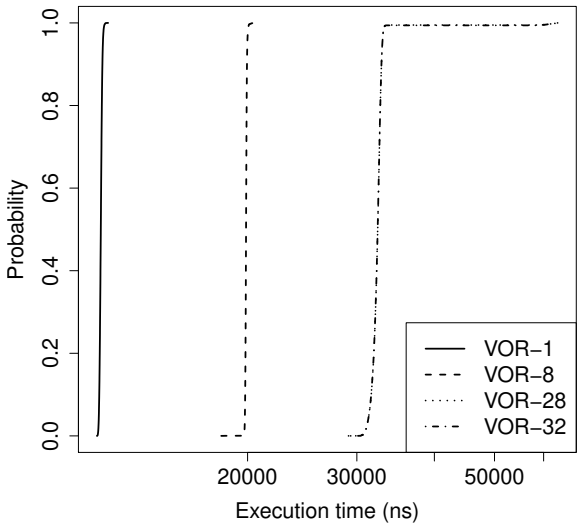
(c) VOR-28 (d) VOR-32

Figure 5: Measurement extremogram up to 20 observations lag. T^{DEV} and T^{HOST} compared.

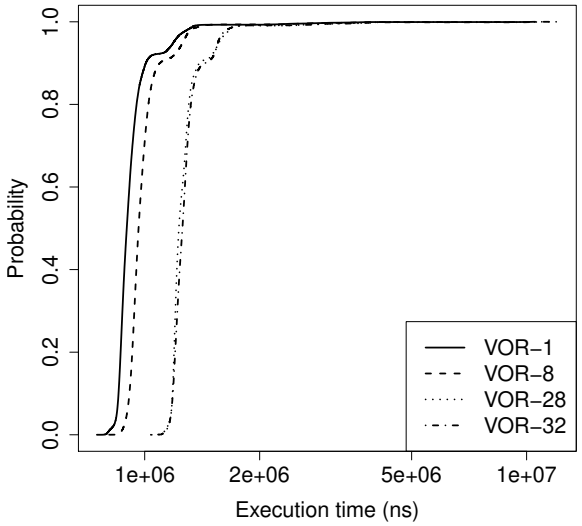
	10^{-6}	10^{-9}	10^{-12}
VOR-1 T^{DEV}	12083	12278	12474
VOR-8 T^{DEV}	20600	20989	21248
VOR-28 T^{DEV}	80061	104613	128051
VOR-32 T^{DEV}	88252	115189	142510
VOR-1 T^{HOST}	6697335	9283349	12127964
VOR-8 T^{HOST}	6561744	9438101	12053516
VOR-28 T^{HOST}	8007463	11350140	14692817
VOR-32 T^{HOST}	8985862	12812711	16345188

Table 2: EVT estimates for T^{DEV} and T^{HOST} at 10^{-6}, 10^{-9}, and 10^{-12} probability thresholds.

behaviour. The EVT has to consider that in order to be safe: possibly much larger values than the measured ones have to be included. For the T^{HOST} cases, the measured distributions are consistently even wider, due to larger peaks on the execution times and two different peaks, visible in the residual representation of Figure 4. This makes the measured distributions resemble bi-variate distributions, Figure 8, and motivates the smaller estimation accuracy from the EVT. Table 2 shows the EVT estimations of the pWCET values at probability 10^{-6}, 10^{-9}, and 10^{-12} for both T^{DEV} and T^{HOST} cases; the probabilistic worst-case execution times are in ns. Those values are exceeding thresholds C, from the 1-CDF representation, and recall that the associated probabilities p are the probabilities of exceeding that threshold, $p(C) = P\{\mathcal{C}^* > C\}$ being $\mathcal{C}^*$ the EVT pWCET distribution estimation. These results illustrate the pWCET variation at different probability thresholds. To explain the large differences of the VOR-28 and VOR-32 T^{DEV} EVT estimations with respect to their measurements, again, we need to con-



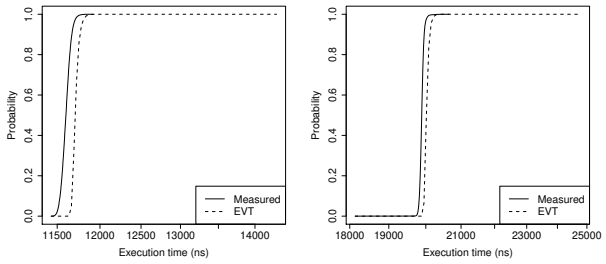
(a) T^{DEV}



(b) T^{HOST}

Figure 6: Measurements for all the VORONOI cases. CDF representation of the distributions.

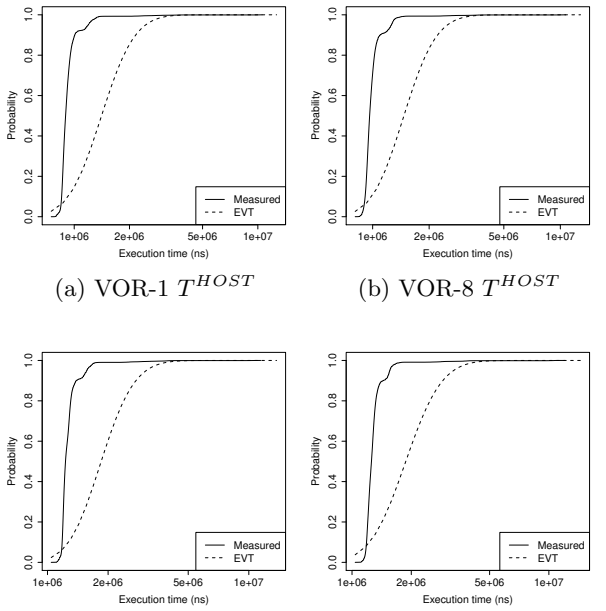
sider the variability of the measurement distributions: in order to be safe, with large variabilities and stationarity, the EVT looses accuracy. With narrow distributions like VOR-1 and VOR-8, the EVT can better model the measurements; the resulting pWCET estimations are closer to the observed execution times. For T^{HOST}, the poorest accuracy is due to the quality of the measured distribution. We also notice that the exceeding values for T^{HOST} for all VOR-x cases, for the same probability threshold, are of similar magnitude at each other. We conclude, empirically, that this is because T^{HOST} is dominated by the one-off costs of the CUDA driver execution and bus transfer launch, rather



(a) VOR-1 T^{DEV} (b) VOR-8 T^{DEV}

(c) VOR-28 T^{DEV} (d) VOR-32 T^{DEV}

Figure 7: EVT applied to VOR-1, VOR-8, VOR-28 and VOR-32 T^{DEV}. Comparison of measurements vs EVT, CDF representations.



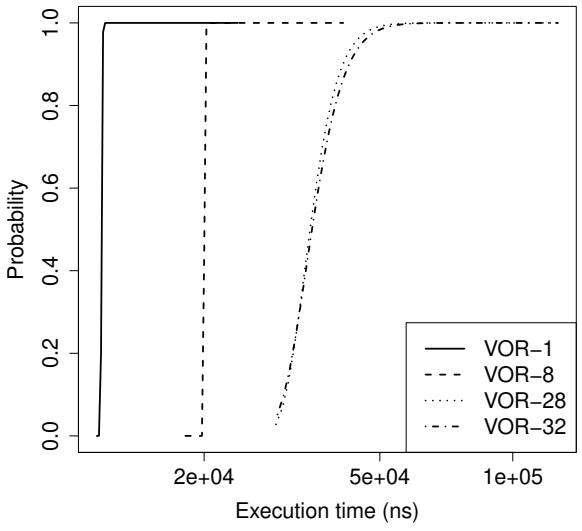
(a) VOR-1 T^{HOST} (b) VOR-8 T^{HOST}

(c) VOR-28 T^{HOST} (d) VOR-32 T^{HOST}

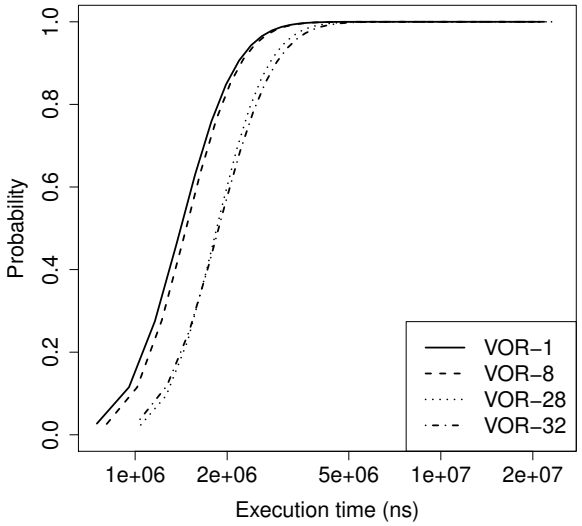
Figure 8: EVT applied to VOR-1, VOR-8, VOR-28 and VOR-32 T^{HOST}. Comparison of measurements vs EVT, CDF representations.

than the size of the problem instance (number of thread blocks). Indeed, $T^{HOST} \gg T^{DEV}$ in our experiments.

Figure 9 is to give informal evidence to EVT pWCET differences. Although the EVT provides the pWCET from a set of measurements $\mathcal{C}$, alone it is not enough to conclude about the task pWCET in any possible execution condition. Since the pWCET estimates for VOR-28 and VOR-32 (the cases with more thread blocks) are not inferable from those of VOR-1 and VOR-8, it is necessary to include the worst-case execution condition (in terms of thread blocks) in order to guarantee safe pWCET estimations $\mathcal{C}^*$. Among the measurements made, VOR-32 is the worst-case for both T^{DEV} and T^{HOST}. The EVT statistical estimation out of the VOR-32 can provide the safety guarantee that real-time analyses require upper-bounding the pWCET estimation for



(a) T^{DEV}



(b) T^{HOST}

Figure 9: CDF EVT distributions for T^{DEV}, T^{HOST}.

all the other measurements.

A few interesting observations on Table 2: (1) the gap between VOR-1 (1 thread block on 1 SM) and VOR-8 (8 thread blocks in parallel, on different SMs) quantifies the effect of contention across SMs for L2 and GPU main memory; (2) unlike the measured values, the EVT for VOR-8 and VOR-32 for a given probability, does not scale linearly with the thread blocks; (3) the almost identical VOR-28 and VOR-32 pWCET estimations are evidence of a balanced thread block assignment to SMs; the in VOR-28 (where some SMs get 3 and some get 4 thread blocks) is determined by those SMs with 4 thread blocks (same as all SMs in VOR-32).

6. CONCLUSIONS

Through this work we demonstrated, for the first time, that it is possible to apply a pWCET analysis approach based on measurements, statistic analysis, and EVT to parallel applications running on GPUs. We have proficiently extended applicability of EVT to less constraining hypotheses than independence. And that provides a way for obtaining accurate WCET estimates, for the desired confidence level, despite the lack of detailed public documentation on the GPU's memory subsystem and its internal scheduling.

Acknowledgements:
Work partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects ref. FCOMP-01-0124-FEDER-037281 (CISTER) and FCOMP-01-0124-FEDER-020447 (REGAIN); by FCT and the EU ARTEMIS JU funding, within project ARTEMIS/0001/2013, JU grant nr. 621429 (EMC2); by FCT and by ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/82069/2011.

7. REFERENCES

[1] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *Proc. IEEE ISPASS*, 2009.

[2] M. Bautin, A. Dwarakinath, and T. Chiueh. Graphics Engine Resource Management. In *Proc. 15th ACM/SPIE MMCN*, 2008.

[3] K. Berezovskyi, K. Bletsas, and B. Andersson. Makespan computation for GPU threads running on a single streaming multiprocessor. In *Proc. 24th ECRTS*, pages 277–286, 2012.

[4] K. Berezovskyi, K. Bletsas, and S. M. Petters. Faster makespan estimation for GPU threads on a single streaming multiprocessor. In *Proc. ETFA*, 2013.

[5] A. Betts and A. F. Donaldson. Estimating the WCET of GPU-accelerated applications using hybrid analysis. In *Proc. 25th ECRTS*, pages 193–202, 2013.

[6] V. Chavez-Demoulin and A. Davison. Modelling time series extremes. *REVSTAT*, 10(1), 2012.

[7] M. R. Chernick. A limit theorem for the maximum of autoregressive processes with uniform marginal distributions. *The Annals of Probability*, 9(1):145–149, 02 1981.

[8] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *Proc. 23nd ECRTS*, 2012.

[9] R. A. Davis and T. Mikosch. The extremogram: A correlogram for extreme events. *Bernoulli Society for Mathematical Statistics and Probability*, 2009.

[10] J. Díaz, D. Garcia, K. Kim, C. Lee, L. Bello, L. J.M., and O. Mirabella. Stochastic analysis of periodic real-time systems. In *23rd RTSS*, pages 289–300, 2002.

[11] G. Elliott, B. Ward, and J. Anderson. GPUSync: Architecture-aware management of GPUs for predictable multi-GPU real-time systems. In *Proc. 34th IEEE RTSS*, pages 33–44, 2013.

[12] P. Embrechts, T. Mikosch, and C. Klüppelberg. *Modelling Extremal Events: For Insurance and Finance*. Springer-Verlag, London, UK, UK, 1997.

[13] E. Gumbel. *Statistics of Extremes*. Columbia University Press, 1958.

[14] J. Hansen, S. Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2009.

[15] T. Hsing. On tail index estimation using dependent data. *The Annals of Statistics*, 1991.

[16] S. Kato. Implementing open-source CUDA runtime. http://www.ertl.jp/~shinpei/papers/pro13.pdf, 2013.

[17] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A responsive GPGPU execution model for runtime engines. In *Proc. 32nd IEEE RTSS*, pages 57–66, 2011.

[18] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa. Timegraph: GPU scheduling for real-time multi-tasking environments. In *Proc. USENIX ATC*, page 17, 2011.

[19] S. Kato, M. McThrow, C. Maltzahn, and S. Brandt. Gdev: First-class GPU resource management in the operating system. In *Proc. USENIX ATC*, 2012.

[20] M. Leadbetter. Extremes and local dependence in stationary sequences. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 65(2), 1983.

[21] M. R. Leadbetter, G. Lindgren, and H. Rootzén. *Extremes and Related Properties of Random Sequences and Processes*. Springer-Verlag, 1983.

[22] R. M. Loynes. Extreme values in uniformly mixing stationary stochastic processes. *The Annals of Mathematical Statistics*, 36(3):993–999, 06 1965.

[23] I. Majdandzic, C. Trefftz, and G. Wolffe. Computation of Voronoi diagrams using a graphics processing unit. In *Proc. IEEE Int. Conf. Electro/Information Technology (EIT)*, pages 437–441, 2008.

[24] R. Mangharam and A. A. Saba. Anytime algorithms for GPU architectures. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS)*, 2011.

[25] P. Northrop. Semiparametric estimation of the extremal index using block maxima. Technical report, Dept of Statistical Science, UCL, 2005.

[26] NVIDIA Corp. NVIDIA's next generation CUDA compute architecture: Kepler GK110. http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf, 2012.

[27] NVIDIA Corp. CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/index.html, 2014.

[28] C. J. Rossbach, J. Currey, M. Silberstein, B. Ray, and E. Witchel. Ptask: Operating system abstractions to manage GPUs as compute devices. In *Proc. 23rd ACM Symp. on Operating Systems Principles*, 2011.