Early Design Phase Cross-Platform Throughput Prediction for Industrial Stream-Processing Applications

Tjerk Bijlsma
ESI
High Tech Campus 25, 5600 HE, Eindhoven, The Netherlands
tjerk.bijlsma@tno.nl

Alexander Lint
Océ Technologies
P.O. Box 101, 5900 MA, Venlo, The Netherlands
alexander.lint@oce.com

Jacques Verriet
ESI
High Tech Campus 25, 5600 HE, Eindhoven, The Netherlands
jacques.verriet@tno.nl

Abstract

Industrial embedded platforms are often used to execute stream-processing applications, from which the results are used by actuators. On average, these stream-processing applications should at least meet the required throughput of their actuators, which poses a real-time requirement on the system. To avoid extra costs and delays, it is desired to estimate during the early design phase if a combination of an embedded platform and a stream-processing application can achieve the required throughput. The throughput of a stream-processing application executed on different embedded platforms can be predicted by modeling them using static or measurement based analysis. However, during the early design phase it can be desirable to have a model that allows a large set of embedded platforms to be considered, where embedded platforms with predictive instructions are supported.

This paper presents a gray-box approach applicable during the early design phase to perform cross-platform throughput predictions for industrial stream-processing applications and their embedded platforms. A three step regression-based approach is presented, which uses an expression based on Amdahl's law for the discrete scaling of workload over cores and a large database with CPU performance scores to perform cross-platform throughput predictions. Validation, with a limited set of platforms, showed the usability of the approach. The pragmatic approach is based on a prototype industrial digital image processing application for a printer from Océ, which is also used to present the approach.

2012 ACM Subject Classification General and reference → Estimation,General and reference → Performance,Computer systems organization → Real-time systems

Keywords and phrases throughput prediction, stream-processing application, early design phase, regression model, cross-platform

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2018.18

Acknowledgements The research is carried out as part of the Octo+ program under the responsibility of Embedded Systems Innovation by TNO (ESI) with Océ Technologies B.V. as the carrying industrial partner. The Octo+ research is supported by the Netherlands Ministry of Economic Affairs.

1 Introduction

Industrial embedded platforms typically execute applications that process streams of information, from which the results are used by an actuator. Variations in processing time can often be absorbed by a small buffer between the stream-processing application and the actuator. However, the stream-processing application should at least achieve the required throughput, such that outputs are available at a fixed rate for the actuator. This throughput puts a real-time requirement on the system, where not meeting the deadline a few times in a row results in an empty buffer towards the actuator. It is necessary to know if an embedded platform in combination with a stream-processing application can achieve its throughput and real-time requirements, as early as possible during the design time of such industrial systems. Not achieving the required throughput either influences the specifications of the industrial system, or causes quality loss for the industrial system. An alternative is to over-dimension the embedded platform, which increases production costs and thereby harms the competitiveness of the industrial system.

A trend is the constant increase of actuation speed and quality for stream-processing applications, thereby requiring more complex processing and a higher throughput. Often architecture or design patterns [4] are applied for stream-processing applications to manage their complexity and achieve scalability. The *Master Slave* pattern is often applied for scalability, as it allows data parallelism to be exploited by using more slave threads, as suggested by Amdahl's law [2]. The *Pipes and Filter* pattern manages complexity by clustering parts of the stream-processing application as filters, thereby allowing function parallelism to be exploited. Stream-processing applications using these patterns are applied throughout industry. Applying these patterns, manages the complexity and enables scalability of the throughput for these stream-processing applications. However, in addition to enabling scalability, knowledge of the achievable throughput for target embedded platforms is desirable. Having the knowledge of achievable performance on different embedded platforms during the early design phase enables Design Space Exploration (DSE), such that a trade-off between achievable throughput and costs can be made.

The throughput of a stream-processing application executed on different embedded platforms can be predicted by modeling their combinations using static analysis or by using measurement based analysis [26]. However, using static analysis for throughput prediction requires worst-case execution times [26], which can be hard to derive for CPUs of embedded platforms that use techniques like caching and predictive instructions. Measurement based analysis requires measurements on considered target CPUs to predict execution times, such that only physically available target CPUs and platforms can be considered. During the early design phase a light-weight analytical model [17, 13] for the performance of the combined application and embedded platform can be desirable, to be able to easily consider a large set of combinations. Preferably gray-box application and platform knowledge is sufficient to create such a model, where measurements on a *reference* platform suffice to predict the throughput for a large set of *target* platforms.

This paper presents a gray-box approach applicable during the early design phase to perform cross-platform throughput predictions for industrial stream-processing applications and their embedded platforms. This pragmatic approach is based on the early design phase of a real prototype Océ digital image processing application for a printer with real-time requirements, which is also used as running example. For stream-processing applications that implement the Master Slave design pattern, a three step regression-based approach is presented. The prediction uses an extended expression based on Amdahl's law that

considers the discrete scaling of workload over slaves, which is to the best of our knowledge a novel contribution. A database with CPU performance scores is used for the cross-platform throughput prediction, enabling predictions for a very large set of CPUs and embedded platforms. Suitability of the approach has been validated, by using measurements for a limited set of platforms. Additionally, DSE can be performed to select a cost-effective embedded platform that delivers the required throughput. For its performance, the digital image processing application requires predictive instructions and caching, such that x86 instruction set based CPUs are considered.

The outline of this paper is as follows. In Section 2 related work is presented, followed by an overview of the throughput-prediction approach in Section 3. Following, the single-core execution time prediction is explained in Section 4. Section 5 presents the throughput prediction for multiple cores in an embedded platform. Next, the cross-platform throughput predictions are presented in Section 6 and validated in Section 7. A design space exploration is presented in Section 8. Finally, conclusions and future-work are discussed in Section 9.

2 Related Work

Typically, a model is used to predict the throughput of a stream-processing application, allowing predictions for different embedded platform configurations and input streams. The model to perform throughput predictions should be chosen depending on the internal application structure and the development phase. The gut feeling of the designer and back-of-the-envelope models are straightforward and suitable during the early design phase, for applications with a clear relation between the execution time and the features of their input stream. A small amount of time is needed to create and use such models, however, their accuracies vary. Detailed static analysis models, like data-flow models [25] or discrete-event simulation [13, 12] can capture complex interactions and perform accurate predictions for an application. However, creating such models requires a fair amount of time and worst-case execution times, which make them less suitable for the early design phase in which the application can change a lot. In [24], detailed modeling of applications in a real-time system and their contention during the early design phase is discussed. This approach focuses on detailed models of applications on a single platform and their contention to ease integration, rather than to predict performance. The approach presented in [11] models an application as a process network with worst-case and best-case execution times for the tasks together with the scheduling policy for the considered platform. This requires detailed knowledge of the underlying platform and does not allow predictions for other platforms without constructing a dedicated model for it. An alternative to detailed behavioral models are predictive models, as used in the SPORE approach proposed by [15], where code is instrumented to measure the impact of features from the input stream on the execution times of parts of the application, resulting in a simple expression to predict execution times. This approach considers predictions for a single platform using an elaborate tool flow for automated instrumentation and feature selection. In [20], a performance modeling approach using probability distributions for the execution times of an application together with a platform is presented. The modeled CPU is restricted to not contain caches or predictive instructions, which are typically present in commercially available CPUs and beneficial for the execution time of stream-processing applications. Furthermore, extensive overviews of software performance-prediction approaches are provided by both [3] and [27], including approaches based on queuing networks, stochastic process algebra, stochastic Petrinets, and stochastic processes. However, the discussed approaches do not address models of the underlying platform or CPUs that support predictive instructions and mostly contain detailed application models that may be time consuming to create.

Our approach performs regression for applications based on the Master Slave pattern [4] executed by a reference embedded platform, followed by a DSE according to the Y-chart approach [11] to identify cost-effective alternatives. The Y-chart approach describes multiple iterations of evaluating a mapping of an application to a target platform, where based on evaluation result the mapping, application, or platform is optimized. To predict the throughput when using multiple cores in a CPU, regression is performed using an extension of Amdahl's law [2] that considers the discrete partitioning and distribution of objects from the input stream. Other approaches predicted throughputs using a support vector machine, a neural network, or machine learning, as presented in [1] and [10]. However, in [1] the focus is on the time used in a production system rather than different embedded platforms and in [10] database queries for a fixed platform are considered. Extensions of Amdahl's law are proposed for multi-core and cloud computing systems [28, 23, 8, 14]. However, these do not consider the discrete scaling of workload over slave processes and cores. An approach applying DSE for heterogeneous system performance is presented in [18], searching mappings of tasks from an application among cores in CPUs, considering latency and energy for which back-of-the-envelope predictive models are used. Where this approach considers mappings for modeled platforms, our approach performs DSE for a large set of CPUs using information obtained from a performance database. Our approach targets the early design phase using gray-box knowledge of applications, such that an expression for throughput prediction is easily obtained and updated during the development of the application.

3 Overview of Throughput-Prediction Approach

This section presents an overview of the piecewise linear expression for the throughput prediction, which is obtained in three steps: a single-core, a multi-core, and a cross-platform prediction step. Also, a description is given of the industrial stream-processing applications that are targeted by this approach.

The proposed throughput-prediction approach predicts the throughput for Master Slave based stream-processing applications, for different input streams and embedded platforms. A piecewise linear expression is employed because of its simplicity and adaptability during the early design phase. The approach uses a piecewise linear expression obtained by the following three steps:

1. Single-core execution time prediction: translate the features of the input stream into a single-core execution time. For the running example, translate the (input) compressed bitmap size to an execution time on a single processing core. To derive this expression, execution times of the test-set processed by the application at a single processing core are used.
2. Multi-core throughput prediction: translate the single-core execution time to a multi-core execution time, from which the throughput can be derived. For the digital image processing application, this involves considering the distribution of data over the slaves. To derive this expression, execution times of the digital image processing application for the test-set while it is executed at one, two, and three processing cores are used.
3. Cross-platform translations: translate the performance to other platforms, using available performance scores. For the digital image processing application the scores from the PassMark [21] performance database are used.

Execution times can be obtained, by using an early version of an application mapped to a reference platform. The preceding three steps will be detailed in sections 4, 5, and 6.

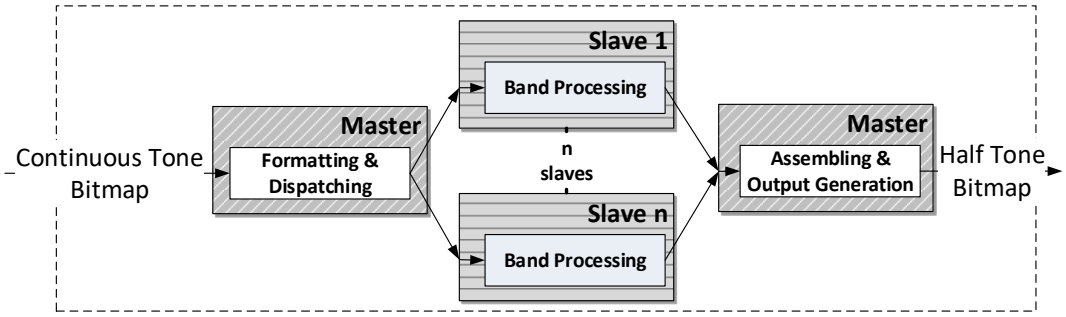


Figure 1 Digital image processing application structure, applying the Master Slave pattern.

The throughput-prediction approach targets industrial stream-processing applications that apply the Master Slave pattern [4]. Additionally, the MapReduce pattern [7] introduced by Google has a structure similar to the Master Slave pattern, such that the predictions may even be applicable for computing clusters. Typically, stream-processing applications have a tight throughput requirement, which means that the output has to be available at a fixed rate for an actuator. A small buffer is often used for the output of the stream-processing application, such that the throughput requirement can be averaged over the number of outputs in this buffer. The input stream consists of a stream of objects that the application should process, where the objects can be sub-divided into parts that can be distributed among the slaves for processing.

We use a prototype of an Océ digital image processing application for a printer as running example. The application has a tight throughput requirement, because converted bitmaps have to be available at a fixed rate to actuate the print head at the moment the paper passes it. A buffer for a few bitmaps can be used towards the print head, because buffering too many bitmaps hampers the ability to correct the print head for detected failures. This puts a firm real-time requirement on the system [5, 25], where it is undesirable that a deadline is missed but the system can continue afterward. Not meeting the deadline a few times in a row results in an empty buffer towards the print head, such that one or possibly multiple pages get lost in the case of duplex printing.

An overview of the digital image processing application is given in Figure 1. The input stream consists of compressed Continuous Tone (CT) bitmaps that are partitioned into 8 bands each for this example, where each band covers a fixed number of lines from a bitmap. The *master* has two function blocks, annotated with a white diagonal pattern. First, it receives a CT input bitmap from which the bands are dispatched to slaves. Next, it assembles the results of the slaves to return a Half Toned (HT) output bitmap. Note that a slave may have to process multiple bands. Internally slaves apply the Pipes and Filter pattern to process the bands, in such a way that typically the available cache memory is sufficient, thereby avoiding interference due to memory access between slaves. At initialization time, the number of slaves, which each will be assigned to a core or hyper-thread of a core, can be configured. For a processor that supports hyper-threading, typically a *physical core* can be seen as two *logical cores*, where the logical cores share the execution resources of the physical core. For the performance required by the digital image processing application, x86 instruction set based CPUs are considered that typically support hyper-threading, predictive instructions, and caching. To benchmark the application, a test-set with 455 bitmaps is used that represents the typical load of the prototype Océ digital image processing application, because the test-set includes simple, typical, and complex bitmaps.

4 Single-Core Execution Time Predictions

A linear expression is used to predict the single-core execution time for stream-processing applications. Typical features of the input stream, like size or resolution, relate to required processing time. The relation between one or more features from an input stream and the single-core execution time is captured using regression.

Gray-box knowledge of the stream-processing application can be used to identify features from the input stream that determine, or have a strong influence on, the execution time. Using a linear expression to relate these features and the execution time results in a simple and easy adaptable model, thereby making it especially suitable during the early design phase when many design decision still have to be made. For a good relation between the identified features and the execution time, an application designed for predictable temporal behavior is desirable, thus it should avoid large and abrupt changes in execution time for minor feature changes.

For the digital image processing application, the compressed size of a bitmap and the size of the bands in this compressed bitmap influence the execution time, as illustrated in Figure 2. The execution times to process bands or compressed bitmaps have been measured on a reference Intel Core i7 platform, where execution times were determined by timestamps at the beginning and end of the processing. Both a) and b) plot the compressed size of the CT bitmap versus the execution time of the slave to process all bands of the 455 bitmaps from the test set using a single core or hyper-thread of a core of the embedded platform. In a) and c) a single core is used, and in b) and d) a single hyper-thread of a core is used while the other hyper-thread is performing computations for another slave process. In Figure 2 c) and d) the compressed size of a band is plotted versus the execution time of the slave to process this band, where all bands of the 455 bitmaps have been processed. Note that Figure 2 c) and d) have significantly more measurements than a) and b), which is because c) and d) plot the relation for each of the 8 bands of the 455 bitmaps.

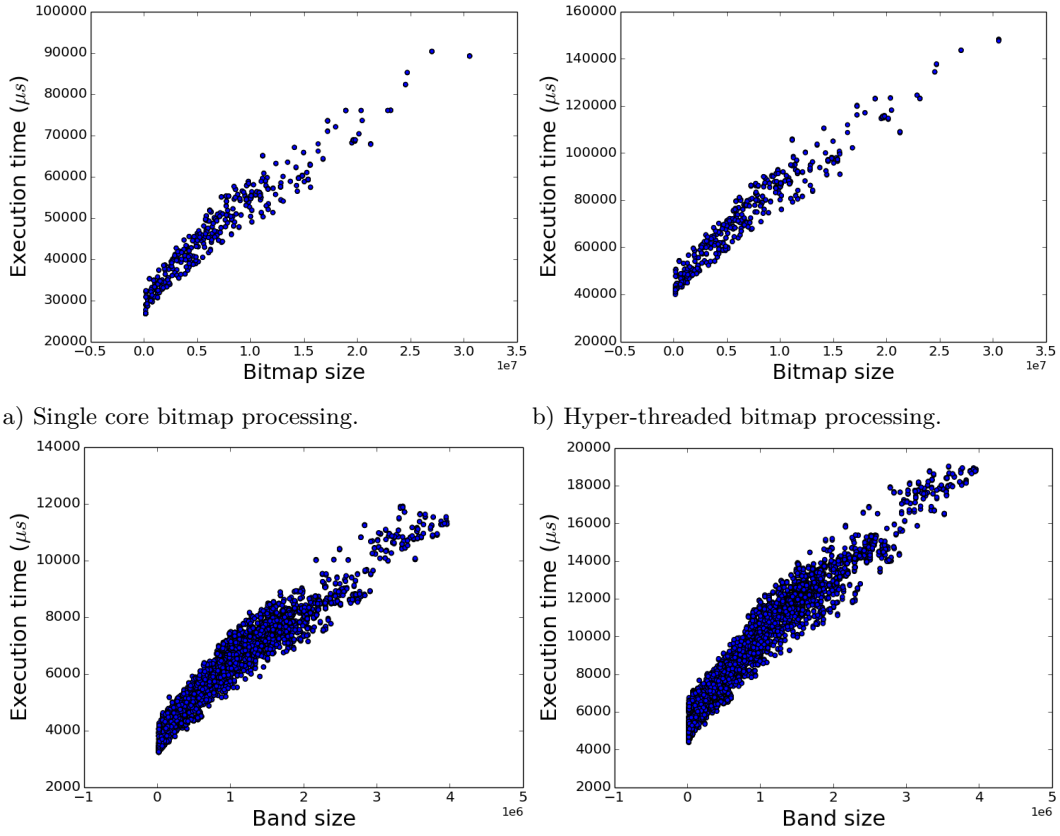
Figure 2 shows a relation between the execution time of the slaves of the digital image processing application and the compressed size of the bitmaps or bands, because the execution time increases when the bitmap or band size increases. The figure suggests a linear relation between bitmap or band size and image processing time, because most points are located around a line. This linear relation can be captured by Equation (1), with a coefficient c_0 that is multiplied with the size of the band or bitmap b to which a constant c_1 is added to obtain the execution time for a single core $T_s(b)$:

$$T_s(b) = c_0 b + c_1 \tag{1}$$

Note that the equation relates one feature, b in this case, to the execution time, but more features can be added if they are present and identified in the input stream.

Linear regression can be applied [19] using a tool box like StatsModels [22] to derive values for c_0 and c_1. Using the so-called ordinary least squares for the errors of the linear regression [19, 6], minimizes the sum of the squares of the differences between the measured execution times and those predicted by $T_s(b)$ via Equation (1). In Table 1, the coefficients c_0 and constants c_1 found by applying regression for Equation (1) for the four considered cases are given.

Additionally, Table 1 provides the R^2 and $P(F)$ values for the quality of the regression. The coefficient of determination, the R^2 value, gives the explained variation divided over the total variation for the regression. For the hyper-threading bitmap size regression, the R^2 value of 0.94 indicates that 6 percent of the variation is unaccounted for. The four cases for



a) Single core bitmap processing.

b) Hyper-threaded bitmap processing.

c) Single core band processing.

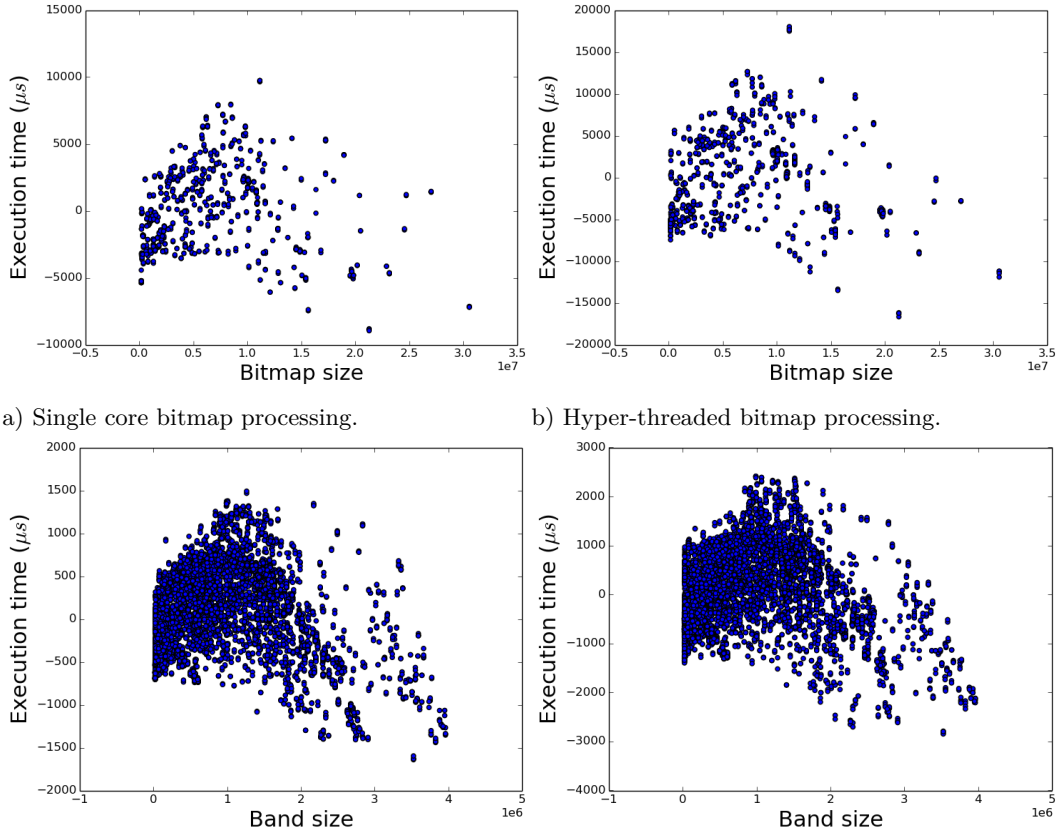
d) Hyper-threaded band processing.

Figure 2 Execution times of digital image processing application (μs) given the compressed bitmap size or compressed band size, for a processor using a single core or a core with a single hyper-thread.

Table 1 Regression results for execution time using a single or hyper-threaded core with bitmap or band size.

	Single core bitmap processing	Hyper-threaded bitmap processing	Single core band processing	Hyper-threaded band processing
c_0	0.0021	0.0037	0.0022	0.0039
c_1	31,730	46,740	3883	5680
R^2	0.93	0.94	0.93	0.94
$P(F)$	0.0	0.0	0.0	0.0

which regression is performed leave only 6 or 7 percent of the variation in the execution time unaccounted for, which is acceptable. The $P(F)$ values indicate whether the regression has significant predictive capability; the $P(F)$ values should be smaller than the significance level of 0.05. For all four regressions the $P(F)$ values are 0.0, indicating that the regression is



a) Single core bitmap processing.

b) Hyper-threaded bitmap processing.

c) Single core band processing.

d) Hyper-threaded band processing.

Figure 3 Residuals for linear relation from Equation (1) using variables from Table 1.

significant. All coefficients c_0 and constants c_1 also had a P-value of 0.0, indicating that it is very unlikely that they deviate significantly from the obtained values. Finally, residual plots of the differences between the measured and predicted execution times are given in Figure 3. These plots show no relation between the band or bitmap size and the difference between the measured and predicted execution times, indicating that it is captured by the regression.

5 Multi-Core Throughput Predictions

A piecewise linear expression for the speedup of the single-core execution time, when using multiple slaves that each process a part of the input stream, is discussed in this section. An expression based on Amdahl's law [2] is used that accounts for the workload of the input stream which is typically partitioned into a number of discrete parts. The multi-core throughput prediction is performed using the single-core execution times based on the compressed bitmap and band size, where using the compressed bitmap size results in the best predictions. The multi-core throughput prediction enables the different mappings of an application and platform to be evaluated, as suggested by the Y-chart approach [11].

An expression that expresses the speedup by using multiple cores or hyper-threaded cores to execute the slaves in the embedded platform, should express the scaling of the single-core execution time. Amdahl [2] gave an expression, where the execution time scales continuously with the number of additional cores that can be used. Amdahl's law assumes that a workload

can be continuously distributed over the number of parallel resources, slaves when the Master Slave pattern is applied, which is given in the following equation:

$$T_a(s,b) = (1-p)T_s(b) + p(\frac{1}{s})T_s(b) \tag{2}$$

where $T_s(b)$ is the execution time when using a single core or hyper-thread for a workload b, p is the fraction of the execution time that benefits from additional parallel resources, and s is the number of parallel cores or hyper-threaded cores. Our observation is that typically an input stream consists of objects that have a discrete partitioning in a number of elements that each require processing. When considering continuous scaling to process a bitmap with 8 bands, going from 4 to 5, 6, or 7 slaves would speedup the throughput. However, due to the discrete partitioning of a bitmap in 8 bands no speedup is realized, because at least one of the slaves has to process two parts and thereby determines the throughput. Therefore, an extension of Amdahl's law is proposed that considers the discrete scaling of the workload. The discrete scaling expression for Amdahl's law $T_d(d,s,b)$ that returns an execution time, considers that a workload b has d discrete parts that are to be distributed over s parallel resources and is expressed as follows:

$$T_d(d,s,b) = (1-p)T_s(b) + p(\frac{\lceil \frac{d}{s} \rceil}{d})T_s(b) \tag{3}$$

where the ceiling $\lceil \frac{d}{s} \rceil$ determines the largest integer number of the discrete parts in the input stream that a slave has to process and this is divided by d to determine the speed up compared to the single-core execution time $T_s(b)$.

By adding coefficients and a constant, regression can be performed using the discrete scaling expression of Amdahl's law to express the execution time dependent on the number of used cores. For the digital image processing application, regression is performed using the following expression:

$$T_d(d,s,b) = T_s(b)c_2 + (\frac{\lceil \frac{d}{s} \rceil}{d})T_s(b)c_3 + c_4 \tag{4}$$

where $T_s(b)$ gives the single-core execution time for one bitmap from the input stream of the digital image processing application which has d discrete parts, which are processed by s slave processes that each run on their own core or hyper-thread of a core. Each part of the expression has a coefficient, c_2 and c_3, and a constant c_4 is added, which are determined by performing regression. Note that compared to Equation (3), Equation (4) does not contain p to represent the sequential fraction. The variable p is left out because regression captures it in the coefficients c_2, c_3, and c_4.

Regression for the digital image processing application is performed using Equation (4), where for the single-core execution time prediction $T_s(b)$ the regression results from Table 1 are used. The single-core execution time prediction for the bitmap size returns the total execution time for all slaves and can directly be used for $T_s(b)$. The single-core execution time prediction for the band size returns the execution time of the slave for the specific band, but the execution time for all bands of a bitmap is required. The average size of the n bands of a bitmap can be used to compute the single-core execution time, by multiplying the execution time for the average band size by n. However, the predicted execution times for the average band sizes are nearly similar to the predicted execution times for the bitmap sizes. Alternatively, the maximum band size can be used by taking the size of the largest band of a bitmap, computing its execution time, and multiply it by n, to get a pessimistic estimation of the execution time. Because execution times predicted for maximum band sizes differ from execution times predicted for bitmaps sizes, below multi-core throughput predictions based on both single-core execution time predictions will be compared.

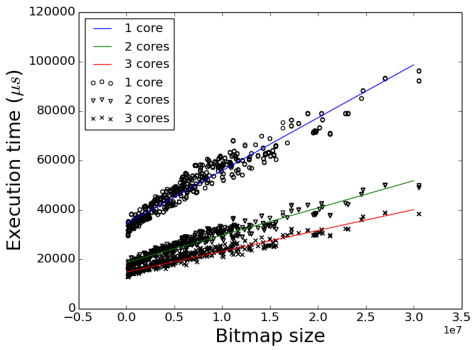
Table 2 Regression results for the execution time at one or more single or hyper-threaded cores, using the bitmap or band size to determine $T_s(b)$.

	Single core bitmap processing	Hyper-threading bitmap processing	Single core band processing	Hyper-threading band processing
c_2	0.0300	0.0151	-0.0369	-0.0146
c_3	0.9903	0.9661	0.8500	0.8131
c_4	1971.4	2795.0	5641.8	5420.5
R^2	0.98	0.95	0.95	0.91
$P(F)$	0.0	0.0	0.0	0.0

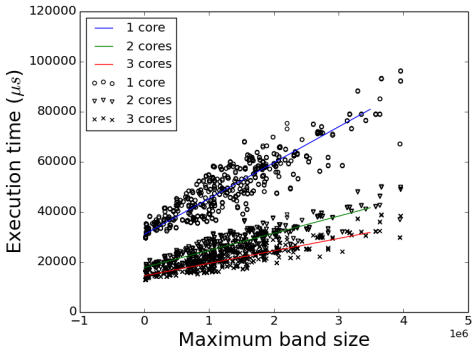
Table 2 gives the regression results for Equation (4) when using the single or hyper-threaded core execution time predictions from Table 1. Regression was performed using measured execution times for the test-set when using 1, 2, or 3 single or hyper-threaded cores in the platform. The R^2 indicates that between 2 and 9 percent of the variation is unaccounted for, which indicates a good result. Note that when using the band size for Equation (4), the obtained value for c_2 is negative. Together, the value of c_2 and the constant c_4 relate to the execution time of the master, where the rather small negative c_2 indicates that with increasing band sizes the execution time for the master decreases slightly. Furthermore, the $P(f)$ value being smaller than 0.05 indicates significant predictive capability of the regression. Also for this regression, each coefficient and constant had a P-value of 0.0, indicating that it is unlikely that the found values deviate significantly from the actual values.

Figure 4 shows measured execution times given the compressed bitmap size or the maximum band size on single or hyper-threaded cores, lines for the relations obtained for Equation (4), and corresponding residual plots. In a), b), e), and f) the execution times when using a single core or a hyper-threaded core are plotted versus the compressed bitmap size or the maximum band size. Additionally, for 1, 2, and 3 cores the line resulting from Equation (4) is plotted, where in e) and f) only two lines are visible because the lines for 4 and 6 hyper-threads overlap. The residual plots in c), d), g), and h) show that there is no remaining relation between the compressed bitmap size or maximum band size and the differences between the predicted and measured execution times. These plots show that the line of the piecewise linear equation relates to the measurements and that there is no remaining relation between the execution time and the compressed bitmap size.

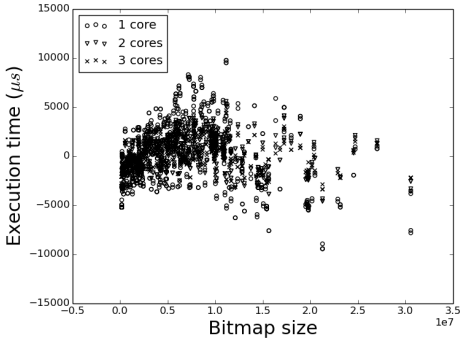
We choose to use the bitmap size to predict the single-core execution time, because the regressions in Table 1 and 2 that use the bitmap size show a slightly better R^2, compared to when the maximum band size is used. When using three cores with hyper-threading for the digital image processing application for a bitmap with a size of 7,255,824, which has 1,316,317 as corresponding maximum band size, an execution time of 20.23 ms is measured, where the bitmap size based expression predicts 21.68 ms and the band size based expression 21.74 ms. The slightly better R^2 indicates that the regression results using the bitmap size cover the variance in the execution times a little bit better compared to the regression using the maximum band size. Additionally, obtaining the size of the bitmap is more convenient compared to identifying the maximum size among the bands in a bitmap.



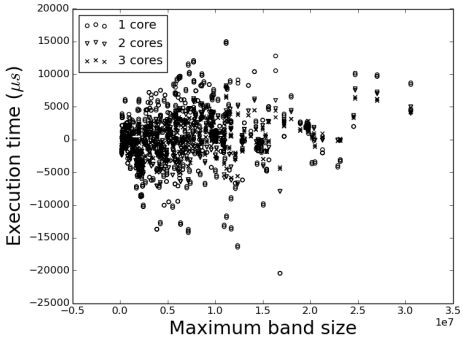
a) Single core bitmap processing.



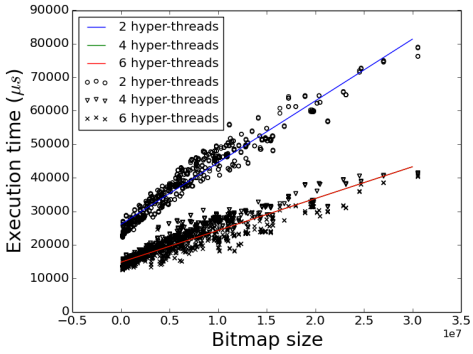
b) Single core band processing.



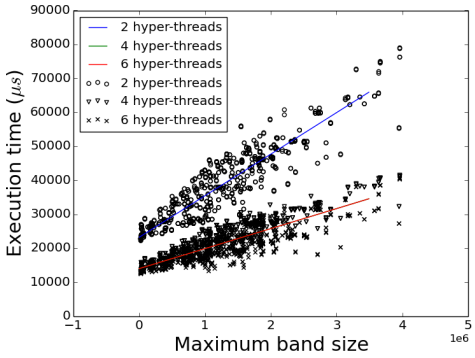
c) Residuals single core bitmap processing.



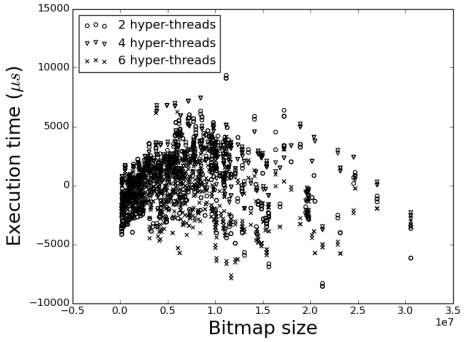
d) Residuals single core band processing.



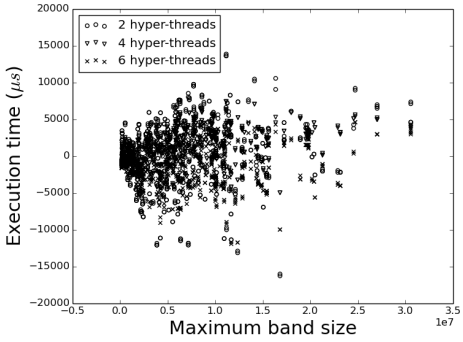
e) Hyper-threaded bitmap processing.



f) Hyper-threaded band processing.



g) Residuals hyper-threaded bitmap processing.



h) Residuals hyper-threaded band processing.

Figure 4 Execution time for the digital image processing application using multiple cores given the compressed bitmap or band size, including lines for Equation (4), and residual plots.

The discrete scaling Amdahl's law expression $T_d(d, s, b)$ returns an execution time in μs, where for the digital image processing application a throughput value is desired. The worst-case or average throughput can be determined by taking the reciprocal of the execution time for a large or average bitmap which is multiplied by $1 \cdot 10^6$, resulting in the Bitmaps Per Second (BPS) that can be processed by the digital image processing application. A buffer of n output bitmaps is used at the output of the digital image processing application to average the variations in execution times. Because the execution time is linearly related to the CT bitmap size, the average size of the n CT bitmaps that can fill the buffer determines the throughput. The average bitmap size of the test-set, being 7,255,824, is used as average size for these n bitmaps that determine the throughput. Note that typically, one would choose a bitmap size slightly larger than the average size that fits in the buffer, to include a margin on the throughput.

6 Cross-Platform Throughput Predictions

Cross-platform throughput prediction can be achieved by combining the multi-core throughput prediction with performance numbers of a reference and a target platform. During the early design phase, such performance numbers allow predicting the potential throughput of an application on considered platforms, before investing the effort of porting the application to the platform.

To translate the throughput from a reference to a target platform, performance numbers of both platforms are required to determine a ratio that represents the relative throughput increase or decrease. It is preferable to base the performance number on a benchmark that performs similar operations, also in similar proportions, as the considered stream-processing application. Accurate performance numbers can be realized by creating a representative benchmark to obtain performance numbers for both platforms. Creating such a benchmark is costly. Furthermore, only physically available platforms can be used for such benchmarks, which can limit the number of platforms that can be considered. An alternative is to use a database with performance numbers, like FutureMark [9] or PassMark [21], which provide performance numbers for server and desktop CPUs and mobile platforms. Note that a drawback of such a database is that the performance numbers may be less accurate in representing the performance of the considered application.

We use the PassMark CPU performance database [21], because it contains thousands of x86 instruction set based platforms, thereby enabling DSE for all these candidate platforms. For a broad range of CPUs, PassMark provides a *Full CPU score* that rates the overall performance of the CPU and a *Single-Threaded CPU score* that rates the performance of a single core or a core with hyper-threading. Users run benchmarks on their CPUs and submit their scores to the PassMark database, such that the score for common CPUs is typically averaged over thousands of benchmark results. The Full CPU score is based on a benchmark with nine tests, where the Single-Threaded CPU score is based on three tests from this benchmark. Both tests use weighting factors for the contribution of test results to the score, where compression, floating point math, and string sorting tests have a high impact. These tests are representative for the type of operations performed by the digital image processing application. Note that the PassMark database for mobile platforms might be interesting, however using these scores requires measured execution times for the digital image processing application at a mobile platform from this database, which we currently do not have.

The Full and Single-Threaded CPU scores can be used for the relative performance of platforms, such that a factor can be derived to translate the throughput to a target platform. Using the available multi-core throughput regression results from a number of platforms available in the PassMark database, we derive factors using the PassMark scores. Note that with multi-core regression results for a sufficiently large number of CPUs in the database, regression could be applied for an even better relation between the Passmark scores and the multi-core throughput regression results.

The factor to perform a cross-platform translation for the single-core execution time (C_s), between a reference and target platform is given by the following expression:

$$C_s(p_s^r, p_s^t) = \frac{p_f^r}{s_t^r} \Big/ \frac{p_f^t}{s_t^t} \tag{5}$$

using $p_s^r = (s_t^r, p_f^r)$ and $p_s^t = (s_t^t, p_f^t)$ to keep the parameter list concise, where p_f^r and p_f^t represent the Full CPU scores and s_t^r and s_t^t the number of cores, for the reference and the target platform, respectively. Note that this equation uses the Full CPU scores rather than the Single-Threaded CPU scores, since attempts using the Full CPU scores resulted in a more accurate prediction, probably because the tests for the PassMark Full CPU score have a better match with the digital image processing application.

The factor (C_m) to perform a cross-platform translation for the throughput, considering the number of used cores, between a reference and target platform is given by the following expression:

$$C_m(p_m^r, p_m^t) = \frac{\left(\frac{p_f^r}{s_t^r}\right)}{p_s^r} \Big/ \frac{\left(\frac{p_f^t}{s_t^t}\right)}{p_s^t} \tag{6}$$

using $p_m^r = (s_t^r, p_f^r, p_s^r)$ and $p_m^t = (s_t^t, p_f^t, p_s^t)$ to keep the parameter list concise, where p_f^r and p_f^t represent the Full CPU scores, p_s^r and p_s^t the Single-Threaded CPU scores, and s_t^r and s_t^t the number of cores in the reference and target platform, respectively. For both the reference and target platform, this equation translates the Full CPU score to a score for a single core and divides it over the Single-Threaded CPU score, which results in a factor that indicates how much more multi-core performance is obtained compared to s_t times the Single-Threaded performance.

The factor C_s to translate the execution time from a reference to a target platform is included in the single-core execution time expressions from Equation (1) as follows:

$$T_{sc}(b, p_s^r, p_s^t) = (c_0 b + c_1)\, C_s(p_s^r, p_s^t) \tag{7}$$

where p_f^r and p_f^t represent the Full CPU scores and s_t^r and s_t^t the number of cores present, of the reference and target platform, respectively.

The factor C_m to consider the scaling of the multi-core performance and T_{sc} for the cross-platform single-core execution time is included in the multi-core throughput prediction from Equation (4) as follows:

$$T_{dc}(d, s, b, p_s^r, p_s^t, p_m^r, p_m^t) = T_{sc}(b)c_2 + (\frac{\lceil \frac{d}{s} \rceil}{d}) T_{sc}(b) c_3 C_m(p_m^r, p_m^t) + c_4 C_m(p_m^r, p_m^t) \tag{8}$$

where T_s is replaced by T_{sc} and both c_3 and c_4, which relate to the amount of used parallelism, are multiplied by C_m.

Table 3 CPU information of benchmark platforms.

CPU	Launch	Cores(s_t)	Speed (GHz)	p_f	p_s
Reference Core i7	2015	4/8		10,040	2,159
Core i7 4770	Q2'13	4/8	3.4	9,797	2,226
Core i7 860	Q3'09	4/8	2.8	5,060	1,226
Xeon E5 2650	Q1'12	8/16	2.0	10,262	1,310
Atom x5 Z8500	Q1'15	4/4	1.44	1,697	503
Atom C2758	Q3'13	8/8	2.4	3,162	512

7 Throughput Prediction Validation

The cross-platform throughput prediction is validated by comparing its predictions based on a reference platform with multi-core predictions for a limited set of platforms, by using execution times measured on these platforms. Differences between 13% and -8% are found, which is acceptable for a light-weight execution time prediction that guides decisions during the early design phase.

The CPUs of the platforms used for the validation are shown in Table 3, where the first row shows the reference platform with an Intel Core i7 from the Skylake generation. The names of the CPUs are given in the first column and the launch dates in the second column. These names and launch dates indicate that high-end and low-end CPUs with varying introduction years will be used for the validation. In the column *cores*, the number of cores and hyper-threads is given. The Full CPU score (p_f) and Single-Threaded score (p_s) from the PassMark database, sampled in May 2017, are given in the last two columns. The platforms contain a varying amount of memory with different speeds, however the slaves of the digital image processing application can typically prefetch the data in the cache, such that the memory has limited influence on the execution time.

To validate the accuracy of the cross-platform prediction, execution times have been measured for the digital image processing application for the platforms from Table 3 and regression has been performed for these measurements. For each of these platforms, for the different number of cores or hyper-threads that could be used, the execution times for the 455 bitmaps from the test-set were measured. Regression was performed for the measurements to obtain compact models from which results can be compared. Table 4, provides the coefficients c_0, c_1, c_2, c_3, and c_4 for the platforms from Table 3. The i7 and Xeon CPUs have two rows with coefficients, one for using them with single-cores (sc) and one for using their cores with hyper-threading (ht), as indicated in the second column. The last column gives the R^2 value of the regression for the multi-core expression from Equation (4), where the values above 0.92 indicate that good relations have been found.

Table 5 provides the relative differences between measured and predicted execution times for the platforms, for the case where four cores or four hyper-threads are used. Differences between results from the expressions are compared, rather than comparing the slopes and constants for the expression. Comparing slopes and constants between the expression for the multi-core and cross-platform throughput predictions showed to be impractical, because the performed regressions for the five coefficients has multiple solutions. The differences shown in Table 5 are given for three bitmap sizes, the average bitmap size in the test set, and a small and a large bitmap size. The large and small bitmap size are determined by subtracting and adding the standard deviation among the bitmap sizes in the test set, respectively. Considering the differences given in Table 5, for the average size bitmap the multi-core and

Table 4 Regression results for the digital image processing application executed at the platforms from Table 3.

		c_0	c_1	c_2	c_3	c_4	R^2
Reference	sc	0.0021	31,730	0.0300	0.9903	1971	0.98
Core i7	ht	0.0037	46,740	0.0151	0.9661	2795	0.95
Core	sc	0.0024	35,860	0.0147	1.0063	1919	0.97
i7 4770	ht	0.0040	56,160	0.0026	0.9995	3058	0.97
Core	sc	0.0047	60,380	0.1059	0.9170	2140	0.96
i7 860	ht	0.0075	105,400	0.0020	0.9972	4285	0.96
Xeon	sc	0.0041	66,310	0.0141	1.0075	3365	0.97
E5 2650	ht	0.0062	95,580	0.0419	0.9239	4542	0.97
Atom x5 Z8500	sc	0.0105	137,400	0.0035	1.0192	6085	0.97
Atom C2758	sc	0.0126	157,800	-0.0603	1.0868	5425	0.92

Table 5 Difference between measurement and prediction for the digital image processing application, using 4 cores.

	Bitmap	small	average	large
	size	1,450,148	7,255,824	13,061,500
Intel	sc	0.01	0.00	0.00
i7 4770	ht	-0.07	-0.05	-0.04
Intel	sc	-0.05	-0.08	-0.10
i7 860	ht	-0.02	0.01	0.04
Intel	sc	0.08	0.10	0.11
Xeon E5 2650	ht	0.03	0.09	0.13
Atom x5 z8500	sc	-0.06	-0.07	-0.07
Atom C2758	sc	0.10	0.08	0.06

cross-platform predictions differ between 10% and -8% and for all bitmaps between 13% and -8%. Similar numbers are obtained using a different number of cores and cores with hyper-threads. These differences do not seem to relate to platform generations nor cores with or without hyper-threads. Given that it is a light-weight model to guide decisions during the early design phase, we find this error acceptable.

8 Design Space Exploration using Throughput Predictions

Cross-platform throughput predictions enable exploration of suitable and cost-effective embedded platforms. First, cost-effective embedded platforms are compared, followed by an exploration of effective combinations of platforms.

Often costs and performance are Key Performance Indicators (KPI) for stream-processing applications. However, cost is a KPI that can be refined in platform purchase costs, development costs, and life-cycle costs. Note that quantifying development and life-cycle costs is difficult and that they are likely larger than the platform purchase cost. By quantifying the throughput of a stream-processing application for a large list of embedded platforms, the most cost-effective platform can be selected. However, among products, the development costs can be reduced by selecting a cost-effective low, medium, and high-performance embedded platform for which development and updates will be performed. For the life-cycle, it would

be even nicer to be able to combine a number of these low, medium, and high-performance embedded platforms to be able to scale the throughput by adding an embedded platform. Note that this requires an adapted stream-processing application that distributes scaled bands to the slaves at the different platforms and balances the load, as described by the MapReduce pattern for computing cluster.

Combining the cross-platform throughput prediction with an extensive and detailed list of Intel CPUs [16], a motherboard list, a list with memory modules, and a PassMark score list, the design space can be explored for cost-effective embedded platforms and embedded platform combinations. Intel provides extensive lists of their CPUs with details like sockets, supported memory types, and the CPU name, which allows linking the CPU information with compatible motherboards, memory modules, and PassMark scores, respectively. In the performed exploration 44 motherboards were considered and 20 different DDR memory modules, with varying technologies and sizes, for which the information and costs were obtained via supplier information. The list of Intel contained 2504 CPUs from which 987 were selected as relevant and the PassMark list contained 2253 entries with 1424 relevant entries. Combining the Intel and PassMark list resulted in a list with detailed CPU information, with amongst others the CPU costs, for 754 CPUs. The combination of this list with detailed CPU information and the list with motherboards and memory modules, resulted in 1838 different combinations that each represent an embedded platform to be considered.

In the list with embedded platforms, for each platform the throughput was added for the digital image processing application, the costs of the platform, and cost per one BPS. For each embedded platform, the throughput in BPS was calculated, using Equation (7) and (8) and the PassMark score. To calculate the throughput, a large bitmap of size 13,061,500 is considered, similar to the large bitmap used in Table 5. Additionally, the number of cores or cores with hyper-threads is decreased by one or two, respectively, to allow room for the master and other processing.

Table 6 provides a low, medium and high-performance platform, obtained from the list of embedded platforms that may be cost-effective alternatives to the reference platforms to reduce development costs. To compare the costs, throughput in BPS, and costs per BPS of these platforms, values relative to the Intel i7 Skylake reference platform are provided. The first column indicates the performance category of the platform and the second column gives the name of the CPU. The relative increase or decrease in cost, predicted throughput, and costs per BPS, are given in the third, fourth, and fifth column, respectively. The final two columns give the PassMark scores, p_f and p_s, of the CPU. The table illustrates that the high-performance platform with an Intel i7 5820k CPU reduces the cost per BPS by a fraction of 0.375, where the low-performance platform costs 1.44 times as much, compared to the reference platform. Still, the low or medium-performance platforms are interesting as cost-effective solutions for products in which the predicted throughput would be sufficient, because their platform costs are a fraction compared to the high-performance platform costs. Note that for industrial systems, the platform selection also considers the period for which CPUs and components will be long-term available for production, which we did not include here.

Figure 5 plots the relative costs to the reference platform, when using one or multiple embedded platforms to realize a required throughput, suggesting that considering multiple embedded platforms results in smoother increasing platform costs. Note that size and power are not considered in this comparison, but that they are an important element in the trade-off for the platform selection. Using the list with embedded platforms, for a throughput of 220 until 280 BPS, the most cost-effective solutions are searched to realize the required

Table 6 CPUs for high, medium, and low-performance embedded platforms.

Performance category	CPU name	Relative costs	Relative BPS	Relative costs/BPS	p_f	p_s
High	i7 5820k	1.29	3.35	0.375	12,994	2,016
Medium	i7 6700te	1.00	1.10	0.875	10,514	2,113
Low	i5 6402p	0.80	0.55	1.44	7,750	2,081

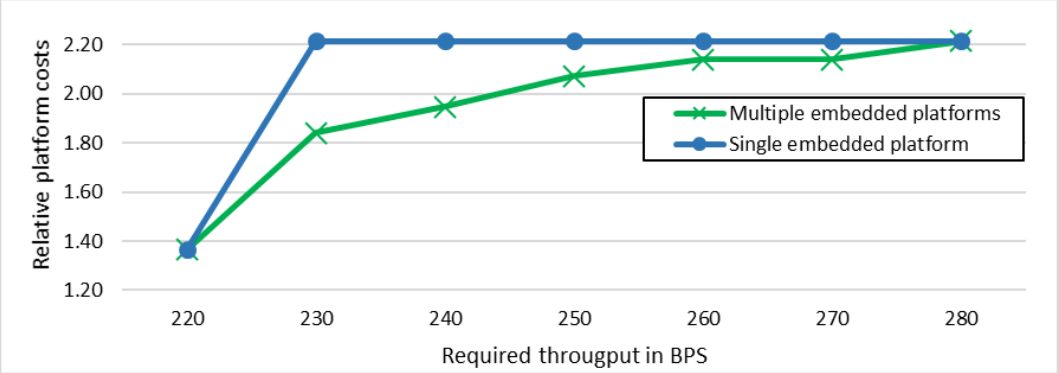


Figure 5 Cost comparison for platform combinations and single platforms to realize a desired BPS.

throughput, either by only using one embedded platform or by allowing multiple embedded platforms to be used. Multiple platforms can be used that each process a part of a bitmap, where partitioning the bitmap requires no additional effort and the effort for combining the parts is negligible. In case multiple embedded platforms are considered, an exhaustive search is performed that considers all platform combinations to realize the required throughput. The speed of communicating bitmaps plays a role, if we compare solutions with one or multiple embedded platforms. Therefore, network interfaces, with their speed and costs, were added to the platforms and the search accounts for the achievable BPS via the network interface. It is sufficient to consider the number of BPS that can be communicated via a network interface, because communicating a next bitmap can overlap with processing the current. Performing the searches requires less than 30 seconds on a single core, making it a scalable approach. An interesting result is that for a throughput of 230 until 270 BPS, the usage of multiple embedded platforms is more cost-effective and also results in a smoother increasing platform cost.

Table 7 provides details for the selected platforms, which are plotted in Figure 5, with costs relative to the costs of the Intel Core i7 reference platform. In the case of multiple embedded platforms, cost-effective Core i7 and i5 CPUs can be combined until a throughput of 270 BPS. In contrast, for a throughput of 230 BPS, a single embedded platform with two powerful Xeon CPUs at one motherboard is selected. For a throughput between 230 and 270 BPS the platform cost for multiple embedded platforms is up to 17% lower compared to a single platform. The cost difference is because cost-effective Core i7 and i5 CPUs can be combined, rather than using two the same Xeon CPUs on a motherboard.

Table 7 Comparison between multiple and single embedded platforms selected to realize a desired number of BPS.

	Multiple embedded platforms			Single embedded platform		
BPS	CPUs	BPS	Costs	CPU	BPS	Costs
220	Core i7 6800K	220	1.37	Core i7 6800K	220	1.37
230	Core i7 5820K Atom x7 Z8700 Atom x5 Z8550	230	1.84	Xeon E5 2620 Xeon E5 2620	349	2.22
240	Core i7 6800K Core i3 4170	241	1.95	Xeon E5 2620 Xeon E5 2620	349	2.22
250	Core i7 6800K Core i5 4590	251	2.07	Xeon E5 2620 Xeon E5 2620	349	2.22
260	Core i7 5820K Xeon E3 1231 v3	272	2.14	Xeon E5 2620 Xeon E5 2620	349	2.22
270	Core i7 5820K Xeon E3 1231 v3	272	2.14	Xeon E5 2620 Xeon E5 2620	349	2.22
280	Xeon E5 2620 Xeon E5 2620	349	2.22	Xeon E5 2620 Xeon E5 2620	349	2.22

9 Conclusion

A throughput-prediction approach for stream-processing applications and their embedded platforms has been presented in this paper. A real prototype industrial Océ digital image processing application for a printer, for which this approach was applied, has been used to demonstrate the approach. For this application, a design space exploration was performed, where the piecewise linear expression for the throughput prediction made it possible to consider combinations of more than 1800 different embedded platforms with the digital image processing application to realize desired throughputs.

The throughput prediction targets stream-processing applications that apply the Master Slave or possibly the MapReduce pattern, during the early design phase. A piecewise linear expression is related to features in the input stream, such that only gray-box knowledge of the application is necessary and updating the expression is easy. First, the execution time when using a single core is related to one or more features from the input stream. The second step scales the single-core execution time for the multiple cores or cores with hyper-threading that are used in the embedded platform. This step uses an expression for the discrete scaling of workload over slaves, which is a novel extension of Amdahl's law. The third step uses performance scores from a performance database to translate the performance from a reference platform to target platforms. To demonstrate the applicability of the approach, it was applied to an Océ digital image processing application for a printer. The obtained piecewise linear expression allowed throughput predictions during the early design phase and exploring a large set of embedded platforms. Validation of the cross-platform throughput prediction, using the digital image processing application and a limited set of platforms, showed an acceptable error and thereby its usability. An interesting extension of this throughput-prediction approach would be the inclusion of key configuration parameters of the application, like the output bitmap resolution, to enable predictions for next generations of the system.

References

1 Abdulrahman Alenezi, Scott A. Moses, and Theodore B. Trafalis. Real-time prediction of order flowtimes using support vector regression. *Elsevier Computers and Operations Research*, 35(11):3489–3503, 2008.

2 Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proc. Spring Joint Computer Conference*, pages 483–485. ACM, apr 1967.

3 Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

4 Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996.

5 Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer Publishing Company, Incorporated, 3rd edition, 2011.

6 Abraham Charnes, Edward L. Frome, and Po-Lung Yu. The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. *journal of the American Statistical Association*, 71(353):169–171, 1976.

7 Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commununications of the ACM*, 51(1):107–113, 2008.

8 Fernando Diaz-del Rio, Javier Salmeron-Garcia, and Jose Luis Sevillano. Extending amdahl's law for the cloud computing era. *IEEE Computer*, 49(2):14–22, feb 2016.

9 Futuremark Corporation© 2016. Best CPUs - July 2017, 2017. https://www.futuremark.com/hardware/cpu/.

10 Chetan Gupta, Abhay Mehta, and Umeshwar Dayal. PQR: Predicting query execution times for autonomous workload management. In *Proc. Int. Conf. on Autonomic Computing (ICAC)*, ICAC '08, pages 13–22, Washington, DC, USA, 2008. IEEE Computer Society.

11 Wolfgang Haid, Matthias Keller, Kai Huang, Iuliana Bacivarov, and Lothar Thiele. Generation and calibration of compositional performance analysis models for multi-processor systems. In *Proc. Int. symposium on Systems, Architectures, Modeling, and Simulation (SAMOS)*, pages 92–99, jul 2009.

12 Martijn Hendriks, Twan Basten, Jacques Verriet, Marco Brassé, and Lou Somers. A blueprint for system-level performance modeling of software-intensive embedded systems. *Springer Software Tools for Technology Transfer*, 18(1):21–40, feb 2016.

13 Martijn Hendriks, Jacques Verriet, Twan Basten, Marco Brassé, Reinier Dankers, René Laan, Alexander Lint, Hristina Moneva, Lou Somers, and Marc Willekens. Performance engineering for industrial embedded data-processing systems. In *Proc. Int. Conf. Product-Focused Software Process Improvement (PROFES)*, pages 399–414, Cham, 2015. Springer International Publishing.

14 Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, 41(7), 2008.

15 Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Proc. Int. Conf. on Neural Information Processing Systems*, NIPS'10, pages 883–891, USA, 2010. Curran Associates Inc.

16 Intel Corporation©. Product specifications, 2016. https://ark.intel.com/.

17 Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees Vissers. A methodology to design programmable embedded systems. In *Proc. Embedded Processor Design Challenges*, pages 18–37. Springer, 2002.

18 Sumit Mohanty, Viktor K. Prasanna, Sandeep K. Neema, and James R. Davis. Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. *ACM SIGPLAN Notices*, 37(7):18–27, 2002.

19 Douglas C. Montgomery, Elizabeth A. Peck, and Geoffrey G. Vining. *Introduction to Linear Regression Analysis*. Wiley & Sons, New York, third edition, 2006.

20 Ayoub Nouri, Marius Bozga, Anca Molnos, Axel Legay, and Saddek Bensalem. Astrolabe: A rigorous approach for system-level performance modeling and analysis. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(2):31:1–31:26, 2016.

21 PassMark© Software. CPU benchmarks, may 2017. https://www.cpubenchmark.net/.

22 Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. In *Proc. 9th Python in Science Conference*, 2010.

23 Xian He Sun and Yong Chen. Reevaluating amdahl's law in the multicore era. *Academic Press Parallel Distributed Computing*, 70(2):183–188, 2010.

24 David Trilla, Javier Jalle, Mikel Fernandez, Jaume Abella, and Francisco J. Cazorla. Improving early design stage timing modeling in multicore based real-time systems. In *Proc. IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, 2016.

25 Maarten H. Wiggers. *Aperiodic Multiprocessor Scheduling for Real-Time Stream Processing Applications*. PhD thesis, University of Twente, 2009.

26 Reinhard Wilhelm et al. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36:1–36:53, 2008.

27 Murray Woodside, Greg Franks, and Dorina C. Petriu. The future of software performance engineering. In *Proc. Future of Software Engineering (FOSE)*, pages 171–187. IEEE, 2007.

28 Leonid Yavits, Amir Morad, and Ran Ginosar. The effect of communication and synchronization on Amdahl's law in multicore systems. *Elsevier Parallel Computing*, 40(1):1–16, jan 2014.