Hessisches Kompetenzzentrum
für Hochleistungsrechnen

Generation of Training Data for Hardware-Friendly Hash-Width Estimation

Project Manager
Yannick Lavan

Principal Investigator
Prof. Dr.-Ing. Andreas Koch

Project Term
2023 - 2023

Clusters
Lichtenberg II Cluster Darmstadt

Institute
Eingebettete Systeme und ihre Anwendungen

University
Technische Universität Darmstadt

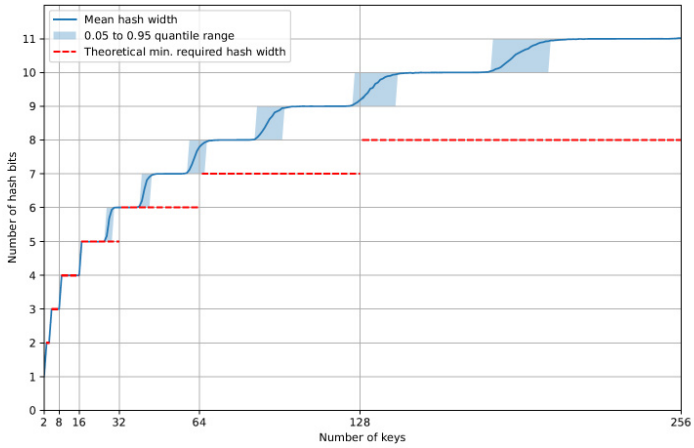


Figure 1: Distribution of the number of keys in a key set to the average number of bits required for the hash values. We can see that with the selected hash function, we can reach the theoretical minimum for small key sets which are typical in IoT. The number of required bits grows faster than the theoretical minimum which will lead to larger memory requirements on the actual hardware.

Introduction

Hash functions play a vital role in modern computing. Their applications range from database queries, over storage of intermediate computation results to cybersecurity. There is a plethora of hash function implementations available for general purpose computing, with ongoing research on making special cases of hash functions, e.g., perfect hashing or minimal perfect hashing, more time- and space-efficient. However, on embedded systems, the use of (perfect) hashing functions is severely limited by the available computational resources and sometimes by real-time requirements in the specific application. Thus, we aim to provide software-programmable perfect hash-functions for statically known key sets. As the result of this work is a piece of hardware, we need to limit the maximum available hash-width at design time. In order to find a good compromise between practicability and hardware cost (i.e., chip area), we need to estimate the width of the perfect hash function for given key sets. An analytical analysis of the key set results in a combinatorial explosion. Thus, we aim to leverage machine learning models for linear-time estimations. For this purpose, we apply resources from high-performance computing to cover a flexible portion of the search space and generate the dataset required for machine learning.

Methods

JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN
TECHNISCHE UNIVERSITÄT DARMSTADT
UNIKASSEL VERSITÄT
Philipps Universität Marburg

As the analytical estimation of required hash widths per key set is intractable, we directly run the search algorithm, leveraging discrete optimization methods. In order to speed up convergence, we parallelize the search over a single key set using thread-based parallelism. Due to the possibility that a search for a single key set still takes a significant amount of time, we parallelize the entire procedure over different key sets, allowing almost linear scaling of throughput (generated key sets per hour) w.r.t. the number of parallel key sets.

Results

During the project, we generated data for over 240000 different key sets of varying lengths in the magnitude which we expect in our targeted Internet-of-Things applications. Figure 1 shows the distribution of generated key sets to their average required hash width with 90-percent confidence intervals shaded. We further highlight the in theory minimal required amount of bits to enumerate a key set of corresponding size in red. We observe that for the selected hash function, larger key sets require significantly more bits than the theoretical minimum, resulting in additional memory overhead. By observing the confidence intervals, we see that there are transmission regions where the number of required hash bits not only depends on the number of keys, but also on the actual keys themselves.

Discussion

This small project has demonstrated that large-scale parallelism can assist in the creation of specialized synthetic datasets for domains typically not related to high-performance computing. We now have to further evaluate the generated data during future research and integrate the results into our toolchain. While the target of this application was mainly the Internet-of-Things, we may also consider generalizing the approach for general-purpose computing.

Last Update: 2023-08-03 17:20