DominoHash - a fast hash function for bioinformatic applications suitable for custom hardware acceleration

Arash Bayat Aleksandar Ignjatovic
Bruno Gaeta Sri Parameswaran

University of New South Wales, Australia
{a.bayat, bgaeta}@unsw.edu.au, {ignjat, sridevan}@cse.unsw.edu.au

Technical Report
UNSW-CSE-TR-201703
Feb 2017

UNSW
SYDNEY

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

The hash-table is a widely used data structure in bioinformatics, for database searching as well as DNA-read mapping. Due to the increasing growth in the size of sequenced data, hardware acceleration has been used to speed up related algorithms. We have developed an alternative hash function to MurmurHash, a hash function commonly used in bioinformatic applications. The main advantage of the proposed hash function (DominoHash) is its suitability for acceleration by custom design hardware. Software and hardware implementations as well as the dataset used in our evaluation are available at `sites.google.com/site/dmhashf`.

Supplementary data section is attached.

1 Introduction

When aligning a query sequence such as a DNA read, a protein or a gene with a huge database sequence such as a genome or a library of proteins or genes, scanning the entire database is not a practical solution; thus the database is preprocessed and indexed into a hash-table. The index is then used to search the database for regions of the database which are similar to a query sequence. Finally, an optimal alignment algorithm is used to align the query to each of the identified regions of similarity.

In order to index the database in a hash-table, fixed-length subsequences along with their position are extracted from the database. Each subsequence along with its position in the database forms a (key, pos) pair to be stored in the hash-table index. The hash-table is able to return the paired pos for a given key. At the time of the search, the position in the database of subsequences of the query are identified using the hash-table index in order to identify regions of the database similar to query. As an example, in [1], all overlapping subsequences of length 21 bases (42-bit keys) are taken from the human reference genome (about 3.2 billion keys) and indexed by hash-tables. Considering key as an index to the table, a table of length $2^{42} \simeq 4.4 \times 10^{12}$ entries is required to store only about 3.2 billion keys. Such a sparse enormous table is neither efficient nor manageable in computer memory.

In a hash-table, the hash function is responsible for mapping keys to the slots of a smaller table which is large enough for storing all keys. Collisions occur in a hash-table for two reasons: first, a key is paired with multiple poss (unavoidable collision); second, the hash function maps two or more keys to the same slot of the table (ideally this should not happen). A collision resolution function is responsible for handling collisions. Since that collision resolution process comes with computational costs, the hash function is expected to distribute keys over the table as evenly as possible to reduce the chance of collisions. The present research is focused on this hash function.

2 Design and Implementation

MurmurHash [2] is a non-cryptographic hash function used by DNA read mappers such as WHAM [3] and SNAP [1]. In this paper, we present DominoHash as an alternative to MurmurHash. DominoHash is inspired by domino shows where each domino tile falls down on and overlaps other tiles. Assume a table is divided into sixteen blocks with four bits indices (from 0000 to 1111). Figure 2.1 illustrates the linear arrangement of blocks as well as considering every four consecutive blocks as a domino tile (a vertical rectangle of distinct colour). Let A and B be the two most significant and two least significant bits of the block index respectively, and C be the two least significant bits of the sum $A + B$. In the block index, by replacing A with C, blocks of Figure 2.1 are rearranged in a form which is shown in Figure 2.2. This operation is called domino-step and results in blocks being scrambled throughout the table. Similar to domino tiles fall down and overlaps each other.

Considering a bit representation of the key with L_k bits, to apply the domino-step, it is possible to take any two bit-fields, A and B of length L_a and L_b respectively where $L_a < L_k$ and $L_b < L_k$, from any part of the key and

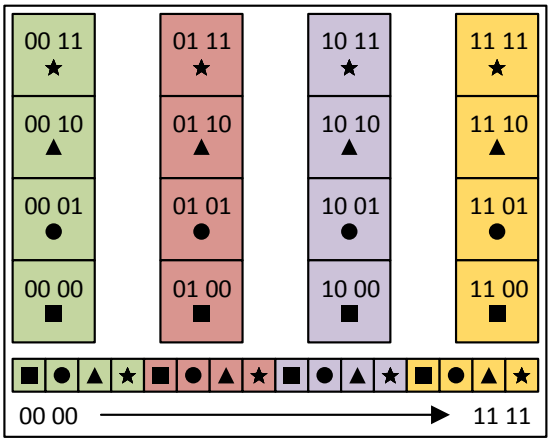

Figure 2.1: Normal arrangement of blocks

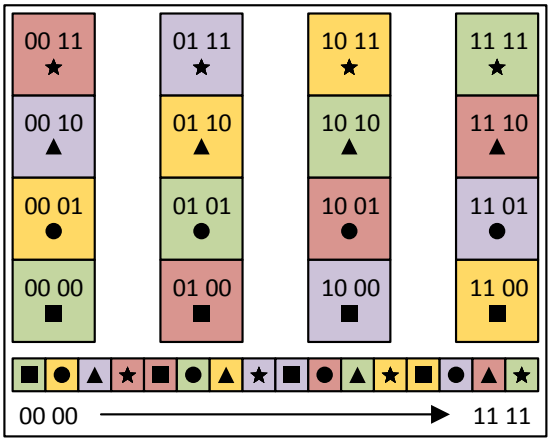

Figure 2.2: Arrangement of blocks after applying DominoHash

replace A with the L_a least significant bits of the sum $A + B$. Domino-step can be repeated on different bit-fields several times, as needed to uniformize the distribution of the data in the table. Bit-fields should be chosen carefully to obtain a sufficiently uniform distribution with a minimal number of steps. Such choice can be obtained using heuristics sketched in *Supplementary Data*.

For evaluation purpose, all possible overlapping subsequences of length 21 bases (42-bit *key*s) were extracted from the human genome, approximately 3.2 billion subsequences. A hash-table of size 2^{32} was used to ensure there was enough space to accommodate all the *key*s. In order to estimate the distribution capability of each hash function, for each C (number of collisions), we counted the number of *key*s (K) that are mapped to a table slot with C collisions. Figure 2.3 represents K as a function of C for three different hash functions: Baseline, MurmurHash and DominoHash. Details of these hash functions are elaborated in *Supplementary Data*.

As seen in Figure 2.3, compared to the Baseline, both MurmurHash and DominoHash were successful in decreasing K for high C values and increasing

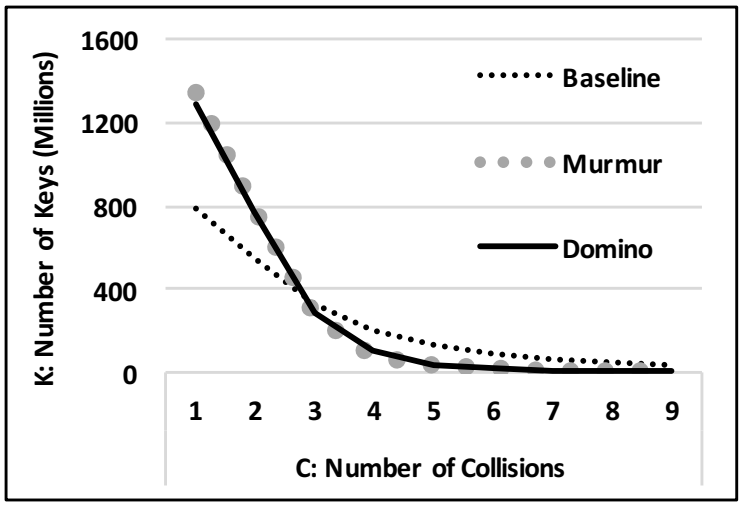

Figure 2.3: Distribution of *key*s based on the number of collisions of the slot of the table that *key*s are mapped to them.

K for lower C values, which means there were fewer slots with a high number of collisions and more slots with a small number of collisions or no collision ($C = 1$). Both DominoHash and MurmurHash resulted in almost identical distributions. However, the main feature of DominoHash is its suitability for implementation using a custom design digital circuit which would makes DominoHash extremely efficient.

In contrast to MurmurHash which uses large multiplications, DominoHash is a series of small additions. While multiplication is considered to be a complex operation for the hardware, addition results in minimum delay, area and power consumption when implemented in a circuit. Although the DominoHash function can be properly fitted in a custom design hardware, its performance cannot exceed MurmurHash's on a standard CPU for two reasons. First, modern CPUs take advantage of pipelined multiplier as well as speculative out-of-order instruction execution which compensate for the long multiplication delay of MurmurHash. Secondly, DominoHash requires multiple bit-field extraction and addition operations, each of which should be executed in a separate standard instruction. However, in a custom design hardware, required bit-fields can be hard-wired into custom sized adders.

We have implemented both MurmurHash and DominoHash in hardware using the Verilog language and synthesised them using Synopsis for 65-nanometer

Table 2.1: Syntesis Report for hardware implementations of MurmurHash and DominoHash

	Murmur	Domino	Improvement
Delay (ns)	6.2	2.2	2.8
Area (nm^2)	11,820.2	854.6	13.8
Power Consumption (mW)	26.5	10.6	2.5

chip fabrication technology. These hardware implementations were verified to be identical to the software implementation of the hash functions. Diagrams of these implementations and additional details can be found in *Supplementary Data*. As shown in Table 2.1, DominoHash was 2.8 times faster and 13.8 times smaller than MurmurHash yet consumed 2.5 times less power using this hardware setup.

3 Conclusion

The decline in the cost of DNA sequencing is resulting in high demand for processing platforms that can handle enormous amounts of sequence data. With the commodification of DNA sequencing, specialised computing hardware is likely to gain considerable advantage over general processing platforms for DNA sequence analysis. DominoHash is one step forward towards the development of such a customised hardware platform for DNA sequence processing.

References

[1] Matei Zaharia and et al. Bolosky. "Faster and More Accurate Sequence Alignment with SNAP". In: *arXiv* (Nov. 2011).

[2] Austin Appleby. *Murmurhash 3 (smhasher)*. Nov. 2010.

[3] Yinan Li, Jignesh M. Patel, and Allison Terrell. "WHAM". In: *ACM Transactions on Database Systems* 37.4 (Dec. 2012), pp. 1–39.

4 Supplementary Data

4.1 Software Implementation

Baseline hash function

For the Baseline hash function elaborated in Algorithm 1 only the right most 32 bits of *key* are taken. Baseline hash function is implemented as a single instruction in software and has no hardware implementation. Note that in all algorithms $X[i:j]$ represent bit i to bit j of X.

Input: KEY 42-bit key
Output: EI 32-bit entry index

$EI \leftarrow KEY[31:00]$;

Algorithm 1: Baseline hash functions

MurmurHash function

The core operation of MurmurHash is elaborated in Algorithm 2. This program consists of three shift-XOR and two multiplication instructions.

Input: KEY 42-bit key
Output: EI 32-bit entry index
Data: D 64-bit variable

$D \leftarrow KEY \oplus (KEY \gg 33)$;
$D \leftarrow D \times FF51AFD7ED558CCD(hex)$;
$D \leftarrow D \oplus (KEY \gg 33)$;
$D \leftarrow D \times C4CEB9FE1A85EC53(hex)$;
$D \leftarrow D \oplus (KEY \gg 33)$;
$EI \leftarrow D[31:00]$;

Algorithm 2: Core operation of MurmurHash function

DominoHash function

Domino-steps of the DominoHash function which is used in our experiment are elaborated in 3. First, in the *key* bit-vector, the most significant 32 bits is added to the least significant 32 bits to get all bits involved in the least significant 32 bits of the *key*. Note that this 32 bit addition is not a domino steps. Then the right most 32 bits are divided into four 8-bit sections (A, B, C and D). Several addition operations (domino-steps) are executed. Finally, all four 8-bit values are merged back to form the output.

considering $Q[A][B][C][D]$ as a four dimensional array, adding D to A ($A \leftarrow A + D$) is similar to scramble forth dimension in the first dimension. In the DominoHash function of Algorithm 3, we scramble the second and the third and the fourth dimension of array in the first dimension. Then we scramble the third and the fourth dimension in the second dimension, and then the fourth dimension in the third dimension. Finally we scramble the first dimension in the forth dimension. Such way we make sure that all dimension are scrambled well in other dimension; thus a sufficiently uniform distribution is obtained.

Input: KEY 42-bit key
Output: EI 32-bit entry index
Data: D 64-bit variable
Data: A, B, C, D, X Array of five 8-bit variables

$KEY \leftarrow KEY + (KEY >> 10)$;

$D \leftarrow KEY[07:00]$;
$C \leftarrow KEY[15:08]$;
$B \leftarrow KEY[23:16]$;
$A \leftarrow KEY[31:24]$;
$X \leftarrow KEY[31:24]$;

$A \leftarrow D + C + B + A$;
$B \leftarrow D + C + B$;
$C \leftarrow D + C$;
$D \leftarrow D + X$;

$EI[07:00] \leftarrow D$;
$EI[15:08] \leftarrow C$;
$EI[23:16] \leftarrow B$;
$EI[31:24] \leftarrow A$;

Algorithm 3: Core operation of differing hash functions

4.2 Hardware Implementation

The Baseline hash function can be implemented by wires in hardware with no cost. Figure 4.1 and 4.2 illustrates the block diagram of implemented hardware for MurmurHash and DominoHash respectively.

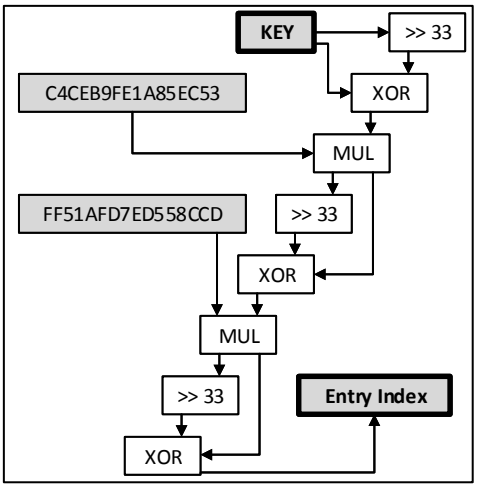

Figure 4.1: Mummer Hardware Implementation

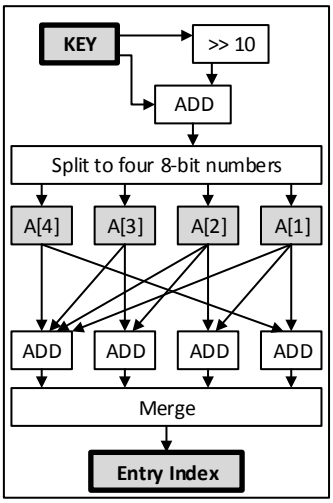

Figure 4.2: Mummer Hardware Implementation