Exploiting SIMD instructions in current processors to improve classical string algorithms*

Susana Ladra[1], Óscar Pedreira[1], Jose Duato[2], and Nieves R. Brisaboa[1]

[1] Database Laboratory. Universidade da Coruña, Spain.
{sladra|opedreira|brisaboa}@udc.es
[2] Parallel Architectures Group, Universitat Politècnica de València, Spain.
jduato@disca.upv.es

1 Introduction

Algorithms and data structures for efficient representation and processing of large databases can be combined with advances in computer architecture, as hardware-aware implementations that exploit particular hardware features. For example, many algorithms have been adapted to exploit the architecture of GPUs, FPGAs, or general-purpose CPUs providing instructions included for particular application domains.

In this paper we explore how the Intel SSE4.2 (*Streaming SIMD Extensions*) SIMD (*Single Instruction Multiple Data*) instructions included in Intel/AMD processors can improve the performance of algorithms for text indexing and searching. The SSE4.2 instruction subset provides instructions for text processing. Our implementations are mainly based on the followings: `POPCOUNT` counts the number of 1 bits in a word of up to 64 bits, `PCMPESTRI` compares two strings of length up to 16 bytes and returns the result as a binary mask.

Despite the benefits these features can bring to text processing, they have been rarely used or evaluated in the existing literature. We present case studies and experimental results that show how much text/string algorithms can benefit from the SIMD extensions. Particularly, we focus on the *rank* and *select* operations in sequences of bits and bytes, and the Horspool string search algorithm.

2 Experimental Evaluation

Rank and Select Given a sequence of bits $B_{1,n}$, $rank_b(B, i)$ is the number of times the bit b appears in B up to position i, and $select_b(B, j)$ returns the position of the j^{th} appearance of bit b in B. Several solutions for efficient *rank* and *select* have been proposed. `GGMN` [3] uses a two-level directory structure storing partial counters, which allows solving *rank* in constant time. When computing $rank_b(B, i)$, these directories give us the number of bits b up to a position j, close to i. The number of bits b in $B[j, i]$ is not counted sequentially, but obtained

* This is a summary of the work published in Procs. of ADBIS 2012, LNCS 7503, 2012, pp. 254–267. Springer (2012).

using precomputed *popcount* tables. We use $B[j, i]$ as an index in that table, and that position gives us the number of 1s in $B[j, i]$.

We modified the implementation of *rank* and *select* by replacing the *popcount* tables with the `POPCOUNT` instruction. We evaluated this implementation in three scenarios: (*i*) computing *rank* in all positions of a randomly generated bitmap of length 1M, (*ii*) implementation of the *RPGraph* [2] graph compression method, and (*iii*) implementation of suffix arrays [4]. Our experimental results showed that the running time can be reduced up to 37%.

Rank and *select* can be extended to sequences of bytes. The directory strategy used in binary sequences can be adapted to this case. We modified the implementation of *rank* and *select* in byte sequences by counting the number of bytes in a block of the array using the `PCMPESTRM` and `POPCOUNT` instructions. `PCMPESTRM` generates a binary mask with a 1 in each position containing the byte b, and `POPCOUNT` counts the number of 1s in this mask. We evaluated this implementation of *rank* and *select* by using these instructions in the implementation of a WTBC (*Wavelet Tree on ByteCodes*), a self-index data structure [1], and ran experiments on a collection containing 8.76GiB of Wikipedia articles. The results showed that we obtain a significant speedup in the SSE4.2 implementation.

String Search with Horspool We also evaluated how string search can improve when applying SSE4.2 instructions to the implementation of the Horspool algorithm. This algorithm traverses a string S looking for a given pattern P. The algorithm uses a search window that is moved along S from left to right. At each step, the substring under the search window is compared with P. Depending on the result, the algorithm skips some positions of S, so it is not fully scanned. We changed the implementation by carrying out the comparison of the search window with P using the `PCMPESTRI` instruction. In the experiments, we searched for 200 patterns in a collection of 1GB of text. The results show that the search time is approximately an 87% of that required by the original implementation.

3 Conclusions

The case studies and experimental evaluations we have presented in this paper reveal how algorithms for text/string processing can benefit from hardware-aware implementations that exploit the SSE4.2 SIMD instructions included in recent general-purpose processors.

References

1. N. R. Brisaboa, A. Fariña, S. Ladra, and G. Navarro. Reorganizing compressed text. In *Procs. SIGIR*, pages 139–146, 2008.
2. F. Claude and G. Navarro. Fast and compact web graph representations. *ACM TWEB*, 4:16:1–16:31, September 2010.
3. R. González, Sz. Grabowski, V. Mäkinen, and G. Navarro. Practical implementation of rank and select queries. In *Procs. WEA*, pages 27–38, 2005.
4. N. Välimäki, V. Mäkinen, W. Gerlach, and K. Dixit. Engineering a compressed suffix tree implementation. *ACM JEA*, 14:2:4.2–2:4.23, 2010.

A Quality references

A.1 Complete reference of the publication

Ladra S., Pedreira O., Duato J., Brisaboa N. R. Exploiting SIMD instructions in current processors to improve classical string algorithms. Proceedings of the 16th Conference on Advances in Databases and Information Systems (ADBIS 2012), Springer. Lecture Notes in Computer Science, Volume 7503, 2012, pp. 254–267.

DOI: 10.1007/978-3-642-33074-2_19

Print ISBN: 978-3-642-33073-5
Online ISBN: 978-3-642-33074-2
Series ISBN: 0302-9743

Available at http://link.springer.com/chapter/10.1007%2F978-3-642-33074-2_19

A.2 About the conference

The 16th East-European Conference on Advances in Databases and Information Systems (ADBIS 2012) was held in Poznan, Poland. The main objective of the ADBIS conferences is to provide a forum for the dissemination of research achievements as well as to promote interaction and collaboration between the database and information systems research communities of the Central and East European countries and the rest of the world. The conferences are initiated and supervised by an international Steering Committee consisting of representatives from Armenia, Austria, Bulgaria, Czech Republic, Greece, Estonia, Finland, Germany, Hungary, Israel, Latvia, Lithuania, Poland, Russia, Serbia, Slovakia, Slovenia, Ukraine, and Italy.

Original papers dealing with both theory and/or applications of database technology and information systems are solicited in the call of papers. The areas of interest include, but are not limited to the following: theoretical foundations of databases and information systems, data models and languages, advanced databases, energy-efficient data management, indexing and search, data streams, text databases and information retrieval, advanced database applications, modern applications of information systems (e.g., social networks), etc.

A.3 Reviewing process and Acceptance Ratio

The ADBIS 2012 conference attracted 122 paper submissions from Algeria, Argentina, Belgium, Bosnia and Herzegovina, Brazil, Colombia, Czech Republic, Egypt, Estonia, Finland, France, FYR Macedonia, Germany, Greece, Hungary, India, Iran, Italy, Japan, Latvia, Poland, Romania, Russia, Slovakia, Spain, Sweden, Sultanate of Oman, The Netherlands, Tunisia, UK, and USA. Using a rigorous reviewing process, the international Program Committee consisting of 74 members from 31 countries selected these 32 contributions for publication (26%).

A.4 Conference rankings

Computing Research and Education (CORE) Since 2006, following an initiative of ANU and NICTA, CORE has been engaged in an exercise to rank fully refereed conferences in which its members publish. In January 2008, a list of such conferences was published. They were ranked A*, A, B, C.

In late 2008, this was overtaken by an Australian Government exercise, the ERA (Excellence in Research for Australia). The Australian Research Council (ARC) began a ranking exercise of research outputs from Australian researchers from 2003-2008. The full list of conferences ranked by the ARC as of February 2010 can be downloaded from their website (http://core.edu.au), where the conferences were also allocated a rank of A, B, or C. ADBIS is ranked as **CORE B**.

CS Conference Rankings The Conference ranking data[3] is based on "Estimated Impact of Conference (EIC)". Paper citations, quality of referees' reports or indexing are used to determine EIC. According to the last update of their ranking (February 2, 2009), **ADBIS conference is 38th** among the top 88 conferences listed (636 were considered) of Topic I (Databases / Knowledge and Data Management / Data Security / Web / Mining), with a **normalized EIC value of 0.79** (range 0.00-1.00).

CiteSeerx CiteSeerx[4] list of venue impact factors is estimated based on Garfield's traditional impact factor. There is not an up- dated ranking list for year 2012, but ADBIS'07 was 302nd out of 581 venues on the current 2007 list.

[3] A copy of the list is available at http://perso.crans.org/ genest/conf.html
[4] http://citeseerx.ist.psu.edu/stats/venues