Journal of Computer Science

Original Research Paper

Evaluating the Efficiency of CPUs, GPUs and FPGAs on a Near-Duplicate Document Detection Via OpenCL

Ercan Canhasi

Gjirafa, Inc. Rr. Rexhep Mala, 28A, Prishtine, Kosove

Article history
Received: 08-06-2017
Revised: 06-03-2018
Accepted: 27-03-2018

Email: ercan.canhasi@uni-prizren.com

Abstract: Discovering identical or near-identical items is urgently important in many applications such as Web crawling since it drastically reduces the text processing costs. Simhash is a widely used technique, able to attribute a bit-string identity to a text, such that similar texts have similar identities. In this study, a real-time solution for a simhash calculation in OpenCL is presented. We also show how it can be utilized by multi-CPUs, GPUs and FPGAs. As a result we indicate that the bottom line computation realized on the FPGA through OpenCL provides significant power advantages.

Keywords: Simhash, OpenCL, CPU, GPU, FPGA, Xilinx, SDAccel

Introduction

Many applications can largely benefit from an effective duplicate or near-duplicate detection algorithm. In Web crawling, which is the most important activity of Gjirafa.com, the first Albanian search engine, near-duplicate web page identification helps to reduce spending valuable resources on parsing and indexing. Similarly, the vertical search services such as news and ads search engines are big benefiters since near-duplicate detection algorithms allow clustering together results with near-duplicate content, which in return increases user-friendliness and avoid information overwhelming. Many high level text processing methods such as document summarization (Canhasi and Kononenko, 2014; 2016) can also benefit from near duplicate sentence identification. One of the most researched method for near-duplicate detection (Canhasi, 2016; Xiao *et al*., 2011; Sood and Loguinov, 2011; Chalamalasetti *et al*., 2012) is the Simhash (Charikar, 2002). The fundamental concept behind the simhash is that each document is depicted by a short integer, fingerprint, which rehash its content.

Method consists of two main steps: (1) The simhash calculation step in which the fingerprints of each document from collection are calculated; (2) the matching step in which the near-duplicate objects are found by comparing their simhash identities. Previous work (Henzinger, 2006; Sood and Loguinov, 2011) has researched the possibilities of optimizing the second stage, i.e., the matching stage, in order to prevent a quadratic complexity of simhash identity similarity calculation. However, previous work show that the simhash identity calculation phase dictates the global execution time (Luo *et al*., 2013). Hence, in this study we suggest a method to treat the first phase of simhash inspired near-duplicate discovery, by using OpenCL in combination with CPUs, GPUs and FPGAs to rapidly process huge numbers of documents and calculate their simhash identities.

Recently, the alternative technologies with higher performance per watt has been seriously examined in the data centers which are known for their high power and cooling requirements. Some of the most promising alternative technologies include multi-core CPUs, Graphics Processing Units (GPUs) and FPGAs. The main shortcoming of utilizing these technologies is their demand for eminently parallel applications in order to fully accomplish their advantage. Lately, as a reaction to this, languages such as OpenCL (Khronos OpenCL Working Group, 2008), CUDA (NVIDIA CUDA) and OpenMP (Chapman *et al*., 2007) have emerge primarily to simplify the complication of developing the parallel applications. Commonly, FPGAs weren't directly targeted by those languages. FPGAs coding requires profound expertise of hardware description language and the fundamental architecture of the targeted equipment. However, a newly announced tool, Xilinx's (Wirbel, 2014) SDAccel the Software-Defined Development Environment for Acceleration, targets OpenCL at FPGA architectures in order to make programming extremely painless. In OpenCL as a platform-independent software development model data parallelism is clearly stated.

OpenCL is based on well-known 'C' programming language and have extensions for easy definition of data parallelism and memory hierarchy. As it is shown in Fig. 1, the typical OpenCL application has two elements, namely the host program and the kernel. The serial part of the application which is engaged in guiding data and the general flow of the algorithm is known as the host program. While the kernel part of the system is the hugely parallel fragment of the code to be sped up on a GPU or an FPGA. Given that the same OpenCL code can be readily used on various platforms, we saw big benefit in using it for performance comparisons. Even that the specific adjustments are still needed in order to fit to each platform for optimal performance, the evaluation method itself is much easier. Consequently, in this study, we present an architectural and programming model study using OpenCL. In it, we implemented simhash, a near duplicate detection algorithm in OpenCL. This code is ported to CPUs, GPUs and FPGAs for comparison.

Near Duplicate Detection-Simhash

The near-duplicate document detection usually includes two steps: (a) The simhash fingerprint calculation; and (b) the matching phase for identifying pairs of near-duplicate documents.

Simhash Calculation Phase

In summary, it is a hashing approach which maps a text document represented by terms to an identity bit-string. The essential preferable attribute of simhash is the one which makes the number of common bits in the simhash identity of two similar documents higher and positively correlated to the similarity of the observed documents. Computing the n-bits simhash of a document is based on computing the n-bits signature of each term, as described below. Further, the frequency of each term t in the document needs to be calculated and denoted as the weight of t. Next, the vector V of f random integers is declared where to each element of $V[i]$ the weight of each term is weather added or subtracted depending on whose signature has a 1 or 0 in the *i-th* position. Consequently, the elements in simhash vector V are set to 1 in positions i with $V[i]>0$ and 0 otherwise.

Term Hash Function

There are many different hashing tactics for computing the hash signatures on term level. Optimal hash function should assign dissimilar signatures to different terms and oppositely similar hash signatures to comparable terms. We use the one presented in Fig. 2. More details on actual OpenCL implementation are given in next section.

Matching Phase

After the simhash fingerprints are ascertained, the issue is the means by which to effectively distinguish near-duplicate sets of documents, that is, documents whose unique fingerprint values have at most k diverse bits. For 64-bit fingerprints, $k = 3$ is a typical value used (Moussalli *et al*., 2011).

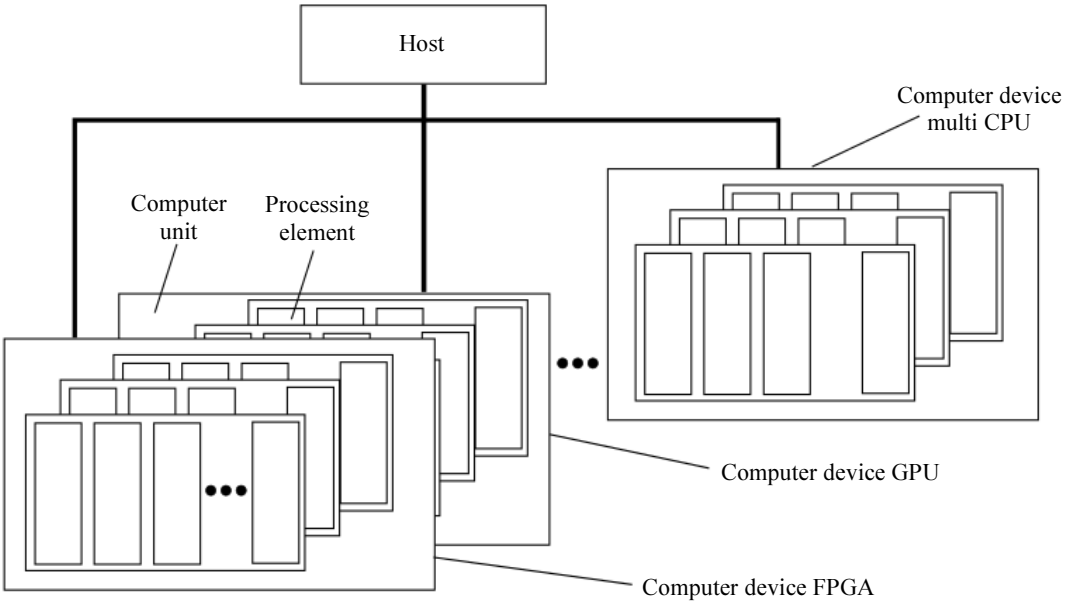

Fig. 1: OpenCL programming model

```
kernel void sh_hashOCL(__global char* a, __global int* b,
__global unsigned long int* c, __global int *countIn, __global int *countOut, const
  unsigned int count, const unsigned int inC)
  {
  int i = get_global_id(0);
  if(i%count < count)
  {
    for (int j = 0; j < inC; j++)
    {
    unsigned int nKeyLength = count;
    unsigned long int hash = 5381;
    int pM = count*i+0;
    for(; nKeyLength >= 8; nKeyLength -= 8)
      {
      for(int k=0; k < 8; k++)
     {
         hash = ((hash << 5) + hash) + a[pM++];
         switch(nKeyLength)
         {
          case 7: case 6: case 5: case 4: case 3: case 2: case 1:
               hash = ((hash << 5) + hash) + a[pM++]; break;
          case 0: break;
         }
       c[i] = hash;
     }
    }
  }
 }
```

Fig. 2: OpenCL implementation of hash function

Many different approaches to upgrade this phase by evading the naïve quadratic cost of unique fingerprints examinations (Manku *et al*., 2007). Specifically, the key issue is the means by which to choose if new document is close copy to any of the current records in the collection. The primary thought in (Moussalli *et al*., 2011) which we have likewise executed in our experiments, is to make a few duplicates of the table of fingerprints of the collection, where in each duplicate the places of the bits are permutated. The primary instinct is that the prefix of another simhash will coordinate the prefix of no less than one of the duplicates of each close copy simhash in the collection. Consequently, the binary search on each of the duplicate sets utilizing the prefix bits to locate all near-identical documents is used. The quantity of duplicates and the length of the corresponding prefix are resolved in view of the estimation of k and space versus time tradeoff contemplations Additional subtle elements are accessible at (Manku *et al*., 2007).

Computing Simhash using OpenCL

The basic implementation is based on a kernel which starts one parallel thread per document which in return calculates the simhash and maps the each term in the document to an output array. This kernel is shown in Fig. 2. We assume that documents have been converted to the bag-of-words format and focus only on the simhash calculation part of the algorithm. In doing so, we used a kernel written in OpenCL.

Experiments

To evaluate the OpenCL implementation of simhash and particularly the efficiency of the Xilinx's SDAccel development environment and their OpenCL-to-FPGA compiler, we implemented simhash on three successful HPC platforms: Multi-core CPU, GPU and the FPGA. The summary of platform is given on Table 1. First, the Intel Xeon E5-2650 processor is used to evaluate the multi-core CPU platform. This cutting edge processor includes twelve cores running at 2.2 GHz with 30 MB cache with hyperthreating.

For the GPU proxy, we selected the NVIDIA Tesla K40C GPU Computing Accelerator - 12GB GDDR5. For the FPGA representative, we use the Xilinx ADM-PCIE-7V3 board with a Xilinx Virtex-7 FPGA. The Intel processor is at a higher leading development node and has a sizeable cache ready for use. Even though the Xilinx Virtex-7 FPGA and the Tesla K40C GPU are based on the equal 28 nm process, the GPU involve ten times higher than the memory bandwidth of the FPGA with more than ten times higher power utilization. In this study, following the conventions, we report the power utilization of GPU and FPGA taking also into account their memory power consumption which is not case for multi-core CPU.

Experimental Data Generation

To evaluate our implementation, we utilized methodology known as random text generation. We designed a statistical model with the close properties of the news web portals. Respectively, the statistically generated documents include a mean of 500 different terms and have the mean length of 4K terms. We produced 256 thousand documents using this overall methodology. For testing purposes, we randomly generated 32K entries. The likelihood of a document term selecting a nonzero value in this setting was calculated as 7.6e-4.

To evaluate each implementation, various experiments are directed to decide the best setup for a selected platform. When detailing the last outcome on every platform, we have bent over backward to guarantee that the OpenCL code has been tuned in suitable approaches to completely use each targeted architecture. The execution is measured in million terms for every second (MT/s). This execution is then separated by the relating board energy to figure the execution per-watt estimation. This is communicated as million terms for each joule (MT/J) in the outcomes beneath.

Multi-Core CPU Experiment Outcomes

We first test the OpenCL coupling algorithm on a multi-core Xeon E5-2650 CPU. Table 2 presents the results of diverse parameter settings and their effect on the efficiency. We denote the count of parallel threads treating different elements as T in the table. The best CPU results are observed when number of treats is set to one which in return arguments the fact that the CPU prefer executing one thread per document.

Supposedly, the reason of this phenomenon is the modern multicore design where each core uses its local on-chip cache for processing the particular document section. The multi core CPU experiments show best results with processing 2080 million terms per second (MT/s). Since the CPU power consumption is 105 Watts, the overall performance-to-power ratio is 19 million terms per joule.

GPU Experiment Outcomes

Graphics Processing Unit is the next platform used in evaluation of our OpenCL application. GPUs are extremely enhanced processors designed to reach high computational power in graphics manipulation. The general-purpose processor is specifically designed to reduce latency while the GPU is optimized to maximize application throughput. In doing so, GPUs uses tens of thousands of threads in parallel on an array of computing units. GPUs also include hardware solutions for context switching among set of threads when the current set of threads are postponed waiting for global memory usage. In this way processing units improve their performance and overcame memory access latencies. As an example GPU architecture we present Kepler architecture. The building block of the GPU is the Streaming Multiprocessor (SM). It contains 32 floating-point computing units, or Stream Processors (SP) and 4 special function units for transcendental calculations. Because a group of threads often work together on the same set of data, the SM features local memory shared between threads as well as caches so that fewer global memory accesses are necessary.

The best results for the GPU is 3760 MT/s, see Table. 3. The board power of the K40C is 235Watts and leads to a performance to power ratio of 16 million terms per joule.

Table 1: Specifications of platforms under test

Test platform	Representative	Process	Memory bandwidth	Cache size	Board power
Multi-Core CPU	Intel Xeon E5-2650 v.4	14 nm	76.8 GB/s	30 MB	105 W
GPU	NVIDIA Tesla K40C	28 nm	288 GB/s	1 MB	235 W
CPU	Xilinx Virtex-7	28 nm	12.8 GB/s	None	20 W

Table 2: CPU results

Configuration	MT/s	MT/J
T = 1	2080	19.0
T = 2	1720	16.3
T = 4	1800	17.1

Table 3: GPU results

Configuration	MT/s	MT/j
T = 64	2341	10.0
T = 128	2782	11.9
T = 256	3760	16.0
T = 512	2794	11.9

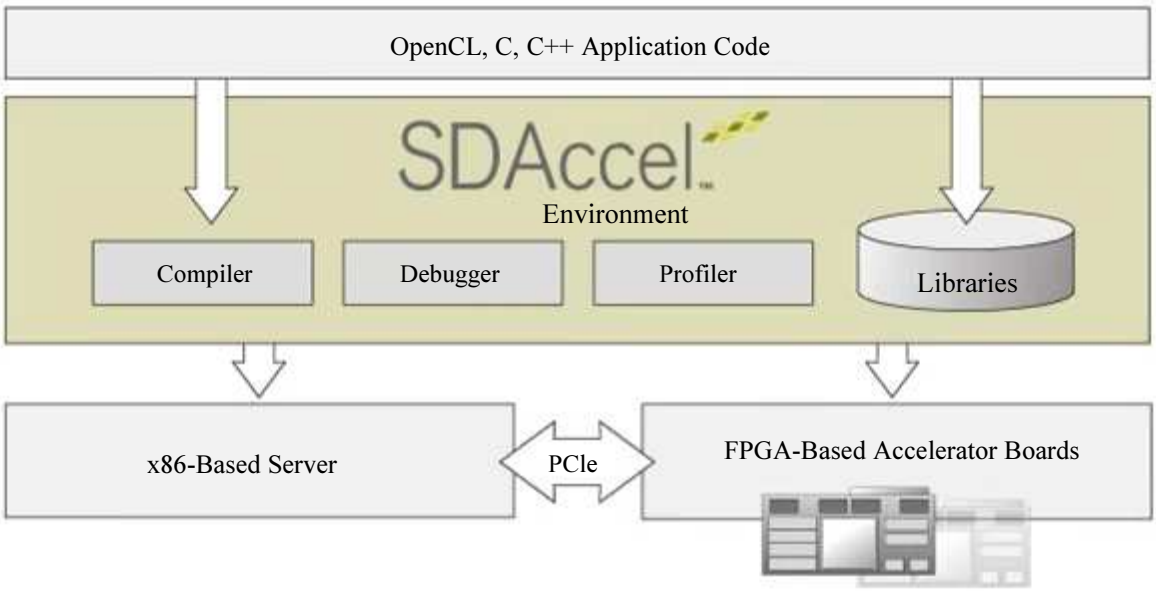

Fig. 3: SDAccel envirovement

Table 4: FPGA results

Configuration	***MT/s***	***MT/j***
T = 32	1690	84.5
T = 64	**1980**	**99.0**
T = 128	1650	84.0

FPGA Experiment Outcomes

As stated previously for FPGA experiments we used Xilinx ADM-PCIE-7V3 board with a Xilinx Virtex-7 FPGA. Actual development, coding, debugging, optimization and testing stage has been realized on SuperVessel OpenPOWER Cloud infrastructure via the SDAccel Application Development environment.

SuperVessel is unique open connection cloud service serving as a virtual R&D instrument for application development, system designing and academic research. It presents a one of its kind platform for creating, testing and prototyping resolutions for rising solutions counting deep learning, machine learning, deep analytics and many others. The Xilinx SDAccel Development Environment is a full software-determined Integrated Development Environment (IDE) which allows coders to compile, profile, debug and deploy FPGA-based acceleration. The combination of SuperVessel, IBM POWER architecture, the SDAccel Development Environment and Xilinx FPGA accelerator boards provide application developers with a high throughput, high availability cloud-based platform to develop and execute the compute intensive applications.

Table 4. shows the results of the OpenCL FPGA implementation. This board contains two DDR2-800 DIMMs (400 MHz memory clock, 800 MT/s) providing a peak bandwidth to external memory of 12.8 GB/s as shown in Fig. 3. Each of the key data buffers is allocated such that half of it resides in each DIMM. This allows the kernel to maximize the amount of data bandwidth available when accessing the data buffers.

The best results for the FPGA is 1980 MT/s. The board power of the Virtex-7 is 20 Watts and leads to a performance to- power ratio of 99 million terms per joule.

Conclusion

In this study, we have demonstrated how OpenCL can be utilized to unleash the power of FPGAs for server farm applications obliged by power and cooling expenses. At the point when thought about on a performance per watt premise, the FPGA can beat a tantamount GPU and CPU by a factor of 5.25x and 6.18x individually as appeared in Table 5.

There are huge groups of solutions where FPGA usage offer a critical preferred standpoint in progressively control obliged situations. The near duplicate identification application inspected in this study truly speaks to a lower bound on one of the best qualities of the FPGA; that is, its capacity to perform profoundly parallel and complex algorithmic calculations on the information brought on-chip. In this application, the data-path is generally basic. Our future work will take a look at more perplexing applications to show the productivity of utilizing FPGAs with high-level languages such as OpenCL. It is likewise fascinating to investigate a heterogeneous blend of FPGAs, GPUs and multi-core CPUs for data centre applications as each has specific qualities.

Table 5: Table type styles

Configuration	*MT/s*	*MT/j*
NVIDIA Tesla K40C	3760	16
Intel Xeon E5-2650 v.4	2080	19
Xilinx Virtex-7	1980	99

Acknowledgement

This work was completely supported by the Gjirafa, Inc. We also thanks POWER Technology Open Lab for allowing us to use their SuperVessel Cloud.

Ethics

There are no ethical issues or conflict of interest.

References

Canhasi, E. and I. Kononenko, 2014. Multi-document summarization via Archetypal Analysis of the content-graph joint model. Knowl. Inform. Syst., 41: 821-842. DOI: 10.1016/j.eswa.2013.07.079

Canhasi, E. and I. Kononenko, 2016. Weighted hierarchical archetypal analysis for multi-document summarization. Comput. Speech Lang., 37: 24-46. DOI: 10.1007/s10115-013-0689-8

Canhasi, E., 2016. Fast document summarization using locality sensitive hashing and memory access efficient node ranking. Int. J. Electr. Comput. Eng., 6: 945-945. DOI: 10.11591/ijece.v6i3.9030

Xiao, C., W. Wang, X. Lin, J.X. Yu and G. Wang, 2011. Efficient similarity joins for near-duplicate detection. ACM Trans. Database Syst., 36: 15-15. DOI: 10.1145/2000824.2000825

Chalamalasetti, S.R., M.V. Margala, W. Wright and P. Ranganathan, 2012. Evaluating FPGA-acceleration for real-time unstructured search. Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, Apr. 1-3, IEEE Xplore Press, New Brunswick, NJ, USA, pp: 200-209. DOI: 10.1109/ISPASS.2012.6189226

Chapman, B., G. Jost and R.V.D. Pas, 2007. Using OpenMP: Portable shared memory. Parallel Programming (Scientific and Engineering Computation), The MIT Press.

Charikar, M.S., 2002. Similarity estimation techniques from rounding algorithms. Proceedings of the 34th Annual ACM Symposium on Theory of Computing, May 19-21, ACM, Montreal, Quebec, Canada, pp: 380-388. DOI: 10.1145/509907.509965

Henzinger, M.R., 2006. Finding near-duplicate web pages: A large-scale evaluation of algorithms. Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Aug. 06-11, ACM, Seattle, Washington, USA, pp: 284-291. DOI: 10.1145/1148170.1148222

Luo, X., W. Najjar and V. Hristidis, 2013. Efficient near-duplicate document detection using FPGAs. Proceedings of the IEEE International Conference on Big Data, Oct. 6-9, IEEE Xplore Press, Silicon Valley, CA, USA, pp: 54-61. DOI: 10.1109/BigData.2013.6691698

Sood, S. and D. Loguinov, 2011. Probabilistic near-duplicate detection using simhash. Proceedings of the 20th ACM International Conference on Information and Knowledge Management, Oct. 24-28, ACM, Glasgow, Scotland, UK, pp: 1117-1126. DOI: 10.1145/2063576.2063737

Khronos OpenCL Working Group, 2008. Khronos OpenCL Working Group, The OpenCL Specification, version 1.0.29.

Manku, G.S., A. Jain and A. Das Sarma, 2007. Detecting near-duplicates for web crawling. Proceedings of the 16th International Conference on World Wide Web, May 08-12, ACM, Banff, Alberta, Canada, pp: 141-150. DOI: 10.1145/1242572.1242592

NVIDIA CUDA Compute Unified Device Architecture - Programming Guide, 2007.

Wirbel, L., 2014. Xilinx SDAccel: A unified development environment for tomorrow's data center. The Linley Group Inc.

Moussalli, R., M. Salloum, W.A. Najjar and V.J. Tsotras, 2011. Massively parallel XML twig filtering using dynamic programming on FPGAs. Proceedings of the IEEE 27th International Conference on Data Engineering, Apr. 11-16, IEEE Xplore Press, Hannover, Germany, pp: 948-959. DOI: 10.1109/ICDE.2011.5767899