# GPU-Based Odd and Even Hybrid String Matching Algorithm

**Ghazal Rahbari, Nur'Aini Abdul Rashid, and Wahidah Husain**
*Universiti Sains Malaysia,Malaysia, {ghazal.rahbari; nuraini; wahidah}@gmail.com*

## ABSTRACT

String matching is considered as one of the fundamental problems in computer science. Many computer applications provide the string matching utility for their users, and how fast one or more occurrences of a given pattern can be found in a text plays a prominent role in their user satisfaction. Although numerous algorithms and methods are available to solve the string matching problem, the remarkable increase in the amount of data which is produced and stored by modern computational devices demands researchers to find much more efficient ways for dealing with this issue. In this research, the Odd and Even (OE) hybrid string matching algorithm is redesigned to be executed on the Graphics Processing Unit (GPU), which can be utilized to reduce the burden of compute-intensive operations from the Central Processing Unit (CPU). In fact, capabilities of the GPU as a massively parallel processor are employed to enhance the performance of the existing hybrid string matching algorithms. Different types of data are used to evaluate the impact of parallelization and implementation of both algorithms on the GPU. Experimental results indicate that the performance of the hybrid string matching algorithms has been improved, and the speedup, which has been obtained, is considerable enough to suggest the GPU as the suitable platform for these hybrid string-matching algorithms.

**Keywords**: Odd and Even, Hybrid String Matching, GPGPU.

## I INTRODUCTION

String matching is considered as one of the fundamental problems in computer science which involves finding one or more occurrences of a given pattern in a text. Although numerous algorithms and methods are available to solve this problem, many researchers are still trying to achieve much more efficient ways to deal with this issue. Hybrid string matching algorithms, a combination of two or more string matching algorithms, have received a great deal of attention in recent years (Abdurrazaq, et al. 2014(Mustafa et al., 2012)). Positive features of the existing string matching algorithms are combined to form a new algorithm in order to improve the searching process. Computer applications which provide string matching utility for their users can take advantage of this kind of string matching algorithms (Almazroi, 2011). In addition, multi-core Central Processing Units (CPUs) as well as many-core Graphics Processing Units (GPUs) have gained an extensive popularity over the last few years. This development at the hardware level has made an inevitable challenges for researchers who are interested in the string matching problem. Existing sequential string matching algorithms should be redesigned in order to exploit computational power of the modern processors. As a result, execution of the parallelized string matching algorithms on these processors helps decrease the searching time. How the existing sequential hybrid string matching algorithms can be parallelized and implemented on the GPU is the focus of this research.

In this research we analyzed the Odd and Even hybrid string matching algorithm in order to identify the compute-intensive portions of the sequential code, parallelize the hotspots of the selected hybrid string matching algorithm and implement it on the GPU and compare the performance of the sequential version on the CPU with the parallel version on the GPU of the selected hybrid string matching algorithm.

This paper presented the existing works in parallel string matching algorithms in section II. The section III discussed the detail hybrid Odd and Even algorithms, followed by the design of the parallel Odd and Even algorithm. Section V presented the results of the research and we conclude the finding in section VI.

## II PARALLEL STRING MATCHING

We studied some of the important work in parallel string matching algorithms (Table 1). Michailidis and Margaritis (2001b) have implemented the Brute Force exact string matching algorithm on a cluster of six personal computers. Noticeable reduction in the execution time has been achieved by using the SPMD parallel programming model over the master-worker paradigm. In their implementation, text is broken down into several subtexts with an overlap of m-1 characters, where m is the length of the pattern. Then, these subtexts are assigned to available processors, which perform the string matching procedure simultaneously on their corresponding data, and send their final results to the master processor. They have used a preprocessing allocation method to avoid the load balancing problem which occurs when the subtexts of the database do not have equal length (Michailidis & Margaritis, 2001b).

Parallel implementation of the Naïve, Karp and Rabin, Zhu and Takaoka, Baker and Bird, and Baeza-Yates and Regnier exact two dimensional pattern matching algorithms has been presented by Kouzinopoulos and Margaritis (2009a). These algorithms have been implemented on a multi-core processor and a homogeneous cluster of workstations as shared memory as well as distributed memory parallel platforms by using OpenMP and MPI APIs, respectively. Master-worker distribution method has been applied on both parallel systems. In the shared memory parallelization, it has been shown that the performance is increased when the assignment of the loop iterations to the threads is performed by static scheduling clause compared with dynamic as well as guided scheduling clauses. Moreover, in order to decrease the communication overhead in the distributed memory parallelization, text and pattern are located in the local memory of each processor, which performs the searching process on the corresponding section of the text. This section is determined by the pointer, which has been sent from the master process to the workers (Kouzinopoulos & Margaritis, 2009a).

**Table 1. Some of the works in Parallel String Matching Algorithm.**

| String Matching Algorithm | Parallel Platform | Application Programming Interface (API) |
|---|---|---|
| Brute Force | Cluster of 6 Personal Computers | MPI |
| Naïve<br>Karp and Rabin<br>Zhu and Takaoka<br>Baker and Bird<br>Baeza-Yates and Regnier | A Multi-core Processor and Homogeneous Cluster of Workstations | MPI<br>OpenMP |
| Knuth-Morris-Pratt<br>Boyer-Moore<br>Horey-Moore-Horspool<br>Zhu-Takaoka<br>Quick Search<br>Berry-Ravindran<br>Fast Search<br>SSABS<br>TVSBS<br>ZTMBH<br>BRBMH | Homogeneous Cluster of Workstations | MPI |
| Naïve<br>Knuth-Morris-Pratt<br>Horey-Moore-Horspool,<br>Quick Search | GPU | CUDA |
| Boyer-Moore | GPU | CUDA |

Parallel execution time of the Knuth-Morris-Pratt, Boyer-Moore, Boyer-Moore- Horspool, Zhu-Takaoka, Quick Search, Berry-Ravindran, Fast Search, SSABS, TVSBS, ZTMBH and BRBMH string matching algorithms have been compared by Prasad and Panicker (2010). Implementation of these algorithms has been performed on a Bewoulf-based homogeneous cluster of workstations with 40 nodes by using MPI API. Based on experimental results of this research, the BRBMH algorithm has the lowest parallel execution time for any pattern lengths as well as text sizes (Prasad & Panicker, 2010).

The Naïve, Knuth-Morris-Pratt, Boyer-Moore-Horspool and Quick Search string matching algorithms have been implemented on the GPU using the CUDA API (Kouzinopoulos & Margaritis, 2009b). In order to calculate the speedup, the practical running time of the algorithms on different data sets as well as different pattern lengths has been measured which includes the preprocessing time, the searching time and the time needed to transfer data between the host and the device. The considerable impact of using the shared memory of the GPU instead of the global memory to store the pattern as well as the pre-computed shift value table of the mentioned algorithms has been illustrated in their research. Moreover, it has been shown that the practical running time reduces by increasing the number of threads in order to keep the GPU entirely utilized (Kouzinopoulos & Margaritis, 2009b). Ryan (n.d.) has implemented the Boyer-Moore string matching algorithm on the GPU using CUDA API. The preprocessing phase of the algorithm is performed on the CPU, while the searching phase is carried out on the GPU. Moreover, the text, the pattern and the pre-processed shift value tables are transferred from the host to the device and stored in global, constant and texture memories, respectively. The performance of the parallel algorithm has been analyzed by using the maximum number of available blocks. Also, it has been shown that transferring the data from the CPU to the GPU decreases the searching time much more than using the page-locked CPU memory which is accessible directly by the CUDA kernel running on the GPU (Ryan, K., n.d.). Other works on parallelizing the string matching algorithms include Naser, M. A. (2010), Atheer Akram AbdulRazzaq et.al, (2013), Abdulwahab and Nur'Aini (2011), Atheer , Nur'Aini and Aziz Nasser Boraik Ali (2013) and Awsan et. al. (2013).

Based on our study, the Odd and Even hybrid string matching algorithms have been selected to be parallelized and implemented on the CPU+GPU parallel platform with the CUDA programming interface. The Odd and Even hybrid string matching algorithm makes an efficient use of the Berry-Ravindran algorithm. It has been compared with the BMH, the QS, the TVSBS, the BRFS, the BRBMH, and the BRQS algorithms and provided better results for searching any lengths of the pattern string and any sizes of the character set (Naser, 2010). The preprocessing phase of both mentioned algorithms is executed once before the searching phase which needs to be run repeatedly as a single program on a huge amount of data. This part of the string matching algorithms is capable of being parallelized to be executed simultaneously on different sections of data which leads to a better performance by decreasing the

searching time. Parallel version of each mentioned algorithms is implemented on the GPU which is based on the SPMD parallel programming model. Its particular architecture is suitable for data parallel applications which involve string matching algorithms. The searching process can be accelerated by using a large number of the GPU threads to execute the searching phase of the string matching algorithms concurrently on independent parts of the text string. Although the BR algorithm has been parallelized and implemented on the shared memory as well as the distributed memory parallel platforms, both mentioned algorithms have not been implemented on the GPU before.

## III ODD AND EVEN ALGORITHM

The Odd and Even algorithm is a hybrid string matching algorithm with a similar preprocessing phase to the Berry-Ravindran algorithm. The comparison between the text characters as well as the pattern characters is performed from right to left with a specific order which is described in the following subsection. This characteristic of the OE algorithm has a considerable impact on the performance. Similar to the Berry-Ravindran algorithm, the searching phase starts from the leftmost character of the text without checking the possible starting search point.

**Table 2. Odd and Even Bad Character Shift Value Table.**

| brBc | a | c | g | * |
|---|---|---|---|---|
| a | 10 | 10 | 2 | 10 |
| c | 7 | 10 | 9 | 10 |
| g | 1 | 1 | 1 | 1 |
| * | 10 | 10 | 9 | 10 |

### A. Pre-processing

In the preprocessing phase of the OE algorithm, shift values are computed based on each pair of characters belonging to the pattern string. This feature makes the preprocessing phase of the OE algorithm different from the preprocessing phase of the Berry-Ravindran algorithm although the former applies the same formula as the latter to construct the bad character shift value table. In other words, the preprocessing phase of the Berry-Ravindran algorithm has been enhanced in the OE algorithm which results in reducing the total execution time as well as the reserved memory size of the computer. The bad character shift value table of the OE algorithm based on a pattern string "gcagagag" of length 8 as well as a character set (a,c,g,t) of size 4 is shown in Table 2.

### B. Searching

In the searching phase of the OE algorithm, the leftmost character of the pattern is aligned with the leftmost character of the text. Then, the comparison between the text characters as well as the pattern characters is performed from right to left. Even positions of the pattern are compared with the corresponding text characters after the odd positions of the pattern match the corresponding text characters, or vice versa. In case of a mismatch or a complete match, two consecutive characters next to the right side of the window are used to achieve the shift value from the bad character shift value table in order to move the pattern along the text. This procedure is repeated until the right end of the pattern exceeds the right end of the text.

The searching process of a pattern string "gcagagag" of length 8 through a text string "tctgtgaggattgattgcagagag" of length 24 in one attempt based on the bad character shift value table, which has been constructed in the previous subsection, is illustrated in Figure 1.

## IV METHODOLOGY

The methodology of this research consists of five steps. It starts with studying various existing hybrid string matching algorithms. The Odd and Even hybrid string matching algorithm, which has its own advantages and disadvantages, has been selected to be parallelized and implemented on the GPU (Samsudin, 2011),(Charras, & Lecroq, 2004),(Klaib & Osborne, 2009b). Similar to many other hybrid string matching algorithms, the OE algorithm takes advantage of the mentioned characteristic of the BR algorithm, but it provides better results for searching any lengths of a pattern string and any size of a character set compared with them), (Klaib & Osborne, 2009b).

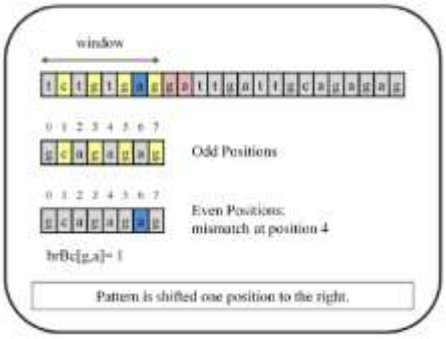


**Figure 1. Searching Phase of Odd and Even Algorithm.**

The second step involves analyzing the sequential version of the selected hybrid string matching algorithm to identify the compute-intensive portions of the serial code. This sequential version is considered as a baseline, which the parallel version is compared with.

The third step consists of designing the parallel version of the OE hybrid string matching algorithm. In fact, hotspot of the sequential version of the mentioned algorithm is parallelized.

In order to parallelize the string matching algorithm, it is necessary to determine how the problem can be decomposed into smaller parts. Data decomposition technique is used to divide the data which the string matching algorithm must deal with. Then, the searching operation is performed on different parts of the data which are independent of each other. Partitioning the text string into the discrete subtexts leads to a problem which occurs when the pattern string is located in the boundary of adjacent subtexts. Therefore, the pattern string cannot be detected, and a false negative result is returned by the string matching algorithm. An overlap of m-1 characters between consecutive subtexts is used to solve the mentioned problem, where m is the length of the pattern string. Since the searching function is performed on all subtexts of the text string simultaneously, Single Program Multiple Data (SPMD) as the most appropriate parallel programming model is used here.

The CUDA API provides different ways to allocate memory on the CPU as well as the GPU. The CPU pageable memory is allocated by the malloc() function, while the CPU page-locked memory is allocated by the cudaMallocHost() function. In this implementation, the CPU memory allocation is performed by the malloc() function. Although the page-locked memory is accessed by the GPU with higher bandwidth than the pageable memory, the system performance can be reduced if an excessive amount of page-locked memory is allocated by the CUDA subroutine which results in decreasing the amount of physical memory available to the operating system for paging. Moreover, the cudaMalloc() function is used to allocate the GPU memory in order to store the text string of both string matching algorithms. Data transfer between the host and the device is performed by the cudaMemcpy() function. The text string which is needed to be shared by all the GPU threads is transferred from the CPU to the GPU by the mentioned synchronous CUDA subroutine which blocks the CPU threads until the data has been transferred completely and stored in the GPU global memory space. The cudaMemcpyToSymbol() function is used to copy the pattern string of both string matching algorithms from the host to the device which is located in the GPU constant memory space. The character set of both string matching algorithms is also transferred from the host to the device by the cudaMemcpyToSymbol() function and resides in the constant memory space. Similar to the text string, the pattern string and the character set are needed to be accessed by all the GPU threads. Finally, the texture memory space of the GPU is used to store the shift value table which is constructed by the preprocessing function on the CPU. Therefore, each thread can achieve the corresponding shift value through the texture memory which can be read by all the GPU threads during the execution of the searching function. In addition, the result of the searching function which includes the number of occurrences of the given pattern string in the text string is transferred from the GPU to the CPU. The GPU memory allocation which is used for the implementation of the parallel the Odd and Even hybrid string matching algorithms is shown in Figure 2.

On the NVIDIA Tesla C2050, which is used in this implementation, data is transferred between the CPU and the GPU through the PCIe × 16 Gen2 bus with the theoretical maximum bandwidth of 8 GBps. This program has been executed with the pageable CPU memory space against the NVIDIA Tesla C2050.

In this implementation, the preprocessing function which calculates the shift values of the string matching algorithm is executed sequentially on the CPU, while the searching function is executed N times by N threads in parallel on the GPU. The device function, which is specified by the two parameters inside the triple angle brackets, is called from the host code. These two parameters represent the number of blocks and the number of threads. The maximum number of these two parameters depends on capabilities of the GPU on which the device code is executed.

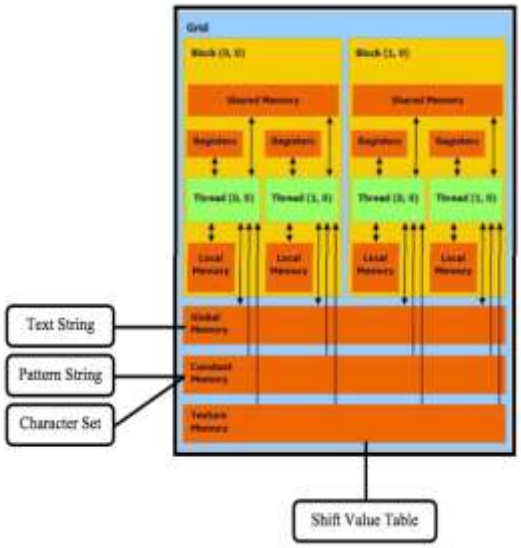


**Figure 2. GPU Memory Allocation in the String Matching Problem.**

Threads within a same block can communicate with each other through the shared memory which is not accessible by threads in other blocks. In the implementation of the parallel Odd and Even hybrid string matching algorithms, each thread of a single block executes the searching function independently and writes the number of occurrences of the given pattern string in its corresponding subtext in the shared memory. In order to avoid the wrong result, threads within a same block must be synchronized which is performed by the __syncthreads() function

before the total number of occurrences is computed. Furthermore, the atomicAdd() function is used to compute the final result of the device code by adding the total number of occurrences of the given pattern string which has been calculated by each block. This function is used to prevent race condition from happening. Implementation of the parallel Odd and Even hybrid string matching algorithms on the CPU+GPU parallel platform is shown in Figure 3.

The fourth step involves implementing the parallel version of the selected hybrid string matching algorithm on the CPU+GPU parallel platform. CUDA programming interface, which has been designed for only NVIDIA's GPUs, is used in this implementation. Karimi, Dickson, and Hamze (2011) have compared the performance of CUDA as well as OpenCL programming interfaces in terms of data transfer times, kernel execution times, and application execution times by using almost identical kernels and have suggested CUDA as a better choice.

Finally, the sequential version as well as the parallel version of the OE hybrid string matching algorithm are tested on the standard data types which consist of English text, protein sequence, and DNA sequence. Execution time, speedup, and percentage of performance gain which are considered as the performance metrics of the parallel system will be calculated and evaluated in this step. The execution time is divided into serial runtime and parallel runtime. They are the execution time of the sequential algorithm and the parallel algorithm, respectively. The speedup is used to determine how much faster the parallel code executes in comparison with the best sequential code, while the percentage of performance gain is applied to specify how much performance is obtained by parallelizing the sequential algorithm (Grama, Gupta, Karypis, & Kumar, 2003), (Naser, 2010).

## V RESULTS AND DISCUSSION

The correctness of the parallel version of the Odd and Even algorithm has been checked by comparing the number of occurrences of a given pattern string in a text string with the result which has been obtained by executing the sequential version of the mentioned algorithm with the same pattern as well as the same text.

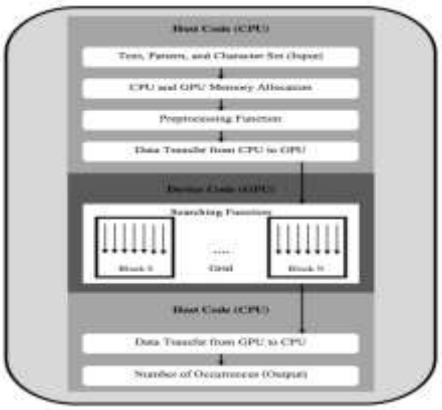


**Figure 3. Implementation of the String Matching Algorithm on GPU.**

The searching phase of the algorithm has been parallelized. Therefore, the sequential runtime and the parallel runtime of the algorithm, which are compared together, do not include the preprocessing phase. The elapsed time from the start to the end of the searching function on the sequential platform is measured as the sequential runtime. The parallel runtime includes the time of the data transfer from the CPU to the GPU, execution of the searching function on the parallel platform, and the data transfer from the GPU to the CPU.

The program is executed five times, and the average of the obtained values, which are shown in Appendix D, is calculated in order to decrease the random variation. In addition, the sequential and the parallel versions of the algorithm are tested on English text, protein sequence, and DNA sequence data types with five different sizes of 100, 200, 300, 400, and 500 MB (Naser, 2010). The program is executed with an 8-character pattern string which is selected randomly from the corresponding text string (Samsudin, 2011).The character set of the pattern string has been obtained by running the program which can be used to achieve all the different characters of the pattern string as well as the size of the character set.

The number of blocks per grid and the number of threads per block as the kernel parameters are set to 14 and 768, respectively. Selected kernel parameters have been tested by the NVIDIA Corporation's occupancy calculator and resulted in 100% occupancy.

Experimental results of this research, as shown in Table 3. which indicates the performance of Odd and Even string matching algorithms on different standard data types has been enhanced considerably by using the CPU+GPU parallel platform.

The performance improvement of Odd and Even is more than 90 percent on English text and protein sequence and between 80 and 90 percent on DNA sequence data types.

Unavoidable flow control instructions of the parallel

OE algorithm which leads to the thread divergence problem prevent it from running faster and have good speed-up.

Experimental results of running the sequential and the parallel versions of the Odd and Even hybrid string matching algorithm has been evaluated. Sequential and parallel version of this algorithm has been executed on English text, protein sequence, and DNA sequence data types as the standard benchmark with different sizes. Then, execution time, speedup, and percentage of performance gain have been calculated in order to determine the impact of parallelization on the performance of these hybrid string matching algorithms.

Obtained results confirm that the parallel version of the algorithm runs faster on the GPU than its sequential version on all three mentioned data types. Therefore, GPU capabilities can be exploited to enhance the performance of the hybrid string matching algorithms. Moreover, in order to make better use of the GPU, the flow control instructions must be used as less as possible in the device code to avoid the thread divergence problem, which has a substantial influence on the performance of the program.

**Table 3. Perfomance Improvement of Odd and Even Algorithm.**

| Data Type | Text Size (MB) | Performance Improvement |
|---|---|---|
| English Text | 100 | 93.83% |
| | 200 | 93.77% |
| | 300 | 93.91% |
| | 400 | 93.55% |
| | 500 | 93.87% |
| Protein Sequence | 100 | 90.55% |
| | 200 | 90.30% |
| | 300 | 90.56% |
| | 400 | 90.31% |
| | 500 | 90.56% |
| DNA Sequence | 100 | 83.18% |
| | 200 | 82.82% |
| | 300 | 83.28% |
| | 400 | 82.89% |
| | 500 | 83.27% |

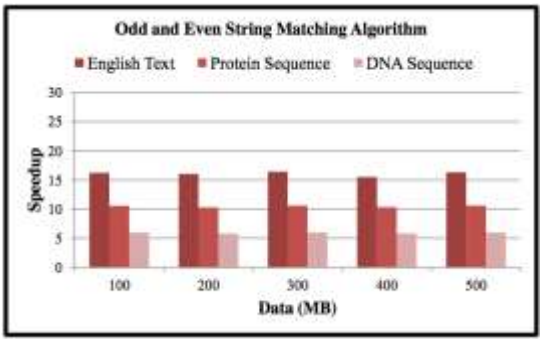


**Figure 4. Parallel Speed-up of the GPU-Based Odd and Even Algorithm**

## VI CONCLUSION

In this paper, we have presented a GPU –based parallel Odd and Even Algorithm. The searching phase of the Odd and Even algorithm were redesign to suit the parallel nature of the GPU processor. Experimental results show that the parallel algorithm has a substantial speed up over the sequential version.

## ACKNOWLEDGMENT

We would like to acknowledge the School of Computer Sciences, Universiti Sains Malaysia for the financial support in the publication of this paper.

## REFERENCES

Accreditation Commission for Programs in Hospitality Administration. (n.d.). *Handbook of accreditation*. Retrieved from http://www.acpha-cahm.org/forms/acpha/acphahandbook04.pdf

Billson, C. J. (1892). The Easter hare. *Folklore*, 3, 441-466. Retrieved from http://www.jstor.org

Bower, B. (2008, Feb. 9). Dawn of the city: Excavations prompt a revolution in thinking about the earliest cities. *Science News*, 173(6), 90-92. Retrieved from http://www.sciencenewsmagazine.org/

*Elementary school math instruction questionnaire results*. *Most significantly improved schools*. (n.d.). Retrieved from http://www.sharingsuccess.org/code/highperf/2002-03/es_math/msi/index.htm

Fuchs, D., Fuchs, L. S., Al Otaiba, S., Thompson, A., Yen, L., McMaster, K. N., Yang, N. J. (2001). K-PALS: Helping kindergartners with reading readiness: Teachers and researchers in partnerships. *Teaching Exceptional Children*, 33(4), 76-80. Retrieved from http://www.cec.sped.org/content/navigationmenu/publications2/teachingexceptionalchildren/

Goyen, A. (2007, February 22). Downtown Marquette dog sled races [Video file]. Retrieved from http://www.youtube.com/watch?v=gW3CNCGGgTY

Hartley, J. T., Harker J. O., & Walsh, D. A. (1980). Contemporary issues and new directions in adult development of learning and memory. In L. W. Poon (Ed.), *Aging in the 1980s: Psychological issues* (pp. 239-252). Washington, DC: American Psychological Association.

Herculano-Houzel, S., Collins, C. E., Wong, P., Kaas, J. H., & Lent, R. (2008). The basic nonuniformity of the cerebral cortex. *Proceedings of the National Academy of Sciences 705*, 12593-12598. doi: 1 0.1 073/pnas.08054171 05

Hipp, E. (2000). *Understanding the human volcano: What teens can do about violence* [Monograph]. Retrieved from http://www.eric.ed.gov/

Inness, S. A. (Ed.). (1998). *Delinquents and debutantes: Twentieth-century American girls' cultures*. New York, NY: New York University Press.

Katz, I., Gabayan, K., & Aghajan, H. (2007). A multi-touch surface using multiple cameras. In J. Blanc-Talon, W. Philips, D. Popescu, & P. Scheunders (Eds.), *Lecture Notes in Computer Science: Vol.* 4678. *Advanced Concepts for Intelligent Vision Systems* (pp. 97-108). Berlin, Germany: Springer-Verlag. doi:10.1007/978-3-540-74607-2_9

Langdon, S. W., & Preble, W. (2008). The relationship between levels of perceived respect and bullying in 5th through 12th graders. *Adolescence, 43*, 485-503. Retrieved from http://find.galegroup.com

Larson, G. W., Ellis, D. C., & Rivers, P. C. (1984). *Essentials of chemical dependency counseling.* New York, NY: Columbia University Press.

Lemay, L. (1997). *Teach yourself web publishing with HTML 4 in a week* (4th ed.). Indianapolis, IN: Sams.net.

Limb, G. E., & Hodge, D. R. (2008). Developing spiritual competency with Native Americans: Promoting wellness through balance and harmony. *Families in society, 89*, 615-622. doi:10.1606/1044-3894.3816

Lopez, J. (2005). *Characteristics of selected multilingual education programs from around the world: A review of the literature* (Unpublished master's thesis). Dominican University of California, Retrieved from ERIC database. (ED491402)

Moriarty, L. J., & Carter, D. L. (Eds.). (1998). *Criminal justice technology in the 21st century*. Springfield, IL: Charles C. Thomas.

Russo, C. A., & Jiang, H. J. (2006). *Hospital stays among patients with diabetes, 2004* (Statistical Brief #17). Retrieved from Agency for Healthcare Research & Quality website: http://www.hcup-us.ahrq.gov/reports/statbriefs/sb17.jsp

Shaw, K., O'Rourke, P., Del Mar, C., & Kenardy, J. (2005). Psychological interventions for overweight or obesity. *The Cochrane Database of Systematic Reviews*, (2). doi:10.1002/14651858.CD003818.pub2

Simon, C. E. (1995). *Information retrieval techniques: The differences in cognitive strategies and search behaviors among graduate students in an academic library* (Doctoral dissertation, Wayne State University). Retrieved from http://www.eric.ed.gov/

Symonds, P. M. (1958). Human drives. In C. L. Stacey & M. DeMartino (Eds.), *Understanding human motivation* (pp. 11-22). doi:10.1037/11305-002

U.S. Department of the Interior, National Park Service. (2004). *Pictured rocks national lakeshore, Michigan final general management plan, wilderness study, environmental impact statement. Washington, DC: Author.*

Wilens, T. E., & Biederman, J. (2006). Alcohol, drugs, and attention-deficit/hyperactivity disorder: A model for the study of addictions in youth. *Journal of Psychopharmacology, 20*, 580-588. doi:10.1177/0269881105058776