Vol.2010-ARC-192 No.21
Vol.2010-HPC-128 No.21
2010/12/17

Efficient PageRank on GPU Clusters

Ali Cevahir,[†1] Cevdet Aykanat,[†2] Ata Turk,[†2]
B. Barla Cambazoglu,[†3] Akira Nukada[†4]
and Satoshi Matsuoka[†4]

In this work, we report scalability of PageRank on multi-GPU clusters. Our target GPU clusters may contain more than one GPU accelerator per node. Iterative solvers for irregular sparse problems poorly scale with increasing number of processors because of load imbalance problem and network bottleneck. GPU computing units are too fast, for which network performance remains too low. Even the latest network hardware cannot provide bandwidth appropriate for high performance GPUs. In our previous work, we have introduced several implementation techniques and algorithms required for scalable sparse iterative solvers on multi-GPU extended clusters and evaluated those techniques on a Conjugate Gradient solver[5),6)]. In this work, we present the GPU cluster performance evaluation of another important iterative method, PageRank. For GPU implementation of PageRank, although we are inspired by the techniques that are presented in [6], we cannot use them as is, since PageRank data has very different characteristics than Krylov Method data. Our PageRank implementation on GPUs is based on our work on an efficient CPU cluster algorithm[7)]. In our experiments, we observe that PageRank achieves better scalability because of the enough data size to saturate GPUs. We observe scalability up to a hundred GPUs for PageRank, being at the same time almost 10 times faster than the CPU cluster implementation with the same number of CPU cores.

1. Introduction

Considering GPUs as high-performance low-cost many core co-processors, GPU clusters are being deployed for high performance scientific computing. GPU-based supercomputing systems have already taken their place in top ranks of the Top 500 list. One of the world's largest GPU clusters is deployed in Tokyo Institute of Technology, called TSUBAME2.0, with 2.4 PetaFlops peak performance. Recently, there is a rapid increase in research on applications running on GPU clusters as number of GPU-enhanced supercomputing systems increases.

Majority of those supercomputers are based on CUDA. Compute Unified Device Architecture (CUDA[8)]) is NVIDIA's new generation GPU hardware and software architecture. A CUDA GPU contains number of SIMD multiprocessors. GPU has a device memory that is accessible by all processors. Memory access latency of many core GPU devices is hidden by running high number of threads in parallel. Each multiprocessor contains its own shared memory and read-only constant and texture caches that are accessible by all processors within the multiprocessor. Threads in the same multiprocessor can communicate through fast shared memory. CUDA API supports programming different memory types.

As briefly explained above, GPU clusters has recently become effective computing resources. Scientists need new algorithms/methods/sources for running their applications on GPUs, in order to utilize these machines efficiently for their purpose. On the other hand, all applications cannot be easily and efficiently carried out on GPU clusters. Iterative solvers for matrices with irregular sparsity patterns are hard to be carried out efficiently on GPU clusters, but efficient implementations can enjoy high memory bandwidth supplied by GPUs. Note that sparse iterative solvers are memory-bound, i.e., there is only a few arithmetic operations per memory access.

Parallelization of iterative solvers for different parallel architectures is an active research area for decades. What makes the parallelization difficult for unstructured problems is the mass amount of communication between fine-grain computations. This problem is more obvious in GPU clusters, because GPU computation units are too fast, where network interconnect between nodes relatively remains extremely slow.

Several BLAS operations are consisted in sparse iterative solvers. Sparse matrix vector multiplication (MxV) is usually the most time-consuming of them. Parallel execution of sparse solvers for unstructured problems on a cluster requires considerable amount of communication, e.g., for sharing input and/or output vector of MxV. Hence, minimization techniques for inter-process communication should be considered for efficient parallel implementations. For a multi-GPU cluster,

†1 Rakuten Institute of Technology
†2 Bilkent University
†3 Yahoo! Research
†4 Tokyo Institute of Technology

parallelization is even harder. To achieve an efficient parallel implementation, parallelization inside a GPU between GPU cores, inside a node between GPUs and between nodes should be carefully handled. Kernels running on GPUs require high degree of fine-grained parallelization between cores of a GPU. This imposes careful workload distribution between GPU threads. Optimization techniques for accessing GPUs' complex memory architecture for memory-intensive kernels, e.g. MxV, should be carefully thought. Moreover, additional communication between GPU and CPU memories is required for utilizing GPU acceleration.

Above problems are handled for a CG solver in [6]. In this work, we present results on a different iterative solver called PageRank, which is an important component of effective Web search. Although this work is similar in the sense that PageRank is also a sparse iterative solver, since PageRank data has its own characteristics, techniques explained in [6] are not directly applicable. PageRank data is usually huge and number of links per page follows a power-law distribution. Experiments show that, although we observe slightly worse flop/s than CG for PageRank because of the power-law nature of the data, PageRank seems to scale better with the increasing number of GPUs.

To handle huge PageRank data, we have proposed repartitioning techniques of initially distributed data[7]. In this work, we accelarate the this PageRank implementation. Details of our previous works are explained in below section.

2. Related Work

Three important problems exist for unstructured sparse iterative solvers on GPUs: minimization of communication time between GPUs, development of efficient kernels for basic operations on GPUs and handling of the heterogeneity of the underlying system.

In [6], algorithms and techniques to overcome above problems are explained and a CG solver is implemented on a multi-GPU cluster. Each node of multi-GPU cluster contains several GPUs. Recent GPUs do not have a direct communication link between them, hence communications between GPUs are coordinated by host CPU(s).

Our major contributions in above-mentioned work can be summarized as:

- Scalable sparse iterative solver with unstructured matrices is achieved on multiple GPUs.
- All basic vector and matrix operations run on GPUs.
- We propose auto-selection for running MxV kernel in iterations.
- In order to decrease the communication time by minimizing total communication volume between GPUs, we enhance graph/hypergraph-partitioning-based sparse matrix decomposition models.
- We adapt matrix decomposition models for heterogenous GPU clusters using hierarchical partitioning.
- We make experiments on a wide range of well known datasets. Our experiments confirm the validity of proposed techniques. Also, experiments suggest that, without using such techniques, it is impossible to obtain scalability.

In our multi-GPU CG implementation, all basic vector and matrix operations of solvers are held on GPUs. For the core operation, MxV, we proposed a JDS-based GPU algorithm[5]. Proposed MxV algorithm achieves high utilization of GPU resources by coalesced memory accesses, caching, and load balancing between working threads. Simultaneously with our proposal, NVIDIA has released several kernels based on different formats[1]. For MxV, the solver automatically selects the fastest between several kernels proposed by NVIDIA and ourselves. For sparse matrix decomposition, to minimize communication and balance loads of MxV between nodes and GPUs, we utilize state-of-the-art 1D hypergraph partitioning models[4], which correctly encapsulate total communication volume in the NP-hard decomposition problem. Partitioning models are applied first between nodes, and then between GPUs within nodes for better reduction of communication for the slower communication link – network of the cluster among nodes.

To demonstrate effectiveness of our proposed techniques for CG, we held experiments on a set of well-known matrices. We show strong scalability by comparing GPU vs. CPU cluster implementations on the same underlying network, providing 20 Gbps per node. We achieve up to 119 Gflops of double-precision CG performance with 32 GPUs on 16 nodes of TSUBAME1.2, and 15.4 times speedup over single GPU implementation. This is 17.4 times faster than CPU implementation of the same number of nodes and CPU cores. We use up to 16 cores per node for CPU experiments, and observe that CG is always faster on

GPU cluster than CPU cluster.

We investigate PageRank in this paper. For PageRank, we mostly focus on minimization of matrix decomposition time, since number of iterations for PageRank is small. In [7], we utilize graph/hypergraph-partitioning-based decomposition techniques to minimize communication for parallel PageRank on CPU clusters. However, these techniques are not practical when applied directly, because of the vast size of the Web. We first focus on reducing this partitioning overhead. For this purpose, we propose two different Web matrix compression schemes by exploiting the site information inherently available in page links. Namely, instead of partitioning page-by-page matrix, we partition site-by-page, page-by-site or site-by-site compressed matrices. These partitioning models significantly decrease the preprocessing overhead of partitioning the original page-to-page matrix, without sacrificing the parallel efficiency.

In a real-world applications, Web dataset is likely to be distributed among many processors. In such a setup, the data has to be redistributed among processors for efficient parallel PageRank computations. Hence, partitioning models should encapsulate the initial data redistribution overhead as well as the communication overhead that will be incurred during the parallel PageRank computations. This problem constitutes a typical instance of the repartitioning (remapping) problem. In the work, we explain repartitioning models to encapsulate the initial matrix redistribution overhead.

PageRank computations conducted on well-known, large Web datasets indicate the effectiveness of the proposed techniques. These techniques result in considerably high speedups while incurring a preprocessing overhead of several iterations (for some instances even less than a single iteration) of the underlying sequential PageRank algorithm.

3. PageRank on GPUs

PageRank experiments on GPU clusters complements our discussions for CG in [6]. Since the matrices that are used in PageRank are much bigger than that of CG, GPU cluster experiments for PageRank demonstrate strong scalability even on a cluster with a hundred GPUs. Data sizes, matrix properties and convergence rates are different for CG and PageRank. Hence, optimization techniques are

Parallel-PageRank($\mathbf{A}_{11}^k$, $\mathbf{A}_{12}^k$, ${\mathbf{A}_2}^k$, $\mathbf{t}_k$, $\mathbf{u}_k$, α, ε)
▷ $\mathbf{p}_k = [\mathbf{p}_{1k}^{\mathrm{T}} \ \ \mathbf{p}_{2k}^{\mathrm{T}}]^{\mathrm{T}}$; $\quad \mathbf{t}_k = [\mathbf{t}_{1k}^{\mathrm{T}} \ \ \mathbf{t}_{2k}^{\mathrm{T}}]^{\mathrm{T}}$; $\quad \mathbf{u}_k = [\mathbf{u}_{1k}^{\mathrm{T}} \ \ \mathbf{u}_{2k}^{\mathrm{T}}]^{\mathrm{T}}$
▷ $\mathbf{p}_{1k} = [\mathbf{p}_{1k}^{\mathrm{T}}(\mathbf{A}_{11}) \ \mathbf{p}_{1k}^{\mathrm{T}}(\mathbf{Z})]^{\mathrm{T}}$; $\quad \mathbf{t}_{1k} = [\mathbf{t}_{1k}^{\mathrm{T}}(\mathbf{A}_{11}) \ \mathbf{t}_{1k}^{\mathrm{T}}(\mathbf{Z})]^{\mathrm{T}}$
▷ $\mathbf{u}_{1k} = [\mathbf{u}_{1k}^{\mathrm{T}}(\mathbf{A}_{11}) \ \mathbf{u}_{1k}^{\mathrm{T}}(\mathbf{Z})]^{\mathrm{T}}$
1. (a) $\mathbf{p}_{1k} \leftarrow \mathbf{t}_{1k}$
   (b) $\mathbf{p}_{1k}^{\mathrm{old}} \leftarrow \mathbf{p}_{1k}$
2. (a) $\pi^k \leftarrow \|\mathbf{p}_{1k}\|_1$
   (b) $\delta^k \leftarrow \|\mathbf{p}_{1k}\|_1$
   (c) $\langle \pi, \delta \rangle \leftarrow$ **AllReduceSum**$(\langle \pi^k, \delta^k \rangle)$
   (d) $\gamma \leftarrow 1 - \pi$
3. (a) $\widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow$ **ParMatVecMult**$(\mathbf{A}_{12}^k, \mathbf{t}_{1k}(\mathbf{Z}))$
   (b) $\widehat{\mathbf{u}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow$ **ParMatVecMult**$(\mathbf{A}_{12}^k, \mathbf{u}_{1k}(\mathbf{Z}))$
4. (a) $\widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) \leftarrow (1-\alpha)\widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z})$
   (b) $\mathbf{t}_{1k} \leftarrow (1-\alpha)\mathbf{t}_{1k}$
5. **while** $\delta > \varepsilon$ **do**
6. $\quad \mathbf{q}_{1k}(\mathbf{A}_{11}) \leftarrow$ **ParMatVecMult**$(\mathbf{A}_{11}^k, \mathbf{p}_{1k}(\mathbf{A}_{11}))$
7. $\quad \mathbf{q}_{1k}(\mathbf{A}_{11}) \leftarrow \mathbf{q}_{1k}(\mathbf{A}_{11}) + \widehat{\mathbf{t}}_{1k}(\mathbf{A}_{12}, \mathbf{Z}) + \alpha\gamma\widehat{\mathbf{u}}_{1k}(\mathbf{A}_{12}, \mathbf{Z})$
8. (a) $\quad \mathbf{p}_{1k}(\mathbf{A}_{11}) \leftarrow \alpha\mathbf{q}_{1k}(\mathbf{A}_{11}) + \mathbf{t}_{1k}(\mathbf{A}_{11}) + \alpha\gamma\mathbf{u}_{1k}(\mathbf{A}_{11})$
   (b) $\quad \mathbf{p}_{1k}(\mathbf{Z}) \leftarrow \mathbf{t}_{1k}(\mathbf{Z}) + \alpha\gamma\mathbf{u}_{1k}(\mathbf{Z})$
9. (a) $\quad \pi^k \leftarrow \|\mathbf{p}_{1k}\|_1$
   (b) $\quad \delta^k \leftarrow \|\mathbf{p}_{1k} - \mathbf{p}_{1k}^{\mathrm{old}}\|_1$
   (c) $\quad \langle \pi, \delta \rangle \leftarrow$ **AllReduceSum**$(\langle \pi^k, \delta^k \rangle)$
   (d) $\quad \gamma \leftarrow 1 - \pi$
10. $\quad \mathbf{p}_{1k}^{\mathrm{old}} \leftarrow \mathbf{p}_{1k}$
11. **end while**
12. (a) $\mathbf{q}_{2k} \leftarrow$ **ParMatVecMult**$(\mathbf{A}_2^k, \mathbf{p}_{1k})$
    (b) $\mathbf{p}_{2k} \leftarrow \alpha\mathbf{q}_{2k} + (1-\alpha)\mathbf{t}_{2k} + \alpha\gamma\mathbf{u}_{2k}$
13. **return** $\mathbf{p}_k$

ParMatVecMult $(\mathbf{A}^k, \mathbf{x}_k)$

(a) $\mathbf{x}'_k \leftarrow$ Expand $(\mathbf{x}_k)$	(a) $\mathbf{y}^k \leftarrow \mathbf{A}^k \times \mathbf{x}_k$
(b) $\mathbf{y}_k \leftarrow \mathbf{A}^k \times \mathbf{x}'_k$	(b) $\mathbf{y}_k \leftarrow$ Fold $(\mathbf{y}^k)$
Row parallel	Column parallel

Fig. 1 Parallel PageRank algorithm (pseudocode for processor P_k).[7)]

diverged for these two methods, although the idea behind is similar.

For GPU cluster implementation, the same parallel PageRank algorithm runs as the CPU cluster version, which is depicted in Fig. 1. The only difference is vector operations and MxV run on GPUs. Unlike GPU cluster implementation of CG, one MPI process is assigned for each GPU. By doing so, communication pattern for PageRank on GPU cluster become the same with CPU cluster. Hence, fairer comparisons can be made with CPU cluster implementation.

Another major difference of PageRank from CG on GPUs is that MxV kernel selection is not applied for PageRank. GPUs run CSR-vector[1] kernel. We observed that it is emprically the fastest MxV kernel on GPUs for Web matrices. Power-law Web matrices makes difficult to optimize MxV. Performance of CSR-vector kernel is satisfactory. However, proposal of a specific kernel for Web matrices may increase the PageRank performance.

Amongst the techniques explained in [7], we only test repartitioning on the GPU cluster, because matrices should be big enough to utilize GPUs and bigger matrices cannot be partitioned on single node. Just like CPU cluster experiments in [7], we use hypergraph partitioning for repartitioning of initially hash-based columnwise-distributed matrices.

3.1 PageRank Computation Times

In our tests, we use up to 96 Tesla GPUs on TSUBAME1.2 supercomputer, having 8 AMD 2.4 GHz Opteron dual core processors on each node. We use 2 Tesla GPUs and 2 CPU cores from each node.

We use 4 real Web data in our experiments: indochina, arabic-2005, uk-2005 and uk-union having 7.4M, 22.7M, 39.5M, 133.6M pages and 145.9, 640, 936, 5500M links, respectively[2),3)].

Matrices smaller than indochina are too small to get speedups on the GPU cluster. Fig. 2 depicts the speedups for `indochina` dataset over 4 GPUs. As you can see, it scales very poorly for GPUs more than 32.

For bigger datasets (`arabic-2005`, `uk-2005` and `uk-union`), we depict PageRank run times until convergence on CPU and GPU clusters in Figs. 3, 4 and 5. We provide run times for 32, 64 and 96 processors. Since `uk-union` data is too large to fit the memory of 32 GPUs, we provide results for 64 and 96 processors for this data. Respective GPU and CPU cluster performances in Gflops are given

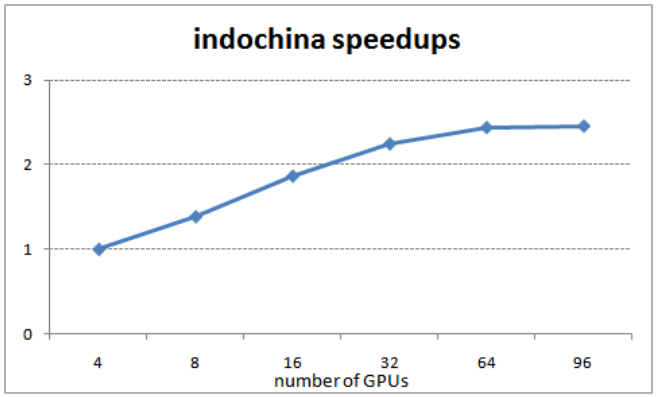

Fig. 2 GPU cluster speedups for `indochina`.

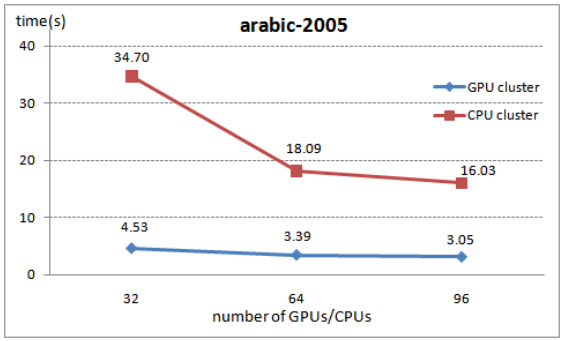

Fig. 3 PageRank run times on GPU and CPU clusters for `arabic-2005`.

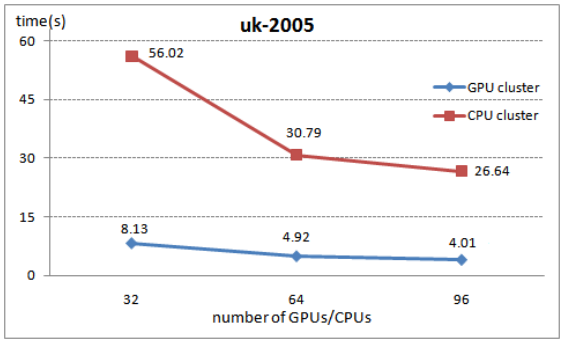

Fig. 4 PageRank run times on GPU and CPU clusters for `uk-2005`.

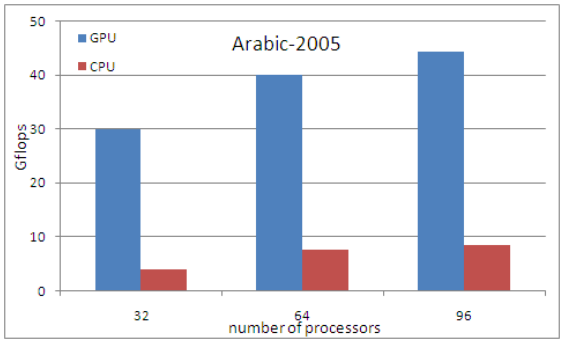

Fig. 6 PageRank performance in Gflops on GPU and CPU clusters for `arabic-2005`.

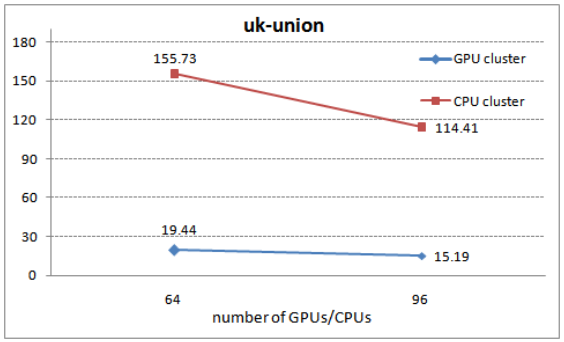

Fig. 5 PageRank run times on GPU and CPU clusters for `uk-union`.

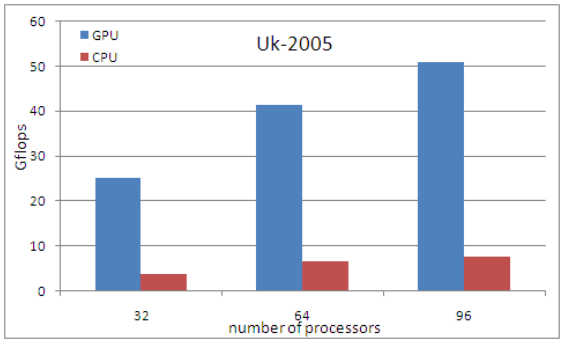

Fig. 7 PageRank performance in Gflops on GPU and CPU clusters for `uk-2005`.

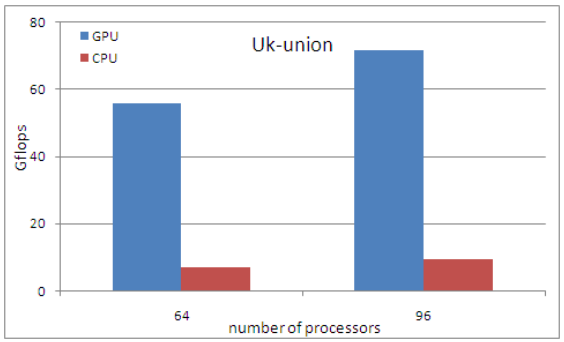

Fig. 8 PageRank performance in Gflops on GPU and CPU clusters for `uk-union`.

in Figs. 6, 7 and 8. From the figures, one can observe that bigger data runs faster on GPUs.

As can be seen from the figures and discussed before, performance gaps between GPU and CPU clusters are bigger for bigger matrices. For example, for `arabic-2005`, GPUs are 5.3 times faster than CPUs with 64 processors, while this ratio is 8.0 with same number of processors for `uk-union` data. These gaps become smaller as the number of processors increase. For 96 processors, speedups go far from the ideal line. One of the main reasons of fall-downs in speedups is the increasing load imbalance with increasing number of processors.

Note that we have hardly achieved speedups for 64 GPUs in our CG study, because of the insufficient matrix sizes[6].

4. Conclusion

In this work, we have shown scalability results for PageRank on a multi-GPU cluster. Previously, we have shown ways to implement efficient sparse iterative solvers on a CG example. However, PageRank has a completely different characteristics than CG. Although PageRank has a few iterations, we show ways to compansate initial partitioning cost[7]. Since we deal with bigger matrices for PageRank, problem better scales with the increasing number of GPUs. This shows the network bottleneck for sparse iterative solvers.

Hypergraph partitioning is executed as a preprocessing to iterative solvers. There are publicly available, effective and efficient partitioning tools for hypergraphs, but they are implemented on CPUs. Peak performance of GPUs is much higher than that of CPUs. Hence, preprocessing incurred by partitioning might be more severe for applications running on GPU(s). There exists several parallel partitioning tools which might be an alternative to reduce preprocessing time. Actually, it is required to use a parallel partitioning tool for bigger matrices, in case of insufficient single-node memory - like in the PageRank case. Still, there is a gap in performance scales of GPU computing and parallel partitioning tools that are developed for CPUs. An alternative is to develop partitioning tools on GPUs, although it is a challenging task. Partitioning tools use many heuristics. Even if these heuristics are implemented efficiently on GPUs, catching the quality of highly-developed CPU implementations requires some time.

References

1) N. Bell and M. Garland, "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors", *Proc. SC '09: ACM/IEEE Conference on Supercomputing*, Portland, OR, USA, 2009.
2) P. Boldi and S. Vigna, "The WebGraph Framework I: Compression Techniques", in *Proc. 13th Int'l WWW Conf.*, pp. 595–602, 2004.
3) P. Boldi, B. Codenotti, M. Santini and S. Vigna, "UbiCrawler: A Scalable Fully Distributed Web Crawler", *Software: Practice & Experience*, vol.43, pp.711–726, 2004.
4) U.V. Çatalyürek and C.Aykanat, "Hypergraph-Partitioning-Based Decomposition for Parallel Sparse-Matrix Vector Multiplication", *IEEE Trans. Parallel and Distributed Systems*, vol. 10, pp. 673–693, 1999.
5) A. Cevahir, A. Nukada and S. Matsuoka, "Fast Conjugate Gradients with Multiple GPUs", *Lecture Notes in Computer Science*, Vol.5544, Springer, pp. 898–903, 2009.
6) A. Cevahir, A. Nukada and S. Matsuoka, "High Performance Conjugate Gradient Solver on Multi-GPU Clusters Using Hypergraph Partitioning", *Proc. International Supercomputing Conference*, Computer Science - Research and Development (Special Issue), Springer, 2010.
7) A. Cevahir, C.Aykanat, A.Turk, and B.BarlaCambazoglu, "Site-Based Partitioning and Repartitioning Techniques for Parallel PageRank Computation", *IEEE Transaction of Parallel and Distributed Systems*, 2010.
8) NVIDIA Corporation: *NVIDIA CUDA Programming Guide*, 2009.