Semi-progressive Network Coding Algorithm on Multi-core Processor

Chang Zhu[1], Jianguo Xu[2], Yanqin Zhu[1*], Lingzhi Li[1]
[1]School of Computer Science & Technology, Soochow University, Suzhou 215006, China.
[2]Geologic Party No.216, China National Nuclear Corporation, Urumqi 830011, China.

* Corresponding author. email: yqzhu@suda.edu.cn
Manuscript submitted January 20, 2014; accepted September 5, 2014.
doi: 10.17706/jcp.10.1.24-33

Abstract: Network coding is a popular research topic which can help to improve throughput, reliability and security of communications. However, its decoding process is usually time consuming and the delay is sometime significant. Though the progressive network coding with Gauss-Jordan elimination can reduce the decoding time, the workload cannot be allocated equally among all processor cores. Those problems degrade network coding performance. In this paper, we put forward Semi-Progressive network coding, a new algorithm to narrow the workload gap between any two cores on a multi-core processor. We convert the decoding process into solving the system of linear equations. In addition, we propose a task allocation method. The result of theoretical analysis and calculation show that our algorithm can improve the performance and reduce the delay of decoding process.

Key words: Network coding, multi-core processor, system of linear equations, balanced workload.

1. Introduction

Network coding has caught widespread concern since it was presented by Ahlswede *et al*. [1]. In traditional network systems without network coding, intermediate nodes can only store and forward packets. But network coding technology breaks the routine to allow every node between a source and a destination to recode the content of multiple packets into a single packet. In the linear network coding system, the source node divides data into *n* blocks and codes them with random coefficient vectors. The destination node recovers the original data only if *n* linear independent packets are received.

Fig. 1 illustrates a directed graph representing simple network model with a simple network coding [1]. Symbols a and b represent information bits which are sent from the source node S to both D and E. As seen from the figure, both a and b arrive at node C. With the limited bandwidth, we can only send either a or b per unit time through C to F. To improve bandwidth efficiency between nodes C and node F, network coding function can be adopted.

In the example presented in Fig. 1, the XOR of a and b is calculated (encoding process) at node C and the result is sent to node F. As a result, both bits a and b are received at node E, by recovering bit a with b and a $\oplus$ b (decoding process). Bit b can be obtained by the same manner at node D. Thus, combining the data using network coding technique increases the bandwidth efficiency by enhanced data throughput [2]-[4].

What's more, using network coding will bring a lot of other benefits. It increases reliability [5] of data transmission and enhance security [6]. A new design prototype for peer-to-peer (P2P) system [7] has been brought up because of network coding technique. In practical P2P systems, network coding improves

performance of content distribution [8] [9] and file sharing [10].

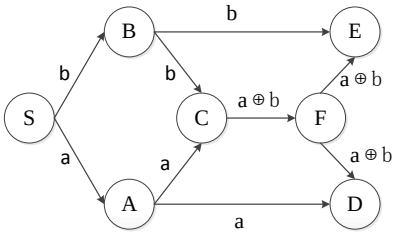


Fig. 1. Communication network for network coding.

Although network coding can bring many advantages above, the high computing overhead affects the practical implementation of network coding. Data on receiving node must be decoded to recover the original information. The complexity of the traditional Gauss-Jordan elimination is $O(n^3)$. The complexity is very high, in fact, especially when large files to be transmitted and offset the benefits of using network coding. Thus, it is extremely important to guarantee short decoding delay.

In this paper, we propose a semi-progressive network coding algorithm based on progressive network coding [11] and non-progressive network coding [12]. With a progressive scheme, the decoding operation starts once the packet is received. On the other side, the core ideal of non-progressive scheme is that the decoding operation cannot start until the last row is received. The timeline of the two schemes is shown in Fig. 2. In our algorithm, each of the received packets should be operated through *Phase A* and *B*, and *Phase B* doesn't start until n linear independent packets are processed in *Phase A*. After *Phase A*, the coefficient matrix becomes a triangular matrix. The algorithm converts the decoding process into solving system of linear equations to recover original information. The result of analysis shows that our algorithm can decline the decoding delay and balance the workload on processor cores.

The structure of this paper is organized as follows. In Section 2, we introduce the related work. In Section 3, we present our algorithm with detailed steps. We evaluate the algorithm with theoretical analysis in Section 4. Finally, we conclude the paper with Section 5.

2. Related Work

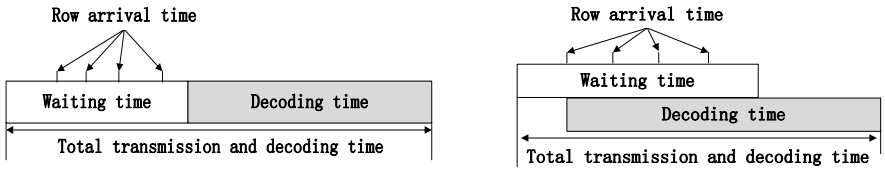
Major research progresses have been accomplished since Ahlswede *et al*. [1] introduced network coding technique and advantages. Chou *et al*. [13] presented the random linear network coding. Then, Koetter and Medard [14] proved that networks using linear network coding can achieve maximum throughput. Moreover, Ho *et al*. [15] first introduced the random linear network coding which chooses coefficient matrix over some finite field randomly. In this paper, we also adopt the random linear network coding. What's more, some schemes on multi-core processor and parallel computing have been studied in [16]-[18].

Table 1. The Process of Progressive Decoding

Stage A	Reduce leading coefficients in the new coefficient row to 0. [50.05%]
Stage B	Find the leading non-zero coefficient in the new coefficient row. [0.05%]
Stage C	Check for linear independence with existing coefficient row. [0.00001%]
Stage D	Reduce the leading non-zero entry of the new row to 1,such that the result is in REF. [0.38%]
Stage E	Reduce the coefficient matrix to the reduced row-echelon form (RREF). [49.5%]

Many schemes have been proposed on parallel network coding. In general, those schemes can be classified as two categories, i.e., progressive network coding and non-progressive network coding. H. Shojania and B. Li [11] first proposed a progressive decoding algorithm with Gauss-Jordan elimination, such that blocks can be decoded as soon as they are received. The process is shown in Table 1 in which Stage A has 50.5 percent and Stage E has 49.5 percent of decoding workload. It significantly improves the decoding performance but

the workload between threads is unbalanced. So Park [19] proposed dynamic partitioning algorithm to make load balanced and improve the performance of network coding. However, their dynamic partitioning method produced additional delay because each row received was partitioned at Stage A and Stage E with different methods. Deokho Kim [20] presented a new data manipulation method to utilize SIMD instruction sets which can be successfully integrated into the dynamic partitioning of thread-level workload distribution. Minwoo Kim [12] revealed the parallelized non-progressive network coding with block-wise Gauss-Jordan elimination and tilling algorithm but the process could not start decoding at the receiver until all n linear independent packets are received. This requires extra waiting time especially in a poor bandwidth network environment.



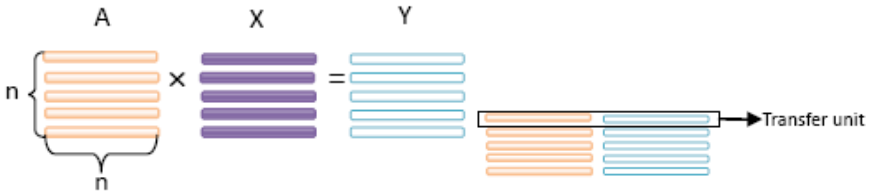
(a) (b)

Fig. 2. Decoding timeline of non-progressive and progressive schemes. (a) non-progressive decoding execution time. (b) progressive decoding execution time.

3. Semi-progressive Algorithm

In this section, we propose a semi-progressive network coding algorithm which can be used in multi-core processor. It can adapt to either poor network environment or good network environment.

3.1. Encoding and Decoding Model



(a) (b)

Fig. 3. (a) Encoding process. (b) Data at received node.

In the random linear network coding system [13], the data is encoded with random coefficient vectors. In the encoding procedure, the data with a fixed size is divided into *n* blocks. The size of each block is *m*. A linear combination of finite field operations with block x_i and a coefficient $a_{i,t}$ constitute the coded data y_i. The encoding procedure is represented in formula (1), where $a_{i,t}$ is *t-th* element of coefficient vector a_i and it is independently and randomly chosen in Galois Field (2^8).

$$y_i = \sum_{t=1}^{n} a_{it} x_t, i = 1, 2, ..., n. \quad (1)$$

Each coefficient vector a_i results in a different coded data y_i. Each of the *n* coded data has a fixed size *m* and each coefficient vector has *n* coefficients. At the receiving node *n* coded data and coefficient vectors constitute the coded data matrix Y. Let $A=[a_1, a_2, ..., a_n]^T$, $X=[x_1, x_2, ..., x_n]^T$, and $Y=[y_1, y_2, ..., y_n]^T$ where superscript T stands for the transpose operation. Then, the encoding process can be expressed as $Y = AX$ and we can obtain the original data *X* from the simple equation $X = A^{-1}Y$. Fig. 3 shows the encoding

process and the transfer unit which is composed by a vector and an encoded packet.

3.2. Semi-progressive Network Coding Algorithm

We assume k is the line number of the current row received and p is the amount of cores. Now, the decoding algorithm is shown as follows:

Algorithm 1. Decoding Algorithm.

Phase A

Step 1: Reduce the leading coefficients in the new coefficient row to 0.

Step 2: Count the number of consecutive 0 in the new row and regard it as s. If $s = k-1$, then goto Step 3; If $s \geq n$, the new row will be discarded; If $(k-1) < s < n$, then the new row should be adjusted to the $(s+1)-th$ line.

Step 3: Reduce the first non-zero element to 1.

Phase B

Step 4: Regard the whole reduced packets as the system of equations. Put each equation to cores one by one as "*S*" type.

The algorithm consists of two phases, *Phase A* and *Phase B*. Instead of waiting for all packets to arrive, decoding process can be performed when each transfer unit is received. Only when n linear independent packets have been executed at *Phase A*, will *Phase B* start.

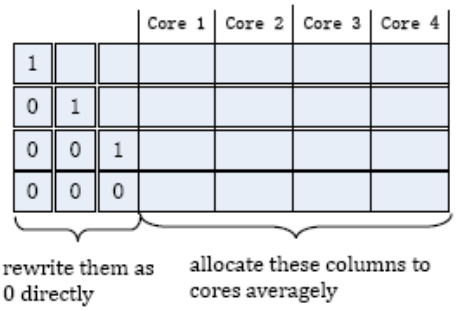


Fig. 4. MDP method.

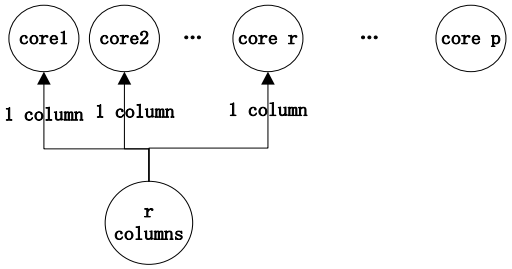


Fig. 5. Allocate the remaining r columns to cores.

3.3. The Design of *Phase A*

In Step 1, the elements $(a_{k,t}, 0 < t < k)$ before pivot element $a_{k,k}$ should be reduced to zero for each received packet. We propose the Modified Dynamic Partitioning (MDP) method based on the Dynamic Partitioning (DP) method discussed in [19]. As shown in Fig. 4 (k=4, p=4), the values before the pivot value can be rewritten as zero directly without any calculation needed. But the values before $a_{k,k}$ should be stored for the calculation of elements $(a_{k,t}, k \leq t < m+n)$ after the pivot element. The elements after pivot element will be partitioned and distributed to a multi-core processor by formula (2).

$$(m+n-k+1)/p = q...r. \tag{2}$$

where q is the average amount of columns which will be processed by one core and r represents the remaining number of columns. Instead of allocating the remaining parts to the last core, we assign and share the remaining parts from the first core to the $r\text{-}th$ core one by one. That is to say, from the first core to the $r\text{-}th$ core, each core will deal with $(q+1)$ columns. From the $(r+1)\text{-}th$ core to $p\text{-}th$ core, each core will deal with q columns. The gap between any two cores is one column. We will discuss the performance in Section 4. The process of dealing with remaining part r is shown in Fig. 5.

After Step 1, we will count the number of consecutive 0 in the new row to check whether the new row is linearly independent with the previously received packets and whether the new row should be adjusted to the right place. Based on the value of s, there are three conditions should be considered:

(1) $s = k-1$. It is the most ideal condition. There is no additional work to do.

(2) $s \geq n$. The new row should be discarded, because it is linear correlation with the packets which have been received previously.

(3) $(k-1) < s < n$. It means that the new row should be adjusted to $(s+1)-th$ line to reduce the calculation times and to make sure the coefficient matrix is a triangle matrix. For the next new rows received, from the $(k+1)-th$ row to $(s+1)-th$ row, each row should be swapped with the row before itself when it completes the Phase A. We will analyze the performance with concrete examples in Section 4. Fig. 6 illustrates the process of adjustment.

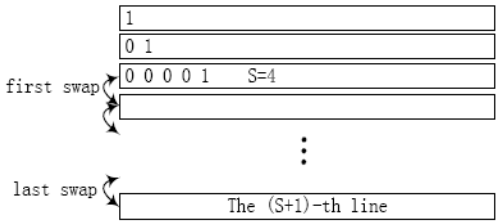
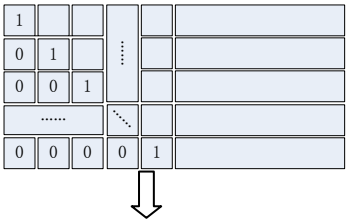


Fig. 6. The process of adjustment.

Step 3 is an easy step and we will not analyze the performance. It just reduces the first non-zero element to 1.

3.4. The Design of *Phase B*

After n linear independent packets have been received and executed in *Phase A*, *Phase B* can start. Now the coefficient matrix becomes the triangle matrix and the main diagonal elements are all 1. We regard each of the reduced transfer unit as an equation from equation (1) to equation (n) shown in Fig. 7.



$$\begin{cases} y_1 = a_{11} \times x_1 + a_{12} \times x_2 + \ldots + a_{1(n-1)} \times x_{n-1} + a_{1n} \times x_n & (1) \\ y_2 = \quad a_{22} \times x_2 + \ldots + a_{2(n-1)} \times x_{n-1} + a_{2n} \times x_n & (2) \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ y_{n-1} = \quad a_{(n-1)(n-1)} \times x_{n-1} + a_{(n-1)n} \times x_n & (n-1) \\ y_n = \quad a_{nn} \times x_n & (n) \end{cases}$$

Fig. 7. Transform the transfer units to system of linear equations.

"One transfer unit and one equation" is a "one-to-one" relationship. The solution of the system of linear equations is the original information we want to recover. Each equation will figure out one block. In other words, the equation (i) will work out block $x_i (i = 1, 2, \ldots, n)$. The first equation has n blocks from x_1 to x_n and the second equation has (n-1) blocks from x_2 to x_{n-1}, and so on. We will figure out block x_n by $x_n = y_n / a_{nn}$ first and then we will use back substitution by formula (3) to figure out the other blocks.

$$x_i = y_i - \sum_{j=i+1}^{n} a_{ij} x_j, i = n-1, \ldots, 2, 1. \quad (3)$$

Now, the key point is how to use multi-core processer to figure out the solution. In order to make full use of multi-core processor and balance the overhead of each core, we allocate the equations to cores like "S" type shown in Fig. 8. We call the new method as "S" Allocation (SA). The core $i(i=1,2,...,p)$ has equations whose numbers are

$$(2\times j-1)\times p-(p-i). \tag{4}$$

and

$$2\times j\times p-(i-1). \tag{5}$$

where j is a loop variable, $j\le\lceil n/2p\rceil$ and $j\in N$. We will prove that the computation overhead on all cores are balanced in section 4. When block x_n is figured out, the core which has equation (n) will store x_n and flushes it to a shared cache. Then every core will get x_n and start calculating the equations simultaneously to eliminate x_n. Block x_{n-1} will be figured out by equation (n-1) after all equations remove block x_n. We will get all blocks in the similar way. In other words, we will get the original information using this process.

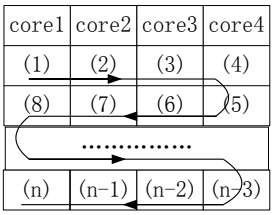


Fig. 8. Allocate the equations to cores like "S" type.

4. Performance Analysis of Semi-progressive Algorithm

In this section, we analyze the performance of our algorithm through theoretical analysis and mathematical calculation. We consider one addition and one multiplication as a unit operation.

4.1. Communication Analysis

For the simplicity, we assume each core just has one thread. At the same time, we assume one of the cores serve as the coordinating core, which synchronizes the task assignment and collection to and from other cores.

In *Phase A*, we use static scheduling. The coordinating core can divide the coded packet into partitions and assign each of them to a different core. Every core maintains the same coefficients $(a_{i,k}, 1\le i<k)$ and different columns in its local cache. Each core operates on its private copy of the coded block without any need to communicate with other cores. Such partitioning also improves cache performance. After the calculation, the result will be flushed to the shared cache. Ideally, all cores can start their new tasks at the same time and will finish around the same since they process equal amounts of data.

In *Phase B*, the tasks on different cores are not completely dependent. Equation (i) will produce the original packet x_i. The packet x_i should be saved in its local cache and flushed to the shared cache to calculate other equations. Then, other cores can get x_i from the shared cache and eliminate all x_i in all equations. At the same time, x_{i-1} will be obtained from the equation (i-1) and we will get all original packets in the same way. In this procedure, all cores can concurrently read data from the shared cache but at any time, only one core has write permission. Every core will communicate with shared cache at least n times because each core will deal with at least n/p equations which will save n/p packets into the shared

cache and get (*n-n/p*) packets from the shared cache.

4.2. Analysis of Load Balancing and Decoding Delay

There are two kinds of network: high speed and low speed. As mentioned previously, a progressive scheme using Gauss-Jordan elimination can hide part of the decoding time in poor environment. If the network speed is high, a node will receive a lot of rows in a short period of time and some of the new rows cannot be processed in time. On the other hand, a non-progressive scheme has better performance of calculating the result. In order to adjust to both conditions, we make a trade-off between the two methods. In a good network environment, waiting time is short but still exists. For the purpose of making full use of the waiting time, we just perform a portion of the progressive scheme — Gauss elimination when a new row is received. The time is about half of the Gauss-Jordan elimination, so we can enter *Phase B* as soon as possible to use a multi-core processor. On the other hand, when the network speed is low, the partial decoding time can be hidden in the waiting time. However, our method no longer has good performance if the transmission delay is too large, because there still exists a waiting time when the received row accomplishes *Phase A*.

Theorem 1. When a destination node receives the k-*th* row, the gap of calculation overheads between any two cores is less than or equal to $(k-1)$ unit operations in Step 1.

Proof. The k-*th* row is received and we allocate the values after the pivot value $(a_{k,k})$ to cores by formula (2). If r equals to zero, each core has the same workload and the gap is zero. If r is not equal to zero, each core in the front of the r-*th* core deals with $(q+1)$ values which need $(q+1)(k-1)$ unit operations and each of the remaining cores deals with q values which need $q(k-1)$ unit operations. So the gap between any two cores is $(k-1)$ unit operations. Above all, theorem is proved.

If remaining r values are just allocated to the last core as DP, the imbalance is larger than our method, because the last core will deal with $(q+r)$ values which need $(q+r)(k-1)$ unit operations. The largest gap between any two cores is $r(k-1)$ unit operations which are more than $(k-1)$ unit operations. So our scheme has better performance in balancing the workload.

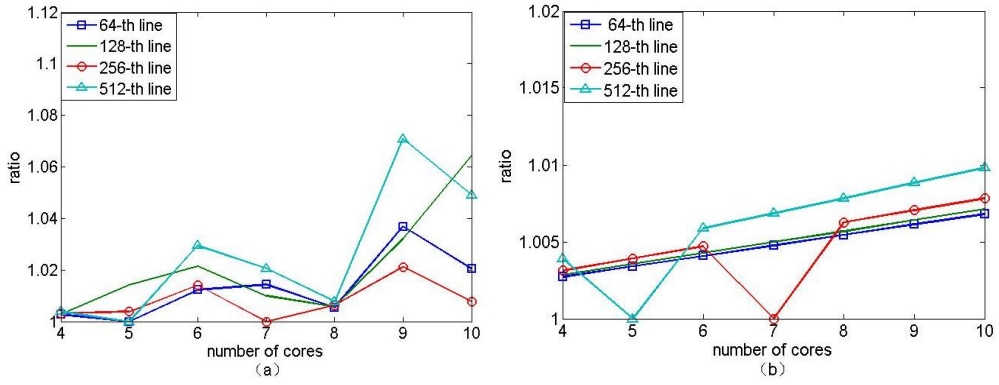


Fig. 9. The ratio of workloads between the first core and the last core (*m*=1024, *n*=512). (a) DP (b) MDP.

Fig. 9(a) and 9(b) show the ratio of workloads between the first core and the last core when receiving the k-*th* (k=64,128,256,512) row with *n*=512 and *m*=1024. The value of k could be chosen from 1 to 512. It shows that the gap between the core with maximum workload and the core with the minimum workload is very small when using MDP. It is obvious that fluctuation of MDP is more stable than DP. In other words, MDP has better performance in balancing workload than DP.

Next we analyze the performance of Step 2 with $k=3, s=4$ just like Fig. 6. The k-*th* row is not at the right place after executing Step 1 and it should be adjusted to the fifth row. For the next rows received, the

forth row and the fifth row, they will not be calculated with the third row any more so we can reduce some unnecessary calculations. When the forth row completes Phase A, it should be swapped with the third row. The fifth row will do the similar operation like the forth row. Then, the left of the matrix is an upper triangle matrix and it is the premise of rewriting values before $a_{k,k}$ to 0 directly in Step 1.

After completing Phase A, the coefficient matrix becomes the upper triangular matrix and the main diagonal elements are all 1. We allocate the equations to cores one by one as "*S*" type to pursue a balanced workload. The core $i(i=1,2,...,p)$ has equations whose numbers are (4) and (5) and j is the loop variables, $j \le \lceil n/2p \rceil$ and $j \in N$. Next we will calculate the unit operation times of each core in Phase B by

$$\sum_{j=1}^{n/2p} m \times n \times (2 \times n - 4 \times j \times p + 2 \times p - 1). \quad (6)$$

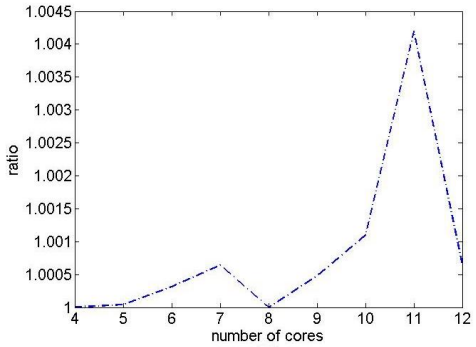


Fig. 10. The ratio of workloads on the cores which have maximum and minimum workloads (*n*=512, *m*=1024 and *p* is from 4 to 12).

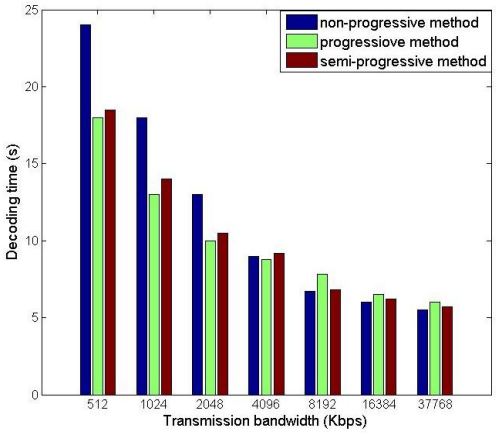


Fig. 11. Total transmission and decoding time versus transmission bandwidth.

Fig. 10 shows theoretical result that is the ratio of workloads on cores which have the maximum and minimum workloads when *n*=512, *m*=1024 in *Phase B*. The abscissa is the number of cores and the ordinate is the ration. We can see that when *n* is a multiple of p that is, $p=4,8$, all of the cores have the same overhead, because each core has the same number of blocks to figure out. On the other hand, *n* is not a multiple of p, the ratio is stable and very close to 1.

We execute our decoding operation on a real desktop system with an Intel Core 2 Quad-Core CPU and compare with progressive and non-progressive methods. Fig. 11 shows the decoding delay versus transmission bandwidth with *n*=512 and *m*=1024. We respectively regard low bandwidth and high bandwidth as the poor and good network environment. When network environment is poor, progressive

and semi-progressive methods have similar decoding time which is better than non-progressive method. Non-progressive method is getting better as transmission bandwidth increases. At some certain bandwidth, it is as good as our method which has lower decoding delay than progressive method.

5. Conclusion

In this paper, an efficient parallel algorithm used in multi-core processor for decoding process of random network coding has been introduced. The algorithm makes full use of multi-core processors to accelerate the decoding process and balances the workload between cores. In order to deal with the remaining columns efficiently, the Modified Dynamic Partitioning (MDP) is proposed. In addition, we also have proposed a task assignment method called "*S*" Allocation (*SA*) which makes the workload allocation more reasonable. Theoretically analysis has been conducted to support the scheme design. In the future work, we will implement our algorithm on the real multi-core network systems and investigate the impact of different parameters on the performance of the proposed scheme.

Acknowledgment

The work is supported by the National Science Foundation of China under Grant No. 61070170, No.61373164, Research Project of Jiangsu Province under Grant No.BY2013030-06.

References

[1] Ahlswede, R., Cai, N., Li, S. Y., & Yeung, R. W. (2000). Network information flow. *IEEE Transactions on Information Theory*, *46(4)*, 1204-1216.

[2] Wang, M., & Li, B. (2006). How practical is network coding? *Proceedings of 14th International Workshop on IEEE Quality of Service* (pp. 274-278).

[3] Maymounkov, P, Harvey, N, & Lun, D. (2006). Method for efficient network coding. *Proceedings of the Allerton Conference on Communication, Control, and Computing* (pp. 482-491). Monticello, IL.

[4] Li, Z., Li, B., Jiang, D., & Lau, L. C. (2005). On achieving optimal throughput with network coding. *Proceedings of 24th Annual Joint Conference of the IEEE Computer and Communications Societies: Vol. 3.* (pp. 2184-2194).

[5] Ghaderi, M., Towsley, D., & Kurose, J. (2008). Reliability gain of network coding in lossy wireless networks. *Proceedings of the 27th Conference on Computer Communications* (pp. 2171-2179).

[6] Cai, N., & Yeung, R. W. (2002). Secure network coding. *Proceedings of IEEE International Symposium on Information Theory*.

[7] Gkantsidis, C., Miller, J., & Rodriguez, P. (2006). Comprehensive view of a live network coding P2P system. *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement* (pp. 177-188).

[8] Gkantsidis, C., & Rodriguez, P. R. (2005). Network coding for large scale content distribution. *Proceedings* of *24th Annual Joint Conference of the IEEE Computer and Communications Societies*: *Vol. 4.* (pp. 2235-2245).

[9] Lee, U., Park, J. S., Yeh, J., Pau, G., & Gerla, M. (2006). Code torrent: Content distribution using network coding in VANET. *Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking* (pp. 1-5).

[10] Wang, N., & Ansari, N. (2011). Downloader-initiated random linear network coding for peer-to-peer file sharing. *Systems Journal*, *5(1)*, 61-69.

[11] Koetter, R., & Médard, M. (2003). An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, *11(5)*, 782-795.

[12] Kim, M., Park, K., & Ro, W. W. (2013). Benefits of using parallelized non-progressive network coding.

Journal of Network and Computer Applications, *36(1)*, 293-305.

[13] Chou, P. A., Yunnan W., & Kamal, J. (2003). Practical network coding. *Proceedings of Allerton Conference on Communication, Control and Computing*.

[14] Koetter, R., & Médard, M. (2003). An algebraic approach to network coding. *IEEE/ACM Transactions on Networking*, *11(5)*, 782-795.

[15] Tracey, H., *et al.* (2006). A random linear network coding approach to multicast. *IEEE Transactions on Information Theory*, *52(10)*, 4413-4430.

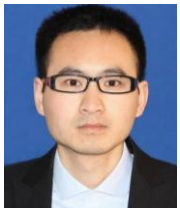
[16] Chen X., Chen S., Lu Z., *et al.* (2011). Hybrid distributed shared memory space in multi-core processors. *Journal of Software*, *6(12)*, 2369-2378.

[17] Li H, Yang Z, & He H. (2014). An improved image segmentation algorithm based on GPU parallel computing. *Journal of Software*, *9(8)*, 1985-1990.

[18] Zhou, Y, Zhu, Q, & Zhang, Y. (2011). Spatial data dynamic balancing distribution method based on the minimum spatial proximity for parallel spatial database. *Journal of Software*, *6(7)*, 1337-1344.

[19] Park, K., Park, J. S., & Ro, W. W. (2010). On improving parallelized network coding with dynamic partitioning. *IEEE Transactions on Parallel and Distributed Systems*, *21(11)*, 1547-1560.

[20] Kim, D., Park, K., & Ro, W. W. (2013). Exploiting SIMD parallelism on dynamically partitioned parallel network coding for P2P systems. *Computers & Electrical Engineering*, *39(1)*, 55-66.



Chang Zhu received the bachelor degree in network engineering from JiangSu Normal University, Xuzhou, China, in 2012. Currently, he is studying in Soochow University seeking for a master degree of computer science and technology. His research interest is network coding.

Jianguo Xu received his bachelor degree of engineering from East China Institute of Technology in 1998. He is working as a senior engineer in Geologic Party No.216, China National Nuclear Corporation, Urumqi, China. His areas of interest include parallel computing, software engineering, computer networks.

Yanqin Zhu received her Ph.D. degree from Soochow University in 2008. She is working as a professor in School of Computer Science & Technology, Soochow University, Suzhou, China. Her areas of interest include network communications, information security, and application of network coding.

Lingzhi Li received his Ph.D. degree from Nanjing University of Aeronautics and Astronautics in 2006. He is working as a lecturer in School of Computer Science & Technology, Soochow University, Suzhou, China. His areas of interest include sensor networks, network coding, parallel computing.