Formalization and derivation of DSP model

Weidong Zhang*, Chang Cui†,

I. Model and Convergence Guarantee

The objective of DSP model is to accelerate the convergence by slashing iteration rounds, consequently to boost the utilization of computation resources. Different from dense datasets, sparse datasets own much lower ratio of computation to communication: $T_{computation}/T_{communication}$. In other words, the time is mainly spent on communication. Therefore, reducing communication time could significantly accelerate the convergence of parallel algorithm.

Besides, we find that only one local computation alone for one BSP superstep could hardly fully explore and utilize the locality within data partition (also termed chunk or portion). In a large family of graph parallel algorithms especially, each local computation corresponds to one step of value propagation, thus more local computations prospectively accelerate the propagation of values. However, more applications of SCStep do not necessarily bring about better effects considering that more SCSteps will increase the computation overhead. What's worse, it is even possible to exacerbate load imbalance when initial task partitions are unbalanced.

Further research has shown that the result of SCStep has the following two attributes: (i) In terms of spatial locality, it can be locally optimized by fully exploring and utilizing the spatial locality within data partition. (ii) In terms of temporal locality, it can be locally optimized through trying to conduct more SCSteps within BSP superstep. To be brief, DSP could realize the temporal spatial local optimization in data partition within superstep. If the temporal spatial local optimization appears on sparse dataset or certain parallel algorithm, it is highly likely to convert to final and global optimization. But if it happens in some unsuitable situations, deviation will be brought in, convergence will be slowed down too.

Beyond that, DSP has some other advantages. For example, (i) When applied in a value propagation algorithm, DSP works for both dense and sparse datasets; (ii) When used to accelerate Jacobi method, DSP shows a similar accelerating effect as successive over-relaxation (SOR) [1].

A. Formal Representations

By inspecting the iteration processes of a large collection of parallel algorithms, we found that the new value of each component of input data is a result aggregated from all related components after working on it. Accordingly, we depict the input data as a vector, and sketch out the interactions between pairs of components as a relation matrix. Then the new values of each iteration can be obtained from a "multiplication of vector and matrix":

$$X_{k+1} = X_k \otimes F_{n\times m}.$$

Different from the mathematical multiplication, every element $F_{i,j}$ in relation matrix takes x_i and x_j as inputs, and outputs the impact of x_i on x_j for further aggregating into a new x_j. Similar representations also appear in the literature [2], [3]. The difference is that we further provide the iterative expressions of input variable changing with iteration steps.

To express in a more convenient and concise way, we define the following symbols and operators:

1) X_0: Input variable, vector-based representation as $(x_0, x_1, ..., x_n)$. x_i can express various types of data, such as the node information in graph, a variable in system of linear equations.
2) F: Relation matrix. The element $F_{i,j}$ describes the computation between X's components of x_i and x_j, $F_{i,j}(x_i, x_j)$, or $F_{i,j}(x_i)$ for short. The result of $F_{i,j}(x_i)$ returns to x_j for further aggregating. (In concrete implementation, $F_{i,j}$ can be expressed as a mathematical formula or a programming function/procedure/method.)
   $F^{(p,q)}$ presents a partial relation matrix. It's defined for distributed computation. $F^{(p,q)}$ only updates the segment from x_p to x_q of X, and leaves others unchanged. Its definition is shown as follow:

$$\begin{pmatrix}
1 & 0 & \dots & 0 & F_{0,p} & \dots & F_{0,q} & 0 & \dots & 0 \\
0 & 1 & \dots & 0 & F_{1,p} & \dots & F_{1,q} & 0 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \dots & 1 & F_{p-1,p} & \dots & F_{p-1,q} & 0 & \dots & 0 \\
0 & 0 & \dots & 0 & F_{p,p} & \dots & F_{p,q} & 0 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \dots & 0 & F_{q,p} & \dots & F_{q,q} & 0 & \dots & 0 \\
0 & 0 & \dots & 0 & F_{q+1,p} & \dots & F_{q+1,q} & 1 & \dots & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \dots & 0 & F_{n,p} & \dots & F_{n,q} & 0 & \dots & 1
\end{pmatrix}$$

3) X_k: Output of the k-th round iteration. $X_k^{(p,q)}$ indicates the output of partial transformation on segment from x_p to x_q of X_k.
4) $\biguplus$: Aggregate operator. The operator aggregates all the results of $\{F_{j,i}(x_j)|j = 0, 1, 2, \dots, n.\}$ into a new value and assigns to x_i as the new value of next

moment:

$$\begin{aligned} x_i' &= (x_0, x_1, \ldots, x_n) \cdot (F_{i,0}, F_{i,1}, \ldots, F_{i,n})^T \\ &= \biguplus [F_{i,0}(x_i, x_0), F_{i,1}(x_i, x_1), \ldots, F_{i,n}(x_i, x_n)] \\ &= \biguplus_{j=0}^{n} F_{j,i}(x_j, x_i), \ \textit{abbreviated as} \ \biguplus_{j=0}^{n} F_{j,i}(x_j) \end{aligned}$$

The common aggregate operators are min(), max(), average(), $\sum$, Π, etc.

5) $\otimes$: Transformation operator. The operator applies the relation matrix F on input variable X once.

Using the above symbols and operations, one round of BSP iteration could be expressed as following:

$$\begin{aligned} X_{k+1} &= X_k \otimes F \\ &= (x_0, x_1, x_2, \ldots, x_n) \otimes \begin{pmatrix} F_{0,0} & F_{0,1} & \ldots & F_{0,m} \\ F_{1,0} & F_{1,1} & \ldots & F_{1,m} \\ & & \ldots & \\ F_{n,0} & F_{n,1} & \ldots & F_{n,m} \end{pmatrix} \\ &= (\biguplus_{i=0}^{n} F_{i,0}(x_i), \ \biguplus_{i=0}^{n} F_{i,1}(x_i), \ \ldots, \ \biguplus_{i=0}^{n} F_{i,m}(x_i)) \end{aligned}$$

k rounds of BSP iterations can be expressed as following: (*The detailed deducing procedures can be found in Appendix*[①].)

$$X_k = (h^k(X_0), h^k(X_0), \ \ldots, h^k(X_0)) \tag{4.1}$$

in wich $h(X) = \biguplus_{j=0}^{n} F_{j,i}(x_j)$ means one round of vector multiplication plus one aggregation operation:

$$x_i = \biguplus_{j=0}^{n} ((x_0, x_1, x_2, \ldots, x_n) \times (F_{0,i}, F_{1,i}, F_{2,i}, \ldots, F_{n,i})^T),$$

also can be interpreted as one round of BSP superstep operation on single variable component $x_i, i = 0, 1, \ldots, n$.

Through the deduction of (4.1), we find that k rounds global synchronous computation could be achieved by k times transformation of relation matrix.

Analogously, by conducting l rounds of DSP superstep on X_0, we get the corresponding $X_{l\Delta}$:

$$X_{l\Delta} = (h^l(g^{\Delta-1}(\alpha_0, \ \beta_0)), \ \ldots, \ h^l(g^{\Delta-1}(\alpha_n, \ \beta_n))) \tag{4.2}$$

in which each DSP superstep contains additional $(\Delta - 1)$ steps of local computation: $g(\alpha_p, \beta_p) = \biguplus(\biguplus_{i \in (p,q)} F_{i,j}(\alpha_p), \ \beta_p)$, whose input consists of two parts: $\alpha_p = \biguplus_{i=0}^{n} F_{i,p}(x_i)$ and $\beta_p = \biguplus_{i \notin (p,q)} F_{i,p}(x_i)$ represent the aggregated result from all the relied variables and the aggregated result from the relied variables that accommodated on different processors respectively, p=0,1,2,...,n. The detailed deducing procedures can be found in Appendix.

Intuitively, the parameters can be interpreted as:

- $g(\alpha_p, \beta_p)$: The variant of original BSP algorithm by increasing local computation from once to twice.
- α_p: The result aggregated from all variables that x_p relies on.
- β_p: The result aggregated from the variables that x_p relies on and at the same time they are accommodated on different processors than x_p's.
- $\gamma_p (= \alpha_p - \beta_p)$: The result aggregated from the variables that x_p relies on and at the same time they are accommodated on the same processor as x_p's.

If we want to use DSP to accelerate BSP, it is to say using formula (4.2) to replace formula (4.1), we first need to prove they could converge to the same state. Unfortunately, formula (4.2) can't guarantee the same convergence state as (4.1). But for the convex optimization problems and local-optimal insensitive problems, the convergence is sufficient. In the following subsection, we will use the mathematical induction to deduce the sufficient condition of convergence for DSP.

B. Convergence Guarantee of DSP Model

Besides PS and DSP, there has been plenty of parallel methods [4], [5], [6], [7] which are based on the idea of speculative computation. However, many of them lack theoretical proof of correctness, applicability and convergence condition. As a result, developers remain cautious to adopt them.

In this section, we will infer the relation between BSP and DSP models and give the convergence condition of DSP model.

Theorem 1. *If algorithm could converge on BSP model, then it converges on DSP model if and only if it meets condition:*

$$Algorithm \ g(\alpha_p, \beta_p) \ converges \ under \ BSP \ model. \tag{4.3}$$

Intuitively, the theorem tells that if the algorithm converges when $\Delta = 2$, then it converges for any $\Delta \geq 1$.

Proof. Base case $\Delta = 1$: If $\Delta = 1$, the DSP model degrades to the BSP model. So, the theorem holds when $\Delta = 1$.

Inductive hypothesis: Suppose the theorem holds for all values of Δ up to some k, $k \geq 1$, it is saying that

$$h^l(g^{k-1}(\alpha_p, \ \beta_p)) \tag{a}$$

could converge.

Inductive step: Let $\Delta = k + 1$. Then (a) becomes

$$h^l(g^k(\alpha_p, \ \beta_p)) = h^l(g^{k-1}(g(\alpha_p, \ \beta_p), \beta_p)) \tag{b}$$

Compare (b) with (a), (a) converges because α_p is a convergent operation under BSP model. So the convergence condition for (b) is:

$$g(\alpha_p, \beta_p) \ converges \ under \ BSP \ model. \tag{4.3}$$

With the condition (4.3), the theorem holds for $\Delta = k+1$. By the principle of mathematical induction, the theorem holds for all $\Delta \in \mathbb{N}$ under condition (4.3). □

[①]https://github.com/wdfnst/DSP_Proof/blob/master/dsp_proof.pdf

To ensure the adaptiveness in various circumstances, the relation and aggregate functions we discussed in this section all are general ones. In terms of the specific algorithm, the convergence of DSP algorithm can be proven by the stationary point analysis and convergence series methods when the relation/aggregate functions are given.

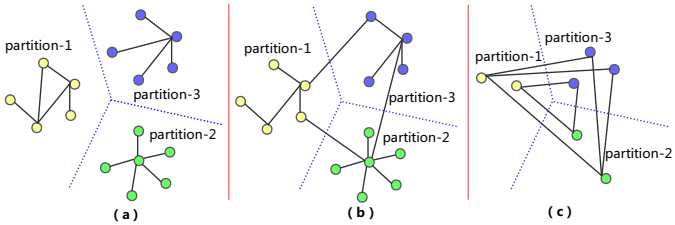

Fig. 1: (a) $\beta_p = 0$: Every variable only depends on the variables residing on the same processor; (b) $\beta_p > 0$: Some variables depend on other variables from both local and remote processors; (c) $\gamma_p = 0$: Every variable only depends on the variables residing on the different processors.

C. Factors of Acceleration of DSP

Through formula

$$g^{\Delta-1}(\alpha_p, \beta_p),$$

we find that the convergence speed of DSP algorithm mainly depends on: Δ, α_p and β_p, where p = 0, 1, 2, ..., n. Furthermore, we conclude the following relationships:

- When $\beta_p = 0$ (To be specific, x_p doesn't depend on the variables which reside on different processors.) In this case, x_p could converge without global data exchange. The additional $\Delta - 1$ steps computation could yield significant acceleration, and even when Δ is big enough, x_p could converge without global synchronization. Figure 1a demonstrates this case, every partition is allocated to a different processor with no dependencies across them. The iteration algorithm will converge without global synchronization.
- When $\gamma_p = 0$ (To be specific, x_p doesn't depend on the variables which reside on the same processor as x_p.) In this case, DSP has no effect on acceleration. Because all the dependent variables of x_p have expired after the first computation. As a result, the additional $\Delta-1$ steps computation gets the same results. Figure 1c illustrates this case. All the dependent variables for every one reside on remote processors.
- When $\gamma_p > 0$ (To be specific, some dependent variables of x_p reside on the same processor as x_p.) In this case, the new updated value of dependent variable will promote the convergence of x_p during the $\Delta - 1$ steps computation. Further, we hold that the acceleration is proportional to γ_p.
- When $\beta_p > 0$ (To be specific, some dependent variables of x_p reside on other computation nodes.) In this case, the expired β_p performs side-effect to x_p's convergence in the additional $\Delta - 1$ steps local computation. Further, we hold that the acceleration is inversely proportional to β_p. Figure 1b shows this case, some variables depend on others from both local and remote processors.

In some ways, γ_p and β_p can be interpreted as the distributions of dependency relationship among variables of inner-chunk and inter-chunk respectively. So we could increase the relationship density of inner-chunk and reduce the relationship density of inter-chunk by employing proper data partition methods, which equals to increase γ_p and reduce β_p in turn. Unfortunately, perfect data partitioning is hard to achieve, for example, graph partitioning problem is a classical NP hard problem. Although it is hard to increase γ_p, we can get relatively small β_p owing to the fact that partitioning on sparse data usually generates small β_p.

To verify the relationship among β_p, γ_p and acceleration, we conduct two groups of tests with PageRank on graphs: (i) fixed inner connections, varying inter-chunk connections; (ii)fixed inter-chunk connections, varying inner-chunk connections.

As is shown in Figure 2, subplot (a) shows that the rate #Iteration$_{bsp}$/#Iteration$_{dsp}$ decreases with increasing density of inter-chunk; subplot (b) shows that the rate #Iteration$_{bsp}$/#Iteration$_{dsp}$ increases with increasing density of inner-chunk. The experimental results also accord with above analysis.

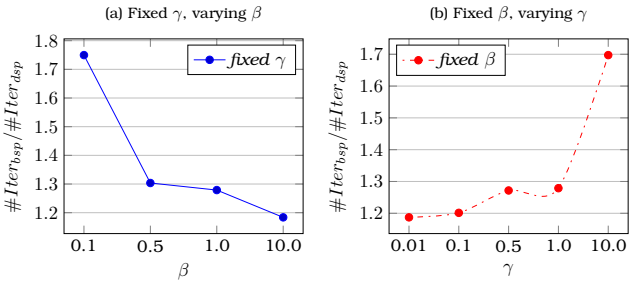

Fig. 2: Acceleration of DSP on PageRank. (a) Fixed γ with varying β; (b) Fixed β with varying γ.

D. DSP Parallel Algorithm

Programs using DSP model can be constructed as Algorithm 2. Compared with BSP in Algorithm 1, the difference is the quantity check of local computations. Accordingly, the transplantation from BSP algorithm to DSP algorithm can be conducted in the following steps:

1) If BSP algorithm converges, check whether the algorithm variant with one more SCStep is convergence; if the algorithm variation converges, go to step 2);
2) Screen proper Δ;
3) Adjust the code to make each DataExchange() correspond with Δ Computing().

Algorithm 1 BSP Parallel Algorithm

```
1: procedure BSP_ALGO(G, v_0)
2:     iter_count← 0
3:     while True do
4:         Computing()
5:         DataExchange()
6:         if is_convergent() then
7:             break
8:         iter_count++
```

Algorithm 2 DSP Parallel Algorithm

```
1: procedure DSP_ALGO(G, v_0)
2:     iter_count← 0
3:     while True do
4:         Computing()
5:         if iter_count % Delta == 0 then
6:             DataExchange()
7:             if is_convergent() then
8:                 break
9:         iter_count++
```

E. The selection of parameter Δ

There is no universally applicable standard for selecting proper Δ yet. According to the analysis in section I-C, a reasonably large Δ can be applied when datasets are sparse enough or good partition algorithm is adopted, otherwise properly small Δ is preferred.

In addition, the algorithms with large nonlinear convergence processes usually need higher synchronization frequency. Because large nolinear makes variables accommodated in different processors change dramatically, the reuse of the expired and seriously deviated data will introduce large deviation to slow the convergence down. Accordingly, it is wise to adopt a reasonably large Δ when algorithm owns linear convergence process, otherwise a reasonably small Δ is preferred.

REFERENCES

[1] A. Hadjidimos, "Successive overrelaxation (sor) and related methods," *Journal of Computational & Applied Mathematics*, vol. 123, no. 1-2, pp. 177–199, 2000.

[2] J. Feldman, S. Muthukrishnan, A. Sidiropoulos, C. Stein, and Z. Svitkina, "On distributing symmetric streaming computations.," *Acm Transactions on Algorithms*, vol. 6, no. 4, p. 66, 2010.

[3] H. Yin, R. Lee, S. Zhang, C. H. Xia, and X. Zhang, "Dot:a matrix model for analyzing, optimizing and deploying software for big data analytics in distributed systems," in *ACM Symposium on Cloud Computing*, pp. 1–14, 2011.

[4] P. Prabhu, G. Ramalingam, and K. Vaswani, "Safe programmable speculative parallelism," in *ACM Sigplan Conference on Programming Language Design and Implementation*, pp. 50–61, 2010.

[5] F. W. Burton, "Speculative computation, parallelism, and functional programming," *Computers IEEE Transactions on*, vol. c-34, no. 12, pp. 1190–1193, 1985.

[6] A. Sohn, "Parallel speculative computation of simulated annealing," in *International Conference on Parallel Processing, 1994. ICPP*, pp. 8–11, 1994.

[7] A. Sohn, "Parallel n-ary speculative computation of simulated annealing," *IEEE Transactions on Parallel & Distributed Systems*, vol. 6, no. 10, pp. 997–1005, 1995.

APPENDIX

A. The iteration process of BSP

Using the operations defined in I-A, we could deduce the BSP iteration process as following:

$$
\begin{aligned}
X_0 &= (x_0, x_1, \ldots, x_n) \\
X_1 &= X_0 \otimes F \\
&= (x_0, x_1, x_2, \ldots, x_n) \otimes \begin{pmatrix} F_{0,0} & F_{0,1} & \ldots & F_{0,m} \\ F_{1,0} & F_{1,1} & \ldots & F_{1,m} \\ & & \ldots & \\ F_{n,0} & F_{n,1} & \ldots & F_{n,m} \end{pmatrix} \\
&= (\biguplus_{i=0}^{n} F_{i,0}(x_i),\ \biguplus_{i=0}^{n} F_{i,1}(x_i),\ \ldots,\ \biguplus_{i=0}^{n} F_{i,m}(x_i))
\end{aligned}
$$

Let $h(X) = \biguplus_{j=0}^{n} F_{j,i}(x_j)$, which means one round of vector multiplication plus one aggregation operation then,

$$
\begin{aligned}
&= (h(X_0), h(X_0),\ \ldots, h(X_0)) \\
X_2 &= X_1 \otimes F \\
&= (h(X_1), h(X_1),\ \ldots, h(X_1)) \\
&= (h(h(X_0)), h(h((X_1)),\ \ldots, h(h(X_1))) \\
&= (h^2(X_0), h^2(X_0),\ \ldots, h^2(X_0)) \\
X_3 &= (h(X_2), h(X_2),\ \ldots, h(X_2)) \\
&= (h(h(X_1)), h(h(X_1)),\ \ldots, h(h(X_1))) \\
&= (h(h(h(X_0))), h(h((h(X_1))),\ \ldots, h(h(h(X_1)))) \\
&= (h^3(X_0), h^3(X_0),\ \ldots, h^3(X_0)) \\
&\vdots \\
X_k &= (h^k(X_0), h^k(X_0),\ \ldots, h^k(X_0)) \qquad (4.1)
\end{aligned}
$$

B. The iteration process of DSP

Using the same deducing method as BSP, we get the formalized representation of DSP iteration process shown in the following steps:

(1) In each iteration, every processor conducts Δ steps of local computation and local data update. To ensure its conciseness and generality, we just show the transformation on the processor whose segment is from x_p to x_q, and the derivation is shown in Table IV.

(2) After Δ steps of local computation and local data update, it follows a global data synchronization. After that, another superstep restarts again. The process could be described as following:

$$
\begin{aligned}
X_\Delta &= (x_{\Delta,0},\ x_{\Delta,1},\ x_{\Delta,2},\ \ldots,\ x_{\Delta,n}) \\
X_{\Delta+1}^{(p,q)} &= X_\Delta \otimes F^{(p,q)} \\
&= (\ldots,\ x_{\Delta,p-1},\ \biguplus_{i=0}^{n} F_{i,p}(x_{\Delta,p}),\ \ldots, \\
&\qquad \biguplus_{i=0}^{n} F_{i,q}(x_{\Delta,q}),\ x_{\Delta,q+1},\ \ldots,\ x_{\Delta,n}) \\
&\qquad \vdots
\end{aligned}
$$

$$X_{t0} = (x_{t0,0}, x_{t0,1}, \ldots, x_{t0,n})$$

$$X_{t1}^{(p,q)} = X_{t0} \otimes F^{(p,q)} = (x_{t0,0}, x_{t0,1}, \ldots, x_{t0,n}) \otimes \begin{pmatrix} 1 & 0 & \ldots & 0 & F_{0,p} & \ldots & F_{0,q} & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 & F_{1,p} & \ldots & F_{1,q} & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & F_{p-1,p} & \ldots & F_{p-1,q} & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & F_{p,p} & \ldots & F_{p,q} & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & F_{q,p} & \ldots & F_{q,q} & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & F_{q+1,p} & \ldots & F_{q+1,q} & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & F_{n,p} & \ldots & F_{n,q} & 0 & \ldots & 1 \end{pmatrix}$$

Using the "matrix multiplication" − like transformation $\otimes$*, we get the follows, only* x_p *to* x_q *are updated.*

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, \biguplus_{i=0}^{n} F_{i,p}(x_{t0,i}), \biguplus_{i=0}^{n} F_{i,p+1}(x_{t0,i}), \ldots, \biguplus_{i=0}^{n} F_{i,q}(x_{t0,i}), x_{t0,q+1}, \ldots, x_{t0,n})$$

$$X_{t2}^{(p,q)} = X_{t1}^{(p,q)} \otimes F^{(p,q)}$$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, \biguplus_{i=0}^{n} F_{i,p}(x_{t1,i}), \biguplus_{i=0}^{n} F_{i,p+1}(x_{t1,i}), \ldots, \biguplus_{i=0}^{n} F_{i,q}(x_{t1,i}), x_{t0,q+1}, \ldots, x_{t0,n})$$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, \biguplus(\biguplus_{i\in(p,q)} F_{j,p}(\biguplus_{j=0}^{n} F_{j,p}(x_{t0,j})), \biguplus_{j\notin(p,q)} F_{j,p}(x_{t0,j})),$$

$$\ldots, \biguplus(\biguplus_{i\in(p,q)} F_{i,q}(\biguplus_{j=0}^{n} F_{j,q}(x_{t0,j})), \biguplus_{j\notin(p,q)} F_{j,q}(x_{t0,j})), x_{t0,q+1}, \ldots, x_{t0,n})$$

Let $\alpha_p = \biguplus_{j=0}^{n} F_{j,p}(x_{t0,j})$, $\beta_p = \biguplus_{j\notin(p,q)} F_{j,p}(x_{t0,j})$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, \biguplus(\biguplus_{i\in(p,q)} F_{i,p}(\alpha_p), \beta_p), \ldots, \biguplus(\biguplus_{i\in(p,q)} F_{i,q}(\alpha_q), \beta_q), x_{t0,q+1}, \ldots, x_{t0,n})$$

Let $g(x,y) = \biguplus(\biguplus_{i\in(p,q)} F_{i,p}(x), y)$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, g(\alpha_p, \beta_p), \ldots, g(\alpha_q, \beta_q), x_{t0,q+1}, \ldots, x_{t0,n})$$

$$X_{t3}^{(p,q)} = X_{t2}^{(p,q)} \otimes F^{(p,q)}$$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, \biguplus(\biguplus_{i\in(p,q)} F_{i,p}(X_{t2}), \beta_p), \ldots, \biguplus(\biguplus_{i\in(p,q)} F_{i,q}(X_{t2}), \beta_q), x_{t0,q+1}, \ldots, x_{t0,n})$$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, \biguplus(\biguplus_{i\in(p,q)} F_{i,p}(g(\alpha_p, \beta_p)), \beta_p), \ldots, \biguplus(\biguplus_{i\in(p,q)} F_{i,q}(g(\alpha_q, \beta_q)), \beta_q), x_{t0,q+1}, x_{t0,q+1}, \ldots, x_{t0,n})$$

$$\vdots$$

$$X_{\Delta}^{(p,q)} = X_{\Delta-1}^{(p,q)} \otimes F^{(p,q)}$$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, g(g(\ldots g(\alpha_p, \beta_p), \ldots, \beta_p), \beta_p), \ldots, g(g(\ldots g(\alpha_q, \beta_q), \ldots, \beta_q), \beta_q), x_{t0,q+1}, \ldots, x_{t0,n})$$

$$= (x_{t0,0}, \ldots, x_{t0,p-1}, g^{\Delta-1}(\alpha_p, \beta_p), \ldots, g^{\Delta-1}(\alpha_q, \beta_q), x_{t0,q+1}, \ldots, x_{t0,n})$$

TABLE I: Derivation of First Δ Steps of Local Computation and Data Exchange.

$$X_{2\Delta}^{(p,q)} = (\ldots, x_{\Delta,p-1}, g^{\Delta-1}(\underbrace{\alpha_{2\Delta,p}}, \underbrace{\beta_{2\Delta,p}}), \ldots,$$

$$\boxed{\biguplus_{i=0}^{n} F_{i,p}(x_{\Delta,p})} \quad \boxed{\biguplus_{i\notin(p,q)} F_{i,p}(x_{\Delta,p})}$$

$$g^{\Delta-1}(\alpha_{2\Delta,q}, \beta_{2\Delta,q}), x_{\Delta,q+1}, \ldots, x_{\Delta,n}),$$

in which each item can be derived as follows :

$$x_{2\Delta,p} = g^{\Delta-1}(\biguplus_{i=0}^{n} F_{i,p}(\underbrace{x_{\Delta,p}}), \biguplus_{i\notin(p,q)} F_{i,p}(\underbrace{x_{\Delta,p}})),$$

$$\boxed{x_{\Delta,p} = g^{\Delta-1}(\alpha_p, \beta_p)}$$

$$= g^{\Delta-1}(\biguplus_{i=0}^{n} F_{i,p}\left(g^{\Delta-1}(\alpha,\ \beta)\right), \biguplus_{i\notin(p,q)} F_{i,p}\left(g^{\Delta-1}(\alpha,\ \beta)\right))$$

(3) Through the deduction of (1) and (2), we find that the output of current bulk round will be used as the input of next bulk round. Thus one round of iteration process

can be expresssed as:

$$h(g(x,y)),$$

in which $g(x,y) = \biguplus_{i=0}^{n}(\biguplus_{i\in(p,q)} F_{i,j}(x),\ y)$.

(4) Through l rounds bulk synchronization, we could get $X_{l\Delta}$:

$$X_{l\Delta} = (x_{l\Delta,0},\ x_{l\Delta,1},\ \ldots,\ x_{l\Delta,m}),$$

in which each item can be presented as

$$x_{l\Delta,p} = h^l(x_{\Delta,p}), p = 0, 1, \ldots, m$$

in which

$$x_{\Delta,p} = g^{\Delta-1}(\alpha_p,\ \beta_p)$$

in which $\alpha_p = \biguplus_{i=0}^{n} F_{i,p}(x_{t0,i})$ and $\beta_p = \biguplus_{i\notin(p,q)} F_{i,p}(x_{t0,i})$, finally, we can get

$$x_{l\Delta,p} = h^l(g^{\Delta-1}(\alpha_p,\ \beta_p)). \tag{4.2}$$