D-iteration based asynchronous distributed computation

Dohy Hong
Alcatel-Lucent Bell Labs
Route de Villejust
91620 Nozay, France
dohy.hong@alcatel-lucent.com

ABSTRACT
The aim of this paper is to explain how the D-iteration can be used for an efficient asynchronous distributed computation. We present the main ideas of the method and illustrate them through very simple examples.

Categories and Subject Descriptors
G.1.0 [Mathematics of Computing]: Numerical Analysis—Parallel algorithms; G.1.3 [Mathematics of Computing]: Numerical Analysis—Numerical Linear Algebra

General Terms
Algorithms, Performance

Keywords
Distributed computation, Iteration, Fixed point, Eigenvector.

1. INTRODUCTION
As an improved or alternative solution to existing iterative methods (cf. [2] [3] [1]), the D-iteration algorithm has been proposed in [3] in a general context of linear equations to solve $X$ (vector of size $N$) such that:

$$X = PX + B.$$  

where $P$ is a square matrix of size $N \times N$ and $B$ a vector of size $N$. In particular, it has been shown how this iterative method can be further applied to solve $X$ such that

$$QX = X \quad \text{and} \quad RX = B$$

where $Q$ and $R$ are square matrices of size $N \times N$ or to solve

$$AX = B$$

where $A$ is a square matrix of size $N \times N$.

We recall the definition of the two vectors used in D-iteration: the fluid vector $F_n$ defined by:

$$F_n = (I_d - J_{in} + PJ_{in})F_{n-1}. \quad \text{(2)}$$

where:

- $I_d$ is the identity matrix;
- $J_k$ a matrix with all entries equal to $1$ except for the $k$-th diagonal term: $(J_k)_{kk} = 1$.

And the history vector $H_n$ defined by ($H_0$ initialized to a null vector):

$$H_n = \sum_{k=1}^n J_k F_{k-1}. \quad \text{(3)}$$

Then, we have (cf. [3]):

$$H_n + F_n = F_0 + PH_n. \quad \text{(4)}$$

It has been shown in [3] that $H_n$ satisfies the equation:

$$H_n = (I_d - J_{in}(I_d - P))H_{n-1} + J_{in}F_0. \quad \text{(5)}$$

In fact, the above equation can be very easily understood remarking that $I_d - J_{in}(I_d - P)$ is a matrix built from $P$ extracting the $i_n$-th line of $P$ and completing the rest with identity line vectors on $i \neq i_n$ (zero everywhere except the $i$-th column equal to one).

Note that for the entry $i \neq i_n$, $(H_n)_i = (H_{n-1})_i$.

2. Preliminary operations

2.1 Initial condition
It is easy to see from the equation [3] that when we choose $i_1 = 1, i_2 = 2, \ldots, i_N = N$, we obtain $H_N = B$. So we can directly start the iteration with $H_0 = B$ without any cost.

2.1.2 Diagonal link elimination
Now we can optionally apply the diagonal link elimination based on the method defined in [3]: when $p_{ii} \neq 0$ is to be suppressed, it implies two modifications:

- modification of the initial fluid: replace $B_i$ by $B_i/(1 - p_{ii})$;
• modification of all link weights pointing to node \( i \) (incoming links to \( i \), namely all \( j \) such that \( p_{ij} \neq 0 \): this operation can be replaced by keeping locally at node \( i \) the information that all incoming fluid need to be multiplied by \( 1/(1 - p_{ii}) \).

3. DISTRIBUTIVE COMPUTATION

In the following we set \( L_i(P) \) the \( i \)-th line vector extracted from \( P \):

\[
(L_i(P))_j = p_{ij}.
\]

We start by assuming that there is a partition of \( N \) in \( K \) disjoint sets \( \Omega_i \), \( i = 1, \ldots, K \), such that \( \bigcup_{i=1}^K \Omega_i = \{1, \ldots, N\} \).

The choice of the partition can be seen as an independent optimization task that will not be discussed here (intuitively, \( \Omega_k \) should be such that most of links are between nodes of the same set).

3.1 Operations in \( \Omega_k \)

We assume here that all computations of \( (H_n)_i, i \in \Omega_k \) is handled by one independent process (or server or virtual machine), that we call \( PID_k \).

\( PID_k \) has as input \( B \) and \( H \). \( H \) is initially set to \( B \).

3.1.1 Local updates

\( PID_k \) updates \( H \) by applying the fluid diffusion model with \( i \in \Omega_k \):

\[
(H)_i = L_i(P).H + (B)_i.
\]

3.1.2 Updates sharing

Periodically, \( PID_k \) sends to all other \( PID_i \) \((i \neq k)\) the updated \( (H)_i \), \( i \in \Omega_k \). When, a \( PID_k \) receives updates of \( (H)_i \), for \( i \in \Omega_{i'}, \) it updates the current \( H \) and can apply the local updates of \( \Omega_k \).

3.2 Evolution of \( P \)

If for some reason, the matrix \( P \) is updated to a new matrix \( P' \) and if one is interested by the solution of \( H \) with \( P' \), the new \( P' \) is sent to all \( PID_k \) that are concerned by the modification.

Upon reception of this modification, each \( PID_k \) does the following updates:

- store the last result \( H \) for entries \( i \in \Omega_k \) (can be used as the new initial vector \( H_0 \));
- replace \( B \) by \( B' = F + (P' - P)H \) for entries \( i \in \Omega_k \).

\( F \) is computed by: \( L_i(P).H + (B)_i - (H)_i \).

Since each \( PID_k \) only requires the information \( (B)_i \) for \( i \in \Omega_k \), we don’t need to synchronize for the new \( B' \) but just update \( B' \) locally and then we can re-apply the methods of Section 3.1 with \( P' \).

The above result is based on the result of Theorem 4 of [4].

3.3 Another version based on two state vectors (V2)

The drawback of the above method is to have to keep the complete \( H \) vector for each \( PID \). For a really very large matrix \( P \) this may be an issue. In such a case, we may use the two fluid diffusion state vectors \( H_n \) and \( F_n \) (equations 3 and 5). Then each \( PID_k \) needs to keep only locally the partial view: \( (B)_i, (H_n)_i \), and \( (F_n)_i \) for \( i \in \Omega_k \).

In such a scheme, the exchanged information between \( PID \) is the quantity \( F_n \) that need to be sent/received: each \( PID_k \) exploits the column vector extracted from \( P \), say \( C_i(P) \) for the \( i \)-th column vector \((i \in \Omega_k)\). When the diffusion is applied on node \( i \in \Omega_k \) with the fluid \( f = (F_{n-1})_i \), the quantity \( f \times p_{i,j} \) need to be sent to a \( PID_{k'} \) such that \( j \in \Omega_{k'} \), so that \( PID_{k'} \) can add this quantity to \((F_{n'})_j \).

The fluid transmission \((f \times p_{i,j})\) to all \( j \) does not require any synchronization. To avoid too much information exchange, the fluid transmission can be delayed and regrouped (we can regroup \((f_1 + f_2 + .. + f_m) \times p_{j,i} \) so that this quantity is not too small; we can regroup on \( i \) as well if going to the same destination \( j \): in fact, we don’t need to know who sent the fluid. The only constraint is that the fluid transmission is not lost: this means that each \( PID_k \) need to keep locally the information of the fluid \((f_1 + f_2 + .. + f_m) \times p_{j,i} \) until its destination \( PID \) (\( PID_{k'} \)) acknowledges its reception (say as TCP).

In this scheme, the convergence is explicitly monitored by observing the total fluid quantity (locally updated \( F_n \) plus all fluids being transmitted).

4. OPTIMIZATION PROBLEM

Given the partition set \( \Omega_k \), the question is when to share the local updates on \( H \). Here is a first possible solution.

4.1 Local remaining fluid

We can define the local remaining fluid \( r_k \) by:

\[
r_k = \sum_{i \in \Omega_k} |L_i(P).H + (B)_i - (H)_i|.
\]

Assuming a non-negative matrix \( P \) and applying ideas of [4], we could decide to share the results of the local computations to other \( PID \) when

\[
r_k < T_k
\]

where \( T_k \) is the local threshold for \( \Omega_k \). When such a condition is satisfied, we could then apply an update of \( T_k \). For instance by a multiplicative division by factor \( \alpha > 1 \):

\[
T_k := T_k/\alpha.
\]

In the version (V2), \( r_k \) is explicitly given by the norm \( L_1 \) of \( F_n \): \( r_k = \sum_{i \in \Omega_k} |(F_n)_i| \).

4.2 Diffusion sequence \( f \)

Here we need to choose the sequence order \( i \in \Omega_k \) for each \( k \). By default, we can apply a cyclic order. We could apply also some greedy approach as in [4] [3]. Finding the optimal sequence or a practical sub-optimal sequence for each \( k \) is an open problem.

4.3 Sharing locally updated results

The transmission of \( H \) to other \( PID \) is triggered when

- \( r_k < T_k \), or
- an update of \( H \) is received from another \( PID \).

In the version (V2), \( F \) may be sent only when:

- \( r_k < T_k \).

When the \( PID \) advance at very different speeds (monitoring \( T_k \)), we can think of splitting the set \( \Omega_k \) associated to the slowest \( PID \) or possibly regrouping \( \Omega_k \) associated to the fastest \( PID \) etc.
4.4 Distance to the limit

The limit is reached when $\sum_k r_k = 0$. In case of PageRank style equations, it has been shown in [4] that $(\sum_k r_k)/(1-d)$ defines an exact distance to the limit or an upper bound in the presence of dangling nodes.

In the general case, the spectral radius of $P$ plays a role (but is not necessarily known). For instance, if for all $i$, $\sum_j |p_{ji}| < 1$, then taking $\epsilon = \min_i (1 - \sum_j |p_{ji}|)$, $(\sum_k r_k)/\epsilon$ defines an upper bound of the distance to the limit.

5. EXAMPLES

5.1 Example with 2 PIDs

Let’s take a simple example to illustrate the above method. We set:

$$A(1) = \begin{pmatrix} 5 & 3 & 0 & 0 \\ 3 & 7 & 0 & 0 \\ 0 & 0 & 8 & 4 \\ 0 & 0 & 2 & 3 \end{pmatrix}$$

And we look for $X$ such that $AX = B = (1, 1, 1, 1)^t$.

In this case, we defined $A(1)$ so that there is no correlation between $\Omega_1 = \{1, 2\}$ and $\Omega_2 = \{3, 4\}$. As expected, then the gain factor is about 2 (assuming no information transmission cost) with 2 PIDs as shown in Figure 1:

![Figure 1: Example: 2 PIDs for $A(1)$](image1)

For the D-iteration, we applied the cyclical sequence $\{1, 2, 3, 4\}$ (using the equation (5) on $H_n$). For 2 PIDs case, we applied jointly the cyclical sequence $\{1, 2\}$ and $\{3, 4\}$ exactly twice before sharing the local computation results.

In this case, we added values in $A(2)$ so that there is correlation between $\Omega_1$ and $\Omega_2$. Then there is still a visible gain factor as shown in Figure 2:

![Figure 2: Example: 2 PIDs with correlation for $A(2)$](image2)

Finally, we set:

$$A(3) = \begin{pmatrix} 5 & 3 & 1 & 1 \\ 3 & 7 & 1 & 1 \\ 1 & 1 & 8 & 4 \\ 1 & 1 & 2 & 3 \end{pmatrix}$$

In this case, we added 1 on $(A(3))_{2,4}$. Then there is no longer any significant gain as shown in Figure 3:

![Figure 3: Example: 2 PIDs with correlation for $A(3)$](image3)

5.2 Example of $A$ updates with 2 PIDs

We set:

$$A = \begin{pmatrix} 5 & 3 & 0 & 0 \\ 3 & 7 & 0 & 0 \\ 0 & 0 & 8 & 4 \\ 0 & 0 & 2 & 3 \end{pmatrix}$$

and

$$A' = \begin{pmatrix} 5 & 3 & 0 & 0 \\ 3 & 7 & 0 & 1 \\ 0 & 0 & 8 & 4 \\ 0 & 0 & 2 & 3 \end{pmatrix}$$
Then $P$ and $P'$ are defined by:

$$P = \begin{pmatrix}
0 & -3/5 & 0 & 0 \\
-3/7 & 0 & 0 & 0 \\
0 & 0 & 0 & -4/8 \\
0 & 0 & -2/3 & 0
\end{pmatrix}$$

and

$$P' = \begin{pmatrix}
0 & -3/5 & 0 & 0 \\
-3/7 & 0 & 0 & -1/7 \\
0 & 0 & 0 & -4/8 \\
0 & 0 & -2/3 & 0
\end{pmatrix}$$

$P$ has been applied up to iteration 5, then we switched to $P'$ from iteration 6. Figure 4 shows the results:

![Figure 4](image)

**Figure 4:** Example: 2 PIDs with evolution of $P$ to $P'$.

The above examples are only for easy illustration. The gain of the distributed approach should be much clearer for the computation of $X$ for large matrix $P$. This will be addressed in a future paper in the context of the PageRank equations, on the web graph (on which the gain of such an approach without distributed computations is shown in [4]) or on the general graph (such as the PageRank extensions on the paper-author graph for the research publications [5]).

6. CONCLUSION

In this paper, we presented two asynchronous computation schemes associated to the D-iteration approach. We believe that its potential is very promising and further investigation (and implementation) for a really large $P$, such as for the PageRank matrix associated to the web graph, will be addressed in a future paper.

Acknowledgments

The author is very grateful to Gérard Burnside for his valuable comments and suggestions.

7. REFERENCES

[1] R. Bagnara. A unified proof for the convergence of jacobii and gauss-seidel methods. *SIAM Review*, 37, 1995.

[2] G. H. Golub and C. F. V. Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[3] D. Hong. D-iteration method or how to improve gauss-seidel method. *arXiv*, [http://arxiv.org/abs/1202.1163](http://arxiv.org/abs/1202.1163), 2012.

[4] D. Hong. Optimized on-line computation of pagerank algorithm. *submitted*, 2012.

[5] D. Hong and F. Baccelli. On a joint research publications and authors ranking. *HAL*: [http://hal.inria.fr/hal-00666405](http://hal.inria.fr/hal-00666405), 2011.

[6] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.