A theorethic treatment of the "Partitioned Iterative Convergence" methods

1 Introduction

Implementing an iterative algorithm in PIC can greatly reduce the run-time of the computation. In our experiments we have observed a typical factor of 4x to 6x speedup. On the other hand, the PIC framework in effect changes the algorithms. As such, we have to address the accuracy issues of PIC in more detail.

Through experimental studies we have shown the PIC system computes results that are either equal in numerical value to the baseline implementations, or are very close (we have observed up to 3% difference in numerical values). The question we try to address in this document is whether we can extend these expectations to a larger class of problems.

We first describe the three issues involved in the discussion of PIC results accuracy in section 2. We then establish the similarity between PIC and domain decomposition methods in section 3, which allows us to consider PIC as a domain decomposition method. Section 4 analyzes PIC as a preconditioner, and shows that it indeed converges to the same set of answers with a convergence rate related to the convergence rate of the original algorithm.

2 Accuracy of solutions

There are multiple aspects to the accuracy issues that PIC has to address. These accuracy issues are enumerated in sections 2.1, 2.2 and 2.3.

2.1 Accuracy requirements of the application domain

The first aspect of accuracy comes from the application domain. Sometimes the application domain itself has forgiving nature, and can withstand slightly inaccurate results from the computational kernels and algorithms that solve a certain computational problem. Note that from this aspect, we are differentiating between a target application and algorithms that solve computational problems towards that application. To illuminate this point of view, we provide the following examples:

- Web search: As long as the order of the web pages are the same, the numerical page rank values of each web page is irrelevant.
- Graphics: It is acceptable to have the values of a few pixels be slightly inaccurate. For example, in computer games, since each frame has a life time of at most $\frac{1}{30} seconds$, a few pixels can be (and in many current games are) inaccurate. Similarly, JPEG or MPEG compressions rely on the fact that human eye cannot discern slight inaccuracies in the pixel values.
- Data mining: Many of the algorithms in data mining are heuristic, and even in their original form at most provide a sub-optimal answer (e.g. K-means clustering).

2.2 Final numerical accuracy

Let us temporarily assume that the our observations in the experimental results are general, i.e. for a large class of problems, one can expect much faster run-times with a final inaccuracy of less than 3%. In other words, we assume the trajectory of the PIC results is towards the correct answer (also see section 2.3). In case the application domain is forgiving enough, it can utilize the computed values, making PIC a suitable choice. Furthermore, even if the application domain requires accurate answers (that result from the original algorithms), we can still benefit from PIC as "pre-conditioners". Basically, we can let PIC quickly reach its final accuracy level, and then switch back to the original algorithm. But now we can use the results computed from the PIC run as initial values for the

original algorithm. From our experimental results, we see that this hybrid approach can still drastically out-perform a regular iterative implementation.

Preconditioners are frequently used to "prime" differential equation solvers, and are otherwise known as domain decomposition methods. In the rest of this document, we will consider PIC as a method based on additive Schwarz preconditioners.

2.3 Trajectory of accuracy

The final aspect of the accuracy issue considers the trajectory of the results created by PIC. In other words, how do we know that an algorithm implemented in PIC will not diverge or create drastically different results. The rest of this document addresses this issue.

3 Applicability of Domain Decomposition analysis techniques to PIC

Domain decomposition methods have received a great deal of attention lately. These techniques are designed for differential equations, and rely on the fact that the computation pattern of the problem domain is local, or more precisely stencil-based. To compute the value of the target function (described by the difference equations) at each point of a grid, the algorithm only needs to know the values of its neighbors. The order of the differential equations dictate how many layers of neighbors information are required. For instance, a differential equation of order 1 requires only the immediate neighbors of a certain point. An order 2 equation requires the values of neighboring nodes and their neighbors.

Domain decomposition methods first partition the grid into separate domains and then solve the the numerical difference equations in each domain separately. A domain decomposition method can choose to compensate for the boundary values of the partitions after each iterations (as multiplicative Schwarz methods do), or they may ignore them (as additive Schwarz methods do) [1, 2]. In either case, the trajectory of the solutions is convergent towards the correct solution (also see section 4.)

As mentioned, the reason domain decomposition techniques are successful is that the computations within each partition (or domain) is truly local. This data locality can be represented in a tridiagonal dependency matrix [3] for differential equations of order 1. Higher order differential equations can be modeled with Block tridiagonal matrices.

Additive Schwarz methods basically treat the block tridiagonal matrix as if it was a block diagonal matrix, ignoring the dependencies among different blocks.

For a computational problem to be a suitable match for implementation in PIC, it should have similar dependency patterns.

The authors would like to note that these problems have been addressed in the Operations Research community as well [4, 5], where the same condition of the problem domain is referred to as "nearly uncoupled" problems, also known as "nearly completely decomposable" problems. [5] shows that a aggregation/disaggregation techniques (including techniques such as "Koury, Mcallister, stewart", "Takahashi" and "Vantilborgh") can compute Markov chain steady state solutions through a process very similar to PIC. Furthermore, they converge to the accurate solution with a certain error bound if a set of "regulatory conditions" are satisfied for the Markov chain matrix. These regulatory conditions dictate the level of "decomposability" of the problem, and in essence are equivalent to the conditions that a domain decomposition technique needs to satisfy (which are described in more detail in section 4.2). [4] extends this notion to the general class of problems, namely the fixed point of nonexpansive mappings (a sub-class of contraction mappings). It is shown [6] that many interesting non-linear (and linear) operators such as Jacobi and Gauss-Seidel operators and their block variants, as well as optimization techniques including gradient projection are indeed contraction mappings, and as such the Takahashi aggregation/disaggregation iterative method can be applicable on them. In a related effort, [6] analyzes many contraction mapping operators used for iterative algorithms. It proposes to go a step beyond aggregation/disaggregation method of Takahashi by synchronizing (aggregating) the sub-problems asynchronously. Similar to the techniques in [4, 5, 1, 2] the conditions on the decomposability of the problem guarantee the convergence. Naturally, the conditions imposed by [6] are more restrictive than the ones proposed by Takahashi [4, 5]. In turn, the conditions imposed by additive Schwarz domain decomposition methods [2, 1] are slightly less restrictive than the Takahashi methods, since these methods are considered as preconditioners for application of Krylov subspace methods (e.g. Conjugate gradient methods), and therefore can withstand slightly inaccurate results [2]. Since PIC targets applications that can withstand some inaccuracy, or otherwise can accept PIC as a preconditioner, we analyze the PIC methods as an additive Schwarz domain decomposition technique.

A computational problem with a high degree of dependence among the data elements is not a good match for PIC (after all "PIC" stands for "partitioned iterative convergence"). However, many application domains have the

"nearly uncoupled" property. Whenever a computational problem has such tendencies it is a potential match for implementation in PIC. Furthermore, if the application has a forgiving nature, we can increase the "nearly uncoupled" property by temporarily ignoring some of the dependencies (using the local iterations mechanism).

As an example, consider the PageRank algorithm. In a parallel implementation, each node requires information from the adjacent nodes and edges for the computation. If the web was a complete graph with $\frac{n.n-1}{2}$ edges it would not be a good match for PIC implementation. Fortunately the web graph is typically local, and by properly partitioning it (for example using the METIS package), the connectivity matrix of the graph becomes nearly uncoupled.

Similar arguments apply to the K-means clustering. The impact of far-away points on a centroid is much smaller than the impact of close points to that centroid. As such, a rough first-pass partitioning can ensure that each sub-problem mostly relies on the points inside that partition.

The image smoothing algorithm is stencil based and clearly the dependencies are local. In the case of the linear system of equations, the "weak diagonal dominant" matrix of the equations guarantees the "nearly uncoupled" property. In fact, the weak diagonal dominance property is powerful enough to ensure even asynchronous convergence [6, pp. 10–15].

4 Conditions for convergence of PIC

4.1 PIC as an additive Schwarz preconditioner

We start by showing that a regular MapReduce implementation of an iterative algorithm (called "baseline" from here on) is equivalent to a block-Jackobi preconditioner. In a baseline implementation the input dataset is partitioned into HDFS-defined chunks, then the *map()* function is applied on each input record within a chunk on one cluster node. The intermediate results from the application of *map()* function are then all gathered before being passed to the local combiner function. As such, since the information used in the combiner are all from the previous iteration and none of them are updated, the information flow is similar to a Jacobi iteration (as opposed to a Gaus-Seidel iteration where the result of each *map()* would impact the values used in the other *map()* functions). The partitioning of data by HDFS resembles the "block" aspect of the block-Jacobi method. In turn, a block-Jacobi operator is equivalent to a simple form of additive Schwarz operator where there is no overlap in the partitions [2].

An additive Schwarz preconditioner is defined as

$P^{ad} = \sum_{i=0}^{n} P_i,$

where each P_i is an orthogonal projection onto the subspace spanned by the elements in the sub-problem i [2]. Formally:

$P_i = B_i A = (R_i^T (R_i A R_i^T)^{-1} R_i) A$

where A represents (in matrix form) the mathematical operation happing in each iteration ($y = Ax$), R_i is the *restriction* operator from the original problem into a sub-problem (and is merely a formalization of the partitioning process in PIC), and R_i^T the *interpolation* represents mapping the sub-results back into the main problem space (ans is the formalization of the merge operator in PIC). The matrix B_i in effect restricts the problem to one sub-domain, solves the problem locally in that sub-domain and generates a local correction vector, and finally extends the correction back to the entire domain [2].

We would like to emphasize again that the R_i and ${R_i}^T$ matrices are never formed in practice; they are simply introduced to express several different types of partitioning and merging in a similar concise manner [2]. In implementation it is often convenient to represent R_i as an integer array containing the global sub-domain membership of each data element.

Since P_i is an orthogonal projection, the local correction vector $e_i = P_i e = B_i(f - Au^n)$ has a most attractive property: It is the closest vector to e in the sub-space spanned by the members of P_i.

Thus, the basic additive Schwarz method operates by projecting the error simultaneously on the subspaces V_i, and uses these projections as local corrections to the approximate solution in each iteration. Similarly, PIC finds the local correction vector ($e_i = P_i e$) multiple times. This can be written as:

$e_i = P_i^{PIC} e = P_i P_i P_i ... P_i e = P_i^k e$

Which in turn means the PIC method can be written as an additive Schwarz method itself, where:

$P^{PIC} = \sum_{i=0}^{n} P_i^{PIC}$

4.2 Sufficient Conditions for the stability of an additive Schwarz preconditioner

We start by assuming the baseline algorithm itself has convergence guarantees. If a set of conditions are sufficient for convergence of an additive Schwarz preconditioner, they are also sufficient for convergence of the block-Jacobi schemes, and in turn sufficient conditions for the convergence of the baseline algorithm.

As such, we can safely assume that the baseline algorithm being targeted for PIC implementation has sufficient conditions that ensure its convergence, specifically the sufficient conditions for the convergence of additive Schwarz preconditioners. We use the set of sufficient conditions presented in [1] for this analysis.

The following three conditions are sufficient for an additive Schwarz preconditioner to be convergent [2, 1]. These three conditions which identify the amount of interaction of the subspaces (or sub-problems) and the approximation properties of the iterative operations.

1. Stable Decomposition: this condition is simply satisfied if there exist a decomposition of the problem space, such that composition of them can successfully represent the original problem.
2. Strengthened Cauchy-Schwarz Inequality: This condition puts a bound on the orthogonality of the subspaces.
3. Local Stability: Ensure that the local operators are coercive and gives a measure of the approximation properties of the iterative operators.

In the next three subsections, we start from the assumption that the baseline implementation is an additive Schwarz preconditioner that satisfies a certain convergence property, and we show that implementation of that algorithm in PIC results in another additive Schwarz operator that converges to the same answer and has a convergence rate that relates to the baseline algorithm. We would like to urge the interested reviewer to consider the following material in the context of the abstract theory of Schwarz methods, specifically pages 35-46 of [1].

4.2.1 Stable Decomposition

Both the stable decomposition condition and the Strengthened Cauchy-Schwarz Inequality condition (in the next section) are mostly independent of the choice of the preconditioner function. Instead, they impose limits on the partitioning function.

The stable decomposition condition simply ensures that the subspaces are able to represent all the elements in the problem input space. Formally, there exists a constant C_0, such that every $u \in V$ admits a decomposition
$u = \sum_{i=0}^{P} {R_i}^T u_i, \quad \{u_i \in V_i,\ 0 \leq i \leq P\}$
that satisfies the following condition
$\sum_{i=0}^{P} \widetilde{a}_i(u_i, u_i) \leq {C_0}^2 a(u, u)$ The C_0 parameter puts a lower bound on the eigenvalues of an additive Schwarz method.

In PIC, this condition clearly holds. The left hand side of the inequality corresponds to merging of the results related to a certain data point across the partitions, while the right hand side refers to the application of the baseline operator on that data item. If the partitioning was performed such that one data item would be split across more than one sub problems, this could be an issue to consider. However, in PIC the default partitioners divide the input data points such that each point belongs to at most 1 sub problem. Formally, if the input element u is placed in partition k we have the following:
$\widetilde{a}_i(u_i, u_i) = 0 \ \ \forall\, i, i \neq k$
$\Rightarrow \sum_{i=0}^{P} \widetilde{a}_i(u_i, u_i) = \widetilde{a}(u_k, u_k)$
The condition then boils down to $\exists C_0 \ \mid \ \widetilde{a}_k(u_k, u_k) \leq {C_0}^2 a(u, u)$.

Therefore, we only need to ensure that none of the local solvers create a value at the end of their local iterations that is infinitely larger than the baseline. In PIC and using the default partitioners, $C_0 = 1$.

4.2.2 Strengthened Cauchy-Schwarz Inequality

The strengthened Cauchy-Schwarz condition puts a bound on the interdependence of the sub-problems. This condition results in a matrix ϵ, whose elements enumerate the interdependence of each two sub-problems. As a result, the spectral radius of this matrix will appear in an upper bound for the largest eigenvalue of an additive Schwarz preconditioner, which in turn puts a bound on the condition number and convergence rate of the preconditioner.

Formally, there exits constants ϵ_{ij} (elements of the "inter-dependence matrix" ϵ, where
$0 \leq \epsilon_{ij} \leq 1,\ 1 \leq i, j \leq P$, where P is the number of partitions, and
$|a({R_i}^T u_i, {R_j}^T u_j)| \leq \epsilon_{ij} a({R_i}^T u_i, {R_i}^T u_i)^{1/2} a({R_j}^T u_j, {R_j}^T u_j)^{1/2}$,
for $u_i \in V_i$ and $u_j \in V_j$. The spectral radius of the inter-dependence matrix ϵ is denoted by $\rho(\epsilon)$.

If two sub-problems V_i and V_j are completely dependent on each other, the strengthened Cauchy-Schwarz Inequality reverts back to the trivial case of the regular Cauchy-Schwarz Inequality. If this is the case for all i and j, we have $\rho(\epsilon) = P$, which would give a poor upper bound and convergence rate. On the other hand, the best bound is achieved for completely orthogonal sub-spaces $\{{R_i}^T {V_i}^T\}$ (e.g. block diagonal matrices), in which case $\epsilon_{ij} = 0$ for $i \neq j$ and $\rho(\epsilon) = 1$.

In reality, $1 \leq \rho(\epsilon) \leq P$. If the sub-problems have small amounts of inter-dependence on each other, the $\rho(\epsilon)$ will be closer to 1. As mentioned earlier, PIC is designed to target problems that can be successfully partitioned. A block diagonal matrix ensures $\rho(\epsilon) = 1$. A block tridiagonal matrix has a $\rho(\epsilon)$ much closer to 1 than N. In any case, the natural decomposability of the problems targeted by PIC (or other additive Schwarz preconditioners) ensures a proper value of $\rho(\epsilon)$.

4.2.3 Local Stability

The last assumption is related to the stability of local operators. Basically we need to find a lower bound (ω) on the required scaling factor of the local solutions to ensure that for any data point the scaled result will be larger than the original solver. Formally, we need to show that:

$\exists\, \omega \in [1, 2) \mid a({R_i}^T u_i, {R_i}^T u_i) \leq \omega.\widetilde{a}(u_i, u_i), \quad u_i \in V_i\ ,\ 0 \leq i \leq P$

where ${R_i}^T$ is the interpolation operator, $a(.,.)$ is a symmetric, positive definite bilinear form representing the computation, $\widetilde{a}(.,.)$ represents the local computations, V_i is a sub-problem and P is the number of sub-problems. In theory if exact local solvers are used, i.e. if $a_i(.,.) = a(.,.)$ then $\omega = 1$. However, the local computations are not exact for two reasons:

- We have omitted the impact of some input points that lie outside a partition
- Over tightening of the local solutions can result in over-strengthened local solutions and slower global convergence, therefore we usually do not run the local solvers until local convergence

For this reason the local computations are represented by $\widetilde{a}(.,.)$ instead.

As mentioned earlier, PIC is equivalent to the baseline if it is run for only 1 local iteration (and keeping the partitioning and merging phases). In this case, PIC is completely equivalent to an additive Schwarz method, and we can assume the local stability assumption holds for it. In other words, there exists an $\omega \in [1, 2)$ which satisfies the stability condition. We will now look at what happens when the number of local iterations is increased to k:

$a({R_i}^T u_i, {R_i}^T u_i) \leq \omega.\widetilde{a}(u_i, u_i) \Rightarrow$

$a({R_i}^T u_i, {R_i}^T u_i)^k \leq \omega^k.\widetilde{a}(u_i, u_i)^k$

(EQ. 1) the bilinear form is positive definite, so the inequality can be raised to the power of k. This is equivalent to application of each computation k times. The left hand side represent k iteration of the baseline algorithm, while the right hand side represents k local iterations.

We first work on the left hand side of the inequality:

$a({R_i}^T u_i, {R_i}^T u_i)^k = ({R_i}^T u)^T A({R_i}^T u)^k$

where A is the stiffness matrix associated with the bilinear form $a(.,.)$, representing the computation in a matrix format. Continuing:

$= (u^T {R_i}^{T^T} A\ {R_i}^T u)^k = (u^T R_i A {R_i}^T u)^k$

$= u^T R_i A {R_i}^T u u^T R_i A {R_i}^T u u^T R_i A {R_i}^T u\ ...\ u^T R_i A {R_i}^T u \quad (k \text{ times})$

Note that u and u^T are one dimensional vectors, therefore their multiplication results in a real number α, representing the dot product of the input vector u with itself, or the squared length of u:

$= u^T R_i A {R_i}^T \alpha R_i A {R_i}^T \alpha R_i A {R_i}^T\ ...\ \alpha R_i A {R_i}^T u \quad (k \text{ times})$

$= \alpha^{k-1}(u^T R_i A {R_i}^T R_i A {R_i}^T R_i A {R_i}^T\ ...\ R_i A {R_i}^T u) \quad (k \text{ times})$

As mentioned earlier, the restriction and interpolation operators (R_i and ${R_i}^T$) are just formalizations of the partition and merge functions. We can assume the partitioning and merging operations by themselves are lossless, i.e. running one after another represents a round of partitioning and merging that cancel each other out with no effect on the computations. Therefore, we set ${R_i}^T R_i = 1$ and get the following:

$= \alpha^{k-1}(u^T R_i AAA\ ...\ A {R_i}^T u) \quad (k \text{ times})$

$= \alpha^{k-1}(u^T R_i A^k {R_i}^T u)$

Therefore, the left hand side of EQ. 1 is shown to be a scaled version of the regular baseline with k

Now we work on the right side of the EQ. 1, using the local stiffness matrix representation $\widetilde{A}_i$:

$\omega^k.\widetilde{a}(u_i, u_i)^k = \omega^k.({u_i}^T \widetilde{A}_i u_i)^k$

$= \omega^k.(u_i{}^T \widetilde{A}_i u_i u_i{}^T \widetilde{A}_i u_i u_i{}^T \widetilde{A}_i u_i ... u_i{}^T \widetilde{A}_i u_i)$ (k times)

Note that u_i and $u_i{}^T$ are one dimensional vectors, therefore their multiplication results is a real numbers β_i, representing the dot product of u_i with itself or the squared length of the partitioned input vector:

$= \omega^k.\beta_i{}^{k-1}(u_i{}^T \widetilde{A}_i \widetilde{A}_i \widetilde{A}_i ... \widetilde{A}_i u_i)$ (k times)

which is the definition of the PIC additive Shwarz preconditioner. Replacing the left and right hand sides back in EQ. 1, we get:

$\alpha^{k-1} a'(R_i{}^T u, R_i{}^T u) \leq \omega^k.\beta_i{}^{k-1} a_i{}^{PIC}(u_i, u_i)$

where $a'(.,.)$ represents k baseline iterations. Therefore:

$a'(R_i{}^T u, R_i{}^T u) \leq (\frac{\omega.\beta_i}{\alpha})^{k-1} \omega a_i{}^{PIC}(u_i, u_i)$

Therefore, compared to k iterations of the baseline, the PIC preconditioner with k local iterations in the $i'th$ subproblem has a scaling factor of $(\frac{\omega.\beta_i}{\alpha})^{k-1}$ in its local stability condition (which also verifies our assumption that in case of only 1 local iteration, PIC and baseline have the same local stability). To arrive at the scaling factor that the PIC method imposes on top of the baseline convergence rate, we can replace local β_i with the maximum of β_i's, which we denote by $\beta = max(\beta_i), \forall i$. Note that as mentioned earlier, α represents the length of the input vector u and β_i (and in turn β) represent the length of the partitioned local input vectoru_i. Therefore, the ratio of $\frac{\beta}{\alpha}$ is a number between 0 and 1. The more we partition a problem, the smaller this ratio will get, and as one can expect, the rate of convergence will become slower.

As a final step, to be able to use the condition number formula of an additive Schwarz preconditioner [2, 1] we need to find a scaling factor that is comparable to one iteration of the baseline. Therefore, we have to use the k^{th} root of this scaling factor $\sqrt[k]{(\frac{\omega.\beta}{\alpha})^{k-1}} = (\frac{\omega.\beta}{\alpha})^{\frac{k-1}{k}}$ in the following formula to compute the condition number of the PIC compared to the baseline. In general, the condition number of an additive Schwarz preconditioner satisfies [2, 1]:

$\kappa(P_{ad}) \leq C_0{}^2 \omega(\rho(\epsilon) + 1)$

Therefore, the condition number of the PIC method, using the parameters computed so far, is as follows:

$\kappa(P_{ad}) \leq C_0{}^2 (\frac{\omega.\beta}{\alpha})^{\frac{k-1}{k}} \omega(\rho(\epsilon) + 1)$ □

5 Conclusions

Through this document a few major insights have become clear.

- In the problems that PIC targets, the interdependency matrix should have a semi-block-tridiagonal matrix. The amount of discrepancy between the PIC solution and the baseline depends on the impact of the triangular blocks between the major blocks.
- The previous insight can be quantified by the ϵ matrix from the strengthened Cauchy-Schwarz inequality condition. $\rho(\epsilon)$, representing the largest eigenvalue of the ϵ matrix, puts a bound on the "decomposability" of the problem and the quality of the partitioning function at the same time.
- Compared to the baseline algorithm's convergence rate, the PIC implementations converge to the same answer with a scaling factor in their convergence rate. The scaling factor is $(\frac{\omega.\beta}{\alpha})^{\frac{k-1}{k}}$. Therefore the convergence rate decrease is proportional to a function of number of local iterations, the convergence power of the algorithm represented by ω, ‘ and the ratio of the length of non-partitioned input vectors to their partitioned counterparts.

References

[1] A. Toselli and O. Widlund, *Domain Decomposition Methods - Algorithms and Theory*, ser. Springer Series in Computational Mathematics. Springer, 2004, vol. 34.

[2] B. F. Smith, P. E. Bjørstad, and W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[3] Q. M. Al-Hassan, *On Powers of Tridiagonal Matrices with Nonnegative Entries*, ser. Applied Mathematical Sciences. New York: Springer Verlag, 2012, vol. 6, no. 48.

[4] T. W., "Iterative methods for approximation of fixed points and their applications," *Journal of the Operations Research Society of Japan*, vol. 43, no. 1, pp. 87–108, 2000. [Online]. Available: http://www.ingentaconnect.com/content/els/04534514/2000/00000043/00000001/art88753

[5] W.-L. Cao and W. J. Stewart, "Iterative aggregation/disaggregation techniques for nearly uncoupled markov chains," *J. ACM*, vol. 32, no. 3, pp. 702–719, Jul. 1985. [Online]. Available: http://doi.acm.org/10.1145/3828.214137

[6] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.