CHAPTER 21

Efficiency of Multicolor Domain Decomposition on Distributed Memory Systems

Luigi Brochard*

Abstract. This paper presents a model for the performance prediction of some domain decomposition methods on MIMD distributed memory systems. This performance analysis is based on a deterministic model of the computation, communication and control complexities of the algorithm on a given architecture. The classes of domain decomposition we are studying are based on multicolor techniques where the convergence properties of the iterative solver are independent of the number of subdomains. This allows us to have a good understanding of the overheads of a parallel computation without dealing with the numerical behavior of the parallel method. Using this analysis, the efficiency of multicolor SOR and multicolor multigrid methods is studied, depending on the domain decomposition (slice, square, cube) and on different parameters. Special attention will be given to the multigrid methods and to some variants.

1.Introduction.

Analyzing and predicting the performance of a multiprocessor system is very complex, since many factors jointly determine the behavior of the system. In this paper, we present a deterministic model which can provide a good understanding of the interactions between the algorithm and the architecture. As a case study, we have chosen domain decomposition methods which seem well adapted to parallel computing, and particularly a class of domain decomposition methods which provides a fully parallel algorithm and a convergence rate independent of the number of subdomains.
The basic idea of domain decomposition methods is very simple: instead of solving a partial differential equation on a given domain, we decompose the initial domain in smaller ones and obtain the overall solution by solving smaller problems on these subdomains. Though this idea is rather old, and related to the Schwarz Alternating Method [27], it has triggered in the past several years a burst of new studies in this area, mainly due to the good fitting between the domain decomposition idea and parallel computing. A lot of these new studies are related to a class of domain decomposition techniques which can be called Preconditoned Gradient Method where the original operator on the whole domain is reduced to an operator on the interfaces of the subdomains which is solved using the iterative preconditioned conjugate gradient method. Several preconditioners have been proposed in the past years, see [19] for a review of some of them, and some new ones have been presented at this conference. A good advantage of this class of methods is that, using good preconditioners, the number of iterations can be kept small. Unfortunately, the condition number

* IBM Scientific Center, 1530 Page Mill Road, Palo Alto, CA 94304
Permanent address:
Ecole Nationale des Ponts et Chaussees, La Courtine B.P 105, 93194 Noisy le Grand Cedex, France

depends on the domain decomposition (slice, square, cube), and on the number of subdomains [9].
Therefore, as our main goal is the study of the parallel efficiency of the method, and of the different overheads related to its parallel implementation, we will move our attention to a particular class of domain decomposition methods where the number of iterations is independent of the number of the subdomains.
The outline of this paper is the following. In section 2, we present the domain decomposition technique we will be studying, the multicolor SOR method and its accelerated version using a multigrid adaptation. In section 3, we present the distributed memory systems we will be dealing with, and the performance analysis model we will be using. This deterministic model is based on the algorithm computation, communication and control complexities on the specific network of the MIMD distributed memory machine. On this basis, the general speed-up behavior of such parallel numerical algorithms will be discussed. In section 4, the performance analysis will be applied to the two parallel solvers we have presented and their efficiency will be compared depending on the domain decomposition and on the number of subdomains. A modified version of the standard multigrid algorithm will be also presented, and compared to the original one. It should be noted that throughout this paper we assume the number of processors and its memory size are large enough to have each subdomain associated with a specific processor.

2. Multicolor domain decomposition.

The basic idea of domain decomposition method is that instead of solving the initial problem, $f(u) = g$ on a domain Ω, the problem is split into P subproblems $f_{|i}(u) = g_{|i} = i = 1...P$, where $f_{|i}$ and $g_{|i}$ are the restrictions of f and g to the subdomains Ω_i , with $\Omega = \bigcup \Omega_i$, and a coupling condition at the subdomain interfaces.

2.1 Multicolor SOR method. In the following, we consider methods where every subdomain is only coupled to its neighbors through the exchange of boundary values between adjacent regions at every iteration. It can be applied as in the original method introduced by Schwarz [27] , as an alternate method, where every subdomain will be successively solved, or using multicolor relaxation schemes. With such coloring techniques, it is well known that for a large class of partial differential equations, there exists c color ordering schemes which decompose the naturally ordered one step Gauss Seidel method into an equivalent c step Jacobi method, where each step is parallel contrary to the sequential original Gauss Seidel method [1,21,22] . The best example is the red black ordering applied to the five point symmetric stencil, which leads to a two step Jacobi equivalent to the natural ordered Gauss Seidel method [31]. Using such an ordering, all the same color points can be processed simultaneously. With the Zebra method, as shown on Fig. 1, it is easy to see that depending on the processor allocation, you can get a scattered or gathered allocation.

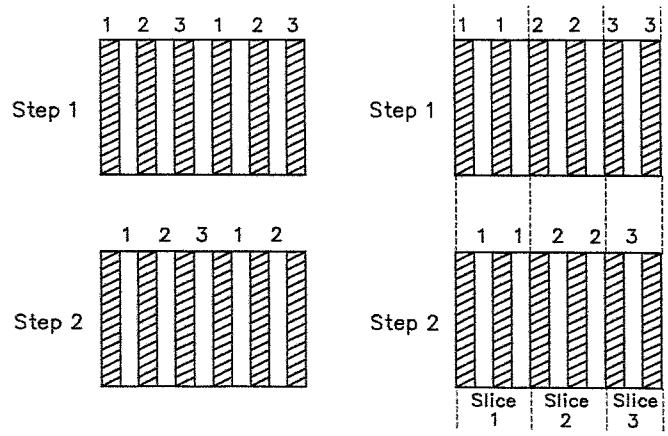

Figure 1
Scattered and gathered processor allocation of a 2 color Zebra method.

The scattered allocation leads to a heavy communication scheme since at every step each processor has to fetch one column from a neighbor processor, while with the gathered allocation the only interface column is be needed. This can be generalized to c color orderings and different geometries, though the coloring and load balancing may become more difficult [2,22]. This gathered processor allocation applied to the multicolor relaxation scheme will be called multicolor domain decomposition.

*2.2 **Multicolor multigrid method.*** In this paper, we only consider the standard multigrid method where the $\ell+1$ grid levels of the multigrid method are processed from finer to coarser and numbered 0 to ℓ. On each grid level a smoothing operator, typically several SOR relaxation steps, is applied and reduction and interpolation operators are used to project the residual of the equation on the the finer grid level to the coarser level, and vice versa. Fig. 2 presents different cycles that can be used between the different grid levels. With an appropriate choice of operators and cycles, it has been proved the error of this iterative method at every step is reduced by a constant factor independent of the mesh size, leading to an optimal sequential method [29].

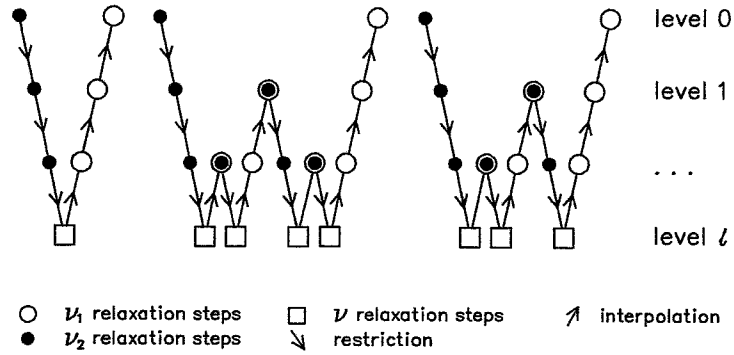

Figure 2
V, W and F cycles of a 4 level multigrid method.

We define the multicolor multigrid method as a parallel multigrid method where each grid level is performed sequentially as in the original sequential algorithm, but where the grid points within each level are performed simultaneously using the multicolor domain decomposition presented above. Embedding of the different subdomains is assumed on all grid levels, cf Fig. 3.

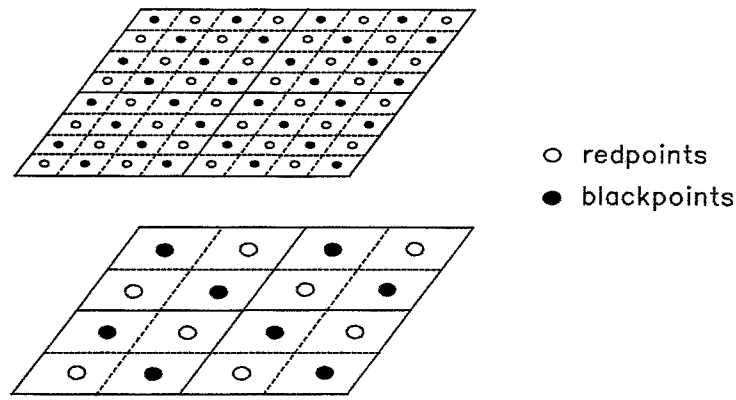

Figure 3
Subgrid projection of a 2 level and 2 color multigrid method.

This standard multigrid method [3], can be seen as horizontal parallel method, in opposition to some recent adaptation where a vertical parallelism can be applied to the method leading to a data driven algorithm where all grid levels are performed simultaneously [8,12,16] .
In the following we will deal with this only standard multigrid method, where each processor handles a subset of the finest grid level and its projection on the coarser levels, and try to understand the impact of communication on this method and on a variant where the last level is processed by a single processor trying to avoid the coarsest grid bad effect.

3. Performance analysis model

3.1 Speed-up evaluation. Throughout the literature, two different definitions of speed-up can be found: an absolute and a relative one. In this paper, as we are mainly interested in the communication and synchronization overheads of a parallel computation, we will use the relative speed-up definition of a P-processor computation of N data:

$$Sp(N,P) = \frac{T(N,1)}{T(N,P)}$$

where $T(N,1)$ denotes the execution time of the sequential algorithm and $T(N,P)$ the execution time of the studied parallel algorithm using P processors. Clearly with such a definition, there are deterministic algorithms which can give super-linear speed-up [10,23] , i.e:

$Sp(N,P) > P$

but we will restrict our study to algorithms leading to speed-up bounded by P.
In the following, we will suppose the execution time using P processors is given by the equality:

$$T(N,P) = T_{cpu}(N,P) + T_{io}(N,P), + T_{ctr}(N,P)$$

where T_{cpu} , T_{io} and T_{ctr} denote the execution times of the computation, communication and control parts of the parallel process. In this definition, the computation time is related to the arithmetic complexity of the algorithm, the communication time to its data movement complexity on a given architecture and the control time to tasks spawning, synchronization and termination detection of a distributed process. Clearly this equation states a non-overlapping condition between the various components of a parallel synchronous process. Following our definition we also state:

$$T(N,1) = T_{cpu}(N,1)$$

meaning there is neither communication nor control overhead in a sequential process.
Concerning the arithmetic complexity of a parallel algorithm, three different types can be found. In the first one, we have:

$$T_{cpu}(N,P) = \frac{T_{cpu}(N,1)}{P} \tag{1}$$

which means the algorithm is fully parallel with a perfect load balance. Matrix matrix multiplication [11] , multicolor relaxation methods [4], Gauss-Jordan matrix elimination [22] can be of this type.
In the second one, we find algorithms which are not fully parallel or where load is unbalanced. We have then:

$$T_{cpu}(N,P) > \frac{T_{cpu}(N,1)}{P}$$

For example, Gauss matrix eliminations [24] and reduction operations [27] belong to this large category. Those algorithms present a non constant degree of parallelism which causes a degraded speed-up independent of the parallel overheads.
At last, we could be interested finding parallel algorithms such that:

$$T_{cpu}(N,P) < \frac{T_{cpu}(N,1)}{P}$$

leading to a potential super-linear speed-up. Typically, this could arise using non-linear complexity algorithms which have been parallelized by partitioning techniques such as aggregation-

disaggregation, sub-structuring or domain decomposition methods. For example if we suppose an $O(N^2)$ algorithm, its computation time will be divided by 4 on a two $N/2$ subsets partition and can lead to a super-linear relative speed-up if the aggregation part of the algorithm does not waste the gain. Matrix rank one tearing and updating belongs to this category [10], which does not imply a better absolute speed-up [20]. As we are mainly interested in the communication and control overheads of a parallel computation, we will suppose a fully parallel algorithm and a perfect load balance (1). Under those hypotheses the relative speed-up is bounded by P and can be expressed:

$$Sp(N,P) = \frac{P}{1 + a(N,P) + b(N,P)} \tag{2}$$

where $a(N,P)$ and $b(N,P)$ are positive functions of N and P measuring the parallel overhead. More precisely, let be a P-processor, with P given by:

$$P = Q^k$$

where k is the dimension of the processors network and Q the number of processors per direction , see paragraph 3.2. Then if we suppose the communication and control algorithms complexities can be expressed as rational functions of Q such as:

$$T_{io}(N,P) = \sum_{i=-k}^{m} \alpha_i(N).Q^i$$

and

$$T_{ctr}(N,P) = \sum_{i=-k}^{m} \beta_i(N).Q^i$$

where k, m, α and β would have to be determined, the relative speed-up $Sp(N,P)$ can be written:

$$Sp(N,P) = \frac{P}{1 + (a_0 + a_1.Q + a_2.Q^2 + ... + a_k.Q^k) + (b_1.Q^{k+1} + ... + b_m.Q^{k+m})} \tag{3}$$

where a and b are respectively polynomial functions in Q of degree $\leq k$ and $> k$ with coefficients a_i and b_i given by:

$$a_i = \frac{\alpha_{i-k} + \beta_{i-k}}{T_{cpu}(N,P)},\ i = 0...k$$

and

$$b_i = \frac{\alpha_i + \beta_i}{T_{cpu}(N,P)},\ i = 1...m$$

This formulation is very general since different behaviors can be seen depending on the function $a(N,P)$ and $b(N,P)$. When $b(N,P) \equiv 0$, speed-up is a monotone increasing function of P . Then depending on the property of $a(N,P)$ we can see that if $a_k \neq 0$, the speed-up is asymptotic to $1/a_k$, while if $a_k = 0$ there is no asymptote, as we have:

$$Sp(P) \sim \frac{Q^{k-j}}{a_j}$$

where j is the greatest index of the non-null coefficients a_i. In the following, multicolor domain decomposition methods will present an application of this property. On the contrary when $b(N,P) \not\equiv 0$, formulation (3) leads to non-monotone speed-up with a critical number of processors P_c beyond which it decreases. Several examples related to global communication effects show this property: global convergence detection of a distributed computation is one [5], the modified multigrid method is another we will study below.

In the following, the execution time of the computational part of the algorithms will be given by the formulation:

$$T_{cpu}(N,P) = arith(N,P)\ .\ t_{cpu}$$

where t_{cpu} denotes the elemental time to perform an arithmetic operation and $arith(N,P)$ the arithmetic complexity of the parallel algorithm. For simplicity, no arithmetic start-up time has been included to the formulation.

3.2 Nearest neighbor network and communication complexity. In this study, we only consider nearest neighbor networks without dealing with the other various classes [17] which are much less used in MIMD distributed memory systems.
Following [15], we introduce the $NN(k,P)$ processor which is a P-processor with $P = Q^k$, $P = 2^p$, $Q = 2^q$, and where each processor is connected to its nearest neighbors with wraparound connection. With such a definition, a $NN(1,P)$ is a ring of P-processors where each one is connected to its two neighbors, a $NN(2,P)$ is a $\sqrt{P} \times \sqrt{P}$ mesh of processors connected to its four neighbors with wraparound connection, and a $NN(p,P)$ is a p-binary cube connected processor, also called hypercube of dimension p, where every processor is connected to its $\log_2 P$ neighbors. An important feature is whether it is possible in practice to use simultaneously the $2.k$ or p channels of a given node on a $NN(k,P)$, leading to communication transfer routed through different paths. In the following , we will suppose a $NN(k,P)$ processor can receive (or send) simultaneously $2.k$ messages to (or from) its $2.k$ neighbors if $k \neq p$, and send (or receive) p messages to (or from) its p neighbors on a hypercube.
A first way to give an evaluation of a static network is the maximum distance between two nodes, related to the diameter of the corresponding graph, which represents the maximum number of steps for a single message to reach one processor. It can be seen easily [15] this diameter d is given by:

$$d(k,P) = k.Q/2 \tag{8}$$

if the communication links are multidirectional, while given by:

$$d(k,P) = k.(Q-1) \tag{9}$$

if the communication links are mono-directional. For example on a $NN(1,P)$ we have $d = P - 1$ with a one-way round algorithm, while $d = P/2$ with a two-way round algorithm which sends simultaneously data in the two opposite directions of the ring, reducing the maximum distance by a factor of two. On a $NN(2,P)$, d will be either $2(\sqrt{P}-1)$ or $\sqrt{P}$, while it will be $\log_2 P$ on a $NN(p,P)$, since the same link cannot be used in two opposite directions at the same time.
A different way to evaluate those networks is to give the complexity of some specific communication function. We just recall here a few results that will be needed in the following, the shift and gather-scatter transfers, and refer to [6,14,18,25] for a detailed presentation. Those evaluations will be given using the two parameters t_{start} and t_{com}, where t_{start} represents the start-up time to send a message and t_{com} represents the elemental transfer time, i.e the inverse of the communication throughput of the link. With such a definition, the time to shift N data from one processor to any of its nearest neighbor is given by:

$$T_{shift} = t_{start} + N.t_{com}$$

The values t_{start} and t_{com} depend specifically on the machine design. For example, on the Intel iPSC1 hypercubes where the packet length is large (1K byte), or on the LCAP system [7], we have $t_{start} >> t_{com}$ and the start-up term is dominant, while on the FPS-T and Caltech [30] hypercubes, where the packet length is small (1 and 8 bytes respectively), we have $t_{start} \sim t_{com}$ leading to a dominant throughput term.
This shift function is used when only local to local communication is needed, which is the case with the multicolor domain decomposition methods. But if some global communication is needed, broadcast, multibroadcast, gather and scatter transfers can be usefull. In the following we present the gather-scatter algorithm, which is the only one we need in the modified multigrid method.
Gathering data consists of moving pieces of data of the same length from all processors to a particular one. This kind of transfer is used for example in the inner product computation when every processor computing partial inner products of its vector chunk , one must sum these partial results to get the global inner-product. Scattering data is the reverse operation in which a particular processor sends a different piece of information to every processor, and has therefore the same complexity. On a P-ring using a one-way round algorithm, the scatter algorithm is performed in $P - 1$ steps where at each step a different packet is sent in an assembly line manner beginning by the most distant one, giving complexity:

$$T_{scatter}(N) = (P-1)(t_{start} + \frac{N}{P} t_{com}) = d.t_{start} + \frac{P-1}{P} .N.t_{com}$$

Consistently on a NN(k,P), the same method can be applied where at each step $i, i = 0...k-1$, scatter algorithms are applied simultaneously to N/Q^{k-i} data on each Q-ring of the network in the i th direction, leading to the complexity:

$$T_{scatter}(N) = \sum_{i=0}^{k-1}(Q-1)\left(t_{start} + \frac{N}{Q^{k-i}}.t_{com}\right) = d.t_{start} + \frac{P-1}{P}.N.t_{com}$$

where d is the maximum distance k.(Q-1). Note that contrary to multibroadcast transfer, scatter and gather algorithms can be performed in a two-way round fashion, leading to the complexity:

$$T_{scatter}(N) = \sum_{i=0}^{k-1}(Q/2)\left(t_{start} + \frac{N}{Q^{k-i}}.t_{com}\right) = d.t_{start} + \frac{Q}{2(Q-1)}.\frac{P-1}{P}.N.t_{com}$$

where d is the maximum distance k. Q/2. For simplicity, in the following we assume the gather and scatter complexities are:

$$T_{scatter}(N) = d.t_{start} + N.t_{com} \tag{14}$$

4. Efficiency of multicolor domain decomposition

We introduce a 1-dimension decomposition as a slice or strip decomposition, a 2-dimension decomposition as a square or box decomposition and a 3-dimension decomposition as a cube decomposition of the s dimension problem , $s = 1...3$. In the following, as we are not interested in the mapping problem, we use the natural mapping and map the r-dimension decomposition into a NN(r,P), leading to the possible re-mapping of the initial NN(k,P) network into the NN(r,P).

4.1 Multicolor SOR methods. Let s be the physical domain dimension and r be the domain decomposition dimension, with $r \leq s$. Assuming the exchange of data between adjacent regions is reduced to the interface grid points, and a perfect load balance between processors, we define $V(N,P)$ as number of grid points to compute and $S(N,P)$ number of grid points to exchange. We have then:

$$V(N,P) = \frac{n^s}{Q^r} = \frac{N}{P} \tag{15}$$

and

$$S(N,P) = 2.r.\frac{n^{s-1}}{Q^{r-1}} \tag{16}$$

Note that our evaluation assumes a two-color red-black relaxation and that if extended to a c-color scheme $S(N,P)$ would become $c.r.\frac{n^{s-1}}{Q^{r-1}}$.

Following our definitions, at every step we have:

$$T_{cpu}(N,P) = arith(V(N,P)).t_{cpu}$$

and:

$$T_{io}(N,P) = 2.r.t_{start} + S(N,P).t_{com}$$

As the work per iteration is constant throughout the process and as hypothesis (1) holds, the speed-up is then given by:

$$Sp(N,P) = \frac{P}{1 + \frac{2.r.}{n}.\frac{t_{com}}{t_{cpu}}.Q + \frac{2.r}{N}.\frac{t_{start}}{t_{cpu}}.P} \tag{17}$$

This algorithm is very efficient since the function $b(N,P)$ is identical to zero due to the only local effect of communication, leading to a monotone increasing speed-up. The function $a(N,P)$ has two components. The first related to the start-up time is linear in P and decreases as N^{-1} with the

problem size The second, related to the communication throughput, is linear in Q and decreases as n^{-1}, which is due to the perimeter effect coming from the ratio of data to compute and data to exchange. It is obvious that for a start-up bound problem, the best efficiency is reached for the smallest domain decomposition dimension, while with a throughput bound problem, the best efficiency is reached with the highest dimension domain decomposition. Although this speed-up function has theoretically an asymptote we note that, if the start-up term can be neglected, a square decomposition decomposition leads for example to a non-asymptotic function increasing as $\sqrt{P}$. Having a slice decomposition, the function $a(N,P)$ is a linear function of P and is equal to:

$$a(N,P) = \left(\frac{2}{n}.\frac{t_{com}}{t_{cpu}} + \frac{2}{N}.\frac{t_{start}}{t_{cpu}}\right).P$$

In this paper, we do not study the effect of convergence checking and refer to [6,26] for a more detail study. A comparison between a linear control algorithm and experimental results is presented in [5] on the LCAP system [7] with start-up bound problems.

4.2 The standard parallel multigrid method. Let v_1 and v_2 be., respectively, the number of relaxation steps per grid i, $i = 0...\ell - 1$, in transition from the finer to the coarser grids and vice versa, and v the number of relaxations on the coarsest grid level ℓ, meaning the coarsest grid level is not solved directly, but using the same relaxation scheme as the smoothing part. In the following, we suppose the restriction and interpolation operators complexities are negligible compared to the smoothing parts and each coarse grid level has 2^s less grid points than the previous one. Then, assuming the previous hypotheses of the multicolor relaxation method, and defining $V(N,P,i)$ and $S(N,P,i)$ as the counterparts of $V(N,P)$ and $S(N,P)$ on each grid level, we have:

$$V(N,P,i) = \left(\frac{n}{2^i}\right)^s.\frac{1}{P} \tag{18}$$

and

$$S(N,P,i) = 2.r.\left(\frac{n}{2^i}\right)^{s-1}.\frac{1}{Q^{r-1}} \tag{19}$$

Restricting our study to V-cycle, and summing on each grid level, the computing and exchange times at each iteration step are then:

$$T_{cpu}(N,P) = \sum_{i=0}^{\ell-1}(v_1 + v_2)T_{cpu}(V(N,P,i)) + v.T_{cpu}(V(N,P,\ell))$$

and

$$T_{io}(N,P) = \sum_{i=0}^{\ell-1}(v_1 + v_2).T_{io}(S(N,P,i)) + v.T_{io}(S(N,P,\ell))$$

where $T_{cpu}(V(N,i,j))$ and $T_{io}(S(N,i,j))$ denote the computing and exchange times for each sub-domain of level j using i processors. Then, as hypothesis (1) holds, the speed-up of a V-cycle and ℓ-level multigrid method can be written:

$$Sp(N,P) = \frac{P}{1 + \left(\frac{2.r.}{n}.\frac{A(s-1,\ell)}{A(s,\ell)}.\frac{t_{com}}{t_{cpu}}\right).Q + \left(\frac{2.r}{N}.\frac{(v_1+v_2).\ell + v}{A(s,\ell)}.\frac{t_{start}}{t_{cpu}}\right).P} \tag{20}$$

with

$$A(s,\ell) = (v_1 + v_2).\frac{1 - 2^{-s\ell}}{1 - 2^{-s}} + v.\frac{1}{2^{s\ell}}$$

We note first this speed-up formulation is related, as the multicolor domain decomposition (17), to the case where the function $b(N,P)$ of (2) is $\equiv 0$, and therefore leads to a monotone increasing speed-up function coming from the only local communication effect. We will now consider two cases, depending on the number of grid level ℓ.

With complete multigrid methods, ℓ is of the order of $\log_2 n$, and as n is usually large, we have $2^{-s\ell} \sim N^{-1} \sim 0$, leading to the approximation:

$$A(s,\ell) \sim \nu_1 + \nu_2 . \frac{1}{1-2^{-s}}$$

and to the corresponding speed-up:

$$Sp(N,P) \sim \frac{P}{1 + \left(\frac{2.r}{n} . \frac{1-2^{-s}}{1-2^{-s+1}} . \frac{t_{com}}{t_{cpu}}\right).Q + \left(\frac{2.r}{N}\left(1-\frac{1}{2^s}\right)\left(\ell + \frac{\nu}{\nu_1+\nu_2}\right) . \frac{t_{start}}{t_{cpu}}\right).P} \tag{21}$$

It shows clearly the communication overhead of a ℓ-level multigrid method compared to the corresponding relaxation method, multiplies the start-up effect by a factor $\ell + \nu/(\nu_1 + \nu_2)$ while the throughput effect remains roughly unchanged, leading to a more start-up bound problem and to preferable small dimension domain decomposition.
With incomplete multigrid methods, where the number of level remains small, as in the original two-grid method, the number of relaxation on the coarsest grid ν has to be large to solve the equation exactly, we have then $(\nu_1 + \nu_2).\ell << \nu$, and the approximation:

$$A(s,\ell) \sim \nu . \frac{1}{2^{s\ell}}$$

leading to the corresponding speed-up:

$$Sp(N,P) \sim \frac{P}{1 + \left(2^{\ell} . \frac{2.r}{n} . \frac{t_{com}}{t_{cpu}}\right).Q + \left(2^{s\ell} . \frac{2.r}{N} . \frac{t_{start}}{t_{cpu}}\right).P} \tag{22}$$

We note this evaluation is exactly the speed-up of the relaxation method on the coarsest grid level. Thinking to more general cycles, as shown on Fig. 2, it is now obvious that the main drawback of multigrid methods being the relaxation steps on the coarsest grid, V-cycles should be preferred to W and F-cycles, at least speaking about relative speed-up and parallel efficiency. Identically, the complete multigrid method should be prefered to the incomplete one.

4.3 Complexity of modified multigrid method. A good strategy to minimize the effect of the coarsest grid level could be to perform this last step using one single processor [13]. Therefore the V- cycle method remains unchanged all the way down the ℓ first levels, then a gather transfer is done on the coarsest grid level prior to its computation by a single processor, and followed by a scatter transfer before the computation goes as previously all the way up. Its complexity is then:

$$T_{cpu}(N,P) = \sum_{i=0}^{\ell-1} (\nu_1 + \nu_2).T_{cpu}(V(N,P,i)) + \nu.T_{cpu}(V(N,1,\ell))$$

and

$$T_{io}(N,P) = \sum_{i=0}^{\ell-1} (\nu_1 + \nu_2).T_{io}(S(N,P,i)). + 2.T_{scatter}(V(N,P,\ell))$$

where $2.T_{scatter}(V(N,P,\ell))$ is the complexity of the sub-domains gather and scatter transfer before and after the resolution on the coarsest grid level. Obviously, this strategy will be more efficient under the condition:

$$\nu.T_{cpu}(V(N,1,\ell)) + 2.T_{scatter}(V(N,P,\ell)) \leq \nu.T_{cpu}(V(N,P,\ell)) + \nu.T_{io}(S(N,P,\ell))$$

Assuming $\ell = \log_2 n$ and $t_{start} >> t_{com}$ we can prove the following result:
If $d \geq \nu.r$, the modified multigrid is not worth compared to the standard one.
If $d < \nu.r$, the modified multigrid method is faster than the standard one, under the condition:

$$\frac{t_{start}}{t_{cpu}} \geq \frac{1}{2.\left(r - \frac{d}{\nu}\right)}$$

If $t_{start} = t_{com}$ the result is still valid with d + 1 instead of d.
It should be noted the result is not valid when $t_{start} < t_{com}$, which seems unlikely in reality.
In any cases, the speed-up of the modified complete multigrid method is given: by:

$$Sp(N,P) = \frac{P}{1 + c.Q + \left(\frac{2}{N}\left(1 - \frac{1}{2^s}\right)\left(r.\ell + \frac{d}{v_1 + v_2}\right)\frac{t_{start}}{t_{cpu}} + \frac{1}{N}.\frac{v}{v_1 + v_2} + \frac{2}{N}.\frac{t_{com}}{t_{cpu}}\right).P} \qquad (23)$$

where c is the coefficient of Q in the standard multigrid speed-up formulation (21). Note that the maximum distance d is a function of P given by (8 & 9) , so that the speed-up is not only a function of Q and P, but also of $k.Q^{k+1}$ leading to a non-monotone increasing function which comes from the global effect of gather-scatter transfers. With this kind of global communication problem, we remark the hypercube topology can be useful to reduce this maximum distance d, leading to a possible efficient algorithm for large value of P, which is impossible using a ring of processors.

Conclusion

The present study is an attempt to achieve a better understanding of the relationship between parallel algorithms and architectures. The above speed-up cannot be taken as certified quantitative results, since several hypotheses have been taken which restrict or ignore certain phenomena, but more as a qualitative approach. The speed-up formulation we have proposed, based on the functions $a(N,P)$ and $b(N,P)$, permits a formal classification of different algorithm behavior, but also provides a good way to understand the different overhead of a parallel computation. Function $a(N,P)$, which is a polynomial function in Q of degree less or equal to the network dimension, is related only to the local communication effects and implies a monotone increasing speed-up. Its asymptotic or non-asymptotic behavior has been also shown. Function $b(N,P)$, which is a polynomial function in Q of degree greater to the network dimension, is related to the global communication effects and implies a non-monotone speed-up with an optimal number of processors and a maximum speed-up beyond which it decreases. We have seen also that, on a distributed memory system using message passing, the communication overhead has two different components related to the through-put of the communication link and to the time to initialize a message. More precisely, the through-put term of the communication overhead is proportional to Rc the ratio of the elemental transfer time over the elemental arithmetic time, t_{com}/t_{cpu} , times the ratio of the communication and arithmetic complexities, $com(N,P)/arith(N,P)$. The start-up term is proportional to the ratio Rs of the time to initialize a message over the elemental arithmetic time, t_{start}/t_{cpu} , times the ratio of the distance that characterizes the communication transfers over the arithmetic complexity, $d/arith(N,P)$. This distance can be an horizontal distance measuring the diameter of the graph, as with global communication problem, and therefore related to $k.Q$ leading to a non monotone increasing speed-up. It can be also a vertical distance between the different grid levels, as with the complete multigrid method, leading to a still monotone increasing speed-up but with a stronger start-up term and a lower asymptote.
These shows clearly the importance of both architecture and algorithms parameters, leading to a "condition number", $Rs \times d/arith(N,P) + Rc \times com(N,P)/arith(N,P)$, which has to be as small as possible. We can therefore speak of a local or global problem, depending on the value of d, of a computation or communication bound problem, depending on the value of $com(N,P)/arith(N,P)$, and of a start-up or through-put bound problem, depending on the value of $Rs \times d/arith(N,P)$ compared to $Rc \times com(N,P)/arith(N,P)$.

References

[1] Adams L., Ortega J. A multicolor SOR method for parallel computation. IEEE Proceedings of International Conference on Parallel Processing, 1982, pp. 53-56.
[2] Berger M., Bokhari S. A partitioning strategy for non uniform problems on multiprocessors. ICASE Report No. 85-55, ICASE NASA Langsley Research Center, Hampton Va., 1985.
[3] Brandt A. Multigrid solvers on parallel computers. Elliptic problems solvers. Schultz M. Ed., Academic Press, New York, 1981, pp 39-84.
[4] Brochard L. Domain decomposition and relation methods. Parallel Algorithms and Architectures, Cosnard M. Ed., Elsevier Sc. Publ., North-Holland, 1986, pp. 61-72.

[5] Brochard L. Communication and control costs of domain decomposition on loosely coupled multiprocessors. Proceedings of First International Conference on Supercomputing '87, Polychronopoulos C. Ed., Lecture Notes in Computer Science, Springer-Verlag, New York, to appear.
[6] Brochard L. Efficiency of some numerical algorithms on distributed systems. Research Report CERIA. Ecole Nationale des Ponts et Chaussees, Paris, 1987.
[7] Clementi E., Detrich J. Large scale paralll computing on LCAP systems. Experimental parallel computing architectures. Dongarra J. Ed., Elsevier Sc. Publ., Amsterdam, 1986, pp. 141-176.
[8] Chan T., Schreiber R. Parallel networks for multi-grid algorithms: architectue and complexity. Siam J. Sci. Comput., Vol 6, No 3, 1985, pp 698-713.
[9] First International Symposium on Domain Decomposition Methods for Partial Differential Equations. Paris 1987. Glowinski, Golub, Meurant, Periaux Ed., SIAM 1988.
[10] Dongarra J.J., Sorensen D.C. A fully parallel algorithm for the symmetric eigenvalue problem. SIAM J. Sci. Stat. Comp. Vol 8, No 2 , 1987, s 139-154.
[11] Fox G.C., Otto S.W., Hey A.J. Matrix algortm hypercube 1: Matrix multiplication. Parallel Computing. Vol.4, No. 2, 1987, pp. 17-32.
[12] Gannon D., Van Rosendale J. On the structure of parallelism in a highly concurrent PDE solver. Journal of Parallel and Distributed Computing, 3, pp 106-135, 1986.
[13] Hempel R. Parallel implementations of the multigrid method. The SUPRENUM project, ICCIAM, Paris, 1987.
[14] Ho C.T., Johnsson S.L. Distributed routing algorithms for broadcasting and personalized communication in the hypercube. Technical Report, YALEU/DCS/483, 1986.
[15] Hockney R.W., Jesshope C.R. Parallel computers, Adam Hilger, Bristol, 1981.
[16] Hoppe H.C., Muhlenbein H. Parallel adaptative full-multigrid methods on message based multiprocessors. Parallel Computing Vol 3, 1986, pp 269-287.
[17] Hwang K., Briggs F. Computer architecture andparallel processing. Mc. Graw Hill, 1984.
[18] Ipsen I.C., Saad Y., Schultz M.H. Complexity of linear system solution on a multiprocessor ring. Research report YALEU/DCS/349, 1985.
[19] Keyes D., Gropp W. A comparison of decomposition technique for elliptic partial differential equations and their parallel implementation. SIAM J. Sci. Stat. Comp. Vol 8, No 2, 1987, s 166-202.
[20] Lo S.S., Philippe B. The symmetric eigenvalue problem on a multiprocessor.
Algorithms and Architectures, Cosnard M. Ed. Elsevier Sci. Publ., North-Hollond, 1986, pp. 31-43.
[21] O'Leary D.P. Ordering schemes for parallel of processing on certain mesh problems. SIAM J. Sci. Stat. Vol 5, No 3, 1984, pp 620-632.
[22] Ortega J.M., Voigt R.G. Solutions of diferential equations on vector and parallel computers. SIAM Review, Vol. 27, No. 2, 1985, pp. 149-240.
[23] Parkinson D. Parallel efficiency can be greater than unity. Parallel Computing, Vol. 3, No. 3, 1986, pp 261-262.
[24] Saad Y. Gaussian eliminations on hypercubes. Algorithms and Architectures, Cosnard M. Ed., Elsevier Sci. Publ., North-Holland, 1986, pp. 5-19.
[25] Saad Y, Schultz H. Data communication in hypercubes. Research report YALEU/DCS/428, 1985.
[26] Saltz J., Naik V., Nicol D. Reduction of the effect of the communication delays in scientific algorithms on message passing MIMD architecture. ICASE Report No. 86-4, ICASE NASA Langsley Research Center, Hampton Va., 1986.
[27] Schwarz H.A. Uber einige Abbildungsaufgaben. Ges. Math. Abh. Vol 11, 1869, pp 65-83.
[28] Schwartz J.T. Ultracomputers. ACM Transaction on programming languages and systems, Vol. 2, No. 4, 1980, pp. 484-521.
[29] Stuben K., Trottenberg U. Multigrid Fundamental algorithm, model problem analysis and application. Proceedings of the Multigrid Methods Conference, Koln 1981, Lectures Notes in Mathematics No. 960, Springer-Verlag, Berlin, 1982, pp. 1-176.
[30] Thole C.A Experiments with multigrid methods on the Caltech hypercube. GMD Studieren Nr.103, St. Augustin, 1985.
[31] Young D. Iterative solution of large linear system. Academic Press, New York, 1971.