ELSEVIER Parallel Computing 22 (1996) 57–74

PARALLEL COMPUTING

A performance model for Krylov subspace methods on mesh-based parallel computers

E. de Sturler [*]

Interdisciplinary Project Center for Supercomputing (IPS-ETH Zurich), Swiss Federal Institute of Technology, ETH Zentrum, CH-8092 Zurich, Switzerland

Received 15 August 1994; revised 18 October 1995

Abstract

We develop a performance model for Krylov subspace methods implemented on distributed memory parallel computers for which the underlying communication network is a two-dimensional mesh. The model is based on the runtime of a single iteration or cycle of iterations (for methods like GMRES(m)), because the iteration count is problem dependent. Moreover, we intend to use the model only for parallel implementations that do not change the mathematical properties of the method (significantly). The main purpose of this model is a qualitative analysis of the performance; the model is not meant for very accurate predictions.

We express the efficiency, speed-up, and runtime as functions of the number of processors scaled by the number of processors that gives the minimal runtime for the given problem size ($P_{\max}$). This provides a natural way to analyze the performance characteristics for the range of the numbers of processors that can be used effectively. The approach is particularly interesting because it turns out that the performance is characterized completely by the sequential runtime and $P_{\max}$. The efficiency as a function of the number of processors relative to $P_{\max}$ is independent of the problem size and parameters describing the machine and solution method. Analogous relations can be obtained for the speed-up and runtime. $P_{\max}$ itself, of course, depends on N and the other parameters, and a simple equation for $P_{\max}$ is given.

The performance model is also used to evaluate the improvements in the performance if we reduce the communication as described in [7,9,8]. Although the scope of the performance model is limited by assumptions on the load balance and the processor grid, there are several obvious generalizations. One important and straightforward generalization is to higher dimensional meshes. We will discuss such generalizations at the end of this article.

[*] Email: sturler@ips.id.ethz.ch

Keywords: Linear algebra; Linear systems of equations; Sparse matrix; Krylov subspace method; Iterative methods; Distributed memory multiprocessors

1. Introduction

For the solution of linear systems of equations where the matrix is (very) large and sparse Krylov subspace methods [13,18,2] are among the most frequently and most successfully used iterative solution methods. These methods are specifically interesting for parallel computers, because most of the computations are easily parallelized. Moreover, these methods are used in particular for very large problems, which in turn are often solved on parallel computers.

Krylov subspace methods basically consist of four major kernels: a matrix-vector product, a preconditioner, a vector update, and an inner product (dot product). For many problems, especially those arising from the discretization of a continuous problem on a computational grid, parallel implementations are based on a domain decomposition approach. In this approach, the communication in the matrix-vector product involves only the exchange of data between neighbouring subdomains leading to communication between nearby processors, whereas the communication in the inner products is global (a reduction operation over all processors). The communication in the preconditioner depends entirely on the choice of preconditioner, so that we cannot make any statements about this. However, for obvious reasons on parallel computers one tends to use preconditioners with modest communication costs (e.g. comparable to the matrix-vector product). Since the performance is dominated either by computation (for a small number of processors) or by global communication (for a large number of processors), as described in [8], we will ignore the communication cost in the matrix-vector product and the preconditioner. For problems where global communication is necessary in the matrix-vector product, this communication will have the same kind of influence on the performance as the global communication in the inner products.

This performance model will show the dramatic influence of global communication in inner products on the performance of Krylov subspace methods (see also [6], where the first part of our performance model was introduced). The high communication overhead in inner products has received much attention, and there are several variants of Krylov subspace methods that might alleviate the problem (some of them were proposed for different reasons) [7, 9, 8, 3, 5, 4, 17]. We will model the standard implementations of GMRES(m) and CG and also the restructured versions proposed in [7, 9, 8], which aim for a reduction of the communication cost. Furthermore, this performance model also provides useful insight in the scalability of Krylov subspace methods on parallel computers.

Although the measured results in this article are based on a 400-processor Parsytec Supercluster, we believe this model or its extension to higher dimensional meshes (tori) to be useful for more modern architectures as well. Among new parallel computers that use a mesh or torus as their basic communication network are the Intel Paragon (2D), the Cray-T3D (3D), and the Tera (3D) [11].

2. The performance model

We will model the runtime of one iteration or one cycle of iterations of Krylov subspace methods on a two-dimensional mesh- or torus-based processor grid using the sum of the computation time and the communication time; see [8]. For P processors and N unknowns we define $T_{comp,P}$ to be the time spent in computation, which we approximate by

$$T_{comp,P} = \frac{f(m)N}{P}, \tag{1}$$

and we define $T_{comm,P}$ to be the time spent in communication, which we approximate by

$$T_{comm,P} = 0, \qquad \text{for } P = 1, \tag{2}$$

$$T_{comm,P} = g(m)\sqrt{P}, \qquad \text{for } P \geq 2. \tag{3}$$

Here, $f(m)$ and $g(m)$ are functions that depend on the Krylov subspace method used. The parameter m indicates the number of iterations within a cycle for methods like GMRES(m); for methods like CG $f(m)$ and $g(m)$ are constant functions. The factor $\sqrt{P}$ in the communication time is derived from the diameter of the graph underlying the processor grid, which for a two-dimensional mesh or torus is $O(\sqrt{P})$. For a d-dimensional mesh or torus this would be $O(P^{1/d})$, and for more general topologies we could use a more general function $d(P)$. The diameter of the processor grid emerges inherently in the performance evaluation if global communication of data is necessary. For examples of the derivation of expressions for $f(m)$ and $g(m)$ see [6, 8]. In Table 1 we give the expressions for $f(m)$ and $g(m)$ for CG [13, 10] (the expressions were derived from the latter reference), CGS [19], BiCGSTAB [20], GMRES(m) [18], and ORTHOMIN(m) [12]. In these expressions we assumed the preconditioner to be a local (M)ILU preconditioner.

We will now describe the performance model. From this model we will derive several expressions, and we will discuss their implications. The model assumes perfect load balance and a square processor grid. However, these restrictions are

Table 1
The functions $f(m)$ and $g(m)$ for different Krylov subspace methods. The parameter m indicates the number of iterations within a cycle for methods like GMRES(m), n_z is the average number of non-zero coefficients per row of the matrix, t_{fl} is the average time for one double precision floating point operation, t_s is the start-up time for nearest neighbour communication, and t_w is the nearest neighbour transfer time for one word

	$f(m)$	$g(m)$
CG	$(9+4n_z)t_{fl}$	$6(t_s+3t_w)$
CGS	$(20+8n_z)t_{fl}$	$6(t_s+3t_w)$
BiCGSTAB	$(22+8n_z)t_{fl}$	$10(t_s+3t_w)$
GMRES(m)	$2(m^2+3m+2n_z(m+1))t_{fl}$	$(m^2+3m)(t_s+3t_w)$
ORTHOMIN(m)	$2(m^2+6m+2n_z(m+1))t_{fl}$	$(m^2+5m)(t_s+3t_w)$

easily relieved by changing the number of unknowns into the number of processors times the maximum number of unknowns over the processor grid and by changing the factor $\sqrt{P}$ to reflect the maximum distance over the processor graph more accurately. Besides that, this model gives a lower bound on the performance of an efficient implementation. For our analysis this is the most important.

Assumption 1. *For Krylov subspace methods on square processor grids we assume that the runtime of one iteration or one cycle of m iterations for P processors and N unknowns is given by*

$$T_1 = f(m)N,$$
$$T_P = \frac{f(m)N}{P} + g(m)\sqrt{P}, \quad \text{for } P \geq 2, \tag{4}$$

where $f(m)$ and $g(m)$ depend on the specific Krylov method and are independent of either N or P (see Table 1 for some examples).

Definition 1. *We define the parallel speed-up S_P for P processors as*

$$S_P = \frac{T_1}{T_P}, \tag{5}$$

and we define the parallel efficiency E_P for P processors as

$$E_P = \frac{T_1}{PT_P}. \tag{6}$$

With the previous assumption and definitions we can state the following theorem.

Theorem 1. *The number of processors $P_{\max}$ for which T_P is minimal, and hence for which S_P is maximal, is given by*

$$P_{\max} = \left(\frac{2f(m)N}{g(m)} \right)^{\frac{2}{3}}. \tag{7}$$

For $P = \alpha P_{\max} \equiv P_\alpha$, with $\alpha \in [1/P_{\max}, 1]$, and $E_\alpha \equiv E_{P\alpha}$, $S_\alpha \equiv S_{P_\alpha}$, $T_\alpha \equiv T_{P_\alpha}$, we have that the parallel efficiency, parallel speed-up, and parallel runtime are given by

$$E_\alpha = \frac{1}{1 + 2\alpha^{\frac{3}{2}}}, \tag{8}$$

$$S_\alpha = \frac{\alpha}{1 + 2\alpha^{\frac{3}{2}}} P_{max}, \tag{9}$$

$$T_\alpha = \left(\frac{1 + 2\alpha^{\frac{3}{2}}}{2^{\frac{2}{3}}\alpha} \right) \left(g(m)^2 f(m)N \right)^{\frac{1}{3}}. \tag{10}$$

Proof. Eq. (7) can be derived by minimizing T_P (4) for P. We can derive (8) by first substituting for T_P in (6), and then substituting for $P = \alpha P_{\max} =$

$\alpha((2f(m)N)/g(m))^{2/3}$. We can now derive (9) from (5) and (6): $S_P = T_1/T_P = PE_P = \alpha E_\alpha P_{\max}$. Finally, (10) is proved using (5): $T_P = T_1/S_P$; which implies $T_\alpha = T_1/S_\alpha$. □

The definition of $P_{\max}$ defines the range of the numbers of processors that can be used effectively for a given number of unknowns. The introduction of the parameter α, in fact a relative number of processors, leads to expressions for runtime, efficiency, and speed-up that allow us to study the performance in a more general way. Theorem 1 gives rise to some interesting observations.

Expressions (7) and (10) describe the scalability of the algorithms. For increasing problem size N the number of processors that minimizes the runtime, $P_{\max}$, increases only as $O(N^{2/3})$, and so the minimum runtime increases necessarily as $O(N^{1/3})$. This means that for a sufficient increase in N the runtime increases as well, *irrespective of how many processors are used*. Therefore, it is not possible to keep the runtime constant for increasing N by increasing the number of processors P sufficiently, which is sometimes referred to as perfect scalability. However, the algorithms scale well in the sense that the minimum runtime increases only slowly as a function of the problem size. For $\alpha < 1$ it is possible to achieve an additional reduction in runtime. For increasing N, we increase the number of processors P more than proportional to $P_{\max}$; hence we increase α and so trade efficiency against performance. However, this is possible only for a limited increase in N. For a sufficient increase in N the situation $\alpha = 1$ will be reached, and then increasing P faster than $P_{\max}$ will actually yield a suboptimal performance (see Figure 4).

Another observation is that E_α (cf. (8)) is independent of N, $f(m)$, and $g(m)$. Moreover, the relative speed-up $S_\alpha/P_{\max}$ and the relative runtime $T_\alpha/T_{P_{\max}}$ as functions of α, are independent of N, $f(m)$, and $g(m)$ as well. *Therefore, the performance is characterized completely by T_l and $P_{\max}$*. So, the behaviour of the efficiency, relative speed-up, and relative runtime that is observed for increasing α,

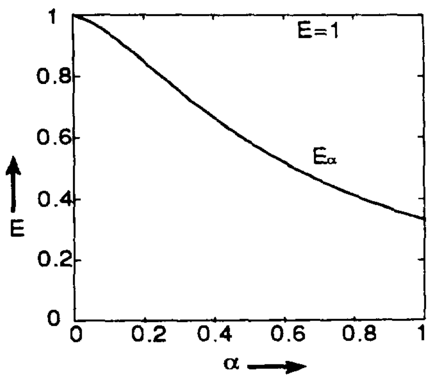

Fig. 1. The parallel efficiency E_α.

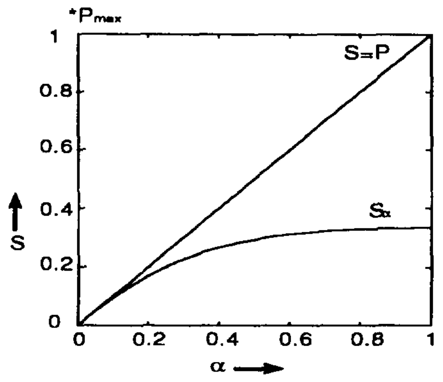

Fig. 2. The parallel speed-up S_α compared with perfect speed-up $S = P = \alpha P_{max}$.

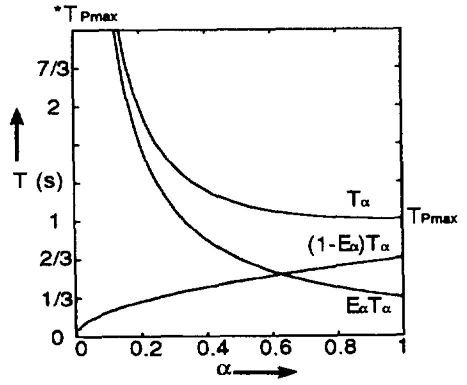

Fig. 3. The runtime T_α, the computation time $E_\alpha T_\alpha$ and the communication time $(1 - E_\alpha)T\alpha$.

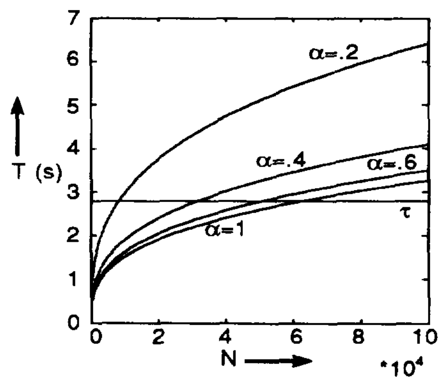

Fig. 4. Increasing runtimes for every fixed value of α.

see Figs. 1–4, will be observed independent of the problem size and of $f(m)$ and $g(m)$; that is, independent of the specific solver and of machine parameters. Obviously, the same value of α indicates different numbers of processors for different values of N, $f(m)$, and $g(m)$, because $P_{\max}$ depends on these parameters.

In Figs. 1, 2 and 3, E_α, S_α and T_α are given as functions of α. The units on the dependent axis are given such that the figures are valid independent of the parameters, N, $f(m)$, and $g(m)$.

Fig. 2 shows the deterioration of the speed-up for higher values of α. We see that for $\alpha \approx 0.6$ we have almost reached the maximum attainable speed-up, which itself is only $(1/3)P_{\max}$. This disappointing performance is entirely attributable to the global communication in the inner products, because this is the only source of overhead considered in the model. However, if other parts of the algorithm also need global communication, e.g. the matrix-vector product, this will have essentially the same effect.

The loss of performance for higher values of α is also illustrated well in Fig. 3. Since in our model all loss in efficiency is caused by global communication in the inner products, the time spent in communication is given by $(1 - E_\alpha)T_\alpha$, and the time spent in computation is $E_\alpha T_\alpha$. We see that for α larger than approximately 0.6 the runtime is dominated by the communication time. For $\alpha = 1$ the global communication accounts for 2/3 of the runtime.

Fig. 4 illustrates the scalability of Krylov subspace methods. For increasing N and fixed α the runtime increases slowly. We can keep the runtime fixed at τ for increasing N by increasing α as long as $\alpha < 1$. However, we cannot achieve runtimes below the curve $\alpha = 1$, because increasing α further would increase the runtime and decrease the speed-up.

Figs. 1–3 clearly indicate that the overhead in global communication for higher values of α severely limits the performance. In the following section we will model the improvements that can be obtained using the methods described in [7,9,8].

3. The effects of communication cost reduction

The performance model given in the previous section provides ample motivations for the reduction of the communication cost in Krylov subspace methods. In our performance model we will consider two ways to reduce the communication overhead. The first is to overlap communication with computation, and the second is to reduce start-up times by the collective communication of multiple messages. An important example of the latter is to replace the separate reductions and broadcasts of a group of partial, i.e. local, inner products by the reduction and broadcast of a vector of these partial inner products.

For CG we overlap the global communication in the inner products (a global reduction and a broadcast) with other operations like preconditioning and vector updates [9]. For GMRES(m) we first generate the basis vectors of the Krylov subspace using some polynomial for stability. After generating the basis, we can stepwise orthogonalize groups of vectors on one single vector, while still using a stable algorithm (modified Gram-Schmidt) [7]. This scheme offers the possibility to group the reduction and broadcast of the partial (i.e. local) inner products into the reduction and broadcast of a vector of partial inner products, which reduces the total amount of time spent in start-ups. However, we can reduce the communication cost even further as follows. Instead of computing all local inner products in a single group and do a reduction and broadcast once for the whole group, we split each orthogonalization step of a group of vectors onto one vector into two blocks. The overlap of communication with computation is then achieved by performing the reduction and broadcast of the local inner products of the first group concurrently with the computation of the local inner products of the second group and performing the reduction and broadcast of the local inner products of the second group concurrently with the vector updates of the first group [6].

We start with modeling the overlap of communication with computation. We assume that a fraction γ of the computations (in time) can be used to overlap communication. This leads to the following approximation for the runtime of a single iteration or cycle of m iterations.

Assumption 2. *For Krylov subspace methods on square processor grids, using a fraction γ of the computation time for overlapping communication time, we assume that the runtime of one iteration or one cycle of m iterations for P processors and N unknowns is given by*

$$T_1 = f(m)N,$$

$$\hat{T}_P = (1-\gamma)\frac{f(m)N}{P} + \max\left(\gamma\frac{f(m)N}{P}, g(m)\sqrt{P}\right), \quad \textit{for } P \geq 2, \tag{11}$$

where $f(m)$ and $g(m)$ depend on the specific Krylov method and are independent of either N or P (see Table 1 for some examples).

Definition 2. *We define the parallel speed-up $\hat{S}_P$ for P processors with overlapped communication as*

$$\hat{S}_P = \frac{T_1}{\hat{T}_P}, \tag{12}$$

and we define the parallel efficiency $\hat{E}_P$ for P processors with overlapped communication as

$$\hat{E}_P = \frac{T_1}{P\hat{T}_P}. \tag{13}$$

From the previous assumption on the runtime we can derive the number of processors for which the computation time used for overlap equals the communication time. We will refer to this number of processors as P_{ovl}:

$$P_{ovl} = \left(\frac{\gamma f(m)N}{g(m)}\right)^{2/3}. \tag{14}$$

As long as the number of processors is smaller than P_{ovl}, we have linear speed-up (in the model), because all communication is assumed to be overlapped; see Figs. 5–7. In order to compare the performance with overlap of communication with the performance without overlap of communication, we derive equations for the runtime, efficiency, and speed-up for numbers of processors relative to $P_{\max}$ (7).

Theorem 2. *The number of processors $\hat{P}_{\max}$ for which $\hat{T}_P$ is minimal, and hence for which $\hat{S}_P$ is maximal, is given by*

$$\hat{P}_{\max}\begin{cases}\left(\dfrac{2(1-\gamma)f(m)N}{g(m)}\right)^{\frac{2}{3}}, & \text{for } \gamma\le\frac{2}{3},\\[2ex] \left(\dfrac{\gamma f(m)N}{g(m)}\right)^{\frac{2}{3}}, & \text{for } \gamma>\frac{2}{3}.\end{cases} \tag{15}$$

For $P=\alpha P_{\max}\equiv P_\alpha$, with $\alpha\in[1/P_{\max},1]$, and $\hat{E}_\alpha\equiv\hat{E}_{P\alpha}$, $\hat{S}_\alpha\equiv S_{P\alpha}$, $\hat{T}_\alpha\equiv\hat{T}_{P\alpha}$, we have that the parallel efficiency, parallel speed-up, and parallel runtime are given by

$$\hat{E}_\alpha=\begin{cases}1, & \text{for } \alpha\le\left(\dfrac{\gamma}{2}\right)^{2/3},\\[2ex] \dfrac{1}{1+2\alpha^{3/2}-\gamma}, & \text{for } \alpha>\left(\dfrac{\gamma}{2}\right)^{2/3},\end{cases} \tag{16}$$

$$\hat{S}_\alpha=\begin{cases}\alpha P_{\max}, & \text{for } \alpha\le\left(\dfrac{\gamma}{2}\right)^{2/3},\\[2ex] \dfrac{\alpha}{1+2\alpha^{3/2}-\gamma}P_{\max}, & \text{for } \alpha>\left(\dfrac{\gamma}{2}\right)^{2/3},\end{cases} \tag{17}$$

$$\hat{T}_\alpha=\begin{cases}\left(\dfrac{1}{2^{2/3}\alpha}\right)\left(g(m)^2f(m)N\right)^{1/3}, & \text{for } \alpha\le\left(\dfrac{\gamma}{2}\right)^{2/3},\\[2ex] \left(\dfrac{(1-\gamma)+2\alpha^{3/2}}{2^{2/3}\alpha}\right)\left(g(m)^2f(m)N\right)^{1/3}, & \text{for } \alpha>\left(\dfrac{\gamma}{2}\right)^{2/3}.\end{cases} \tag{18}$$

Proof. This proof largely follows the proof of Theorem 1. Eq. (15) is derived by minimizing $\hat{T}_P$ (11) for P. For $\gamma\ge 2/3$ we have $\hat{P}_{\max}=P_{ovl}$. The efficiency, $\hat{E}_P=T_1/P\hat{T}_P$, is given by

$$\hat{E}_P=\frac{f(m)N}{P(f(m)N/P)}=1,\quad \text{for } P\le P_{ovl},$$

$$\hat{E}_P=\frac{f(m)N}{P\left((1-\gamma)\dfrac{f(m)N}{P}+g(m)\sqrt{P}\right)}=\frac{f(m)N}{(1-\gamma)f(m)N+g(m)P^{3/2}},$$

for $P>P_{ovl}$.

Substitution of $P = \alpha P_{\max}$ for the case $P > P_{\mathrm{ovl}}$ gives

$$\hat{E}_\alpha = \frac{1}{1 + 2\alpha^{3/2} - \gamma}, \quad \text{for } P > P_{ovl}.$$

Furthermore, we can derive from (7) and (14) that $P_{ovl} = (\gamma/2)^{2/3} P_{\max}$. This proves (16). We can derive (17) from $\hat{S}_\alpha = P\hat{E}_\alpha = \alpha \hat{E}_\alpha P_{\max}$. Eq. (18) for $\hat{T}_\alpha$ can be derived from $\hat{T}_\alpha = T_1/\hat{S}_\alpha$. □

We see from (15) that $\hat{P}_{\max} \le P_{\max}$; in fact, $\hat{P}_{\max}$ can be considerably smaller than $P_{\max}$, as is indicated by the following equation that also shows the values of α that correspond to $\hat{P}_{\max}$.

$$\hat{P}_{\max} = \begin{cases} (1-\gamma)^{2/3} P_{\max}, & \text{for } \gamma \le (2/3), \\ \left(\frac{\gamma}{2}\right)^{2/3} P_{\max}, & \text{for } \gamma > (2/3). \end{cases} \tag{19}$$

Moreover, the speed-up $\hat{S}_\alpha$ improves significantly when the communication is (partly) overlapped (17). This has two important consequences. The first and most obvious one is that we can solve problems faster. The second one, which is at least as important, is that problems that require a certain minimal performance to be tractable in practice can be solved on significantly fewer processors if the communication is overlapped. In short, with overlapping we can solve the same problem faster with fewer processors; see Figs. 5–7.

The efficiency, speed-up, and runtime for $\hat{P}_{\max}$ processors are given by

$$\hat{E}_{\hat{P}_{\max}} = \begin{cases} \frac{1}{3(1-\gamma)}, & \text{for } \gamma \le 2/3, \\ 1, & \text{for } \gamma > 2/3, \end{cases} \tag{20}$$

$$\hat{S}_{\hat{P}_{\max}} = \begin{cases} \frac{1}{3(1-\gamma)^{1/3}} P_{\max}, & \text{for } \gamma \le 2/3, \\ \left(\frac{\gamma}{2}\right)^{2/3} P_{\max}, & \text{for } \gamma > 2/3, \end{cases} \tag{21}$$

$$\hat{T}_{\hat{P}_{\max}} = \begin{cases} (1-\gamma)^{1/3}(2^{-2/3} + 2^{1/3})(g^2(m) f(m) N)^{1/3}, & \text{for } \gamma \le 2/3, \\ \gamma^{-2/3}(g^2(m) f(m) N)^{1/3}, & \text{for } \gamma > 2/3. \end{cases} \tag{22}$$

For $P \le P_{ovl}$ there is no loss of efficiency through communication, which results in linear speed-up and efficiency equal to one. In practice one cannot achieve this, but the comparison with experimental data in the next section shows that we can stay close to linear speed-up for a relatively large number of processors. Eqs. (14) and (15) show that $\hat{P}_{\max}$ and P_{ovl} get closer and closer for increasing γ, and for $\gamma \ge (2/3)$ we have $\hat{P}_{\max} = P_{ovl}$ and hence 'optimal' efficiency for the entire range of useful numbers of processors.

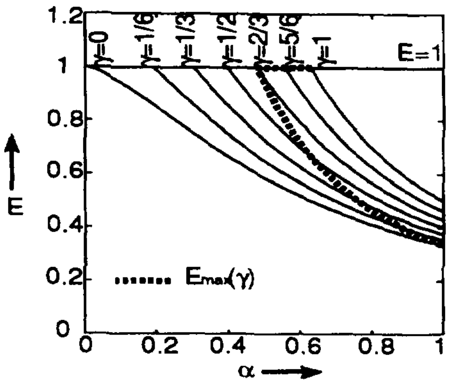

Fig. 5. The parallel efficiency $\hat{E}_\alpha$ for several values of γ in the case of overlapped communication.

The scalability of Krylov methods is not improved by overlapping the communication. The number of processors that minimizes the runtime, $\hat{P}_{\max}$, increases as $O(N^{2/3})$, so that the runtime still increases as $O(N^{1/3})$.

The efficiency, speed-up, and runtime as functions of α and γ are given in Figs. 5–7. Fig. 5 gives the efficiency as a function of α for different values of γ. The efficiencies for $\hat{P}_{\max}$ as a function of γ are indicated through the thick dotted line. We see that for $\alpha \le (\gamma/2)^{2/3}$ $(P \le P_{ovl})$ the efficiency is optimal, but that for larger α the efficiency decreases rapidly. For $\alpha = 1$ the efficiencies for the different values of γ differ only slightly.

This behaviour is also illustrated in Fig. 6 for the speed-up as a function of α and γ. As soon as the communication cannot be entirely overlapped $(P > P_{ovl})$ the speed-up hardly increases further, and for $\gamma \ge 2/3$ $(\hat{P}_{\max} = P_{ovl})$ the speed-up decreases immediately. The thick dotted line gives the maximum speed-ups for the different values of γ; see (21). The values of α that correspond to $\hat{P}_{\max}$ as a

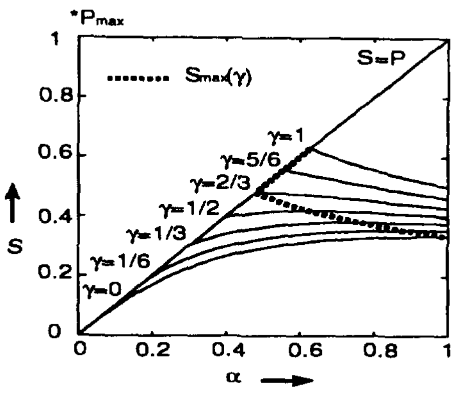

Fig. 6. The parallel speed-up $\hat{S}_\alpha$ for several values of γ in the case of overlapped communication.

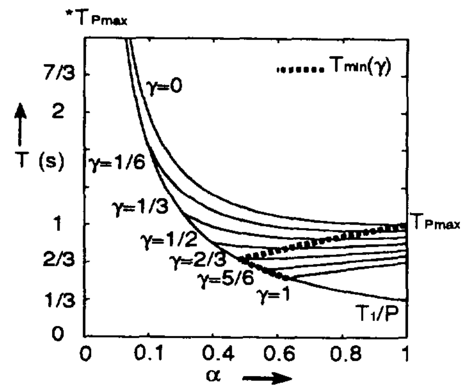

Fig. 7. The parallel runtime $\hat{T}_\alpha$ for several values of γ in the case of overlapped communication.

function of γ can be derived from (19). Fig. 6 shows clearly the increase of the maximum speed-up for a decreasing $\hat{P}_{\max}$ if $\gamma \leq 2/3$ and the communication is overlapped. $\hat{P}_{\max}$ increases again for $\gamma \geq 2/3$, but the speed-up remains optimal. This figure also shows that the maximum speed-up without overlap of communication ($\gamma = 0$) can be achieved with much fewer processors for even a moderate amount of overlap (e.g. $\gamma = 1/3$).

Finally, Fig. 7 gives the runtime for different values of γ and α. The thick dotted line gives the minimum runtimes for the different values of γ.

The second way of communication cost reduction that we consider is to reduce the communication time; that is, to make the function $g(m)$ smaller. An example of this is the collective accumulation of multiple inner products in GMRES(m) discussed in [7,8]. A smaller $g(m)$ does not change the model, so the effects of overlap remain the same if we combine the two approaches. Eqs. (7)–(10) indicate

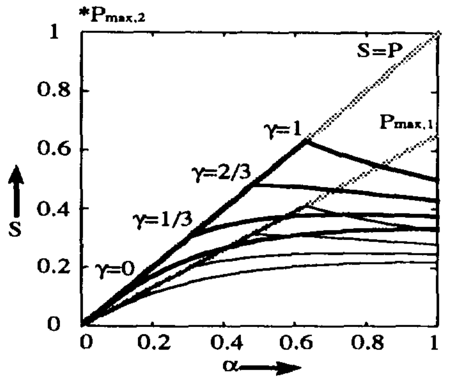

Fig. 8. The parallel speed-up in the case of overlapped communication for reduced $g(m)$, the thick lines, and for original $g(m)$, the thin lines, for several values of γ and α.

that $P_{\max}$ increases for decreasing $g(m)$, and that for a fixed value of α the speed-up increases proportionally with $P_{\max}$.

In Fig. 8 the thick lines represent the speed-up curves for different values of γ for an implementation with reduced $g(m)$ combined with overlapped communication, and for comparison the thin lines represent the speed-up curves for the original $g(m)$. The model parameters are derived from the GMRES(50) example in the next section. The two sets of curves have different numbers of processors corresponding to the same value of α, because the values for $P_{\max}$ differ. The ratio of the values of $P_{\max}$ gives the ratio of the numbers of processors for the two sets of curves.

4. Comparing the model with timings

In this section, we will compare the estimates from the model with measured performance values taken from [8]. This comparison seems to indicate that the model adequately predicts the performance. Moreover, especially for GMRES(50) the model seems relatively accurate. The measured results are derived from the solution of two model problems: a convection-diffusion problem solved with GMRES(50) [18] and a diffusion problem solved with CG [13]. The adapted versions of the algorithms are discussed in [7,9,8]. We solved both problems using local ILU(0) preconditioning [16,1,10] on a 400 processor, parallel distributed memory computer (a Parsytec Supercluster at the Koninklijke/Shell-Laboratorium Amsterdam). The values for the different parameters of the performance model are given in Table 2.

Table 3 shows the results of GMRES(50) for a standard implementation and for an implementation with reduced communication time and overlapped communication. The function $g(m)$ for the reduced communication cost is given by [8],

$$g(m) = 4mt_s + (2m^2 + 10m)t_w,$$

and the part of the computation time that can be overlapped with communication is given by ([8]),

$$(m^2 + 2m)t_{fl}\frac{N}{P}.$$

Table 2
The values of the parameters and their meaning

Parameter	Value	Meaning
t_w	4.80 μs	Communication transfer time for one word
t_s	5.30 μs	Communication start-up time
t_{fl}	3.00 μs	Average time for one double precision floating point operation
N	10000	Total number of unknowns
n_z	5	Average number of non-zero elements per row in the matrix
m	50	Size of the Krylov space over which GMRES(m) minimizes

Table 3
Comparison of estimated runtimes and measured runtimes for GMRES(50)

Processor grid	Standard implementation		Reduced $g(m)$ and overlapped communication	
	Estimated (s)	Measured (s)	Estimated (s)	Measured (s)
10×10	2.42	2.47	1.90	1.93
14×14	1.70	1.90	0.967	1.05
17×17	1.54	1.66	0.854	0.891
20×20	1.52	1.75	0.829	0.851

Using the parameters in Table 2, we find $\gamma = 0.41$, $P_{\max} \approx 375$ for the standard implementation, $P_{\max} \approx 576$ for the implementation with reduced $g(m)$, and $P_{ovl} \approx 201$. The sequential runtime $T_1 = 190$ s was obtained through estimation, since the problem did not fit on a single processor. The comparison of estimated and measured values for this particular example indicates that the model gives relatively good estimates if the assumptions with respect to the load balance and the processor grid (diameter $\approx \sqrt{P}$) are fulfilled. For a qualitative analysis this is more than sufficient. We see that for the standard implementation the runtime reduction stagnates quickly. For 100 processors the speed-up is about 77, for 196 processors it is only about 100, and for $P > 196$ processors ($\alpha > 0.5$) the runtime hardly decreases. For the implementation with overlapped communication we see an almost perfect speed-up going to 196 processors, which is to be expected because $P < P_{ovl}$. For $P > P_{ovl}$ the runtime hardly decreases any further.

Table 4 shows the results for CG for a standard implementation and for an implementation with overlapped communication. The part of the computation time that can be overlapped with communication is given by [8],

$$2n_z t_{fl} \frac{N}{P}.$$

Using the parameters in Table 2, we find $\gamma = 0.34$, $P_{\max} \approx 600$, and $P_{ovl} \approx 186$. The measured sequential runtime is given by $T_1 = 0.788s$. In this case the model is not as accurate as for GMRES(50). The reason for this is probably the costs that have

Table 4
Comparison of estimated runtimes and measured runtimes for CG

Processor grid	Standard implementation		Overlapped communication	
	Estimated (ms)	Measured (ms)	Estimated (ms)	Measured (ms)
10×10	9.88	10.7	8.70	10.2
14×14	6.09	6.90	4.58	5.84
17×17	5.02	6.09	3.99	5.29
20×20	4.54	5.59	3.80	5.04

been neglected, like the explicit implementation of (global) communication and perhaps the communication cost of the matrix-vector product. These, in principle, small costs do not show up so clearly in the GMRES(m) timings because that algorithm is much more expensive. This is also indicated by the fact that for both the standard implementation and overlapped implementation there is an almost constant difference between the measured and the estimated values.

However, the qualitative behaviour is modeled adequately. For the standard implementation we see that the reduction of the runtime stagnates for larger numbers of processors. Going from 100 to 196 processors we gain only some forty percent, and going from from 100 to 400 processors we reduce the runtime by about a factor of two. For the overlapped communication implementation we see that going from 100 to 196 processors the runtime reduction is almost optimal, which means that the efficiency is almost constant over this range. So, the increase of the communication cost has no influence, which is to be expected because $P \lesssim P_{ovl}$. The fact that the efficiency for 100 processors is about 90% and for 196 processors it is still about 90% indicates that some fixed overhead plays a role in the initial loss of efficiency. These costs seem to come mainly from the explicit implementation of the (global) communication. For the larger processor grids we have $P > P_{ovl}$, and the runtime reduction stagnates.

5. Other performance models

The model described in Section 2 (no overlap) could be represented within the framework of the model described by Hockney and Jesshope in [14] Section 1.3 with some changes in the interpretation of their parameters.

Let s be the total amount of work [1] ($s = Nf(k)/t_{fl}$), m the total amount of communication ($m = g(k)/2(t_s + 3t_w)$), $t_a(P)$ the average time for one floating point operation on a P-processor machine ($t_a(P) = t_{fl}/P$), and $t_c(P)$ be the time for one global communication on a P-processor machine ($t_c(P) = 2(t_s + 3t_w)P^{1/2}$). Then we can express the Hockney and Jesshope parameters for describing the performance as follows: the ideal computation rate $\hat{r}_\infty(P) = t_a^{-1}(P)$, the computational intensity (here better referred to as software granularity) $f = s/m$, and the half performance intensity (granularity) $f_{1/2} = t_c(P)/t_a(P)$. The parameters $\hat{r}_\infty(P)$, $f_{1/2}$, $t_a(P)$, and $t_c(P)$ describe the machine, and the parameters f, s, and m describe the program. We can express the sequential runtime as $T_1 = s/\hat{r}_\infty(P) = s \cdot t_a(P)$ and the parallel runtime on P processors as $T_P = s \cdot t_a(P) + m \cdot t_c(P)$. It is interesting to see that $f_{1/2} = O(P^{3/2})$ and $f = O(N)$, which indicates that the problem size has to increase proportional to $P^{3/2}$ to keep the efficiency constant: $E = pipe(f/f_{1/2}) = (1 + f_{1/2}/f)^{-1}$ (cf. the equation for $P_{\max}$ (7)).

[1] We use $f(k)$ and $g(k)$ here instead of $f(m)$ and $g(m)$ to avoid confusion with the total amount of communication m in the Hockney and Jesshope model. Also, our function $f(k)$ should not be confused with the computational intensity f.

The advantage of the model derived from [14] is that the performance model is separated in machine dependent properties, e.g. $f_{1/2} = O(P^{3/2})$, and program dependent properties, e.g. $f = O(N)$. However, there are several disadvantages. The model does not provide a clear insight in the relations between the performance, the problem size, and the number of processors, which is very important for our purposes. Furthermore, the extensions used for modeling overlap in [14] are not applicable to the type of overlap described in Section 3; indeed, the type of overlap that we want to model is not easily expressed in the model in [14]. The parameter $P_{\max}$, and the efficiency, runtime, and speed-up for a relative number of processors $P = \alpha P_{\max}$ are important concepts introduced in this paper (and partly in [6]). Especially the fact that the performance is completely characterized by the sequential runtime T_1 and $P_{\max}$ is important for the generality of our model. These concepts can be expressed in a model derived from [14], but the resulting expressions are not very insightful. We realize, of course, that the model in [14] was not proposed in particular for the algorithms and machines that we consider in this article.

There are some links of the model proposed here with the performance metrics discussed in [15] Chapter 4. However, there the main focus is on scalability and cost-optimality (isoefficiency), not on maximum speed-up as a function of the problem size. Also the concept of performance for a relative number of processors $P = \alpha P_{\max}$ is not considered there. It is interesting to note that the fixed α curves in Fig. 4 are isoefficiency curves, because the efficiency E_α depends only on α. However, this is a coincidence; for another architecture, say a hypercube, this would not be the case.

6. Conclusions

We have presented a simple performance model by concentrating on the main properties of the algorithms and architecture only. This limits the scope of the model, but it simplifies the analysis and helps to focus on the most important properties. Moreover, our test examples indicate that the qualitative behaviour of the performance is predicted adequately. Furthermore, the model is applicable to Krylov subspace methods in general, and it permits an analysis of the performance for varying numbers of processors that is independent of the problem size. In fact, the qualitative behaviour is completely characterized by the sequential runtime and the maximum number of processors that can be used effectively.

Several extensions of the model that would make it more general are straightforward. One such extension is to apply the model to parallel computers based on higher dimensional meshes or tori, which leads to essentially the same type of expressions. The runtime for $P > 1$ on a d-dimensional mesh could be approximated by

$$T_P = \frac{f(m)N}{P} + g(m)P^{1/d},$$

which would lead to a maximum number of processors of $O(N^{d/(d+1)})$ and a minimum runtime of $O(N^{1/(d+1)})$. This could be extended further to more general architectures by using a more general distance function for the maximum distance over the processor graph.

Acknowledgement

The author wishes to acknowledge Shell Research B.V. and STIPT for the financial support of his research.

References

[1] C. Ashcraft and R.G. Grimes, On vectorizing incomplete factorization and SSOR preconditioners, *SIAM J. Sci. Statist. Comput.*, 9 (1988) 122–151.

[2] R. Barret, M. Berry, T. Chan, J. Demmel, J. Dunato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. A. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* SIAM Publications, Philadelphia, PA, 1993.

[3] E.F. D'Azevedo and C.H. Romine, Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessors, Technical Report ORNL/TM-12192, Oak Ridge National Lab., 1992.

[4] A.T. Chronopoulos and C.W. Gear, s-Step iterative methods for symmetric linear systems, *J. on Comp. and Appl. Math.* 25 (1989) 153–168.

[5] A.T. Chronopoulos and S.K. Kim, s-Step Orthomin and GMRES implemented on parallel computers, Technical Report 90/43R, UMSI, Minneapolis, 1990.

[6] E. De Sturler, A parallel restructured version of GMRES(m). Technical Report 91-85, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1991.

[7] E. De Sturler, A parallel variant of GMRES(m), in: J.J.H. Miller and R. Vichnevetsky, editors, *Proc. of the 13th IMACS World Congress on Computation and Applied Mathematics*, pages 682–683, Dublin, Ireland, 1991. Criterion Press.

[8] E. De Sturler and H.A. Van der Vorst, Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers, Technical Report 832, Mathematical Institute, University of Utrecht, Utrecht, The Netherlands, 1993. Also *Applied Numerical Mathematics (IMACS)*, (accepted for publication).

[9] J.W. Demmel, M.T. Heath and H.A. van der Vorst, Parallel numerical linear algebra, *Acta Numerica*, 2, 1993.

[10] J.J. Dongarra, I.S. Duff, D.C. Sorensen, and H.A. Van der Vorst, *Solving Linear Systems on Vector and Shared Memory Computers* (SIAM Publications, Philadelphia, PA 19103-5052, USA, 1991).

[11] K.E. Gates and W.P. Petersen, A technical description of some parallel computers, *International Journal of High Speed Computing*, 6 (1994) 399–449.

[12] L.A. Hageman and D.M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.

[13] M.R. Hestenes and E. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Nat. Bur. Standards* 49 (1952) 409–436.

[14] R.W. Hockney and C.R. Jesshope, *Parallel Computers 2: Architecture, Programming and Algorithms, second edition* IOP Publishing Ltd, Bristol England, 1988.

[15] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms*. The (Benjamin/Cummings Publishing Company, Inc. Redwood City CA, 1994).

[16] J.A. Meijerink and H.A. Van der Vorst, An iterative solution method for linear equations systems of which the coefficient matrix is a symmetric M-matrix, *Math. Comp.*, 31 (1197) 148–162, 1977.
[17] G. Meurant, Numerical experiments for the preconditioned conjugate gradient method on the CRAY X-MP/2, Technical Report LBL-18023, University of California, Berkeley, CA, 1984.
[18] Y. Saad and M. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 7 (1986) 856–869.
[19] P. Sonneveld, CGS, a fast Lanczos-type solver for nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.* 10 (1989) 36–52.
[20] H.A. Van der Vorst, BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.*, 13 (1992) 631–644.