NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2011; **18**:805–825
Published online 14 January 2011 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nla.764

Minimizing synchronization in IDR(s)

Tijmen P. Collignon and Martin B. van Gijzen*, †, ‡

Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands

SUMMARY

IDR(s) is a family of fast algorithms for iteratively solving large nonsymmetric linear systems. With cluster computing and in particular with Grid computing, the inner product is a bottleneck operation. In this paper, three techniques are investigated for alleviating this bottleneck. First, a recently proposed IDR(s) algorithm that is highly efficient and stable is reformulated in such a way that it has a single global synchronization point per iteration step. Second, the so-called test matrix is chosen so that the work, communication, and storage involving this matrix is minimized in multi-cluster environments. Finally, a methodology is presented for a-priori estimation of the optimal value of s using only problem and machine-based parameters. Numerical experiments applied to a 3D convection–diffusion problem are performed on the DAS-3 Grid computer, demonstrating the effectiveness of our approach. Copyright © 2011 John Wiley & Sons, Ltd.

Received 25 February 2010; Revised 28 September 2010; Accepted 17 October 2010

KEY WORDS: iterative methods; numerical linear algebra; nonsymmetric linear systems; IDR(s); cluster and Grid computing; performance model

1. INTRODUCTION

The recent IDR(s) method and its derivatives are short recurrence Krylov subspace methods for iteratively solving large linear systems

$$Ax = b, \quad A \in \mathbb{C}^{N \times N}, \quad x, b \in \mathbb{C}^N, \tag{1}$$

where the coefficient matrix A is nonsingular and non-Hermitian [1]. The method has attracted considerable attention, e.g. see [2–10]. For $s = 1$, IDR(s) is mathematically equivalent to the ubiquitous Bi–CGSTAB algorithm [11]. For important types of problems and for relatively small values of $s > 1$, the IDR(s) algorithms outperform the Bi–CGSTAB method.

The goal of this paper is to construct an efficient IDR(s) algorithm for cluster and Grid computing. Global synchronization is a bottleneck operation in such computational environments and to alleviate this bottleneck, three techniques are investigated:

(i) The efficient and numerically stable IDR(s)-*biortho* method from [12] is reformulated in such a way that it has one global synchronization point per iteration step. The resulting method is named IDR(s)-*minsync*;
(ii) *A priori* estimation of the optimal parameter s and number of processors is performed to minimize the total computing time using only problem and machine-dependent parameters;
(iii) Piecewise sparse column vectors for the test matrix are used to minimize computation, communication, and storage involving this matrix in multi-cluster environments.

*Correspondence to: Martin B. van Gijzen, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands.
†E-mail: M.B.vanGijzen@TUDelft.nl
‡Associate Professor.

The target hardware consists of the distributed ASCI Supercomputer 3 (DAS-3), which is a cluster of five geographically separated clusters spread over four academic institutions in the Netherlands [13]. The DAS-3 multi-cluster is designed for dedicated parallel computing and although each separate cluster is relatively homogeneous, the system as a whole can be considered heterogeneous.

A parallel performance model is derived for computing the *a priori* estimations of the optimal parameter s. Extensive experiments on a 3D convection–diffusion problem show that the performance model is in good agreement with the experimental results. The model is successfully applied to both a single cluster and to the DAS-3 multi-cluster. Comparisons between IDR(s)-biortho and IDR(s)-minsync are made, demonstrating superior scalability and efficiency of the reformulated method IDR(s)-minsync.

Techniques for reducing the number of synchronization points in Krylov subspace methods on parallel computers have been studied by several authors [14–21]. With the advent of Grid computing, the need for reducing global synchronizations is larger than ever. The combination of the three strategies used in this paper results in a highly efficient iterative method for solving large nonsymmetric linear systems on Grid computers.

Note that the prototype method IDR(s)-*proto* from the original IDR(s) paper [1] also has a single global synchronization point per iteration step, except when computing a new ω each $s+1$st step, which requires two global synchronizations. However, the IDR(s)-biortho method exhibits superior numerical stability and has less floating point operations per IDR(s) cycle. Therefore, the IDR(s)-biortho method was used as a basis for the new IDR(s)-minsync method.

The following notational conventions, terminology, and definitions will be used in this paper. Let the matrix $Q_k = [q_1, q_2, \ldots, q_k]$. If not specified otherwise, the norm $\|\cdot\|$ denotes the 2-norm. Let $v \perp Q$ be shorthand for v is orthogonal to all column vectors of Q. The orthogonal complement to the column space of Q is denoted by $Q^{\perp}$. The number of iterations refers to the number of matrix–vector multiplications and the index n refers to the *iteration* number, while the index j always refers to the jth IDR(s) *cycle*. Note that one IDR(s) cycle consists of $s+1$ iteration steps. No preconditioner will be applied.

This paper is organized as follows. In Section 2 an IDR(s) variant with a single global synchronization point per iteration step is presented. Section 3 describes the parallel performance model that is used to estimate the optimal value of s. It also describes a technique of using piecewise sparse column vectors for the test matrix to minimize work involving this matrix on a multi–cluster. Section 4 contains extensive experimental results on the DAS-3, demonstrating the effectiveness of the three strategies. Concluding remarks are given in Section 5.

2. AN EFFICIENT IDR(s) VARIANT WITH MINIMAL SYNCHRONIZATION POINTS

The IDR(s)-based methods are new iterative algorithms for solving large nonsymmetric systems and much research is needed on efficient parallelization on distributed memory computers. When applying the so-called IDR(s) theorem to derive practical algorithms, certain choices can be made. This freedom allows for efficient tuning of the numerical algorithm to specific computational environments. In this section the highly stable IDR(s)-biortho method is reproduced and used as a basis for the IDR(s)-minsync method, which has a single global synchronization point per iteration step.

Given an initial approximation x_0 to the solution, all IDR(s) methods construct residuals r_j in a sequence $(\mathcal{G}_j)$ of shrinking subspaces that are related according to the following theorem.

Theorem 1 (*Induced Dimension Reduction (IDR)*)
Let $A \in \mathbb{C}^{N\times N}$, let $B \in \mathbb{C}^{N\times N}$ be a preconditioning matrix, let $Q \in \mathbb{C}^{N\times s}$ be a fixed matrix of full rank, and let $\mathcal{G}_0$ be any non-trivial invariant linear subspace of A. Define the sequence of subspaces $(\mathcal{G}_j)$ recursively as

$$\mathcal{G}_{j+1} \equiv (I - \omega_{j+1} A B^{-1})(\mathcal{G}_j \cap Q^{\perp}) \quad \text{for } j = 0, 1, \ldots, \tag{2}$$

where (ω_j) is a sequence in $\mathbb{C}^N$. If $Q^{\perp}$ does not contain an eigenvector of AB^{-1}, then for all $j \geqslant 0$

DOI: 10.1002/nla

- $\mathscr{G}_{j+1} \subset \mathscr{G}_j$;
- $\dim \mathscr{G}_{j+1} < \dim \mathscr{G}_j$ unless $\mathscr{G}_j = \{0\}$.

Ultimately, the residual is forced in the zero-dimensional subspace $\mathscr{G}_j = \{0\}$ for some $j \leqslant N$. For a proof the reader is referred to [1, 4].

Iterative algorithms based on the IDR(s) theorem consist of two separate steps, which constitute the jth cycle of an IDR(s) method (i.e. $s+1$ (preconditioned) matrix–vector multiplications):

1. The dimension reduction step: given s vectors in $\mathscr{G}_j$ and a residual $r_j \in \mathscr{G}_j$, a residual r_{j+1} in the lower-dimensional subspace $\mathscr{G}_{j+1} = (I - \omega_{j+1} A B^{-1})(\mathscr{G}_j \cap Q^{\perp}) \subset \mathscr{G}_j$ is computed after choosing an appropriate ω_{j+1};
2. Generating s additional vectors in $\mathscr{G}_{j+1}$.

It can be shown that in exact arithmetic, IDR(s) methods terminate within Ns^{-1} dimension reduction steps, or equivalently, within $N(1+s^{-1})$ (preconditioned) matrix–vector multiplications [1, Section 3]. In practical applications, the iteration process will exhibit much faster convergence rates according to $\widehat{N}s^{-1}$ IDR(s) cycles, where $\widehat{N} \ll N$.

Shown in Algorithm 1 is the (right) preconditioned IDR(s)-biortho variant [12], which not only has slightly less operations but is also numerically more stable than the IDR(s)-proto variant. The elements of the small $s \times s$ matrix M are denoted by $\mu_{i,j}$ for $1 \leqslant i,\ j \leqslant s$ and M is initially set to the identity matrix. The dimension reduction step (i.e. lines 32–37 in Algorithm 1) consists of one preconditioned matrix–vector product, two vector updates, and two inner products. Combined with the operations for constructing s vectors in $\mathscr{G}_j$ (i.e. lines 6–31 in Algorithm 1), this amounts to $s+1$ preconditioned matrix–vector products, $s(s+1)+2$ inner products, and $2(s(s+1)+1)$ vector updates per cycle of IDR(s). The computation of the (combined and separate) inner products is highlighted by boxes, which shows that there are $\frac{1}{2}(s(s+1))+2$ global synchronization points per IDR(s) cycle.

In the IDR(s)-biortho method, certain bi-orthogonality conditions with the columns of Q are enforced that result in improved numerical stability and in reduced number of vector operations compared with IDR(s)-proto. To be more specific, let r_{n+1} be the first residual in $\mathscr{G}_{j+1}$. In IDR(s)-biortho, vectors for $\mathscr{G}_{j+1}$ are made to satisfy

$$g_{n+k} \perp q_i, \quad i = 1, \ldots, k-1, \ \ k = 2, \ldots, s, \tag{3}$$

and intermediate residuals are made to satisfy

$$r_{n+k+1} \perp q_i, \quad i = 1, \ldots, k, \ \ k = 1, \ldots, s. \tag{4}$$

In the implementation presented in Algorithm 1, these conditions are enforced using a modified Gram–Schmidt (MGS) process for oblique projection (lines 15–24 in Algorithm 1). The disadvantage of this approach is that the inner products cannot be combined, which poses a bottleneck in parallel computing environments. By using a classical Gram–Schmidt (CGS) process for these projections, this bottleneck can be alleviated. The general idea is that *all* the inner products can be recursively computed with a one-sided bi-orthogonalization process using solely *scalar updates*. For a more detailed discussion on using either CGS or MGS for the oblique projections, see [3].

Shown in Algorithm 2 is the reformulated variant IDR(s)-minsync. In the following, the two phases of the new IDR(s) variant are discussed separately, where we often specifically refer to line numbers of both Algorithms 1 and 2.

2.1. The dimension reduction step

In the following, let r_n be the last intermediate residual in $\mathscr{G}_j$ before the dimension reduction step. In accordance with condition (4), this residual is orthogonal to all the columns of Q (cf. r_{n+s+1} from (4)) and therefore we have $r_n \in \mathscr{G}_j \cap Q^{\perp}$. Using Theorem 1 the first residual r_{n+1} in $\mathscr{G}_{j+1}$ is thus computed as

$$r_{n+1} = (I - \omega_{j+1} A B^{-1}) r_n \quad \text{(line 35 of Algorithm 2)}. \tag{5}$$

DOI: 10.1002/nla

Algorithm 1 IDR(s)-biortho with bi–orthogonalisation of intermediate residuals.

INPUT: $A \in \mathbb{C}^{N\times N}$; $x, b \in \mathbb{C}^N$; $Q \in \mathbb{C}^{N\times s}$; preconditioner $B \in \mathbb{C}^{N\times N}$; parameter s; accuracy ε.
OUTPUT: Approximate solution x such that $\|b - Ax\| \leqslant \varepsilon$.

1: *// Initialisation*
2: Set $G = U = 0 \in \mathbb{C}^{N\times s}$; $M = [\mu_{i,j}] = I \in \mathbb{C}^{s\times s}$; $\omega = 1$
3: Compute $r = b - Ax$
4: *// Loop over nested $\mathcal{G}_j$ spaces, $j = 0, 1, \ldots$*
5: **while** $\|r\| \geqslant \varepsilon$ **do**
6: *// Compute s linearly independent vectors g_k in $\mathcal{G}_j$*
7: $\boxed{\phi = Q^H r}$, $\phi = (\phi_1, \ldots, \phi_s)^T$ *// s inner products (combined)*
8: **for** $k = 1$ to s **do**
9: Solve $M\gamma = \phi$ for γ, $\gamma = (\gamma_k, \ldots, \gamma_s)^T$
10: $v = r - \sum_{i=k}^{s} \gamma_i g_i$
11: $\widetilde{v} = B^{-1} v$ *// Preconditioning step*
12: $u_k = \sum_{i=k}^{s} \gamma_i u_i + \omega \widetilde{v}$
13: $g_k = A u_k$
14: *// Make g_k orthogonal to $q_1, \ldots, q_{k-1}$*
15: **for** $i = 1$ to $k-1$ **do**
16: $\boxed{\alpha = q_i^H g_k / \mu_{i,i}}$ *// $k-1$ inner products (separate)*
17: $g_k \leftarrow g_k - \alpha g_i$
18: $u_k \leftarrow u_k - \alpha u_i$
19: **end for**
20: *// Update column k of M*
21: $\boxed{\mu_{i,k} = q_i^H g_k}$ for $i = k, \ldots, s$ *// $s-k+1$ inner products (combined)*
22: *// Make the residual orthogonal to $q_1, \ldots, q_k$*
23: $\beta = \phi_k / \mu_{k,k}$
24: $r \leftarrow r - \beta g_k$
25: $x \leftarrow x + \beta u_k$
26: *// Update $\phi = Q^H r$*
27: **if** $k+1 \leqslant s$ **then**
28: $\phi_i = 0$ for $i = 1, \ldots, k$
29: $\phi_i = \phi_i - \beta \mu_{i,k}$ for $i = k+1, \ldots, s$
30: **end if**
31: **end for**
32: *// Entering $\mathcal{G}_{j+1}$, the dimension reduction step*
33: $\widetilde{v} = B^{-1} r$ *// Preconditioning step*
34: $t = A\widetilde{v}$
35: $\boxed{\omega = (t^H r)/(t^H t)}$ *// Two inner products (combined)*
36: $r \leftarrow r - \omega t$
37: $x \leftarrow x + \omega \widetilde{v}$
38: **end while**

Premultiplying this expression with A^{-1} results in the corresponding recursion for the iterate

$$x_{n+1} = x_n + \omega_{j+1} B^{-1} r_n \quad \text{(line 36 of Algorithm 2)}, \tag{6}$$

DOI: 10.1002/nla

Algorithm 2 IDR(s)-minsync with bi–orthogonalisation of intermediate residuals and with minimal number of synchronisation points.

INPUT: $A \in \mathbb{C}^{N\times N}$; $x, b \in \mathbb{C}^N$; $Q \in \mathbb{C}^{N\times s}$; preconditioner $B \in \mathbb{C}^{N\times N}$; accuracy ε.
OUTPUT: Approximate solution x such that $\|b - Ax\| \leqslant \varepsilon$.

1: *// Initialisation*
2: $G = U = 0 \in \mathbb{C}^{N\times s}$; $M_l = I \in \mathbb{C}^{s\times s}$, $M_t = M_c = 0$: see eq. (17); $\omega = 1$
3: Compute $r = b - Ax$
4: $\phi = Q^H r$, $\phi = (\phi_1, \ldots, \phi_s)^T$
5: *// Loop over nested $\mathcal{G}_j$ spaces, $j = 0, 1, \ldots$*
6: **while** $\|r\| > \varepsilon$ **do**
7: *// Compute s linearly independent vectors g_k in $\mathcal{G}_j$*
8: **for** $k = 1$ to s **do**
9: *// Compute $v \in \mathcal{G}_j \cap Q^\perp$*
10: Solve $M_l \gamma_{(k:s)} = \phi_{(k:s)}$
11: $v = r - \sum_{i=k}^{s} \gamma_i g_i$
12: $\widetilde{v} = B^{-1} v$ *// Preconditioning step*
13: $\widehat{u}_k = \sum_{i=k}^{s} \gamma_i u_i + \omega \widetilde{v}$ *// Intermediate vector $\widehat{u}_k$*
14: $\widehat{g}_k = A\widehat{u}_k$ *// Intermediate vector $\widehat{g}_k$*
15: $\boxed{\psi = Q^H \widehat{g}_k}$ *// s inner products (combined)*
16: Solve $M_t \alpha_{(1:k-1)} = \psi_{(1:k-1)}$
17: *// Make $\widehat{g}_k$ orthogonal to $q_1, \ldots, q_{k-1}$ and update $\widehat{u}_k$ accordingly*
18: $g_k = \widehat{g}_k - \sum_{i=1}^{k-1} \alpha_i g_i, \quad u_k = \widehat{u}_k - \sum_{i=1}^{k-1} \alpha_i u_i$
19: *// Update column k of M_l*
20: $\mu^l_{i,k} = \psi_i - \sum_{j=1}^{k-1} \alpha_j \mu^c_{i,j}$ for $i = k, \ldots, s$
21: *// Make r orthogonal to $q_1, \ldots, q_k$ and update x accordingly*
22: $\beta = \phi_k / \mu^l_{k,k}$
23: $r \leftarrow r - \beta g_k$
24: $x \leftarrow x + \beta u_k$
25: *// Update $\phi \equiv Q^H r$*
26: **if** $k + 1 \leqslant s$ **then**
27: $\phi_i = 0$ for $i = 1, \ldots, k$
28: $\phi_i \leftarrow \phi_i - \beta \mu^l_{i,k}$ for $i = k+1, \ldots, s$
29: **end if**
30: **end for**
31: *// Entering $\mathcal{G}_{j+1}$. Note: $r \perp Q$*
32: $\widetilde{v} = B^{-1} r$ *// Preconditioning step*
33: $t = A\widetilde{v}$
34: $\boxed{\omega = (t^H r)/(t^H t);\ \phi = -Q^H t}$ *// $s+2$ inner products (combined)*
35: $r \leftarrow r - \omega t$
36: $x \leftarrow x + \omega \widetilde{v}$
37: $\phi \leftarrow \omega \phi$
38: **end while**

which is essentially modified Richardson. A typical (minimal residual) choice of ω_{j+1} is $\arg\min_\omega \|(I - \omega A B^{-1}) r_n\|$ or equivalently

$$\omega_{j+1} = \frac{(AB^{-1} r_n)^H r_n}{(AB^{-1} r_n)^H AB^{-1} r_n}. \tag{7}$$

By reordering operations, the computation of ω_{j+1} can be combined with the computation of $\phi = Q^H r$ in line 7 from Algorithm 1 as follows. Premultiplying the recursion for computing the

DOI: 10.1002/nla

new residual (5) with Q^H gives

$$Q^H r_{n+1} = Q^H r_n - \omega_{j+1} Q^H A B^{-1} r_n \tag{8}$$

$$= -\omega_{j+1} Q^H A B^{-1} r_n, \tag{9}$$

since $Q^H r_n = 0$ by construction. Setting $t_n = AB^{-1} r_n$, the computation of $t_n^H r_n$, $t_n^H t_n$, and $Q^H t_n$ can then be combined (line 34 of Algorithm 2).

2.2. *Generating s additional vectors in* $\mathcal{G}_{j+1}$

In addition to the standard orthogonalization step performed in IDR(s) for computing a vector $v \in \mathcal{G}_j \cap Q^\perp$ (line 9–11 in Algorithm 2), the goal is to enforce the extra orthogonality conditions (3) and (4) to the newly computed vectors g and residuals r in $\mathcal{G}_{j+1} \subset \mathcal{G}_j$. In practice, this means that there are now essentially two main orthogonalizations that need to be performed. Since $\mathcal{G}_{j+1} \subset \mathcal{G}_j$, the two orthogonalizations use vectors g that are either in both $\mathcal{G}_{j+1}$ and $\mathcal{G}_j$ or only in $\mathcal{G}_j$.

In the following, let $k = 1, 2, \ldots, s$ and let $Q_k = [q_1, \ldots, q_k]$. Suppose that after $n+k$ iterations we have exactly $s-k+1$ vectors $g_i, i = n+k-s-1, \ldots, n-1$ in $\mathcal{G}_j$ and $k-1$ vectors $g_i, i = n+1, \ldots, n+k-1$ in $\mathcal{G}_{j+1}$, which gives a total of s vectors g_i. In addition, suppose that we have s corresponding vectors u_i such that $g_i = Au_i$ for all i.

From the dimension reduction step we have a residual $r_{n+1} \in \mathcal{G}_{j+1}$. In addition, let $\widehat{g}_{n+k} \in \mathcal{G}_{j+1}$. Then the two main orthogonalizations that need to be performed are

$$\begin{aligned} v_{n+k} &= r_{n+k} - \sum_{i=k}^{s} \gamma_i g_{n+i-s-1} \quad \in \mathcal{G}_j \cap Q^\perp \quad \text{('standard', line 11 of Algorithm 2),} \\ g_{n+k} &= \widehat{g}_{n+k} - \sum_{i=1}^{k-1} \alpha_i g_{n+i} \quad \in \mathcal{G}_{j+1} \cap Q_{k-1}^\perp \quad \text{(additional, line 18 of Algorithm 2).} \end{aligned} \tag{10}$$

The γ_i's are chosen such that $v_{n+k} \in \mathcal{G}_j \cap Q^\perp$ and the α_i's are chosen such that $g_{n+k} \in \mathcal{G}_{j+1} \cap Q_{k-1}^\perp$ (i.e. condition (3)).

The intermediate update $\widehat{u}_{n+k}$ for the iterate and the intermediate vector $\widehat{g}_{n+k} \in \mathcal{G}_{j+1}$ using explicit multiplication by A are computed according to

$$\begin{aligned} \widehat{u}_{n+k} &= \sum_{i=k}^{s} \gamma_i u_{n+i-s-1} + \omega_{j+1} B^{-1} v_{n+k} \quad \text{(line 13 of Algorithm 2),} \\ \widehat{g}_{n+k} &= A\widehat{u}_{n+k}. \quad \text{(line 14 of Algorithm 2).} \end{aligned} \tag{11}$$

In the implementation of IDR(s)-biortho from Algorithm 1, the vector $\widehat{g}_{n+k}$ is subsequently orthogonalized against $q_1, \ldots, q_{k-1}$ using an MGS process (lines 15–19 in Algorithm 1), whereas v_{n+k} is orthogonalized using a CGS process (lines 9–10 in Algorithm 1).

By performing *both* oblique projections in (10) using a CGS process, it will be shown that in each iteration step k only s (combined) inner products have to be computed and that the rest of the inner products can be computed using scalar updates.

Using the orthogonality condition (3), define the $(s-k+1) \times (s-k+1)$ and $(k-1) \times (k-1)$ lower triangular matrices M_l and M_t as

$$M_l \equiv [\mu_{i,j}^l] = \begin{cases} q_i^H g_{n+j-s-1} & \text{for } k \leqslant j \leqslant i \leqslant s \\ 0 & \text{otherwise} \end{cases} \qquad (g \in \mathcal{G}_j) \tag{12}$$

and

$$M_t \equiv [\mu_{i,j}^t] = \begin{cases} q_i^H g_{n+j} & \text{for } 1 \leqslant j \leqslant i \leqslant k-1 \\ 0 & \text{otherwise} \end{cases} \qquad (g \in \mathcal{G}_{j+1} \subset \mathcal{G}_j), \tag{13}$$

DOI: 10.1002/nla

respectively. Using the orthogonality condition (4), define the $s \times 1$ column vectors ϕ and ψ as:

$$\phi_i = \begin{cases} q_i^H r_{n+k} & \text{for } k \leqslant i \leqslant s, \\ 0 & \text{otherwise,} \end{cases} \tag{14}$$

$$\psi_i = q_i^H \widehat{g}_{n+k} \quad \text{for } 1 \leqslant i \leqslant s \quad \text{(line 15 of Algorithm 2).} \tag{15}$$

Using these definitions, the following two small lower triangular systems have to be solved in order to perform the oblique projections (10):

$$\begin{aligned} M_l \gamma_{(k:s)} &= \phi_{(k:s)} \quad \text{(line 10 of Algorithm 2),} \\ M_t \alpha_{(1:k-1)} &= \psi_{(1:k-1)} \quad \text{(line 16 of Algorithm 2).} \end{aligned} \tag{16}$$

Here, the notation $\phi_{(m:n)}$ denotes the column vector $[\phi_m, \phi_{m+1}, \ldots, \phi_n]^{\mathrm{T}}$.

Most of the inner products that are computed during iteration step k can be stored in a single lower triangular matrix M. To be more precise, define at the start of iteration step k the following $s \times s$ lower triangular matrix M, consisting of the three submatrices M_t, M_l, and M_c:

$$M \equiv \begin{bmatrix} M_t & 0 \\ M_c & M_l \end{bmatrix} = \left[\begin{array}{ccc|ccc} \mu_{1,1}^t & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \ddots & 0 & & & \vdots \\ \mu_{k-1,1}^t & \cdots & \mu_{k-1,k-1}^t & \ddots & & \vdots \\ \hline \mu_{k,1}^c & \cdots & \mu_{k,k-1}^c & \mu_{k,k}^l & 0 & 0 \\ \vdots & & \vdots & \vdots & \ddots & 0 \\ \mu_{s,1}^c & \cdots & \mu_{s,k-1}^c & \mu_{s,k}^l & \cdots & \mu_{s,s}^l \end{array}\right], \tag{17}$$

where M_c is defined as the $(s-k+1) \times (k-1)$ block matrix

$$M_c \equiv [\mu_{i,j}^c] = q_i^H g_{n+j} \quad \text{for } k \leqslant i \leqslant s, \;\; 1 \leqslant j \leqslant k-1. \tag{18}$$

We are now ready to compute the vector $v_{n+k} \in \mathcal{G}_j \cap Q^{\perp}$ as follows. If

$$\gamma_{(k:s)} = M_l^{-1} \phi_{(k:s)}, \quad v_{n+k} = r_{n+k} - \sum_{i=k}^{s} \gamma_i g_{n+i-s-1} \quad \text{(lines 10–11 in Algorithm 2),} \tag{19}$$

then

$$v_{n+k} \perp q_1, \ldots, q_k. \tag{20}$$

In addition, to compute the vector $g_{n+k} \in \mathcal{G}_{j+1} \cap Q_{k-1}^{\perp}$ and corresponding update u_{n+k}, let

$$\alpha_{(1:k-1)} = M_t^{-1} \psi_{(1:k-1)}; \tag{21}$$

$$g_{n+k} = \widehat{g}_{n+k} - \sum_{j=1}^{k-1} \alpha_j g_{n+j}; \tag{22}$$

$$u_{n+k} = \widehat{u}_{n+k} - \sum_{j=1}^{k-1} \alpha_j u_{n+j}, \tag{23}$$

then

$$g_{n+k} \perp q_1, \ldots, q_{k-1} \quad \text{and} \quad u_{n+k} \perp_A q_1, \ldots, q_{k-1}, \tag{24}$$

which is exactly condition (3).

DOI: 10.1002/nla

To efficiently compute the new column k of M using a scalar update, premultiply the recurrence for g_{n+k} with q_i^H to obtain

$$q_i^H g_{n+k} = q_i^H \widehat{g}_{n+k} - \sum_{j=1}^{k-1} \alpha_j q_i^H g_{n+j} \tag{25}$$

$$\mu_{i,k}^l = \psi_i - \sum_{j=1}^{k-1} \alpha_j \mu_{i,j}^c \quad \text{for } i = k, \ldots, s, \tag{26}$$

where the second expression uses the block matrix M_c from (17).

To summarize, this gives for step k while referring to the line numbers in Algorithm 2:

$$\psi = Q^H \widehat{g}_k \quad \text{(line 15, } s \text{ combined inner products)},$$

$$\alpha_{(1:k-1)} = M_t^{-1} \psi_{(1:k-1)} \quad \text{(line 16, lower triangular system } M_t^{-1}),$$

$$g_{n+k} = \widehat{g}_{n+k} - \sum_{j=1}^{k-1} \alpha_j g_{n+j} \quad \text{(line 18, orthogonalize against } q_1, \ldots, q_{k-1}),$$

$$u_{n+k} = \widehat{u}_{n+k} - \sum_{j=1}^{k-1} \alpha_j u_{n+j} \quad \text{(line 18, } A\text{–orthogonalize against } q_1, \ldots, q_{k-1}),$$

$$\mu_{i,k}^l = \psi_i - \sum_{j=1}^{k-1} \alpha_j \mu_{i,j}^c, \quad i = k, \ldots, s \quad \text{(line 20, new column } k \text{ of } M \text{ using } M_c).$$

In accordance with condition (4), the updated residual r_{n+k+1} can be made orthogonal to $q_1, \ldots, q_k$ by

$$r_{n+k+1} = r_{n+k} - \frac{\phi_k}{\mu_{k,k}^l} g_{n+k} \quad \text{(line 23 of Algorithm 2)}, \tag{27}$$

since

$$q_k^H r_{n+k+1} = q_k^H r_{n+k} - \frac{\phi_k}{\mu_{k,k}^l} q_k^H g_{n+k} \tag{28}$$

$$= q_k^H r_{n+k} - q_k^H r_{n+k} = 0. \tag{29}$$

Premultiply (27) with A^{-1} to obtain the corresponding update to the iterate

$$x_{n+k+1} = x_{n+k} + \frac{\phi_k}{\mu_{k,k}^l} u_{n+k} \quad \text{(line 24 of Algorithm 2)}. \tag{30}$$

Finally, premultiplying (27) with q_i^H for $i = k+1, \ldots, s$ gives the scalar update for the vector ϕ,

$$\phi_i \leftarrow \phi_i - \frac{\phi_k}{\mu_{k,k}^l} \mu_{i,k}^l, \quad i = k+1, \ldots, s \quad \text{(line 28 of Algorithm 2)}, \tag{31}$$

which concludes the generation of the s vectors for $\mathcal{G}_{j+1}$.

Therefore, in each iteration step k the s combined inner products $Q^H \widehat{g}_k$ in the vector ψ have to be computed. The remaining inner products $Q^H r$ in the vector ϕ and the inner products $Q^H g$ in the small matrix M can then be computed using only scalar updates.

As with Algorithm 1, the computation of the (now solely combined) inner products is highlighted by boxes in Algorithm 2. The small systems in lines 10 and 16 of Algorithm 2 involving M_l and M_t, respectively, are lower triangular and can be solved efficiently using forward substitution. Note that the system in line 16 involves the first $k-1$ elements of the column vector ψ, while in line 20

DOI: 10.1002/nla

the remaining elements are used. This shows that there is now a *single* global synchronization point per iteration step. The number of operations is the same as the original IDR(s)-biortho method from Algorithm 1.

3. CHOOSING s AND Q

The parameter s and the test matrix Q can be chosen freely in IDR(s) methods. In this section this freedom is exploited in order to minimize parallel execution time. In Section 3.1 performance models will be given that are used to estimate the optimal s in IDR(s) methods. Section 3.2 describes a method of minimizing the work and storage involving operations with the test matrix Q in a multi–cluster environment.

3.1. Parallel performance models for IDR(s)

Performance models are derived to estimate the total execution time of the IDR(s)-biortho variant and the IDR(s)-minsync variant. The model is applied to both a single cluster and to a multi-cluster. After first presenting a general model for the total execution time of parallel IDR(s) methods, specific expressions are given for the communication steps of the IDR(s) algorithms.

3.1.1. A general model. The overall performance model is based on message passing communication. It is assumed that no load imbalance occurs and as a result the parallel computing time for a fixed-sized problem on p processors can be described in general by

$$T_{\text{total}}(p) = \frac{T(1)}{p} + T_{\text{comm}}, \tag{32}$$

where T_{comm} denotes the total communication time of the algorithm and $T(1)$ the computation time on a single processor.

Generally speaking, there are two operations that require communication in parallel iterative methods, which are the inner product (both single and combined) and the matrix–vector product. The first operation requires global communication, whereas the second operation requires nearest neighbour communication for the current application. No preconditioning is included in the performance model.

The following simple linear model for the time to communicate a message of k bytes length is assumed,

$$T_{\text{mess}} = l + k/b, \tag{33}$$

in which l is the latency and b the bandwidth. Using this linear model, expressions for the communication time of inner products (denoted by T_{dots}) and matrix–vector multiplication (denoted by T_{mmult}) in each IDR(s) cycle can be derived. These expressions will in general depend on both p and s.

To complete the performance model, the total number of IDR(s) cycles has to be determined. As mentioned before, it can be shown that in exact arithmetic, IDR(s) methods terminate within $N(1+s^{-1})$ matrix–vector multiplications, or equivalently, within Ns^{-1} dimension reduction steps [1, Section 3]. In practical applications, the iteration process will exhibit much faster (superlinear) convergence rates according to $\widehat{N}s^{-1}$, where $\widehat{N} \ll N$.

To summarize, the total theoretical computing time on p processors for a given value of s is then (cf. (32))

$$T_{\text{total}}(p,s) = \frac{\widehat{N}}{s} \times \underbrace{\left[\frac{\widetilde{T}(1,s)}{p} + T_{\text{dots}}(p,s) + T_{\text{mmult}}(p,s) \right]}_{\text{time of a single IDR}(s)\text{ cycle}}, \tag{34}$$

DOI: 10.1002/nla

where $\widetilde{T}(1,s)$ is the computing time of one *sequential* IDR(s) cycle. This quantity can be estimated by counting the number of floating point operations and using the value for the computational speed of a single processor.

Using the expression (34), the optimal value of s and p for solving the test problem in a minimal amount of time can be obtained. Note that the parameter $\widehat{N}$ corresponds to the total number of cycles for $s=1$ and that it is a constant. Therefore, it does not play any role in minimizing (34). Only problem and machine-dependent parameters are needed to find the optimal p and s.

3.1.2. Models for different architectures. The performance model (34) will be applied to a single cluster and to a multi-cluster for both IDR(s) variants as follows.

For the single cluster model, the values for latency, bandwidth, and computational speed of that cluster are used. In this way, the model is used to compute the optimal s and corresponding number of *nodes* p in a cluster.

In multi-cluster architectures, *inter*cluster latencies are often several orders of magnitude higher than *intra* cluster latencies. For the DAS-3 multi-cluster, these latencies differ by three orders of magnitude, while the values for the bandwidth differ by one order of magnitude. In the model for multi-clusters, intracluster latency and bandwidth is therefore excluded and each cluster is treated as a *single entity*. In this way, the model is used to compute the optimal s and corresponding number of *clusters* P in the DAS-3 multi-cluster. The intercluster latency and bandwidth of the DAS-3 multi-cluster are then used by the performance model. This also means that in the performance model, the computational speed of a cluster is taken to be the combined computational speed of all the nodes in that cluster.

In the following, specific expressions for the communication time of the inner products T_{dots} and the matrix–vector multiplication T_{mmult} will be presented.

The communication time of a properly implemented inner product on a single (tightly coupled) cluster is given by:

$$T_{\text{dot}} = \lceil \log_2(p) \rceil (l + 8/b). \tag{35}$$

In the algorithms, some inner products can be combined. The communication time of the combined broadcasting of c partial inner products is then

$$T_{\text{cdot}} = \lceil \log_2(p) \rceil (l + 8c/b). \tag{36}$$

For the multi-cluster model, we note that the network topology of the DAS-3 multi–cluster is structured like a ring (see Section 4.1 for more details). However, the number of clusters that is used is always relatively small, so it is assumed that the hierarchical model for the inner product also applies to the multi-cluster setting.

A minimal residual strategy is used to compute ω, so the *total* communication time $T_{\text{dots}}(p,s)$ spent on inner products in each cycle of IDR(s)-biortho is then

$$T_{\text{dots}}^{\text{biortho}}(p,s) = \lceil \log_2(p) \rceil [(\tfrac{1}{2}s(s+1)+2)l + 8(s^2+s+2)/b] \tag{37}$$

$$= \lceil \log_2(p) \rceil [\mathcal{O}(s^2)l + \mathcal{O}(s^2)/b] \tag{38}$$

and for the new IDR(s)-minsync variant it is

$$T_{\text{dots}}^{\text{minsync}}(p,s) = \lceil \log_2(p) \rceil [(s+1)l + 8(s^2+s+2)/b] \tag{39}$$

$$= \lceil \log_2(p) \rceil [\mathcal{O}(s)l + \mathcal{O}(s^2)/b]. \tag{40}$$

The expressions (38) and (40) show that for relatively large values of bandwidth b, the communication time spent on inner products in each cycle grows differently with s in the two variants. For larger values of latency l, the time per cycle is almost linear in s for IDR(s)-minsync, whereas the time per cycle behaves quadratically in s for IDR(s)-biortho.

DOI: 10.1002/nla

Expressions for the communication time of the matrix–vector multiplication $T_{\text{mmult}}(p,s)$ can be derived in a similar manner. For a single cluster, the cubical domain is partitioned into rectangular cuboids and each subdomain is assigned to a single node. Supposing that $n_x = n_y = n_z$, the communication time spent on the $s+1$ matrix–vector multiplications in each IDR(s) cycle for an interior subdomain is (in both IDR(s) variants)

$$T_{\text{mmult}}(p,s) = 6(s+1)\left(l + \frac{8n_x^2}{b\sqrt[3]{p}}\right) \quad \text{(single cluster)}. \tag{41}$$

Considering again the fact that the DAS-3 network has a ring structure, the domain partitioning in the multi-cluster setting can be seen as two-dimensional. That is, each subdomain is assigned to a single cluster. In addition, each interior subdomain has two neighbouring subdomains, which gives for the matrix–vector multiplication (again in both IDR(s) variants)

$$T_{\text{mmult}}(s) = 2(s+1)(l + 8n_x n_y/b) \quad \text{(multi–cluster)}. \tag{42}$$

Note that this expression is independent of the number of clusters.

Using this information, the expression (34) can then be minimized which allows for *a priori* estimation of the optimal parameter s and corresponding number of nodes/clusters using only problem and machine-based parameters.

This also shows that the performance model can be as complex as one desires, depending wholly on the type of architecture and application. For example, the Multi-BSP model from [22] is a hierarchical performance model for modern multi-core architectures and could be used to find the optimal s of IDR(s) methods on such architectures.

In Section 4 the performance models given here will be compared with numerical experiments on the DAS-3 multi-cluster. For experiments using nodes from one cluster, the model for a single cluster is used. When using more than one cluster, we switch to the performance model for multi-clusters.

3.2. Using piecewise sparse column vectors for Q

In multi-cluster environments such as the DAS-3, the latency between clusters may be several orders of magnitude larger than the intracluster latency. The majority of the inner products in the algorithm consist of computing $Q^H r$ for some vector r. The $N \times s$ matrix Q can be chosen arbitrarily and this freedom can be exploited [1, Section 4.1]. By using sparse column vectors for Q, the total cost of the inner products may be reduced significantly in the context of a multi-cluster environment. However, such a strategy may influence the robustness of the algorithm and this phenomenon will be illustrated experimentally.

The outline of the algorithm for the computation of $Q^H r$ using sparse column vectors for Q is as follows. Each cluster in the multi–cluster is considered as one (large) subdomain. The columns of Q are chosen in such a manner that they are nonzero on one of these subdomains and zero on the other subdomains.

A *coordinator* node is randomly chosen on each cluster and each node computes its local inner product with its local part of r. A reduction operation is then performed locally on each cluster and the result is gathered on the coordinator node. The coordinators exchange the partial inner products across the slow intercluster connections, combining them to make the total inner product. Finally, this result is broadcasted locally within each cluster. Therefore, the number of times that data is sent between the clusters is reduced considerably.

For ease of implementation, the value s is chosen as an integer multiple of the total number of clusters γ in the grid. Using sparse column vectors decreases the computational work and the storage requirements on each cluster. Instead of computing s inner products of length N, the total computational cost is reduced to s inner products of length N/γ. Note that this approach is valid for arbitrary s.

DOI: 10.1002/nla

As an example, suppose that there are three clusters in the multi-cluster and that each cluster has two nodes, giving six nodes $\{a, b, c, d, e, f\}$ in total. If $s = 6$, the computation of $Q^H r$ using the sparse column vectors $q_1, \ldots, q_6$ for Q has the following form:

$$Q^H r = \left[\begin{array}{cc|cc|cc} q_1^a & q_1^b & & & & \\ q_2^a & q_2^b & & & & \\ \hline & & q_3^c & q_3^d & & \\ & & q_4^c & q_4^d & & \\ \hline & & & & q_5^e & q_5^f \\ & & & & q_6^e & q_6^f \end{array}\right] \times \left[\begin{array}{c} r^a \\ - \\ \dfrac{r^b}{r^c} \\ - \\ \dfrac{r^d}{r^e} \\ - \\ r^f \end{array}\right]. \tag{43}$$

In this case, parts of two columns of Q are nonzero on one cluster and zero on the remaining two clusters. Therefore, two partial local inner products are computed by each node and the results are gathered on one of the two nodes of the cluster. These results are then exchanged between the three clusters and broadcasted locally to the two nodes in each cluster.

4. NUMERICAL EXPERIMENTS

Three parallel implementations of IDR-based methods will be compared:

(i) the IDR(s)-biortho variant given in Algorithm 1 and using a dense matrix Q;
(ii) the IDR(s)-minsync variant given in Algorithm 2 and using a dense matrix Q;
(iii) the IDR(s)-minsync variant given in Algorithm 2 and using the sparse matrix Q as described in Section 3.2.

Note that in exact arithmetic, the first two variants produce residuals that are identical in every iteration step.

This section is structured as follows. In Sections 4.1 and 4.2, a description is given for the target hardware and test problem, respectively. In Section 4.3 the parameter $\widehat{N}$ for this test problem and the (true) computational speed of a single core are estimated. In Section 4.4 the optimal parameter s for a particular computational environment is computed using the performance model. In Section 4.5 the performance model is compared with the numerical results. In Section 4.6 the experimental results are investigated more closely by comparing the time per cycle with the performance model. Finally, both strong and weak speedup results are given in Section 4.7, which includes results for sparse Q.

4.1. Target hardware

The numerical experiments are performed using the distributed ASCI Supercomputer 3 (DAS-3), which is a cluster of five clusters, located at four academic institutions across the Netherlands [13]. The five sites are connected through SURFnet, which is the academic and research network in the Netherlands. Each local cluster is equipped with both 10 Gbps Ethernet and high speed Myri-10G interconnect. However, the TUD site only employs the Ethernet interconnect. If an experiment includes the TUD site, the other sites will automatically switch to the slower Ethernet interconnect. Specific details on the five sites are given in Table I.

The network topology of the DAS-3 cluster is structured like a ring, connecting the five sites as follows: TUD, LU, VU, UvA, UvA–MN, and again to the TUD site. Although nodes on some sites

DOI: 10.1002/nla

Table I. Specific details on each DAS-3 site, taken from [13].

Site	Nodes	Speed (GHz)	Memory (GB)	Network
TUD	68	2.4	4	GbE
LU	32	2.6	4	Myri-10G/GbE
VU	85	2.4	4	Myri-10G/GbE
UvA	41	2.2	4	Myri-10G/GbE
UvA–MN	46	2.4	4	Myri-10G/GbE

Table II. Specifications DAS-3: all values (except WAN latencies) courtesy of Henri Bal [23].

	LU site
One-way latency MPI (μs)	2.7
max. throughput (MB/s)	950
Gflops (HPL benchmark [24])	6.9
	DAS-3
WAN latencies (μs) (average)	990
WAN bandwidth (MB/s)	5000

may contain multiple cores, we always employ a single core on each node for our computations. Table II lists values provided by Henri Bal on latency and bandwidth for the LU site and for the wide-area bandwidth on all five clusters [23]. These values were corroborated by the authors using the Intel MPI Benchmarks suite (IMB v2.3). The value for the WAN latency in Table II is the average value from several IMB benchmarks performed at different times during the day and is similar to (albeit somewhat below) the values given in [25].

The algorithms are implemented using OpenMPI v1.2.1 and level 3 optimization is used by the underlying GNU C compiler. The implementation is matrix-free: the coefficient matrix is not explicitly formed and stored.

4.2. Test problem and experimental setup

Consider the following three-dimensional elliptic partial differential equation taken from [26]:

$$\nabla^2 u + w u_x = f(x, y, z), \tag{44}$$

defined on the unit cube $[0, 1] \times [0, 1] \times [0, 1]$. The predetermined solution

$$u = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z) \tag{45}$$

defines the vector f and Dirichlet boundary conditions are imposed accordingly.

Discretization by the finite difference scheme with a seven point stencil on a uniform $n_x \times n_y \times n_z$ grid results in a sparse linear system of equations $Ax = b$ where A is of order $N = n_x n_y n_z$. Centered differences are used for the first derivatives. The grid points are numbered using the standard (lexicographic) ordering and the convection coefficient w is set to 100.

When not specified otherwise, the matrix Q consists of s orthogonalized random vectors. The iteration is terminated when $\|r_n\|/\|r_0\| \leqslant \varepsilon \equiv 10^{-6}$ and the initial guess is set to $x_0 \equiv 0$. At the end of the iteration process convergence is verified by comparing the true residual with the iterated final residual. In addition, no preconditioner is applied, i.e. $B = I$.

Parallel scalability will be investigated in both the strong and weak sense. In strong scalability experiments a fixed *total* problem size is used, while in weak scalability experiments a fixed problem size *per node* is used.

The experiments for investigating strong scalability are performed using the four sites that employ the fast interconnect. On each cluster, 32 nodes are used, which gives a total of 128 nodes for the largest experiment. Experiments that use less than 32 nodes are performed on the LU site

DOI: 10.1002/nla

Table III. Processor grids and problem size for the strong scalability experiments.

# Nodes	$p_x \times p_y \times p_z$	# Equations
1	$1\times1\times1$	
2	$2\times1\times1$	
4	$2\times2\times1$	
8	$2\times2\times2$	
16	$4\times2\times2$	128^3
32	$4\times2\times4$	
64	$4\times4\times4$	
96	$6\times4\times4$	
128	$8\times4\times4$	

Table IV. Processor grids and problem sizes for the weak scalability experiments.

	$p_x \times p_y \times p_z$		
# Nodes	First strategy	Second strategy	# Equations
30	$5\times3\times2$	$1\times6\times5$	398^3
60	$5\times4\times3$	$2\times6\times5$	501^3
90	$5\times6\times3$	$3\times6\times5$	574^3
120	$5\times6\times4$	$4\times6\times5$	631^3
150	$5\times6\times5$	$5\times6\times5$	680^3

and in each subsequent experiment 32 nodes are added with each additional site, in the following order: VU, UvA, and UvA–MN. In this way, the ring structure of the DAS-3 wide-area network is obeyed. The number of grid points in each direction is $n_x = n_y = n_z \equiv 128$, which gives a total problem size of approximately two million equations.

For the weak scalability experiments, 30 nodes per site are used and the TUD cluster is included, which means that in this case the slower interconnect is used on every cluster. The number of equations per node is set to approximately two million equations, yielding the problem sizes shown in Table IV.

The computational domain is partitioned using a three-dimensional block partitioning, where the subdomains are arranged in a Cartesian grid $p_x \times p_y \times p_z$. The nodes are numbered according to

$$p(i, j, k) = p_z^k + (p_y^j - 1)p_z + (p_x^i - 1)p_y p_z \tag{46}$$

with $p_x^i \times p_y^j \times p_z^k \in [1, p_x] \times [1, p_y] \times [1, p_z]$. The nodes are mapped sequentially across the clusters, one cluster after another. In the multi-cluster experiments, the domain is partitioned along the x-direction. This partitioning divides the domain into slices and in our experiments, each slice is mapped onto a cluster. Partitioning in this way ensures that adjacent slices in the domain correspond to adjacent clusters in the DAS-3. In addition, this partitioning corresponds to the multi-cluster performance model as discussed in Section 3.1.

Using this strategy, Tables III and IV list the dimensions of the processor grid for each number of nodes used in the strong and weak scalability experiments, respectively.

As shown in Table IV, the problem size for the weak scalability experiments will be increased using two different strategies. In the first strategy the nodes are equally divided between the five DAS-3 sites, starting with $\frac{30}{5} = 6$ nodes per site for the smallest experiment. This means that there are always five slices, one for each cluster. In addition, this allows comparing the use of a dense test matrix Q and of a sparse test matrix Q.

In the second strategy, one whole site is added each time, starting with the LU site and ending with the TUD site. Here, the number of slices corresponds to the number of clusters. In addition, for this experiment only a dense test matrix Q is used.

DOI: 10.1002/nla

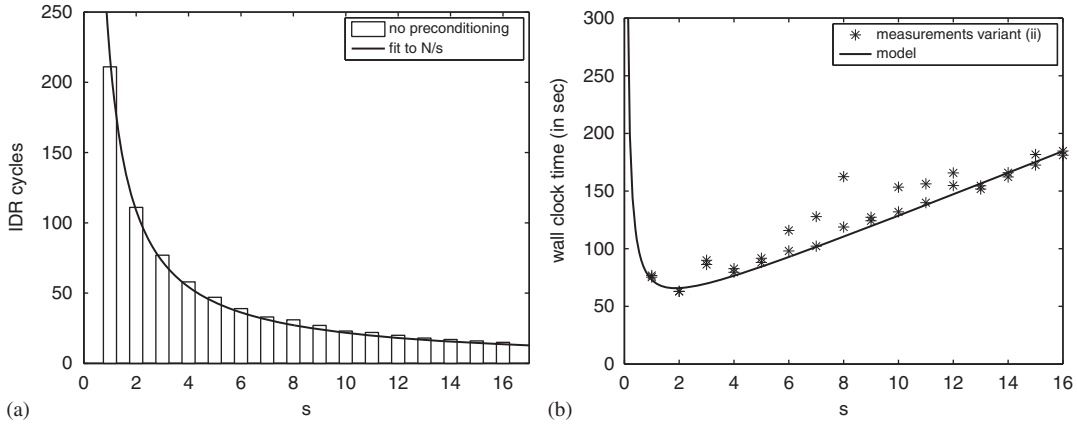

Figure 1. Estimating model parameters: $\widehat{N}$ and processor speed: (a) Total IDR cycles, fitted to $\widehat{N}/s$ with $\widehat{N} \approx 218$ and (b) modeling computing times single node.

4.3. Estimating parameters of performance model

In order to estimate $\widehat{N}$ for the test problem, the total number of dimension reduction steps for $s \in \{1, \ldots, 16\}$ are shown in Figure 1(a). These experimental data are obtained using variant (ii) on a single LU node. Variant (i) gives the same results and the data are fitted to the curve $\widehat{N}/s$, giving $\widehat{N} \approx 218$. This shows that the number of IDR(s) cycles behaves in accordance with the theoretical estimate. Interestingly, the value for $\widehat{N}$ is of the same order of magnitude as the number of grid points in each direction $n_x = n_y = n_z \equiv 128$. Note that a *single* (possibly parallel) experiment is sufficient to estimate the parameter $\widehat{N}$.

In order to estimate the effective computational speed of a single core, wall clock times of two executions of variant (ii) on an LU node for each value of s are shown in Figure 1(b). Since variants (i) and (ii) have the same number of operations, only results from the second variant are given. The theoretical computing time of the algorithm on a single node is equal to (cf. (34))

$$T_{\text{total}}(1, s) = \frac{\widehat{N}}{s} \times \widetilde{T}(1, s). \tag{47}$$

Using the previously obtained value of $\widehat{N}$, the value for the computational speed is obtained by fitting the measurements to the model (solid line in Figure 1(b)), which gives a value of 3×10^{-1} Gflops. According to the HPL benchmark (i.e. Table II), the peak performance of a single core is 6.9 Gflops. However, it is not uncommon that only a fraction of the processor's peak performance can be attained in practice. These results also indicate that when using a single core, setting $s = 2$ results in the fastest computing times.

4.4. A priori estimation of optimal parameter s

By using the expression for the theoretical computing time (34) from Section 3.1, the optimal parameter s that minimizes the total execution time can be computed. Note that every parameter needed to compute these optimal values is problem or machine-dependent and can be obtained *a priori*. In particular, the parameter $\widehat{N}$ is not required.

As described in Section 3.1, estimates for both a single cluster (the LU site) and for a multi–cluster (the DAS-3) will be given. Using the data from Table II, the optimal (rounded value of) s for a given (theoretical) number of nodes of the LU cluster are computed and shown in Table V for variants (i) and (ii), respectively. Similarly, Table VI shows the optimal s for a given (theoretical) number of clusters in the DAS-3 multi-cluster—with 32 nodes per cluster—for variants (i) and (ii), respectively. The corresponding wall clock times are also shown.

According to Table V, the optimal s for using one node on the LU cluster is $s = 2$ for both variants. This is in agreement with results from Figure 1(b) from Section 4.3, where the computing

DOI: 10.1002/nla

Table V. *A priori* estimation of s for the LU cluster.

	Variant (i)		Variant (ii)	
# Nodes	Optimal s	Wall clock time	Optimal s	Wall clock time
1	2	66.14	2	66.14
2	2	33.15	2	33.15
4	2	16.64	2	16.64
8	2	8.378	2	8.376
16	2	4.235	2	4.233
32	2	2.156	2	2.153
64	2	1.111	2	1.107

Table VI. *A priori* estimation of s for the DAS-3 multi-cluster.

	Variant (i)		Variant (ii)	
# Clusters	Optimal s	Wall clock time	Optimal s	Wall clock time
1	2	2.058	2	2.058
2	2	2.231	3	1.968
3	2	2.203	3	1.772
4	2	2.255	4	1.69
5	2	2.325	4	1.657
6	2	2.398	5	1.639
7	2	2.469	5	1.633

times on a single node were given (i.e. latency equal to *zero*). For that case the optimal value was also $s=2$.

Table V also shows that for all number of nodes the optimal value of s is always $s=2$. In this case, latency is low and the communication cost for generating the vectors of the nested spaces $\mathscr{G}_j$ is relatively small. Apparently the reduction in number of cycles for larger s is not worth the increased cost of generating more vectors in $\mathscr{G}_j$. In addition, there is practically no difference in optimal s and wall clock time for both variants. The reason is that for $s=1$, both variants are equal in cost and that for very low $s>1$, the differences in cost are only marginal.

For the multi-cluster model, the latency value is much higher. In this case, the differences between the two variants become more apparent, as illustrated by Table VI. For variant (i), $s=2$ is again an optimal value for all number of clusters, but the wall clock time grows with increasing number of clusters. In contrast, using variant (ii) not only gives lower wall clock times, it also shows that increasing the number of clusters (and correspondingly the value of s) results in improved execution times. It is interesting to see that in parallel IDR(s) methods the optimal value of s can be determined in such a manner.

4.5. Validation of parallel performance model

Using a log–log scale, the predicted total computing times defined by (34) in Section 3.1 for variants (i) and (ii) using $s \in \{1, 3, 5, 10\}$ and using up to (in theory) 512 nodes (i.e. sixteen clusters) are shown in Figure 2. Experiments that use up to 32 nodes employ the LU site and corresponding parameters in the performance model. In the larger experiments, a cluster consisting of 32 nodes is added each time and the corresponding multi-cluster performance model is used. The measured wall clock times using up to 128 nodes (i.e. four clusters of the DAS-3) for $s \in \{1, 3, 5, 10\}$ are also shown.

Communication overhead on the LU site (i.e. using less than 32 nodes) is relatively small. As a result, the total wall clock time scales almost linearly with the number of nodes in both variants. This holds for both the model and the measurements. Note that Table V also shows this linear scaling behaviour for $s=2$. However, when more clusters are added (i.e. using more than

DOI: 10.1002/nla

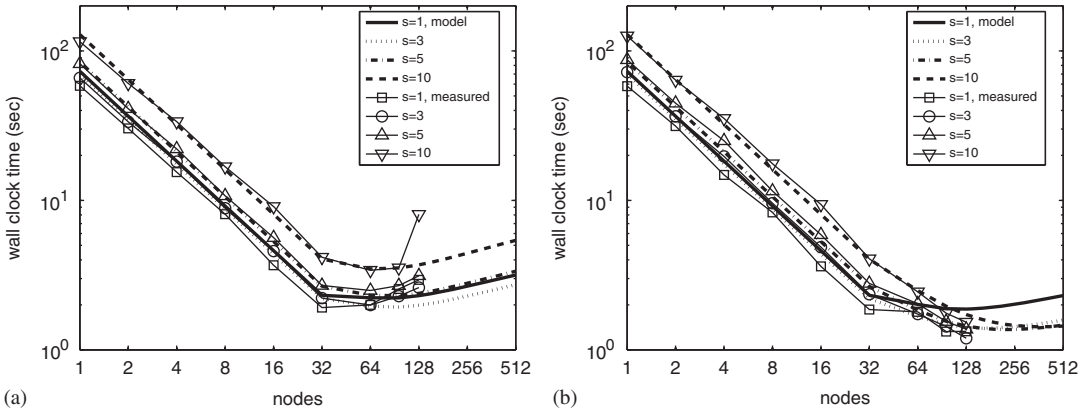

Figure 2. Performance model results for variants (i) and (ii) using 32 nodes per cluster: (a) IDR(s)-biortho (i.e. variant (i)) and (b) IDR(s)-minsync (i.e. variant (ii)).

32 nodes), the effect of communication begins to play a larger role. In this case, the optimal s and number of nodes differ for both variants.

According to the measurements from Figure 2(a), the optimal value of s for solving this test problem using variant (i) lies between $s=1$ and $s=3$. The corresponding number of nodes lies between 32 and 64 nodes (i.e. between one and two clusters). This is in accordance with the predictions from Table VI, which showed that the estimated optimal value of s is $s=2$ using one cluster.

The measurements from Figure 2(b) show that for variant (ii) using 128 nodes (i.e. four clusters) gives the minimum computing times. The trend seems to be that using more than four clusters only results in a marginal reduction in wall clock time. The corresponding value of s is between $s=3$ and $s=5$. Indeed, the predictions from Table VI show that using more than four clusters results in small reductions in total time. Correspondingly, the estimated optimal value of s is $s=4$.

These results nicely illustrate the fact that there is an optimal value of s and number of nodes for solving this test problem in a minimal amount of time.

4.6. Comparing the time per IDR(s) cycle to the performance model

To investigate the relation between the value of s and the time per IDR(s) cycle, the following experiment is performed. Figure 3 shows results of experiments using all three variants and using a total of 64 nodes, divided equally between the four DAS-3 sites that employ the fast interconnect.

For completeness, the total number of IDR cycles is shown in Figure 3(a), which is practically identical for all three variants. The experiments showed that the use of sparse column vectors for Q can result in numerical instability issues for large s. That is, when using variant (iii) the iteration did not converge for values of $s>8$ for this test problem. As before, the total number of IDR cycles is fitted to the curve $\widehat{N}/s$, which gives in this case $\widehat{N}\approx 211$. Naturally, this value is almost identical to the previously obtained value for $\widehat{N}$ from Section 4.3.

More interestingly, Figure 3(b) shows the wall clock time per IDR cycle for increasing values of s. As mentioned in Section 3.1, the performance model (i.e. expression (40)) predicts that for larger bandwidth values, the time spent on inner products per cycle in variant (ii) scales almost linearly with s. The measurements are in agreement with this prediction.

Similarly, expression (38) from Section 3.1 shows that the time per IDR(s) cycle in variant (i) has more quadratic behaviour in s, which is also in agreement with the measurements from Figure 3(b). As a result, there is a significant increase in time per iteration with increasing s for variant (i).

In general, the performance model is in good agreement with the measurements. The outlier for $s=1$ for variants (i) and (ii) seems related to C compiler optimizations, since *disabling* these optimizations *reduced* the time per cycle. For some reason, variant (iii) conforms well to the performance model for $s=1$ when using the compiler optimizations.

DOI: 10.1002/nla

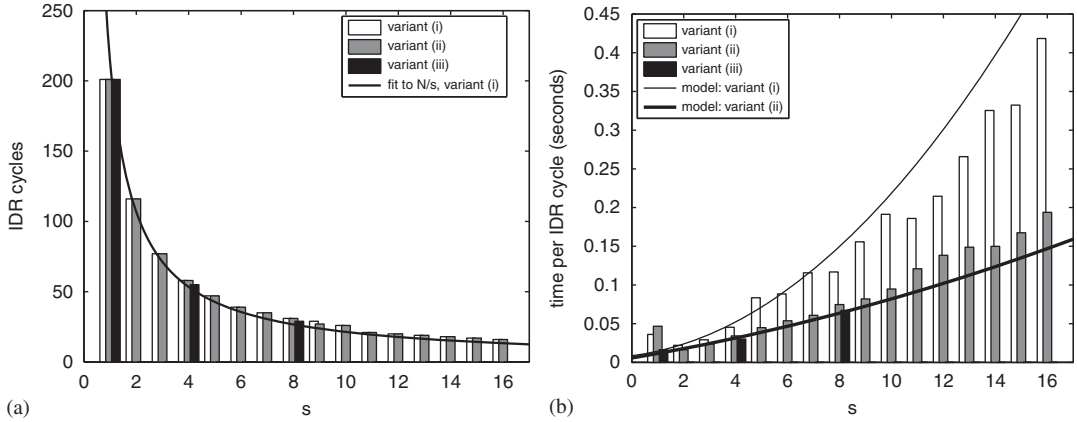

Figure 3. Investigating s-dependence for $s \in \{1, \ldots, 16\}$ using 64 nodes, four sites, and fast network: (a) Total IDR(s) cycles, $\widehat{N} \approx 211$ and (b) time per IDR(s) cycle.

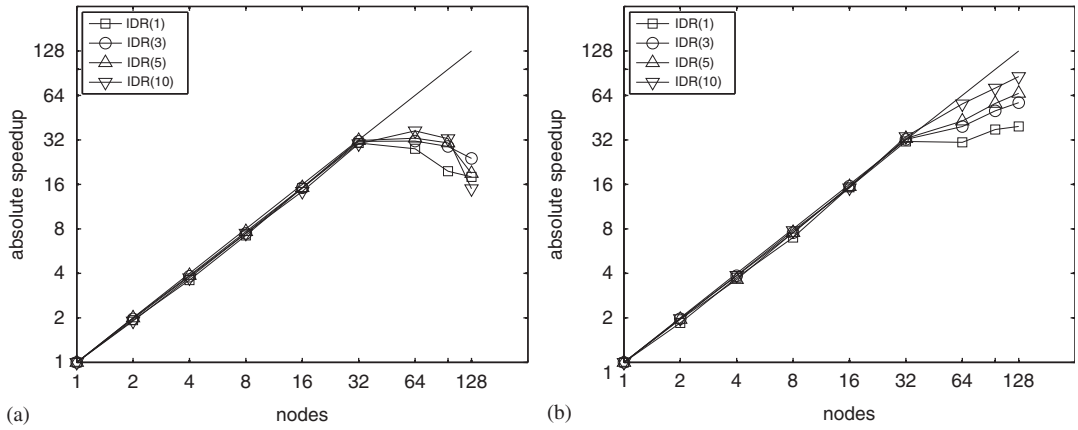

Figure 4. Absolute speedup, scaled to number of iterations and using a log–log scale: (a) IDR(s)-biortho (i.e. variant (i)) and (b) IDR(s)-minsync (i.e. variant (ii)).

4.7. Parallel speedup results

In this section strong and weak scalability of the three variants is investigated. The standard definition for strong scalability S is used, i.e.

$$S(p) = \frac{T(1)}{T(p)}, \tag{48}$$

where $T(1)$ is the execution time of the parallel algorithm on one node and p is the number of nodes. Figure 4 shows strong scalability results using variants (i) and (ii) for $s \in \{1, 3, 5, 10\}$, in Figure 4(a) and (b), respectively. The scalability results of variant (iii) are similar to that of variant (ii) and are therefore omitted. Optimal speedup $S(p) = p$ is also shown.

The near to linear speedup of both variants using up to 32 nodes on the LU site is not surprising, considering the fact that in this case communication overhead is almost negligible. However, as more sites are added, the results show that IDR(s)-minsync (i.e. variant (ii)) scales much more favourably than IDR(s)-biortho (i.e. variant (i)). Since for $s = 1$ variants (i) and (ii) almost have the same implementation, speedup is roughly identical. In addition, the results show that variant (i) exhibits the same (bad) scalability for all values of s, while increasing s in variant (ii) gives significant gains in scalability.

DOI: 10.1002/nla

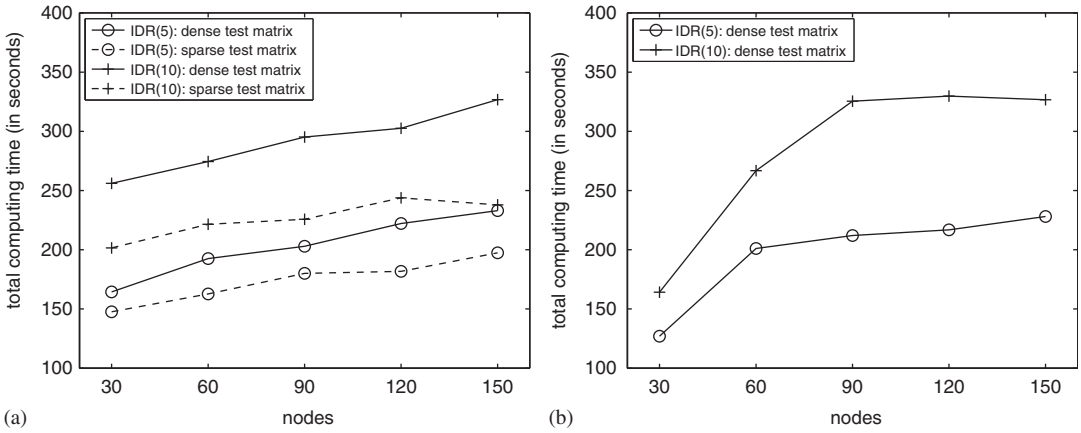

Figure 5. Weak scaling experiments on the DAS-3: (a) Comparing sparse and dense test matrices (first strategy) and (b) using a dense test matrix (second strategy).

To investigate the weak scalability of IDR(s)-minsync, the number of equations per node is fixed to approximately two million and a fixed number of iterations of 275 is performed. The TUD site is also used in this case and the number of nodes in each experiment is $30, 60, \ldots, 150$. The corresponding total problem sizes are listed in Table IV. Note that for variant (iii), the value of s has to be a multiple of the number of clusters in the grid. In the ideal case, the time per iteration is constant for increasing number of nodes.

As explained in Section 4.2, the problem size is increased using two different strategies. Figure 5 shows the weak speedup results for $s \in \{5, 10\}$. In Figure 5(a) results are given when employing the first strategy, showing that using the sparse matrix Q gives increased gains in execution time for increasing s.

Figure 5(b) gives results using the second strategy. Not surprisingly, adding the second cluster results in a large jump in execution time, because the relative increase in communication time is rather high in this case. However, adding subsequent clusters show weak scalability results comparable to Figure 5(a).

5. CONCLUSIONS

The recent IDR(s) method is a family of fast algorithms for solving large sparse nonsymmetric linear systems. In cluster computing and in particular in Grid computing, global synchronization is a critical bottleneck in parallel iterative methods. To alleviate this bottleneck in IDR(s) algorithms, three strategies were used.

First, by reformulating the efficient and numerically stable IDR(s)-biortho method [12], the IDR(s)-minsync method was derived which has a *single* global synchronization point per iteration step. Experiments on the DAS-3 multi-cluster show that the new IDR(s)-minsync method exhibits increased speedup for increasing values of s. In contrast, the original IDR(s)-biortho variant has no speedup whatsoever on the DAS-3 multi–cluster for our test problem.

In addition, the test matrix in IDR(s)-minsync was chosen in such a way that the work, communication, and storage involving this matrix is minimized on the DAS-3 multi-cluster. Experiments on the DAS-3 show that this approach results in reduced execution times, albeit relatively moderate. However, the experiments also showed that this technique of using a sparse test matrix in IDR(s) methods can result in numerical instabilities, in particular for larger values of s. Given the fact that the reduction in execution times was limited, we do not recommend using sparse test vectors in the context of IDR(s) methods and multi-clusters.

Finally, using only problem and machine-dependent parameters, the presented parallel performance models were utilized for *a priori* estimation of the optimal value of s and corresponding

DOI: 10.1002/nla

number of nodes. This approach can be used to minimize the total execution time of parallel IDR(s) methods on both a single cluster and on a multi–cluster. The estimations were in good agreement with the experimental results.

Although the results shown in this paper have been obtained without any preconditioning, it is possible to apply the techniques for estimating the optimal s when a preconditioner such as a block incomplete LU factorization is used. This can be accomplished by adding the communication and computational cost of the preconditioner to the performance models.

ACKNOWLEDGEMENTS

The work of the first author was financially supported by the Delft Centre for Computational Science and Engineering (DCSE) within the framework of the DCSE project entitled *'Development of an Immersed Boundary Method, Implemented on Cluster and Grid Computers'*. The Netherlands Organization for Scientific Research (NWO/NCF) is gratefully acknowledged for the use of the DAS-3 system. The authors thank the referees for their careful reading of the manuscript and their valuable comments.

REFERENCES

1. Sonneveld P, van Gijzen MB. IDR(s): a family of simple and fast algorithms for solving large nonsymmetric linear systems. *SIAM Journal on Scientific Computing* 2008; **31**(2):1035–1062.
2. Simoncini V, Szyld DB. Interpreting IDR as a Petrov–Galerkin method. *SIAM Journal on Scientific Computing* 2010; **32**(4):1898–1912. DOI: 10.1137/090774756.
3. Gutknecht MH. IDR Explained. *Electronic Transactions on Numerical Analysis* 2010; **36**:126–148.
4. Sleijpen GLG, Sonneveld P, van Gijzen MB. Bi-CGSTAB as an induced dimension reduction method. *Applied Numerical Mathematics* 2009; In Press, Corrected Proof, DOI: 10.1016/j.apnum.2009.07.001.
5. Sleijpen GLG, van Gijzen MB. Exploiting BiCGstab(ℓ) strategies to induce dimension reduction. *SIAM Journal on Scientific Computing* 2010; **32**(5):2687–2709. DOI: 10.1137/090752341.
6. Onoue Y, Fujino S, Nakashima N. Improved IDR(s) method for gaining very accurate solutions. *World Academy of Science, Engineering and Technology* 2009; **55**:520–525.
7. Onoue Y, Fujino S, Nakashima N. An overview of a family of new iterative methods based on IDR theorem and its estimation. *Proceedings of the International MultiConference of Engineers and Computer Scientists, IMECS 2009*, 18–20 March 2009, Hong Kong, vol. II, 2009; 2129–2134.
8. Sonneveld P. On the convergence behaviour of IDR(s). *Technical Report*, Delft University of Technology, Delft, The Netherlands, 2010, DUT report 10-08.
9. Sonneveld P. On the statistical properties of solutions of completely random linear systems. *Technical Report*, Delft University of Technology, Delft, The Netherlands, 2010. DUT report 10-09.
10. Gutknecht MH, Zemke JPM. Eigenvalue computations based on IDR. *Technical Report*, Seminar für Angewandte Mathematik, ETH Zürich, SAM, ETH Zürich, Switzerland 2010. Research Report No. 2010–13.
11. van der Vorst HA. Bi–CGSTAB: a fast and smoothly converging variant of Bi–CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1992; **13**(2):631–644.
12. van Gijzen MB, Sonneveld P. An elegant IDR(s) variant that efficiently exploits bi-orthogonality properties. *Technical Report*, Delft University of Technology, Delft, The Netherlands 2010. DUT report 10–16 (revised version of DUT report 08–21).
13. Seinstra FJ, Verstoep K. DAS-3: the distributed ASCI supercomputer 3 2007. Available from: http://www.cs.vu.nl/das3/.
14. Chronopoulos AT, Gear CW. S-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics* 1989; **25**(2):153–168. DOI: http://dx.doi.org/10.1016/0377-0427(89)90045-9.
15. de Sturler E. A performance model for Krylov subspace methods on mesh-based parallel computers. *Parallel Computing* 1996; **22**(1):57–74. DOI: http://dx.doi.org/10.1016/0167-8191(95)00057-7.
16. Gu TX, Zuo XY, Liu XP, Li PL. An improved parallel hybrid bi–conjugate gradient method suitable for distributed parallel computing. *Journal of Computational and Applied Mathematics* 2009; **226**(1):55–65. DOI: http://dx.doi.org/10.1016/j.cam.2008.05.017.
17. Gu TX, Zuo XY, Zhang LT, Zhang WQ, Sheng Z. An improved bi-conjugate residual algorithm suitable for distributed parallel computing. *Applied Mathematics and Computation* 2007; **186**(2):1243–1253.
18. Yang TR. The improved CGS method for large and sparse linear systems on bulk synchronous parallel architecture. *ICA3PP '02*: *Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing*. IEEE Computer Society: Washington, DC, U.S.A., 2002; 232–237.
19. Yang L, Brent R. The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. *Fifth International Conference on Algorithms and Architectures for Parallel Processing*. IEEE Computer Society: Los Alamitos, CA, U.S.A., 2002; 324–328. DOI: http://doi.ieeecomputersociety.org/10.1109/ICAPP.2002.1173595.

20. Yang T, Lin HX. The improved quasi-minimal residual method on massively distributed memory computers. *HPCN Europe '97*: *Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*. Springer: London, U.K., 1997; 389–399.
21. Krasnopolsky B. The reordered BiCGStab method for distributed memory computer systems. *Procedia Computer Science* 2010; **1**(1):213–218. DOI: 10.1016/j.procs.2010.04.024.ICCS 2010.
22. Valiant LG. A bridging model for multi-core computing. *ESA '08*: *Proceedings of the 16th annual European Symposium on Algorithms*. Springer: Berlin, Heidelberg, 2008; 13–28. DOI: http://dx.doi.org/10.1007/978-3-540-87744-8_2.
23. Bal H. DAS-3 Opening Symposium 2007. Available from: http://www.cs.vu.nl/das3/symposium07/das3-bal.pdf. Retrieved 05/02/2009.
24. Petitet A, Whaley RC, Dongarra J, Cleary A. HPL—a portable implementation of the high–performance Linpack benchmark for distributed-memory computers 2008. Available at http://www.netlib.org/benchmark/hpl/.
25. Verstoep K, Maassen J, Bal HE, Romein JW. Experiences with fine-grained distributed supercomputing on a 10G testbed. *CCGRID '08*: *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*. IEEE Computer Society: Washington, DC, U.S.A., 2008; 376–383. DOI: http://dx.doi.org/10.1109/CCGRID.2008.71.
26. Sleijpen G, Fokkema D. BiCGstab(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions on Numerical Analysis* 1993; **1**:11–32.